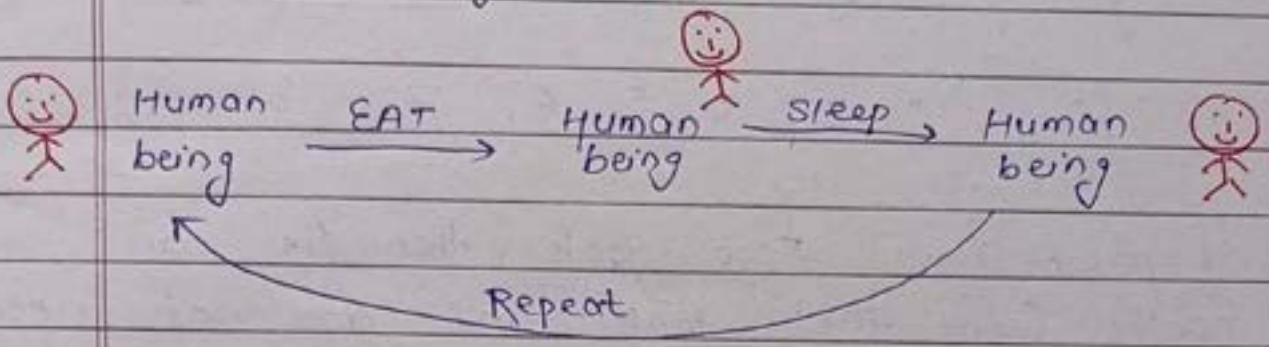
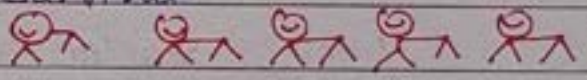
Lecture - 1

Introduction to programming

10,000 years ago,



→ These people does not know Counting, because at that time Couting is not discoverable.

→ But they also works for survival.
A man have 5 goats 
He Count the number of goats by picking a stone for one goat.

→ We he go to field with goats he filled his pocket with stones for one goat one stone.

When he Comes home he throw stone from pocket We one goat enter one stone throw.

At last if ~~0~~ ~~0~~ not any stone ~~is~~ in packet then all goats are safe.

If stone are kept in packet means Wild animal can eat the goat.

Teacher's Signature

After many years, Counting System Comes.

The first Counting system is decimal number.

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

If person have 5000 goats, then he can easily Count the goats and does not need to carry any Stones.

0 is discovered by Aryabhata.

At 1800, Some problem arises.
people started to doing bussiness and maintain a log book of sales or import-Export.

Let,
A bussinessman maintain 4 log book of his import - Export details.

Now, he want to calculate total profit / loss.

Bussiness man

x 4 = 4000 pages

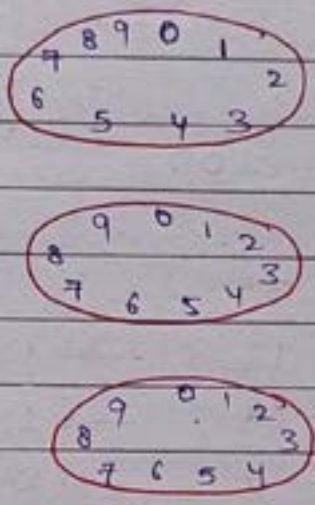
logBook
1000 pages

Now he started calculating.
There is a high probability to make mistakes while calculating.

Teacher's Signature.....

Then, Charles Babbage Makes a Computer called as Mechanical Computer which only calculate.

Looks like



Size is very big It is adjust in 1 room.

→ After some year, we found transistor.

→ At present computer can only understand binary language → 0 or 1.

0 → 0
1 → 1
2 → 10
3 → 11
4 → 100
5 → 101
6 → 110
7 → 111

→ [10110] → Convert into Decimal

$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$16 + 0 + 4 + 2 + 0$

22 (Decimal)

Teacher's Signature.....

→ Decimal to binary

4 →

		rem
2	4	
2	2	→ 0
2	1	→ 0
	0	→ 1

Ans : 100

35 →

2	35	
2	17	→ 1
2	8	→ 1
2	4	→ 0
2	2	→ 0
2	1	→ 0
When 0 comes ←	0	→ 1

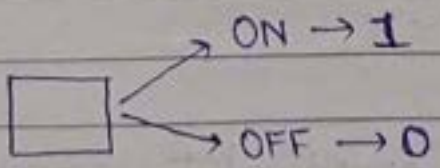
We stop.

Ans : 100011

* **Transistor :**

Before Transistor → Computer use decimal Number system

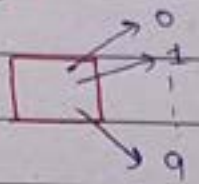
After Transistor → Computer use binary Number system.



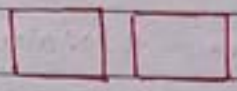
Transistor

Teacher's Signature.....

Decimal :

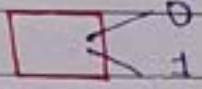


We can fill 10 Number in one place.

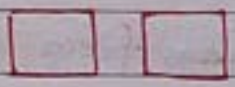


→ On this we can fill 10 × 10 = 100 Number.

Binary :

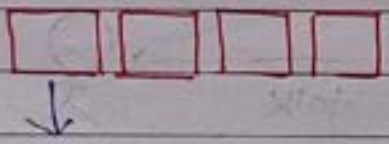


→ We can fill two types in one place



→ On this we can fill 2 × 2 = 4 types.

0 0
0 1
1 0
1 1



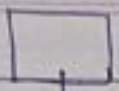
→ On this we can fill 16 types

This one box is one transistor.

Moore tells that in every 2 year the capacity of transistor get doubled.

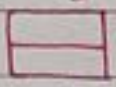
→ He predict this in 1956-57, and till now it is true.

Teacher's Signature.....

→  → Initially

↳ Stores 2 value.

After 2 year,



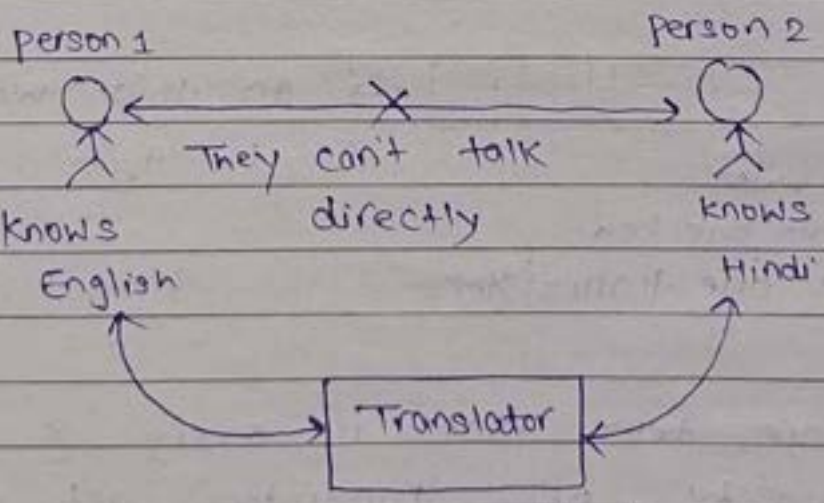
↳ Now, It stores 4 values.

capacity of transistor is increased very much.

Computer can only understand Binary language.
At 1960-70 programmer can write code only in Binary form.

Writing code in Binary form is very complex.
While writing code in Binary ~~from~~ form there is a high probability of mistake.

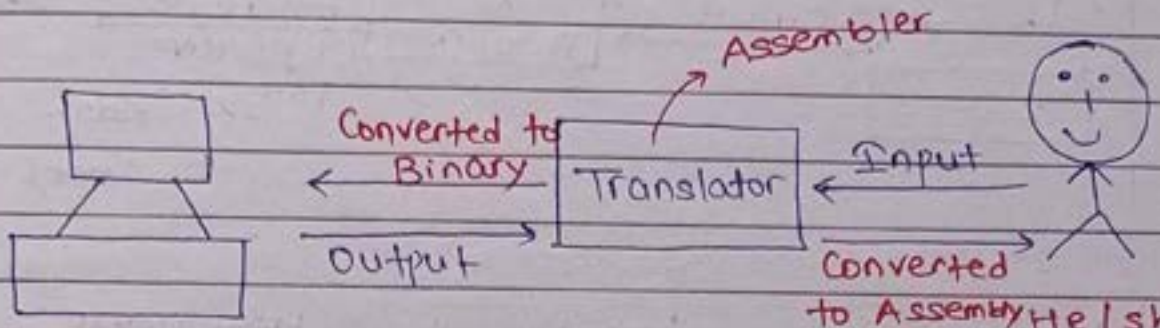
So, it does not give correct solution.



Translator can translate the language into known language of both person.

Teacher's Signature.....

Same in Computer World



Assembly lang: ADD 2 3

Computer does not understand Assembly language. So, a translator translate the Assembly Code into Binary Code.

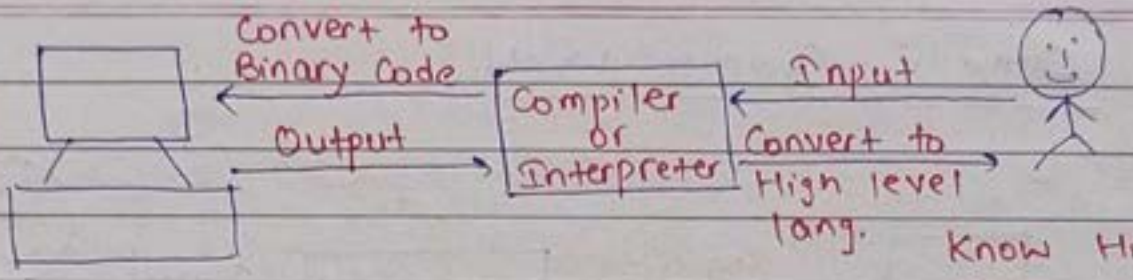
Assembler can convert Assembly Code to Binary Code and vice versa.

In Assembly language the length of Code is very high.
To reduce the length of code we interoduce high level language.

↳ Java, C++.
↳ Similar to english.

High level language is programmer friendly.

Teacher's Signature



Machine language > Assembly language > High level language

Execution time

* find sum of

$1 + 2 + 3 + 4 + \cdots + 100$

1st Approach:

$1 + 2 + 3 + 4 + \cdots + 100$

3 6 10

Ans = 5050

It takes more time to solve and take more space.

2nd Approach:

formula for first n natural number:

$$\frac{n(n+1)}{2} = \frac{\overset{50}{\cancel{100}} \times 101}{2}$$

$$= 5050$$

It take very less time and very less pace.

Teacher's Signature

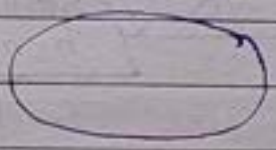
Time Complexity :

Total time taken by the algorithm with respect to input size.

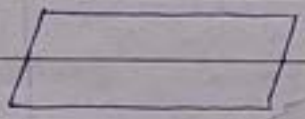
* Flowchart :-

A diagram that shows the connection b/w different stage of a process or part of system.

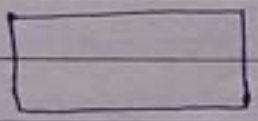
Components of flowchart



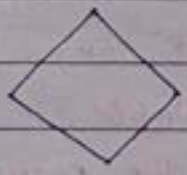
Start / End



Input / output



process



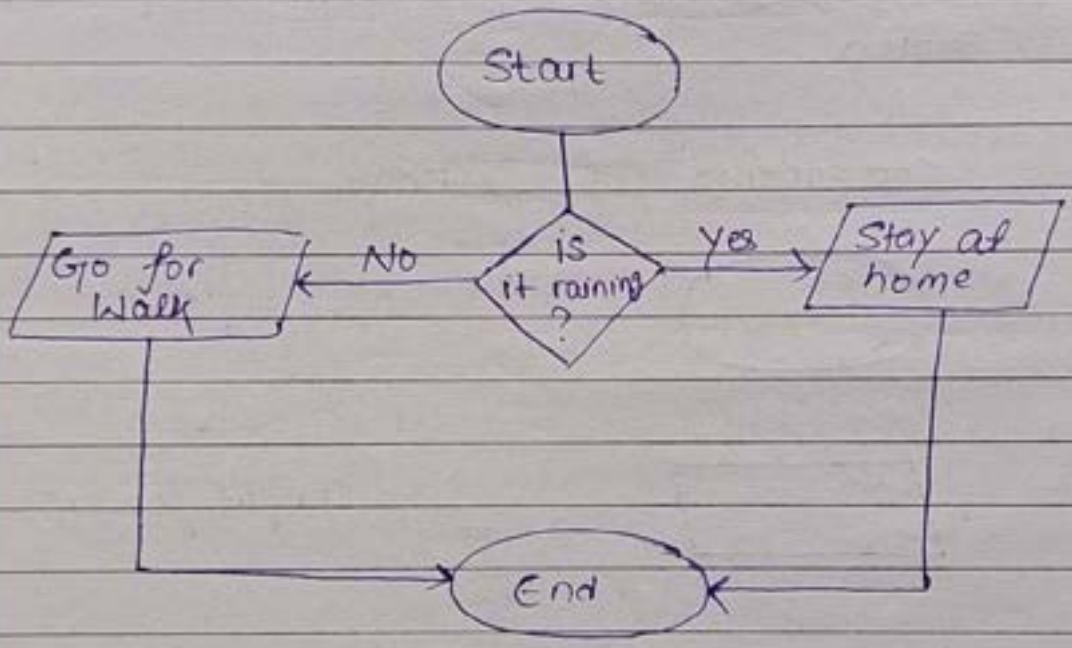
Descision Making

flow / direction

Example - 1

* Decide you have to go out or not, if it is a Rainy Season.

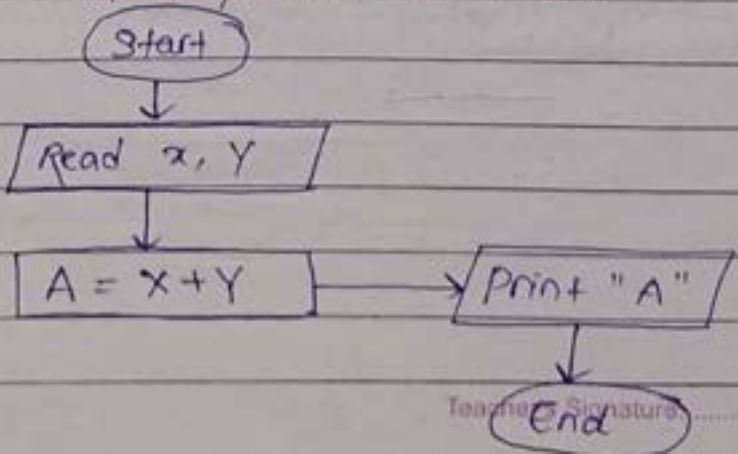
→ If raining, then stay at home
→ If not raining, then go for walk



Example - 2

Add 2 number

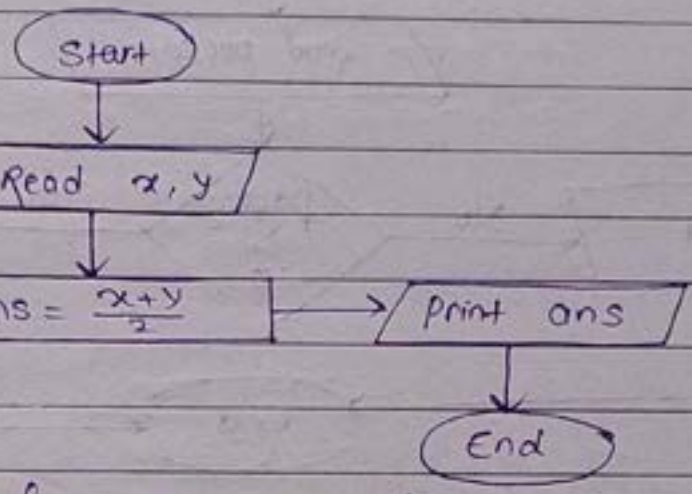
→ Take input of two number
→ Sum both number
→ Return / show the result.



Teacher's Signature .....

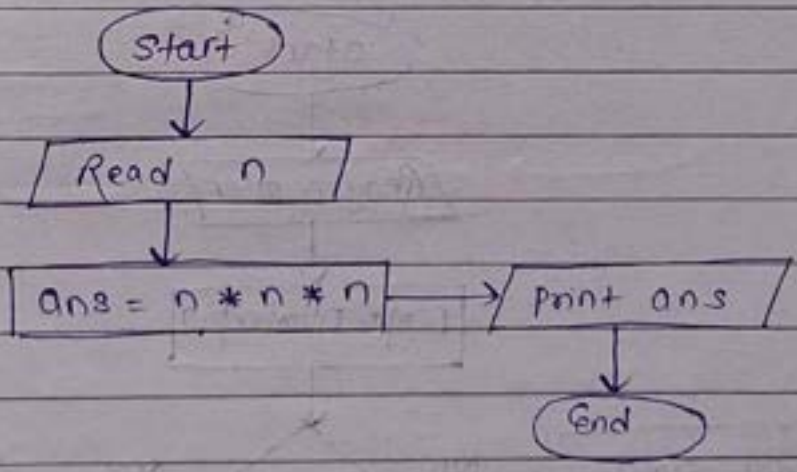
Example : 3 Average of 2 Number

pseudoCode
- → Take input of two number
- → sum both number and divide by 2
- → show the result.



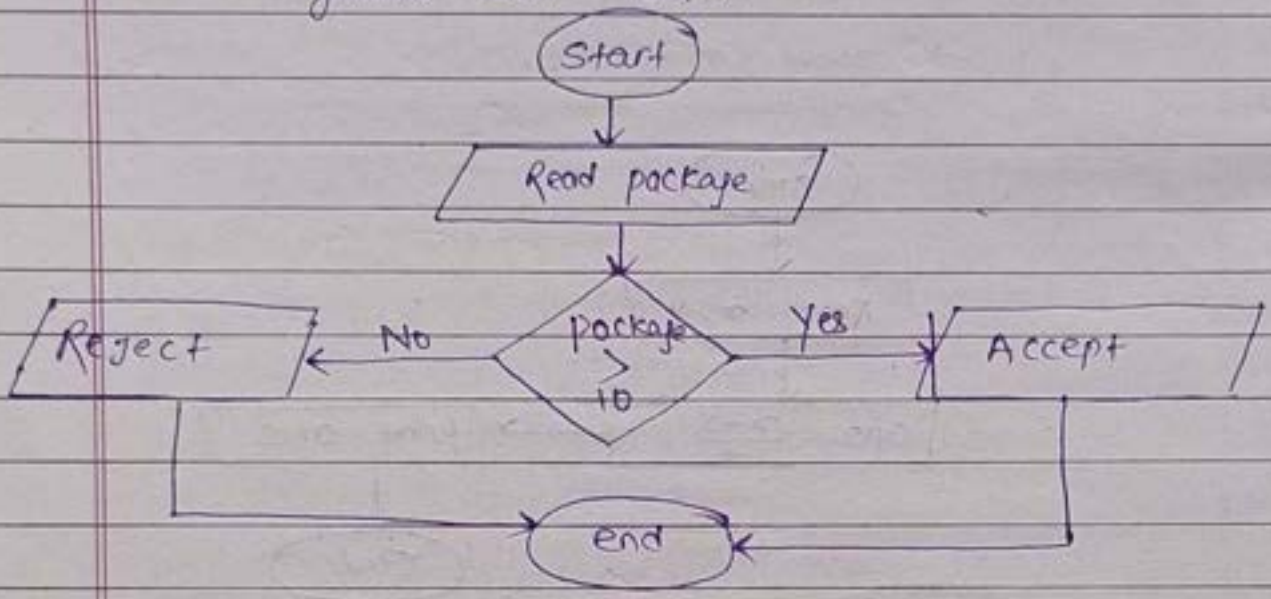
Example : 4 find the cube of number

pseudoCode
- → Take input x
- → find cube by num * num * num
- → print the result.



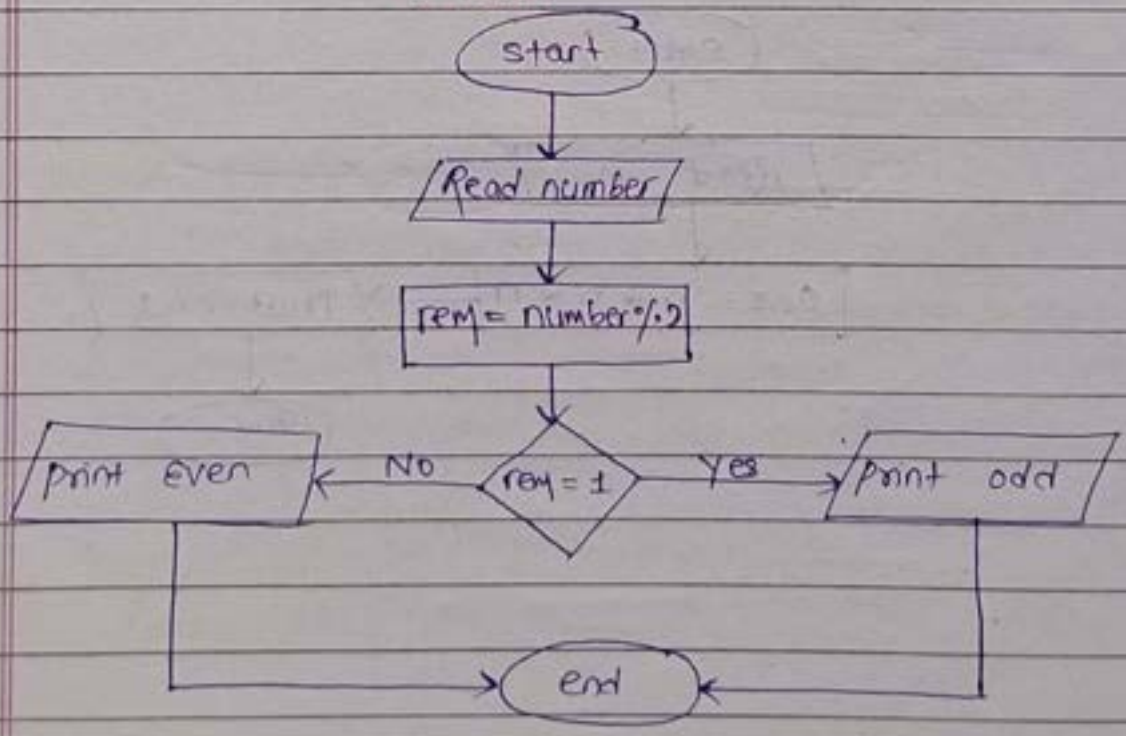
Teacher's Signature.....

Example - 5 : You got an offerletter from X Company. You will accept offer only when package is greater than 10LPA.



Example - 6 Check number is Even or odd

→ Input number
→ If number %. 2 is 0, then even
→ If number %.2 is 1, then Odd
→ end.



Teacher's Signature.....

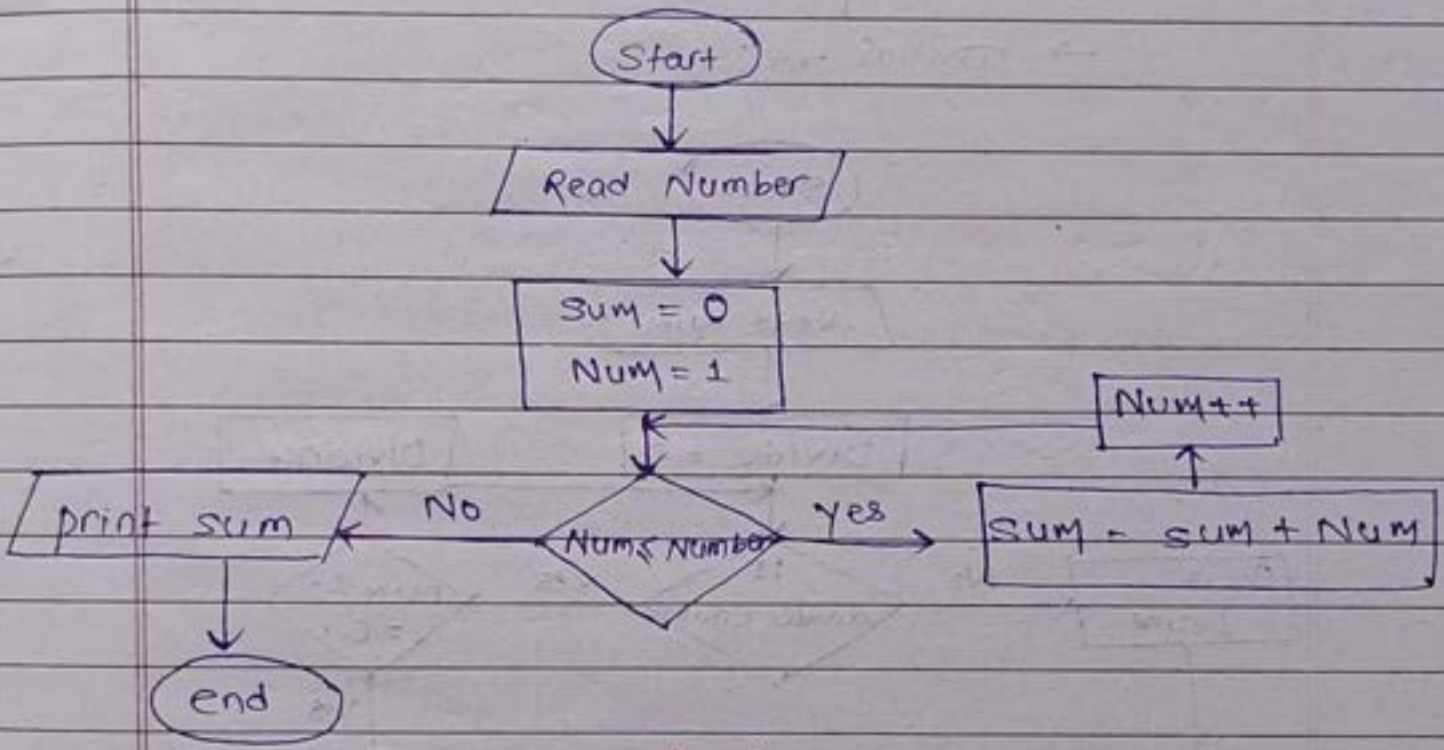
Example - 7 Sum of n natural number

→ Take input

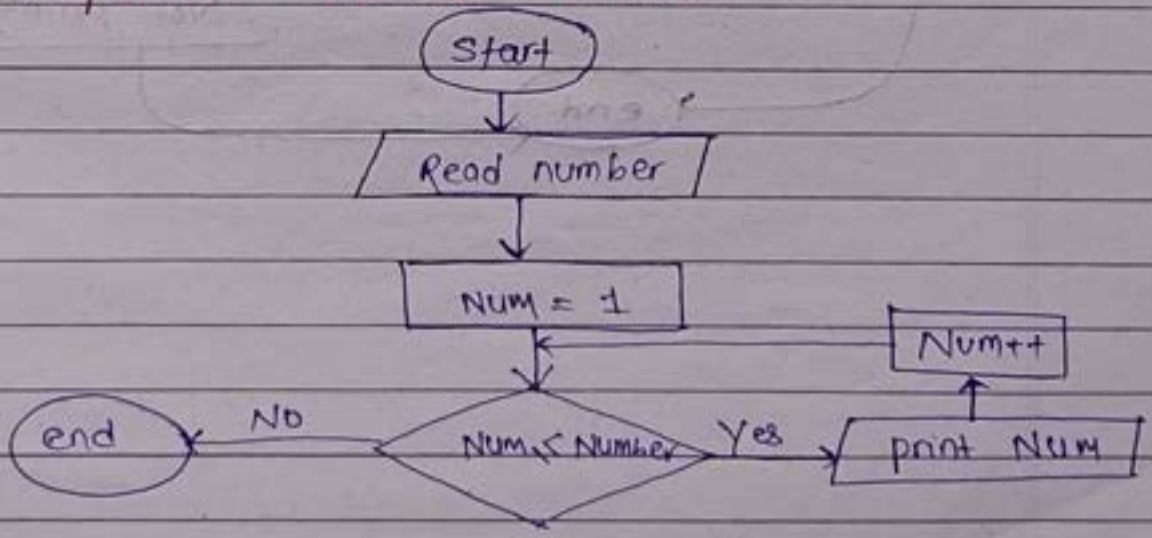
→ Take 1 by 1 number from 1.

→ Add the number in sum.

→ Repeat this process until n reached.



Example : print n natural number



Teacher's Signature.....

→ Divided by two number and both are different.

* Check for prime number:

→ Take input n.

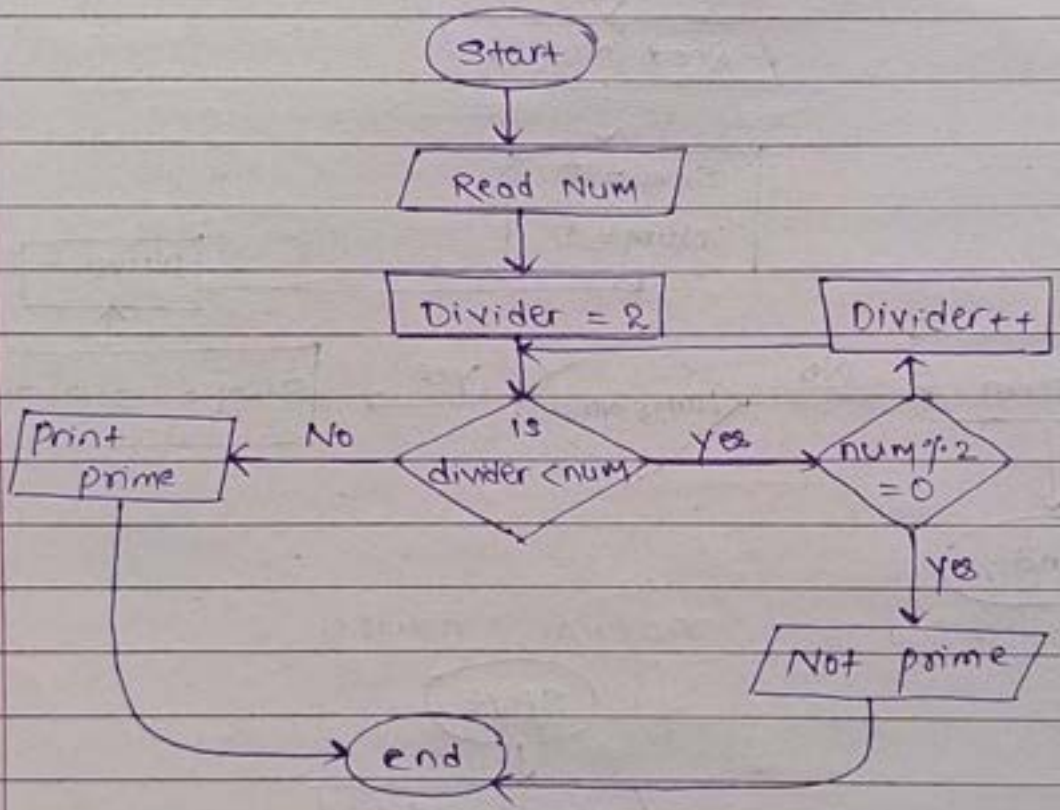
→ Take Initial = 2

→ check n % Initial == 0

→ if 0, then print Not prime, return.

→ else prime

→ Initial ++;



Teacher's Signature.....

Lecture - 2

Introduction to C++ and Write first code

Computer Memory units :

Smallest unit of Memory : 1 bit
(1 transistor can represent 1 bit)

	1 Byte	=	8 bit
(Kilo Byte)	1 KB	=	2^{10} byte
(Mege Byte)	1 MB	=	2^{20} byte = 2^{10} KB
(Giga Byte)	1 GB	=	2^{30} byte = 2^{10} MB
(Tera Byte)	1 TB	=	2^{40} byte = 2^{10} GB

Computer only understand Binary number. 0 & 1.

ASCII TABLE :

Every Symbol, Sign present in keyboard have unique ASCII Value.

A → 65 → 1000001

B → 66 → 1000010

0 → 48 → 00110000

a → 97 → 01100001

z → 122 → 01111010

Teacher's Signature.....

Page No.
Date : / /

Introduction to c++ :

C++ is extension of c language.

* Simple code to print a message:

```
int main() {  → Tells from where it start reading
    Cout << "Hello CoderArmy";
    return 0;  → tell end of program.

}  → tells where it can stop reading
```

Cout → print output
↓
character output / Console output

<< → insertion operator

→ `#include <iostream>` → Header file, Library where meaning is defined.
`using namespace std;`

```
int main() {
    Cout << "Hello CoderArmy";
    return 0;
}
```

→ If we don't put "using namespace std;" at starting then we have to write at every statement when we use Cout.

```
std::Cout << "Hello CoderArmy";
```

Teacher's Signature.....

\n : nextline or endl

#include <iostream>

↳ It is a library where meaning of every term which is used in program is defined.

Cout << a << b << c;

↳ It print value of a, b, c.

* Variables and datatypes

A human can understant Number System, Alphabet, Word, Boolean.

Just like Computer also understand these with different names.

① int → Example : 4, 69, 0, -9, etc.

Size : 4 byte

for storing a integer value it takes 4 bytes.

Ex : 10 → 1010

In int

00000000 00000000 00000000 00001010

↳ 10 in Binary number.

int a = 10;

datatype variable value

Page No.:
Date: / /

Rules for Writing a variable name:-

→ Name your variable is related to your code
→ Variable name always start with alphabet or underscore.
→ Variable name consist of number, alphabet or underscore.
→ Keyword are not put as variable name.

KeyWords:

The keywords are reserved and the meaning are fixed.

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

② char

Ex: char c = 'b';

↳ represent within single quote
↳ single value stored inside the quote.

Size : 1 byte.

Teacher's Signature.....

→
```
int main() {
    int a = 10;
    cout << a;        → It prints 10.
}
```

→
```
int main() {
    int a = 10;
    char c = 'z';
    cout << "a";      → prints a
    cout << a;        → prints 10
    cout << c;        → prints z
    return 0;
}
```

(3) float → decimal value

Ex: float a = 9.62;
float height = 5.9;

Size: 4 byte.

(4) Double → big decimal value

Size: 8 byte

Ex: 634.98239864832 6
↳ This can't be stored in float.

Size: 8 byte.

Teacher's Signature.....

⑤ long int :

Ex: long int a = 56789234;

for storing large value

which can't be stored in int.

Size = 8 byte.

→ int main(){

float num = 23.45;

Cout << num << endl;

double num1= 29.685674;

Cout << num1 << endl;

return 0;

}

* Comment :

Single line

// This is single line Comment

Multi line

/* This is

Multi line

Comment */

→ How to Store Negative Number :

Num = 5

↳ 101

for -5

find 2's Complement of 5

Teacher's Signature.....

5 → 101

1's Complement (010)

2's Complement (011)

2's Complement : 011

~~Negative~~ ← 0 (_ _ _ _ _ _) positive

~~positive~~ ← 1 (_ _ _ _ _ _) Negative

for 3 space (bit)

_ _ _

Total possible → 8

positive = 3

Negative = 4

zero = 1

for 32 bit

Total possible → 2^{32}

positive = $2^{31} - 1$

Negative = 2^{31}

zero = 1

⑥ bool

ex : bool isprime = true;

bool isOdd = 0;

size = 1 byte

0 or 1

true or false.

Teacher's Signature

→ int main() {

bool num = true;

Cout << num << endl;

return 0;

}

→ To find the size of any datatype:

Code:

int main() {

int number = 10;

Cout << sizeof(number) << endl; return → 4

float num = 10.99;

Cout << sizeof(num) << endl; return → 8

* Typecasting:

int main() {

char c = '8';

Cout << c << endl; → output = 8

char a = 65;

Cout << a << endl; → output = A

return 0;

}

↳ Typecasting

Teacher's Signature.....

Page No.
Date / /

Char a = 65;

↳ We take char datatype and store integer

So Computer Convert integer value to Binary number

65 = 1000001

then it is Converted and checked in the ASCII table and print the character associate With it.

1000001 → A

→ int main() {
int c = 'A'; → Converted into Binary and then check in Ascii And return the integer Value associated With it.
Cout << c << endl;
return 0;
}

Output = 65

→ int main() {
char c = 590;
Cout << c << endl;
return 0;
}

Output = N.
↓
Gives a Warning Statement on character We can store only 2^8 value.

Maximum We can store $2^8 = 256$ value.
0 to 255.

Teacher's Signature.....

But We give as input
590
so, it take Modulo With Maximum storage capacity

590 % 256
↳ (78)
↳ It give output which character is associated With Binary representation of 78.

64 32 16 8 4 2 1
78 → 0 1 0 0 1 1 1 0
↳ It give 'N'
↓
output of previous code.

* Conditional Statement : (If - else condition)

Find greatest among two number.

a = 10;
b = 15

```
int main() {
    int a = 10, b = 15;
    if (a > b)                    // Output : 15
        Cout << a;
    else
        Cout << b;
    return 0;
}
```

Teacher's Signature

Page No.
Date : / /

Operator :

Multiplication : 2 * 3 = 6
Division : 8 / 2 = 4
Modulo : 9 % 2 = 1

Comparison Operator :

< → less than
> → greater than
<= → less than or equal to
>= → greater than or equal to
!= → Not equal to

→ check given number is even or odd.

→ even : 0, 2, 4, 6, ...
odd : 1, 3, 5, 7, ...

Number % 2 = 0 (Even)
Number % 2 = 1 (Odd)

Code :

```
int main() {
    int num = 25;
    if (num % 2 == 1)          // output
        cout << "ODD";         : ODD
    else
        cout << "EVEN";
    return 0;
}
```

Teacher's Signature.....

* For Multiple Condition :

```
if ( ) {

}
else if ( ) {

}
else if ( ) {

}
:
:
else {

}
```

→ find given number is +ve, -ve or zero.

```
int main () {
    int number = 16;
    if (number > 0)
        cout << "positive";
    else if (number < 0)
        cout << " negative";
    else {
        cout << " zero";
    }
return 0;
}
```

output
positive.

Teacher's Signature.....

→ && → AND operator

|| → OR operator

Given 3 number, print Maximum

```
int main() {
int a = 10, b = 15, c = 20;
if (a>=b && a>=c)
    Cout << a;
else if (b>=a && b>=c)
    Cout << b;
else
    Cout << c;
return 0;
}
```

Output
20

0	AND	0	→	0
0	AND	1	→	0
1	AND	0	→	0
1	AND	1	→	1

0	OR	0	→	0
0	OR	1	→	1
1	OR	0	→	1
1	OR	1	→	1

Teacher's Signature

Page No.
Date : / /

Lecture - 3

If-Else Condition, Operators and Basic of loops

* Why We Write return 0 ?
  return 0 tells that the program ends now. and after return 0 program is returned and terminated.

* Why "using namespace std" ?
  We can define the properties of keywords using "using namespace std".

→
```
int main() {
    int Cout = 10;
    Cout << Cout;
}
```
→ It gives error because it can't understand.

→
```
int main() {
    int Cout = 10;
    std::Cout << Cout;
}
```
→ It Work properly.

Teacher's Signature.....

* Write a program to check given character is Vowel or Consonant.

Vowe : a e i o u

if ~~vowel~~ character is Vowel
then print Vowel

Otherwise
Print Consonant

In this we use OR operator.
a OR e OR i OR o OR u

→

```
#include <stdio.h>
#include <iostream>
using namespace std;
int main() {
    char c = 'u';

if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
        cout << "Vowel";
else
        cout << "Consonant";
return 0;

}
```

Output = Vowel

Teacher's Signature

* Write a program to check whether given year is leap or not.

For leap:

year % 400 = 0
↳ then leap year

year % 4 = 0 & year % 100 != 0
↳ then leap year

Otherwise
↳ Not Leap year.

→
```
#include <iostream>
using namespace std;

int main() {
    int year;
    cin >> year;          // Take input by user.

    if (year % 400 == 0)
        cout << "leap year";
    else if (year % 4 == 0 && year % 100 != 0)
        cout << " leap year";
    else
        cout << " Not a leap year";

    return 0;
}
```

Output:
1996
Leap year

Teacher's Signature.....

Page No.
Date: / /

* Given a number, check if it is divisible by 3 & 5?

Ex: 10

10 % 3 = 1 ✗

Not perfect

15 15 % 3 = 0 ✓

15 % 5 = 0 ✓

prefect

9

9 % 3 = 0 ✓

9 % 5 = 4 ✗

Not perfect.

Pseudcode:

```
if (num % 3 == 0) {
    if (num % 5 == 0) {
        Cout << " Perfect";
    } else {
        Cout << " Not perfect";
    }
}
else {
    Cout << " Not perfect";
```

Code:

```
#include <iostream>
using namespace std;

int main() {
    cin >> num;
```

Teacher's Signature.....

```
if (num %.3 == 0  && num %.5 == 0)
        Cout << " perfect";
else
        Cout << " ~~Perfect~~ Not perfect";

return 0;
}
```

→ Output:

1) 15
Perfect

2) 20
Not perfect

* For taking input From user:

cin >> Variable_name;

cin : character input or Console input

>> → Extraction

Teacher's Signature

* Add two number by taking input from user:

```
#include <iostream>
using namespace std;

int main() {
  int num1, num2;
  cin>> num1;
  cin>> num2;
  int sum;
  sum = num1 + num2;
  Cout << "sum is " <<sum;
  return 0;
}
```

Output

→ 24

→ 26

50

* Loops:

Repeat one task a finite number of time.

```
for (initialize ; condition ; decrement/operation/Increment) {
        || lines of code
    }
```

→ We want to print whole number from 1 to 100.

for this we have to write 100 line of code.

```
Cout << 1;
Cout << 2;
:
```

Teacher's Signature.....

From For loop We can Write easily.

```
for (int i = 1;  i<=100; i++){
        Cout<<i;
    }
```

Code for printing 1 to n.

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;

  for(int i= 1; i<= n; i++) {
        Cout << i<<" ";
      }

  return 0;
}
```

Output

20

1 2 3 4 5 6 7 8 9

10 11 12 13 14 15 16 17 18

19 20

Teacher's Signature.....

* Pre increment & post increment :-

↓ ++num

Sum = 0

num = 0;

Sum = num++

Sum = 0 + 1

Sum = 1 + 1

= 2

↓ num++

first print then add.

num = num + 1

first add then print.

num = 0

Cout << ++num;

↓

output = 0

Cout << num++;

↓

output = 1.

* print all even number 1 to 20

2, 4, 6, 8, 10, 12, 14, 16, 18, 20

2%2 4%2 6%2

= 0 = 0 = 0

num%2 = 0

↳ then it is odd.

We can solve this by for loop to check all number present inside limit.

Teacher's Signature

Code :

```
#include <stdio.h>
using namespace std;

int main() {
    for (int i = 0; i <= 20; i++) {

        if (i % 2 == 0)
            Cout << i << " ";

    }
    return 0;
}
```

Output :-

2 4 6 8 10 12 14 16 18 20

* code for sum of first n natural number.

let n = 10;

start from 1; i = 1;

Initially, Sum = 0

Sum = 0 + 1 = 1

i = i + 1

Sum = 1 + 1 = 2 i = 3

Sum = 5

:

Sum = 55

Teacher's Signature

Code :

```
#include <stdio.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    sum = 0;
    for (int i = 1; i <= n; i++) {
        sum = sum + i;
    }
    cout << sum;
    return 0;
}
```

Teacher's Signature.....

Lecture - 4

For loop + While loop + pattern printing

A pandit say "OM" 108 times till locket does not end.

For repeating one work we use loops:

If Computer is a pandit then he write:

```
for (int i=0; i<108; i++)
    cout << "OM";
```

* Write a table using program:

```
3 * 1  = 3
3 * 2  = 6
 :
3 * 10 = 30
```

Constant ↙ Increment by 1. ↘ 3xi (always)

i have some limit till where it go.

```
for (int i=1; i<=10; i++) {
    cout <<"3 * " << i << " = " << 3*i <<endl;
}
```

* Write a program to check prime Number:

Prime number? divided by 1 & itself.

Let check for 15

We can check 2 to 14

if any number from 2 to 14 can divide 15 then it is not prime.

15 % 2 == 1 ✗

15 % 3 == 0 ✓ (Not prime)

Break the loop.

* Code:

```
#include <iostream>
using namespace std;
int main() {
    int num;
    cin >> num;
    for (int n = 2; n <= num - 1; n++) {
        if (num % n == 0) {
            cout << " Not prime";
            break;
        }
        cout << " prime Number";
    }
    return 0;
}
```

Teacher's Signature.....

break : If we put break, then it exit from loop permanently.

put some edge case :

1 is Not prime.

Code :

```
int main () {
    int n;
    cin >> n;
    if (n < 2) {
        Cout << " Not prime";
        return 0;
    }
    for (int num = 2; num <= n-1; n++) {
        if ( n % num == 0) {
            Cout << " Not a prime";
            return 0;
        }
    }
    cout << " prime";
    retun 0;
}
```

Teacher's Signature

* Fibonacci series :-

0 1 1 2 3 5 8 13 21 34

→ 1st element → 0 (fixed)
→ 2nd element → 1 (second)

Third → 1st + 2nd = 0 + 1 = 1
4th → 2nd + 3rd = 1 + 1 = 2
5th → 3rd + 4th = 1 + 2 = 3

$n^{th} \rightarrow (n-1)^{th} + (n-2)^{th}$.

First = 0
second = 1
current = 0 + 1 = 1 (Third)

Now,

First ← second
First = 1
Second ← current
second = 1
current = 1 + 1 = 2 (Fourth)

→ Code :

```
int n;
cin >> n;
int first = 0;
int Second = 1;
int Current;
for (int i = 3; i <= n; i++) {
    Current = first + second;
```

Teacher's Signature.....

```
        first = second;
        second = current;
    }

    Cout << current;
    return 0;
}
```

→ The upper code work on n > 2.

→ Edge case. (put before loop)

```
if(n < 3)
    return (n-1);
```

or

We write

```
if(n==1 || n==2)
    return (n-1);
```

* n = 3>2; // True
Cout << n; // output = 1

n = 3 > 2 > 1;
Cout << n;

n = (3 > 2) > 1
     1
= 1 > 1 // false
output = 0.

Teacher's Signature

* Operator & (precedancy)

	Operator	Associativity
	(), [], →, .	Left to right
unary	!, ~, ++, --, +, -, *, & (type), sizeof	Right to left
Arithmetic	*, /, %.	Left to Right
	+, -	
shift	<< >>	,,
Comparison	<, <=, >=, >	,,
	== , =	,,
BitWise	&	,,
logical	&&, ||, ?:	,,
Assignment	=, +=, -=, *=, /=, != , <<= , >>=	Right to left.

```
n = (3 > 2) > 1
      1
1 > 1 → false
return 0;
```

Teacher's Signature.....

* Pattern :

1)

1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

	Col 1	2	3	4
Row 1	00	01	02	03
Row 2	10	11	12	13
Row 3	20	21	22	23
Row 4	30	31	32	33

First Approach :

Make 4 for loop and in every for loop print 1 four times.

```
for (int i = 1 ; i <= 4; i++)
    Cout << "1";
for (int i = 1; i <= 4; i++)
    Cout << "1";
```

What if pattern is of 100 lines
In this we can repeat same loop.
We have to use loop for this. We use nested loop here.
(loop ke andar loop) :-

```
int main() {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            Cout << "1";
        }
        Cout << endl;
    }
return 0;
}
```

Teacher's Signature

2)

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

00 1	01 2	02 3	03 4
10 1	11 2	12 3	13 4
20 1	21 2	22 3	23 4
30 1	31 2	32 3	32 4

↓

It print value of (col + 1) in each box.

We use nested loop.

```
int main() {
    for (int row = 0; row < 4; row++) {
        for (int col = 0; col < 4; col++) {
            cout << col+1;
        }
        Cout << endl;
    }
    return 0;
}
```

3)

4	3	2	1
4	3	2	1
4	3	2	1
4	3	2	1

00 4	01 3	02 2	03 1
10 4	11 3	12 2	13 1
20 4	21 3	22 2	23 1
30 4	31 3	32 2	33 1

At every box (4 - col):

```
for (int row = 0; row < 4; row++) {
    for (int col = 0; col < 4; col++) {
        Cout << 4 - col;
    }
    Cout << endl;
}
```

Teacher's Signature

4)

1	2	3	4	00(1) 01(2) 02(3) 03(4)
5	6	7	8	10(5) 11(6) 12(7) 13(8)
9	10	11	12	20(9) 21(10) 22(11) 23(12)
13	14	15	16	30(13) 31(14) 32(15) 33(16)

At Every box it in incremented by 1

Start With

Cout = 1

When it prints:

then cout++;

Code:

```
int cOUnt = 1;
for (int row = 0; row < 4; row++) {
    for(int col = 0; col < 4; col++){
        Cout << cOuNt << " ";
        Count ++;
    }
    Cout << endl;
}
```

5)

```
*
* *
* * *
* * * *
```

```
00
10 11
20 21 22
30 31 32 33
```

Every row कहाँ तक जा रहा है।

row के value तक।

1 row = 1

2nd row = 2

3rd row = 3

Teacher's Signature

Code :

```
int n = 4;
for(int row = 0; row < n; row++) {
    for(int col = 0; col <= row; col++) {
        cout << "* ";
    }
    cout << endl;
}
```

5)

```
1
1 2
1 2 3
1 2 3 4
```

```
00 1
10 1  11 2
20 1  21 2  22 3
30 1  31 3  32 3  33 4
```

It stops when

col <= row

and

prints col + 1.

```
for (int row = 0; row < 4; row++) {
    for (int col = 0; col <= row; col++) {
        cout << col + 1;
    }
    cout << endl;
}
```

Teacher's Signature.....

6)

a	a	a	a	00a	01a	02a	03a
b	b	b	b	10b	11b	12b	13b
c	c	c	c	20c	21c	22c	23c
d	d	d	d	30d	31d	32d	33d

1st Approach:

Make 4 loop one by one and print 'a' in frost loop and 'b' in second loop and increase in Next loop.

```
for (int i=0; i<4; i++){
        cout <<" a";
    }
for (int i=0; i<4; i++){
        cout <<" b";
    }
```

2nd Approach (By Nested loop):-

In this it uses typecasing.

In 1st loop it print a

ASCII value of a = 98

Code:

```
for (int row = 0; row < 4; row++){
        char c = 'a' + row;
    for (int col=0; col < 4; col++){
        cout << c <<" ";
    }
    cout <<endl;
}
```

Teacher's Signature

7)

a	00 a
a b	10 a 11 b
a b c	20 a 21 b 22 c
a b c d	30 a 31 b 32 c 33 d

→ on every row print till row number.

→ what to print

→ at every column change the value changes.

```
for (int row = 0; row < 4; row++) {
        char c = 'a';
    for (int col = 0; col < 4; col++) {
            c = 'a' + 0;
            cout << c << " ";
    }
    cout << endl;
}
```

Lec - 5

pattern printing + While loop + Basic Maths

1)

```
        *          1
      * * *        3
    * * * * *      5
  * * * * * * *    7
```

Row	print
1	1
2	3
3	5
4	7

	01	02	03	04	05	06
10	11	12	13	14	15	16
20	21	22	23	24	25	26
30	31	32	33	34	35	36

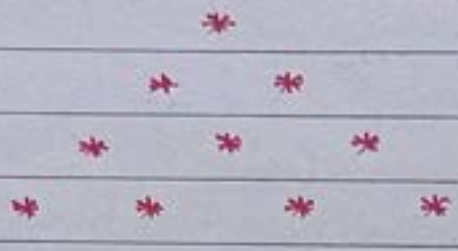
* { print : (2 × ROW - 1); }

Space : (n - ROW) ← Total Rows

```
int row, col;
for(row = 1; row <= n; row++) {
    for(col = 1; col <= (n - row); col++) {
        Cout << " ";
    }
    for(col = 1; col <= (2 * row - 1); col++) {
        Cout << "*";
    }
    Cout << endl;
}
```

Teacher's Signature

2)



Row	Print
1	1
2	2
3	3
4	4

00	01	02	03 *	04	05	06
10	11	12 *	13	14 *	15	16
20	21 *	22	23 *	24	25 *	26
30 *	31	32 *	33	34 *	35	36 *

Row	space
1	3
2	2
3	1
4	0

What to print : " * ";

one start and one space.

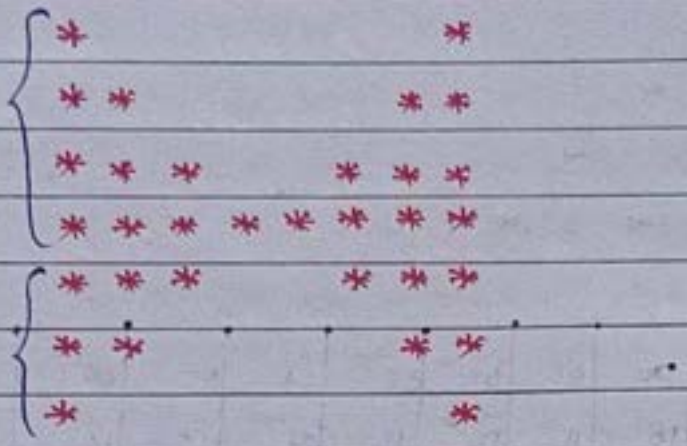
front space : (Total row - row);

Code :

```
int totalRow;
Cin >> totalRow;

for(int row = 1; row <= totalRow; row++) {
    for (int col = 1; col <= totalRow - row; col++)
            Cout << " ";
    for (int col = 1; col <= row; col++) {
            Cout << " * ";
    Cout << endl;
}
return 0;
}
```

Teacher's Signature.....

3)



This question is very difficult to print this pattern in one go.

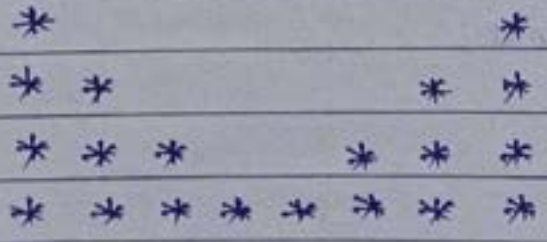
This question is converted in 2 phase.

For n = 4

Star	Row	
2	1	* *
4	2	* * * *
6	3	* * * * * *
8	4	* * * * * * * *
6	5	* * * * * *
4	6	* * * *
2	7	* *

Total Row
= n x 2 - 1
= 7

For 1st part :



Teacher's Signature

Star	ROW
2	1
4	2
6	3
8	4

Star = 2 x ROW

Total col = Input x 2 = 4 x 2 = 8

Space in between

= Total col - No. of start

For 2nd part:

r

1 * * * * * *

2 * * * *

3 *

n = 4

Total ROWS = 3 (n-1)

Star = 2 * (ROW - r)

Code:

```
// for first part:
int main() {
    int n;
    cin >> n;
for(int row = 1; row <= n; row++) {
    for(int col = 1; col <= row; col++) {
            cout << "*";
        }
    for(int col = 1; col <= n*2 - 2*row; col++) {
            cout << " ";
        }
    for(int col = 1; col <= row; col++){
            cout << "*";
        }
    cout << endl;
}
```

// for second part

```
for (int row = 1; row <= n-1; row++) {
    for (int col = 1; col <= n-row; col++)
        cout << "*";
    for (int col = 1; col <= 2*row; col++)
        cout << " ";
    for (int col = 1; col <= n-row; col++)
        cout << "*";
    cout << endl;
}
return 0;
}
```

Teacher's Signature

Basic Math :

Ex - OR

A	B	output
0	0	0
0	1	1
1	0	1
1	1	0

Symbol : $\wedge$

$A \wedge B$

* Num = 2 3 6 8 9

Find ~~total~~ No. of digit.

Approach 1 :

Divide No. by 10 and if Queint is 0. then return 2.

If Quotient is not 0 then divide by 100 check Quotient.

and increase the divisor 10, 100, 1000

Ex $10\overline{)278}\,(27$ — 70 — 8

$100\overline{)278}\,(2$ — 200 — X78

$1000\overline{)278}\,(0$

digit = 3

Teacher's Signature

Eg 276

Digit = 1 10

276 % 10 = 0 ✗

Digit++ 10×10

276 % 100 = 0 ✗

Digit++ 100×10

276 % 1000 = 0 ✓

Digit = 3 return 3

```
int num = 10
int digit = 1
```

this is may be wrong for very big number

```
for (int i = 1; i <= 1000; i++) {

    if (n / num == 0) {
        cout << digit;
        return 0;
    }
}
```

This code is not good because it take more time.

Teacher's Signature

Code:

```
int num = 10, digit = 1;
for ( int i=0; i<1000; i++){
    if ( n / num == 0){
        Cout << digit
        return 0;
    }
    num = num * 10;
    digit ++;
}
```

→ Code from While loop

```
int n;
cin >> n;
int digit = 0;

While ( n > 0) {
    ~~int digit = n %~~ 10;
    ~~Sum = Sum~~
    digit ++;
    n = n / 10;
}
    Cout << digit ;
    return 0;
}
```

Teacher's Signature

* print the reverse of a number

No : 3 2 6 5

Reverse : 5 6 2 3 ← Reverse.

3 2 6 5

$3 \times 1000 + 2 \times 100 + 6 \times 10 + 5$

Sum = 0

digit = 5

Sum = 5

digit = 6

Sum = $5 \times 10 + 6 = 56$

digit = 2

Sum = $56 \times 10 + 2 = 562$

digit = 3

Sum = $562 \times 10 + 3$

$= 5623$ → Answer

```
sum = 0;
while ( num) {
    int digit = num %10;
    Sum = sum *10 + digit;
    num = num /10;
}
Cout << sum;
return 0;
}.
```

Teacher's Signature

* Decimal to Binary (Conversion)

Decimal = 11

Binary = 1011

2	11	rem
2	5	1
2	2	1
2	1	0
	0	1

1011

Kab tak

while (num > 0)

Binary ko kaise store karenge.

binary = binary + (digit * product)

↓

digit ⇒ num % 2;

At every step

num = num / 2;

product = Product * 10;

* Code:

```
int product = 1;
int num;
cin >> num;
int sum = 0;
While ( num > 0) {
    int digit = num % 2;
Sum = Sum + (digit * product);
```

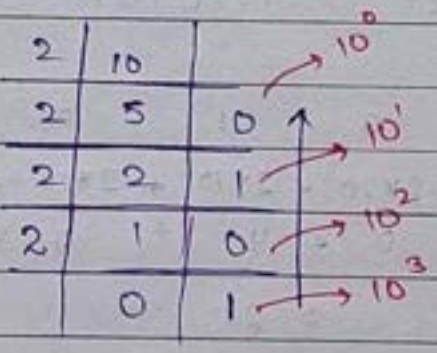
Teacher's Signature

```
        num = num / 2;
        product = product * 10;
    }
    Cout << sum;
}
```

Lecture - 6

Decimal to Binary + Functions

Decimal to Binary



1010

0101

⇓

reverse While

When we solve.

Code : (using While loop)

```
int sum = 0;
cin >> num;
int mul = 1
While ( num > 0 ) {
    int rem = num % 2;
    Sum = sum + ~~Digits~~ (rem * mul);
    num = num / 2;
    mul = mul * 10;
}
```

Teacher's Signature.....

Page No.
Date

Note 8

→ If we want to find the Binary of big number like 53289. it gives wrong answer because it can overflow.

→ We can solve this by giving more space to store. In place of int we can use long.

* Binary to Decimal Conversion :-

Binary :

Let 10101

1 0 1 0 1

$2\times10^4 + 2\times20^3 + 2\times10^2 + 2\times10^1 + 2\times10^0$

16 + 0 + 4 + 0 + 1

= 21

→ 1010 → 10

1 0 1 0

1×2^3 0×2^2 1×2^1 0×2^0

8 + 0 + 2 + 0

= 10

Binary → 0, 1 only two

So, We use power of 2.

Teacher's Signature

1 0 1 0 ⇒ 1010 %. 10

↓

0 × 2⁰

↙

1010 / 10 → 101 %. 10

↓

1 × 2¹ = 2

↙

101 / 10 → 10 %. 10

↓

0 × 2² = 0

↙

10 / 10 → 1 %. 10

1 × 2³ = 8

↙

1 / 10 → 0 (End)

8 + 2 + 0 + 0 = 10 (Ans)

Code:

```
int sum = 0;
int mul = 1;
while ( num > 0) {
    int rem = num %. 10;
    sum = sum + rem * mul;
    num = num / 10;
    mul = mul x 2;
}
cout << sum;
```

Teacher's Signature.....

→ 1's Complement :

101 → 010

How to Convert

101 % 10 = 1

1 ^ 1 = 0

101 / 10 = 10

10 % 10 = 0

0 ^ 1 = 1

10 / 10 = 1

1 % 10 = 1

1 ^ 1 = 0

1 / 10 = 0

↳ end:

Ans : 010

* Scope :

Who can access and from which it can access.

→ global variable

```
int num;
for ( ) {

}
```

Teacher's Signature

```
int num;
for ( ) {
    int rem; ⟶ It is valid only
                under the for loop.
}
```

If we access rem after for loop then it can't be accessed. It gives error.

* **Break & Continue :**

↓ Break

If code can read break, it break the program immediately and return.

↘ Continue

If we put Continue at any value then it can true for all other value.

Break :

```
for (int i = 0; i < 10; i++) {
    if (i == 6)
        break;
    System.out.println(i);
}
```

Output : 1 2 3 4 5

It can break the loop when i == 6.
It can't print the value after 6.

Teacher's Signature.....

Continue:

```
for (int i = 0; i < 9; i++) {
        if (i == 6)
            continue;

        System.out.print(i + " ");
    }
```

output : 0 1 2 3 4 5 7 8 9

It can only neglect 6 because where we see Continue we can break only that condition. It can work as previous for all next condition.

* Switch statement :

Check number is +ve, -ve or 0.

```
Switch (num) {
    case 1 :
        cout << " ";
        break;
    case 2 :
        Cout << " ";
        break;
    default :
```

Teacher's Signature

print for Week :

```
Switch (num){
    case 1:
            Cout << " Monday";
            break;
    case 2:
            Cout << " Tuesday";
            break;
    case 3:
            Cout << " Wednesday";
            break;
    case 4:
            Cout << " Thrusday";
            break;
    defautt;
            Cout << " Vacation Day";
}
```

* Function :

We have print number from 1 to 10.
It can done many times repeatly.

We can write same code many times.

By writing same code it takes more space.
So, we can use function in this.

Teacher's Signature

Function: If a task can done many times then we use function in this.

Function for checking a number is every or odd.

```
int main() {                    → main function.
    int num;
    cin >> num;
int ans = isEvenOdd(num);       → Function call
    Cout << ans;
    return 0;
}
```

```
int isEvenOdd(int num) {
↓                                  ↑ Arguments
return type   if (num % 2 == 0)
                  return 1;     → return statement
              else
                  return 0;     → return statement
}
```

→ Function Name.

* User have to print 1 to 10 many time. He wants to write a function. so he easily writes.

→
```
int main() {
    print1to10();   → call the function
    return 0;           to print.
}
```

Teacher's Signature

does not return anything because it can print here.

```
Void printS1to10 () {
    for (int i = 1;  i<= 10;  i++) {
            cout << i << " ";
        }
    return ;
}
```

Program Execution always starts from main function.

* Default Argument:
  ↳ If any argument is not given, then it is run according to default Argument

```
int main () {
    int num;
    cin >> num;
    prints (num);   → print 1 to 10
    prints ();      → print 1 to 5
    return 0;
}
Void prints (int num = 5) {
    for (int i = 1;  i<= num;  i++) {
            cout << i << " ";
        }
    return;
}
```

Page No.:
Date / /

* Factorial by Function :

```
int main () {
    int num;
    cin >> num;
    int ans = fact (num);
     Cout << ans;
    return 0;
}
int factorial (int num) {
     int fact = 1;
    for (int i = 1; i <= num; i++)
          fact = fact * i;
     return fact;
}
```

Teacher's Signature

Lecture-7

Function & Array

```
returntype functionname (parameter1, parameter2, ...){
    // code

    return value;
}
```

→ Why We need function :

Let us understant by example

You have given 8 number to check that it is even or odd.

You can Write the same code for all the input.

Here can write same code many time.

To minimize this problem we use function.

Sfn function we can write once for a specific work and use the same code for all the input value.

Now, we understand how to write function

Function for checking a number is Even or odd.

```
#include <iostream>
using namespace std;

Void isEvenOdd (int num){

    if((num %. 2) = = 0)
        Cout << " Even" << endl;

    else
        Cout <<"odd" <<endl;

  return;
}

int main () {
    int num1, num2, num3, num4;
    Cin >> num1 >> num2 >> num3 >> num4;
        isEvenOdd (num1);
        isEvenOdd (num2);
        isEvenOdd (num3);
        isEvenOdd (num4);

  return 0;
}
```

In this code, we can check for four number that it is Even or Odd.

With the help of function, we can write once and use it for many time.

Teacher's Signature.....

* Take an another Example : function for print table :

We have to Write table for the given number.

```
Void printTable (int num) {

    for (int i = 1;  i <= 10;  i++ ) {
        Cout << num << "*" << i << " = " << num * i;
    }
  return ;
}

int  main() {
     nt n;
     cin >> n;
     table (n);
     table (13);
  return 0;
}
```

* factorial of a number

$5! = 5 \times 4! \rightarrow 24 \rightarrow 120$

$4 \times 3! \rightarrow 6$

$3 \times 2! \rightarrow 2$

$2 \times 1! \rightarrow 1$

Teacher's Signature.....

```
int factorial (int num){
        int ans = 1;
    for (int i=1; i<= num; i++ ){
            ans = ans * i;
        }
    return ans;
}

int main() {
    int num;
    cin >> num1;

  int fact = factorial (num1);
   cout << fact;

return 0;
}
```

* function overloading :

In our previous problem we can give a fix number of parameter.

In we write a function to add 2 number then we give 2 parameter.
And in same function, we want to add 3 number, then it can't be possible.

Teacher's Signature.....

So, for this problem, We can use Method overloading.

The name of the function is same and the parameter is difference.

We have to write code for add, where we wanto add 2 number, in some case we add 3 number and in few we add upto four number.

```
int add (int num1, int num2) {
        int ans = num1 + num2;
    return ans;
  }
float add ( float num1, float num2) {
        float ans = num1 + num2;
  return ans;
  }
int add ( int num1, int num2, int num3) {
        int ans = num1 + num2 + num3;
   return ans;
  }

int main() {
     int a = add ( 9, 6);
       Cout << a;
         Cout << endl;
    int b = add ( 24, 9, 13);
       Cout << b;
       Cout << endl;
```

Teacher's Signature

```
    float c = 9.6 + 11.3;
      Cout << c;
      Cout << endl;
  return 0;
}
```

The above concept of function are known as function called by value.

Arrays

* A person want to store Apple, he rent a storeroom in nearby bulding. Room No: 109.

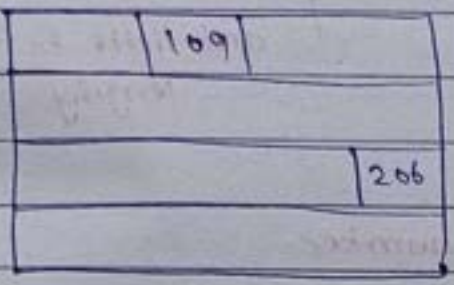
The person have coming more Apple, he rent one more storeroom is same bulding. Room No: 206.

The sale of apple is increased rapidly. He then rent 20 room in same bulding. But he want the room which are all continous.

The owner has alloted room no. 101 to 120 to the person.

Teacher's Signature

Previously :



In this, person has to remember the room number.

Now :

101	102	103	104	105	106	107	108
109	110	111	112	113	114	115	116
117	118	119	120				206

In this, person has to remember the room number of 1st room only. and rest of the room are continous.

Array : Collection of same datatype.

Why We need ?

Example : Sum of 3 number

```
int a = 3
int b = 10
int c = 13
```

Easily we do

Sum of 10 number

```
int a, b, c, d, e, f ...
```

easily we do

Teacher's Signature.....

Sum of 100 number
Make 100 variable
then sum
} takes some time and little bit lengthy.

Sum of 1000 number
Make 1000 variable
It take more time and very lengthy task.

So, for storing the same datatype element we use array.

For storing 1000 number:
int a1, a2, a3, a4, a5. . . .
By this way we can store
By seeing deeply,
only a(1), a(2), a(3), a(4) . . .
this changes.

So, we can done this in another way by array.

a[i] -> a[0], a[1], a[2] . . .

Array is a Collection of same data type.
It stored at Configous location.

Teacher's Signature

Page No.
Date / /

Type of element is same in array.

arr[] = type → int
→ float
→ char

Declaration :

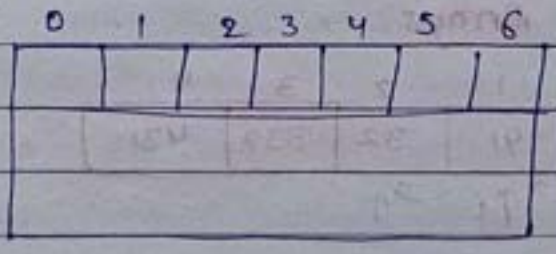
int arr[7]; → It takes 7 space in Memory to store integer.

0	1	2	3	4	5	6
9	2	6	0	-3	18	26

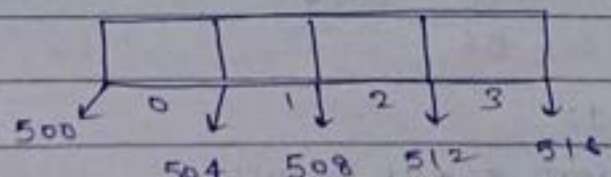
→ Why indexing is start from 0?

arr[0] = 9;
cout << arr[0]; print 9.

In Computer, it uses byte addressable.



Integer type array



Memory address

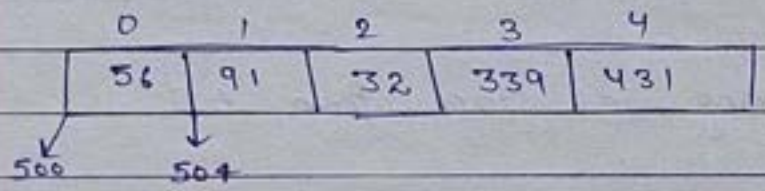
Teacher's Signature.....

We use indexing from 0 because for accessing the address in 1 based indexing can take more time.

In 1 based indexing:
Initial address + (index - 1) * size;

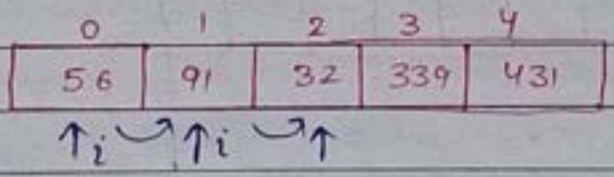
In 0 based indexing:
Initial address + index * size;

```
int arr[5];
```



Integer can take 4 byte to store an integer.

* Sum of an Array:



Size = 5

sum = 0

sum = sum + a[i]

= 56

i = 1;

sum = 56 + 91 = 147

i++;

i = 2

sum = 147 + 32 = 179

i++ ; i = 3

sum = 179 + 339 = 518

i++ ; i = 4

sum = 518 + 431 = 949

i++ ; (i = 5)

↳ Not present

loop break;

print sum = 949.

Code :

```
int main () {
    int arr[5];
    arr[0] = 56 , arr[1] = 91, arr[2] = 32 ;
     arr[3] = 339, arr[4] = 431;
     int sum = 0;
for (int i = 0; i<5; i++ ) {
     sum = sum + arr[i];
 }

cout << sum ;

return 0;
}
```

Teacher's Signature.....

Lecture - 8

(problems on Array)

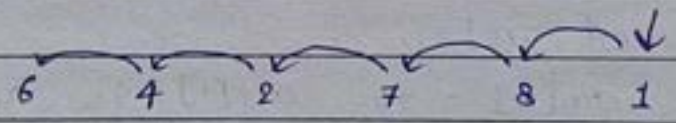
problem 1 : Reverse of an Array

Index :	0	1	2	3	4	5
	6	4	2	7	8	1

Approach 1

Traverse the array and print array from the last.

Array after reverse :



Arr_rev = {1, 8, 7, 2, 4, 6}

```
for (int i= arr.length -1 ;  i>=0 ; i--)
        cout <<arr[i] <<" ";
```

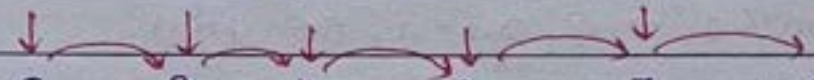
Problem 2: Maximum Element of an array

0	1	2	3	4	5
2	3	11	8	7	-2

Array includes with negative number also.

So, initialise MAX as INT_MIN:

Now, traverse the array and update Max is number bigger than Max found.



2 > INT_MIN → MAX = 2

3 > 2 → Max = 3

11 > 3 → Max = 11

8 < 11 → No change

7 < 11 → No change

-2 < 11 (No change)

Max = ~~INT_MIN~~ ~~2~~ ~~3~~ 11

Code:

```
int arr = {2, 3, 11, 8, 7, -2};
    int largest = INT_MIN;
for (int i = 0; i < arr.size(); i++) {
        if (arr[i] > largest)
            largest = arr[i];

Cout << largest;
```

Teacher's Signature.....

problem 3: print all odd number of array:

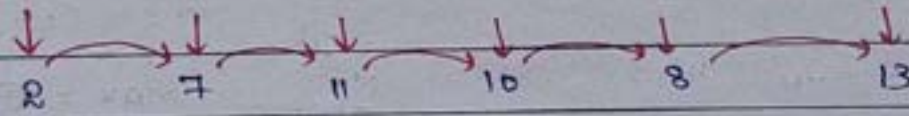
arr: {2, 7, 11, 10, 8, 13}

for odd:

condition

↳ When we divide by 2 remaineder comes 1.

① traverse the array and check an element by num%.2 → if ans = 1, then print otherwise even → do not print.



2%.2 = 0	7%.2 = 1	11%.2 = 1	10%.2 = 0	8%.2 = 0	13%.2 = 1
Even	odd	odd	Even	Even	odd
Not print	print	print	Not print	Not print	print

ans = 7 11 13

Code:

```
int arr = {2, 7, 11, 10, 8, 13};
for(int i = 0; i < arr.size(); i++){
    if(arr[i]%.2 == 1 || arr[i]%.2 == -1){
        Cout << arr[i] << " ";
    }
}
```

problem 4 : print all prime number from array.

arr = { 2, 3, 7, 11, 13 }

arr = { 2, 3, 4, 8, 7, 11, 16, 19 }

for prime :

↳ for all number, we have to check that it is divisible by 1 & self.

↳ the number have more than 2 factor the it is not prime.

8 : ~~1 x 2 x~~ 1, 2, 4, 8

More than 2 → not prime.

13 : 1, 13

Exactly 2 → Prime.

Code :

```
int arr[] = {2, 3, 4, 8, 7, 11, 16, 19};
    for (int i = 0; i<arr.size(); i++){
            prime(arr[i]);
      }
Void prime(int num){
        if (num < 2)
            return;
      for(int i = 2; i<=num-1; i++){
            if (num % i == 0)
                return;
            }
      Cout<<num<<" ";
 return;
}
```

Teacher's Signature.....

Problem 5: Rotate 1 place of Array:

arr: { 2 3 7 11 4 }

output: 4 2 3 7 11

process: 2 3 7 11 4

int num = 4

arr[i] = arr[i-1]

2 3 7 11 11

arr[i] = arr[i-1]

2 3 7 7 11

2 3 3 7 11

2 2 3 7 11

arr[0] = num

Ans: 4 2 3 7 11

Code:

```
int arr[] = {2, 3, 7, 1, -11, 8, 13};
        int num = arr[arr.size()-1];

for (int i = arr.length-2; i >= 0; i--){
        arr[i+1] = arr[i];
    }
    arr[0] = num;
for (int i=0; i<arr.length; i++){
        cout << arr[i] <<endl;
    }
```

Teacher's Signature

problem 6 : Maximum step to make the product 1

Example : {-2, 4, 0}

For product = 1

Even no. of (-1) and Even/odd number of 1.

{-2, 4, 0}

↓① ↓③ ↓①

-1 1 -1 → इसको -1 इसलिए बनाए है क्योंकि मेरे पास Negative number एक ही था, ↓ odd.

5 steps.

-1 x 1 x -1 = 1

→ If we have odd number of negative number then, we make one zero as (-1).

↳ If 0 is not present then make a negative number to 1.

↳ If 0 is present add 1 step

↳ If 0 is not present add 2 step

-ve → -1

+ve → 1

0 → -ve / +ve (-1/+1)

↓

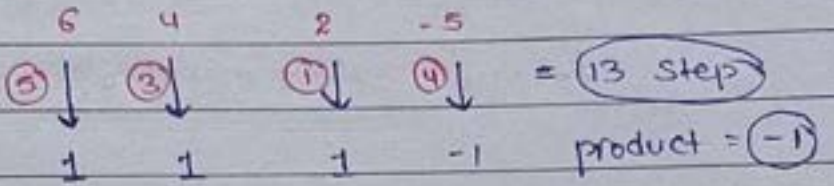
If 0 is not present then,

in few case (-ve → 1)

Even No. of (-1) x odd/even number of 1 = 1.

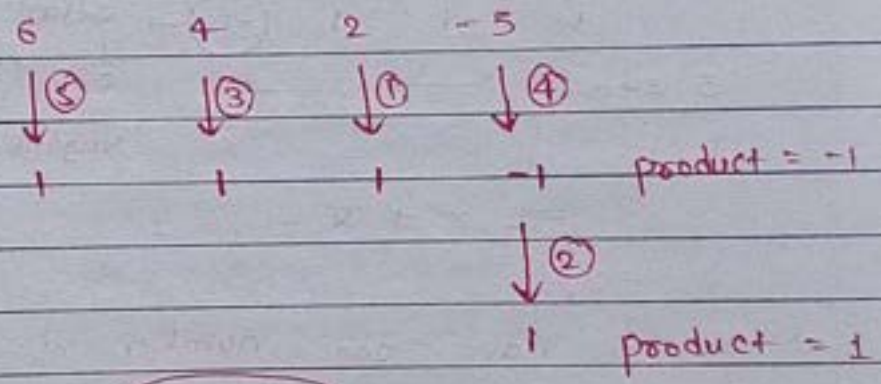
Teacher's Signature.....

Ex:



So, We make 1 (-1) as 1.

then add 2 step



Step = 15

code:

```
int Count_zero = 0;
int step = 0;
int multiplier = 1;

for (int i = 0; i < arr.length; i++){
    if (arr[i] > 0) {
        step = step + (arr[i] - 1);
        Multiplier = Multiplier * 1;
    }
    else if ( arr[i] < 0){
        step = step + (-1 - arr[i]);
        Multiplier = Multiplier * (-1);
    }
```

Teacher's Signature

```
else {
        step = step + 1;
        count_zero++;
    }

if (multiplier == 1 || count_zero > 0) {
        ans = step;
    }
else {
        ans = step + 2;
    }
return ans;
}
```

if Multiplier = 1
↳ then NO problem
↳ return step.

if Multiplier != 1
↳ Check count_zero > 0
↳ if count_zero > 0
↳ return step.
↳ if count_zero = 0
↳ then step = step + 2;

Lecture - 9

(Array problems)

1. Nested loop :

	0	1	2	3	4
arr :	2	3	13	8	7

print same array 5 times :

Approach 1 :

Repeat for loops 5 times :

Code :

```
for (int i=0;  i<n;  i++){
        Cout << arr[i];
}
for(int i = 0;  i<n;  i++){
        Cout<< arr[i];
}
```

But what if we have to print same array 100 times.

We use nested loop :

```
for(int i=0;  i<5;  i++){
    for(int j=0;  j<n;  j++){
        Cout << arr[j] << " ";
    }
    Cout << endl;
}
```

Teacher's Signature

problem 2:

arr: 2 3 13 8 7

output:

row	col					
0	5	2	3	13	8	7
1	4	2	3	13	8	
2	3	2	3	13		
3	2	2	3			
4	1	2				

↳ j < 5-i

col < 5-row

After each row the last col get eliminated.

code:

```
for (int row = 0;  row < 5 ;  row++ ) {
    for (int  col =0;  col < 5- row;  col++ ) {
            Cout << arr[col]<< " ";
        }
        Cout << endl ;
}
```

Teacher's Signature.....

Problem 3:

arr : 2 3 13 8 7

output :

row	col					
0	5	2	3	13	8	7
1	4	3	13	8	7	
2	3	13	8	7		
3	2	8	7			
4	1	7				

In 0st row → 0 element delete from first
(0 - 4) print

In 1st row → 1 element
(1 - 4) print

In 2nd row → 2 element
(2 - 4) print

In 3rd row → 3 element
(3-4) print

In 4th row → 4 element
(4 - 4) print

```
for (int i=0; i<5; i++){
    for (int j = i ; j<5; j++ ){
        cout << arr[j] <<" ";
    }
    cout <<endl;
}
```

Teacher's Signature

Problem 4:

array : 3 13 12 8 4

row	col					
0	(n-row)	4	8	12	13	3
1	(n-row)	8	12	13	3	
2	(n-row)	12	13	3		
3	(n-row)	13	3			
4	(n-row)	3				

For 1st row : 0 element deleted and print in reverse (4 → 0)

For 2nd row : 1 element deleted from last and print in reverse. (3 → 0)

For 3rd row : 2 element deleted from last and print in reverse. (2 → 0)

for 4th row : 3 element deleted from last and print in reverse (1 → 0).

for 5th row : (0 → 0)

Code :

```
for ( int row = 0; row < 5; row++ ) {
    for (int col = 4 - row; col >= 0; col--) {
        cout << arr[col] << " ";
    }
    cout << endl;
}
```

Teacher's Signature.....

Problem 5 :

array : { 3 13 12 8 4 }

output :

row	col					
0	4-0	4	8	12	13	3
1	4-1	4	8	12	13	
2	4-2	4	8	12		
3	4-3	4	8			
4	4-4	4				

↳ all in reverse order :

Code :

```
for (int i = 0;  i < 5;  i++){
        for (int j = 4;  j >= i ;  j--){
                cout << arr[j] << " ";
        }
        cout << endl;
}
```

Teacher's Signature

problem 6 : Find duplicate in an array

Example :

N = 4 Input : 0 to N-1

arr[] = {0, 3, 1, 2} 0 to 3

output = (-1)

↳ because No any element is repeated

Example :

N = 5

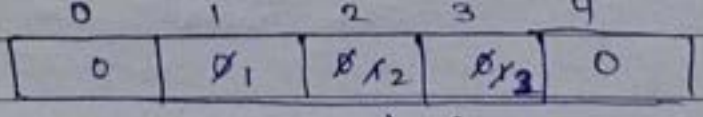
arr[] = { 2, 3, 1, 2, 3}

Output = {2, 3}. Input : 0 to N-1

0 to 4

2 3 1 2 3

No.	Count	
0	0	
1	1	
2	2	return because they are repeated.
3	2	
4	0	



greater than 1

print as output.

Teacher's Signature.....

Code :

```
int arr[] = {2, 3, 1, 2, 3};
    int n = 5;
    int Count[n] = {0};

for (int i=0;  i<n;  i++){
        int index = arr[i];
        Count[index]++;
    }

for(int i=0 ; i<n; i++){
        if (Count[i] > 1){
            Cout << i << " ";
        }
    }
return 0;
}
```

Teacher's Signature

Sorting

arr : 2 7 11 4

The data is sorted when this is either in ascending or descending.

ascending : 2 4 7 11

descending : 11 7 4 2

Sort of array of 0's, 1's and 2's

N = 5

arr[] = { 0, 1, 2, 0, 1 }

Output = { 0, 0, 1, 1, 2 }

Count_zero = 2

Count_one = 2

Count_two = 1

Code :

```
for (int i = 0; i < Count_zero; i++) {
    arr[i] = 0;
}
for (int i = Count_zero; i < Count_zero + Count_one; i++) {
    arr[i] = 1;
}
```

Teacher's Signature

```
for (int i = Count_zero + Count_one; i<n; i++){
        arr[i] = 2;
}
```

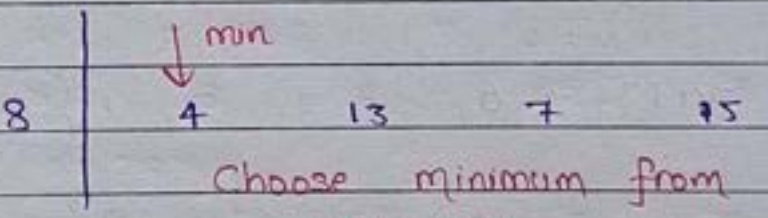
(Lecture - 10)

(Sorting : Basic)

* Selection Sort :

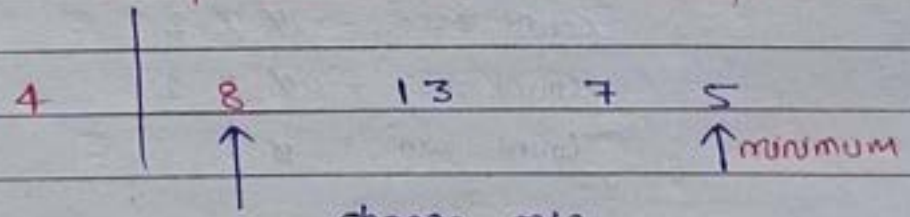
arr : {8, 4, 13, 7, 5}

Output : 4 5 7 8 13



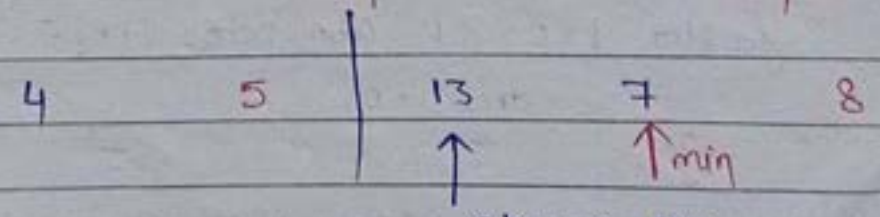
Choose minimum from this side

Swap this with 1st element



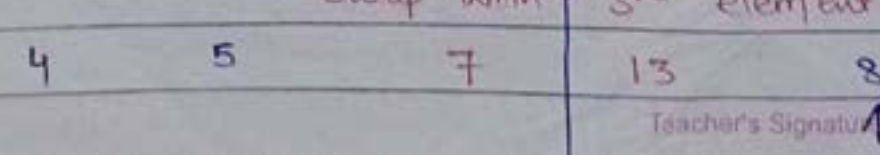
choose min

Swap with 2nd element.



Choose Min

Swap with 3rd element.



Teacher's Signatu

chose Minimum
swap with 4th element.

4 5 7 8 13

↳ final result

Code:

```
for (int i = 0;  i < n;  i++) {
        int index = i;
    for (int j = i ;  j < n-1;  j++) {
            if (arr[index] > arr[j]) {
                index = j;
            }
    }
        int temp = arr[index];
        arr[index] = arr[i];
        arr[i] = temp;
}
```

* Bubble sort

arr[]: { 7 2 4 3 6 5 }

In this we compare two value and swap if

In Every pass one element goes at its right place.
The last element goes at right place

We have do (n-1) pass to complete this.

Teacher's Signature.....

Page No.
Date / /

pass 1:

(7) (2) 4 3 6 5

7 > 2 (swap)

2 (7) (4) 3 6 5

7 > 4 (swap)

2 4 (7) (3) 6 5

7 > 3 (swap)

2 4 3 (7) (6) 5

7 > 6 (swap)

2 4 3 6 (7) (5)

7 > 5 (swap)

2 4 3 6 5 | 7

↳ got its right place.

pass 2:

(2) (4) 3 6 5 | 7

2 < 4 (No swap)

2 (4) (3) 6 5 | 7

4 > 3 (swap)

2 3 (4) (6) 5 | 7

4 < 6 (No swap)

2 3 4 (6) (5) | 7

6 > 5 (swap)

2 3 4 5 | 6 7

pass 3:

(2) (3) 4 5 | 6 7

2 < 3 (No swap)

2 (3) (4) 5 | 6 7

3 < 4 (No swap)

Teacher's Signature

2 3 (4) (5) | 6 7

4 < 5 (No swap)

pass 4:

(2) (3) 4 | 5 6 7

2 < 3 (No swap)

2 (3) (4) | 5 6 7

3 < 4 (No swap)

pass 5:

(2) (3) | 4 5 6 7

2 < 3 (No swap)

2 3 4 5 6 7

↳ sorted

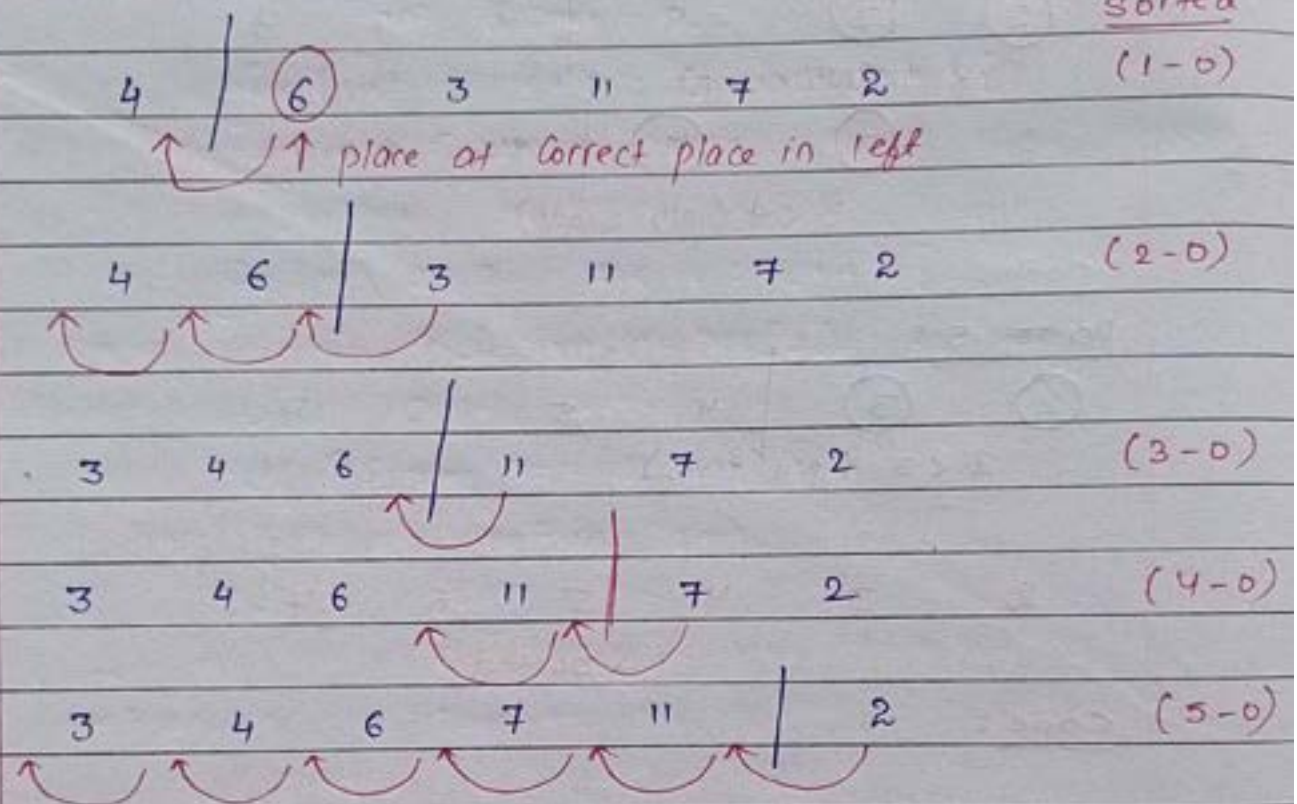
code:

```
for (int i=0; i<n-1; i++) {
    for (int j=0; j<(n-1)-i; j++) {
        if (arr[j] > arr[j+1]) {
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
```

Teacher's Signature.....

* Insertion Sort :

arr[] : 4 6 3 11 7 2



2 3 4 6 7 11

↳ sorted.

Code :

```
for (int i = 0 ; i < n-1 ; i++) {
    for (int j = i ; j >= 0 ; j--) {
        if (arr[j] > arr[j+1]) {
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
        else
            break;
    }
}
```

Teacher's Signature

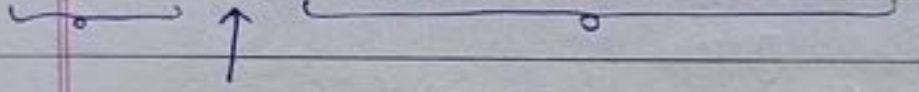
* Problem : Equilibrium point

The left side subarray and right side subarray of an element is equal then the element is known as Equilibrium point:

Example:

→

1	2	3	4	5
1	3	5	2	2



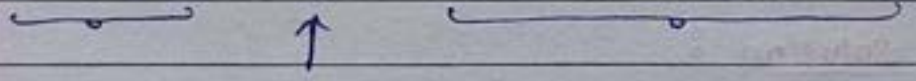
l_sum = 0 right = 3 + 5 + 2 + 2 = 12

l_sum != right

1 is not Equilibrium point

→

1	3	5	2	2



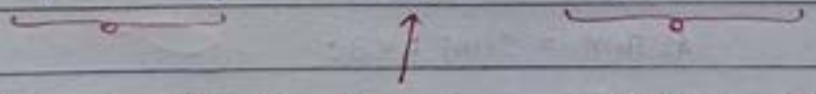
l_sum = 1 right = 5 + 2 + 2 = 9

l_sum != right

3 is not Equilibrium point

→

1	3	5	2	2



l_sum = 1 + 3 = 4 right = 2 + 2 = 4

l_sum = right

5 is the Equilibrium point.

Teacher's Signature.....

Code : (Brute force.) T.C : $O(N^2)$

```
for (int i = 0;  i < n;  i++) {
        int r_sum = 0;
        int l_sum = 0;
    for (int j = i+1 ; j < n; j++ ) {
            r_sum += arr[j];
    }
    for (int j=0;  j < i ; j++) {
            l_sum += arr[j];
    }
    if (l_sum == r_sum) {
            return i+1;
    }
}
return -1;
```

Optimal solution :

1	3	5	2	2

calculate sum

sum = 13

R_sum = sum = 13;

→

1	3	5	2	2

↑

R_sum = 13 - 1 = 12 > Not Equal.

L_sum = 0

Teacher's Signature.....

1 3 5 2 2

↑

R_sum = 12 - 3 = 9 > Not Equal
L_sum = 1

1 3 5 2 2

↑

R_sum = 9 - 5 = 4 > Equal
L_sum = 4

Equilibrium point.

Time Complexity : O(N)

Code :

```
if(n==1)
    return 1;
long totalSum = 0;
int res = -1;
long sum = 0;
for(int i=0; i<n; i++)
    totalSum += arr[i];

for(int i=1; i<n; i++) {
    sum += arr[i-1];

    if(totalSum - sum - arr[i] == sum) {
        res = i;
        break;
    }
}
return res == -1 ? -1 : res+1;
}
```

Teacher's Signature.....

(Lecture - 11)

Time and space Complexity

→ Time Complexity != Time taken

Time Complexity is defined as the time taken by the algorithm as our input size.

For same program time is different to run on different - different System.

* Asymptotic Notation :

```
for (int i = 0;  i < N;  i++){
        sum += arr[i];
}
```

Algorithm : Worst case : Big-O :

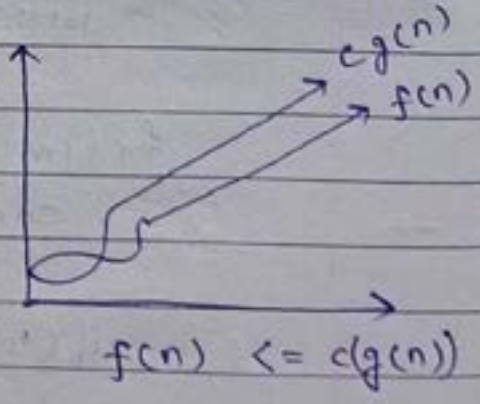
$n > n_0$

$3x + 2 <= C(n)$

$3x + 3 <= Cn$

$C >= 4$

Worst $O(N^2)$



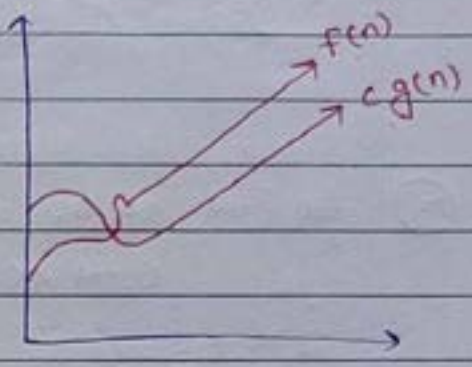
Teacher's Signature

Best case : Big-omega (Ω)

$f(n) >= c(g(n)$

$n >= n_0$

$c > 0, \quad n_0 >= 1$



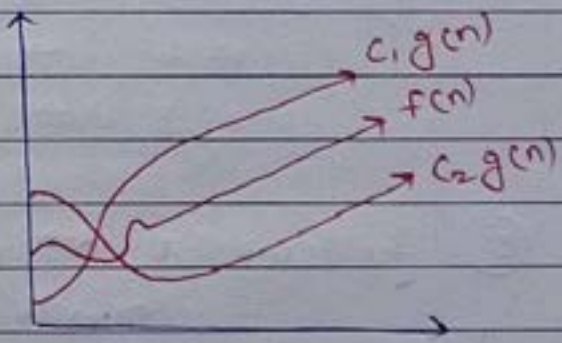
$f(n) = 3n+2, \quad g(n) = n$

$3n+2 >= cn$

$3n+2 >= \Omega$

Average Case : Big theta (θ)

$0 <= c_1 g(n) <= f(n) <= c_2 g(n)$



(1) Search an Element :

4 3 9 7 6 5 Input

```
for(int i=0; i<n; i++){
    if(arr[i]==input){
        cout<<i;
        break;
    }
}
```

Best : $\Omega(1)$

Worst : $O(N)$

Average : $\theta(N/2)$

Teacher's Signature.....

(2)
```
for (int i=0; i<n; i++){
    Cout <<i<<" ";
}
```
O(N)
Θ(N)
Ω(N)

(3)
```
for (int i=0; i<n; i++){
    Cout <<" pari";
]
```
O(N)
Θ(N)
Ω(N)

(4)
```
for(int i=0; i<n; i++){
    for (int j=1; j<=100; j++){
        Cout <<" Hey";
    }
}
```
100 × N
= 100N
↳ neglect constant
O(N)

(5)
```
for (int i=0; i<n; i++){
    Sum += i+1;
}
```
O(N)

(6)
```
Sum = n(n+1)/2;
```
O(1)

(7)
```
for (int i=1; i<=n; i++){
    for (int j=1; j<=n; j++){
        Cout <<" DSA";
    }
}
```
O(N²)

(n+n+n+n+ ... + n times)
(n × n) = n²

Teacher's Signature

(8) for (int i = 1; i <= n; i++)
    for (int j = 1; j <= i; j++)
        cout << "placement";

For i = 1 → j = 1 to 1 = 1
i = 2 → j = 1 to 2 = 2
i = 3 → j = 1 to 3 = 3
! !
i = n → j = 1 to n = n

$(1 + 2 + 3 + 4 + \ldots + n)$

$\frac{n^2+n}{2} \Rightarrow O(n^2)$

(9) for (int i = 2; i <= n; i++) {
    for (int j = 1; j <= i^2; j++) {
        cout << "Rohit";
    }
}

For i = 2 → j = 1 to 4 = 4
i = 3 → j = 1 to 9 = 9
i = 3 → j = 1 to 16 = 16
! !
i = n → j = 1 to n^2 = n^2

$(4 + 9 + 16 + \ldots + n^2)$

~~$O(n^2)$~~

Teacher's Signature

$1^2 + 2^2 + 3^2 + 4^2 + \cdots + n^2$

$$\frac{n(n+1)(2n+1)}{6} \Rightarrow \frac{(n^2+n)(2n+1)}{6}$$

$$= \frac{2n^3 + n^2 + 2n^2 + n}{6}$$

$$\Rightarrow O(n^3)$$

(10)
```
for (int i = 1; i<=n; i++)
    for (int j=1; j<= m; j++)
        Cout << "Complexity";
```

For
- $i = 1 \rightarrow j = 1$ to $m = m$
- $i = 2 \rightarrow j = 1$ to $m = m$
- $i = 3 \rightarrow j = 1$ to $m = m$
- ⋮
- $i = n \rightarrow j = 1$ to $m = m$

$m + m + m + m + \cdots + n$ times

$$O(m \cdot n)$$

(11)
```
for (int i= 1; i<= n; i++)
    for (j=1; j<= i^2; j++)
        for (k= 1; k<= n/2; k++)
            cout << "pari";
```

Teacher's Signature

For $i=1 \rightarrow j=1$ to $1 \rightarrow k \rightarrow n/2$ time

For $i=2 \rightarrow j=1$ to $4 \rightarrow k \rightarrow 4 \times n/2$ times

For $i=3 \rightarrow j=1$ to $9 \rightarrow k \rightarrow 9 \times n/2$ times

For $i=n \rightarrow j=1$ to $n^2 \rightarrow k \rightarrow n^2 \times n/2$ times.

$$\underbrace{(1^2 + 2^2 + 3^2 + 4^2 + \cdots + n^2)}_{0}\ n/2$$

$$(n^3) \times n/2$$

$$= O(n^4)$$

(12) for (int i = 1; i < n; i = i * 2)
Cout << "logarithm";

$O(\log n)$

1	2	4	8	16	n
2^0	2^1	2^2	2^3	2^4	2^k

k+1 times

$n = 2^k$

$k = \log_2 n$

Teacher's Signature

(13)
```
for( i= n/2 ; i<=n; i++)        → n/2
    for ( j= 1 ; j<=n/2 ; j++)  → n/2
        for (k=1 ; k<n; k++)    → n
```

All are independent:

$n \times n/2 \times n/2$

$O(n^3)$

(14)
```
for ( int i = n/2;   i<=n ; i++)     → n/2
    for ( j= 1 ;   j<=n ;  j++ )     → n
        for( k=1 ;  k<n ;  k=k*2)    → logn
```

$\frac{n}{2} \times n \times \log n$

Ans: $O(n^2 \log n)$

(15) Decimal to Binary:

$O(\log n)$

(16)
```
for ( int  i=n/2 ;  i<=n :  i++)          → n/2
    for ( int j=1 ;  j<=n ;  j= 2 * j)    → logn
        for ( int k = 1 ;  k<=n;  k= 2 * k)  → logn
```

$n/2 \times \log n * \log n$

$O(n \log n^2)$

Teacher's Signature.....

(17)

```
For ( int i = 1 ; i <= n ; i++ )
    for ( int j = 1 ; j <= n ; j = j + 1 )
        Cout << " RAM" ;
```

For $i = 1$: $j = 1$ to n $\Rightarrow n$

For $i = 2$: $j = 1, 3, 5, 7 \ldots n$ $\Rightarrow n/2$

For $i = 3$: $j = 1, 4, 7, 10, \ldots n$ $\Rightarrow n/3$

For $i = n$: $j = 1, n$ $\Rightarrow n/n$

$$n + n/2 + n/3 + \cdots + n/n$$

$$n\left[1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right]$$

$$n(\log_e n)$$

$$O(n \log n).$$

Teacher's Signature.....

lecture - 12

Time and space Complexity

Time Complexity states that the amount of time taken by an algorithm to run as a function of length of the input.

Compare : $O(\log n)$ & $O(\sqrt{n})$

1.	$n = 10$	3.3
2.	$n = 100$	10
3.	$n = 1000$	100
4.	$n = 10,000$	1000
6.	$n = 10^6$	$100000 = 10^5$

$O(\log n)$ is best.

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n)$$

$$< O(n^2) < O(n^3) < O(2^n) < O(n!)$$

$O(\log n)$:

↳ 1 2 4 8 16 32 ... n

↳ n $n/2$ $n/4$ $n/8$ ... 1

Teacher's Signature.....

Space Complexity states that amount of space taken by an algorithm to run a function of the length of input.

Auxiliary space : Not included input size

Total time Complexity : Include input size.

* Input arr[n]

Create an array of size n and copy the element.

Input :

20	10	25	15	27	17

ans :

20	10	25	15	27	17

Auxiliary space : O(N)

Total time taken : N + N = 2N

O(N)

*
```
int arr = {10, 20, 30, 40, 50};
int ans[5];
for(int i=0; i<5; i++){
        ans[i] = arr[i] * 10;
}
for(int i=0; i<5; i++){
        cout << ans[i] << " "<<;
}
```

T.C : O(N)

S.C : O(N)

T.T.C : O(N)

output : 100, 200 300 400 500

Teacher's Signature.....

→ For prime Number:

We have to check from 2 to (N/2).

→ prime No: check until $\sqrt{n}$ (optimal)

* prime No:

```
if (N<2)
    return 0;
for (int i=2; i<= sqrt(N); i++) {
        if (N%i == 0)
            return 1;
    }
```

* Insertion Sort:

```
for (int i=0; i<n-1; i++) {
    for (int j=i; j>=0; j--) {

    }
}
```

i = 0	i = 1	i = 2	i = 3	- - -	i = n
0 to 0	1 to 0	2 to 0	3 to 0		n to 0

$$[0 + 1 + 2 + 3 + 4 + 5 + 6 - - - + n]$$

$$\frac{n(n+1)}{2} = \frac{n^2+n}{2} = O(n^2)$$

↓

Time Complexity

Space Complexity : O(1)

Teacher's Signature

* Bubble sort :

```
for (int i = 0;  i < n-1;  i++)
    for (int j = 0;  j < n-1-i;  j++)
```

$i = 0 \Rightarrow j = 0$ to $n-1$

$i = 1 \Rightarrow j = 0$ to $n-2$

$i = 2 \Rightarrow j = 0$ to $n-3$

$i = 3 \Rightarrow j = 0$ to $n-4$

$i = n \Rightarrow j = 0$ to 1

$$\{1 + 2 + 3 + 4 + \cdots + n-3 + n-2 + n-1\}$$

$$\frac{n(n+1)}{2} \Rightarrow O(n^2)$$

Teacher's Signature.....

Lecture - 13

Binary Search

Binary Search:

↳ Applied only on sorted array

Index	0	1	2	3	4
Array	2	7	11	13	18

Linear search:

```
for (int i=0; i<n; i++)
    if (arr[i]==target)
        cout << "i";
```

O(N)

Binary search (Example):

0	1	2	3	4	5	6	7
2	4	7	18	23	29	32	38

Target = 32

Step 1: Find mid:

(1) * $mid = \frac{low + high}{2}$

$low = 0$, $high = 7$

$mid = \frac{0+7}{2} = 3$

Teacher's Signature

Check (arr[mid] == target)

arr[3] == 32

18 == 32 (False)

18 < 32

arr[mid] < target

⇓

Start = mid + 1 = 4

end = 7

(2) * Mid = $\frac{4+7}{2}$ = $\frac{11}{2}$ = 5

arr[5] = target

arr[5] == 32

29 == 32 (False)

29 < 32

arr[mid] < target

Start = mid + 1 = 6

end = 7

(3) * Mid = $\frac{6+7}{2}$ = 6

arr[6] == target

32 == 32 (True)

return 6

Algorithm :

① Find mid element

mid = (low + high) / 2;

② Check arr[mid] is equal to target

case 1: arr[mid] = target

return mid

case 2: target > arr[mid]

search in right

low = mid+1

case 3: target < arr[mid]

search in left

high = mid - 1;

Code :

```
int mid;
int low = 0;
int high = n-1;

while ( low <= high) {
    mid = (low + high)/2;

    if ( arr[mid] == target) {
        cout << mid;
        return;
    }
```

Teacher's Signature

```
        else if (target > arr[mid]) {
                low = mid + 1;
            }
        else {
                high = mid - 1;
            }
    }
```

Time Complexity:

Worst case : O(log N).

* Left most and right most index :

Given : Sorted array
↳ With duplicate element

Task : find first & last occurence of element x.

If element is not present return {-1, -1} as pair.

Example:

N = 9

V[] : 1 3 5 5 5 5 67 123 125

X = 5

output = 2 5

1	3	5	5	5	5	67	123	125

For finding leftmost occurence of x,

0	1	2	3	4	5	6	7	8
1	3	5	5	5	5	67	123	125

$$mid = \frac{0+8}{2} = 4$$

arr[mid] == target (True)

leftMost = ~~-1~~ ~~4~~ 2 end = mid - 1 :

0	1	2	3
1	3	5	5

$$mid = \frac{0+3}{2} = 1$$

arr[mid] == target (False)

arr[mid] < target

↳ start = mid + 1

2	3
5	5

mid = 2

arr[mid] == target (true)

end = mid - 1

↳ No Exist.

leftMost = 2

Teacher's Signature

For rightMost :

0	1	2	3	4	5	6	7	8
1	3	5	5	5	5	67	123	125

rightMost = -1 ~~4~~ 5 mid = 4

arr[mid] == target (true)

start = mid + 1

5	6	7	8
5	67	123	125

mid = 6

arr[mid] = target (false)

arr[mid] > target

end = mid - 1

5

5

mid = 5

arr[mid] = target (true)

start = mid + 1

↳ Not Exist.

rightMost = 5

return {2, 5}

code :

```
pair<long, long> indexes(vector<long, long> v, long long x){
        long long start = 0;
        long long end = v.size() - 1;
        long long leftindex = -1;
        long long rightindex = -1;
        long long mid;
```

Teacher's Signature.....

```
While ( start <= end) {
    mid  = end + (start -end) / 2;
        if (v[mid] == x){
            leftindex = mid,
                end = mid-1;
            }
        else if (v[mid] > x){
                end = mid-1;
        } else {
                start = mid+1;
            }
}
start = 0;
end = v.size() - 1;
 While ( start <= end ) {
        mid =  end  + (start - end) / 2:
        if (v[mid] == x){
            rightindex  = mid;
            start =  mid+1;
        }
        else if (v[mid]  > x ){
                end = mid-1;
            }
        else {
                start = mid + 1;
            }
    }
return {leftmost leftindex, rightindex};
}
};
```

Teacher's Signature.....

* Search insert position of K in a sorted array:

num = {1, 3, 5, 6} target = 5

output = 2

case 1: target present in array

↳ then return index.

case 2: target does not present in array

num = {1, 3, 5, 6} target = 2

output = 1

For this we apply Binary search :-

1 (3) 5 6

$Mid = \frac{0+3}{2} = 1$

2 < 3 index = 1

{ 1 }

$\frac{0+0}{2} = 0$

index = 0 + 1 = 1

1 < 2

→ अगर target < arr[mid]

↳ index = mid

→ अगर target > arr[mid]

↳ index =

Teacher's Signature.....

Code:

```
int searchInsertK(vector<int> arr, int N, int target){
        int start = 0;
        int end = N-1;
        int index;

    while (start <= end) {
        int mid = start + (start - end)/2;
        if (arr[mid] == target) {
            index = mid;
            break;
        }
        else if (arr[mid] < target){
            start = mid + 1;
            index = mid + 1;
        }
        else {
            end = mid - 1;
            index = mid;
        }
    }
    return index;
  }
};
```

Teacher's Signature.....

* Peak Element :

Given : Array arr[]
Size N

Task : Return index of any one of its peak element.

peak element is an element whose value is not smaller than the adjacent element.

Example :

N = 3
arr[] = {1, 2, 3}

↓

{1, 2, 3}

1 > 2 (false) Not peak

↓

{1, 2, 3}

2 > 1 (True) } False Not peak
2 > 3 (False }

↓

{1, 2, 3}

3 > 2 (True) peak Element.
3 last element

Teacher's Signature.....

→ arr[i-1] < arr[i] > arr[i+1]

Code:

```
int peakElement (int arr[], int n) {
        int start = 0;
        int end = n-1;

        while (start <= end) {
            int mid = end + (start - end) / 2;

        if (arr[mid] > arr[mid + 1] && arr[mid] >
                    arr[mid - 1])
            return mid;

        else if (arr[mid] > arr[mid - 1] &&
                arr[mid] < arr[mid + 1])
            start = mid + 1;

        else
            end = mid + 1;
        }
        return -1;
    }
};
```

Teacher's Signature.....

Lecture - 14

Binary Search

* Find smallest absent element :

Example :

0	1	2	3	4	5	6
0	1	2	3	4	6	7

↑

(5) missing

```
for (int i = 0; i <= n; i++)
    if (arr[i] != i)                    O(n)
        return i;
return n;
```

By Binary search :

0	1	2	3	4	5
0	1	2	3	5	8
low					high

$$mid = \frac{0+5}{2} = 2$$

arr[mid] = mid (true)

→ index 2 tak sahi hai ab mid ke
aage check karenge

start = mid + 1.

Teacher's Signature.....

3 | 4 | 5

3 | 5 | 8

low — high

$mid = \frac{3+5}{2} = 4$

arr[mid] == mid (false)

mismatched found

ans = arr[mid] = 5

↳ check in left

high = mid - 1.

3

3

low

high

$mid = \frac{3+3}{2} = 3$

arr[mid] = mid (true)

Final answer

↳ index = 4

element = 5

Code:

```
int start = 0, end = N-1, mid, index;
while (start <= end) {
    mid = end + (start - end)/2;
    if (mid < arr[mid]) {
        index = mid;
        end = mid - 1;
    }
```

Teacher's Signature.....

```
        else {
            index = mid + 1;
            start = mid + 1;
        }
    }

    return mid;
}
```

* Rotate Array :

Arr : Sorted in ascending, All are distinct Element

Size : N

Rotate D times

D = 0	1	2	3	(4)	} repeat this
D = 1	4	1	2	(3)	
D = 2	3	4	1	(2)	
D = 3	2	3	4	(1)	
D = 4	1	2	3	4	

So the the series is repeat after N Element.

Teacher's Signature.....

* search in a rotated array :

Sorted and rotated array
Size = N distinct element

Find index of given element

N = 9

A[] = { 5, 6, 7, 8, 9, 10, 1, 2, 3 }

key = 10

l = 0 h = 8

Output = 5

If Not present : return -1,

0	1	2	3	4	5	6	7	8
5	6	7	8	9	10	1	2	3

↓
pivot

key = 10

```
if (arr[mid] == key)
    return mid;
```

$mid = \frac{0+8}{2} = 4$

arr[4] = 9

9 < 10

Teacher's Signature .....

{ arr[mid] < target }

↳ two condition

1) A[l] <= key || key < A[mid]

↳ end = mid - 1

2) condition 2

↳ start = mid + 1;

{ arr[mid] > target }

↳ if (arr[start] < target)

↳ end = mid - 1 ;

↳ if (arr[start] > target)

↳ start = mid + 1;

code :

```
int search ( int A[], int l, int h, int key) {
    int index = -1;
    while (l <= h) {
        int mid = h + (l - h) / 2;

        if ( A[mid] == key){
            return mid;
        }
        else if ( A[mid] > A[l]) {
            if ( A[mid] < key || key < A[l]) {
                l = mid + 1;
            }
```

```
            else {
                h = mid - 1;
            }
        }
        else {
            if (A[l] <= key || key < A[mid]) {
                h = mid - 1;
            }
            else {
                l = mid + 1;
            }
        }
    }
    return index;
  }
};
```

* Allocate Minimum Number of pages :

Given : N Books
Each have A_i pages
M students
need to allocated Contiguous Books.
Each student get atleast one Book.

0	1	2	3
10	15	10	20

M = 2

Teacher's Signature.....

	A	B		Diff
10	10	15, 10, 20	45	35
25	10, 15	10, 20	30	(5) ✓
35	10, 15, 10	20	20	15

In this we can use binary search

Example:

N = 4

A[] = {12, 34, 67, 90}

Student M = 2

{12}	{34, 67, 90}	Max : 191
{12, 34}	{67, 90}	Max : 157
{12, 34, 67}	{90}	Max = 113

Take Min ↓ Answer.

Answer nahi Mil rha hai to
↳ Right side

Answer mil rha hai to
↳ Left side.

Let Explain With an Example:

Teacher's Signature.....

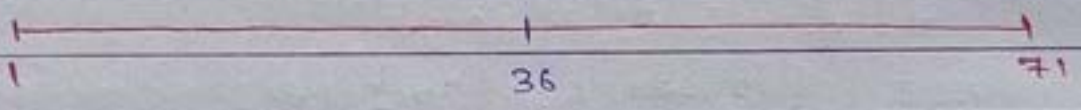
Example:

0	1	2	3
10	20	10	31

Start = 1

End = sum of No. of pages

end = 71



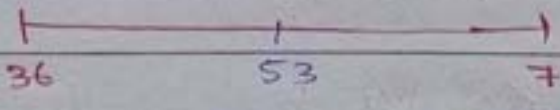
$mid = \frac{start + end}{2} = 36$ ↳ Ek student

S1 = 10 + 20 + ~~15~~ Ko Max 36 page.

S2 = 10 + ~~31~~ Not possible

S3 = 31

go to Right side.



$mid = \frac{36 + 71}{2} = \frac{107}{2} = 53$

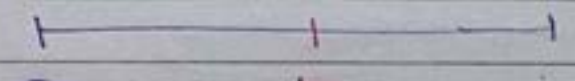
possible { S1 = 10 + 20 + 10 + ~~31~~

S2 = 31

Go to left.

Teacher's Signature

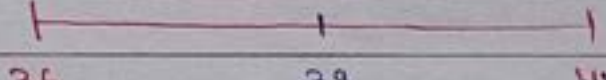
S3 पे Answer मिल रहा है लेकिन ये possibility hai ki usse kam pe bhi answer Mil Jaye.



$$Mid = \frac{36+53}{2} = 44$$

possible { S1 = 10 + 20 + 10 + ~~1~~
S2 = 31

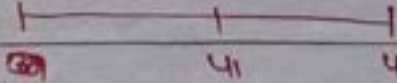
Left



$$Mid = \frac{36+43}{2} = 39$$

Not possible { S1 = 10 + 20 + ~~10~~
S2 = 10 + ~~21~~
S3 = 31

Go to Right



$$Mid = \frac{39+43}{2} = \frac{82}{2} = 41$$

possible { S1 = 10 + 20 + 10 + ~~1~~
S2 = 31

Go to left

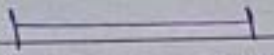
Teacher's Signature.....

S1 = 10+20+10 (Mid = 40)

S2 = 31

possible left

Condition fails :



{ Answer : 40 }

Code :

```
Public Static boolean isTrue (int A[], int N, int M,
                              int mid) {
    int student = 1;
    int sum = 0;

    for( int i = 0; i<N; i++) {
        sum = sum + A[i];
        if (sum > mid) {
            student++;
            sum = A[i];
            if ( student > M) {
                return false;
            }
        }
    }
    return true;
}
```

Teacher's Signature

```
public static int findPages(int []A, int N, int M){

    if (N < M) return -1;
    int sum = 0;
    int start = A[0];
    int ans = Integer.MAX_VALUE;
    for(int i=0; i<N; i++){
        sum += A[i];

        if( A[i] > start){
            start = A[i];
        }
    }
    int end = sum;
    while ( start <= end){
        int mid = start + (end - start)/2;

        if (isTrue ( A, N, M, mid ){
            ans = Math.min (ans, mid);
            end = mid - 1;
        }
        else {
            start = mid + 1;
        }
    }
    return ans;
  }
};
```

Teacher's Signature.....

Lecture -15

Data structure and Algorithm

* Search in a rotated Array :

0	1	2	3	4	5	6	7	8
5	6	7	8	9	1	2	3	4

key = 4

By Binary search :

Originally,

	1	2	3	4	5	6	7	8	9
Rotate 1:	9	1	2	3	4	5	6	7	8
Rotate 2:	8	9	1	2	3	4	5	6	7
Rotate 3:	7	8	9	1	2	3	4	5	6

In Rotate Array,

It is sorted in one direction only.

← or →

*

0	1	2	3	4	5	6	7	8
5	6	7	8	9	1	2	3	4

$\frac{0+8}{2} = 4$

arr[4] == key

↳ False.

Teacher's Signature.....

Check :

arr[start] <= arr[mid]

5 <= 9

↳ true.

↳ Check

arr[start] <= key <= arr[mid]

↳ True

↳ check in left

↳ False

↳ check in right

A/c to problem

↳ it is false

↳ go to right

*

5	6	7	8
1	2	3	4

$$\frac{5+8}{2} = 6$$

arr[6] == key

↳ False.

Check :

arr[start] <= arr[mid]

↳ true.

Check :

arr[start] <= key <= arr[mid]

↳ false

↳ go to right

7 | 8

3 | 4

$mid = \frac{7+8}{2} = 7$

arr[mid] == key

↳ False

go to right

8

4

$mid = \frac{8+8}{2} = 8$

arr[mid] == key

↳ true.

return mid.

Code:

```
int search (int A[], int l, int h, int key) {
        int index = -1;
        while ( l <= h) {
            int mid = h+ (l-h) / 2;

        if ( A[mid] == key) {
                return mid;
        }
```

```
else if ( A[mid] > A[l] ) {
        if ( A[mid] < key || key < A[l] ) {
                l = mid + 1;
            }
        else {
                h = mid - 1;
            }
    }
  else {
        if ( A[l] <= key || key < A[mid] ) {
                h = mid - 1;
            }
        else {
                l = mid + 1;
            }
      }
  }
  return index;
}
```

Aggressive Cows

Given : An Array
↳ consist of n integer
Array denote the position of stall.

K : No. of cows

Task :
Assign stalls to K cows
↳ So the distance b/w two cows are maximum.

Example :

N = 5 K = 3
↓ ↳ No. of cows
No. of stalls

Stall No : 1 2 4 8 9

3 cow ko stall me dalna hai Maximum distance pe taki koi ek dusre se ladai na kare.

1 2 4 8 9 min

$C_1 \xleftarrow{1} C_2 \xleftarrow{2} C_3$ ⇒ (1)

$C_1 \xleftarrow{3} C_2 \xleftarrow{4} C_3$ ⇒ (3)

$C_1 \xleftarrow{3} C_2 \xleftarrow{5} C_3$ ⇒ (3)

Teacher's Signature.....

Max = 3

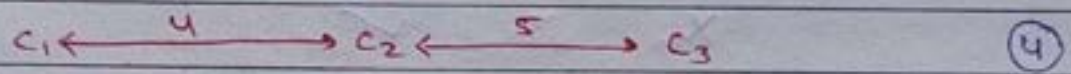
Example 2

n = 5 K = 3

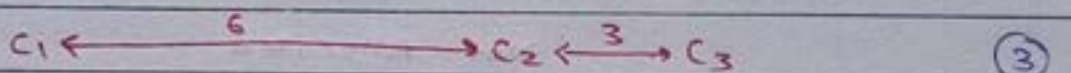
10 1 2 7 5

↳ sort

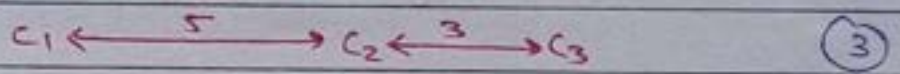
1 2 5 7 10



(4)



(3)



(3)

Maximum = 4

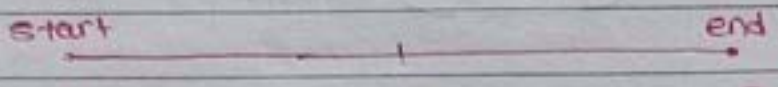
start from taking distance → 1

If answer found (No. of cows allocated)
↳ Move right

If answer Not found (No. of cows not allocated)
↳ Move left

start = 1
end = Max array - Min value.

Teacher's Signature.....



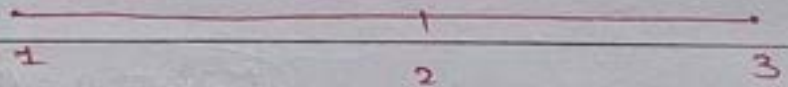
1 4 8

$mid = \frac{1+8}{2} = 4$

1	2	4	8	9
C1	2-1	4-1	8-1	9-8 = 1
	=1	=3	=7	X
	X	X	✓	
			C_2	

Not Allocated

high = mid - 1



1 2 3

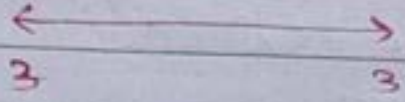
$Mid = \frac{1+3}{2} = 2$

1	2	4	3	9
C1	2-1=1	4-1	8-4	
	X	=3	=4	
		C_2	C_3	

Allocated

answer = 2

low = mid+1.



mid = 3

1	2	4	8	9
C1	2-1	4-1	8-4	
	= 1	= 3	= 4	
	X	C_2	C_3	

Allocated

Answer = 3

low = mid + 1 = 4

low > high → condition fails.

```
start = 1
end = Max - Min
int ans = -1
While ( start <= end ){
int mid = ( start + end) / 2;
    int Count = 1;
int last_pos = arr[0];

for(int i = 0; i<n; i++ ){
    if ( arr[i] - last_pos >= mid ){
        Count ++ ;
        last_position = arr[i];
    }
```

Teacher's Signature.....

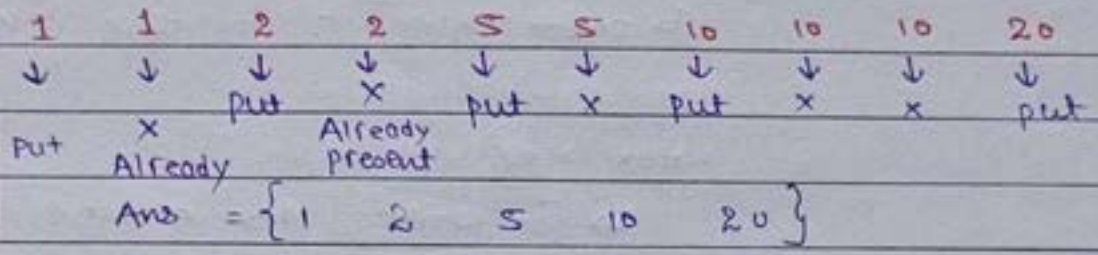
```
            if (count == k) {
                break;
            }
        }
        if (count == k) {
            ans = mid;
            start = mid + 1;
        }
        else {
            end = mid - 1;
        }
    }
    return ans;
}
```

Teacher's Signature.....

Lecture - 16

Two pointer

* Remove Duplicate from Array :



Ans = { 1 2 5 10 20 }

Brute Force :

```
int Note = 0;
int arr[N];
int index = 0;

for(int i = 0; i < N; i++) {
    if(A[i] != Note) {
        arr[index] = A[i];
        Note = A[i];
        index++;
    }
}

for(int i=0; i<index; i++) {
    cout << Arr[i] << " ";
}
```

T.C : O(N)

S.C : O(N)

Optimal Approach :

0	1	2	3	4	5	6	7	8	9
1	1	2	2	5	5	10	10	10	20

int Note = 0

↳ sabse pahle 0 initialise karenge.

index = 0

↳ for writing new element and increase the index.

1	1	2	2	5	5	10	10	10	20

↑

unique because Note ≠ pahle se zero hai

then,

A[index] = A[i];

Note ≠ 1 daal denge.

Note = 1;

index++;

1	1	2	2	5	5	10	10	10	20

↑ ↑ → unique

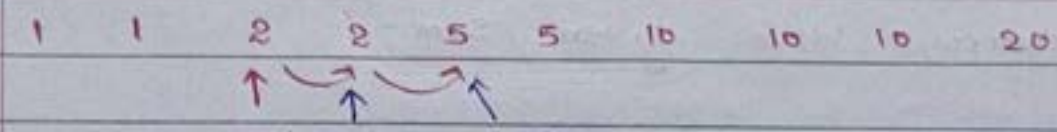
Not unique

A[index] = A[i];

Note = A[i];

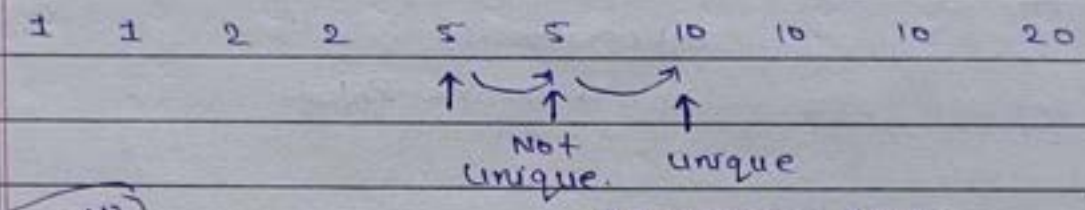
index++;

Note = 2



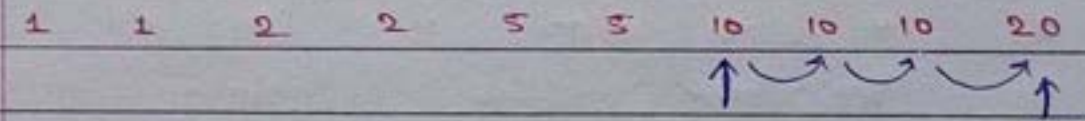
Note = 5

```
A[index] = A[i];
Note = A[i];
index++;
```



Note = 10

```
A[index] = A[i];
Note = A[i];
index++;
```



Note = 20

```
A[index] = A[i];
Note = A[i];
index++;
```

Code:

```
int note = 0;
int index = 0;
for (int i=0; i<n, i++) {
        if (note != a[i]) {
            a[index] = a[i];
            note = a[i];
            index++;
        }
    }
    return index;
}
```

Teacher's Signature

* Subarray With given Sum :-

2 4 5 10 11 13 17

sum = 19

If subarray present
↳ return true

otherwise
↳ return false.

Brute force Approach :

check one by one

2 4 5 10 11 13 17

→ 2 != 19
→ 2 + 4 = 6 != 19
→ 2 + 4 + 5 = 19 != 19
→ 2 + 4 + 5 + 10 != 19
→ 2 + 4 + 5 + 10 + 11 != 19
→ 2 + 4 + 5 + 10 + 11 + 13 != 19
→ 2 + 4 + 5 + 10 + 11 + 13 + 17 != 19
→ 4 != 19
→ 4 + 5 != 19
→ 4 + 5 + 10 = 19 (return true)

Teacher's Signature

Brute force Approach : $O(N^2)$

```
for (int i = 0; i<n; i++) {
        int sum = 0;
        for (int j= i; j<n; j++) {
            sum += arr[j];
            if (sum == givensum) {
                return true;
            }
        }
}
return false;
```

* Two pointer :

2 4 11 17 25 29

Find 2 number whose sum is equal to 40.

① Brute Force Approach :

Take all possible operation :

2 + 4 = 6 ✗
2 + 11 = 13 ✗
2 + 17 = 19 ✗
2 + 25 = 27 ✗
2 + 29 = 31 ✗

4 + 11 = 15 ✗
4 + 17 = 21 ✗
4 + 25 = 29 ✗
4 + 29 = 33 ✗

11 + 17 = 28 ✗
11 + 25 = 36 ✗
11 + 29 = 40 ✓ return true.

Teacher's Signature

code:

```
for (int i = 0; i<n-1; i++) {
        for (int j = i+1; j<n; j++) {
                if (arr[i] + arr[j] == sum) {
                        return true;
                }
        }
}
return false;
}
```

$O(N^2)$

→ 2 4 11 17 25 29

Can we solve this by Binary search?

We have to find two number whose sum is 40.

$x + y = 40$

find x and y

x = start

y = end

2	4	11	17	29	29
↑					
x					

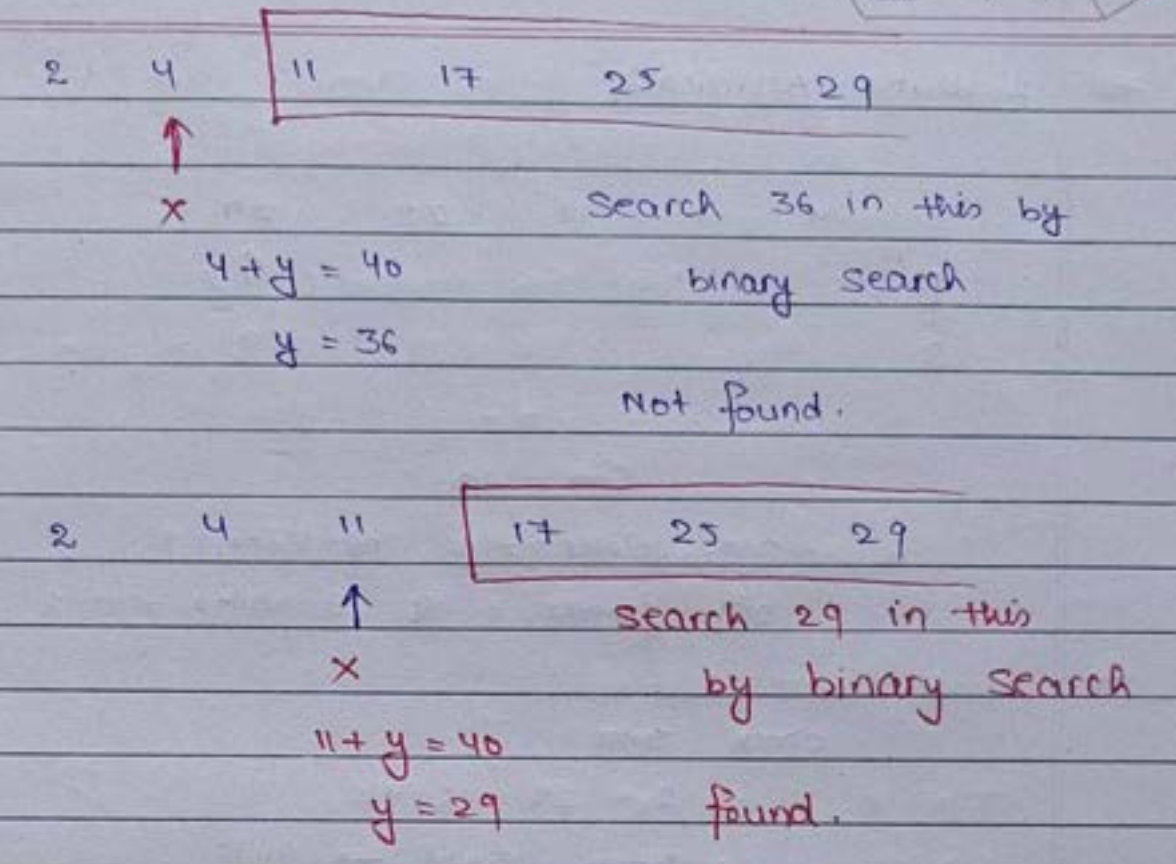
Search 38 in this by Binary search

$2 + y = 40$

$y = 38$

Not found

Teacher's Signature

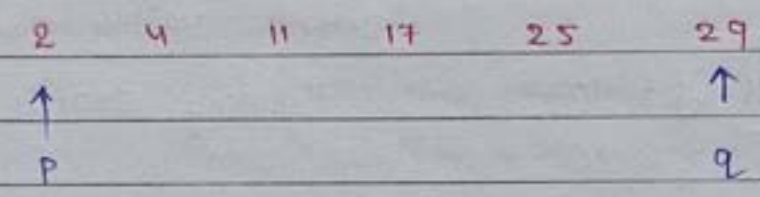


Time Complexity : O(N log N)

```
for (int i=0; i<n-1; i++) {
        find = Sum - A[i];
        start = i+1;
        end = n-1;
        { logic of binary search; }
}
```

Teacher's Signature.....

→ 2 pointer Approach :



Sum = 40

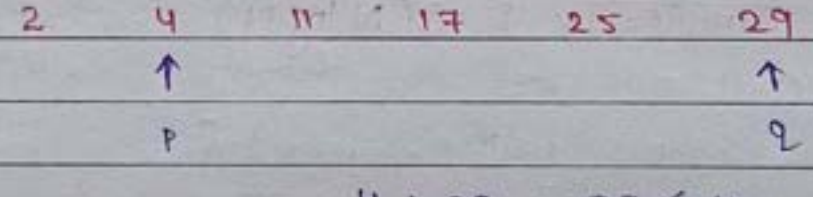
one pointer = p at index = 1

2nd pointer = q at index = n-1

check sum

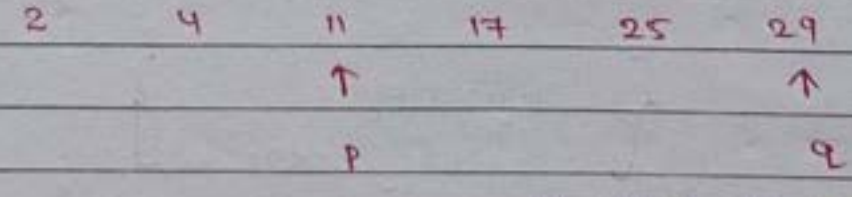
2 + 29 < 40

Move right → (p)



4 + 29 = 33 < 40

Move right →



11 + 29 = 40 == 40

||

sum

return true;

Teacher's Signature.....

→ Kab tak chalega
↳ (p < q)

→ Jab value big chaye → Move right (p)

→ Jab value small chaye → Move left (q)

Code:

```
left = 0, right = n-1;
int sum = 0;
while (left < right) {
    if ( arr[left] + arr[right] == tsum) {
        return true;
    }
    else if ( arr[left] + arr[right] > sum){
        right--;
    }
    else {
        left++;
    }
}
return false;
}
```

Teacher's Signature

* Array : 3 7 13 41 49 51

sorted

Find two number :

x & y → given

x * y = product

product = 91

→ Brute Force Approach :

3 x 7 = 21 x

3 x 13 = 39 x

3 x 41 = 123 x

3 x 49 = 147 x

3 x 51 = 153 x

7 x 13 = 91 ✓ (True)

→ Two pointer Approach :

3 7 13 41 49 51

↑ ← ↑

3 x 51 = 153 > 99

3 7 13 41 49 51

↑ ← ↑

3 x 49 = 147 > 99

Teacher's Signature

3 7 13 41 49 51

↑ ← ↑

3 × 41 = 123

3 7 13 41 49 51

↑ → ↑

3 × 13 = 39 < 91

3 7 13 41 49 51

↑ ↑

7 × 13 = 91 (True)

```
left = 0, right = n-1;
    int sum = 0;
While (left < right) {
        if ( arr[left] * arr[right] == sumgiven){
                    return true,
          }
        else if ( arr[left] * arr[right] > sumgiven){
                    right--;
          }
        else {
                left++;
          }
}
```

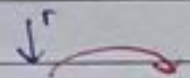
Teacher's Signature.....

* Subarray With given Sum :

Two pointer Approach :

2 4 7 11 14 18

Sum = 25



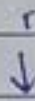
→ 2 4 7 11 14 18

l

SUM = 25

Count_sum = arr[0]

2 != 25 2 < 25



→ 2 4 7 11 14 18

l

Count_sum = 2 + arr[r] = 6

6 != 25 6 < 25

→ 2 4 7 11 14 18

l

Count_sum = 13

13 != 25 13 < 25

→ 2 4 7 11 14 18

l

Count_sum = 24

24 != 25 24 < 25

→ 2 4 7 11 14 18

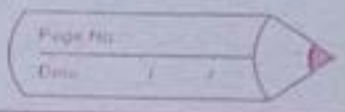
l

Count_sum = 38

38 != 25 38 > 25

Move l.

Teacher's Signature



→ 2 4 7 11 14 18

Count_sum = 36

36 != 25 36 > 25

→ 2 4 7 11 14 18

Cout_sum = 32

32 != 25 32 > 25

→ 2 4 7 11 14 18

Cout_sum = 25

25 == 25

found

Code :

```
Arraylist<Integer> ans = new Arraylist<>();
        if ( s == 0) {
            ans.add(-1);
            return ans;
        }
       int  l = 0;
       int  r = 0;
       int  Countsum = arr[0];
       boolean  isTrue  = false;
         while ( r < n) {
            if ( Countsum == s) {
                isTrue = true;
                break;
            } else if ( Countsum < s){
                r++;
```

```
        if (r < n)
            countSum += arr[r];
        }
        else {
            countSum -= arr[l];
            l++;
        }
    }
    if (isTrue = true) {
        ans.add (l+1);
        ans.add (r+1);
        return ans;
    }
    ans.add (-1);
    return ans;
}
```

* Union of two sorted array:

Arr1[] = { 1, 1, 2, 3 } size = n

Arr2[] = { 2, 3, 3, 4, 5 } size = m

Ans[] = { 1, 2, 3, 4, 5 }

Step 1: Remove all the duplicates element from both the array.

Arr1[] = { 1, 2, 3 }

Arr2[]: { 2, 3, 4, 5 }

Teacher's Signature.....

Arr1 = {1, 2, 3} Arr2[] = {2, 3, 4, 5}

1 < 2

ans[] = {1}

Arr1[] = {1, 2, 3} Arr2[] = {2, 3, 4, 5}

2 == 2

ans[] = {1, 2}

Arr1[] = {1, 2, 3} Arr2[] = {2, 3, 4, 5}

3 == 3

ans[] = {1, 2, 3}

Arr1[] = {1, 2, 3} Arr2[] = {2, 3, 4, 5}

ans = [] = {1, 2, 3, 4, 5}

Code :

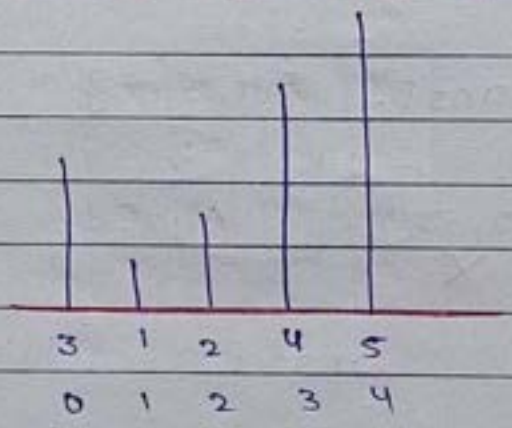
```
While (i<n && j<m) {
    if [arr1[i] == arr2[j]) {
        ans[k] = arr1[i];
        k++;
        i++;
        j++;
    }
    else if ( arr1[i] > arr2[j]) {
        ans[k] = arr2[j];
        k++;
        j++;
    }
```

Teacher's Signature.....

```
        else {
            ans[k] = arr1[i];
            k++;
            i++;
        }
    }
```

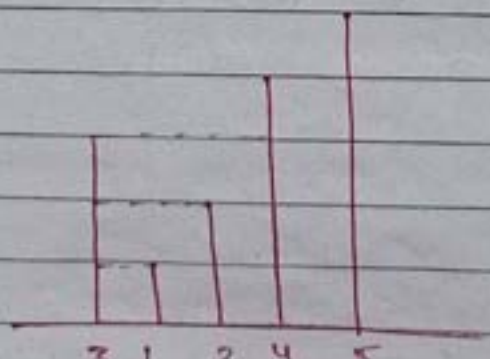
* Container With Most Water :

Container : { 3, 1, 2, 4, 5 }



Brute force :

Check for all possible and find maximum.



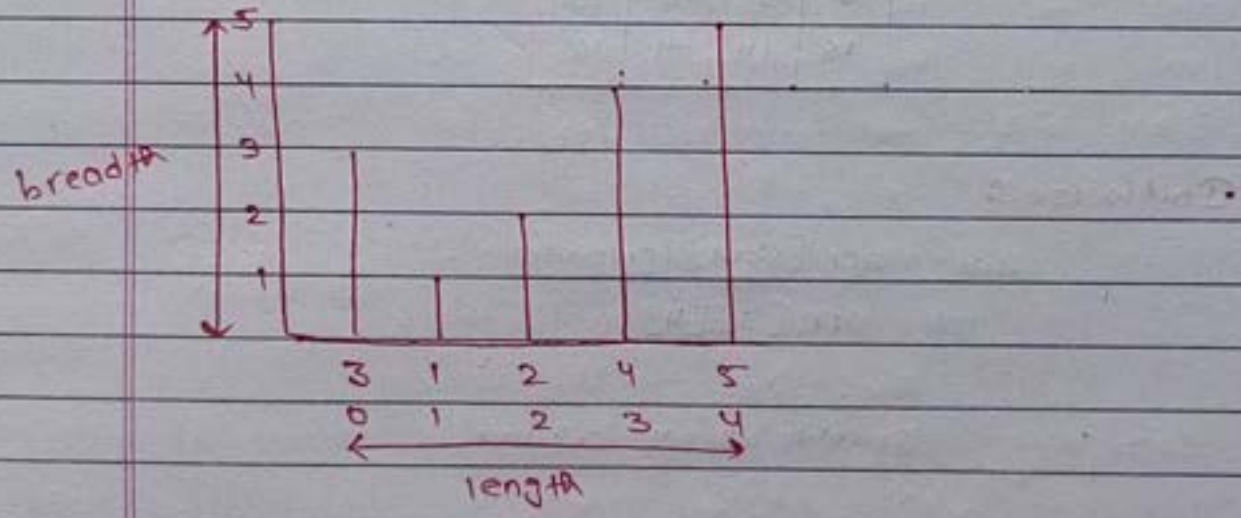
MaxWater = 0

(3,1) (1,2) (2,4) (4,5)
(1,4) (2,5)
(3,2) (1,5)
(3,4)
(3,5)

```
for(int i = 0;   i < n-1;   i++) {
    for (int j= i+1;   j<n;   j++) {

        int bd  = Math.min ( Arr[i], Arr[j]);
        int length  = j - i;
        int water  = bd * length;
        MaxWater =  Math.max ( water, MaxWater);
    }
}
return 0;
}
```

$O(N^2)$

Optimise:



Answer = lengnt * breadth;

Both side → one pointer

i at 0

J at 4

Find Min-m from both & calculate MaxWater.
Where we find min-m increase that pointer one step.

Teacher's Signature.....

Lecture - 17

2D - Array

* Two Dimensional Array

row ↓

Name : [Rohit, paridhi, Riya, Jack, John]
M_NO : [92364, 9432, 9192, 9061, 9054]
R_NO : [32, 64, 66, 61, 39]

→ Column

1	3	5	8
2	4	6	7
9	12	13	15
16	14	14	16

2D Matrix

Initialise :

```
int arr[row][column]
int arr[3][4]
```

3 row & 4 column.

Array : A[3][4]

	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23

3x4

Teacher's Signature

How it Stored in Memory?

Row Major

0	1	2	3	4	5	6	7	8	9	10	11
00	01	02	03	10	11	12	13	20	21	22	23

How to find Which Element are present in Which row major?

Index = (row_index) * column + column_index;

Ex: for 20

2 x 4 + 0 = 8

At 8th index 20 is present.

Index of Row Major is given:

Find Row index & Column index.

Row_index : Index / column

Column_index : Index % column.

* Col-Major order:

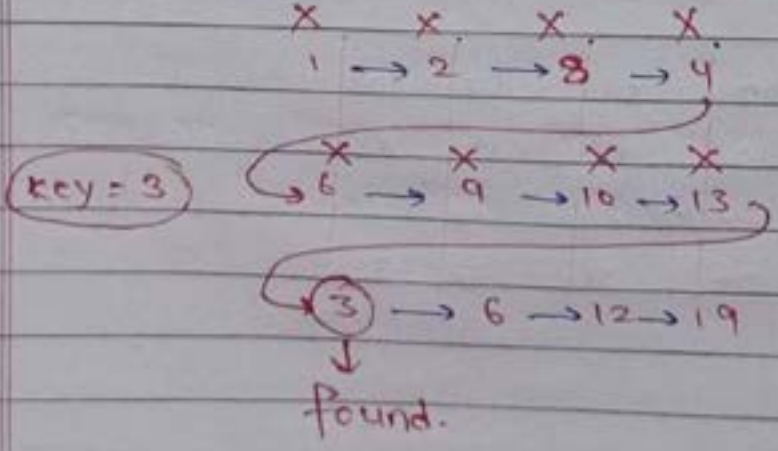
00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

Teacher's Signature

```
int main(){
    int arr[3][4];
for(int i=0; i<3; i++){
    for(int j=0; j<4; j++){
        cin >> arr[i][j];
    }
}
for(int i=0; i<3; i++){
    for(int j=0; j<4; j++){
        Cout << arr[i][j];
    }
    Cout <<endl;
}
```

3×4	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23

* Searching an element:



Code:

```
int arr[3][4];
for(int i=0; i<3; i++){
    for(int j=0; j<4; j++){
        cin>> arr[i][j];
    }
}
int key =19;
for(int i=0; i<3; i++){
    for(int j=0; j<4; j++){
        if(arr[i][j] == key){
            return true;
        }
    }
}
return false;
```

O(MXN)

* Sum of all element of Array:

1 + 2 + 3 + 4 = 10

1	2	3	4	
5	6	7	8	10+5+6+7+8 = 36
9	10	11	10	36+9+10+11+10 = 76

Code:

```
int arr[3][4];
for(int i=0; i<3; i++){
    for(int j=0; j<4; j++){
        cin >> arr[i][j];
    }
}
```

Teacher's Signature.....

```
int sum = 0;
for (int i=0; i<3; i++){
    for (int j=0; j<4; j++){
        sum += arr[i][j];
    }
}
```

* Transpose of a 2D Matrix :

1	2	3	Transpose	1	4	7
4	5	6	⟷	2	5	8
7	8	9		3	6	9

00	01	02		00	10	20
10	11	12	⟶	01	11	21
20	21	22		02	12	22

reverse the index

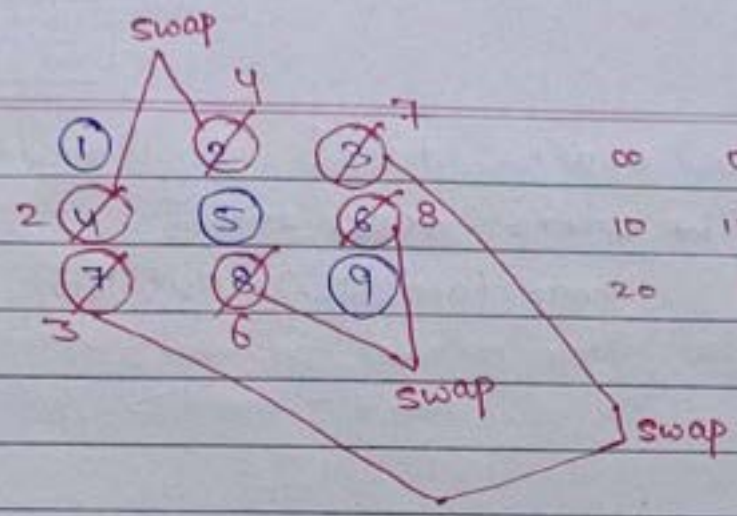
i to j & j to i

Ans[i][j] = A[j][i]

- Transpose is only possible if size is equal.

* Can we transpose the array without creating a new array?

Teacher's Signature



00	01	02
10	11	12
20	21	22

1	4	7
2	5	8
3	6	9

Exchange value

[i][j] With [j][i]

01 With 10

Change only Half Matrix

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

→ Change Made only one this

Diagonal Does not change

if (i == j)

No change

Teacher's Signature.....

```
for(int i=0; i<n-1; i++){
    for(int j=i+1; j<m; j++){
        swap(arr[i][j], arr[j][i]);
    }
}
```

* Flip the array:

```
1  2  3
4  5  6
7  8  9
```

0-2 = 2
1-1 = 2

```
7  8  9
4  5  6
1  2  3
```

Code:

```
for(int i=0; i<n/2; i++){
    for(int j=0; j<m; j++){
        swap(arr[i][j], arr[row-1-i][j]);
    }
}
```

Teacher's Signature

* Rotate Image : (90°)

```
1  2  3          7  4  1
4  5  6  --90°-> 8  5  2
7  8  9          9  6  3
```

```
1  2  3          1  4  7
4  5  6  --Transpose-> 2  5  8
7  8  9          3  6  9
```

↓ Flip vertical

```
7  4  1
8  5  2
9  6  3
```

* Search in a 2D Matrix (sorted):

```
1   3   5   7
10  11  16  20      Target = 16
23  30  34  60
```

sorted : then apply binary search

0	1	2	3	4	5	6	7	8	9	10	11
1	3	5	7	10	11	16	20	23	30	34	60

```
start = 0
end = 11
mid = (strat + end) / 2;
row_index = mid / column;
column_index = mid % column;
```

Teacher's Signature

```
if (arr[row_index][col_index] == target) {
        return mid;
    }
else if (arr[row_index][col_index] < target) {
        start = mid + 1;
    }
else {
        end = mid - 1;
    }
```

* 2nd Approach :

1	2	5	7
11	13	18	20
21	27	30	35

key = 18

18 kis row me Rahega

18 > 1 && 18 < 11 false

18 lies in 2nd row

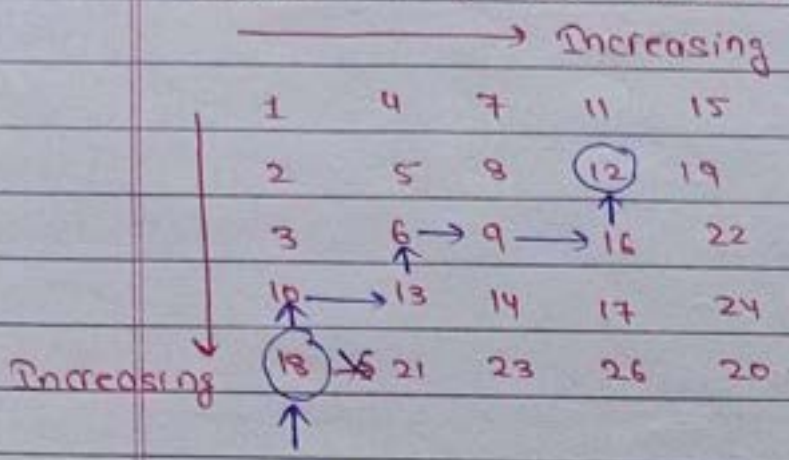
Now Apply binary search in 2nd row :

11 13 (18) 20

↑

key.

* Search in 0 2D Matrix:



From this corner

one pattern is found

If go up ↑ : value decreased

If go right → : value increased

12 < 18 → go up (↑)

10 < 12 → go right (→)

13 > 12 → go up (↑)

6 > < 12 → go right (→)

9 < 12 → go right (→)

16 > 12 → go up (↑)

12 = 12 → found.

Code:

```
int row = matrix.length;
int col = matrix[0].length;
int i = row-1, j = 0;
while (i>=0 && j<col){
    if(matrix[i][j] == target)
        return true;
```

Teacher's Signature

```
else if (matrix[i][j] < target){
                j++;
        }
        else {
                i--;
        }
        return false;
}
```

Teacher's Signature

Page No.
Date : / /

~~Dec~~ Lecture -18

2D - Array and vector

* **Diagonal point :**

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Code :

```
col = 0;
while (col < 4) {
    i = 0, j = col;
    while (j >= 0) {
        cout << arr[i][j];
        i++;
        j--;
    }
```

0 1 4 2 5
8 3 6 9
12

```
row = 1;
while (row < 5) {
    i = row, j = 3;
    while (i <= 4) {
        cout << arr[i][j];
        i++;
        j--;
    }
}
```

7 10 13 11
14 15

Teacher's Signature

* Count zeroes in a sorted Matrix:

N = 3

		0	1	2
A =	0	0	0	0
Small ↑ size	1	0	0	1
	2	0	1	1

→ large, same.

Find last zero of every a row:

0 0 (0) → last

Count = Count + j + 1

= 3

0 (0) 0 1

↓ last

Count = Count + j + 1

= 3 + 1 + 1

= 5

(0) 1 1

↓ last

Count = 5 + 0 + 1

= 6

* Start from top right:

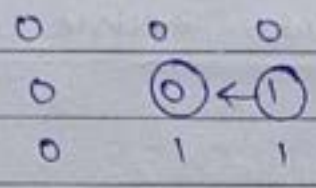
All zero ←

0 0 (0)

0 0 (1) ↓ All 1.

0 1 1

Teacher's Signature



When 1 found
↳ Move left.

When 0 found
↳ Move down.

count = j+1;

Till when:

1) i >= row
2) col >= 0

Code:

```
int count = 0, row = A.size(), col = A[0].size();
    int i = 0; j = col - 1;
While ( i < row && j >= 0) {
    While ( j >= 0 && A[i][j] == 1)
        j--;
    count += j+1;
    i++;
}
return count;
}
};
```

Teacher's Signature

STL (Standard Template Library)

Dynamic Array (vector)

↳ Size in not fixed at initial.

```
vector<int> vector_name;
```

↳ arr

Insert value :

↳ arr.push_back (Element);

Delete value:

↳ arr.pop_back();

↳ pop last element

Size :

↳ arr.size();

For accessing first element:

↳ arr[0] or arr.front();

For accessing last element:

↳ arr[n-1] or arr.back();

For deleting all element of arr / vector:

↳ arr.clear();

Vector size is always : { 2^n

0, 1, 2, 4, 8, 16, 32 ...

Teacher's Signature

It double array size while pushing element.

* Create a vector and insert 10 element:

```
#include <vector>
  vector <int> v;
   for (int i= 1;  i<=10; i++ ){
         v.push_back(i);
    }
         i< v.size();
   for (int  i= 0;  i<10;  i++ ){
         cout << v[i]<<" ";
        }
```

```
Sort (v.begin(), v.end(), greater<int>());        O(NlogN);
cout << v.capacity() << endl;  → (2^n)
cout << v.size() << endl;  → actual size

v.pop_back()  → delete last element
```

→ `vector <int> arr(size);`

→ `vector <int> arr (size, initialise);`

```
vector < int> arr (5, 2);
```

↳ 2 2 2 2 2

↳ In this capacity is 5.

2^n → Not applicable

If we add one element then it double

5 to 10.

Teacher's Signature.....

2D - Array / vector :

```
vector<vector<int>> arr (4, vector <int> (3));
```

↑ array name ↑ rows ↑ column

```
rows   = arr.size();
column = arr[0].size();
```

Total Element = rows x column;

```
vector<vector <int>> arr (3, vector <int>(3));
    for(int i=0;  i<3;  i++)
      for (int j=0;  j<3;  j++)
          cin >> arr[i][j];

//    Sort( arr.begin(),  arr.end());

    for (int i=0;  i<3;  i++)
        sort (arr[i].begin(), arr[i].end());

    for (int i= 0;  i<3;  i++){
        for (int j=0;  j<3;  j++){
            cout << arr[i][j] << " ";
        }
        cout <<endl;
    }
```

```
char arr[10];
  for(int i=0;  i<10;  i++){
        cin >> arr[i];
  }

cin >> arr;
```

Teacher's Signature.....

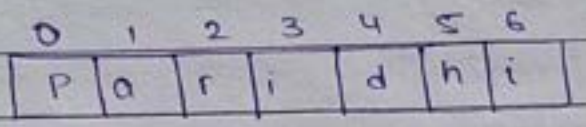
Lecture - 19

String

" paridhi "

String String_name;

cin >> String_name;



String_name[2] = 'r'

S = " Rohit"

M = "Mohit"

① T = S + M

output : RohitMohit

② T = S.append(M)

output = RohitMohit

S = "10";

T = '20';

S = S + T

ouput : 1020

String_name.size();

↓

output = 7

Teacher's Signature

S = " CoderArmy"

① S.pushback('s'); → Add s at last

S = CoderArmys

② S = S + 's';
S.pop_back();
↳ delete last letter

Sort:
sort(str.begin(), str.end())
str before sort : abchb
str after sort : abbch.

→ Code:

```
String str;
cin >> str;          // Input : paridhi
Cout >> str;         // output : paridhi
cout << str.size();  // output : 7
cout << str[1];      // output : a
```

→ String str1 = " paridhi singh"

```
Cout << str1;          // paridhi singh
Cout << str1.size();   // 13
```

→ String str2;

```
cin >> str2;    // paridhi singh
Cout << str2;   // paridhi
```

In input if we give space, tab then it can be terminate.

Teacher's Signature.....

If you want to consider space also then use getline :

```
String str;
getline (cin, str);      // Input : Paridhi singh
Cout << str;             output : paridhi singh
```

```
String str1, str2;
getline (cin, str1);     // Input : paridhi singh
getline (cin, str2);     // Input : Riya Rani
Cout << str1 << " " <<str2;  output: paridhi singh Riya Rani
```

→ string str : "Ram is "a" boy";
↳ Given Error

→ string str : " Ram is \"a"\ boy";
↳ output : Ram is "a" boy.
\ → ignore next character and print as written

→ string str : " Ram is \\a\\ boy";
↳ output : Ram is \a\ boy.

→ Char c = 'd' + 2;
Cout << c; // output : f

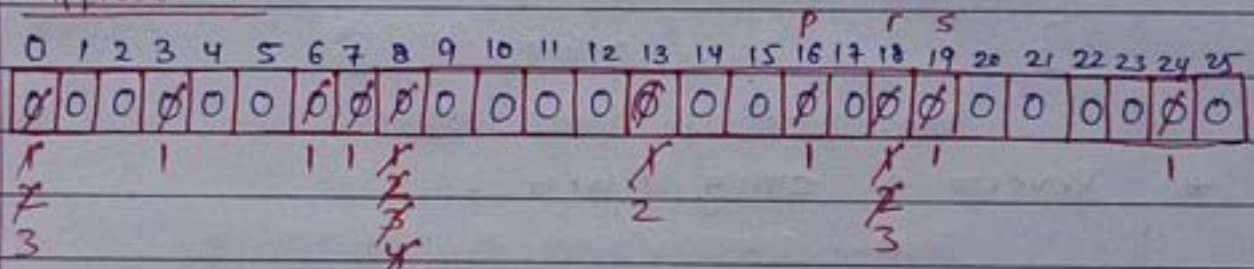
Teacher's Signature.....

* Sort by Comparing ASCII value:

```
for (int i=0;  i<n-1;  i++){
        for (int j=1;  j<n;  j++){
                if ( s[j] < s[j-1]){
                        swap(s[j], s[j-1]);
                }
        }
}
```

$O(N^2)$

* Approach - 2



s = pandhusinghRiyaRoni

↳ Assum all are small

p a n i d h i s i n g h R i y a R o n i

a a a d g h i i i i i n n p r r r s y

↳ sorted.

Code:

```
String s;
cin >> s;
n = s.size();
int count[26];
for (int i=0;  i<26;  i++)
        count[i] = 0;
```

Teacher's Signature.....

```
for (int i=0; i<n; i++){
        int index = s[i] - 'a';
        Count[index]++;
    }
    for (int i=0; i<26; i++){
        for (int j=0; j<Count[i]; j++){
            char c = 'a' + i;
            cout<<c;
        }
    }
    return 0;
}
```

* Reverse a string (GFG):

```
p  a  r  i  d  h  i
↑                 ↑
start             end
```

```
start < end
swap (p, i)
start++;
end--;
```

```
i  a  r  i  d  h  p
↑              ↑
start          end
```

```
start < end
swap (a, p)
start++;   end--;
```

Teacher's Signature

i h r i d a p

↑ ↑

start end

start < end

swap (r, d)

start++;

end--;

i h d i r a p

↑

start

end

start < end (false)

Code :

```
int start = 0;
int end = str.size() - 1;

    while (start < end) {
        swap (str[start], str[end]);
        start++;
        end--;
    }
return str;
}
```

* Anagram of string :

Example: S1 = cdagk
S2 = gcd

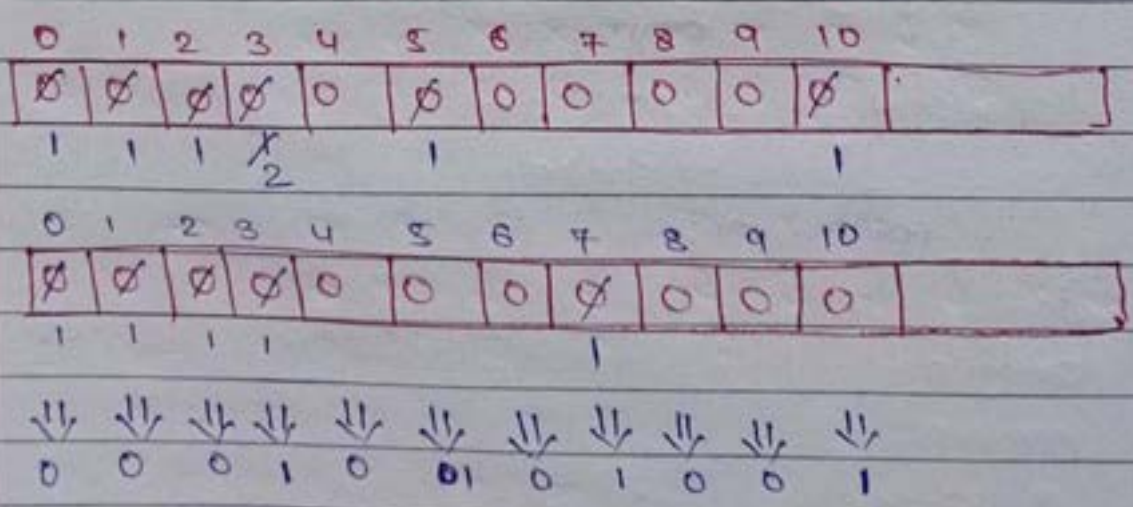
output = 2

from S1 = one d is deleted
one k is deleted
total 2 operation

Example:

S1 = abcddkf
S2 = abcdh

from S1 : one d is deleted
one k is deleted
one f is deleted
S2 : one h is deleted
→ 4



SUM = 4

Teacher's Signature

* palindrome string:

Example = a b b a return 1
True

a b c return 0.
False

```
int  start = 0, end = s.size()-1;
     while (start <= end) {
          if (s[start] != s[end]) {
               return 0;
          start++;
          end--;
          }
     }
     return 1;
}
```

2nd Approach:

→ Reverse string and store in another string.
→ Match each character.

```
string k; k = s;
reverse (k.begin(), k.end());
     if (k == s)
          return 1;
     else
          return 0;
```

Teacher's Signature.....

* Minimum Number of Flips :

S = "001"

Alternate me alg alg character hona chaiye.

0 ke bagal me 1 होना
1 ke bagal me 0 hoga.

Hum Ek- Ek character ko idhar udhar kar sakte hai to humko kitna flip flop kar sakte hai.

Minimum flip return karna hai.

→ 0001
↑
1 0101 1 flip
return 1.

→ 0 0 0 1 0 1 0 1 1 1
↓ ↓ x x x x x x ↓ x
8 flips 1 0 1 0 1 0 1 0 1 0

→ 0 0 0 1 0 1 0 1 1 1
↓ x ↓ ↓ ↓ ↓ ↓ ↓ x ↓
2 flips 0 1 0 1 0 1 0 1 0 1

return 2

Teacher's Signature.....

* Merge two string :

S1 : Hello

S2 : Bye

Ouptput : HBeyleloD

```
string s3;
While (i < s1.size() && j < s2.size()){
        s3 = s3 + s1[i];
        s3 = s3 + s2[j];
        i++;
        j++;
}
While (i < s1.size()) {
        s3 = s3 + s1[i];
        i++;
}
While (j < s2.size()) {
        s3 = s3 + s2[j];
        j++;
}
```

* To lower case:

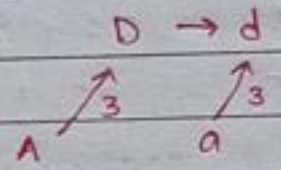
S1 = "LOVELY"

Output: lovely

S2 = "Here"

output: here

→ Code:




```
char c = s[i] - 'A'
    c = 'd' + s[i] - 'A';
if (s[i] >= 'A' && s[i] <= 'z')
```

* Sum of two large Number:

X = "25"

Y = "23"

Output = 48

X = "2500"

Y = "27"

output = 2527

Teacher's Signature

int num = '7' - '0' = 7

int num = '0' - '0' = 0

carr = 0, num = 0

↳ sum Niklega.

Code:

```
int i = x.size() - 1;
int j = y.size() - 1;

String str = "";
int num, rem, carry = 0;
Char c;

While (i >= 0 && j >= 0) {
    int num1 = x[i] - '0';
    int num2 = y[i] - '0';
    num = num1 + num2 + carray;
    carry = num / 10;
    rem = num % 10;
    c = '0' + rem;
    str = str + c;
    j--;
    i--;
}
While (i >= 0) {
    int num1 = x[i] - '0';
    num = num1 + carry;
    carry = num / 10;
    rem = num % 10;
    c = '0' + rem;
    str = str + c;
```

Teacher's Signature.....

```
        i--;
    }
    While ( j>=0) {
        int num1 = y[j] - '0';
        num = num1 + carry;
        carry = num/10;
        rem = num %10;
        c = '0' + rem;
        str = str + c;
        j--;
    }
    if(carry == 1)
        str = str + '1';

    int size = str.size()-1;
    while ( size > 0 && str[size] == '0'){
        str.pop_back(); size;
    }
    reverse( str.begin(), str.end());
    return str;
}
};
```

Lecture 20

String : Hard problem

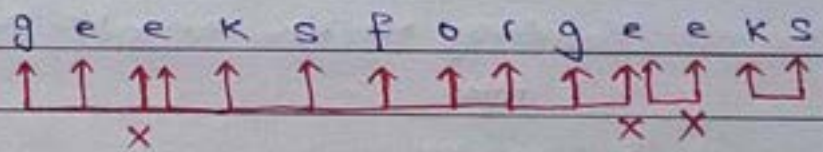
* Length of the longest substring

Given a string,

Str = geeksforgeeks

Task : Find largest substring which have all elements are distinct.

geeksgeeks



length = 7

ge, eksforg, eks

↓

Max = 7

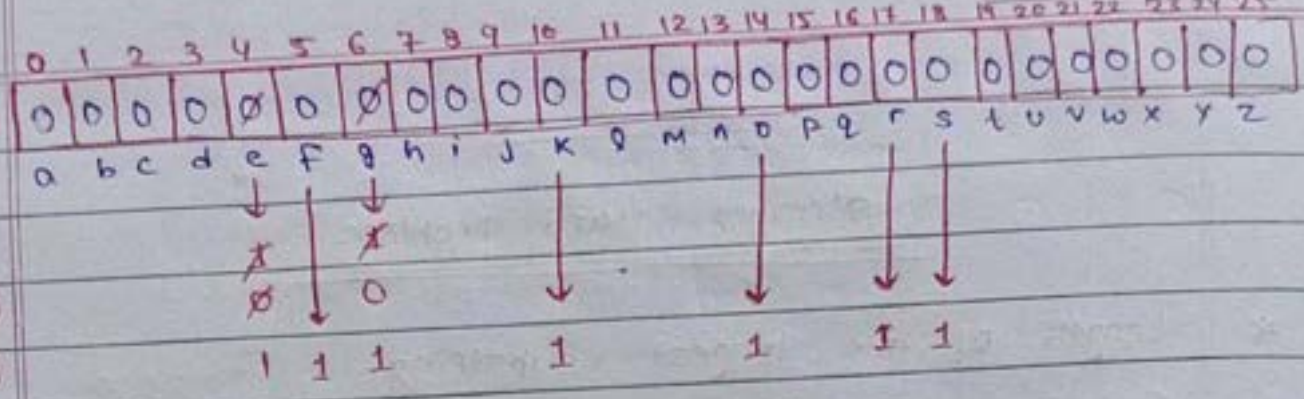
geeksforgeeks

i j

Take 2 pointer

First at index 0 → i

Second at index 1 → j



g e e k s f o r g e e k s

↑ ↑ ↑ ↑

i j j

x

count = 2

Increase j

↳ Jab tak koi element dubara nahi mil jata. or

Array Me kisi index Me 1 Nhi mil jata.

Agar Koi element dubara Mil Jata hai to i ko Aage badhao Jabtak Wo repeated element ko Ek baar cross Na kar Jaye.

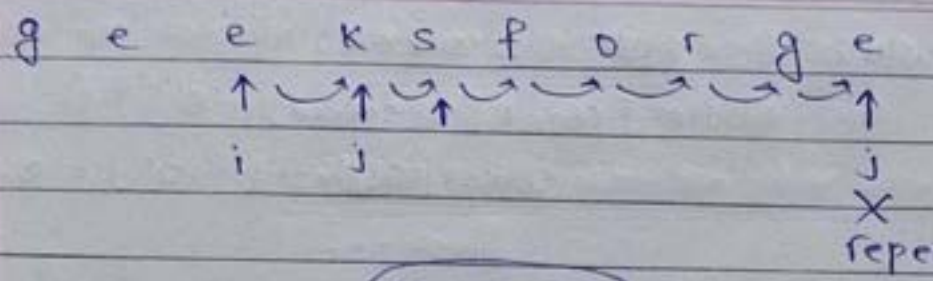
Array Me zero karte Jao.

g e e k s f o r g e e k s

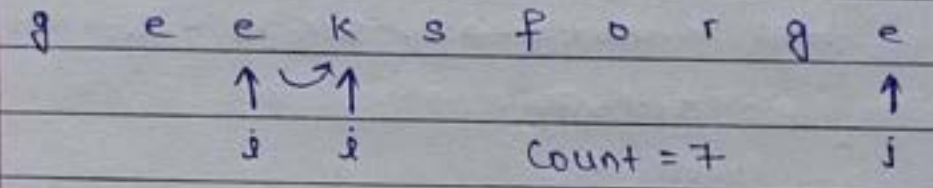
↑ ↑

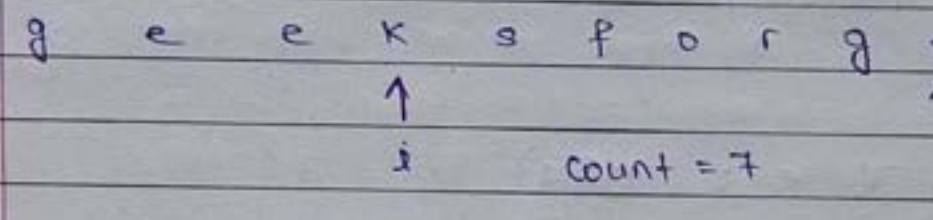
i j

Again increase j and same process ko repeat karo.



Count = 7





Max = 7

This is continue until
j reach the last element

While (j < str.size())

Code :

```
int count[26];
for (int i = 0; i < 26; i++){
        count[i] = 0;
    }
int first = 0;
int second = 1;
count[s[0] - 'a'] = 1;
int total = 1;
```

Teacher's Signature

```
While (second < s.size()){
    While (count[s[second] - 'a']){
        count[s[first] - 'a'] = 0;
        first++;
    }
    count[s[second] - 'a'] = 1;
    total = max(total, second - first + 1);
    second++;
}
return total;
}
```

T. C : O(N)

* Longest Common prefix in an Array :

N = 4

arr[] = {geeksforgeeks, geeks, geek, geezer}

In this problem, we have to return substring for first at which the substring are same for all the string.

arr[0] = g e e k s f o r g e e k s
arr[1] = g e e k s
arr[2] = g e e k
arr[3] = g e e z e r

arr[0][2] = e

The largest possible prefix is the string which size is minimum.

So, we found first the string which size is minimum.

```
int mn = INT_MAX;
for (int i = 0; i < N; i++) {
        if (mn > arr[i].size()){
                mn = arr[i].size();
        }
}
```

By this we can find the size of mn-m string and we check the prefix for only that length.

Now we check all the string and compare for index 0 to mn.

mn = 4

arr[0] = geeksforgeeks
arr[1] = geeks
arr[2] = geek
arr[3] = geezer

↓

We can check in this only because the size of smaller string is 4.

Teacher's Signature.....

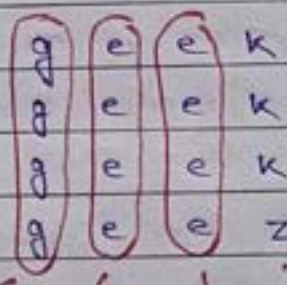
Now, we check for index 0 that all element are same.

If the element in same then we do nothing only we increase the count by 1.

If the element is not same in all string then:

If count is greater than 0 then we return substring from 0 to count - 1.

If count is not greater than 0 then return '-1'.



Not same
↳ then check for count

same
then
Count++
Count = 1

Same
Count++
Count = 2

same
Count++
Count = 3

Is Count > 0
↓
Yes
return
substr(0, count).

Output: gee

Teacher's Signature

Code :

```
int Count = 0;
int mn = INT_MAX;
for (int i = 0; i < N; i++) {
        if (mn > arr[i].size()) {
            mn = arr[i].size();
        }
}
for (int i = 0; i < mn; i++) {
    for (int j = 1; j < N; j++) {
        if (arr[j-1][i] != arr[j][i]) {
            if (Count > 0)
                return arr[0].substr(0, Count);
            else
                return "-1";
        }
    }
    Count++;
}
if (Count > 0)
    return arr[0].substr(0, Count);
else
    return "-1";
```

* Longest prefix suffix :

↳ Not be equal to entire string

1) S = "abab"

output = 2

ab → suffix and prefix

2) S = "aaaa"

output = 3

aaa → suffix and prefix

We have return a string which is longest prefix and suffix both for string.

a b c d e

prefix : a, ab, abc, abcd

suffix : e, de, cde, bcde

Approach 1 :

a a a c h a a a

↑ ↑

same, size = 1

a a a c h a a a

↑ ↑

same, size = 2

Teacher's Signature

* Approach 2:

a a a c h a a a

↑ ↑

i j

→ a b c a b

↑ ↑ j++;

i j

Not equal

a b c a b

↑ ↑ j++;

i j Not equal.

a b c a b

↑ ↑ i++, j++

i j equal Size = 1

a b c a b

↑ ↑ equal

i j size = 2.

We also check j must be the last index then we return the size.

If j is not last then, i become to first.

Teacher's Signature.....

We can check j from every letter which is equal to the first character of the string.

Note:

We can start j only from those index where element present at j index is equal to first character of string.

→ i is always start from first index.

a b c a b d e a b c a b c a b d

Size = 6

return 6

Teacher's Signature

Lecture 21

* Longest prefix suffix :

Better Approach : O(N)

str = a b c d a b c e a b c d a b c d a b

a b c d a b c e a b c d a b c d a b

First start index from 1 for the string.

And we need two pointer :

1st : at 0th index

2nd : at 2nd index.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
a	b	c	d	a	b	c	e	a	b	c	d	a	b	c	d	a	b	

On this we move 2nd pointer only front side
↳ dont move back side.

First pointer Move both side.

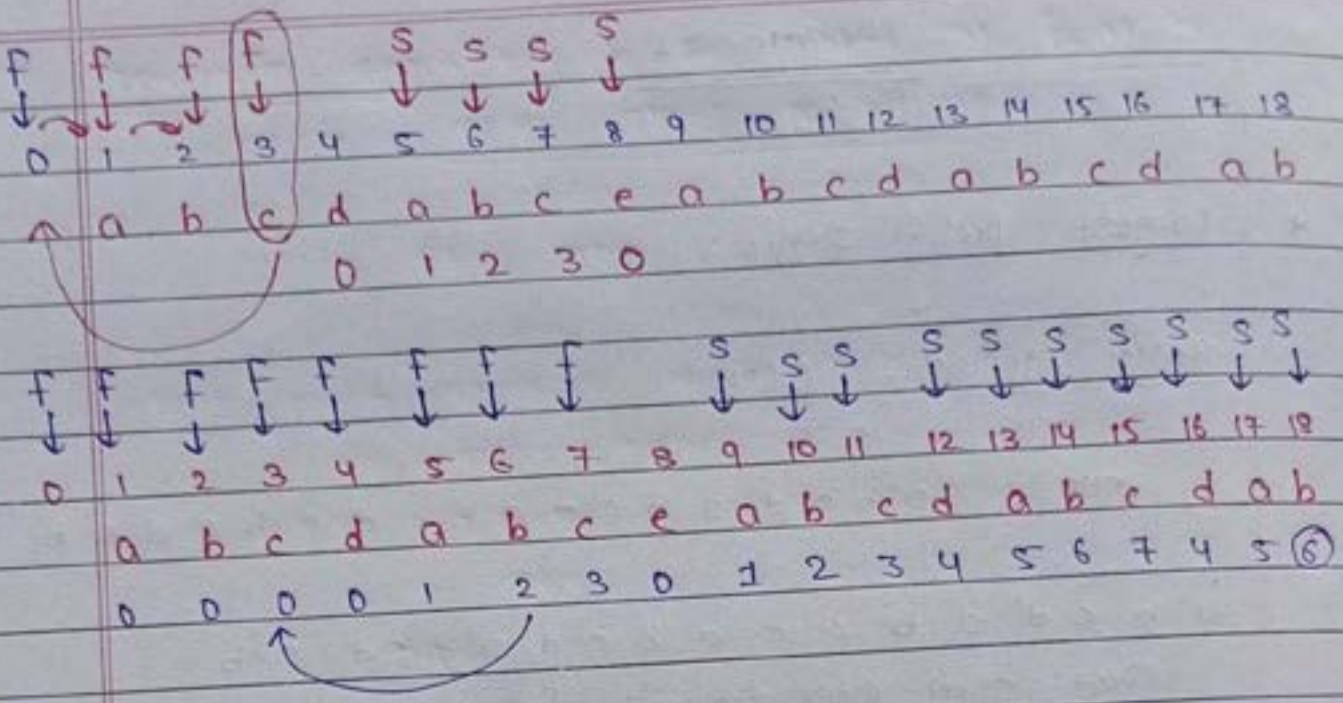
first ↓ second ↓

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
a	b	c	d	a	b	c	e	a	b	c	d	a	b	c	d	a	b	
Lps	0	0	0	0	1													

Now, Both First
& second increase

First + 1 = second
↳ False (print 0).

Teacher's Signature




```
int LPS[s.size() + 1];
char str[s.size() + 1];

for (int i=0; i<s.size(); i++) {
        str[i+1] = s[i];
        LPS[i] = 0;
}

LPS[s.size()] = 0
int first = 0, second = 2;

While ( second <= s.size()) {
    if ( str[first + 1] = str[second]){
        LPS[second] = first + 1;
        first++; second++;
    }
elsef{
    if ( first == 0)
        second++;
else
```

Teacher's Signature

```
            first = lps[first];
        }
    }
    return lps[s.size()];
}
```

* **Check if string is rotated by two places:**

Str1 : amazon

Str2 : anamaz

tell that How many time string 1 rotate to make str2.

str1 & str2.

	a	m	a	z	o	n	&	a	n	a	m	a	z
lps :	0	0	1	0	0	0		0	0	0	1	2	3	4

lps = 4

6 - 4 = 2

↳ return.

Teacher's Signature.....

* Minimum character to be added at front to make string palindrome:

str : abc

How many element to be added at front to make palindrome.

abc

add

cb

cbabc

↳ 2 element.

Example 2:

str1 : tictk

str2 : reverse of str1 : ktcit

str1 $ str2

t i c t k $ k t c i t

LPS: 0 0 0 1 0 0 0 1 0 0 1

5 - 1 = 4

Teacher's Signature

Example:

Str1 = a a c e c a a a

Str2 = a a a c e c a a

↙

Reverse of Str1

Str1 & Str2

a	a	c	e	c	a	a	a	$	a	a	a	c	e	c	a	a
0	1	0	0	0	1	2	2	0	1	2	2	3	4	5	6	7

8 - 7 = (1)

Code:

```
int index [2 * (S.size() + 1)];
char str [2 * (s.size() + 1)];
int n = 2 * (S.size() + 1);

for (int i = 0; i < size(); i++) {
    str[i+1] = str[n-i-1] = s[i];
    index [i] = index [i + s.size()] = 0;

    index [n-1] = 0;
    Str [s size() + 1] = '$';
    int first = 0, second = 2;
  While ( second < n ){
    if ( str[first + 1] == str [second] ){
        index[second] = first + 1;
        first++;
        second++;
    }
```

Teacher's Signature.....

```
else {
    if ( first == 0)
        second++;
    else
        first = index [first];
    }
}
return  s.size() - index [str.size()];
}
```

Teacher's Signature

Lecture 22

Pointer

```
int * p;
```

p is a pointer which point a integer type of value.

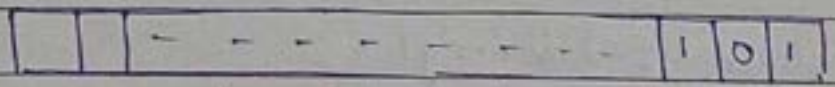
```
int num = 5;
```

4 byte = 32 bit.

Byte addressable : Every byte has assigned with a address.

```
int num = 5  → 101
```

4 byte = 32 bit.



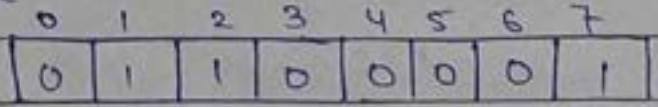
↳ 5 is stored like this in memory.

char c = 'a'; → ASCII → 97 → Binary → 1100001

1 byte : 8 bit

Byte addressable

200



↳ 'a' is stored like this in memory.

It means it have to read 8 bit because it is Byte addressable.

Teacher's Signature

int num = 5

address

p

p = #

p = 200 → address of num

* p = 20 ;

↳ it change the value of variable whose address stored in p.

200

200 | num | 20

p

NoW, num = 20.

*p → ye apne adress pe jisko bhi store karwaya hua hai uske stath change kar sakta hai.

pointer : memory

↓↓

address

4 GB Ram : 2^{32} byte.

pointer : → 32 bit

→ 64 bit

Teacher's Signature

0000 - - - - 0000

0000 - - - 0001

0000 - - - 0010

⋮

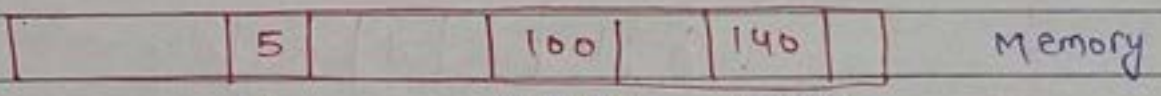
$2^{32} - 1$

0 to $2^{32} - 1$

Symbol table :

Variable	Address	Type

Num = 5 int * p = #



4 byte — 8 byte — 8 byte

↓

iska bhi Kuch address hoga.

(140)

int *p;

↳ p can contain address.

It store garbage value if not any value stored.

We put for clarification

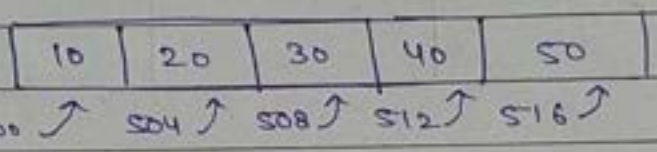
Put → NULL; or 0;

Teacher's Signature

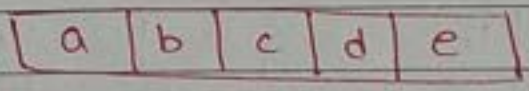
```
int num = 30
int * p;
p = & num;
* p = 20;
cout << num;
```

int arr [5] = { 10, 20, 30, 40, 50 }

20 byte : 4 byte each



Continous



arr[0] = arr + 0
arr[1] = arr + 1
arr[2] = arr + 2

→ int arr [5] = { 10, 20, 30, 40, 50 };
cout << arr ; // print address
↳ print in Hexadecimal Form.

cout << * arr;
↳ print arr[0];

Teacher's Signature

```
cout << *(arr+0);
        ↳ print arr[0];
cout << *(arr+1);
        ↳ print arr[1];

p = arr;
   ↳ it is already a adress

Ex, int num = 5;
    int * p;
    p = & num;
     *p;
     *p(+1);
     *(p+2);

cout << *p;
     ↳ address of arr[0]

→ int arr[5] = {10, 20, 30, 40, 50}
    int * p;
     p = arr;
Error { for (int i = 0; i < arr.size(); i++){
          Cout << arr " ";
           arr++;
        }
```

Teacher's Signature

```
int arr;
p = arr;
{ 10, 20, 30, 40, 50 }
```

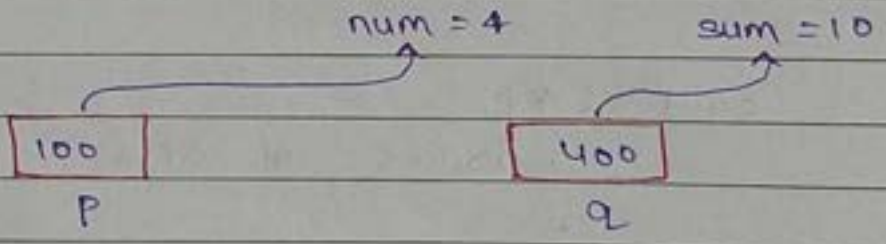
By doing p++

↓

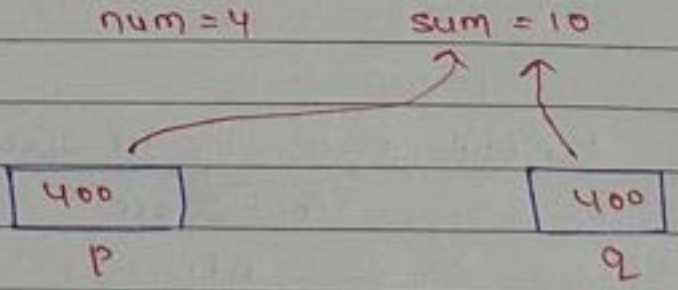
Error because it is increment the adress.

→

```
int *p, *q;
p = &num;
q = &sum;
p = q;
```



After:




```
Void fun (int c, int d) {
    c = c * 2;
    d = d * 2;
}
int main() {
    int a=5, b=8;
```

→ change c & d but in main can't reflect

Teacher's Signature

```
fun(a, b);                    → print 5 and 8
Cout <<a<< " "<<b;
return 0;
}
```

pass by value

pass by address :

```
int main () {
    int a = 5, b = 8;
    fun (&a, &b);
    Cout <<a << " " <<b;
    return 0;
}

Void fun ( int *c, int * d) {
    *c = *c * 2;
    *d = *d * 2;
}
```

* call by Reference :

```
Void fun ( int &c, int &d ) {
        c = c * 2;
        d = d * 2;
}
```

Teacher's Signature

```
* void reverse (int * arr , int size){
        int start = 0, end = size - 1;
        while (start < end){
            int temp = arr [start];
            arr[start] = arr [end];
            arr[end] = temp;
            start ++;
            end - - ;
        }
    }

int main() {
    int a[5] = {10, 20, 30, 40, 50};
        reverse (a, 5);
        for (int i = 0; i < 5; i++){
            cout << a[i] << " ";
        }
        return 0;
    }
```

Teacher's Signature.....

Lecture : 23

Pointer

① Reference variable :

```
int i = 20;
int &j = i;
```

Swap :

```
int main(){
    cin >> a >> b;
    swap(a, b);
    return 0;
}
void swap (int &c, int &d){
    int temp = c;
    c = d;
    d = temp;
}
```

In this we can pass address value. We cannot pass direct value of variable.

→

```
int main(){
    int arr[] = {1, 2, 3, 4, 5};
    double_value(arr, 5);
}
```

Teacher's Signature

```
Void double_value( int *a, size) {
        for (int i=0; i<size; i++){
            a[i] = a[i]* 2;
        }
    }
```

```
Void double_value (int *a, size){
        for(int i=0; i<size; i++){
            *a = *a * 2;
            a++;
        }
    }
```

① & ② Both do same task.

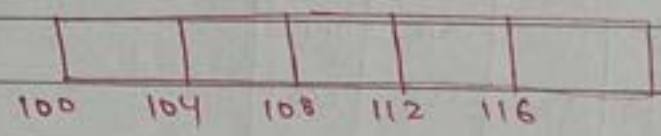
```
int main() {
int arr[5] = {1, 2, 3, 4, 5}
    for(int i=0; i<5; i++){
        *arr = *arr * 2;
        arr++;
    }
}
```

This gives error because we can't change the array.

We can only change the pointer in a copy part.

Teacher's Signature

```
int main() {
    int arr[5] = {1, 2, 3, 4, 5};
        cout << arr;
        cout << &arr;
        cout << &arr[0];
```



&arr;

↳ array kis memory location pe present hai

output = 100

&arr[0]

↳ address of 0th element

output = 100;

arr :

↳ arr ki memory kis index se suru ho raw hai.

All 3 statement print same output.

* 2D Array:

arr[4][4]

	0	1	2	3
0	100	104	108	112
1	116	120	124	128
2	132	136	140	144
3	148	152	156	160

arr : Store address of first row
: 100

arr + 1 : Store address of 2nd row
: 116

& arr :
↳ address of starting index
100

& arr[0]:
↳ address of first row
100

& (*(arr + 0):
↳ address of first row
(*arr) + 1 : 104.

& arr[0] = 100;

** arr = *(*arr) = *100
Address of address of arr

Teacher's Signature

Page No.:
Date: / /

→ address.

= new int;

int *p = new int[10];

↳ It makes an array.

```
int main(){
    int *p = new int;
    *p = 10;
    cout << *p;
    return 0;
}
```

cout << p;

↳ address of value stored in p.

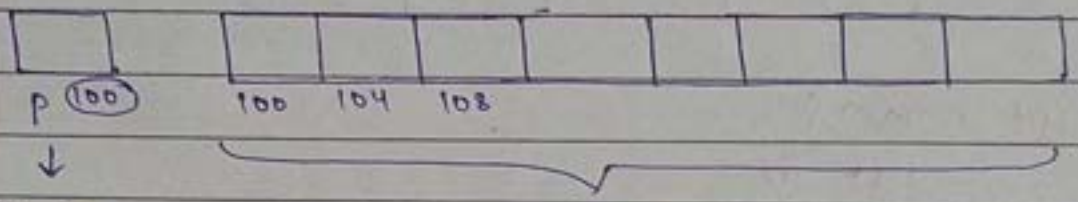
Teacher's Signature.....

Lecture 24

2D - Array and Recursion

One - Dimensional Array :

int * p = new int [10]; → new memory in heap



pointer

Integer Array

* In this Array has No name.

```
Const int col = 5;
Void fun (int arr[][col], ROW) {

}

int main(){
    int m, n;
    cin >> m >> n;
    int arr[m][n];
```

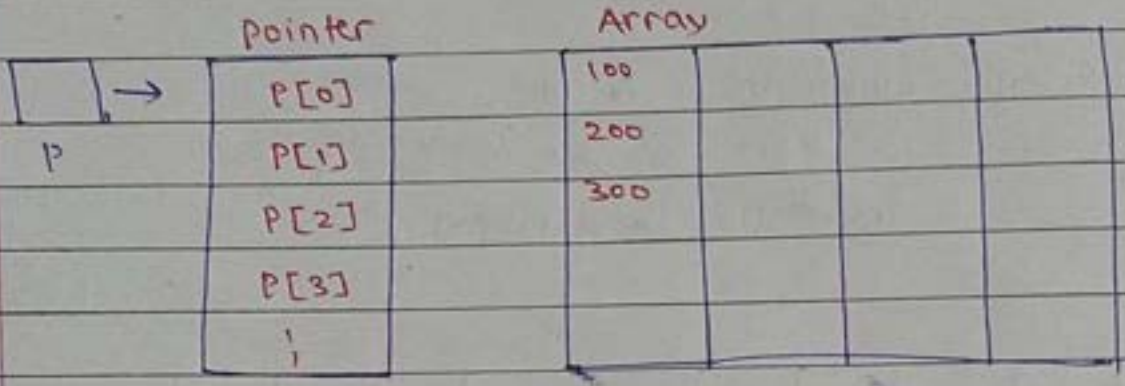
↓ ka chlti hai

Const

Teacher's Signature

2D Array :

```
int **p = new int *[10];
```




```
int main() {
    int m;
    cin>> m;

    int * p = new int[m];
    for (int i = 0; i<m; i++)
        p[i] = i * 2;

    for (int i=0; i<m; i++)
        cout << p[i];

    return 0;
}
```

p is a pointer which point to an array.

Teacher's Signature

```
For (int i=0; i<m; i++) {
        *p = i * 2;          // if array p++ gives
         p++;                   error, so it is
    }                           pointer.
    p--;
for (int i=0; i<m; i++){
        Cout << *p << " ";
         p--;
}
```

→

```
Void fun( int *a, int size) {
      for (int i=0; i<size; i++)
            a[i] = i;
   }
int main (){
        int m;
        cin >> m;
       int *p = new int[m];
          fun(p, m);
      return 0;
   }
```

code for 2D - Array

```
int main () {
    int n, m;
    cin >> n >> m;
    int ** p = new int *[n];
    for (int i=0;  i<n;  i++ )
            p[i] = new int[m];

    fun ( p, n, m);
    return 0;
}

void fun ( int ** x, int row, int col ){

    for (int i=0;  i < row; i++ ){
        for (int j = 0; j<col; j++) {
            x[i][j] = i+j;
        }
    }
]
```

For print :

```
cout << p[i][j] << " ";
```

Recursion

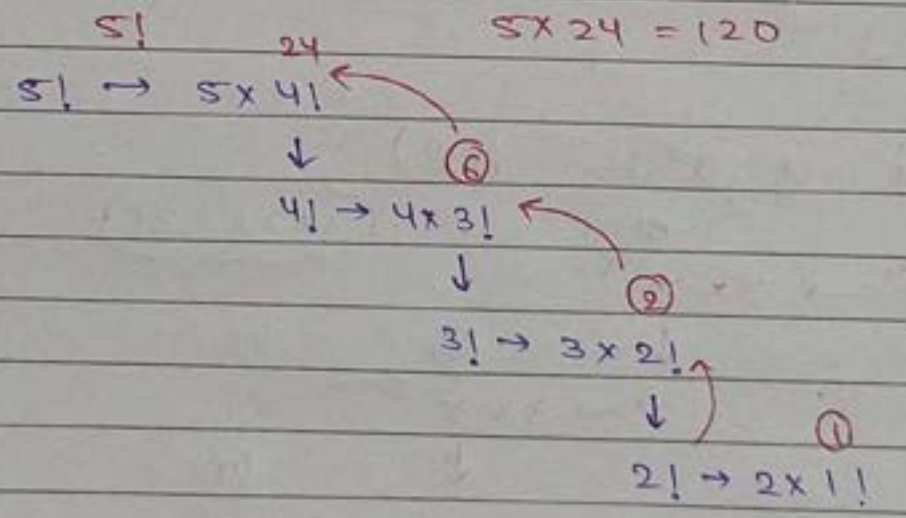
A function which call itself.
After some iteration, it has to break for output.

```
Void fun() {
                    // Infinite loop function.
    fun();
}
```

2 Condition:

1) Breaking statement
2) calling itself

factorial:



Teacher's Signature

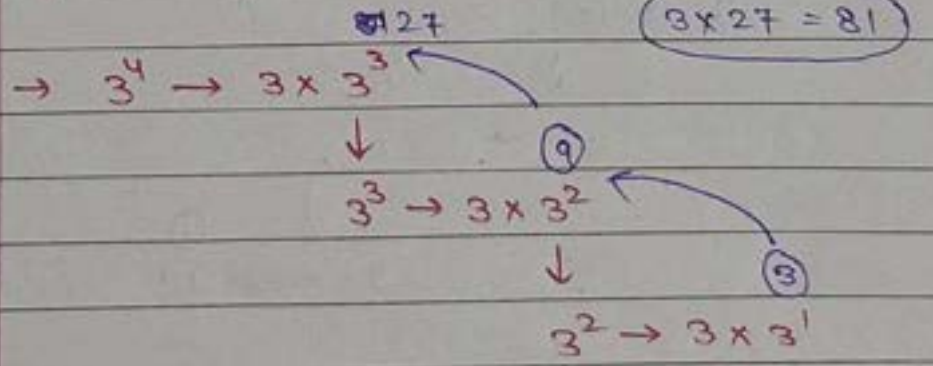
code :

```
factorial (int n) {
    if (n == 1)
        return 1;
    int num = factorial (n-1);
    num = n * num;
        return ans;
}
```

* Recursion :

```
int factorial (int n) {
        if (n == 1)
            return 1;

        return   n * factorial (n-1);
}
```

* power of a^b : (3^n)



Teacher's Signature

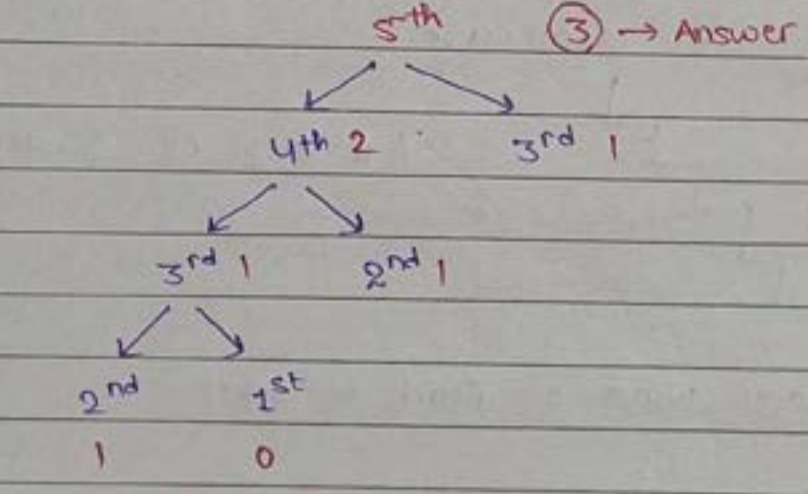
```
power (int n) {
    if (n == 1)
        return 3;
    return  9 * power (n-1);
}
```

* Fibonacci series :

0 1 1 2 3 5 8 13 21 ....

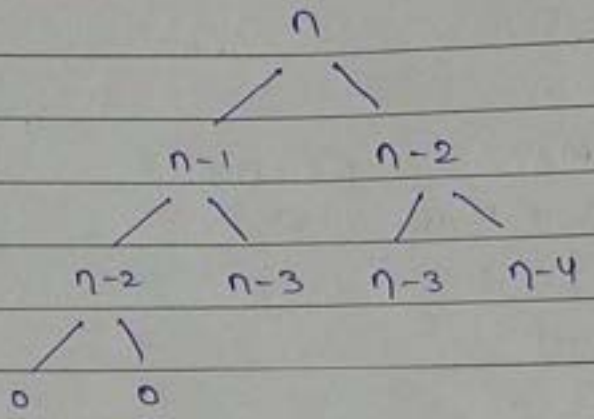
n^{th} fibonacci series

$n = 5$




```
int fib (int n) {
    if (n == 1)
        return 0;
    else if (n == 2)
        return 1;

    return  fib(n-1) + fib(n-2);
}
```

Teacher's Signature



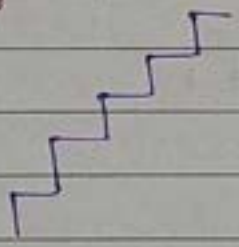
$$1 + 2 + 4 + 8 + \cdots 2^N$$

$$= 2^N$$

$$O(2^N).$$

* Climbing stairs:

Climbing a staircase



(1 step or 2 step)

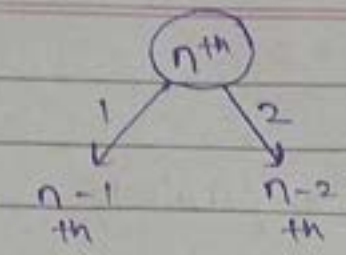
how many ways to reach on top?

$n = 3$

1+1+1
1+2
2+1

} 3 ways

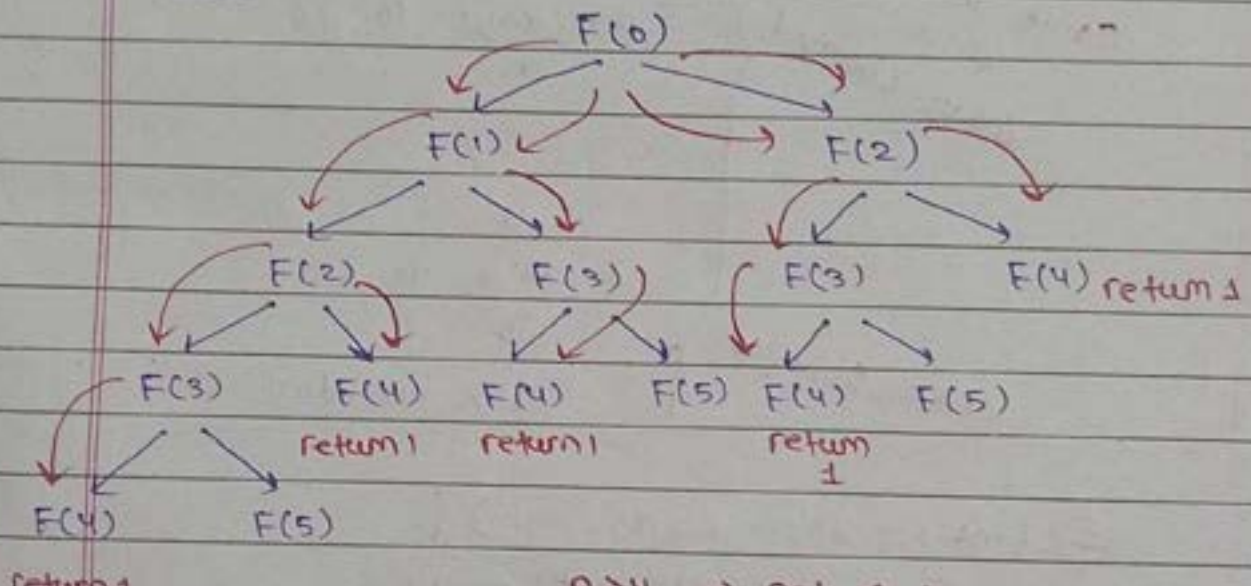
Teacher's Signature



$f(1) = 1$

$f(2) = 2$

N=4



$n > 4 \rightarrow$ return 0

5 possible ways.

Teacher's Signature.....

Lecture - 26

Height of some students are given:

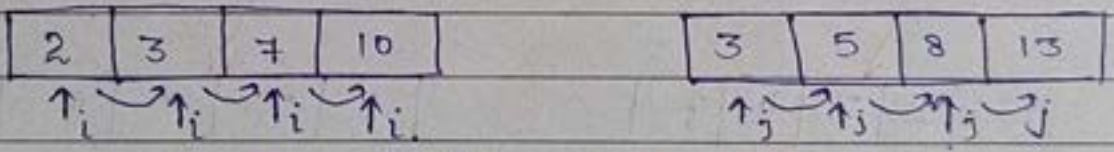
4 3 7 1 2 8 9

We have to arrang the student in ascending order by sorting.

previously, we learn Bubble sort, insertion sort, Selection sort.

Now, we can sort this by a new sorting method called Merge sort.

* Merge sort:



2 < 3 , 3 <= 3

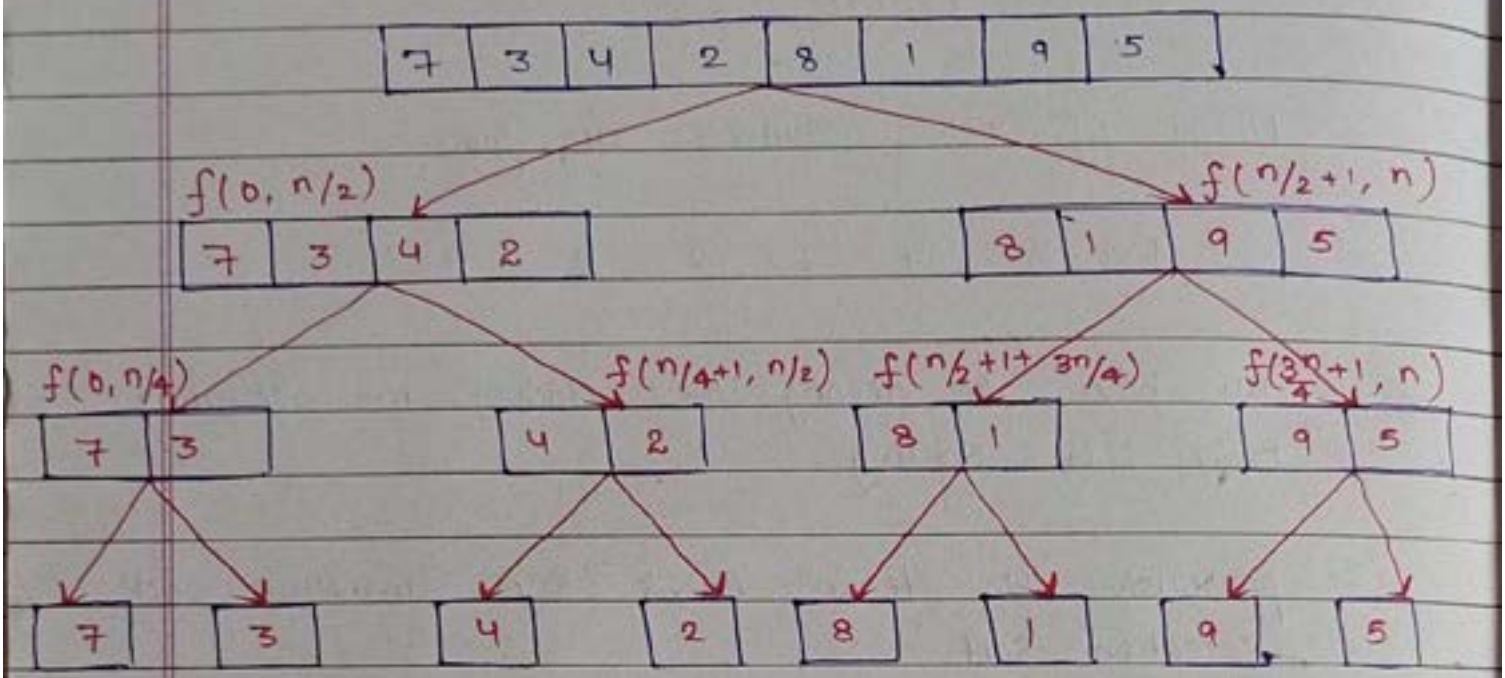
3 < 7

Depth first search

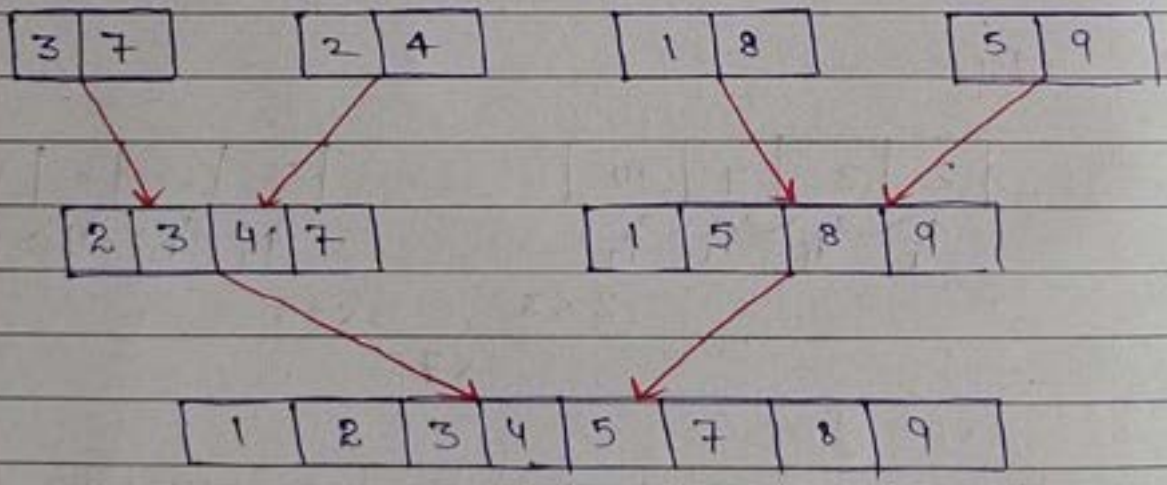
2	3	3	5	7	8	10	13

→ Divide the array in two parts of equal size.
→ Divide it until at the last the size of array becomes 1.
→ Now, start Merging the single size array in sorted order and then Merge all upper step.

Teacher's Signature



Now, it can reverse the process with comparing and sorting.



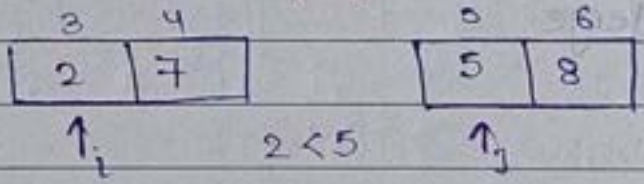
↳ This is sorted by Merge sort.

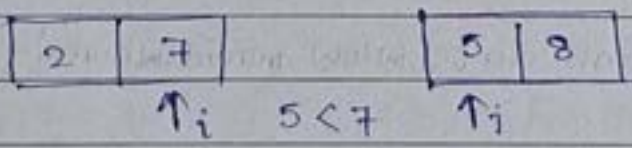
Teacher's Signature.....

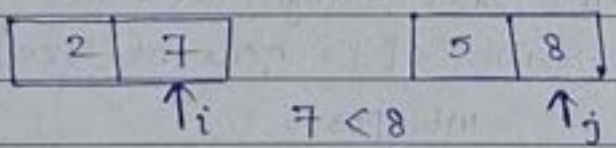
```
Merge_sort (int arr[], int low, int high) {

    if(high == low)
        return;

    mid = (low + high)/2;
    Merge_sort (arr, low, mid);
    Merge_sort (arr, mid+1, end);
     Merge_array (arr, low, mid, high);

}
```

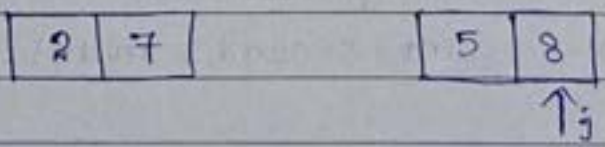
Merge_sort function can divide the array until the size of array becomes 1.

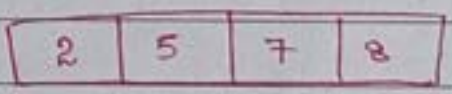
Merge_array can merge the after dividing with size = 1 and merge the element in sorted order.











↳ Final answer after sorting

Teacher's Signature.....

Size :

$n \rightarrow n/2^0$

$\downarrow$

$n/2 \rightarrow n/2^1$

$\downarrow$

$n/4 \rightarrow n/2^2$

$\downarrow$

$\vdots$

$\downarrow$

$n/2^k = 1$

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n$

→ Merge sort can follow divide and conquer method.

η swap in each heigh.

height = log n.

Total Time = n log n.
for Merge

⇒ Time Complexity : O(n log n)

```
Merge_array (int arr[], intlow, intmid, inthigh) {

    int size = high - low + 1;
    int a[] = new int[size];
    int pos = 0;
    int first = low;
    int second = mid+1;

while ( first <= mid && second <= high) {
```

Teacher's Signature

```
if (arr[first] <= arr[second]) {
        a[pos] = arr[first];
            pos++;
            first++;
        }
    else {
        a[pos] = arr[second];
            pos++;
            second++;
        }
}
while (first <= mid) {
    a[pos] = arr[first];
        pos++;
        first++;
    }
while (second <= high) {
    a[pos] = arr[second];
        pos++;
        second++;
    }

for (int i=0; i<arr.length; i++) {
    arr[i] = a[i];
}

    pos = 0, mid = low;
}       while(mid <= high)
        arr[mid++] = a[pos++];
}
```

Teacher's Signature.....

Space Complexity:

At every step we make array
↳ O(n)

Stack size : log n

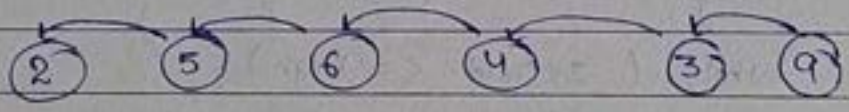
(n + log n)

* Quick Sort

0	1	2	3	4	5
2	5	6	4	3	9

Choose one element as pivot element

put the pivot element at Correct place



2 3 4 5 6 9

↳ Now the array is sorted.

pivot element : Choose any element as pivot Element. But we can organise in Correct Way.

So, We choose last element as pivot element.

Example-1

2 5 6 4 3 (9)

2 5 6 4 (3) 9

↑ pivot

Teacher's Signature.....

2 3 5 6 (4) 9

less than 3 | greater than 3

2 3 4 5 6 9

less than 4 | greater than 4

Now, Array is Sorted.

Example-2

3 1 5 6 2 (4) ← pivot

3 1 (2) 4 5 6

less than 4 | greater than 4

1 (2) 3 4 5 (6)

less than 2 | greater than 2

1 2 3 4 5 (6)

less than 6 | greater than 6

Now, Array is Sorted.

Teacher's Signature.....

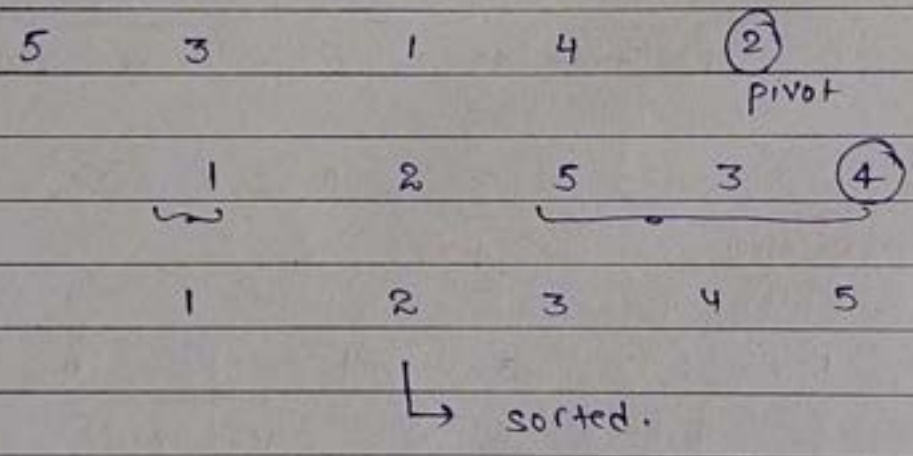
Page No.
Date : / /

Lecture - 27

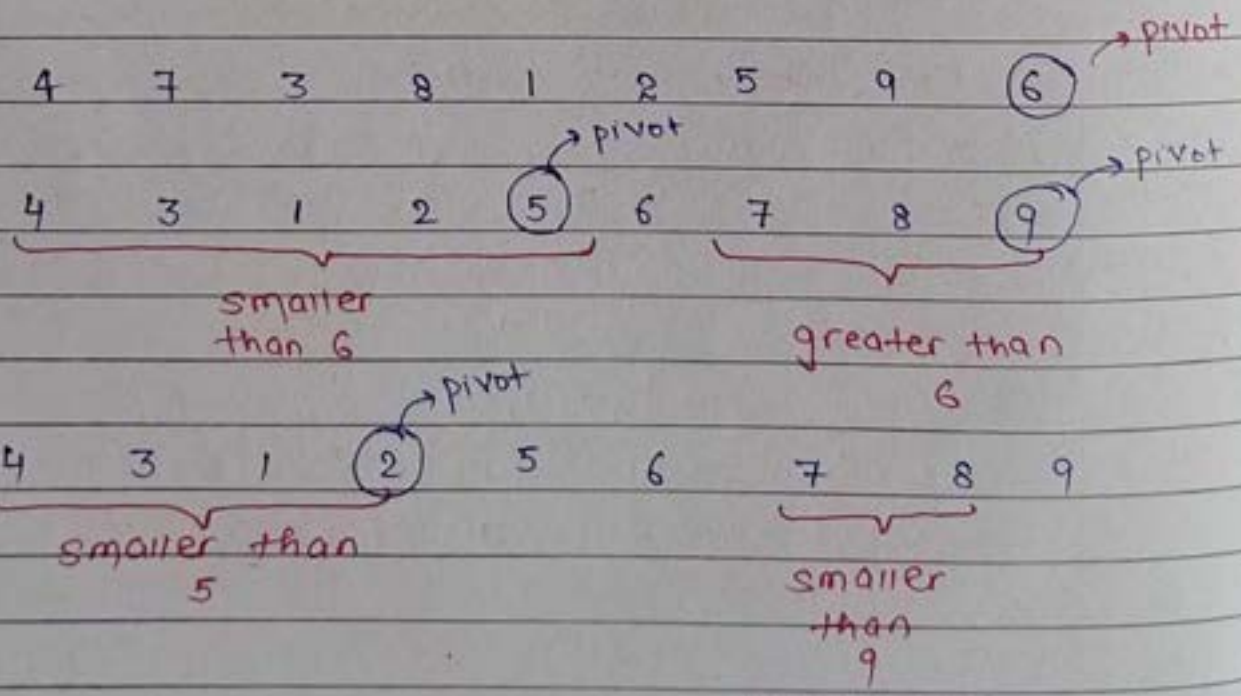
Quick sort

5 3 1 2 4

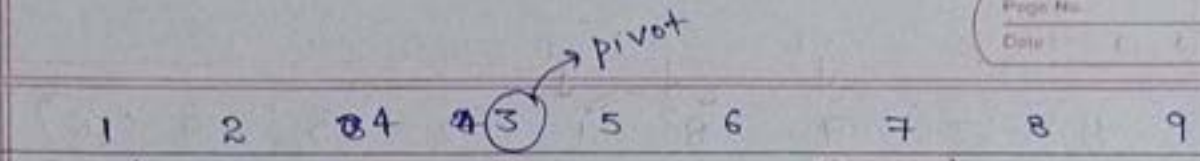
→ Choose a pivot element
→ start from first smaller element shift to left side and greater element shift to right side.



Ex:



Teacher's Signature.....

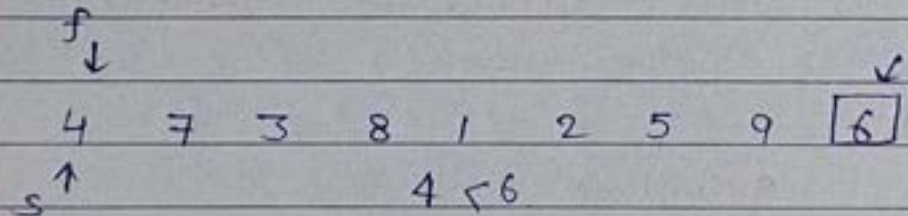


1 2 3 4 5 6 7 8 9

greater than 3

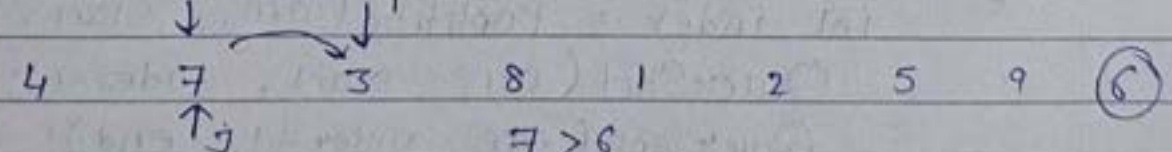
We got sorted result.

→ Find pivot Element
→ put pivot element in its Correct position.
→ Visit left part of array and visit right part of array.
→ Repeat 3 step until Array size become 1.



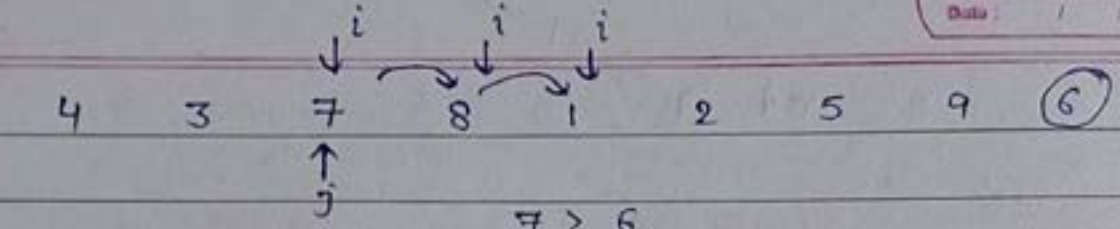
1st pointer : Value ko fill kaha karna hai uska position batayega.

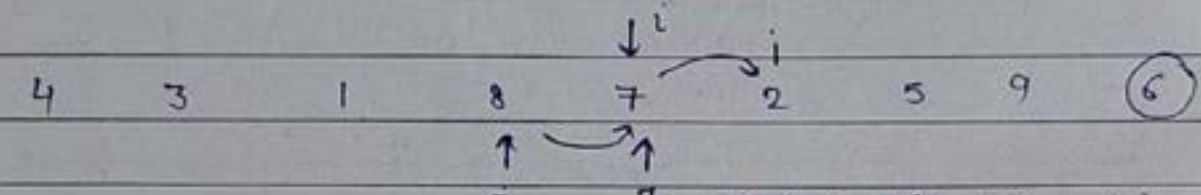
2nd pointer : Array ko traverse karega.



3 < 6 swap (i, j)

Teacher's Signature.....





4 3 1 2 5 7 8 9 6

4 3 1 2 5 6 7 8 9

```
if ( arr[second]  <= pivot ) {
        swap ( arr[first], arr[second] );
        first++;   second++;
    }
else
    second++;
```

→
```
QuickSort ( int [] arr, int start, int end ) {

    if ( start >= end )
        return;
    int index = Partition ( arr, start, end );
    QuickSort ( arr, start, index-1 );
    Quicksort ( arr, index+1, end );

}
```

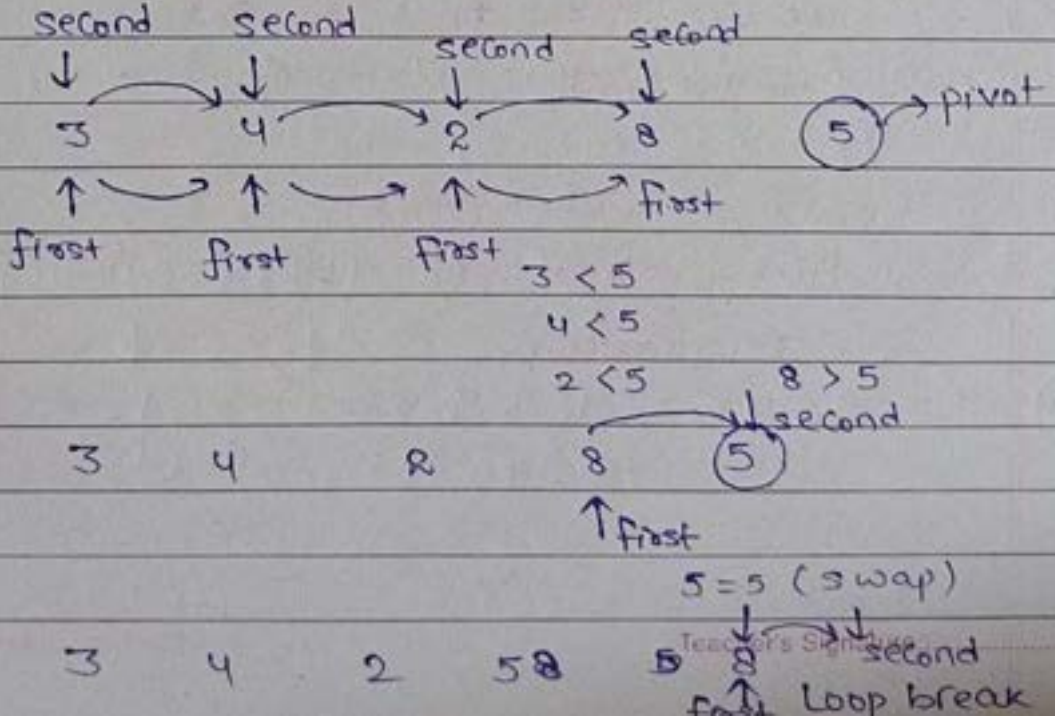
Teacher's Signature.....

Page No.
Date

Array ko Break tab tak karna hai.

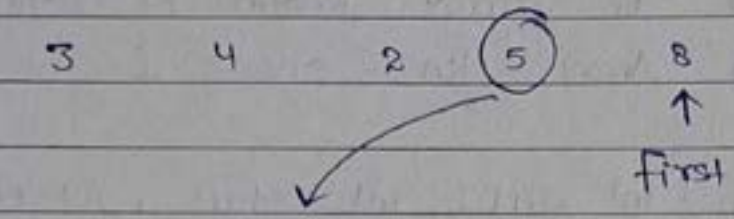
1) ya to Array khatm ho Jaye.
2) ya Array ka size 1 ho Jaye.

```
int partition (int arr[], int start , int end) {

    int pos = start;
    int first = start;
    int second = start;

    While ( second <= end ) {
        if (arr[end] >= arr[second]) {
            int temp = arr[first];
            arr[first] = arr[second];
            arr[second] = temp;
        } first++;   second++;
    } else
        second++;
}
return first - 1;
}
```



Loop break

second ↓



Time Complexity :

$O(N \log N)$

If array sorted : $O(N^2)$

Space Complexity :

$O(N)$ → of array Sorted

$O(\log N)$

* Inversion of array :

N = 5,

arr[] = {2, 4, 1, 3, 5}

output = 3

{2, 1}, {4, 1}, {4, 3}

3 inversion

Teacher's Signature

Check for every element, How many smaller element are present after element.

Count and return.

2 3 1 4 5

Condition

$a[i] > a[j]$ } inversion
$i < j$

2 4 1 3 5

for 2 : 1

1 element is smaller

Count = 1

for 3 : 1

1 element is smaller

Count = 2

for 1 : 0

for 4 : 1

Count = 3

for 5 : 0

return 3

By this we take $O(N^2)$

But we have to solve in $O(N \log N)$

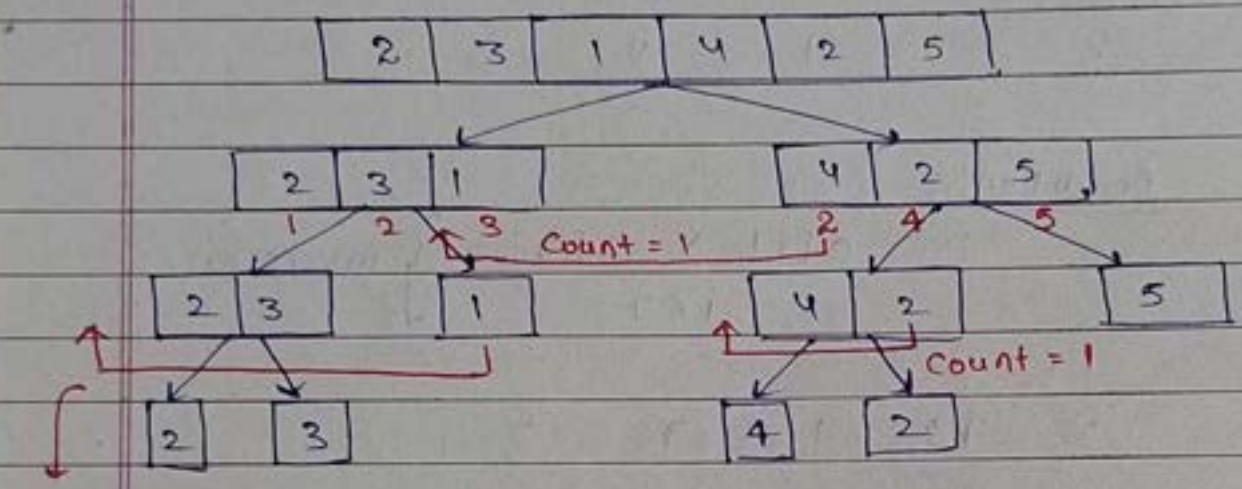
Teacher's Signature.....

Merge sort (NlogN)

Quick sort (Nlogn)

↓

Try By this



Total = 2 + 1 + 1

= 4

Merge sort

↳ and take a variable Count and increment whenever we shift any element.

Code :-

```
class solution {
    static long ans;
Static Void merge (long arr[], int l, int mid, int r) {
        long []c = new long [r-l+1];
        int i = l, j = mid+1, k = 0;
    While (i<= mid && j <= r) {
        if (arr[i] > arr[j]) {
            ans += mid - i + 1;
            c[k++] = arr[j++];
        }
```

Teacher's Signature.....

```
        else {
            c[k++] = arr[i++];
        }
    }
    while (i <= mid) {
        c[k++] = arr[i++];
    }
    while (j <= r) {
        c[k++] = arr[j++]; }
    k = 0, mid = l;
    while (mid <= r)
        arr[mid++] = c[k++];
}
static void mergeSort (long arr[], int l, int r){
        if (l = r) return;
        int mid = (l + r) / 2;
    mergeSort (arr, l, mid);
    mergeSort (arr, mid+1, r);
    merge (arr, l, mid, r);
}
static long inversionCount (long arr[], long N) {
        ans = 0;
    mergeSort (arr, 0, (int)(N-1));
    return ans;
}
}
```

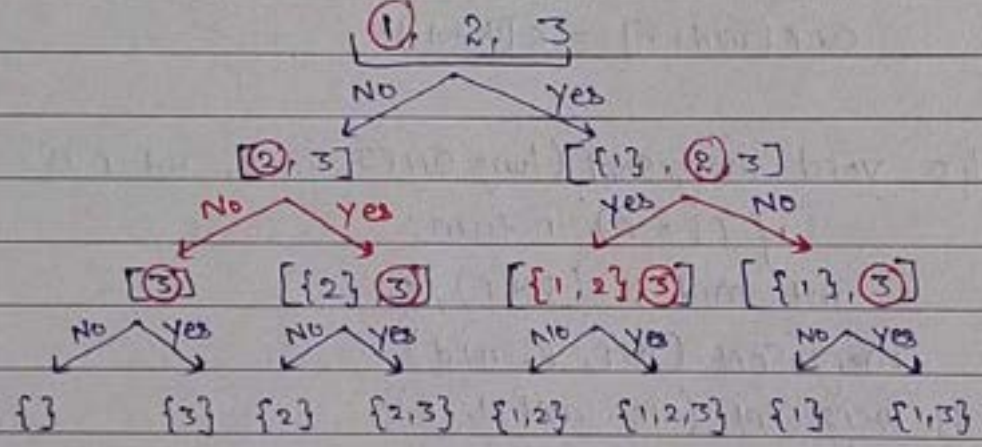
Teacher's Signature.....

* Coins : 1, 2, 3

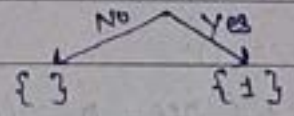
You have select as you want
either 0 Coin, 1 Coin, 2 Coin, 3 Coin at one time.

possibility : { }, {1}, {2}, {3}, {1, 2}, {1, 3},
{2, 3}, {1, 2, 3}

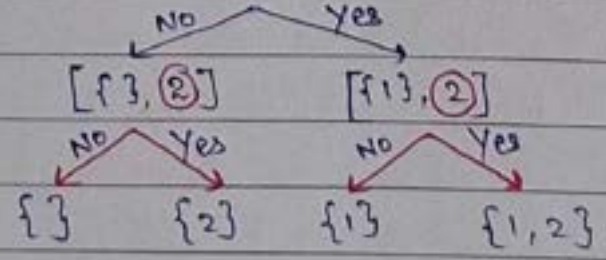
3 Coins → 2^3 possibility
= 8



* Coin = 1



* Coins = 1, 2



Teacher's Signature.....

*

6	3	1	2	4	8

Sum = 13

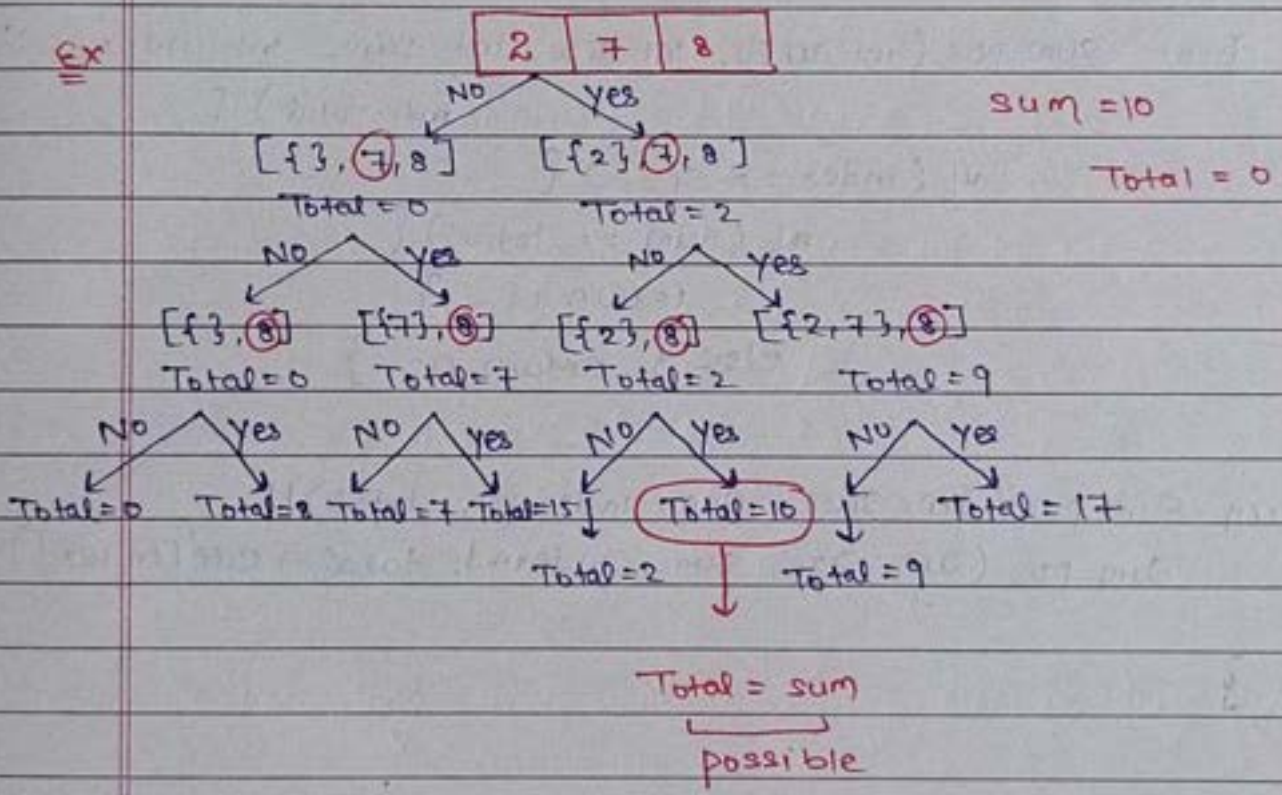
Choose any element any number of time. can we find sum of element which is equal to sum.

When we make tree:

Total possible case

$2^6 = 64$.

Ex



Total = sum possible

Time Complexity : 2^N.

Teacher's Signature.....

```
int main() {
    int arr[] = { 2, 5, 1, 6, 7, 11 };
    int sum = 20;
    int total = 0;

    bool ans = sum_pos ( arr, 6, sum, 0, total );
        cout << ans;
    return 0;
}

bool sum_pos ( int arr[], int size, int sum, int index,
                                        int total ) {
        if ( index == size ) {
            if ( sum == total ) {
                return 1; }
            else { return 0; }

return sum_pos ( arr, size, sum, index+1, total) ||
    sum_pos ( arr, size, sum, index+1, total + arr [index]);

}
```

Teacher's Signature.....

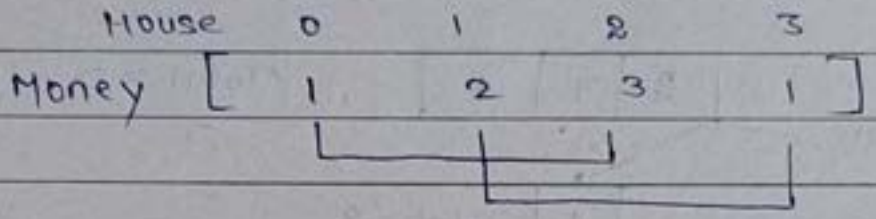
Recursion

Lecture - 28

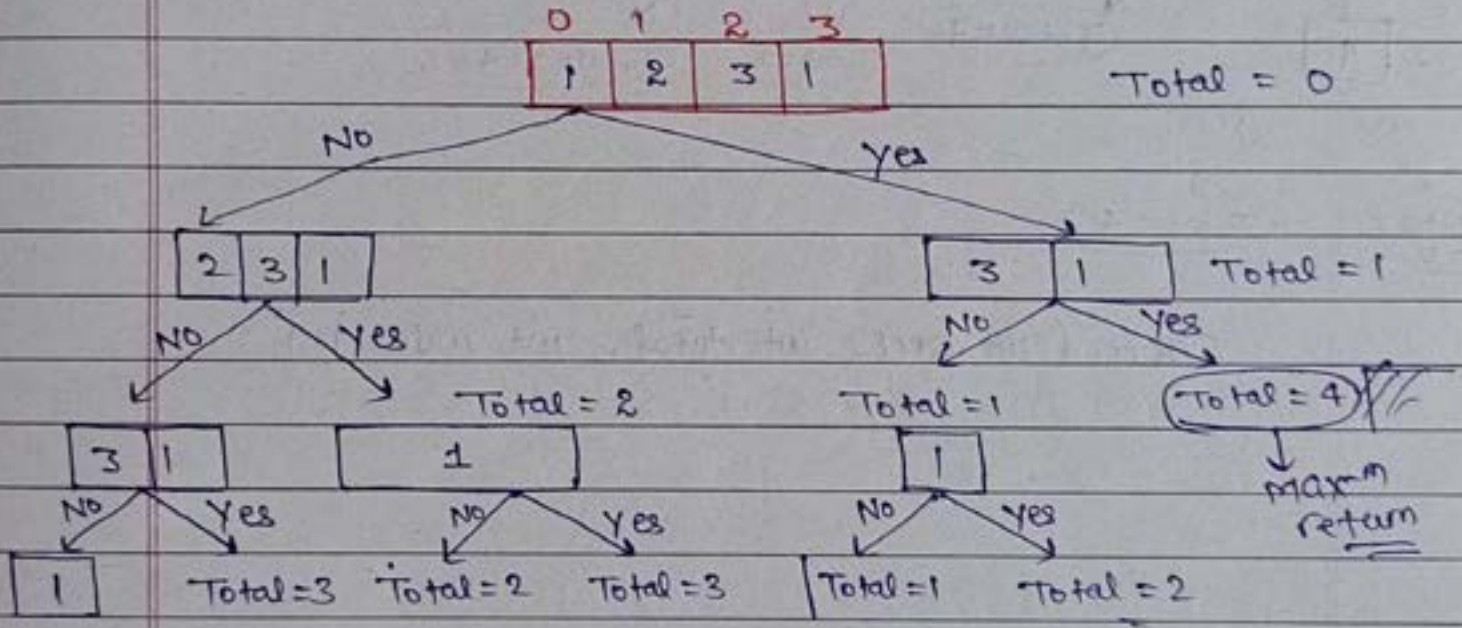
* House Robber : (leetcode) :

nums = [1, 2, 3, 1]

चोरी Alternate घर से करनी है।



Either (0 & 2) or (1 & 3)



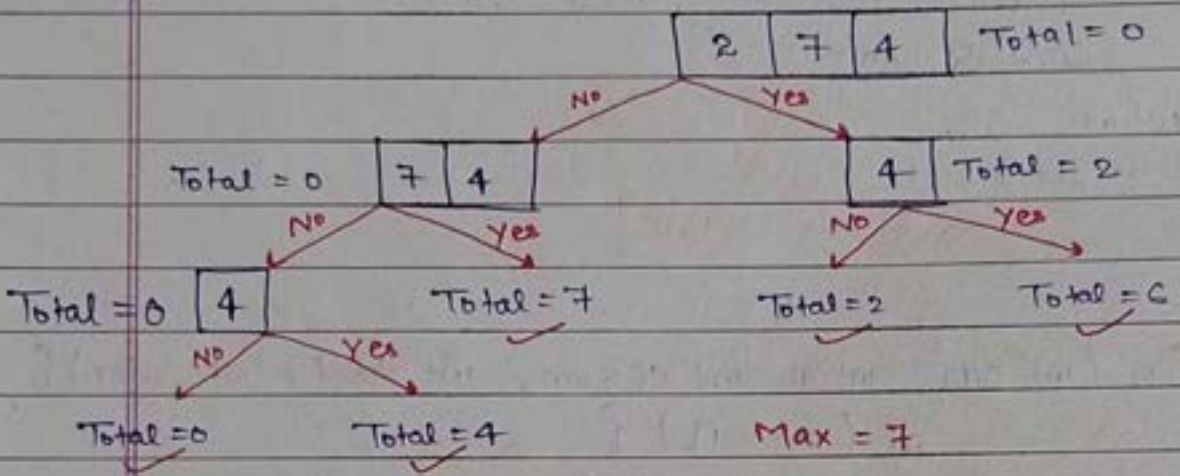
House :	0	1	2	3	4
Money :	1	2	3	1	4

Alternate grah में चोरी करेगा
Maximum kitna चोरी कर सकता है।
↓
return.

Teacher's Signature

House	0	1	2
Money	2	7	4

$2+4=6$ (7) → Max



Code in C++ :

```
#include <iostream>
using namespace std;

int main() {

    int arr[] = {2, 7, 4};
        int total = 0;
        int index = 0;
        int sum = 0;
        Chori (arr, 3, sum, total, index);
            Cout <<sum;
        return 0;

}
```

```
int main () {
    int n;
    int arr[n];
    cin >> n;
    int sum = 0;

ans = chori ( arr, n, sum, total, 0 );   // 0 : index
```

```
Void chori (int arr[], int size, int &sum, int total,
                                    int index) {

    if (index >= size) {
        sum = max( sum, total);
        return;
    }

chori ( arr, size, sum, total, index + 1 );
chori ( arr, size, sum, total + arr[index] , index + 2);

}
```

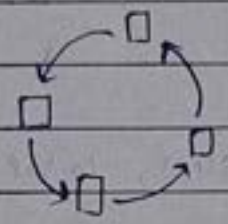
This Codes gives Time limit Exceed.

Teacher's Signature.....

House Robber - II

Circular array

Alternate House से चोरी।



Circular House.

House	0	1	2	3	X
Money	2	4	3	7	

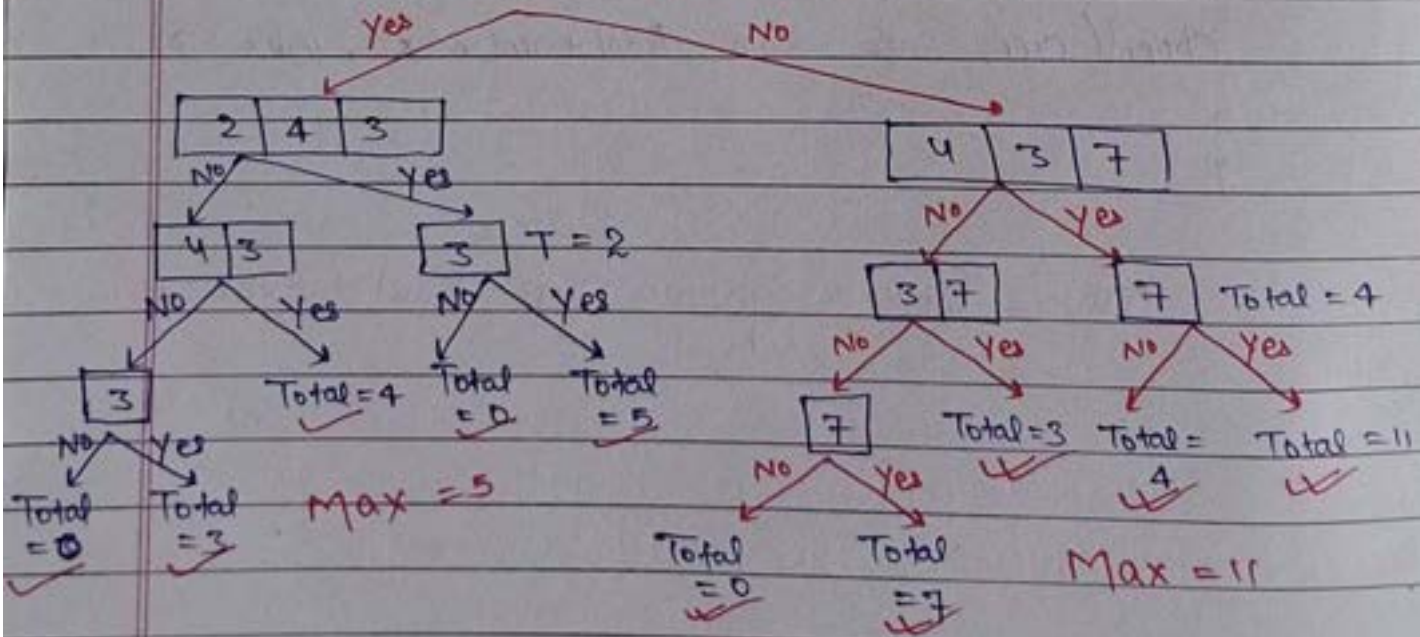
{2} {4} {3} {7}

{2,3} {4,7} {x}

↳ Maximum = 11

↳ return.

0	1	2	3
2	4	3	7



Max = 11

Teacher's Signature.....

Page No.
Date / /

* Combination Sum :-

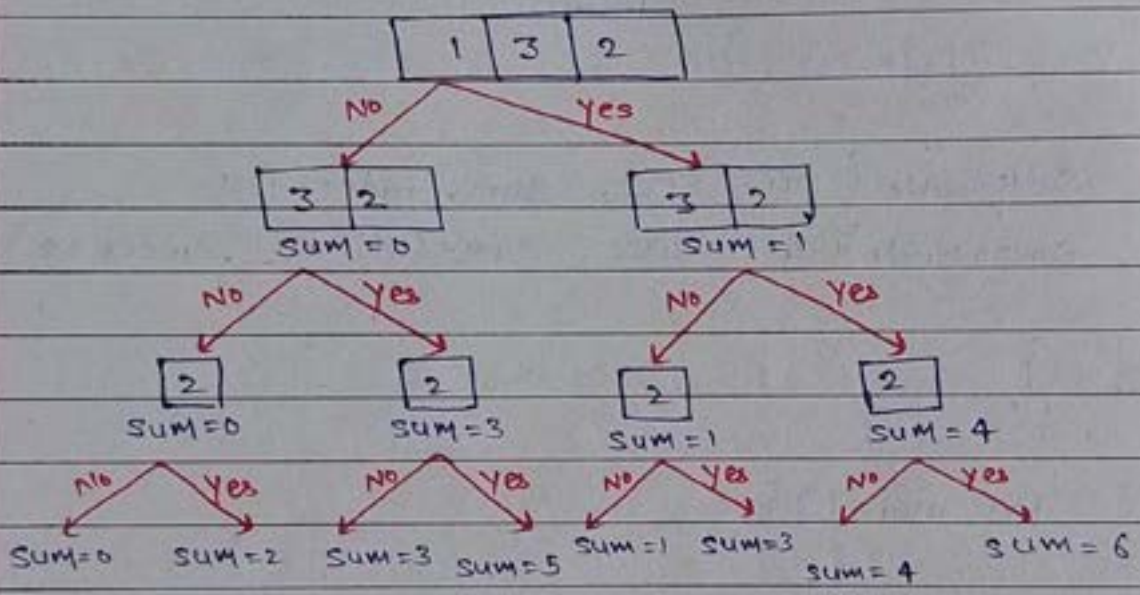
2 1 4

{ } {2} {1} {4}

{2+1=3} {1+4=5} {2+4=6}

{2+1+4=7}

8 possibility




```
SumPossible ( int arr[], size, sum, index ) {
        if (index >= size) {
            cout << sum << " ";
            return ;
        }
    sumpossible( int arr[], size, sum, index+1 );
    SumPossible ( int arr[], size, sum+arr[index], index+1);

}
```

Teacher's Signature .....

→ Check that sum = 7 is possible by any operation.

```
int main() {
    int arr[5] = { 2, 4, 1, 5, 8 };
    int index = 0, sum = 0;
    int ans = 0;
    int target = 7;
ans = Sumpossible ( arr, n, index, sum, ans, target );
    cout << ans;
}
```

```
int main () {

    int arr [5] = { 2, 4, 1, 5, 8 };
    int index = 0, sum = 0;
    Sumpossible ( arr, 5, sum, index );

    return 0;
}
```

Teacher's Signature.....

House Robbery II (code)

```
Void find(vector <int> &nums, int i, int &sum, int total){

        if (i >= nums.size()) {
            sum = max (sum, total);
            return;
        };

    find( nums, i+2, sum, total + nums[i]);
    find (nums, i+1, sum, total);

}

int robbery (vector<int> & nums) {
        if (nums.size() == 1) return nums[0];
        int sum = 0, total = 0;
      find (nums, 1, sum, total);
        total = 0;
        nums.pop_back();
      find ( nums, 0, sum, total);

    return sum;

}
```

Teacher's Signature.....

Check that the sum target is possible or not.

if possible return 1.
Not possible return 0.

only Minor change in Code:

```
Void sumpossible (int arr[], int size, int index, int
                  sum, int target, int &ans){

    if (index == size) {
        if (sum == target) {
            ans = 1;
        }
        return;
    }

sumpossible (arr, size, index + 1, sum, target, ans);

sumpossible (arr, size, index + 1, sum + arr[index],
                  target, ans);

}
```

In Main:

```
ans = 0;

cout << ans;
```

Teacher's Signature.....

Code in Java :

```
public class TargetSum {

    static boolean Sumpossible (int arr[], int size, int index,
                    int sum, int target) {
        if (index == size) {
            if (sum == target) {
                return true;
            }
            else {
                return false;
            }
        }
        return Sumpossible (arr, size, index+1, sum, target) ||
            Sumpossible (arr, size, index+1, sum+arr[index], target);
    }

    public static void main (String[] args) {
        int arr[] = {1, 2, 4, 7, 3};
        int index = 0, sum = 0;
        int target = 12;

        boolean ans = Sumpossible (arr, 5, index, sum, target);
        System.out.println (ans);

    }
}
```

Teacher's Signature.....

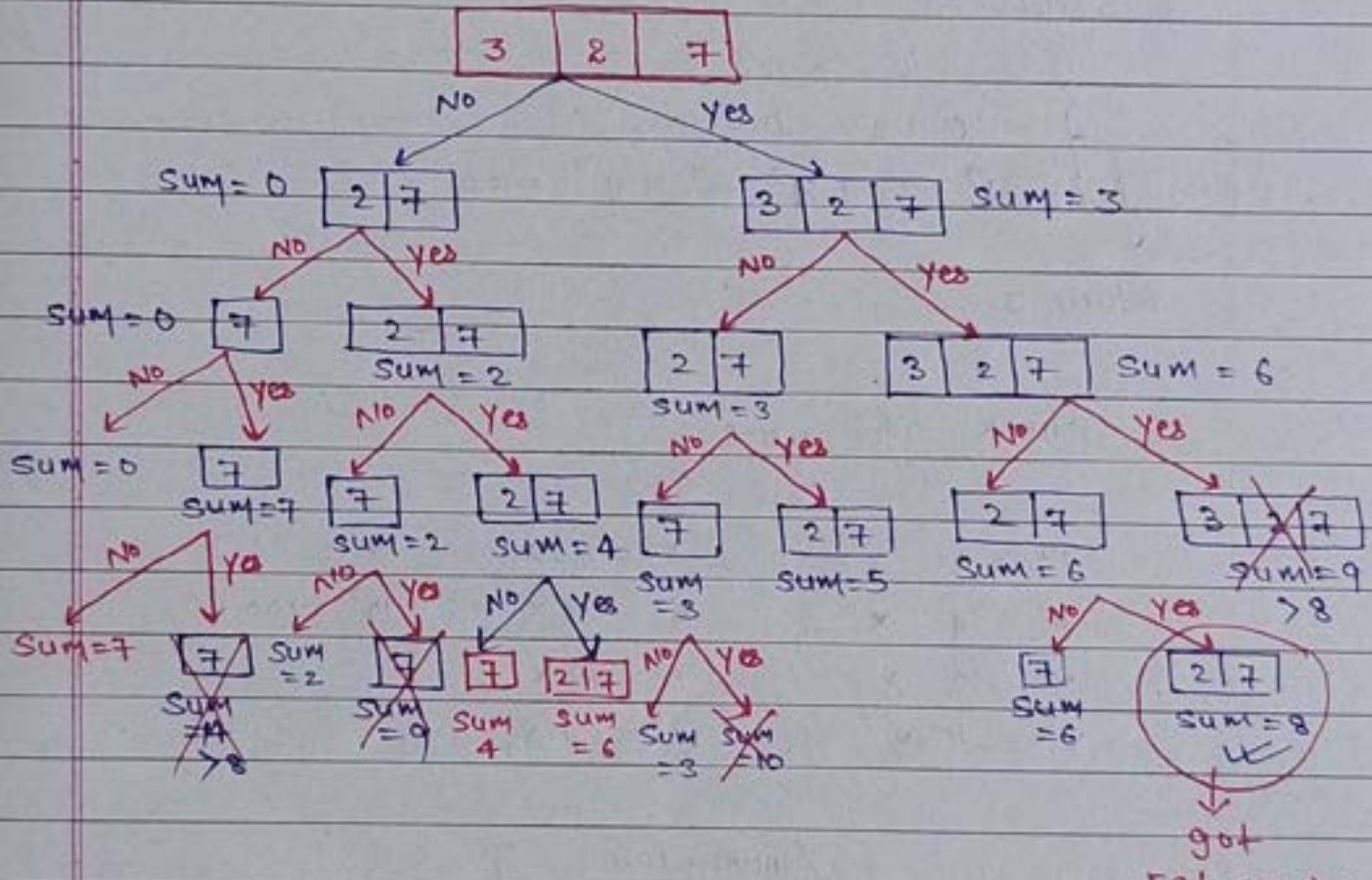
3, 2, 7

Target = 8

one number used many times.

possibility : 3+3+2 = 8

2+2+2+2 = 8



return 1;

```
Void sumpossible(int arr[], int size, int index, int total, int sum,
                                                    int &ans){
    if(index == size){
        if( sum == total) ans = 1;
        return ;
    }
    if(sum > target) return ;
    sumpossible( arr, size, index+1, total, sum, ans);
    sumpossible( arr, size, index, total, sum+arr[index], ans);
}
```

Teacher's Signature.....

* Count Ways to express a number as Sum of powers

Number = 100

power = 2

possibility :

$10^2 = 100$

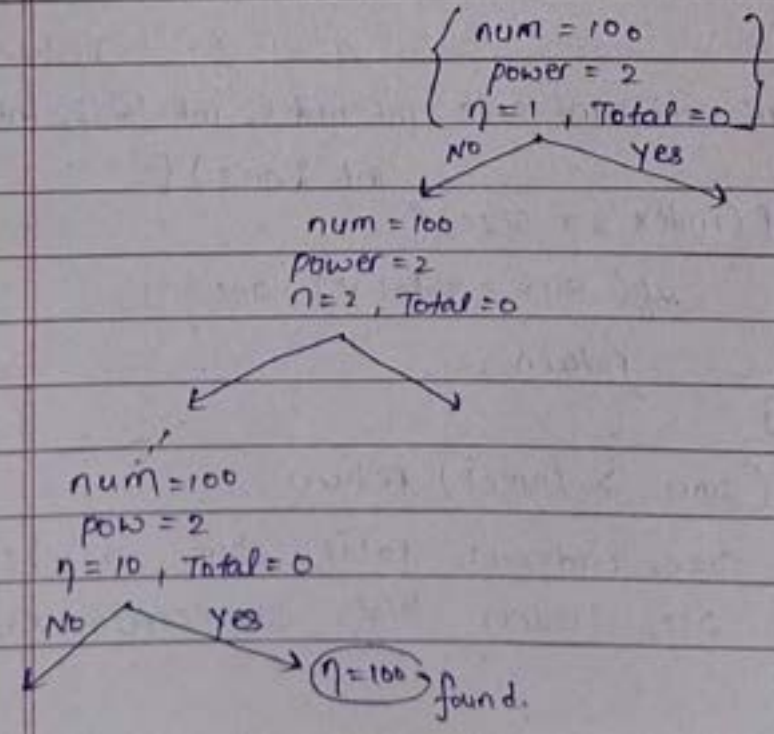
$6^2 + 8^2 = 100$

$1^2 + 2^2 + 4^2 + 5^2 + 7^2 = 100$

return 3.

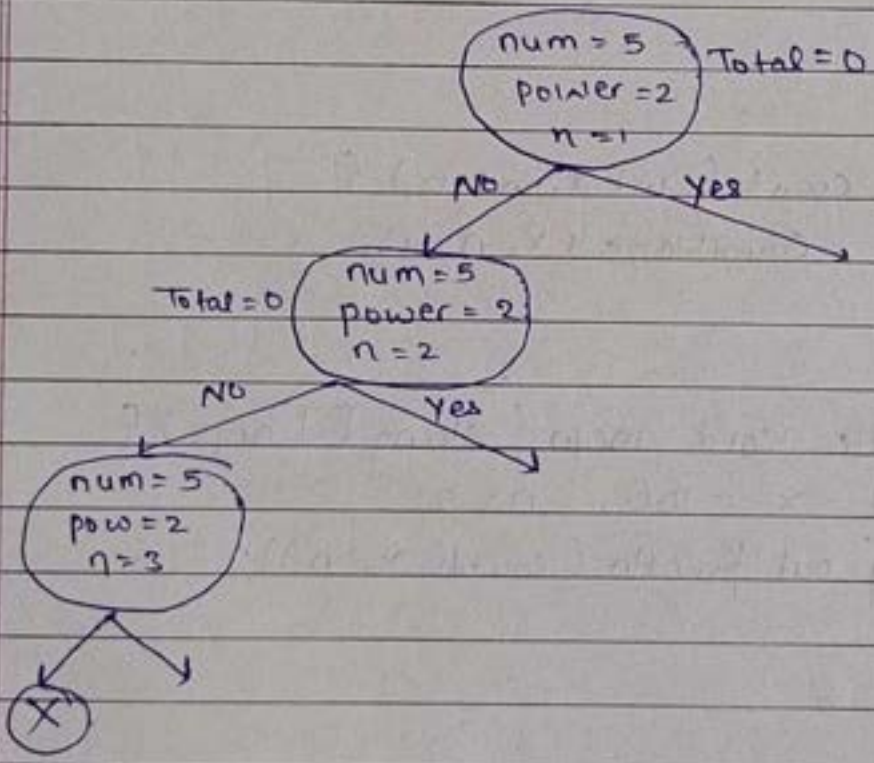
check for all :

$1^2 = 1$ x	$5^2 = 25$ x	$9^2 = 81$ x
$2^2 = 4$ x	$6^2 = 36$ x	$10^2 = 100$ ✓
$3^2 = 9$ x	$7^2 = 49$ x	
$4^2 = 16$ x	$8^2 = 64$ x	



Teacher's Signature

```
Sumpossible (int n, int num, int power, int total, int Count){

    if( total == num) {
        Count++; return;
    }

    if (total > num)
        return n;

Sopossible (n+1, num, power, total, Count);
Sumpossible ( n+1, num power, total + pow (n, power), Count );

}
```



Teacher's Signature

Code (Java):

```
Static int CountWays (int X, int n, int num) {
        int val = (int)(X - Math.pow(num, n));
          if (val == 0)
              return 1;
          if (val < 0)
              return 0;

return CountWays (val, n, num + 1) +
          CountWays (X, n, num+1);

}

Static int Count ( int X, int n) {
      return CountWays (X, n, 1);
  }

public static void main (String [] args) {
          int X = 100, n = 2;
    System.out.println (Count( X, n));
    }
}
```

Teacher's Signature

Lecture - 29

Recursion, palindrome, permutation

1) Given a string s and the task is to print all the substring present inside the string s.

s = abc

↳ Total substring

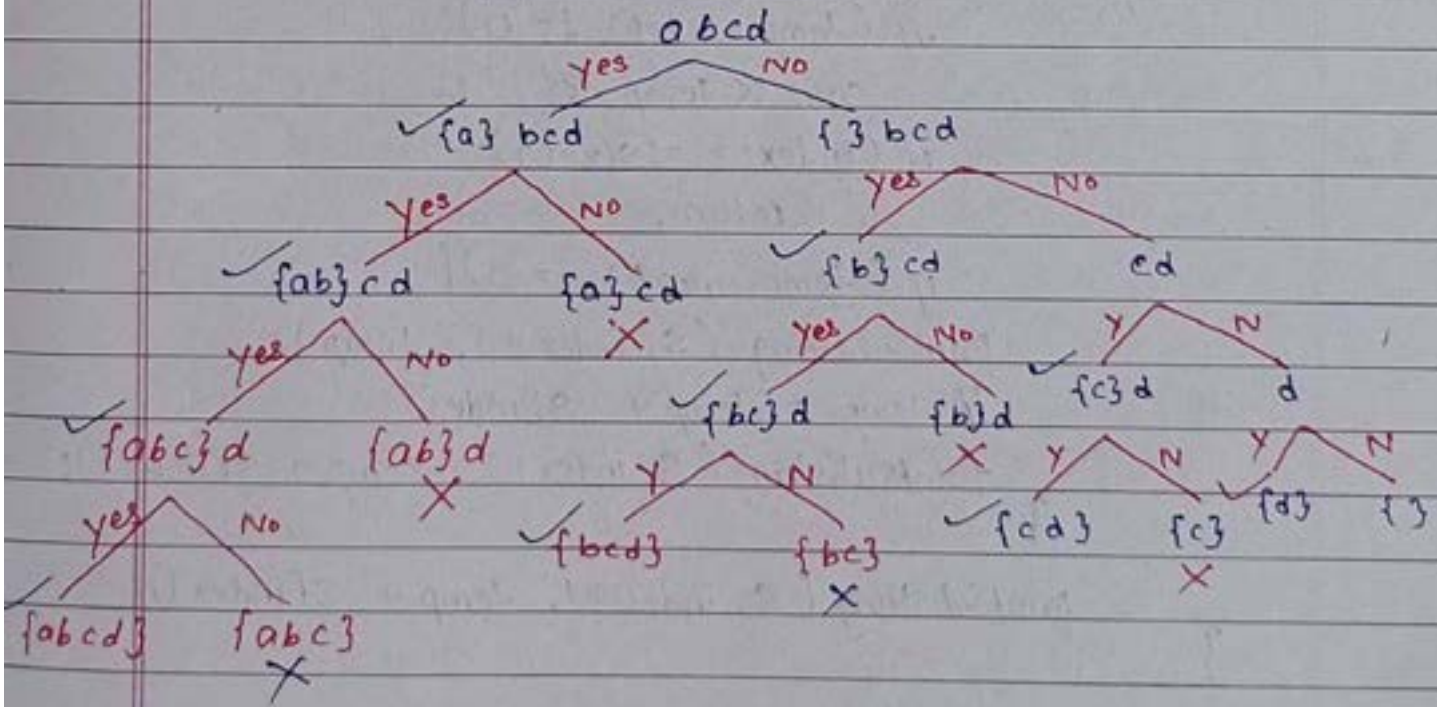
a, b, c, ab, bc, abc

s = abcd

a, b, c, d, ab, bc, cd, abc, bcd, abcd

↳ Total substring

By recursion tree



Total = 10

Teacher's Signature.....

1st Approach

print all substring using for loop.

```
for (int i=0;  i< s.size(); i++){
    for(int j=0 ; j< s.size(); j++){
        print substring ( i to j);
    }
}
```

2nd Approach

By recursion:

→ Element found : print it
→ After that which ever element found print it

```
Void printSubstring ( String S, int index, String temp){
        if (temp.size() != 0)
            cout << temp << " ";
        if (index == s.size())
            return;
        if ( temp.size() == 0){
        printSubstring (S, index+1, temp);
        // temp = temp + S[index];
        PrintSubstring ( S, index+1, temp + S[index]);
        }
    printSubstring (S, index+1, temp + S[index]);
}
```

Teacher's Signature.....

```
int main() {
    String s;
    cin >> s;
    printSubstring (s, 0, " ");
    return 0;
}
```

2) Combination sum

You have given an array of n elements and a target. By adding elements from array find the target and return the array of element.

Note: one element can be repeated any No. of time.

Ex: Candidate = [2, 3, 6, 7] target = 7

output: [2, 2, 3]
[7]
→ It is like a 2D-Array

Size is not fixed, so we used arraylist or vector

```
vector <vector <<int>> ans; → 2D vector
vector <int> vec;           → 1D vector
```

In 2D vector, we can directly push 1D vector by

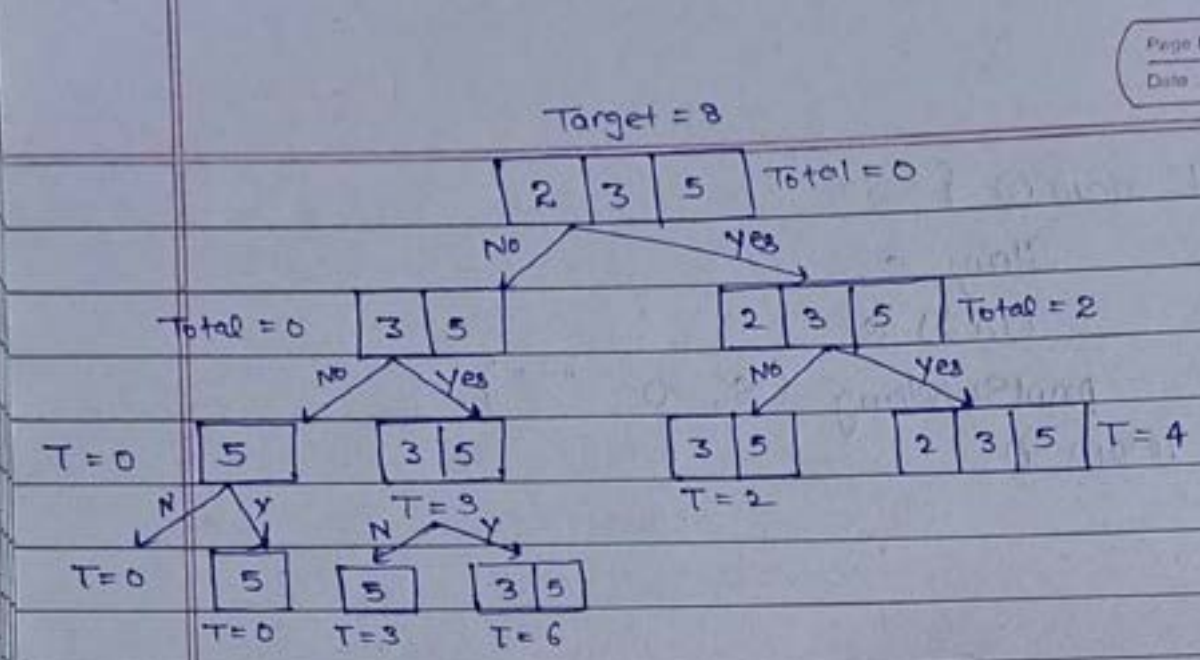
```
ans.push_back(vec);
```

or

```
ans[0].push_back(5);
```

↳ In this we put element by element in the array.

Teacher's Signature



Breaking point :

① if (sum > target) {

}

② if (sum == target) {

}

③ if (index == arr.size) {

}

Here we need a vector or Arraylist.

* Code in C++ :

```
Vector<vector<int>> CombinationSum (vector<int>&
        candidates, int target) {
    vector<vector<int>> ans;
        vector<int> temp;
```

Teacher's Signature.....

```
        int sum = 0;
    find (candidates, ans, temp, sum, target, 0);
     return ans;
}

void find (vector<int> & candidates, vector<vector<int>> & ans,
      vector<int> temp, int sum, int target, int index){

        if (index == candidate.size()) {
              if (sum == target)
                 ans.push_back (temp);
              return;
        }
        if (sum > target)
           return;

    find (candidates, ans, temp, sum, target, index+1);
      sum += candidates [index];
     temp.push_back (candidates [index]);
    find (candidates, ans, temp, sum, target, index);

}
```

Teacher's Signature.....

3) Permutation of a given string

You have given a string.
You have to find all the permutation of given string.

Ex: ABC

→ ABC ACB
BAC BCA
CAB CBA

What we do here

take 2 possibility

A _ _
A _ _
A B C A C B

Similarly,

B A/C _
B A C B C A

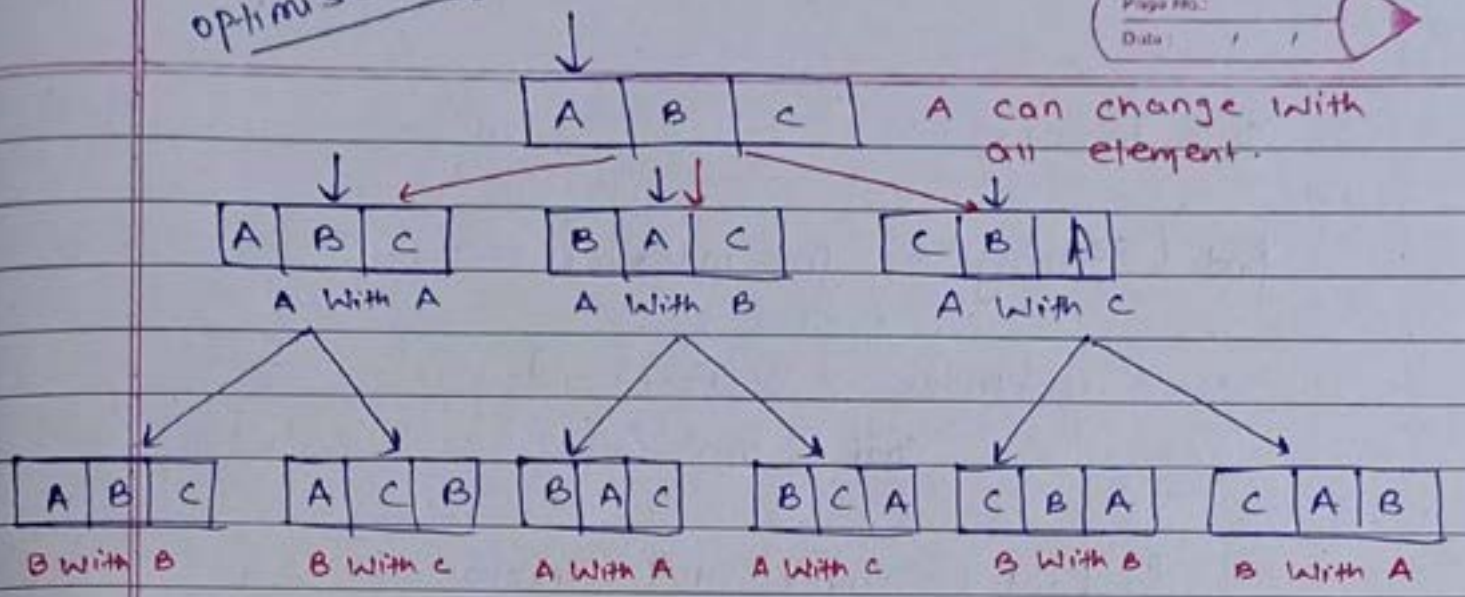
C A/B _
C A B C B A

No, we can understand it by Recursion tree:

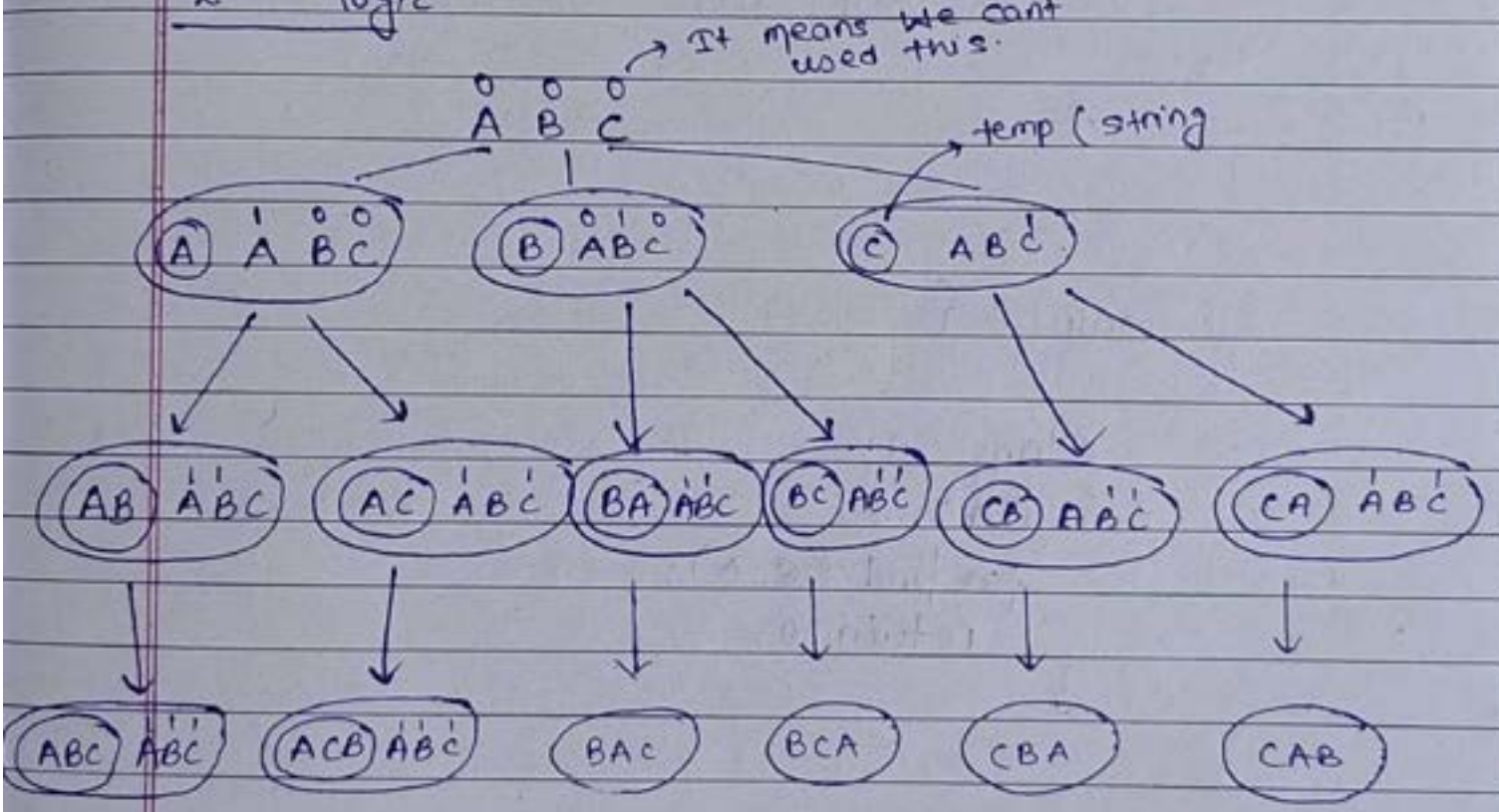
How it works:

Teacher's Signature.....

optimise logic



2nd logic



Teacher's Signature.....

Code in C++;

```
find ( String s, int index) {

    if (index = s.size()-1)
        Cout << s << " ";

for (int i= index; i < s.size(); i++) {
        Swap( s[i], s[index]);
        find ( s, index+1);
        Swap ( s[i], s[index]);

    }
}

int main() {

    String s;
    Cin >> s;
    ~~pr~~ find (s, 0);
    return 0;
}
```

Teacher's Signature.....

4) Letter Combination of a phone number :

→ On keypad phone

↳ If we want to write alphabet then we use number button from 2 to 9

→ you have get a number and find all the possible string which is generated by phone.

2 abc 3 def

4 ghi 5 jkl 6 mno

7 pqrs 8 tuv 9 wxyz

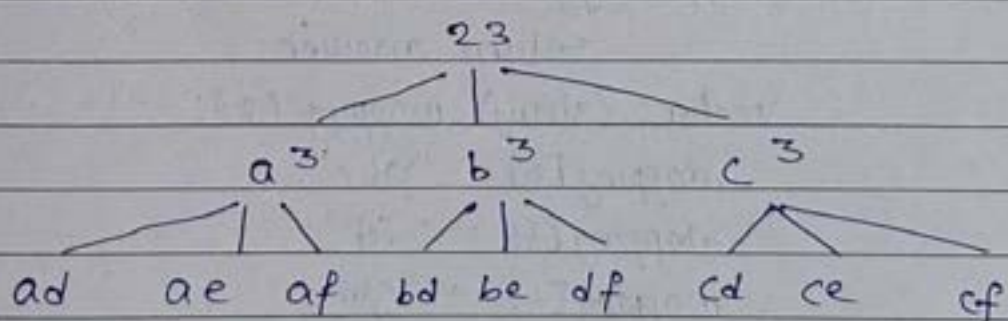
If number = 5

possible string : j, k, l

let num = 23

2 3

a b c d e f

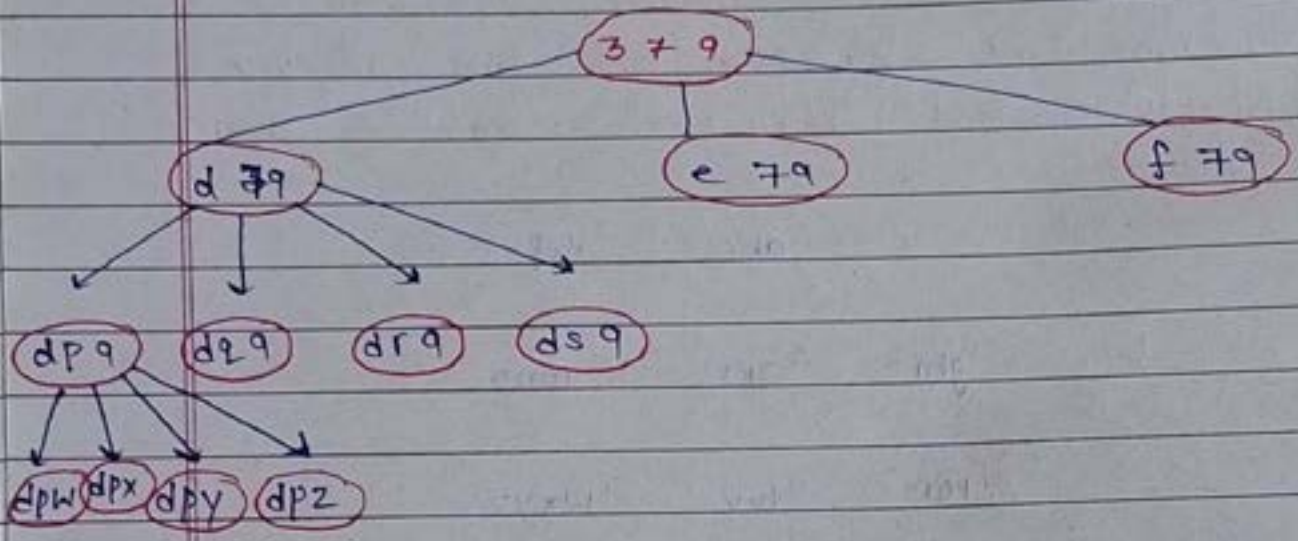


These are the possibilities which can be make by 23.

Teacher's Signature.....

Num = 379

3	7	9
def	pqrs	wxyz



By this we can calculate other strings.

Code:

```
Vector <string> letterCombination (string digits){
        Vector <string> answer;
        if (digits.size() == 0)
            return answer;
        Vector <string> mapping (8);
        mapping[0] = "abc";
        mapping[1] = "def";
        mapping[2] = "ghi";
        mapping[3] = "jkl";
        mapping[4] = "mno";
        mapping[5] = "pqrs";
        mapping[6] = "tuv";
```

Teacher's Signature.....

```
        mapping[7] = "wxyz";
    find(digits, answer, mapping, "", 0);
    return answer;
}

Void find(String &digits, Vector<string> &answer,
          Vector<string> &mapping, String temp, int index){

        if(index == digit.size()){
            answer.push_back(temp);
            return;
        }

        int pos = digits[index] - '2';
    for(int i=0; i<mapping[pos].size(); i++)
find(digits, answer, mapping, temp+mapping[pos][i],
                                index+1);

}
```

Page No.
Date : / /

Lecture - 30

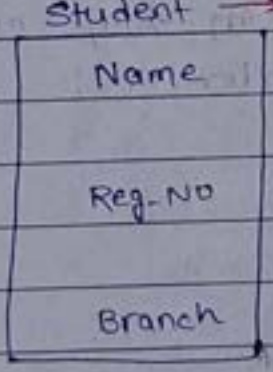
Object oriented programming (OOPs)

OOPs : { Class, Object }

We will talk about class and object in this lecture.

Class : It is a userdefined datatype. It is a blueprint for the object. It is a logical construct.

Object : It is a instance of a class. Object can occupies space in Memory.



Earlier, if i have 10 student and we want to store the information of student like name, Reg-No and Branch, then we have make an array for all which we want to store.

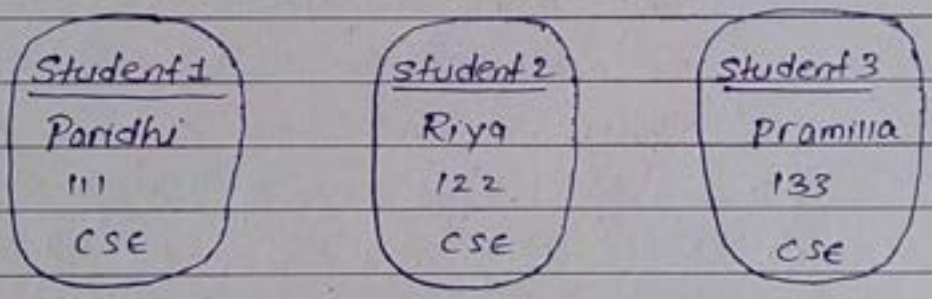
```
String name [10];
int Reg-No [10];
String Branch [10];
```

Teacher's Signature.....

We have to make 3 array for storing.
Accessing the data of 1 student can be difficult that we have to access all the array.

If we wont to add one more data then we have to add a different array.

→ Can we make it easier to store the data?
Yes, if data of a student can be stored in a object. then it is easier to access.



By this way we can access easily.

→ class name (user defined datatype)

→
```
Class Student {
    String name;
    int Reg_No;
    String Branch;
}
```

How to access data of 1 student:

→ Object

```
int main() {
    Student Std1;
    Std1.name = "Paridhi";
    Std1.Reg_No = 111;
    Std1.Branch = "CSE";
```

Teacher's Signature.....

```
    Cout << St1.Reg_NO;
    return 0;
}
```

By default class is private.
So, in above program it gives error.
We have to make the class as public.

```
Class student {
    public:
    String name;
    int Reg_NO;
    String branch;
}
```

Now, the Code Works.

Access specifier :

- → public
- → private (By default)
- → protected

Class → Behaviour
Class → State / property

property like :

- Memory
- Color
- Company

Teacher's Signature.....

Behaviour / Action :

→ Click : photo clicked

→ setting : execute.

We can also make function inside class

↳ to show behaviour.

→ let us take an example of Bank:

```
Class Bank {
    public :
     int balance;
     int name;
    Void Balance_check () {
        Cout << balance << " ";
    };
    Void Withdraw () {
        balance -= 100;
        Cout << "₹ 100 Withdrawn";
        Cout << "New Balance " << balance;
    };
}
int main() {
        Bank Customer1;
        Customer1.balance = 500;
        Customer1. Name = "Riya";
        Customer . Balance_check();
        customer. Withdraw();
    return 0;
}
```

Teacher's Signature.....

In, this program there is a problem:
User can self modify the balance, if user have access then it is not safe.
We can change the data by self.
So, what we do?
Make balance as private:
And make public to those which is required:-

```
class Bank {
    int Balance;
    String name;
} -> private

public:
Void setValue (int amount, String names){
        Balance = amount;
        name = names;
    }
Void checkBalance (){
        Cout << Balance;
    }
Void Withdraw(){
        Balance -= 100;
    Cout << "₹100 Withdrawn";
    Cout << Balance;
    }
}
int main (){
        Bank Customer1;
    Customer1.setValue (10000, "Paridhi");
        Customer1.check_Balance ();
        Customer1.Withdrawn();
}
```

Teacher's Signature.....

This can also create problem. Because user or anyone can update the Balance easily.

So, we have to make this like, whenever we want to update balance, we cannot update it directly. We can update it with the permission of bank employee. and they can verify it.

```
class Bank {
        int Balance;
        String name;

public:
    Void setvalue(int amount, String name,
                  String password) {
        if (Name == "clerk_Name" &&
            password == "password_clerk") {
            Balance += amount;
        }

    Void checkBalance() {
        Cout << Balance;
    }

    Void Withdraw() {
        Balance -= ₹100;
        Cout << "₹ 100 Withdrawn";
        Cout << Balance;
    }
}

int main() {
    Bank Customer1;
```

In this the name & password is of clerk. If clerk can login and update then the balance is updated.

Teacher's Signature.....

function called by clerk ↑ Name is of clerk ↑ clerk password ↑

```
Customer1.setValue (1000, "Rajesh", "******");
    Customer1.checkBalance();
    Customer1.withdrawn();
return 0;
}
```

* ANOTHER EXAMPLE :

```
Class car {
        int Mileage;
        int cost;
        public:
        Void Inc_Mileage() {
            Mileage += 2;
        }
int main() {
    car obj;
    Cout << sizeof(obj);        // return 8.
    return 0;
}

Class car {
    int Mileage;
    char cost;                  // return 8
}
```

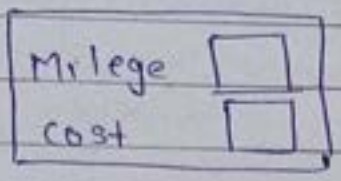
Why both gives 8 ?

int + int = 8 (Sms ayya)

int+char = 8 (How)

It happens by the padding & alignment.

Teacher's Signature.....



Default value :

↳ If we don't give any value then it take default value which is done by Constructor.

जब तक user input नहीं देता है तब तक कुछ initial value आ जाए तो तो Constructor करता है।

```
class car {
        int Cost;
        int Mileage;
public :                    → Constructure
    car(){
        Cost = 2,00,000;
        Mileage = 30;
    }
int main(){
        car obj;
}
```

The name of Constructor is same as class.

If you don't made any Constructor then Computer made a default Constructor and put random value to it.

Teacher's Signature.....

Also possible by:

```
class car {
        int Cost;
        int Milege;
    public:
Constructor → car(int price, int speed){
                Cost = price;
                Milege = speed;
                Cout << "Hello Constructor";
            };
        Void print() {
            Cout << Cost " " << Milege;
        }
}
int main(){
        car. Adi (20, 40);
        Adi.print();
    }
```

If We Write Constructor as:

```
    Car( int Cost, int Milege) {
This gives error    Cost = Cost;
because it get Confused  Milege = Milege;
                Cout << "Hello Constructo";
```

int can be Correct by:

```
this → Cost = Cost;
this → Milege = Milege;
```

Teacher's Signature.....

```
int main (){
    Student *p = new Student();
        *p.reg = 10;
         p->reg = 10;
```
} Both are same.

→
```
class car {
    public:
        int cost;
        int Milege;
};
int main (){
    car *p = new car;
     (*p).cost = 20;
      p-> milege = 25;
      Cout << p.cost;
```

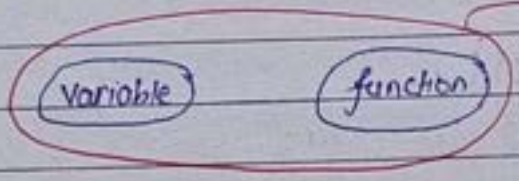
* **Destructor:** जो भी memory occupy की हुई है उसको Release कर दो। use होने के बाद।

```
class car {
    public:
        int cost;
        int Milege;
    car() {
        cout << " Constructor";
        }
    ~car(){
        Cout << " Destructor";
        }
}
```

```
int main() {
    car *p = new car;
    delete p;
    return 0;
}
```

* **Encapsulation :**

Combination of variable & function



* **Abstraction :**

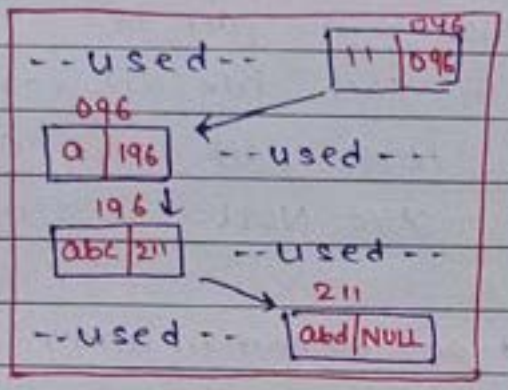
It can hide the data which are not used.
Show only that data which you need.

By this we can achieve Code Reusability.

Linked List

Memory

Address = 046



user demand for 4 space for storing.

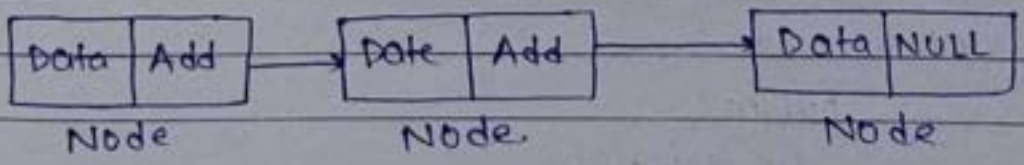
In the memory, It can't available at one place means continous.

So, what we do. We give non continous memory.

So, how he know where the next memory occur. Ether we give the address of all memory space. It is very tedious task to remember.

Then, what we do. We can discover linked list.

In linked list we can give address of 1st space and the second space address can be stored in first space.



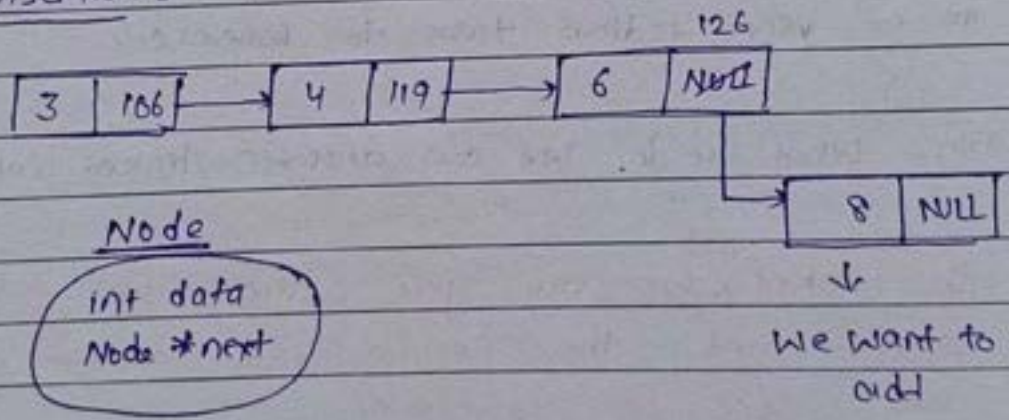
Teacher's Signature.....

You have a movie of 4 GB.
In your laptop there is space of 4GB vacant, but Not at one place.

So, we store the movie in part in different location and in each file we can write the address of Next part and at the last part we store NULL.

→ If we want to insert an element in array, then it is very difficult.
→ But in LinkedList we simply add an element and insert the index.

Insertion:



Node

```
int data
Node *next
```

Code:

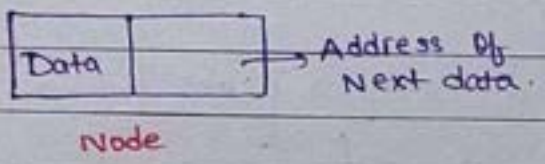
```
Class Node {
    public:
      int data;
      Node * next;
    }
```

Teacher's Signature.....

```
int main () {
    Node * first = new Node;
    first → data = 10;
    return 0;
}
```

Lecture - 31

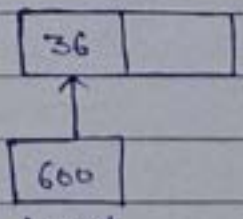
Linked List (Insertion , Delection)



Node

```
Class Node {
    public :
    int data;
    Node * next;
}
```

Head is always point to first Node.
Collection of Node is known as linked list



Head

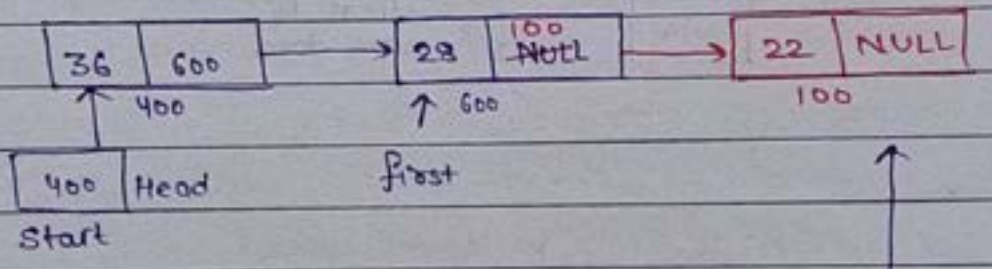
Head can store address of first Node.

Teacher's Signature .....

Page No.
Date :

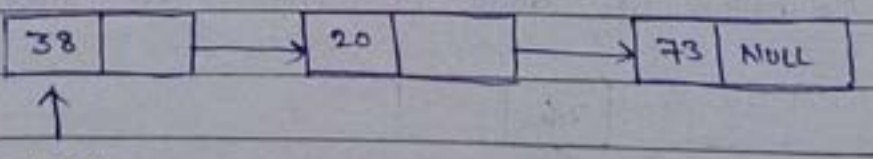
Creation of New Node :

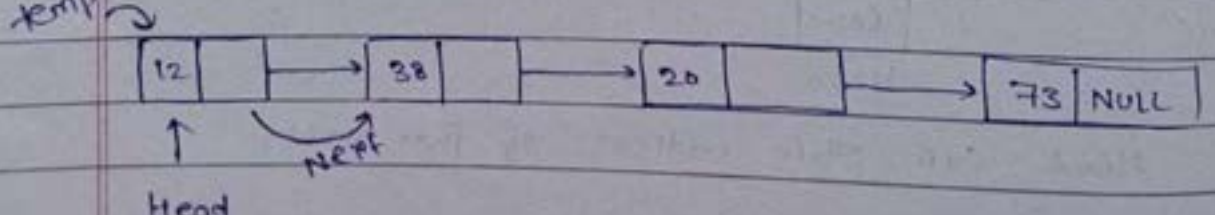
```
int main() {
// 1st Node    Node * Head = new Node();
                Head → data = 36;

// 2nd Node    Head → next = new Node();
                First = Head → next;
                First → data = 28;
```




```
// 3rd Node
                First → next = new Node();
                First = First → Next;
                First → data = 22;
}
```

* Insertion at Beginning :

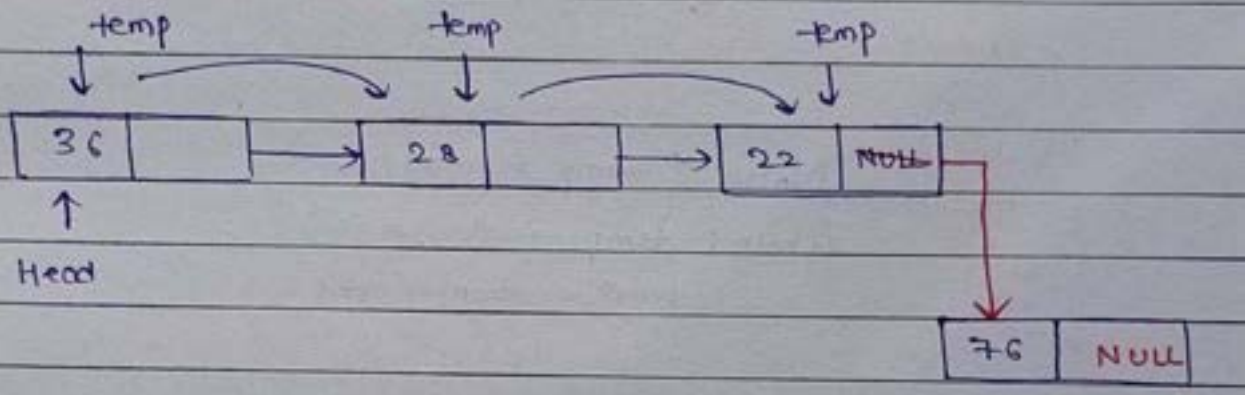




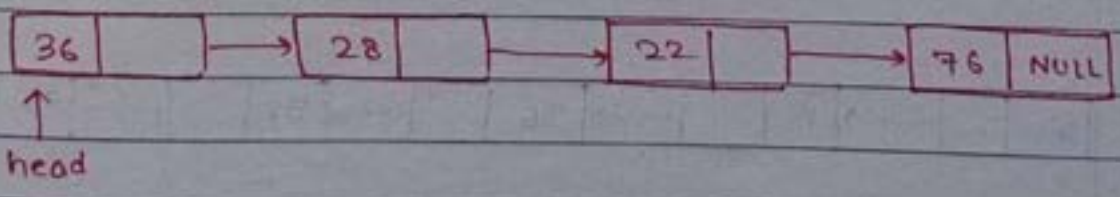
Teacher's Signature.....

```
Node * temp = new Node();
temp->data = 17;
temp->next = Head;
Head = temp;
```

* Insertion at End :

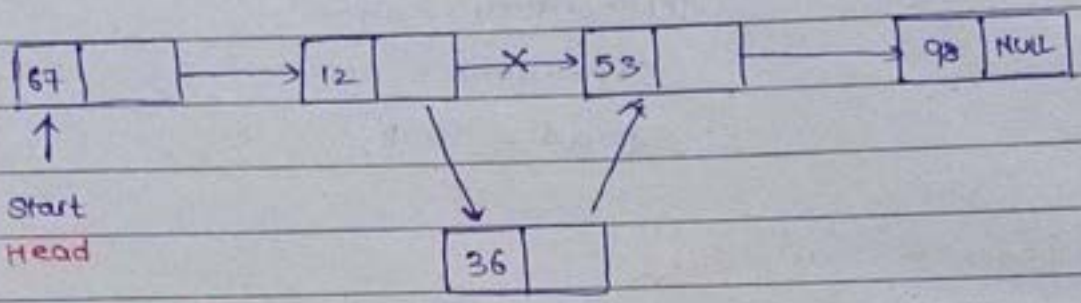



```
Node* temp = head;
While(temp -> next != NULL) {
    temp = temp->next;
}

temp->next = new Node();
temp = temp->next;
temp->data = 76;
```



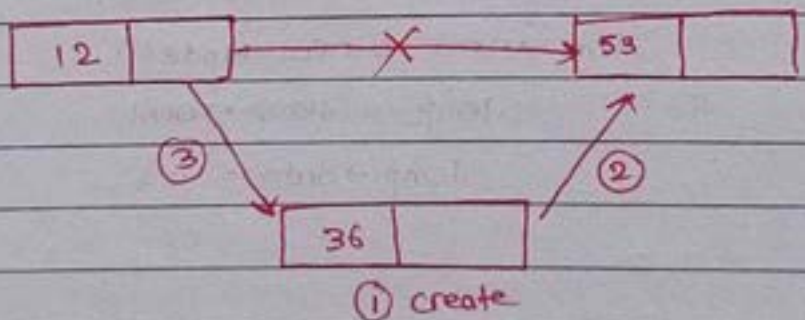
Teacher's Signature.....

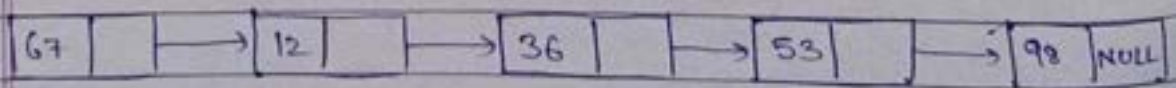
* Insertion At middle (After Node):



Insert After 12.

```
Node * temp = head;
While ( temp -> data != 12)
        temp = temp -> next;

Node* newNode = new Node();
newNode -> next = temp -> next;
temp -> next = newNode;
newNode -> data = 36
```





Teacher's Signature.....

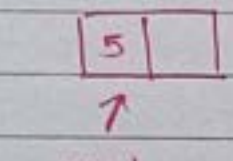
* Code for Converting array into Linked list

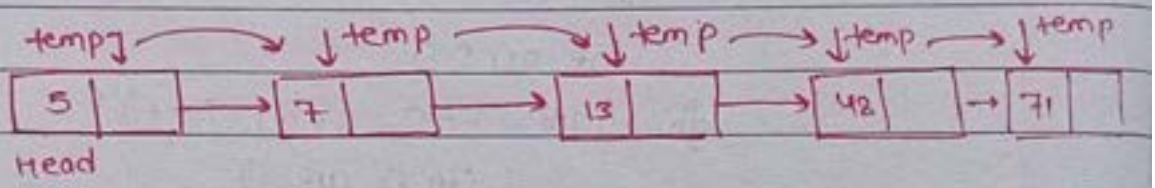
```
Class Node {
    public:
     int data;
      Node * Next;
};

int main () {

    int n;
    int arr[n];
  for ( int i=0; i<n; i++) {
        cin >> arr[i];
    }

  Node * head = new Node();
      head → data = arr[0];
       Node*temp = head;
  for (int i=1; i<n; i++ ) {
        temp → next = new Node();
            temp = temp → Next;
            temp → data = arr[i];
      }

    print (head);
  return 0;
}

Void print ( Node * head) {
      While ( head != NULL) {
```

Teacher's Signature.....

```
        cout << head -> data << " ";
        head = head -> next;
    }
}
```

Arr :- 5 7 13 48 71

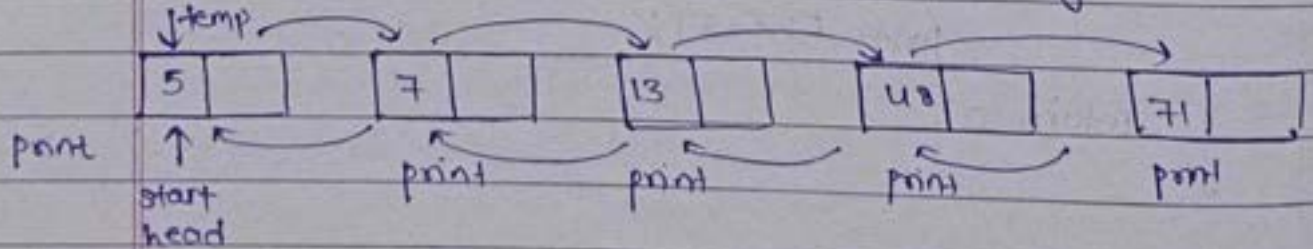




Time Complexity : O(N)

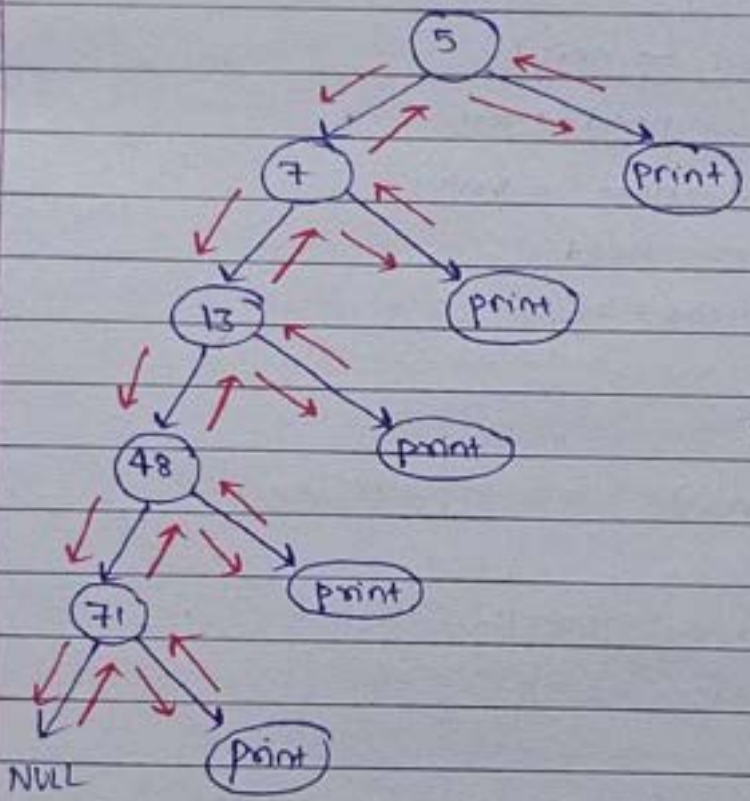
* Reverse of a Linkedlist
- ↳ By recursion
- ↳ By Loop

→ By recursion
- ↳ traverse without printing to the last
- ↳ then back to front with printing.



Teacher's Signature.....

```
reverse (Node * head) {
    if (head == NULL)
        return;
    reverse (head → next);
    Cout << head → data;
}
```

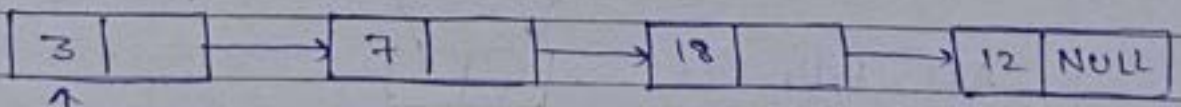


Time Complexity : O(N)

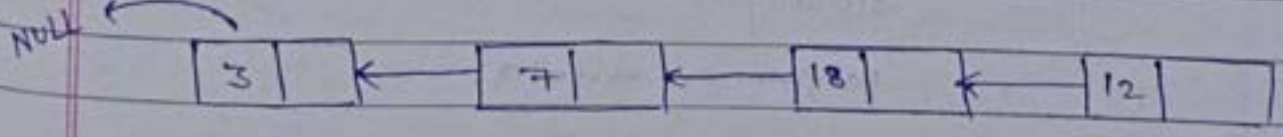
Space Complexity : O(N)

recursion → stack.

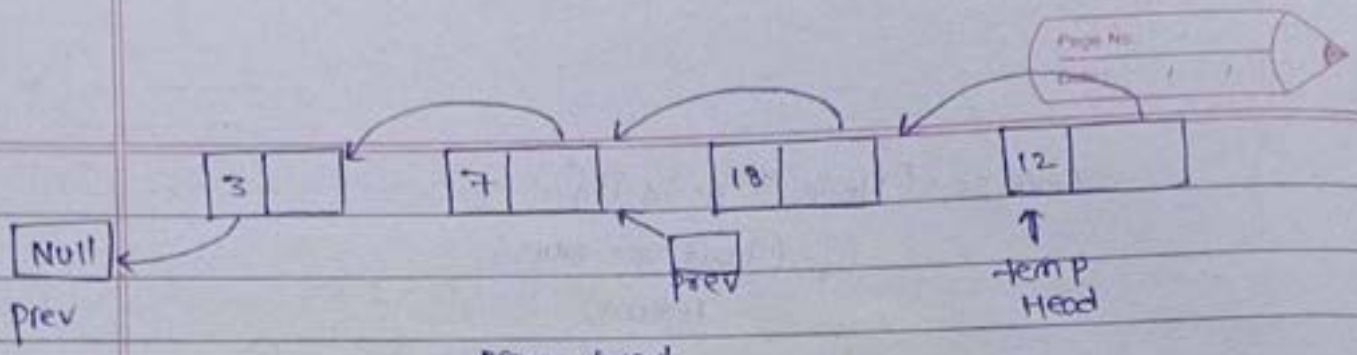
→ By loop :



For reverse :



Teacher's Signature

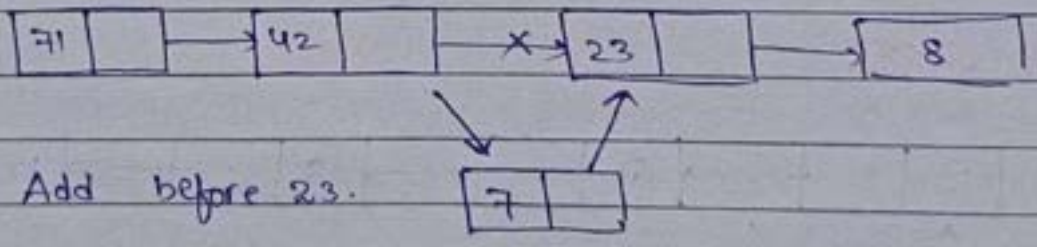


prev = head

```
Node * Prev = NULL;

While ( Head → next) {
    temp = Head → next;
    Head → next = prev;
    prev = Head;
    Head = temp;
}

print (prev);
```

Time Complexity : O(N)
Space Complexity : O(1)

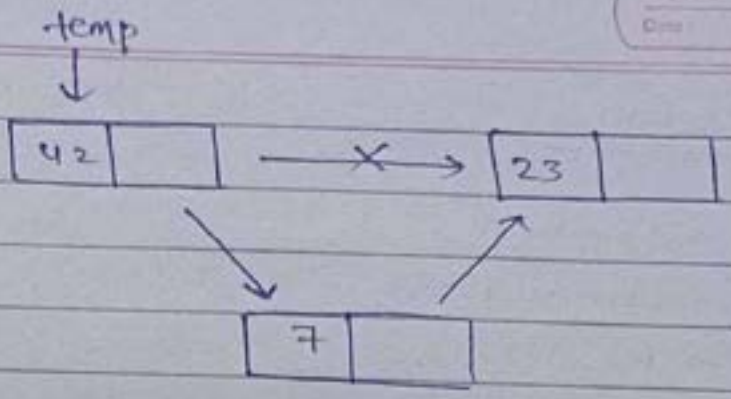
* Insert in Middle (Before the element):



temp → 23 के एक पहले तक आएगा।

temp → next → data.

Teacher's Signature.....



Head → next → Data.

While (temp → next → data != 23)
temp = temp → next;

Now, insert the value after temp.

Condition : If 23 is first element

if (Head → data == 23) {
Add Node at first
// Write Code.
}

Teacher's Signature.....

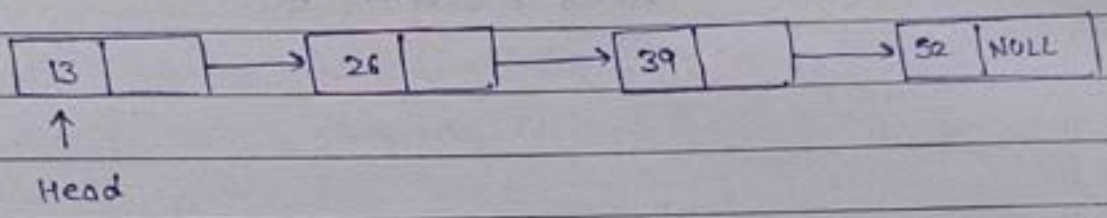
* Delection :

→ 1st Node

→ Last Node

→ Middle Node

1st Element :

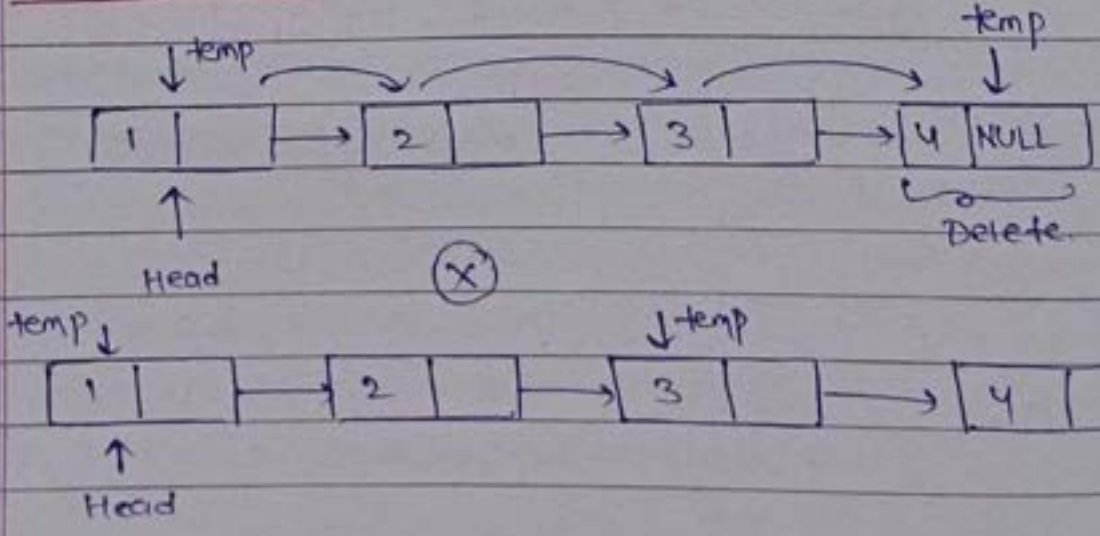


For Delection 1st Element

① Head = Head → next;

② first = Head;
Head = Head → next;
delete (first);

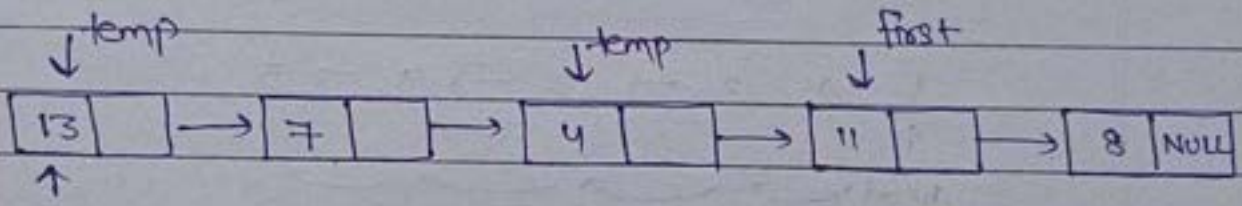
Last Element :



Teacher's Signature.....

```
delete (temp → next)
temp → next = NULL;

While (temp → next → next) {
    temp = temp → next;
}  temp → next = NULL;
delete (temp → next);
temp → next = NULL;
```

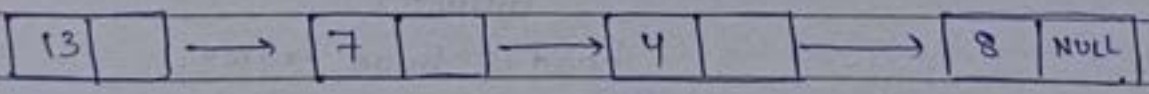
→ Delete the Middle Element :



delete After 4

first

```
first = temp → next;
temp → next = temp → next → next;
delete (first);
```



Teacher's Signature.....

* Count Nodes of Linked list :

A Single linked list

Count Number of Nodes

(1) 1 → 2 → 3 → 4 → 5

traverse the linked list till last
and Count++.

1 → 2 → 3 → 4 → 5
↑
head → → → → →

Count = 1+1 = 2+1 = 3+1 = 4+1 = 5

While (head)
head जब तक NULL नहीं होता।

```
int getCount ( Struct Node * head){
        int Count = 0;
        While (head){
            Count++;
            head = head → next;
        }
    return Count;
}
```

Teacher's Signature.....

* Insert in a Sorted List :

Given a linked list :
in sorted order.

You have given a number. Insert the number at the space where it can be fit.

⇒ 25 → 36 → 47 → 58 → 69 → 80

(19) → Insert at first.

⇒ 50 → 100

(75) → Insert in Middle.

We have to write code for insertion for all the case.

```
class solution {
    public :
    Node * sortedInsert ( struct Node *head, int data) {
            if (head → data > data){
            Struct Node * temp = New Node(data);
                temp → next = head ;
                head = temp;
                return head;
            }
```

Teacher's Signature.....

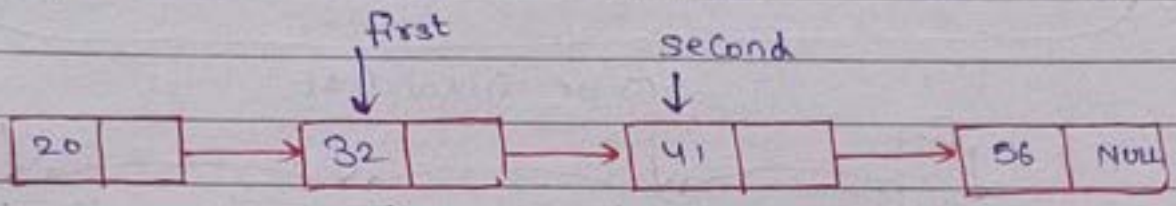
```
struct Node * first = head;

While( first -> next && first -> next -> data < data){
        first = first -> next;
    }
    Struct Node * temp = new Node (data);
        temp -> next = first -> next;
        first -> next = temp;
        return head;
    }
};
```

Teacher's Signature.....

Lecture - 32

(Linked list {Problems})

If you want to delete a Node in linked list.



Delete 41

Approach 1 :

point the Node of 32 to direct 56
41 is not linked to anyone.
so, it can't be shown.

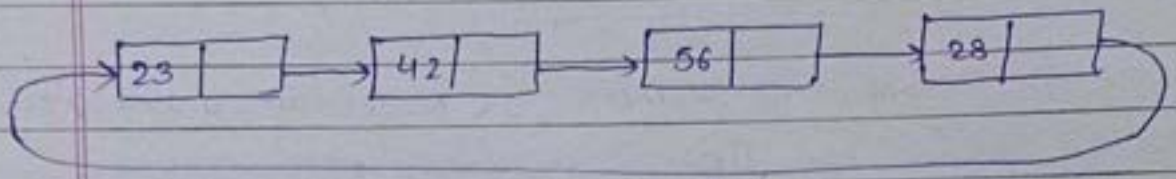
first → next = first → next → next

Approach 2 :

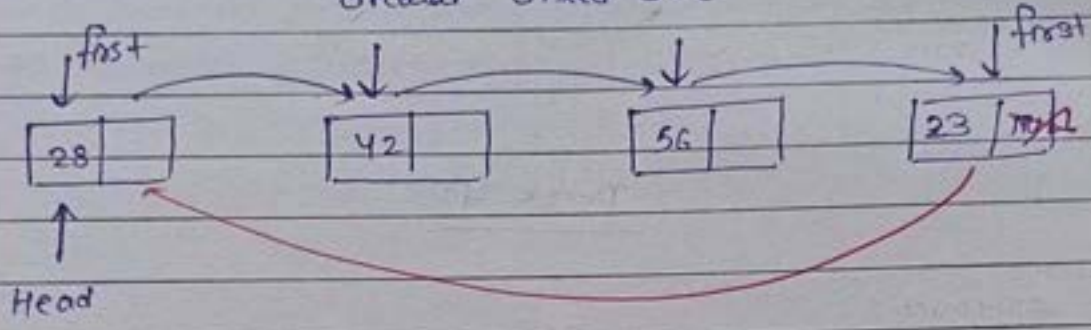
put a second pointer on the node which we want to delete. and after we do.
delete (second);

Teacher's Signature.....

Circular Linked List



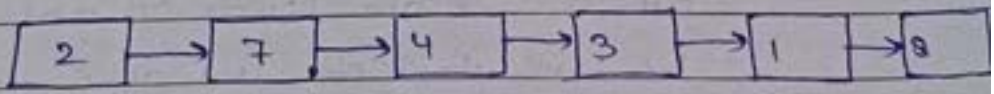
circular linked list




```
Node * first = Head
While( first → next){
    first = first → next
}
first → next = Head;
```

Array :

0	1	2	3	4	5
2	7	4	3	1	8



Accessing 5th element in Array :

O(1) arr[4] = 1

In linked list

O(N) Manually Checking one - one element

Teacher's Signature.....

But Insertion and deletion can be easy in linkedlist.

Array : T.C : O(N) At start

Linked list : T.C : (O(1)) At start

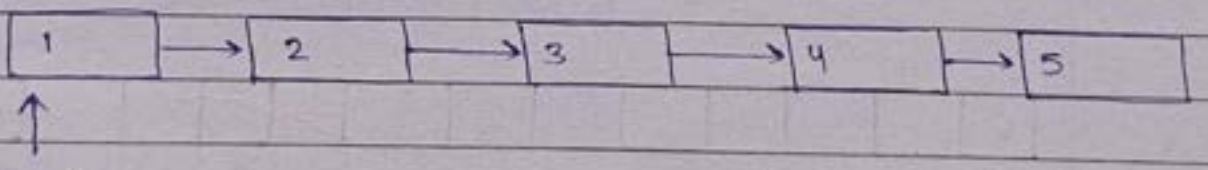
Searching:

Array (if sorted): O(log n)
↳ Binary search

LinkedList (if sorted): O(n)

problem 1: Check if Circular Linked list

Given, a linked list with head.



Head.

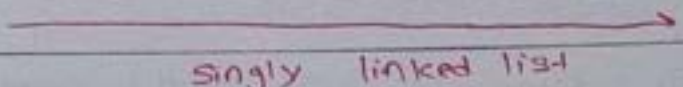
Approach 1:

If NULL present in linkedlist, then the linked list is not circular.

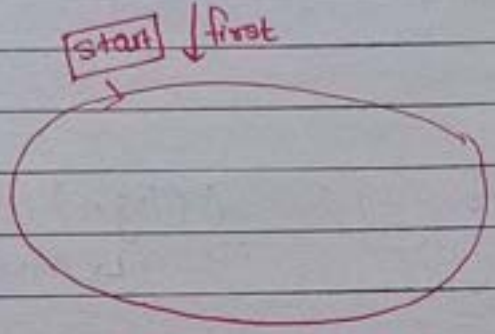
Traverse the whole array.

Teacher's Signature.....

But here is a problem, if linked list is very long then it create problem.

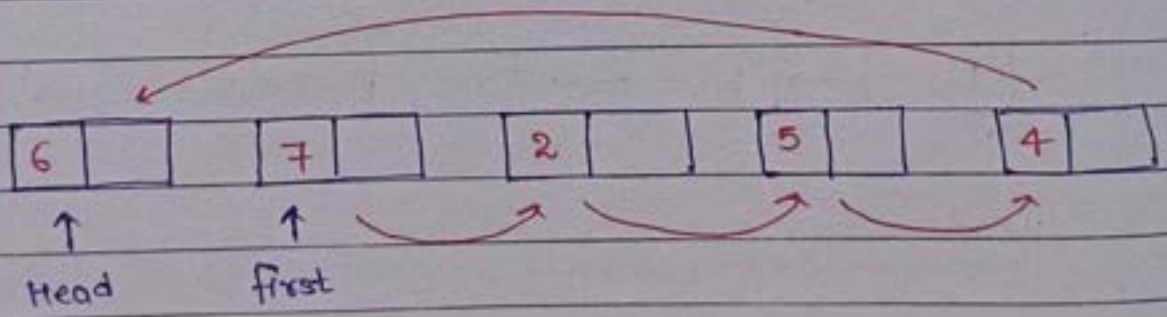


Singly linked list



Circular linked list

Fix the start at first Node.
Make an another pointer called 'first' and traverse, if start found
then it is circular.



First can find either NULL or Head.
If NULL
then singly
If Head
then circularly.

Teacher's Signature.....

first pointer को घुमाते घुमाते या तो NULL मिलेगा या start/Head मिलेगा।

अगर NULL मिला तो
↳ circular नहीं होगा।

अगर start मिल गया तो
↳ circular हो जाएगा

Code :

```
bool isCircular (Node * head) {
    Node * first = head;
    while (first → next && first → next != head) {
        first = first → next;
    }
    if (first → next == NULL)
        return 0;
    else
        return 1;
```

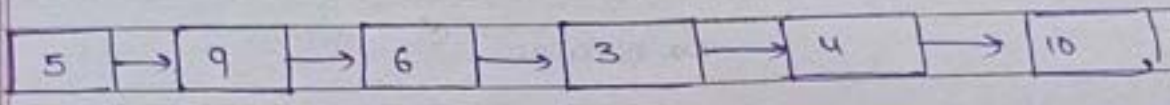
Teacher's Signature.....

Problem-8

Find the Sum of Last N nodes of Linked list :

A Single linked list of size M,

Return the sum of last n nodes.



Total Size = M = 6

n = 3

sum of last 3

We have to neglect some Nodes from starts.

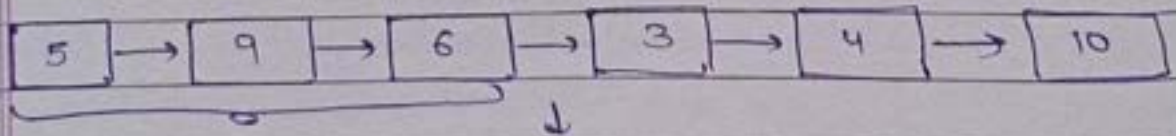
Total No. of Neglected Nodes, a

= M - n

= 6 - 3

= (3)

↳ We have to neglect 3 nodes.



Neglect a = 3 Nodes

Sum = 0

0 + 3 + 4 + 10

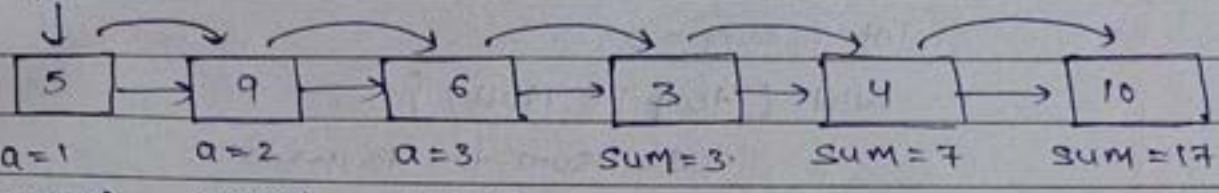
= 17

return 17.

Teacher's Signature.....

put a pointer on first Node. First traverse 'a' Nodes Without adding element.

Then sum all the element after a = 3 Nodes.



step-1 : Count total Number of Nodes

step 2 : Find the no. of Node for Neglecting.

step-3 : A temp pointer can neglect first some nodes while traversing.

step 4 : The start Adding the element.

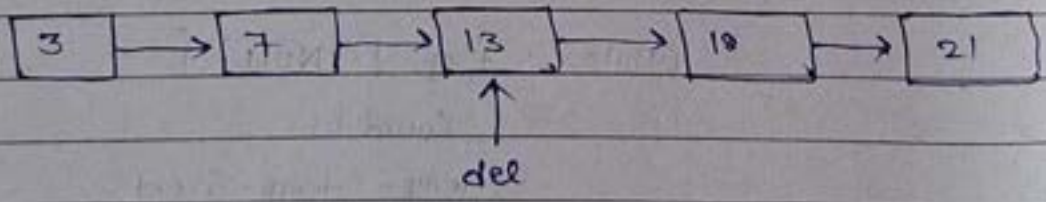
Code :

```
int SumofLastN_Nodes ( struct Node* head, int N) {
        int Count = 0;
        Node* temp = head;

        While ( temp != NULL) {
                Count++;
                temp = temp->next;
        }

int neglect = Count - N;

temp = head;
```

```
While ( neglect > 0) {
        temp = temp → next;
        Neglect - - ;
    }

int sum = 0;
While ( temp != NULL) {
        sum += temp → data;
        temp = temp → next;
    }

return sum;
}
```

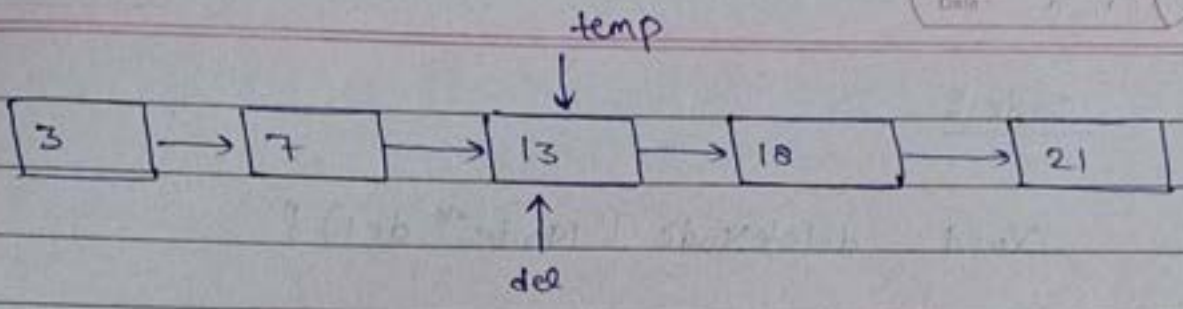
* Delete Without head pointer :

A linked list
↳ you only have given a pointer of node which you have to delete.



You have give a del pointer pointing to a node.
Delete that node.

Teacher's Signature.....

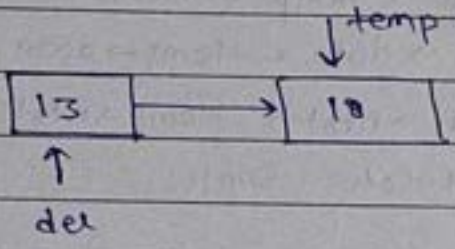


Make a pointer temp at same Node of del.

```
Node * temp = del;
```

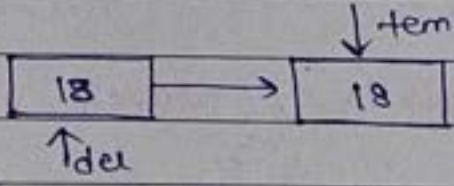
Move one step the temp pointer.

```
temp = temp -> next;
```

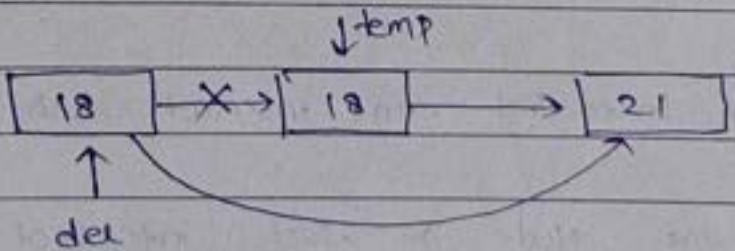


Copy the data of temp in del

```
del -> data = temp -> data
```



del में temp के Next Node1 का pointer / address point करो।




```
del -> next = temp -> next.
```

At last delete the temp.

```
delete temp;
```

Teacher's Signature.....

Code :

```
Void deleteNode (Node* del) {

    if (del == NULL || del -> next == NULL)
        return;

    ~~del * temp~~
    Node * temp = del;
        temp = temp -> next;
        del -> data = temp -> data;
        del -> next = temp -> next;
        delete temp;
    }
};
```

* Problem - 4

Intersection point in Y shaped linked list.

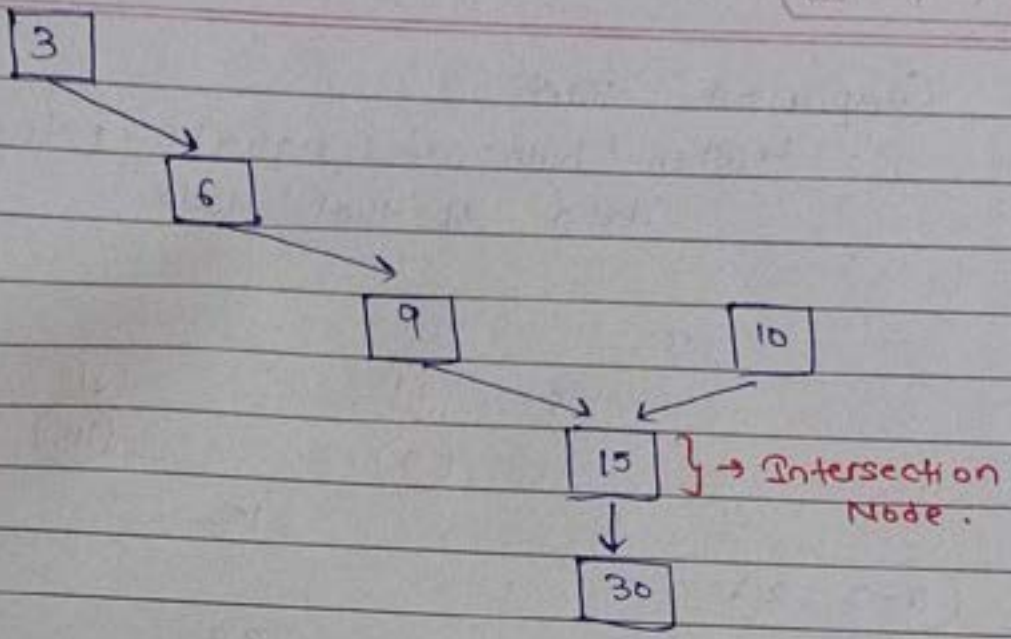
Given : Two single linked list
Both size are different

Both linked list can intersect each other

Find the Node at which intersect.

Two linkedlist :
LinkedList1 : Size → N
LinkedList2 : Size → M

Teacher's Signature.....



How to find:

Check both linked list.

↳ For this the size should be same.

Make the size equal

Bigger linked list ko small kareneg.

Size 1 = 5

Size 2 = 3

5 - 3 = (2)

↓

LinkedList 1 2 Node ke baad ek pointer lega aur LinkedList 2 first Node pe ek pointer dega.

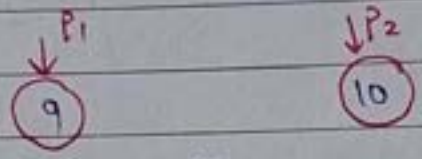
Teacher's Signature.....

Comparison start

↳ When both are equal return the data of that node.

3

6



15

5-3=2

30

9 10

P1 != P2 (True)

P1 = P1 → next;

P2 = P2 → next;

15 15

P1 == P2 (True)

return P1 → data

(15)

Code:

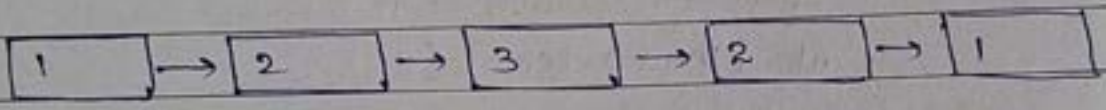
```
int intersectpoint ( Node * head1, Node * head2) {

// step1 : calculate the length of Both linked list
    int len1 = 0, len2 = 0;
    Node * temp1 = head1;
    Node * temp2 = head2;
```

Teacher's Signature.....

```
While ( temp1 != NULL) {
        len1++;
    temp1= temp1→next;
}

While ( temp2 != NULL) {
        len2++;
    temp2 = temp2→next;
}

// step 2 : Make longer linked list smaller as same
//          size to smaller one.
if (len1 > len2) {
    int diff = len1 - len2;
    While ( diff > 0) {
        head1 = head1→next;
        diff--;
    }
else {
        int diff = len2 - len1;
        While (diff > 0) {
            head2 = head2→next;
            diff--;
        }
}

// step 3 : start Comparing both linked list
//    if Matched return data otherwise -1.
```

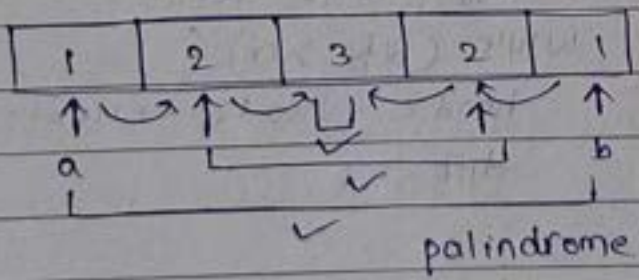
```
While ( head1 != NULL && head2 != NULL) {
        if (head1 == head2) {
            return head1 -> data;
        }
        head1 = head1 -> next;
        head2 = head2 -> next;
    }
    return -1;
}
```

* Check if Linked list is palindrome :



Approach 1 :

Linked list को Traverse कर के Array में store कर लो उस के बाद palindrome check कर लो।



T.C : O(n)

S.C : O(n)

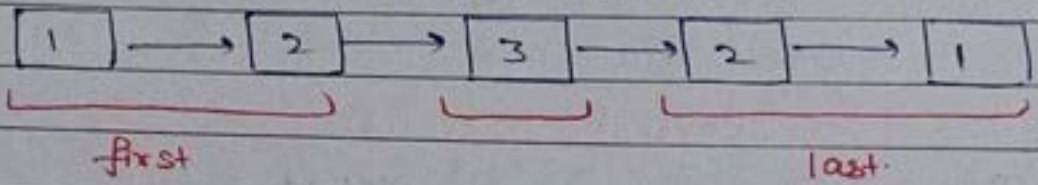
↳ But We want O(1).

Teacher's Signature.....

Approach 2:

Size of linked list is
either odd or even.

If odd:



Size = 5

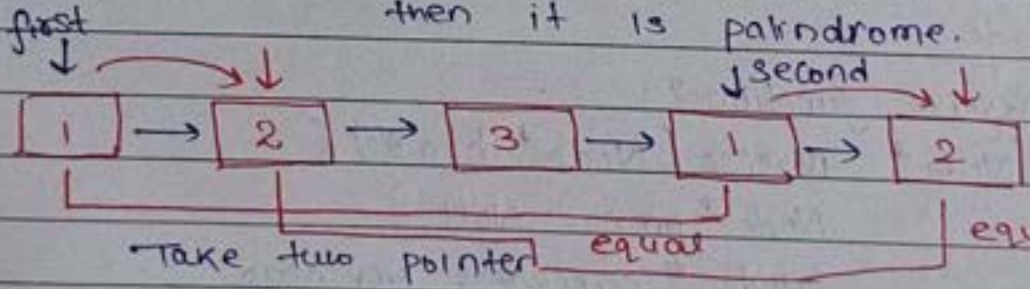
Divide in two parts.

n = 5/2 = 2

2 from first & 2 for last

Reverse the last part

If Fast is equal to second
then it is palindrome.



first on Head

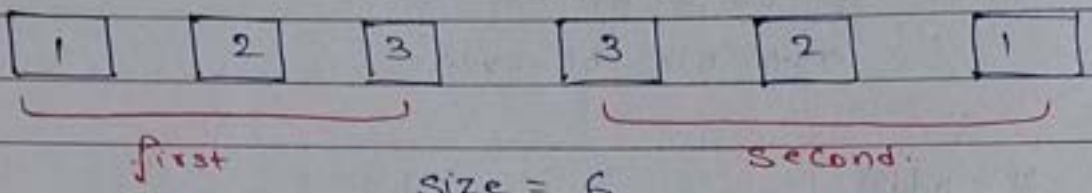
second on size - n

5 - 2 = 3 + 1 = 4

palindrome.

Teacher's Signature

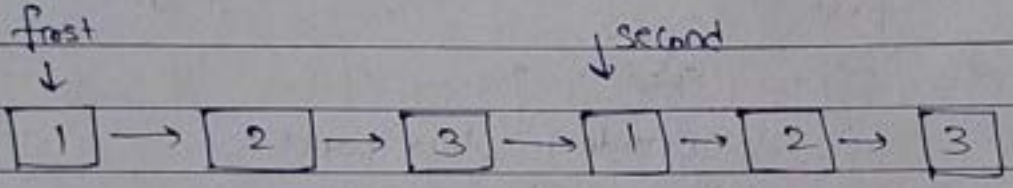
If size is even.



size = 6

n = 6/2 = 3

reverse second.



2nd pointer on → Size - n + 1

Traverse and check.

Code:

Reverse function:

```
Node * reverse (Node * head) {
    Node * prev = NULL;
    Node * current = ~~prev~~ head;
    Node * next = NULL;

    While ( current != NULL) {
        next = current → next;
        current → next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
```

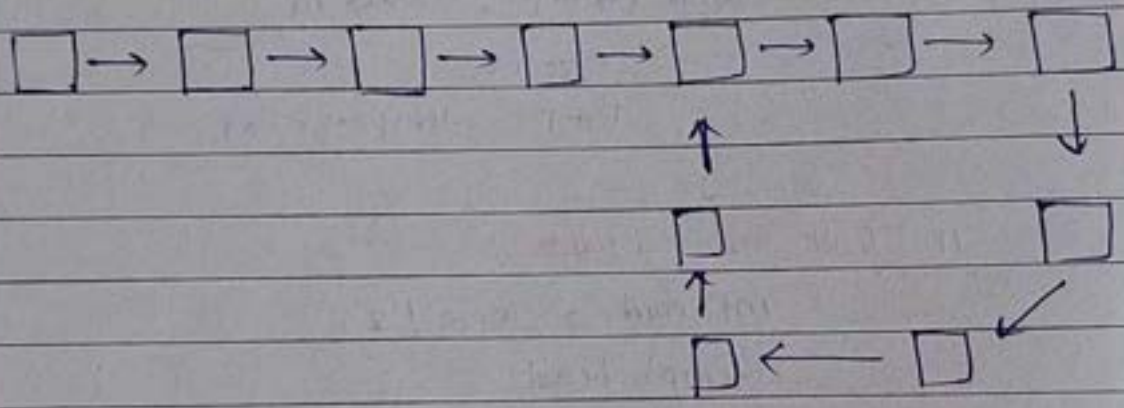
Teacher's Signature.....

```
bool ispalindrome (Node* head) {

if (head = null || head -> next == NULL)
        return true;


// size of the linked list
        int size = 0;
        Node * temp = head;
        While (temp != NULL) {
            size++;
            temp = temp -> next;
        }
// find the Midpoint
        int mid = size / 2;
        temp = head;
// Reverse the second half of linked list
        ~~Node * head~~
        Node * prev = NULL;
    for (int i = 0; i < mid; i++) {
            prev = temp;
            temp = temp -> next;
        }
    prev -> next = reverse (temp);

// Comparing the both part of linked list.
        temp = head;
    Node* second = prev -> next;
for (int i = 0; i < mid; i++) {
    if (temp -> data != second -> data)
        return false
```

Teacher's Signature

```
            temp = temp → next;
            second = second → next;
        }
        return true;
    }
};
```

* Check that loop present or Not



How to solve:

Two Condition
- ↳ loop present
- ↳ loop Not present.

How to Approach:

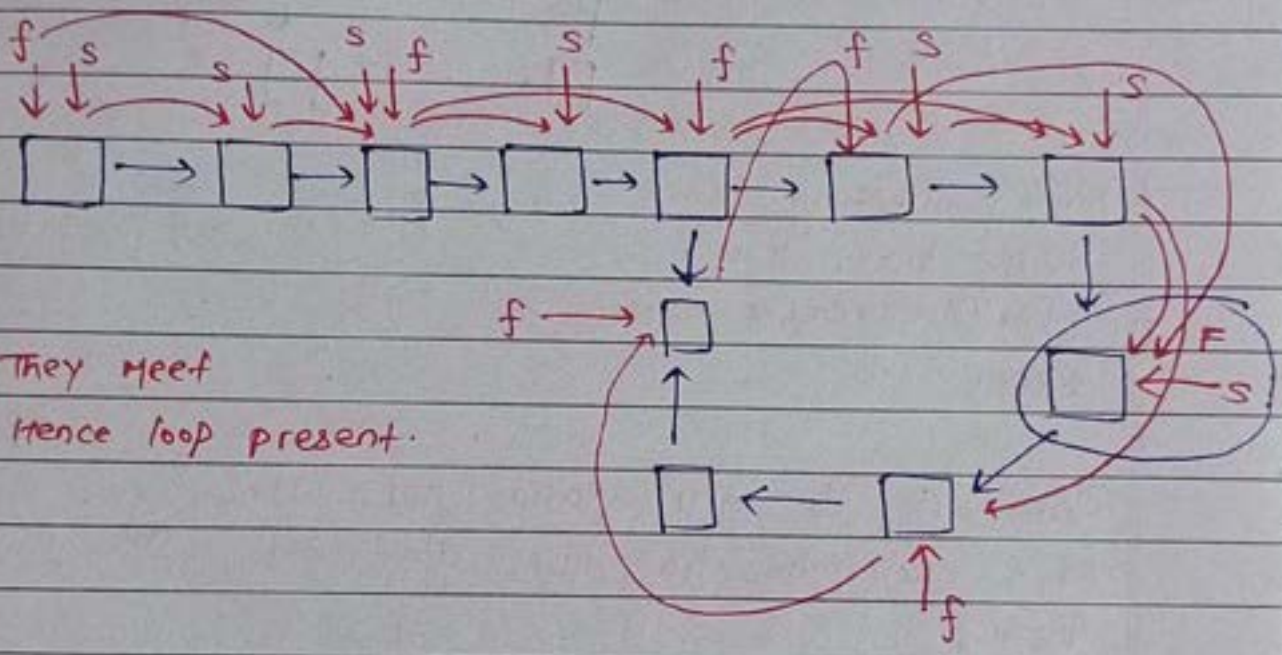
Take two pointer at first (Head) Node.

1st pointer can traverse 2 node at a time
2nd pointer can traverse 1 node at a time.

Teacher's Signature.....

If In the path, they meet NULL
↳ the linked list is Not having any loop.

If both pointer meet again excet NULL.
↳ the linked list in having loop.



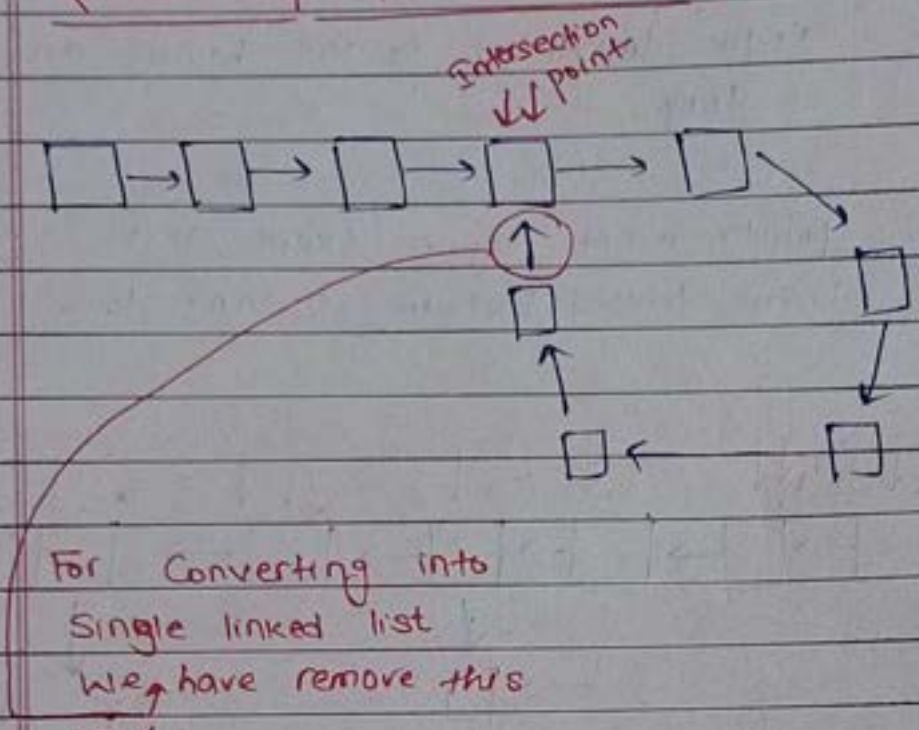
They Meet
Hence loop present.

They Meet if NULL Not present.

They either Meet after n number of round

Take Example of Race in Real life.

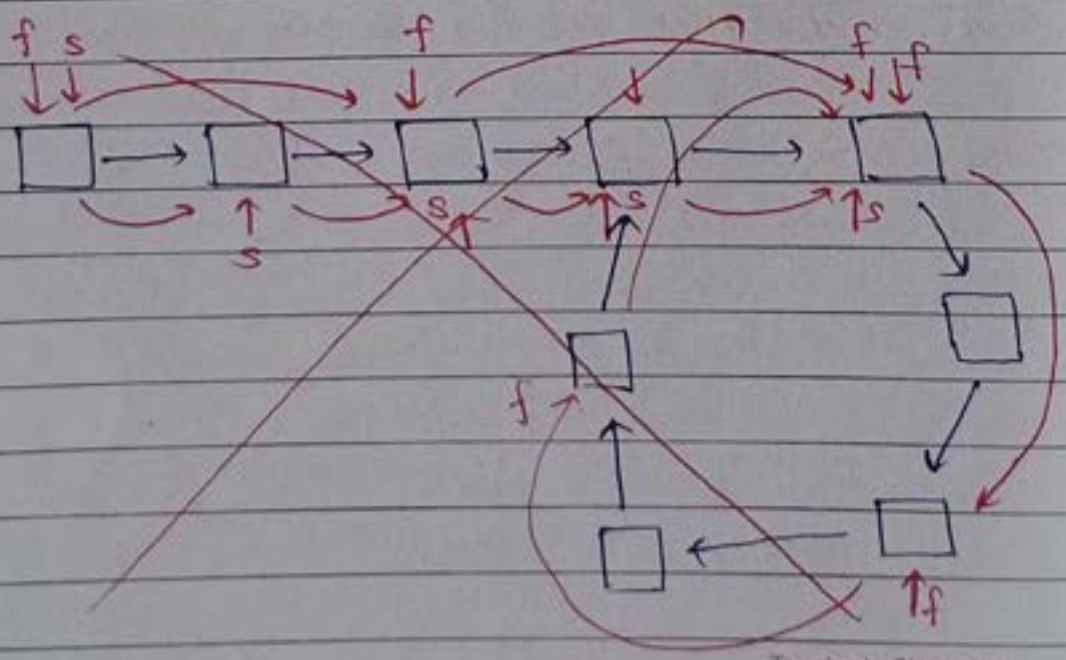
* Remove loop in linked list :



For Converting into Single linked list we have remove this Node.

Jaha pe do Mill Raha hai Wahi se loop suru ho rha hai तो उसी को तोड़ना है।

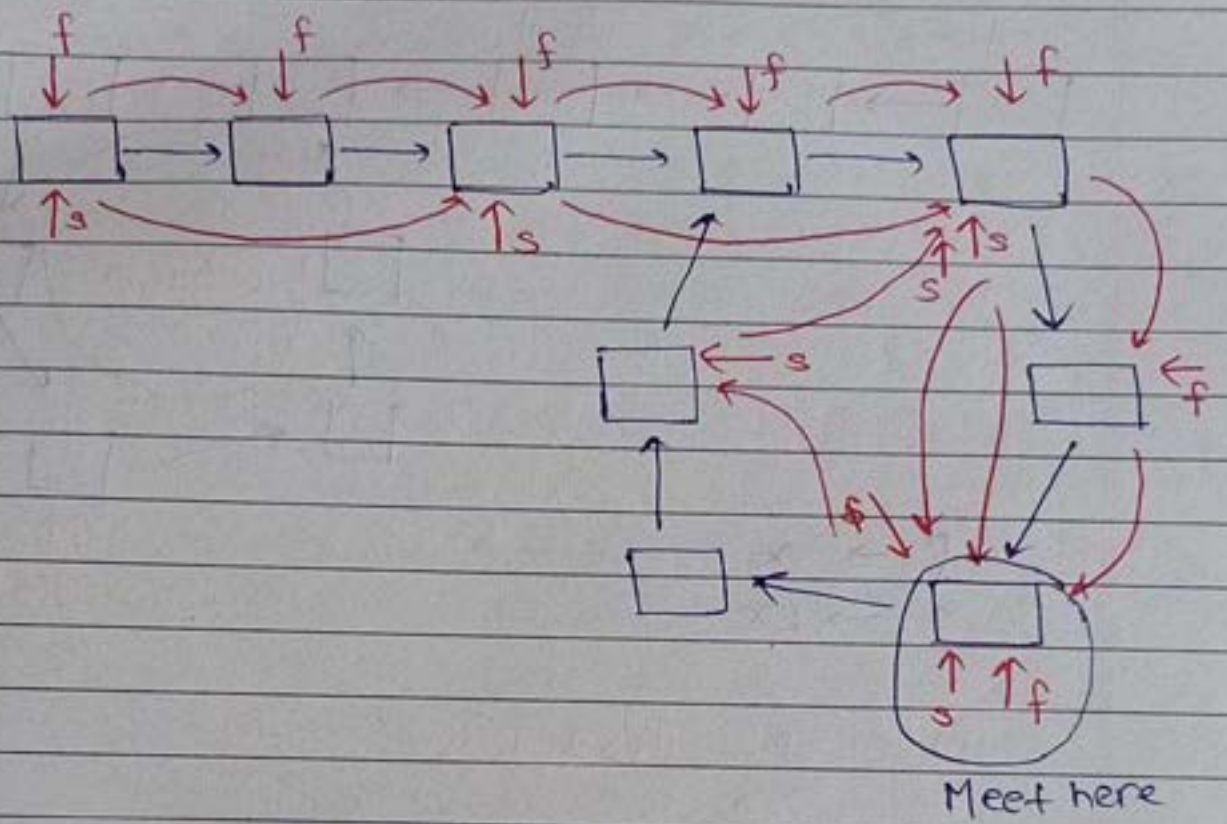
हमें ये कैसे पता चलेगा कि हमेशा दोनो उसी Node पे मिलेगा।



Teacher's Signature.....

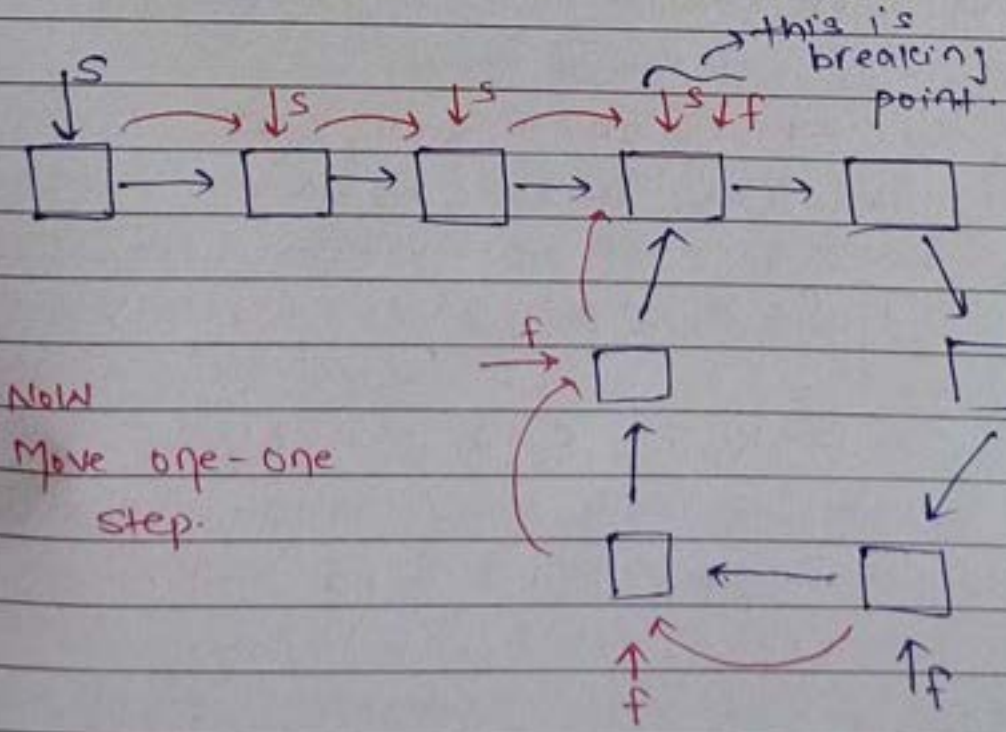
पहले First → 1 step

फिर Second → 2 step



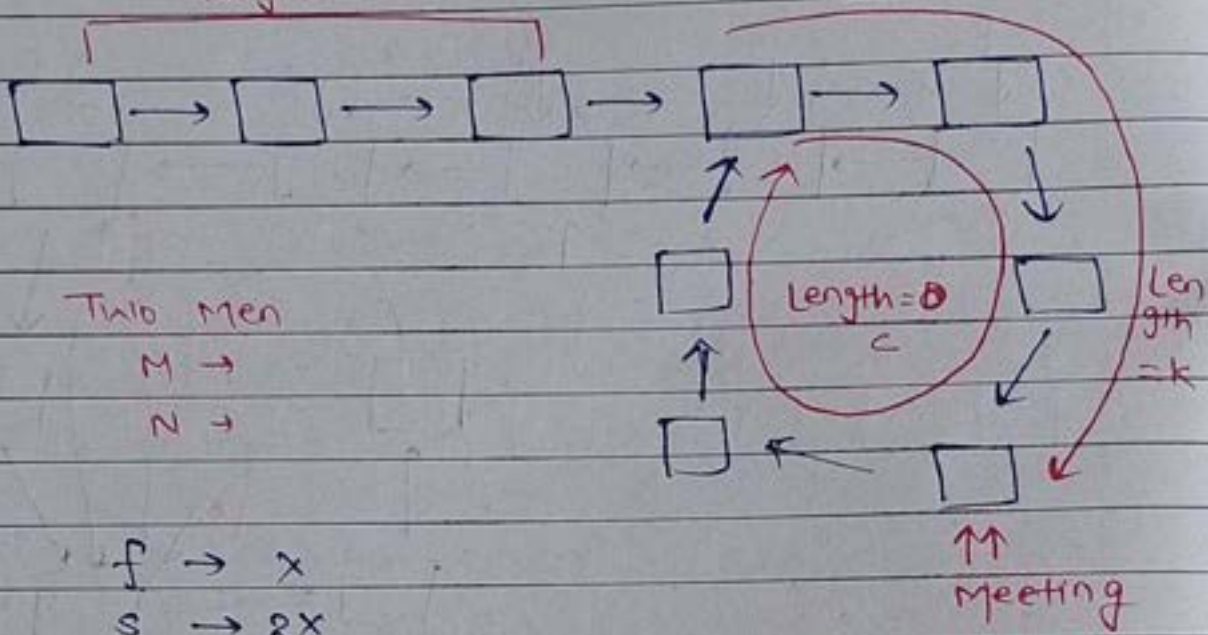
Send Second to head Node

first to at place.



Teacher's Signature.....

अब हम समझेंगें ये हुआ कैसे
ये एक Mathematics proof है।



Two Men
M →
N →

f → x
s → 2x

$N = D + C \times i + K$ — ①

$M = 2N = D + C \times j + K$ — ②

→ Multiply by 2 → ①

$$2N = 2D + 2C \times i + 2K$$
$$-\ 2N = D + C \times j + K$$
$$0 = D + 2C \times i - C \times j + K$$

$$D + K = C \times j - 2C \times i$$

$$D = C(j - 2i) - K.$$

Teacher's Signature.....

Code:

```
Node *ptr1 = head;
Node *ptr2 = head;

While ( ptr2 != NULL && ptr2 -> next != NULL){
        ptr1 = ptr1 -> next;
        ptr2 = ptr2 -> next -> next;

    if ( ptr1 == ptr2){
        unsigned int Count = 1;
        While (ptr1 -> next != ptr2) {
            Count++;
            ptr1 = ptr1 -> next;
        }
        Ptr1 = head;
        Ptr2 = head;

for ( int i = 1;  i <= Count ;  i++ ){
        ptr2 = ptr2 -> next;
    }
    While( ptr1 = ptr2) {
        ptr1 = ptr1 -> next;
        ptr2 = ptr2 -> next;
    }

While ( ptr2 -> next != ptr1 ) {
        ptr2 = ptr2 -> next;
    }
    ptr2 -> next = NULL;
}
}
```

Teacher's Signature.....

Lecture - 33

Linked list Advance

* **Code : (Remove loop in Linked list)**

O(N)

```
if ( head == head -> next)
    head -> next = NULL;

if ( ! head -> next)
    return

Node * slow = head -> next;
Node * fast = head -> next -> next;

While ( Fast && Fast -> next && Fast != slow) {
        Fast = Faot -> next -> next;
        slow = slow -> next;
};

if ( !Fast || !Fast -> next)
    return

Fast = head;

if ( fast == slow) {
    While( slow -> next != Fast)
        slow = slow -> next;
        slow -> next = NULL;
        return ;
}

While( Faot -> next != slow -> next ){
        fast = fast -> next;
        Slow = slow -> next;
}
```

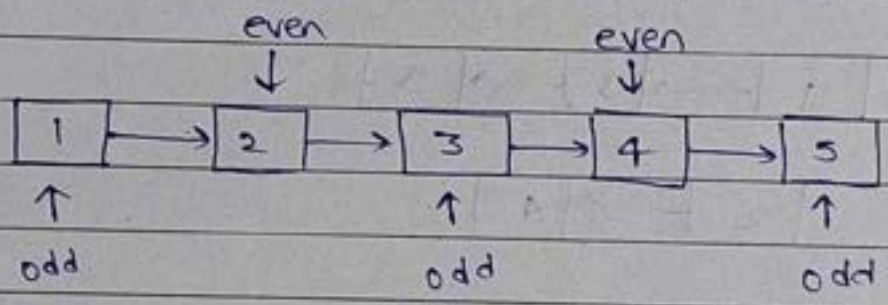
Teacher's Signature.....

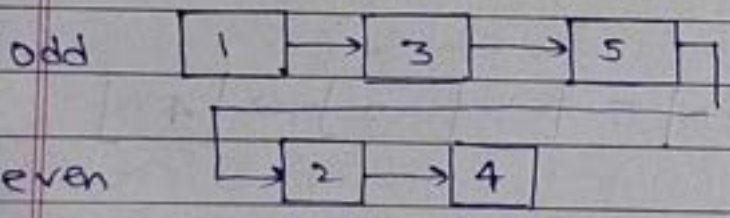
```
slow → next = NULL;
return;
}
```

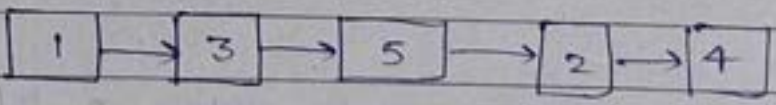
* **Rearange Linkedlist :**

Given : A single linked list

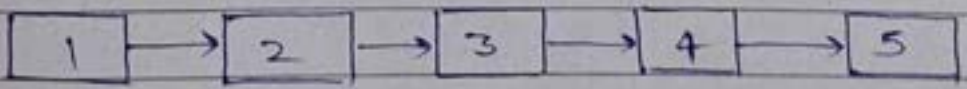
Rearrange like : odd position Node Comes first then Even position Node Comes



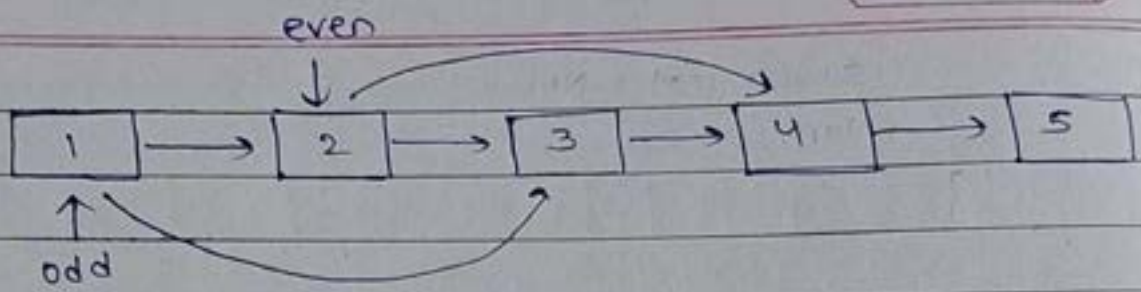


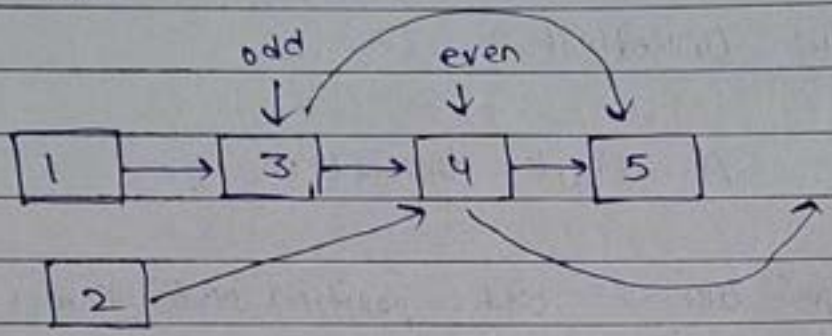
Final Answer: 

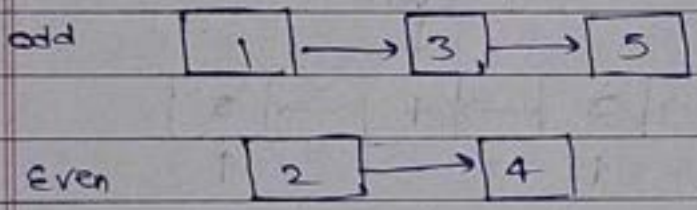
How to solve this logically.



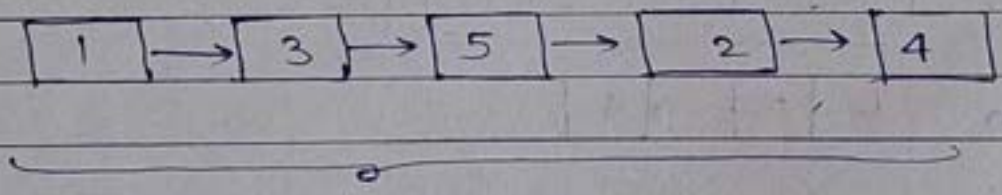
Teacher's Signature.....







→ Join odd & even



output.

Time Complexity : O(N)
Space Complexity : O(1)

Teacher's Signature.....

Code :

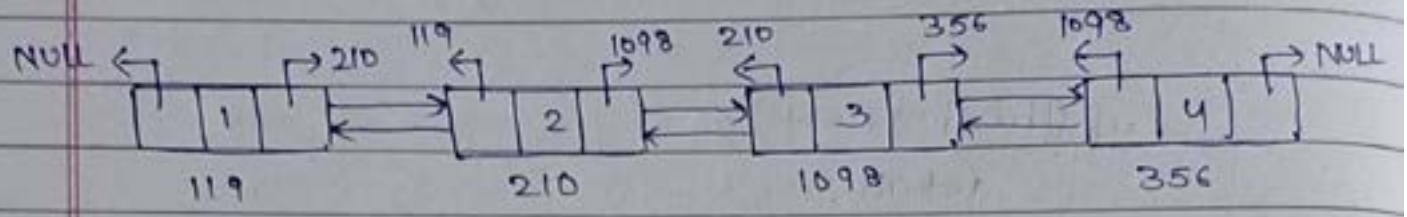
```
if (!head -> next)
    return;

Node * first = head;
Node * second = head -> next;
Node * temp = head -> next;

While ( second && second -> next ) {
        first -> next = second -> next;
        first = first -> next;
        second -> next = first -> next;
        second = second -> next;
    };

first -> next = temp;
```

* Till Now, We can only access the Node which is present forward

If we want to access the Node which is present backward, we have to use circular linked list. But it takes O(N) time Complexity.

If we want to access in O(1) time Complexity then we use the doubly linked list.

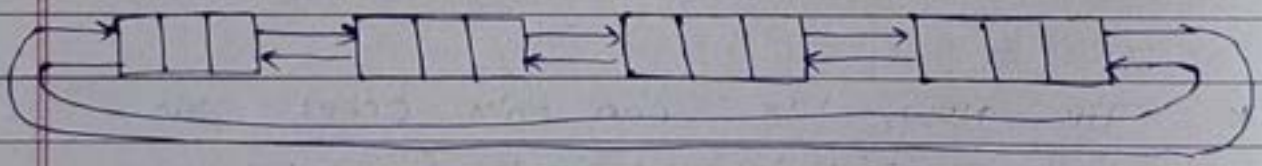
Teacher's Signature



on this we can access previous element in O(1) time.

```
class Node {
    public:
     int data;
     Node * next;
      Node * prev;
    }
```

* Circularly Doubly linked list :



* Convert the array int doubly linked list

arr = [1, 2, 3, 4, 5]

Teacher's Signature.....

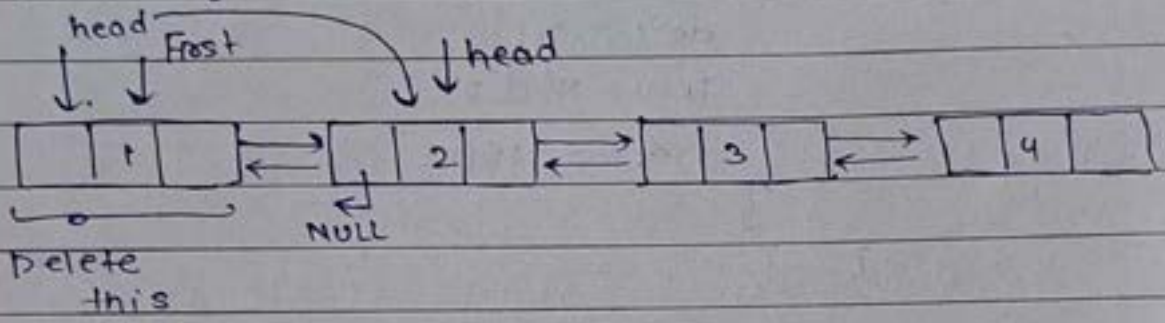
```
class Node {
    public:
    int data;
    Node * prev;
     Node* next;
};

Node (int x) {
    data = x;
    prev = NULL;
    next = NULL;
  }
}

int main () {

    int arr [5] = {1, 2, 3, 4, 5};
     Node * head;
    head = new Node (arr [0]);
     Node * first = head;

for(int i = 1;  i<5;  i++){
  first → next = new Node (arr [i]);
     first → next → prev = first;
        first = first → next;
  };
   first = head;
     while ( first) {
          cout << first → data << " ";
            first = first → next;
        }
    return 0;
}
```

Teacher's Signature

* Deletion in doubly linked list :

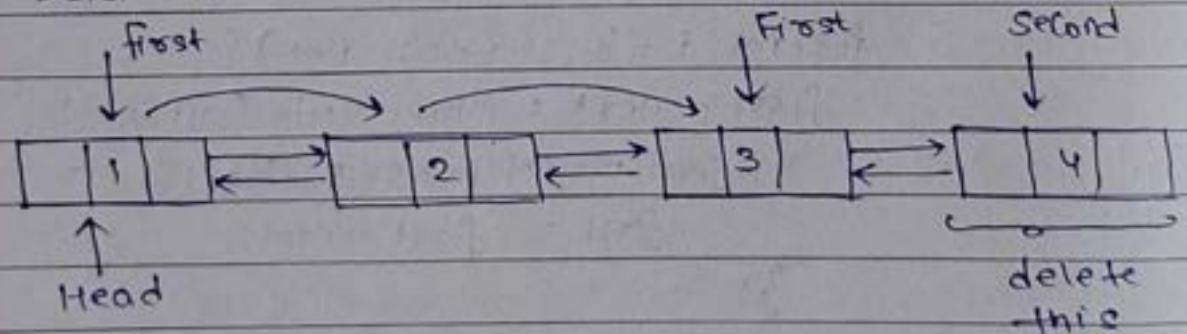
1. Delete first Node
2. Delete last Node
3. Delete Mid Node

* Delete first Node :




```
Node* first = head,
head = head→next;
head → prev = NULL;
Delete (first);
```

* Delete last Node :




```
Node * first = Head,
While (Frost → next) {
    first = first → next;
}
Node * second = first → next;
```

Teacher's Signature

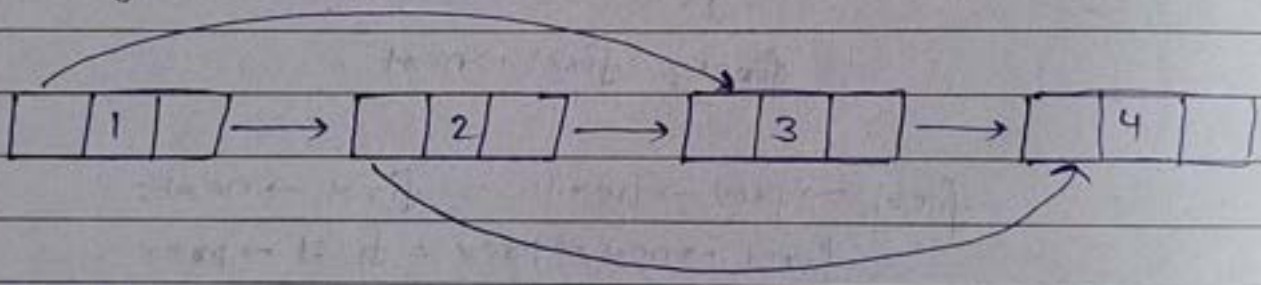
Lecture - 34

(Linked List - Hard)

* problem - 1

Clone a linked list With next and Random pointer :

→ The linked list is a type of doubly linked list.
→ One pointer is pointing to next Node.
→ Second pointer is pointer to any random Node.
→ It is Not Necessary that Every Node is pointing to a random Node.



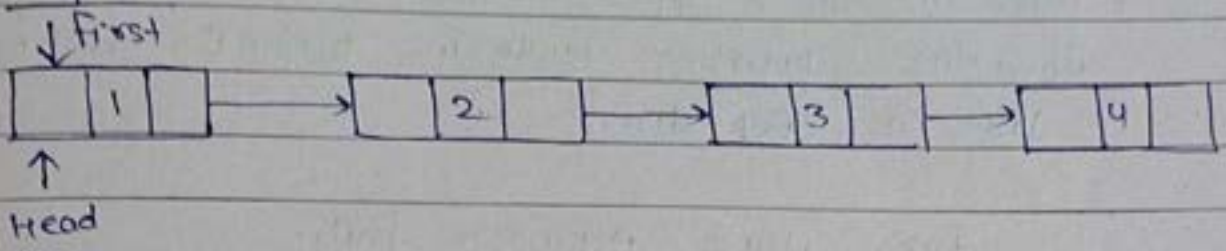
Task :

Copy the same linked list and return as it is.

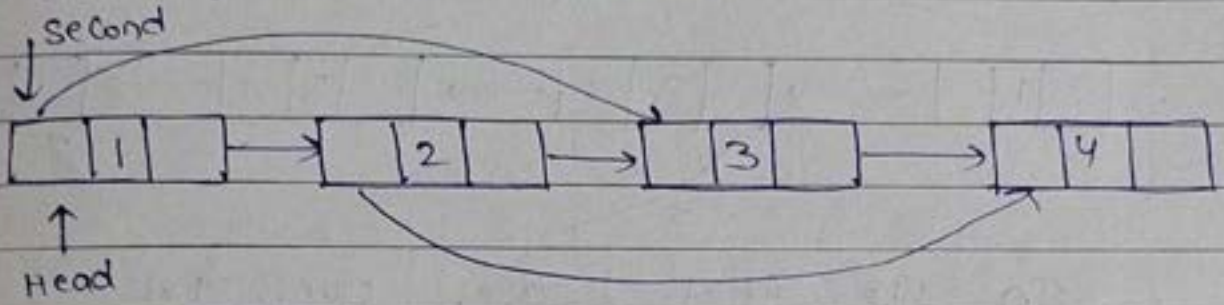
↔ Approach 1

Make a duplicate linked list having only one pointer (next pointer).

Duplicate:



Make original linked list also:



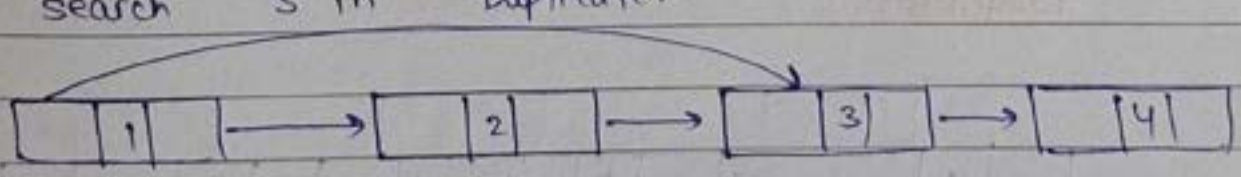
put Head on both linked list.

First pointer on duplicate linked list and second pointer on original linked list.

Check in original array that the first Node pointer to which Node which is Random.

We find 1 point to 3.

search 3 in Duplicate.



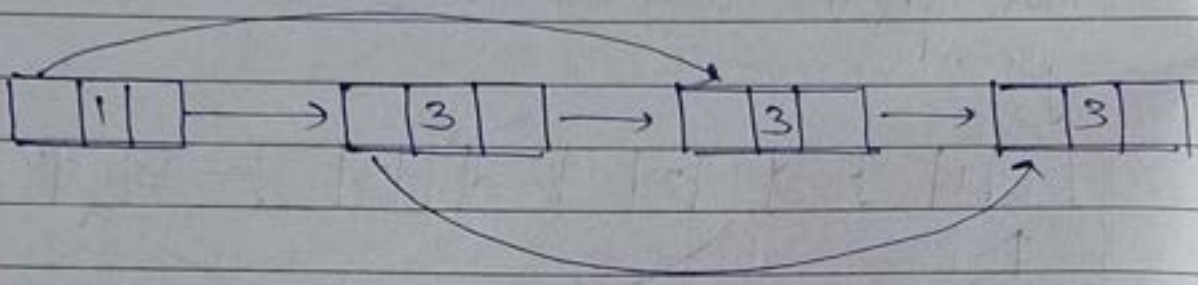
2 == 3 ⟶ 3 == 3
No Yes

point 1 in 3

Teacher's Signature.....

But, in this Approach we have a problem, if the pointing Node in original Linked list in repeating,

then this Approach fails.

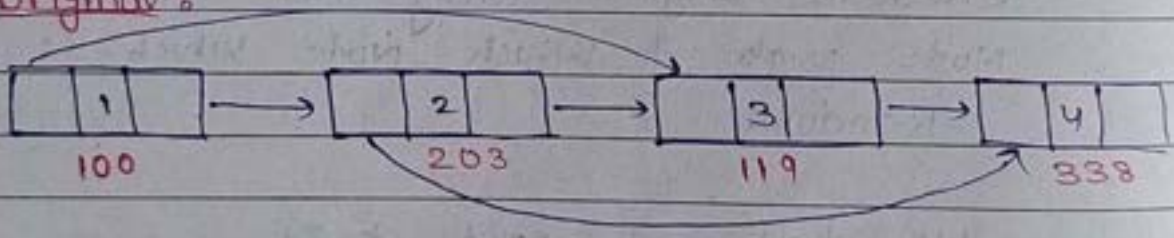


In this test case, our 1st Approach fail.

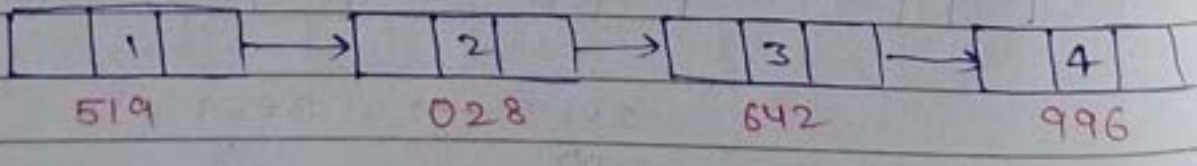
2nd Approach :

We have an another approach in which we can search by address.

Original :



Duplicate :



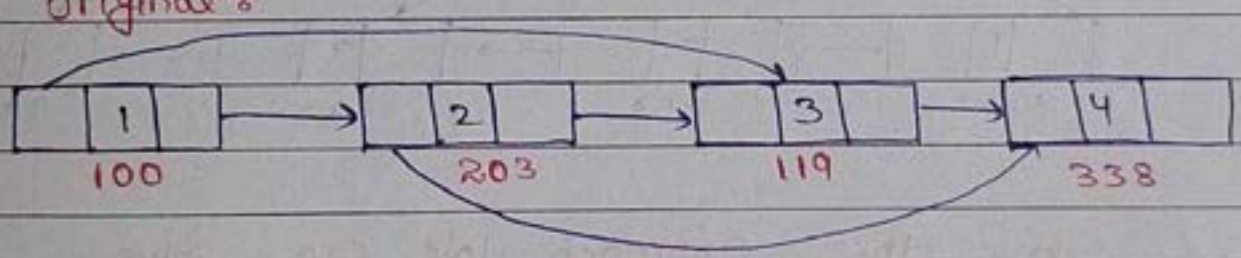
Teacher's Signature

We Will store the address of original linked list and duplicate linked list with equivalent Nodes.

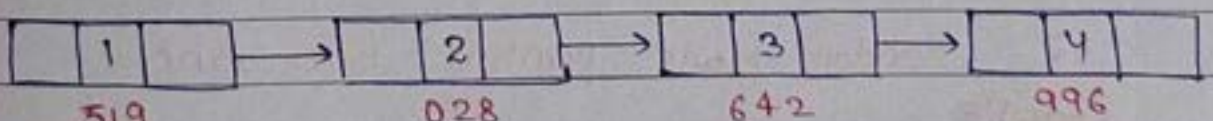
[100 → 519]
[203 → 028]
[119 → 642]
[338 → 996]

In original ~~array~~ LinkedList 1 → 3 and their address 100 → 119.

In duplicate linkedlist We traverse and check with the original array.



Duplicate :



Check in the Memory We
100 pointing to 119
find equivant address of duplicate
100 → 519
119 → 642

Teacher's Signature

① 519 → 642 (Random)

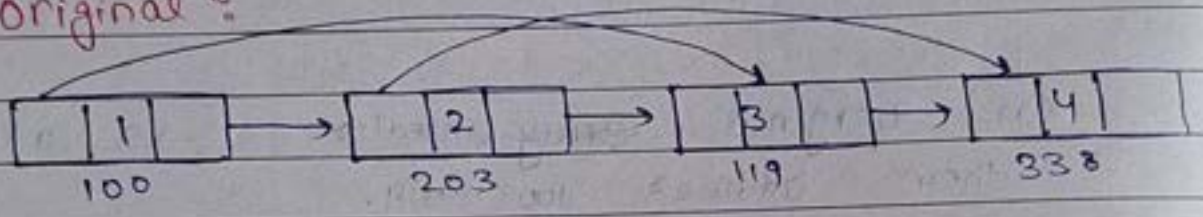
and 203 pointing to 338

Equivalent of 203 is 028

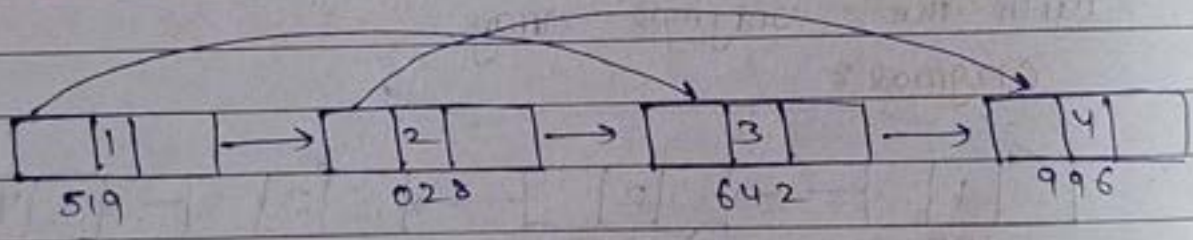
Equivalent of 338 is 996

⑪ 028 → 996 (Random)

Original :



Duplicate :

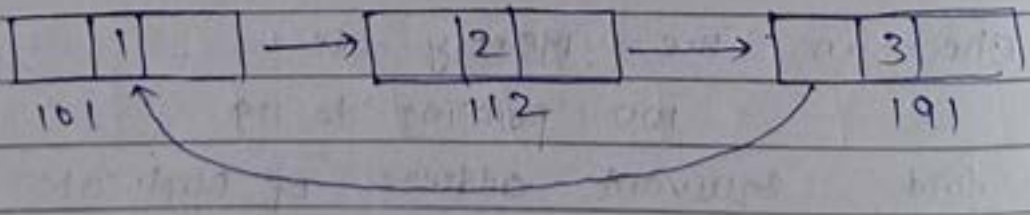


By this approach We can solve the problem

for the Random Node pointing forward.

If Random Node pointing Backward

Example



Teacher's Signature

In this it is access the back Node. But dont traverse backward.

So, this Approach fails also if we have address.

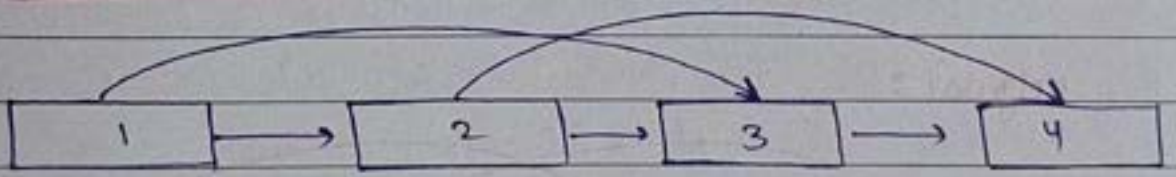
one More issue is

on this Approach, We have Space Complexity is O(N).

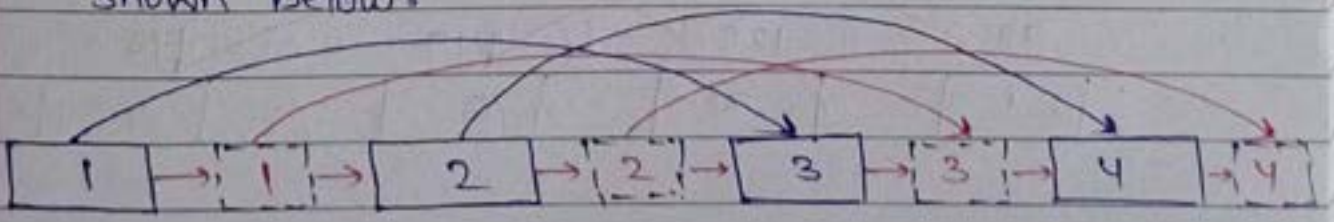
But We have to solve this in O(1). This Approach also fails.

We have to solve this Without using any Extraspace

Approach 3

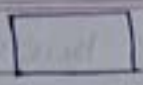


on this Approach, We will make a duplicate Node, Just after the Node Which is present in original Node. Shown below:

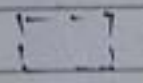


Like this.

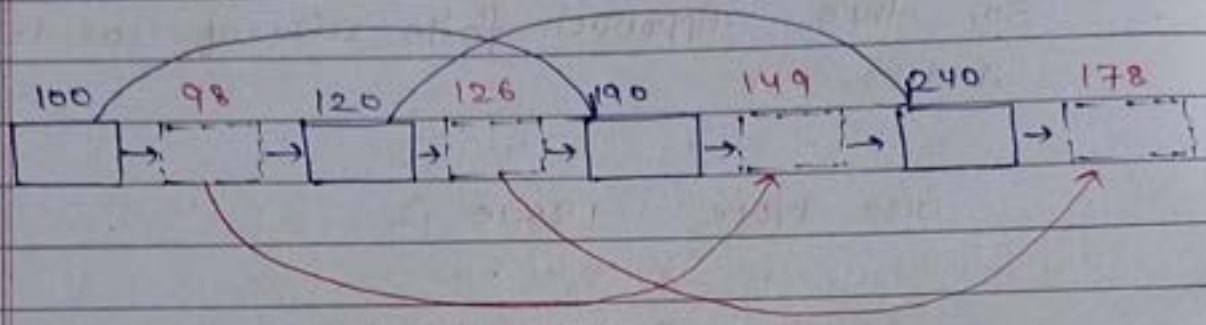
Teacher's Signature



→ original



→ Duplicate.



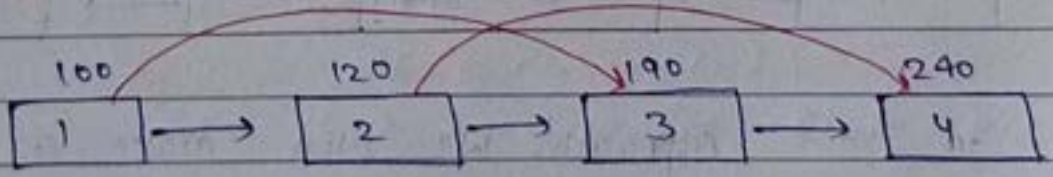
After this we will divide the original and duplicate linked list.

100 = 98
120 = 126
190 = 149
240 = 178

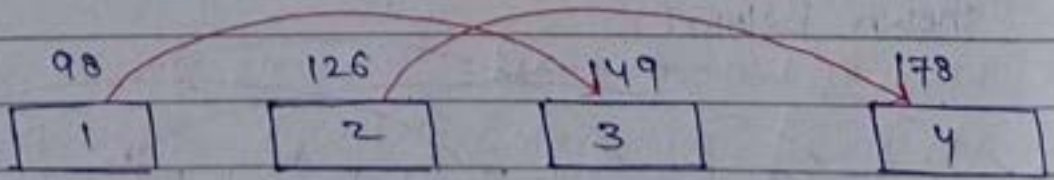
But we cant use this, we can done by next only.

Dividing the linkedlist which is Merged.

original :



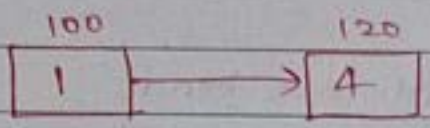
Duplicate :



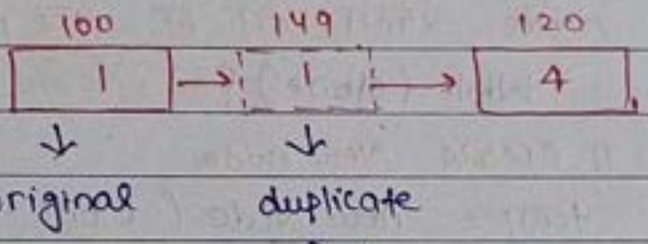
Teacher's Signature

How to code ?

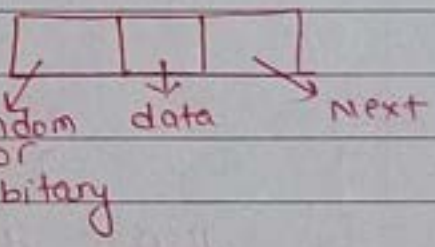
① हर Node के आगे उसका Duplicate Node copy कर के Attach kar denge



duplicating of 1

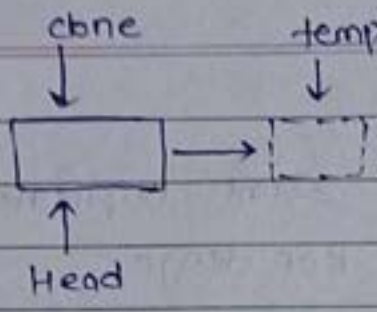


// class of Node




```
class Node {
    public :
     int data;
     Node * next;
     Node * arbt;
    Node (int x) {        // Constructor
        data = x;
        next = NULL;
        arbt = NULL;
    }
}
```

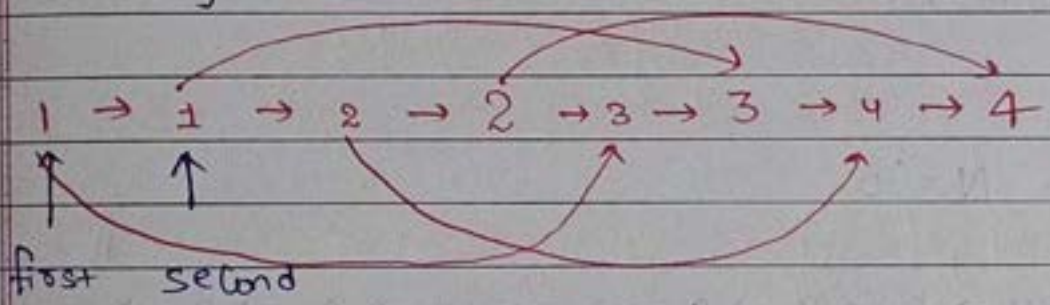
Teacher's Signature

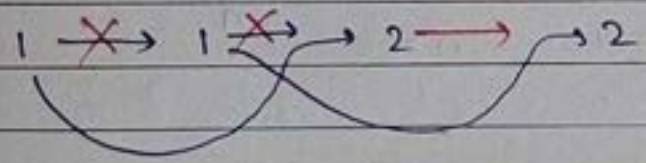



```
Node * clone = head;
Node * temp;
// duplicate कब तक होगा → जबतक original linked list
// का Node खत्म ना हो जाए।
While (clone) {
// Create New node
temp = new Node ( clone → data );
// temp अब Clone के आगे जाएगा
temp → next = clone → next;
clone → next = temp;
Clone = clone → next → next;
}
// All duplicate Node are Created
clone = head;
// Now, We point the random pointer
While (clone) {
if (clone → arb) {
clone → next → arb = clone → arb → next;
}
clone = clone → next → next;
}
// Now we will break the linked list one
// for original and another for duplicate.
```

Teacher's Signature

```
Node * ans = head → next;
    Clone = head;
temp = head → next;
While (temp) {
    clone → next = temp → next;
    clone = temp;
    temp = temp → next;
}
return ans;
}
```





1 → 2 ~

1 → 2

* Problem : 2

Reverse a sublist of a linked list

You have given a linked list.

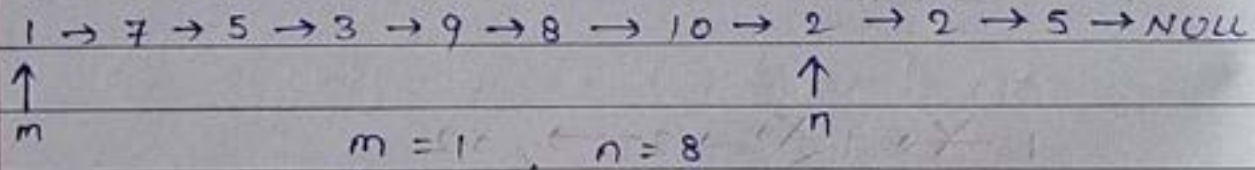
you have given 2 pointer m and n pointing 2 Nodes in a linked list.

Task :

Reverse the Node present from m to the node n.

Ex :

N = 10



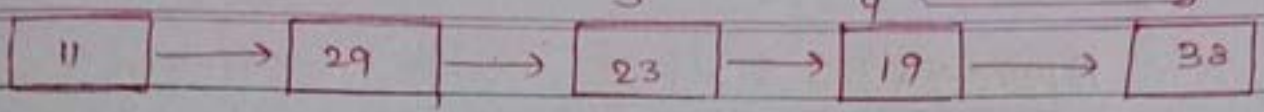
m = 1 , n = 8

Output :

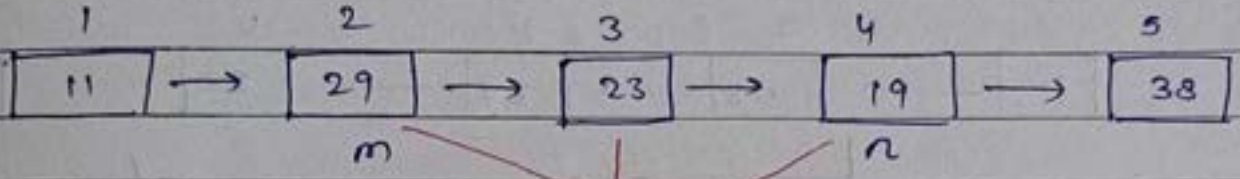
2 → 10 → 8 → 9 → 3 → 5 → 7 → 1 → 2 → 5

reversed

Teacher's Signature



m = 2
n = 4
} → Reverse the Node present inside m and n.



After Reverse

We have to reverse the linked list without doing swap.

Approach 1

Copy the Node of linked list into an array.

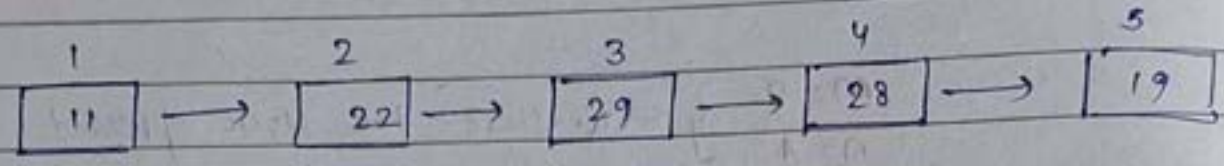
The reverse in array easily.

But we use here extra space and we dont have to use.

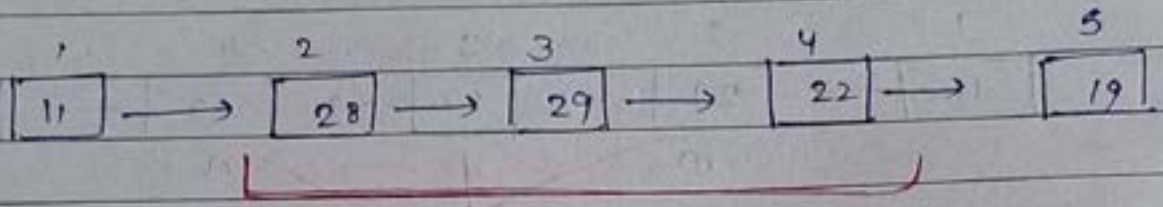
This Approach fails.

Teacher's Signature

Approach 2 :



m = 2
n = 4



→ Start
→ End
→ Mid

Here 4 cases Anisers :

(1) start to End (1 to 5)

(2) start to mid (1 to 4)

(3) mid to end (3 to 5)

(4) mid to mid (2 to 4)

We have to take care for all four cases.

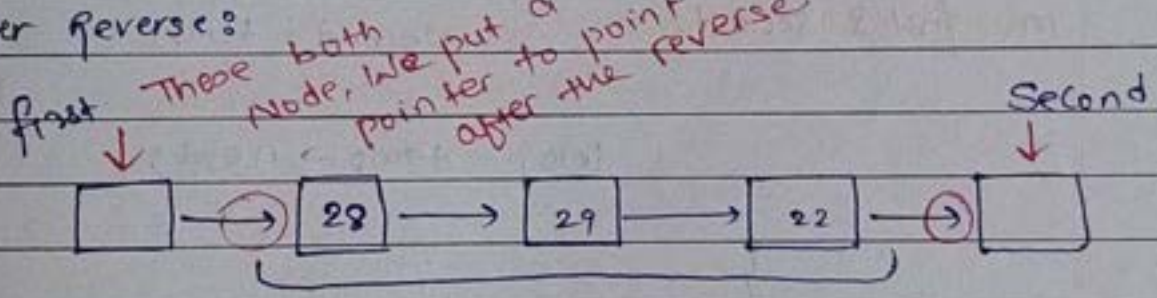
Teacher's Signature

Find How many No. we have to reverse ?

↗ From where we start

```
Node * Reverse ( Node * head, int n) {
        Node * prev = NULL;
        Node * prev = temp;
        While ( n--) {
            temp = head → next;
            head → next = prev;
            prev = head;
            head = temp;
        }
    return prev;
}
```

After Reverse :



we have to point the reverse part with previous part or last part.

first and second ka address store karwanta tha or phir reverse ke baad point kar dena hai.

Teacher's Signature.....

13 → 21 → 15 → 17 → 30 → 28 → 91 → 40

reverse part

m = 3 n = 7

```
First = NULL;
Second = NULL;
Node * temp = head;
int Count = 1;

While ( Count <= n+1) {

// here address of 3rd & 7th Node are stored in first & second
if ( Count == m+1)
    first = temp ;
if ( Count == n+1)
    second = temp;

temp = temp → next;

}
```

→ Cases :

```
if ( First == NULL){ // started from 0 1st Node

    if ( second == NULL){ // ended to last node.
        return Reverse (head,5); // n
    }
    else { // first to middle;
    temp = Reverse (head, 3); // n
    head → next = second;
    return temp;
    }
```

Teacher's Signature

// If not started from 1 Node
// started for middle to end & middle to middle.

// Middle to end.

```
if (second == NULL) {  // second NULL
first->next = Reverse(first->next, n-m+1);
return head;
}
```

// Middle to Middle

```
temp = first->next;
first->next = Reverse(First->next, n-m+1);
temp->next = second;
return head;
```

1 → 2 → 3 → 4 → 5 → 6 → 7

4 case :

Case ① Start to end (Total)

Reverse ही Return होगा।

Case ② start to Midd.

Reverse + Some last element

```
temp = Reverse(head, n)
head->next = second;
return temp;
```

Teacher's Signature.....

Case (3) : Mid to end

Some element + Reverse

~~int temp;~~

first → next = Reverse (first → next, n-m+1)

Case (4) Mid to Mid

Some Element + Reverse + some element

temp = first + next;

first next = Reverse (first → next , n-m+1)

temp → next = second;

return head;

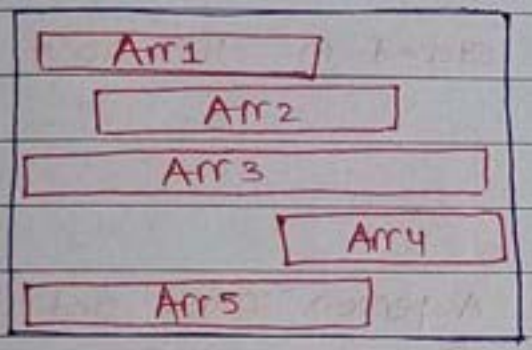
Teacher's Signature.....

(Lecture - 35)

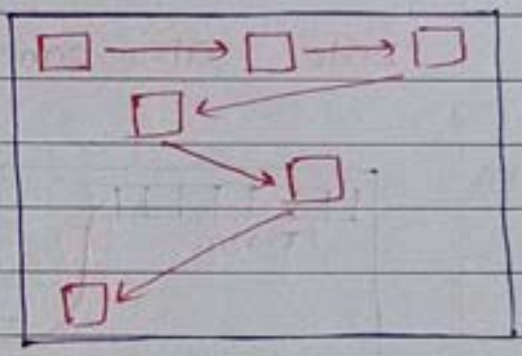
Stack

Till now, for data storage we can use either Array or Linked List.

For Array :



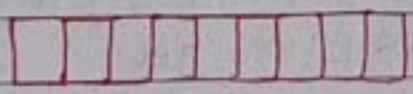
For Linked list :



We have keep the data in two ways.
How to use this data ?

Let us understand with the help of real Example :

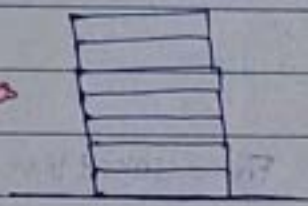
1) packet of Biscuit are arranged on a table -



If a person demand for biscuit, then he take either from front or back.

Teacher's Signature.....

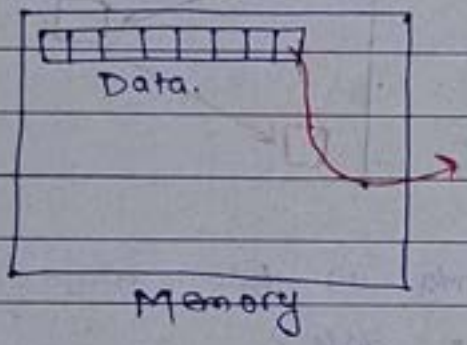
2) In a stationary shop, Copy is stored on table :-



A student demand for a copy then shopkeeper give you 2 copies from top.

Data also can be stored in this way.

3)



A person come and demand for some data.
then from memory he get from last.

If he demand more memory then the next memory he get.

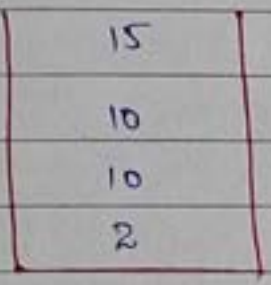
What if he demand memory / data from Middle, then how he get.

Here, We assume that he only want data from last.

We can also see the case We he get from first or middle.

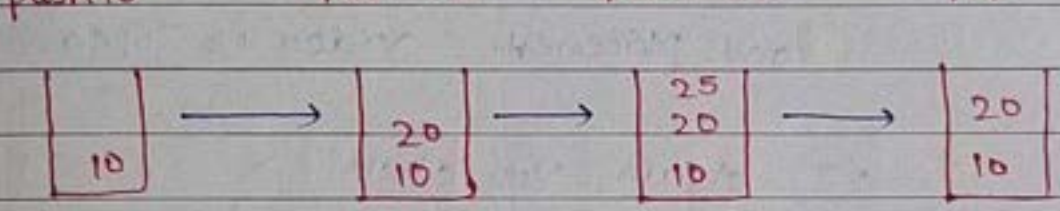
Teacher's Signature.....

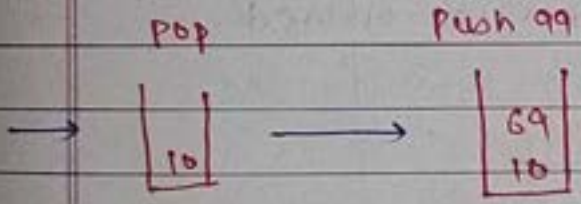
Stack (visual)



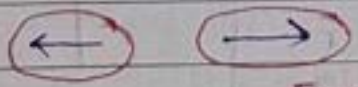
push : Insert an element
pop : Take the top element
top : Just Watch top element

How stack Mode?





In Web browser, you see :



back
undo

Forward
redo

Example:

on youtube you watch videos :

Teacher's Signature.....

Open Rohit Negi channel

4	Blockchain
3	placement
2	IIT Madras
1	Gate Video

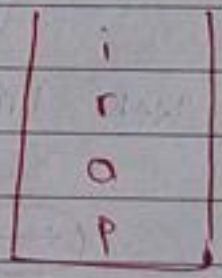
When you click this ←

Blockchain is delete from stack and placement Video is open.

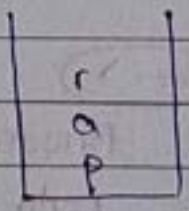
← Again you click placement video is deleted and IIT Madras is opened.

* undo operation (ctrl + z)

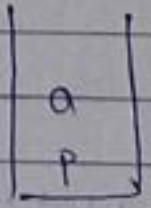
Write : Pari



type ctrl + z
i deleted



type ctrl + z
r deleted

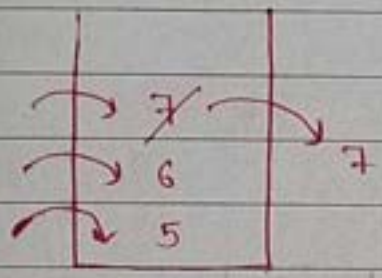


Teacher's Signature

* How to create a stack :

Push()
pop()

LIFO (Last in first out)
↳ used in stack.



We can create a stack in two ways :

1) Creation by Array
2) Creation by Linked list

1) Creation By Array :

2	3	7	11	19	23

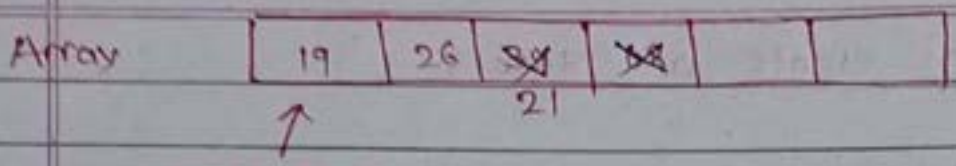
DrawBack :

size is fixed. If we want to store more data, then it is not possible.

Array size = 6

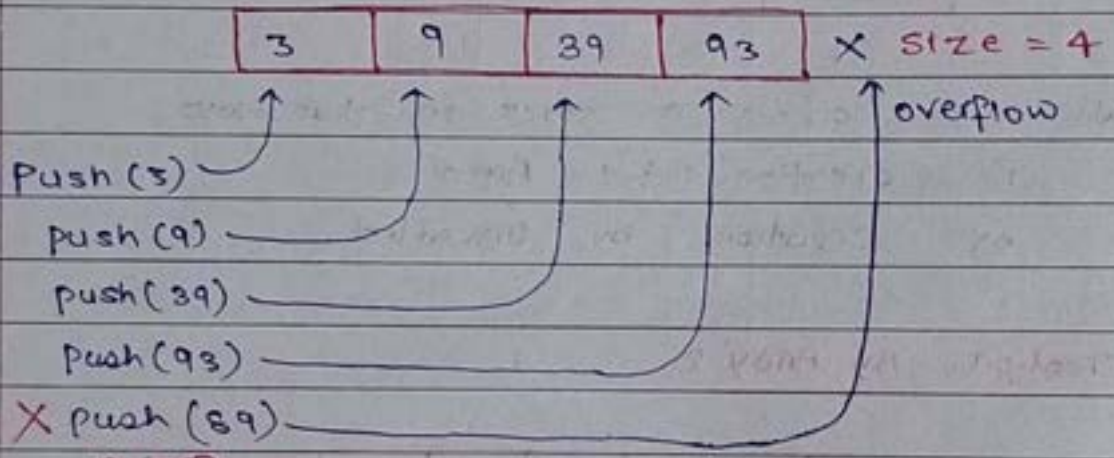
Let Make an array of size = 6

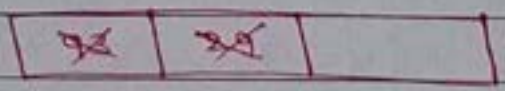
Teacher's Signature.....



push (19)
push (26).
push (99)
push (08)
pop → delete 08
pop → delete 39
push (21)

* Overflow & underflow :

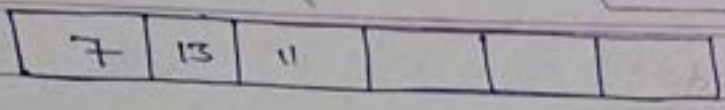




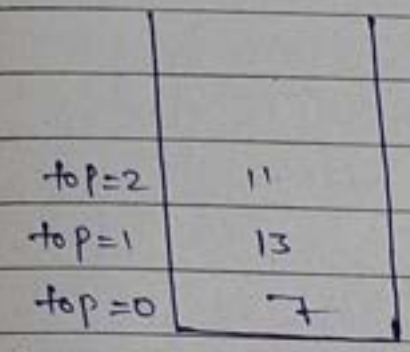
Push (99)
Push (29)
pop → delete 29
pop → delete 93
pop → underflow because No element present.

Teacher's Signature.....

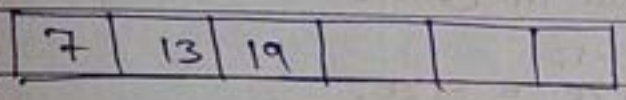
Arr →

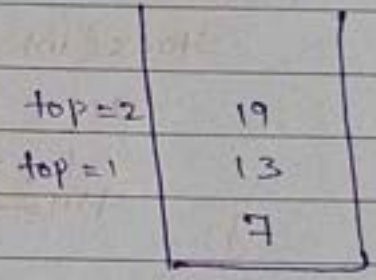


Initially top = -1



① push 7 top++
② push 13 top++
③ push 11 top++
④ pop top--





⑤ push 19

* Condition for overflow & underflow :

(top == arr.size())
Stack overflow // push not possible

(top == -1)
Stack underflow // pop not possible.

S.top() → from this we can print the top element.

Teacher's Signature

Code :

```
#include <iostream>
using namespace std;

class Stack {
    int top;
    int * arr;
    int size;
  public:
    Stack(int size) {
        arr = new int [size];
        top = -1;
        this->size = size;
    };

    Void push( int data) {
        if (top == size-1) {
        Cout << " Stack overflow \n" ;
            return;
        }
        top++;
        arr[top] = data;
    }

    Void pop () {
        if (top == -1) {
        Cout << " underflow \n";
            return;
        }
        top--;
        return;
    }
```

Teacher's Signature.....

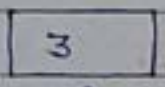
```
int peak () {
        if (top == -1) {
            Cout << " stack underflow";
            return -1;
        }
        return arr[top];
    }

boolean empty () {
    return top == -1;
}

int main () {
    Stack S(4);
    S.push(4);
    S.push (19);
    S.push (413);
    Cout << S.peek();
    return 0;
}
```

Output : 413

Teacher's Signature.....

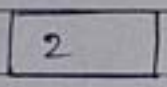
* Stack by Linkedlist :-

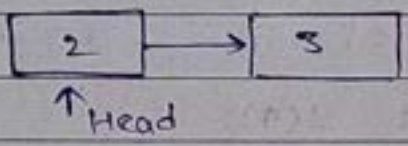
push (3)



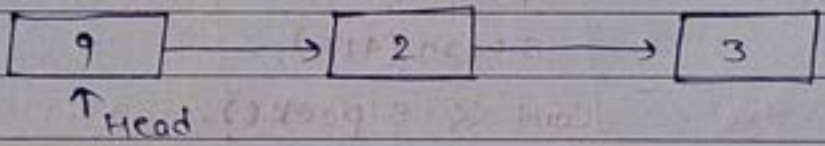
New Element आते जाएगा उसको Head बनाते जाएंगे।

push (2)

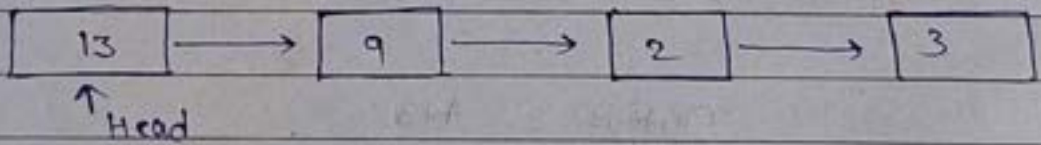




push (9)

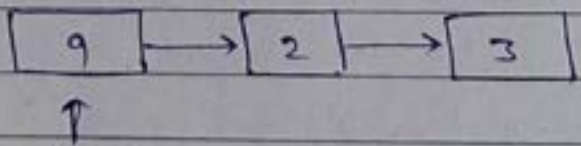


Push (13)



pop ()

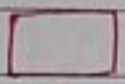
Delete 13



Teacher's Signature.....

push() → O(1) ✓
pop() → O(1) ✓
top() → O(1) ✓

Empty : Condition



If Head → NULL
then Empty

Where we use stack :
→ On Browser
→ For undo operation. ←
→ for Redo operation →

Youtube → channel (Rohit Negi) → placement → 2 Crore
→ Data → IIT Madras → Blockchain

Blockchain
IIT Madras
Gate
2 Crore
Placement

After this, if we press
← ↳ Delete the Blockchain
← ↳ Delete the IIT Madras
→ ↳ Again come IIT Madras.

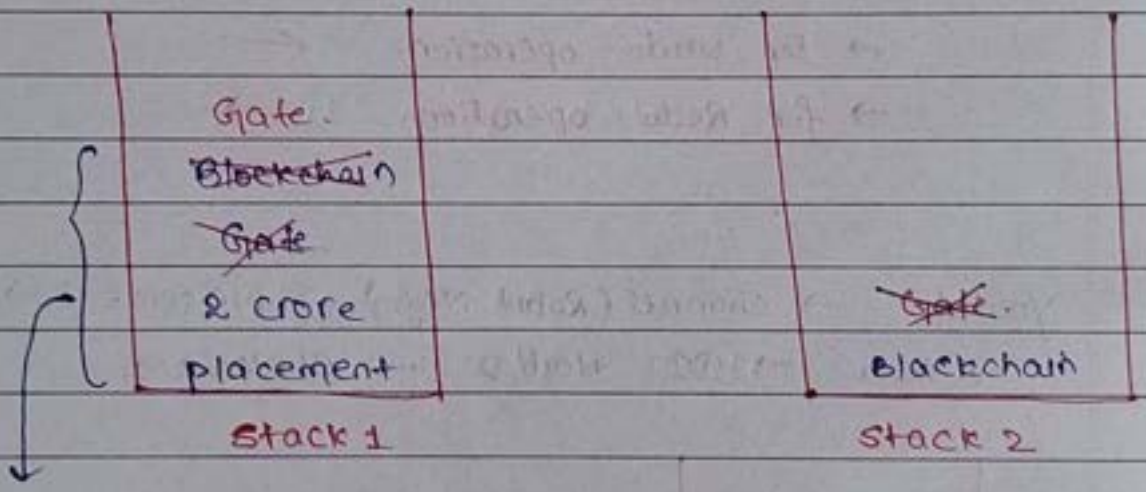
Teacher's Signature.....

After deleting How it Comes again by clicking this (→)

By the help of 2 stack.

When we undo (←) it is deleted from one stack and stored in another stack.

And, When we do Redo (→) it is again Come is 1st stack and deleted in 2nd stack.



After this if we put undo (←) it is deleted from stack 1 and stored in stack 2.

← (undo) Again we put undo, it delete Gate from stack 1 and stored in stack 2.

→ (Redo) If we put Redo, it delete from stack 2 and again Come to stack 1.

Teacher's Signature.....

Code :

```
#include <iostream>
#include <stack>
using namespace std;

class Node {
    public :
        int data;
        Node * next;
        Node (int data) {
            this -> data = data;
            next = NULL;
        }
};

class Stack {
        Node * top;
        public :
            Stack () {
                top = NULL;
            }

    Void push (int data) {
            Node * temp = new Node (data);
            if (!temp) {
                Cout << "stack overflow \n";
                return;
            }
            temp -> next = top;
            top = temp;
        }
```

Teacher's Signature.....

```
Void pop() {
    if (! top) {
        Cout << " stack underflow \n";
            return;
        }
        Node * temp = top;
        top = top → next;
        delete temp;
}

int peek() {
    if (!top) {
        Cout << "Stack is empty \n" ;
        };
        return top → data;
    };

    bool empty() {
        return top == NULL;
    };
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    Cout << s.empty();
return 0;
}
```

Teacher's Signature.....

Problem : 1

Stack : function call

* Stack with help of STL

```
Stack<int> s;
Stack<char> s;
s.push(4);
s.pop();
s.push(9);
```

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
    Stack<int> s;
    s.push(14);
    s.push(19);
    s.pop();
    cout << s.empty();
    return 0;
}
```

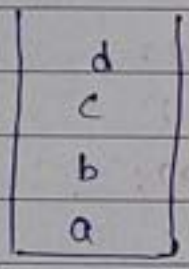
s.size()

* problem - 1 : (Not from GFG)

String : "abcd"

Reverse this by stack

put string in stack



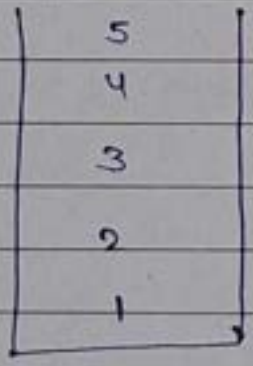
Now pop one by one and print

"dcba"

result.

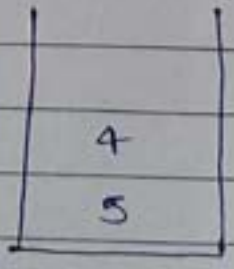
* problem - 2 :- (delete Middle element of a stack)

GFG



Delete Mid element

① Copy n/2 element in another stack



Teacher's Signature.....

② pop next element
→ 3

③ Then, return the element from stack 2

5.
4
2
1

O(N)

If length = even
then put n/2 element in another stack.

If length = odd
then put n/2 element in another stack.

Code:

```
Void deleteMid ( Stack <int> &s, int size) {
        Stack <int> temp;
        int Count = size / 2;
        While ( Count --) {
            temp.push ( s.top());
            s.pop();
        };
        s.pop();
        While(temp.size()) {
            s.push (temp.top());
            temp.pop();
        }
}
```

Teacher's Signature.....

GfG Problem : 3 (parenthesis checker 2744)

Given Expression string x.

string can consists only
{ } [ ] ( )

Check that parenthesis is Correctly placed or not.

[( )] → balanced

{ } → balanced

([) → Not balanced / invalid.

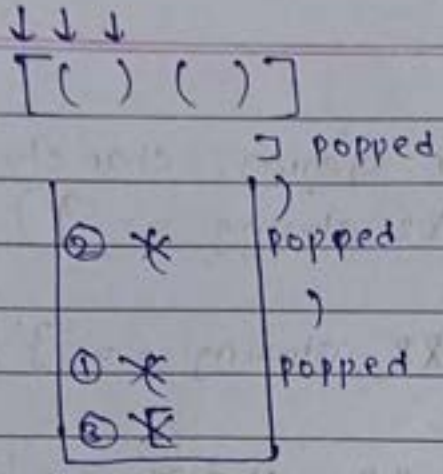
How to check :

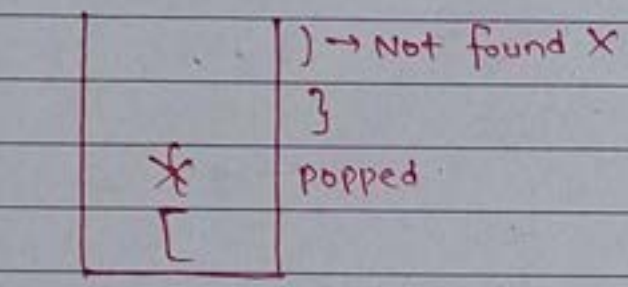
[( ) ( )] → valid

If we met an open, then we find for closed, if closed found then valid otherwise invalid.

If we met an close before open
↳ then it is invalid..

Teacher's Signature.....





Invalid

Code :

```
bool ispar (string exp) {
    stack <char> s;
    for (int i = 0; i<exp.length(); i++) {
        if (exp[i]=='(' || exp[i]=='{' || exp[i]=='[')
            s.push(exp[i]);
        else if(exp[i]==')' || exp[i]=='}' || exp[i]==']')
        {
            if (s.empty() || !isMatchingpair(s.top(), exp[i])
                return false;
            else
                s.pop();
        }
    }
    return s.empty();
}
```

Teacher's Signature .....

~~bool is M~~

```
bool isMatchingpar(char opening, char closing){
    if(opening == '(' && closing == ')')
        return true;
    else if(opening == '{' && closing == '}')
        return true;
    else if(opening == '[' && closing == ']')
        return true;
    return false;
}
```

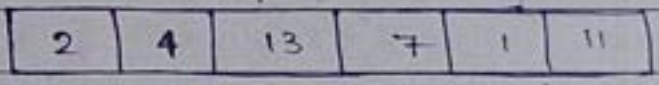
Teacher's Signature.....

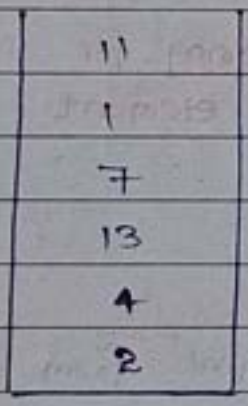
Lecture - 36
(Stack - Medium)

Problem 1 : Get Minimum element from stack

You have N element
↳ find the min-m element while implementing a stack.

Required time Complexity : O(1)





push ✓
pop ✓
get_min()

Ans = 1

We check one by one all the element and find min-m
but time Complexity is O(N)
and required is O(1)

This approach fails.

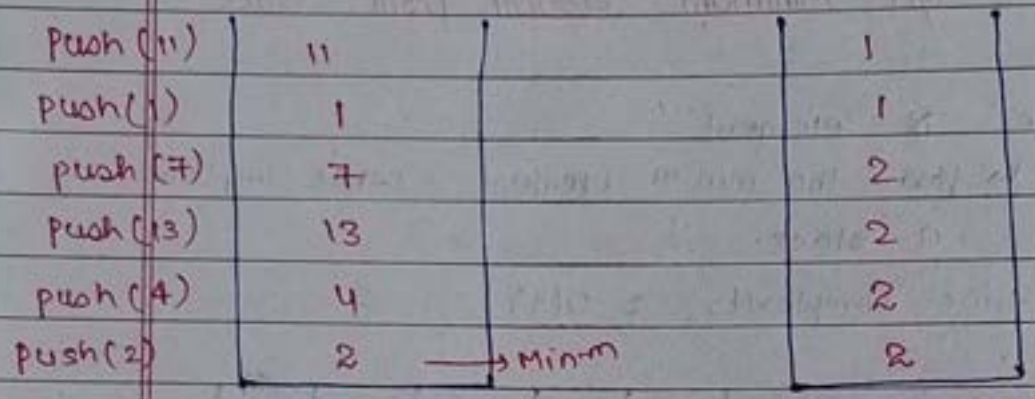
We can solve this by Another Approach.

Teacher's Signature.....

Approach 2:

2	4	13	7	1	11

Make 2 stack



↳ This stack is storing for Minm element

We will push all the element in stack in constant time.

We will return the top element from 2nd stack which is minimum.

But here we can take extra space for second stack.

Space Complexity : O(N)

Not Required

We want time Complexity and space Complexity both as O(1).

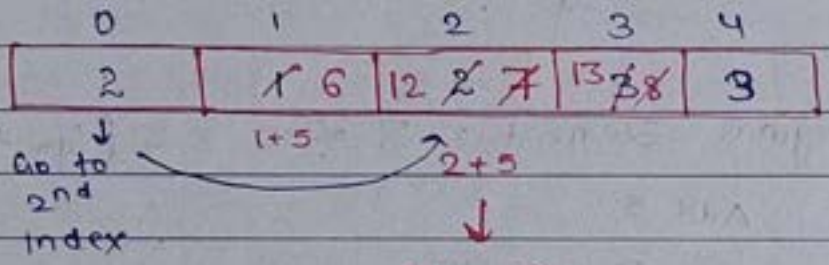
Teacher's Signature.....

Approach 3:

$$\frac{\text{freq} * \text{size} + \text{original_value}}{\text{size}} = \text{value}$$

frequency = divide

original value = Mod



7 % 5 - (2)

↓

original value

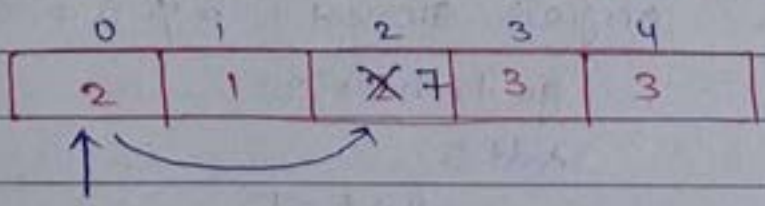
+5

Explanation

Limit of Array element → (0 to 4)

We take a higher value

Let we take 5.



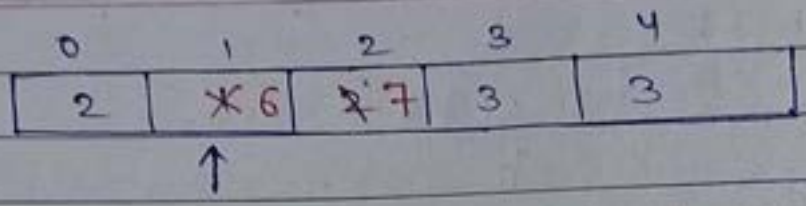
Element = 2 % 5 = 2

Go to index' 2

Add 5 to that index

2 + 5 = 7

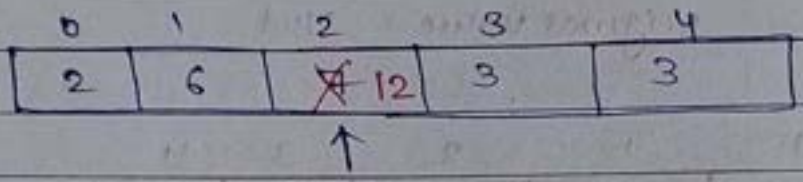
Teacher's Signature.....



Element = 1%.5 (Go to index 1)

Add 5

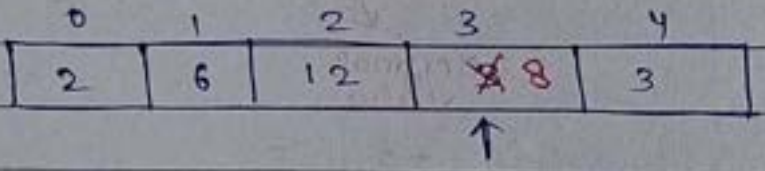
1 + 5 = 6



original Element = 7 %. 5 = 2 (Go to index 2)

Add 5

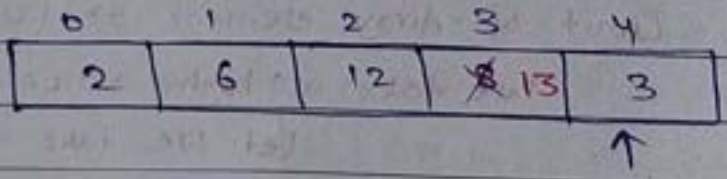
7 + 5 = 12



original Element = 3%.5 (Go to index 3)

Add 5

5 + 3 = 8



original Element = 3 %. 5 = 3

go to index 3

Add 5

8 + 5 = 13

Teacher's Signature

For Finding Frequency, divide all element by 5

Frequency =	0	1	2	2	0

$2/5 = 0$ $6/5 = 1$ $12/5 = 2$ $13/5 = 2$

$3/5 = 0$.

* Same Concept We use in stack.

Range of Number is 1 to 100.

Take a number bigger than Range.

Let n = 101

originalvalue + minelement * 101

101

for this = mod

For this = divide.

$1 <= num <= 100$

Why We take 101, Why not 102, ... ;

↳ because when we mod We get 0 to 100 which is limit of num.

[original value + minelem * 101] = value

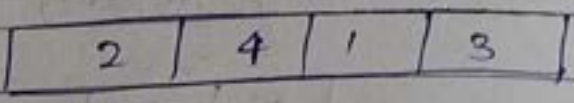
↓

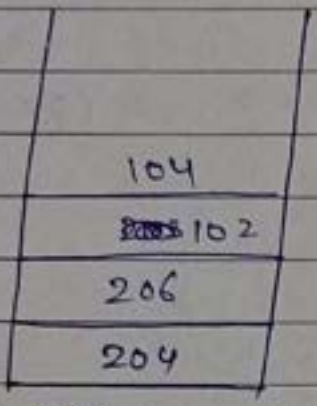
put in stack

value % 101 = original_value

value / 101 = min_element.

Teacher's Signature.....





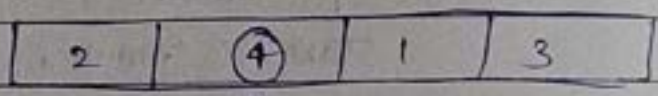
Stack

~~Value = 2 + 100~~

Value = 2 + 101 * (2)

= 204

because till now no min-m present



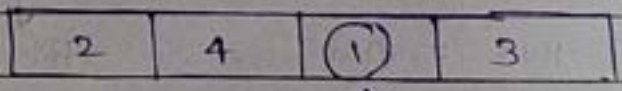
min-m value = $\frac{204}{101}$

= (2)

(2, 4) → min = 2

Val = 4 + 101 * 2

= 206



min-m element

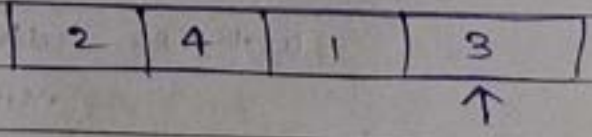
= $\frac{206}{101}$

= (2)

(2, 1) → min = 1

val = 1 + 101 × 1

= ~~203~~ 102



min = $\frac{102}{101}$

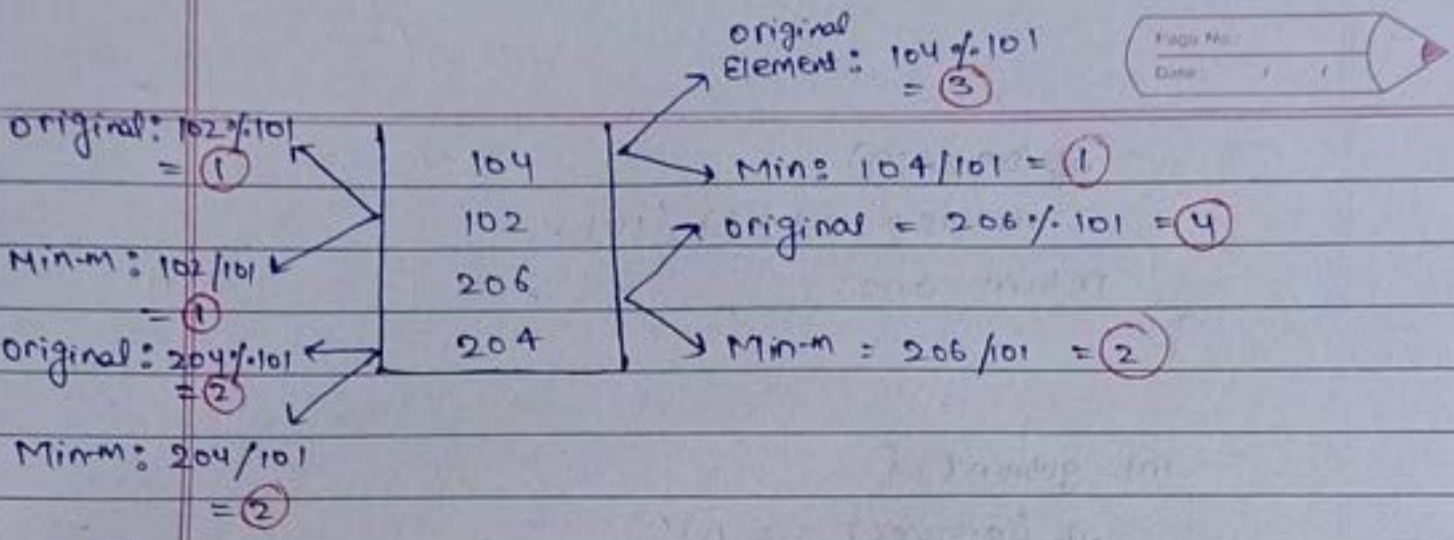
= 1

(1, 3) → min = 1

Val = 3 + 101 × 1

= 104

Teacher's Signature



By this We can get both by one stack.

→ original element

→ Min-m Number.

* Implementation of push:

```
Void push (int x) {
    if (s.size() == 0) {
        minEle = x;
        S.push (x + (101 * x));
    }
    else {
        minEle = min (x, S.top()/101);
        S.push (x + (101 * minEle));
    }
}
```

* Implementation of pop:

```
int pop() {
    if (S.size() == 0)
        return -1;
    int ans = S.top();
    ans = ans%101;
    S.pop();
```

Teacher's Signature.....

```
    if (s.size())
        minEle = s.top() / 101;
    return ans;
}

int getMin() {
    if (s.size() == 0)
        return -1;
    else
        return s.top() / 101;
}
```

Problem 2: String Manipulation

Sequence of n words.

If two same word come together then he destroy both.

After all destruction happen, how many numbers of words left.

V[] = { ab, aa, aa, bcd, ab }

destroyed

output = 3

Teacher's Signature.....

* v = [tom, jerry, jerry, tom]

destroy

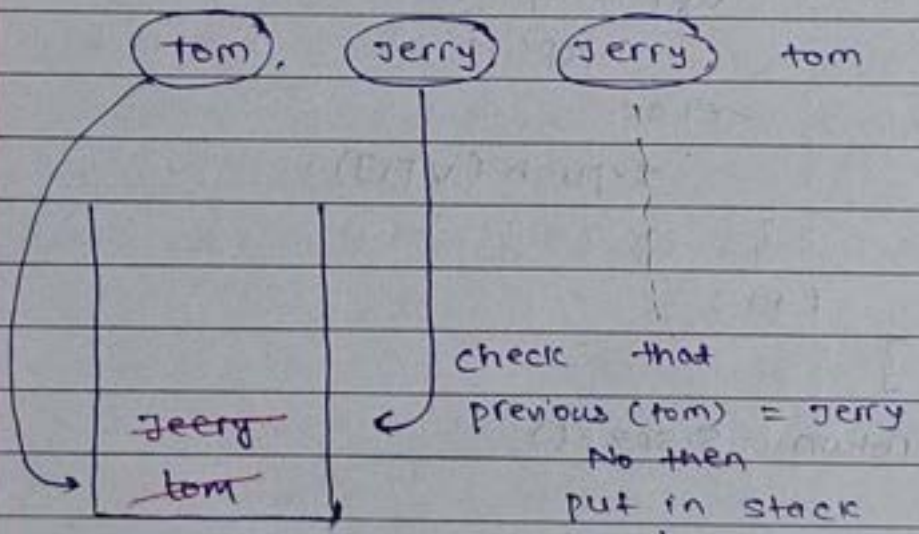
v = [tom, tom]

destroy

v = [ ]

output = 0

* How to solve :



Again jerry

Check stack.top() == jerry

yes

then delete jerry

pop (top)

Again tom, check

stack.top() == tom

Yo

pop (top).

Teacher's Signature.....

then, Count the number of element present in stack and return.

Code:

```
Stack <string> s;
  int i= 0;
  While( i< v.size()) {
      if ( s.size() == 0)
         s.push (v[i]);
      else {
           if ( s.top() == v[i])
              s.pop();
           else
              s.push (v[i]);
        }
      i++;
  }
  return  s.size();
```

Problem : 3 Make the array beautiful

Given a Array → With negative & non-negative integers.

Condition for beautiful :

two adjacent integer,

arr[i] and arr[i+1] are either Negative or non-negative.

You have to do some operation until your array becomes beautiful :

If two adjacent integer are different i.e. one of them is negative and other is non-negative, remove them.

Empty array is also a beautiful array.

Example

destroyed

Input : 4 (2 -2) 1

output : 4 1

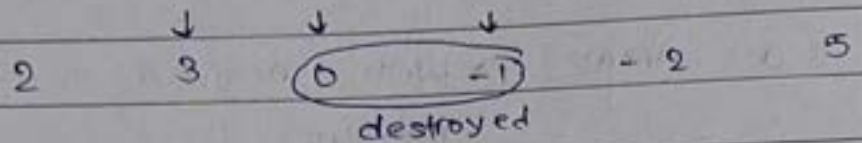
Example :

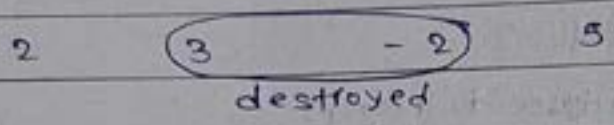
Input : (2 -2) (1 -1)

destroyed destroyed.

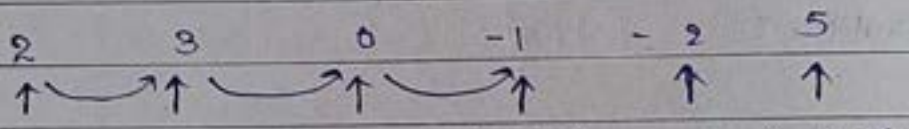
Teacher's Signature

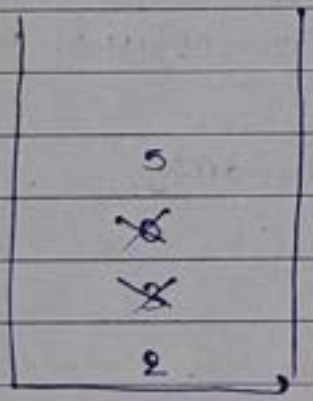
By stack :



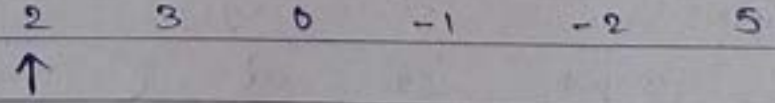


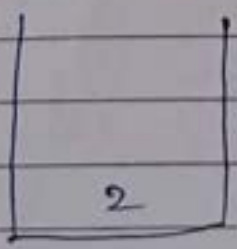
[2, 5] → answer.





detailed Explanation :





2 . (stack empty)
Push (2)

Teacher's Signature.....

2 3 0 -1 -2 5

↑

+ve

3
2

top = 2 (+ve)
push(3)

2 3 0 -1 -2 5

↑

0
3
2

0 +ve
top = 3 (+ve)
push

2 3 0 -1 -2 5

↑

~~x~~
3
2

-1 -ve
top = 0 +ve
pop(top)

2 3 0 -1 -2 5

↑

~~x~~
2

-2 (-ve)
top = 3 +ve
pop(top)

2 3 0 -1 -2 5

↑

5
2

5 (+ve)
top = 2 +ve
push

Now, return the element which are present in stack.

Teacher's Signature.....

Code :

```
Vector <int> makeBeautiful (vector <int> arr) {
        Stack <int> s ;
    for (int i=0;  i< arr.size();  i++) {
          if ( s .empty ()) {
              s. push (arr[i]);
            }
          else {
           if ((s.top() >= 0 && arr[i] < 0) ||
               (s.top() < 0 && arr[i] >=0 )) {
                  s. pop();
             }
           else {
               s.push (arr[i]);
             }
        }
    }

Vector <int> beautifullArray;
    While (! st.empty()) {
        beautifullArray . push_back ( s.top());
            s.pop ();
      }

 reverse ( beautifullArray. begin (), beautifullArray . end());

 return  beautifullArray ;

}
```

Teacher's Signature.....

Problem - 4

* Next Greater Element :

Given an Array :

return an Array with next greater Element.

Ex: arr[] = [1, 3, 2, 4]

ans = [

~~1, 3, 2, 4~~

~~for 1~~

1, 3, 2, 4

for (1) → Next greater = 3

ans = { 3, }.

for (3) → Next greater = 4

ans = { 3, 4 }

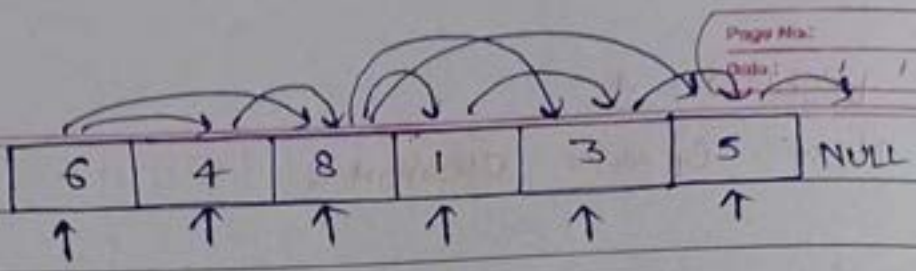
for (2) → Next greater = 4

ans = { 3, 4, 4, }

for (4) → Next greater = Not present (-1)

ans = { 3, 4, 4, -1 }.

Teacher's Signature.....

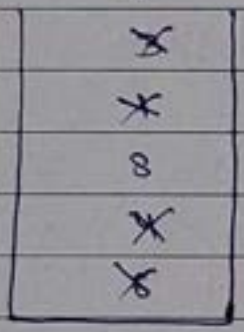


output :

8	8	-1	3	5	-1

*

6	4	8	1	3	5
8	8	-1	3	5	-1



6 → put 1st element

4 → 4 < 6 put in stack

8 → 8 > 4 ,8>6

1 → 1 < 8 put in stack

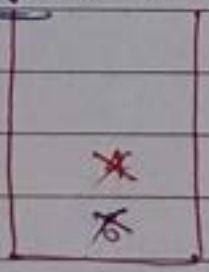
3 → 3 > 1

5 → 5 > 3

*

6	4	8	3	1	2	5
↑	↑	↑	↑	↑	↑	↑
8	8	-1	5	2	5	-1

Next greater



6 → put in stack

4 → 4 < 6 (then put in stack)

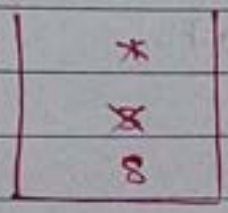
8 → 8 > 4 (then for 4 '8' is next greater

Pop(4)

Teacher's Signature.....

8 > 6 then

For 6 '8' is next greater



pop (top);

3 → 3 < 8 push

1 → 1 < 3 push

2 → 2 > 1 for 1

2 is Next greater

5 → 5 > 2

then for 2

(5 is greater.)

(iii)

for rest (-1) is next greater

Code:

```
stack <int> s;
int i = 0;
while (i < n) {
  if (s.size() == 0)   // stack is empty
     s.push (i);
  else {
     if (arr[s.top()] >= arr[i])
        s.push(i);
     else {
```

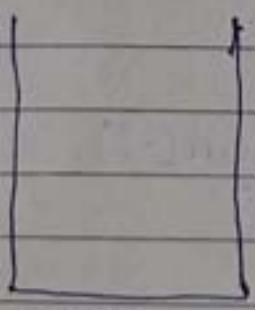
Teacher's Signature.....

```
        while ( s.size() && arr[s.top()] < arr[i])
        {
            ans[s.top()] = arr[i];
            s.pop();
        };
        s.push(i);
    }
  ]
  i++;
]

while ( s.size()) {
    ans[s.top()] = -1;
    s.pop();
};
return ans;
}
}
```

* Problem - 5 : Smallest number on left :

a = [1, 5, 0, 3, 4, 5]

Output : -1 1 -1 0 3 4



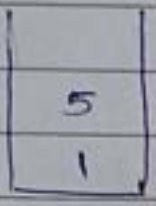
1 5 0 3 4 5

↑

Stack

Teacher's Signature

(1) → Stack is empty
then store (-1)



Output = [-1,

1 5 0 3 4 5
↑

(5) → stack is not empty.
if smaller than arr[i] found
then return.
1 is smaller. and then

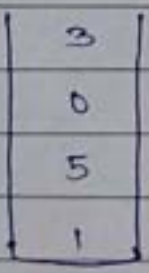
output = [-1, 1

1 5 0 3 4 5
↑

(0) → stack is not Empty
check, stack if smaller than arr[i] then return.
otherwise, return -1
5 > 0
1 > 0
return -1

output = [-1, 1, -1

(3) stack is not empty
check smaller in stack
found 0 return 0.



output = [-1, 1, -1, 0

Teacher's Signature

4
3
0
5
1

④ Stack is not empty
check smaller than 4
return 3

output = [-1, 1, -1, 0, 3,

⑤ stack is not empty
check smaller than 5
return 4

output = [-1, 1, -1, 0, 3, 4]

Loop Ends
Return output.

Code :

```
Vector<int> leftsmaller(int n, int a[]){
    Vector<int> result(n, -1);
    stack<int> s;

    for(int i=0; i<n; i++){
        while(!s.empty() && s.top() >= a[i]){
            s.pop();
        }
        if(!s.empty()){
            result[i] = s.top();
        }
        s.push(a[i]);
    }
    return result;
```

Teacher's Signature

3: 3

(Lecture - 37)

Stack - Hard

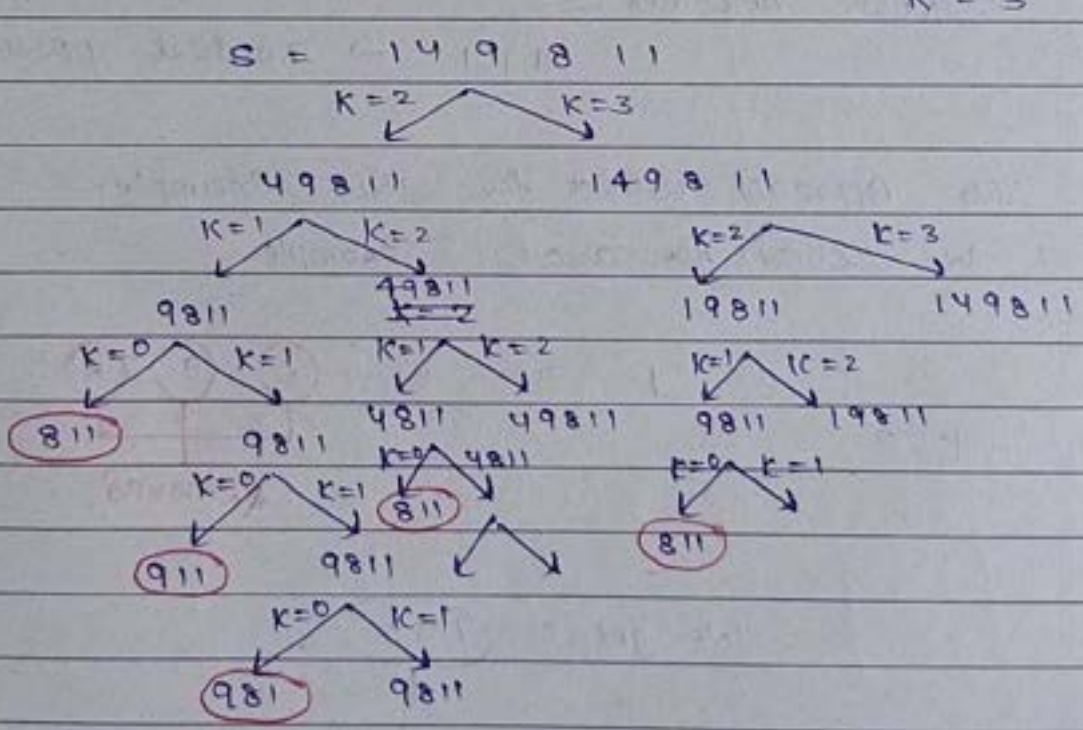
Problem 1 : Remove k digits

Given : A Non negative Integer represented as string

Task : Remove k digits from number, so, new number becomes smallest possible.

give num does not contain any leading zero.

Approach 1 : (By recursion)



Compare all the outcome where K = 0, and find the smallest among them and return

Teacher's Signature.....

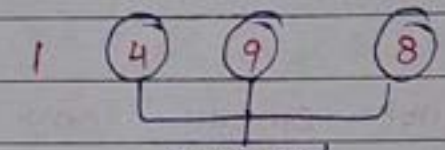
Time Complexity : $O(2^n)$

But we have to solve in $O(n)$.
We find another Approach.

Approach 2 :

1 4 9 8 1 1 (k = 3)

Remove the K digit which is largest



Removed
3 digit

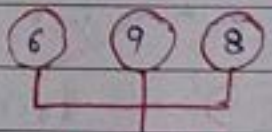
Now, becomes :

1 1 1 → smallest possible

This Approach works for this Example.
We check for another example.

1 3 2 6 9 8

K = 3



Removed

We get : 132

↓

is it smallest possible ? No.
because 126 is smaller than 132.

This Approach fails.

Teacher's Signature

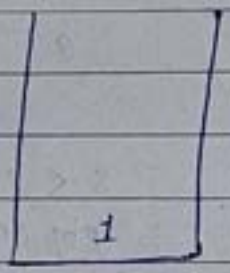
Page No.
Date: / /

Approach 3:

1 3 2 6 9 8

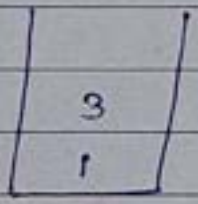
k = 3

Make a stack



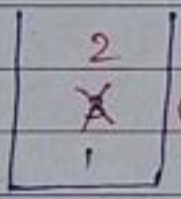
1 3 2 6 9 8
↑

Stack empty
push(arr[i]);



1 3 2 6 9 8
↑

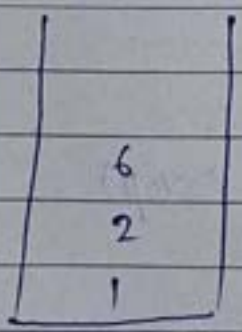
if (arr[i] < top())
pop(top());
k--;
else if (arr[i] > top())
push(arr[i]);



k = 2

1 3 2 6 9 8
↑

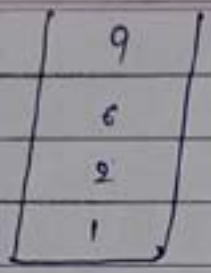
2 < 3 ← top
pop() → 3
K--;
push(2)



1 3 2 6 9 8
↑

6 > 2

Teacher's Signature.....

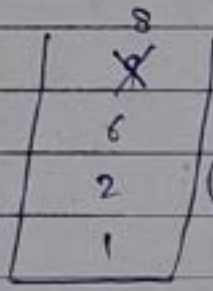


1 3 2 6 9 8

↑

9 > 6

push (9)



1 3 2 6 9 8

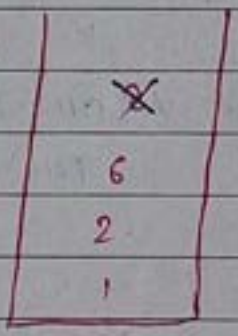
↑

8 < 9

pop (9)

push (8)

At last, we have



K = 1 → remove from top

pop (8)

Ans = (1 2 6)

Example 2)

2 4 5 1 7 8

↑ ↑ ↑ ↑ ↑ ↑

push(2) 4>2 push(4) 5>4 push(5) Push push

1 < 5

pop(5) K=1

1 < 4

pop(1) K=0 (stop)

8 7 1 2

↳ Reverse

Ans 2 1 7 8.

Teacher's Signature

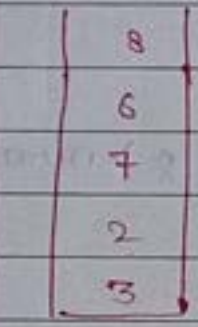
Example

6 3 9 4 2 7 6 8

k = 3

push(6) | 9>3 push(9) | 2<4 pop(9) k=0 | push after k=0

3<6 delete 6 k=2 push(3) | 4<9 pop(9) 4>3 push(4) k=1 | push(2)



→ 8 6 7 2 3 → Reverse = Ans

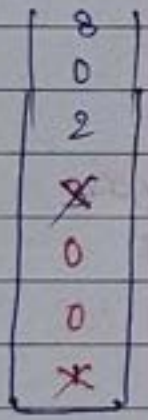
Ans = 3 2 7 6 8

Example

1 0 0 3 2 0 8

(k=2)

push(1) | 0<1 pop(1) k=1 | push(0) | 3>0 push | 2<3 pop(3) k=0 | push after k=0



8 0 2 0 0 → rev → 0 0 2 0 8

Ans : 2 0 8.

Teacher's Signature

Code :-

```
String removeKdigits ( String S, int K) {

    Stack <int> st;
     int i = 0;
     int num;
    While( i < S.size()) {
        num = S[i] - '0';
      While(st.size() && st.pop() > num && K){
                st.pop();
                 K--;
            }
           st.push(num);
           i++;
    }

    While( K--) {          // After all element traverse
        st.pop();          and K greater than 0
     }                     then pop from top.

    String ans;
    char c;                // stack से क्या
    While ( st.size()) {   ईश्वर character
        c = '0' + st.top(); निकाल लो।
         st.pop();
        ans += c;
}
```

Teacher's Signature.....

```
        i = ans.size() - 1;
        while (i >= 0 && ans[i] == '0'){
            ans.pop_back();
            i--;
        }
```

// अगर Number के पीछे में zero है तो उसको Remove.

```
        reverse(ans.begin(), ans.end());

        if (ans.size() == 0)
            return "0";

        return ans;
    }
```

Problem : 2 Clumsy Factorial (LeetCode)

factorial : $6! = 6 * 5 * 4 * 3 * 2 * 1$

But in clumsy factorial :

$6! = 6 * 5 / 4 + 3 - 2 * 1$

priority of * & / > priority of + & -

Example :

$10! = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1$

first step : We perform only Multiplication & division.

Teacher's Signature.....

$10! = 90/8 + 7 - 6 * 5/4 + 3 - 2 * 1$

$= 11 + 7 - 30/4 + 3 - 2 * 1$

$= 11 + 7 - 7 + 3 - 2$

Now, we perform addition & substraction

$= 18 - 7 + 3 - 2$

$= 11 + 3 - 2$

$= 14 - 2$

$= 12$

→ Whenever we use stack, we use stack as a Memory.

→ Stack remembered the previous task that he had done.

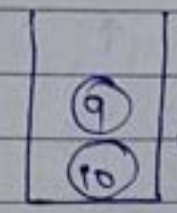
Solving using stack:

$10! = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1$

	*	/	+	-
	↓	↓	↓	↓
1 =	0	1	2	3

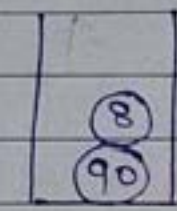
Teacher's Signature.....

10 9 8 7 6 5 4 3 2



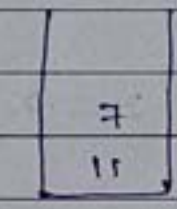
push push

pop both → 9 * 10 = 90 (store this in stack.



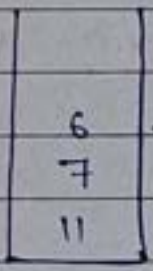
10 9 8 7 6 5 4 3 2 1

pop both → 90 / 8 = 11 (store in stack



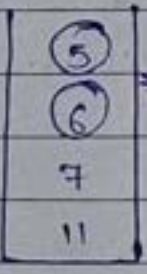
10 9 8 7 6 5 4 3 2 1

We can perform addition, we only perform here Multiplication & division



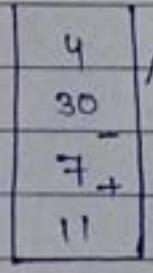
10 9 8 7 6 5 4 3 2 1

can't perform substraction



10 9 8 7 6 5 4 3 2 1

pop both → 5 x 6 = 30 push



10 9 8 7 6 5 4 3 2 1

poth both → 30 / 4 = 7 push.

Teacher's Signature

3	+
7	-
7	+
11	

10 9 8 7 6 5 4 3 2 1 ↑

Don't Do Add.

2	-
3	
7	
7	
11	

10 9 8 7 6 5 4 3 2 ↑ 1

Don't subtract.

1	*
2	
3	
7	
7	
11	

10 9 8 7 6 5 4 3 2 1 ↑

Pop both → 2 * 1 = 2 push.

2	-
3	+
7	-
7	+
11	

Now loop ends for ~~so~~ Multiplication & Division.

No, We pop() one by one element and copy to another stack.

2	
3	
7	
7	
11	

→ Copy in New Stack

11	
7	
7	
3	
2	

Teacher's Signature.....

pop 2 element & add.

11 + 7 = 18

18
7
3
2

pop 2 element & subtract

18 - 7 = 11

11
3
2

pop 2 element & add

11 + 3 = 14

14
2

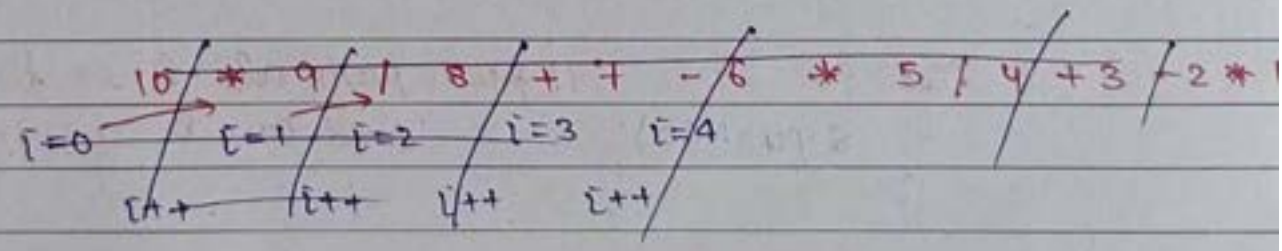
pop & Subtract

14 - 2 = (12)

↳ Required answer.

How to check which operation performed

	*	/	+	-
i =	0	1	2	3



10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1

i=0 i=1 i=2 i=3 (i=4)

i++ i++ i++ i++

↓

This is Not available.

Here we use Modulo.

$$[i = (i+1) \% 4]$$

Teacher's Signature.....

Code :

```
Stack <int> s;
int num, int i=0;
   s.push(n);   // if n = 10, then to goes to
   n--;         stack and loop start with 9
while(n) {
// is while loop में सिर्फ * & / करेंगे।
      if (i==0){  // 0 for Multiply
      num = s.top();
       s.pop();
       s.push(num * n);
     }

else if (i==1){          // for divide.
      num = s.top();
       s.pop();
       s.push(num / n);
     }

else {                   // ~~push~~ push as it is for
     s.push(n);          addition & substraction.
   }

   i = (i+1) % 4;        // i value changes (0,1,2,3)
    n--;
};

// Till Now Multiplication & division done.
```

Teacher's Signature.....

// Now, Copy in another stack

```
        stack<int> ans;
        while (s.size()) {
            ans.push(s.top());
            s.pop();
        };

    int sum = ans.top();
        ans.pop();
        bool flag = 0;
    while(ans.size()){

        if(flag == 0)
            sum += ans.top();
        else
            sum -= ans.top();
            ans.pop();
            flag = !flag;
    }

    return sum;
};
```

This code gives Time limit Exceed

We have to change some of our Approaches

Teacher's Signature.....

~~Problem 9: Max. Rectangle in a Histogram~~

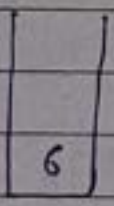
When Multiply divide Come we can Multiply & divide then store.

In addition, or substraction We put here (-n) for substraction so, we have perform only addition and no. new stack made.

Ex:

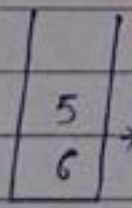
6! = 6 * 5 / 4 + 3 - 2 * 1

↑
i=0

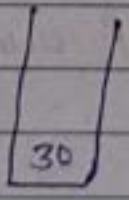


6 * 5 / 4 + 3 - 2 * 1

↑
i=0



* = 30



Teacher's Signature

6 5 4 3 2 1

↑

i = 1

7

30/4 = 7

6 5 4 3 2 1

↑

i = 2

3
7

6 5 4 3 2 1

↑

i = 3

put (-1) in stack

-2
3
7

6 5 4 3 2 1

↑

i = 1

-2
3
7

-2 * 1 = -2

After this, we have to only add.

-2 + 3 + 7

= (8)

Ans

Teacher's Signature

Code :

```
stack <int> s;
  s.push(n);
  n--;

int index = 0;
while(n > 0){
    if(index % 4 == 0){
        int num = s.top();
        s.pop();
        s.push(num * n);
    }

  else if(index % 4 == 1){
        int num = s.top();
        s.pop();
        s.push(num / n);
    }

  else if(index % 4 == 2){
        s.push(n);
    }

  else{
        s.push(-n);
    }

  index++;
  n--;
}
```

Teacher's Signature.....

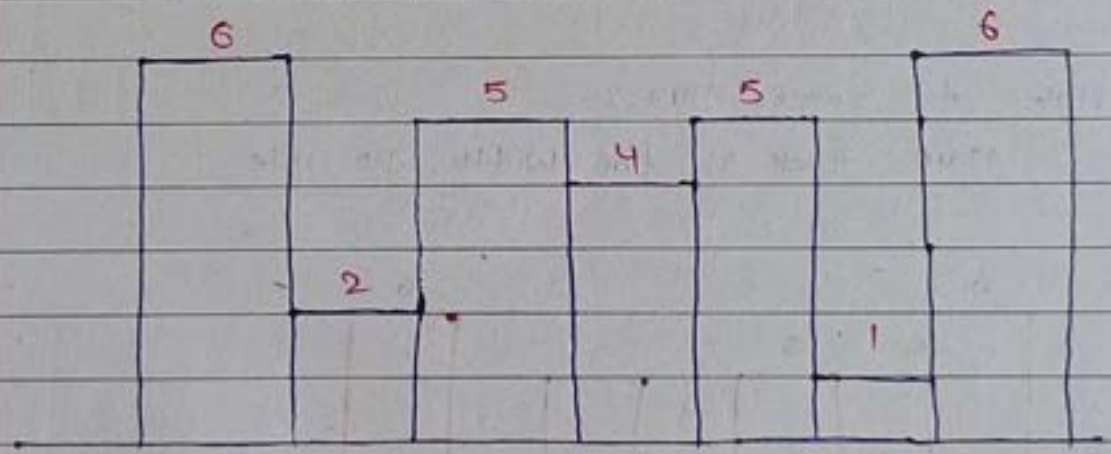
```
int sum = 0;
While (S.Size()){
        sum += S.top();
        S.pop();
}

return ~~ans~~ sum;
}
```

Problem 3: Maximum Rectangle area in a Histogram:

largest rectangular are possible

N = 7

arr[] = {6, 2, 5, 4, 5, 1, 6}



Approach 1:

Go and check every every number, how much it can acquire.

for 6 : height = 6 area = 6

Width = 1 Max = 6

Teacher's Signature.....

for 2

height = 2 area = 10

Width possible = 5 Max = 10

for 5

height = 5 area = 5

Width possible = 1 Max = 10

For 4

height = 4 area = 12

Width possible = 3 Max = 12

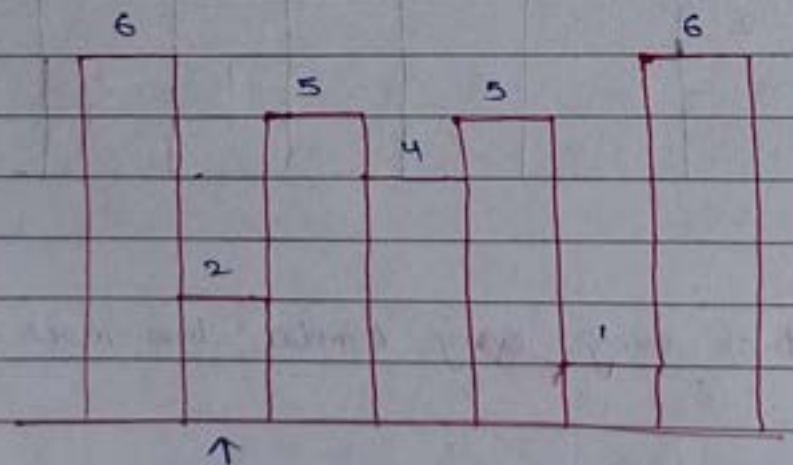
For 5

height = 5 area = 5

Width possible = 1 Max = 12

How to solve this :

Main task : find Width possible.



For 2

How to find Widtth possible.

Teacher's Signature

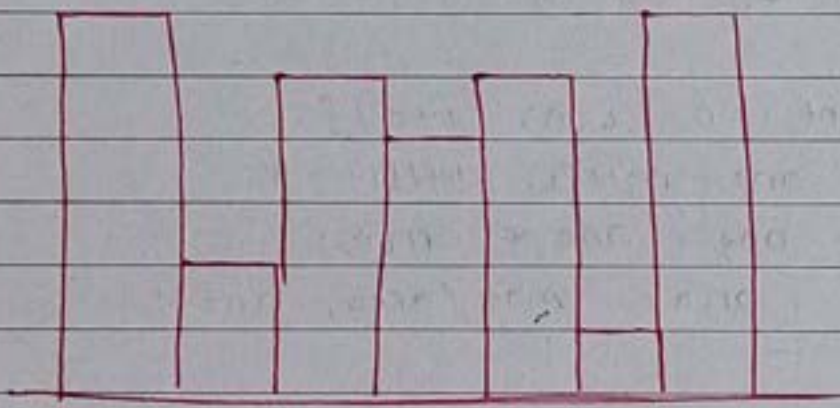
We know that,

Width possible = 5

It can find next min^m in both direction, in right and left.

- next smallest right
- next smallest left

We can find for all and then check :



Height	6	2	5	4	5	1	6
Index	0	1	2	3	4	5	6
Right	1	5	3	1	1	7	7
left	-1	-1	12	1	3	-1	5

> This is index.

Maximum possible

breadth = Right - left - 1;

Area = height[i] * breadth.

Teacher's Signature.....

Code:

```
long long getMaxArea ( log long arr[], int n ) {

    int * right  = new int [n];
    int * left  =  new int [n];

    push_right (arr, n, right);
    push_left (arr, n, left);

    long long area = 0;
    long long ans;

    for(int i=0;  i<n;  i++) {
        ans = right[i] - left[i] - 1;
        ans = ans * arr[i];
        area =  max (area, ans);
    }
    return area;
}

Void push_left ( long long arr[], int n, int left[]) {
    stack <int> s;
    int i = n-1;
    While ( i >= 0) {
        if ( s.size() == 0){
            s.push (i);
        }
        else {
            if ( arr[i] > arr [s.top()]){
```

```
                s.push(i);
            }
            else{
                while(s.size() && arr[i] < arr[s.top()]){
                    left[s.top()] = i;
                    s.pop();
                }
                s.push(i);
            }
        }
        i--;
    }
    while (s.size()){
        left[s.top()] = -1;
        s.pop();
    }
}

void push_right (long long arr[], int n, int right[]){
    stack<int> s;
    int i = 0;
    while (i<n){
        if (s.size()==0)
            s.push(i)
        else{
            if (arr[i] >=arr[s.top()])
            ~~s.to~~ s.push(i);
            else{
                while(s.size() && arr[i] < arr[s.top()]){
                    right[s.top()] = i;
                    s.pop();
                }
```

Teacher's Signature.....

```
            s.push(i);
        }
    }
    i++;
}
while(s.size()){
    right[s.top()] = n;
    s.pop();
}
}
```

Problem 4 : Max Rectangle :

0	1	1	0
1	1	1	1
1	1	1	1
1	1	0	0

find the Maxm Rectangle of 1's.

We will check here row by row and find maxm area.

Row 1 :

0	1	1	0

heights : 0 1 1 0

Max = 2 Area = 2

Teacher's Signature

Row 2 :

	0	1	1	0
	1	1	1	01
Heights :	1	2	2	01

Area = 4 Max = 4

Row 3 :

	0	1	1	0
	1	1	1	1
	1	1	1	1
Heights :	2	3	3	2

Area = 8 Max = 8

Row 4 :

	0	1	1	0
	1	1	1	1
	1	1	1	1
	1	1	0	0
Hight =	3	4	0	0

zero because in last Row is 0 so, be without how we make rectangle.

Area = 6 Max = 8

↳ return this

Teacher's Signature.....

Code :

```
int maxArea ( int M[MAX][MAX] , int n, int m) {
        int heights[m] = {0};
        int maxArea = 0;

    for (int i=0; i<n; i++) {
        for (int j=0; j<m; j++) {
            if ( M[i][j] == 0) {
                heights[j] = 0;
            }
            else {
                heights[j] += M[i][j];
            }
        }
        maxArea = max (maxArea , getMaxArea (heights, m));
    }
    return maxArea;
}

int getMaxArea (int heights[], int n ) {
        stack <int> st;
        int maxArea = 0;
        int i = 0;

    while ( i < n ) {
        if (st.empty() || heights[i] >= heights[st.top()]) {
            st.push(i);
            i++;
        } else {
```

Teacher's Signature

```
            int tp = st.top();
            st.pop();
            int area = heights[tp] * (st.empty() ? i : (i - st.top() - 1
            maxArea = max(maxArea, area);
        }
    }

    while (!st.empty()) {
        int tp = st.top();
        st.pop();

        int area = heights[tp] * (st.empty() ? i : (i - st.top() - 1));
        maxArea = max(maxArea, area);
    }
    return maxArea;
}
```

problem 5: The celebrity problem:

Celebrity : known by all
but celebrity don't know anyone.
in party.

find celebrity present or not in party.

1 → i^{th} know j^{th}

M[i][j] → always 0.

Teacher's Signature

N = 3

M[][] =	0	1	2
0	0	1	0
1	0	0	0
2	0	1	0

person 0 → knows 1 can't be celebrity

~~pose~~

person 1 →

person 2 → knows 1 can't be celebrity

person 1 May be celebrity.

check row of 1 → all 0]
check colum of 1 → all 1] person 1 is celebrity.

* How to solve :

celebrity ke row me sare 0
↳ then May be celebrity.

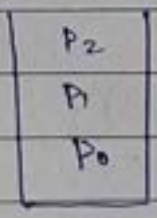
check Column All are 1 except self
↳ then it celebrity.

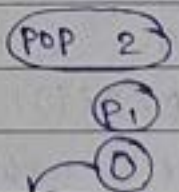
Teacher's Signature

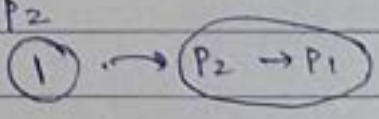
first	second	
0	0	→ Both don't know other → Not celebrity.
0	1	→ second Not celebrity
1	0	→ first Not celebrity
1	1	→ Not celebrity.

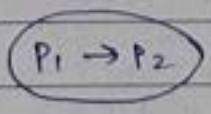
	P_0	P_1	P_2
P_0	0	1	0
P_1	0	0	(0)
P_2	0	(1)	0

put all person in stack









P_2 can't be celebrity

P_1, may be celebrity

push (P_1)

Teacher's Signature

P_1
P_0

	P_0	P_1
	1	0

P_0 can't be celebrity

P_1 may be celebrity.

push(P_1)

P_1

check the row of P_1

↳ all zero

check the Col-n of P_1

↳ all 1 except self.

Code:

```
int celebrity (vector<vector<int>> &M, int n) {
        stack<int> s;
    for (int i = 0; i<n; i++) {
        s.push(i);
    int first, second;

    while (s.size() > 1) {
        first = s.top();
        s.pop();
        second = s.top();
        s.pop;
```

Teacher's Signature

```
            if (M[first][second] && !M[second][first]){
                    s.push(second);
            }
            else if (M[second][first] && !M[first][second]){
                    s.push(first);
            }
        }
    }

    if (s.size() == 0){
        return -1;
    }

    int check = s.top();
    int count_row = 0, count_col = 0;

    for (int i = 0; i<n; i++){
        count_col += M[i][check];
        count_row += M[check][i];
    }

    if (count_row == 0 && count_col == n-1){
        return check;
    }
    else{
        return -1;
    }
};
```

Teacher's Signature.....

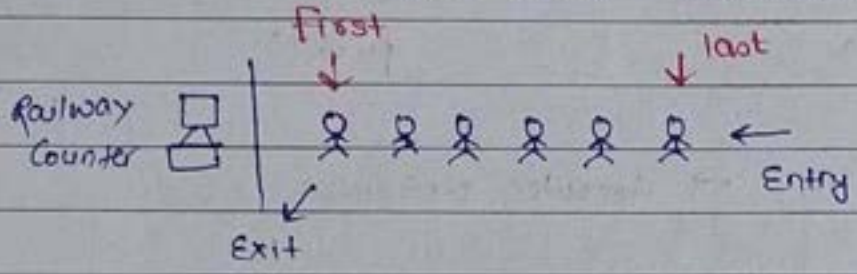
Lecture : 38

Queue : Basic

Queue → line (Any)

↓

Ex: Bank, Ticket Counter



Follow FIFO (First in first out)

Example:

You text you Friend on whatshapp:

Hii

Hello

How are you

I am coming.

You send the 4 message:

→ First this message go into server and stored in a queue.

→ Then the message stored first in queue can go first to your friend whatsApp

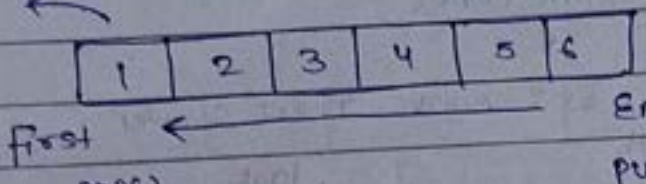
Teacher's Signature

* How to create a queue:

Queue creation

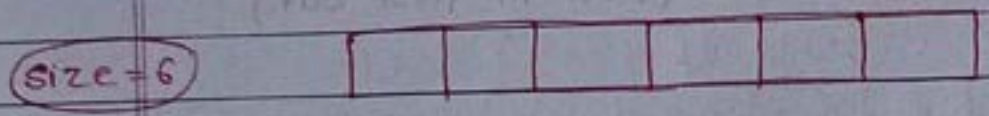
↳ By Array

↳ By Linked list



used in → Computer Networks

① Creation by array:



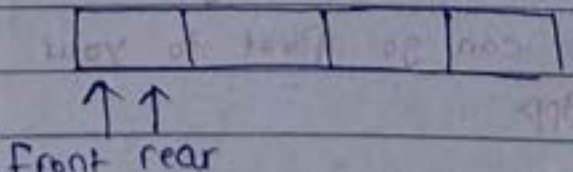
Two pointer

↳ front : from where we have to pop

↳ rear : from where we have push.

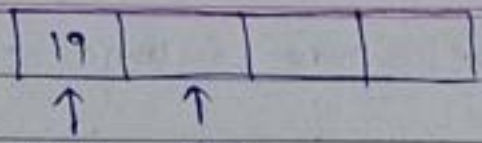
front always on first position.

When Queue is empty the front & rear both on first place.

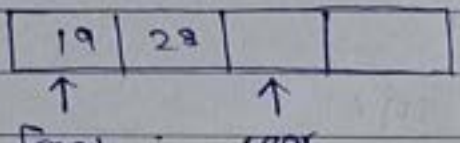


push(19)

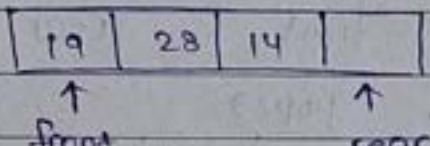
Teacher's Signature.....



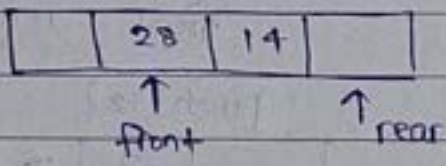
push(28)



Push(14)



pop()



code for push:

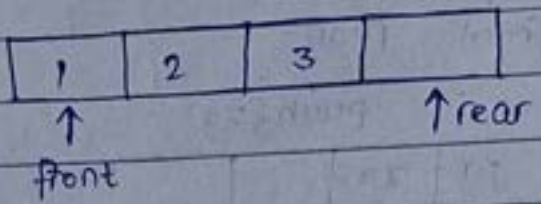
```
void push (int x) {
    if (rear == size){
        cout << "Queue is full";
        return;
    }
    arr[rear] = x;
    rear++;
}
```

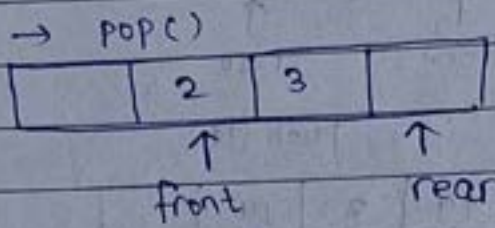
code for pop:

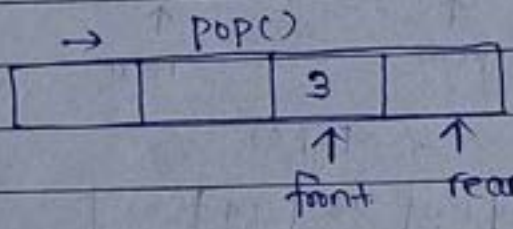
```
void pop() {
    if (front == rear) {
        front = rear = 0;
        cout << "Queue is empty";
        return;
    }
    front++;
}
```

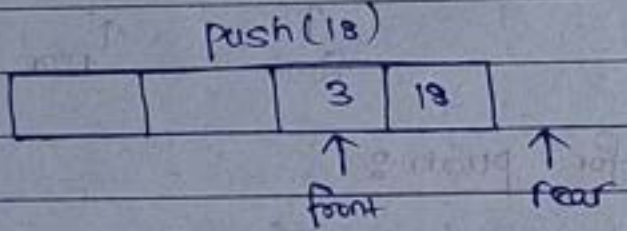
Teacher's Signature.....

In this queue, there is some problem :-







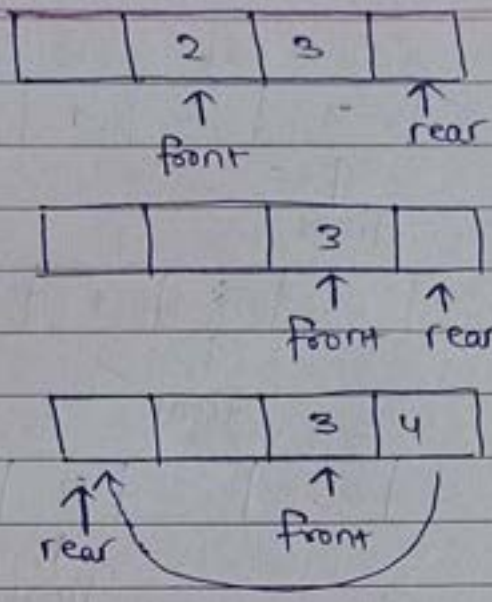


Now, if we want to add More element we can't able to add.

So, We make Circular Queue

अगर rear last तक आ जाएगा उसके बाद rear को first Block में भेज देंगे।

Ex:



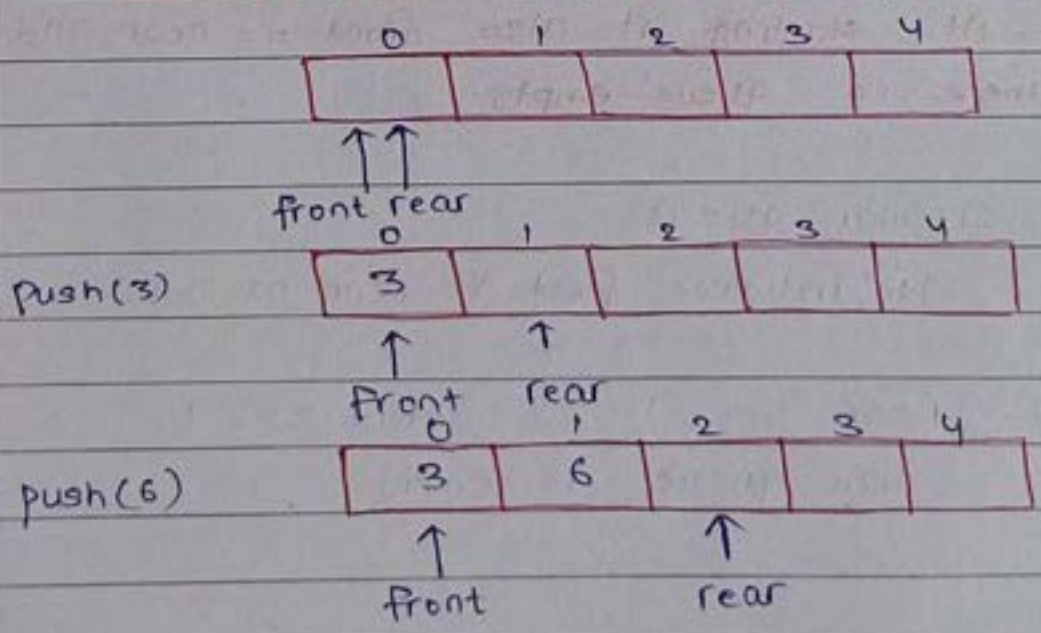
Condition:

```
rear = (rear + 1) %5;
```

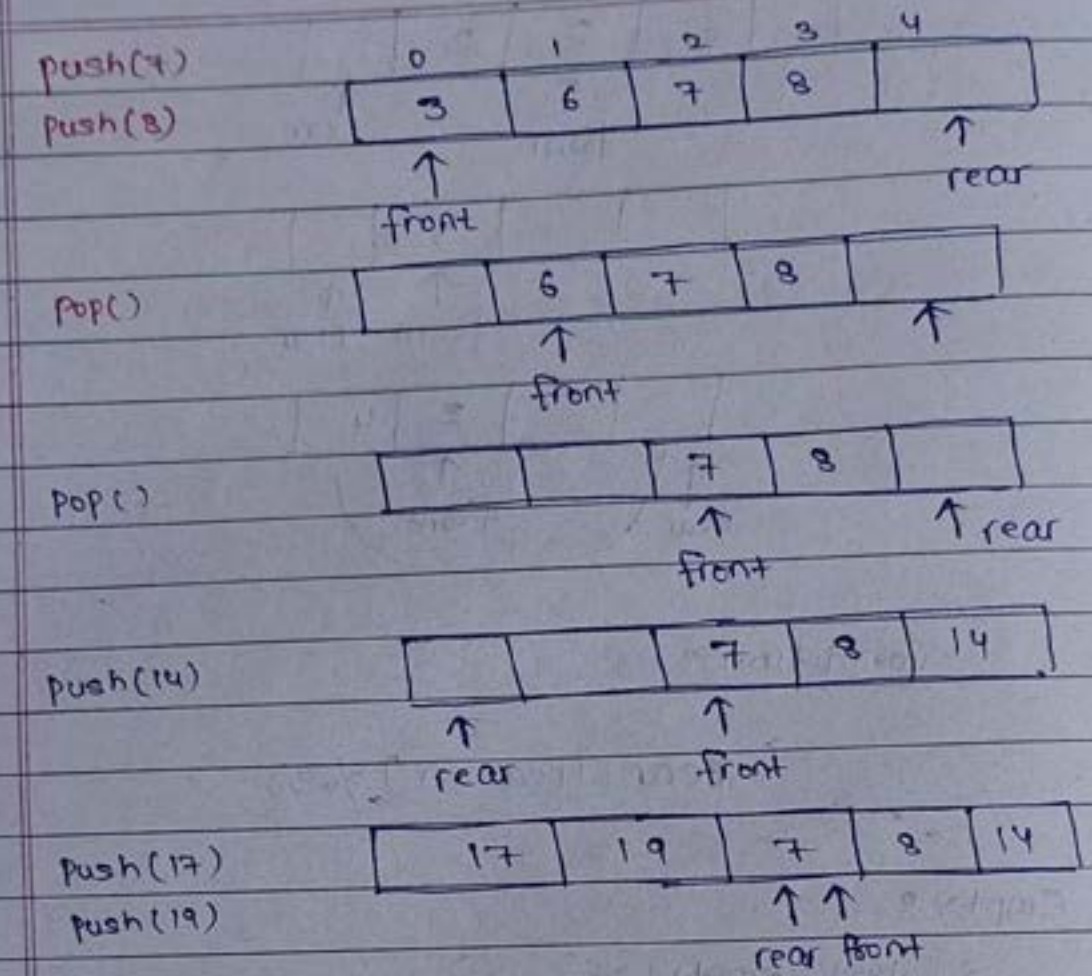
Empty:

```
bool empty(){
    return front == rear;
}
```

* Circular Queue: (How to implement)



Teacher's Signature.....



condition:

```
if (front == rear)
    queue full;
```

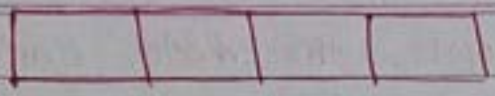
→ But, At starting, it also front == rear and there is queue empty.

In Circular queue,

We initialise first & rear as -1.

if rear == -1 & front == -1
then queue is empty.

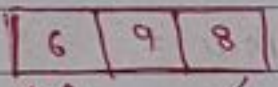
Teacher's Signature.....



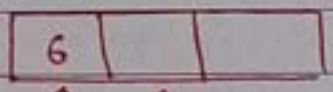
front rear.

Code:

```
Void push(int x) {
    if ((empty())) {
        front = 0;
        rear = 0;
        arr[front] = x;
    }
```




```
    else if (full()) {
        cout << "Queue is full";
    }
    else {
        rear = (rear + 1) % 5;
        arr[rear] = ~~data~~ x;
        return;
    }
```



Code: for pop():

```
Void pop() {
    if (empty()) {
        cout << "Queue is Empty";
    }
    else if (front == rear) {
        cout << "Element popped" << arr[front];
        front = rear = -1;
        return;
    }
    else {
        cout << "Element popped" << arr[front];
        front = (front + 1) % size;
        return;
    }
}
```

Teacher's Signature.....

When we pop(), the data can't be deleted only foont pointer is moving here to there.

Eg: Data recover:

Data is not deleted, it only move the pointer.

* Empty()

```
bool empty(){
    return front == -1 && rear == -1;
}
```

* full():

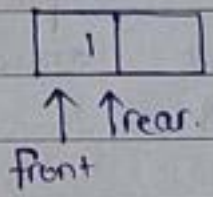
```
bool full(){
    return front == (rear+1) % size;
```

Teacher's Signature

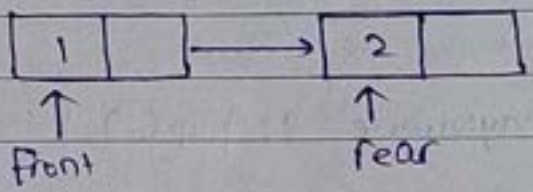
* Queue by UnicedList :

front = NULL;
rear = NULL; } → Queue empty

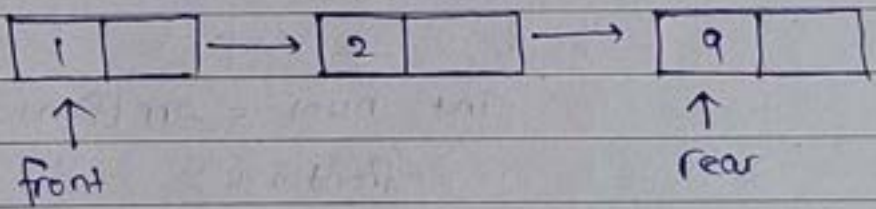
push(1)



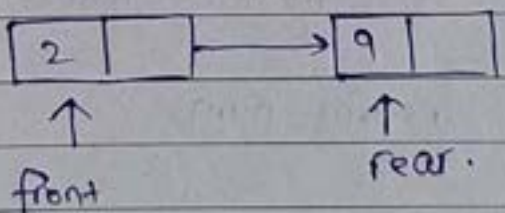
push(2)



push (9)



pop( )



Teacher's Signature.....

Problem 1: Implement Queue using array

Code:

```
Void MyQueue :: push(int x){
                          → size of array is mentioned in class.
    if (rear == 100005)
        return;

    arr[rear++] = x;
}

int MyQueue :: pop() {

    if (front == rear)
        return -1;

    int num = arr[front];
    front++;

    if (front == rear)
        front = rear = 0;

    return num;
}
```

Teacher's Signature.....

problem 2: Implement Queue using Linked List

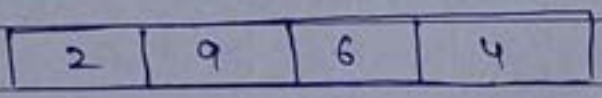
```
Void MyQueue :: push (int x) {

    if (! front) {    // if Queue is empty.
        front = new QueueNode(x);
        rear = front;
        return;
    }
    else {
        rear → next = new QueueNode (x);
        rear = rear → next;
        ~~rear;~~
        return;
    }
}

int MyQueue : pop () {
    if (! front)
        return -1;

    else {
        int data = front → data;
        if (front == rear)
            front = rear = NULL;
        else
            Front = front → next;

        return data;
    }
}
```

Teacher's Signature.....

* Queue By STL:

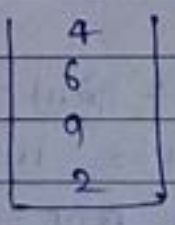
```
#include <Queue>
    int main() {
        queue<int> q;
            q.push(1);
            q.push(11);
            q.push(111);
            q.push(1111);
        cout << q.front() << endl;
            q.pop();
        return 0;
    }
```

* problem 3: Queue Reversal

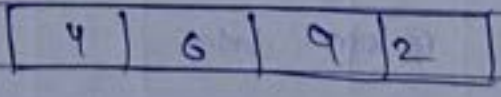


How to reverse this?

Store this in stack:-



Then one by one push into queue



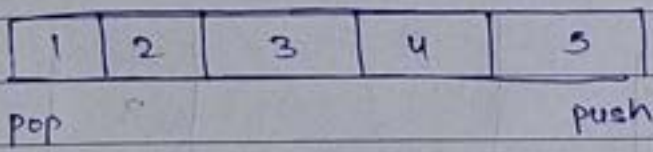
You get the answer:-

Teacher's Signature

code:

```
queue <int> rev (queue <int> q) {

    stack <int> s;
    while (q.size()) {  // copy from queue to
        s.push(q.front());   stack
        q.pop();
    }

    while (s.size()) {  // copy from stack to
        q.push(s.top());   queue.
        s.pop();
    }
  return q;
}
```

* problem 4 : Queue using two stack :



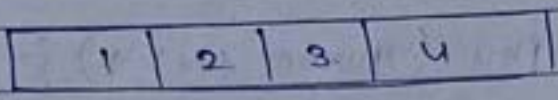
FIFO

How we can implement this queue using two stack?

Let's see.

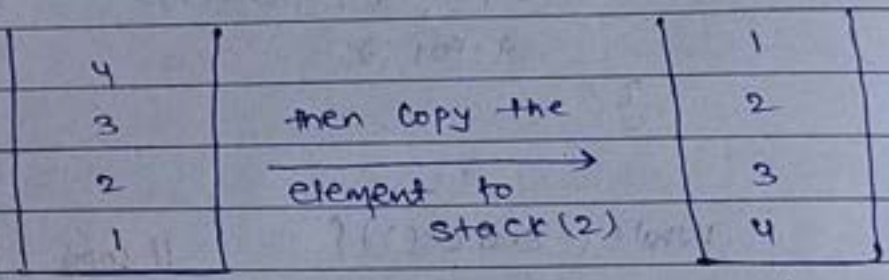
Teacher's Signature.....

For pop:



pop()
then 1 deleted

How we can do it from 2 stack.

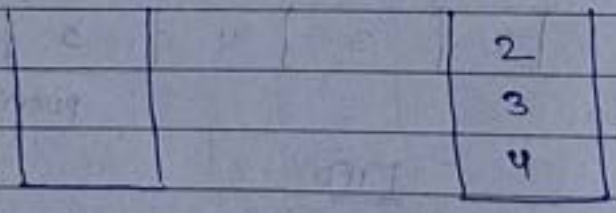


copy in stack 1.

Now pop()
1 → deleted.

pop is performed perfectly using two stack.

For push:



Now we have to push 7.
push it in stack 1.

Teacher's Signature

If We Want to pop then pop it from stack 2.

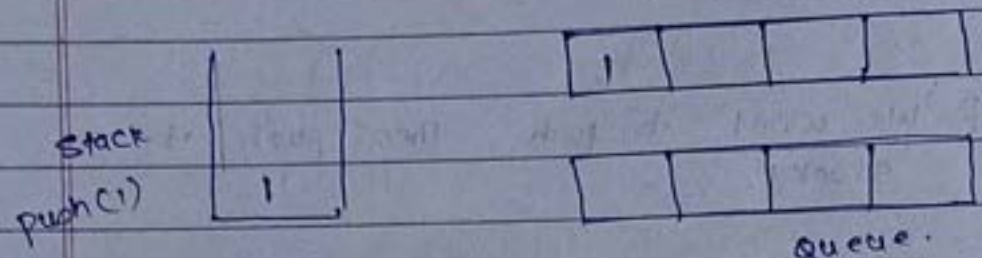
If we want to push then push it in stack1.

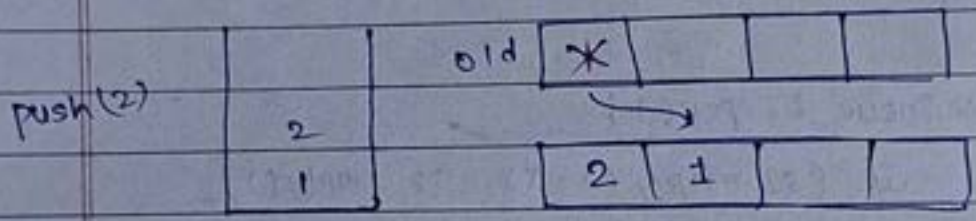
Code :

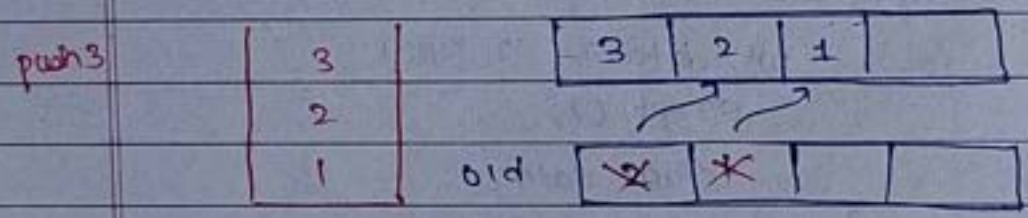
```
Void StackQueue :: pop() {
        if (s1.empty() && s2.empty())
            return -1;
        if (s2.size()) {
            int data = s2.top();
            s2.pop();
            return data;
        }
        while (s1.size()) {
            s2.push(s1.top());
            s1.pop();
        }
        int data = s2.top();
        s2.pop();
        return data;
}

Void StackQueue :: push(int x) {
        s1.push(x);
}
```

Teacher's Signature.....

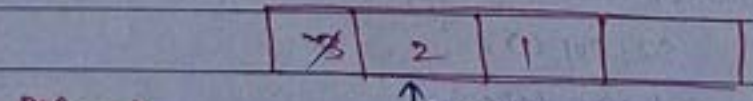
* problem - 5 stack using two Queues







Push always done in empty Queue and Copy all the element from another Queue.



remove (3)
↳ first

Teacher's Signature.....

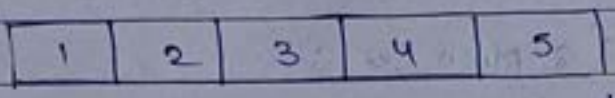
Code:

```
Void QueueStack :: push (int x) {

        q2.push (x);

        While (!q1.empty ()) {
            q2.push (q1.front ());
            q1.pop ();
        }

        swap (q1, q2);
}

int QueueStack :: pop () {

        if (q1.empty ())
            return -1;

        int topElement = q1.front ();
            q1.pop ();

        return topelement;
}
```

Teacher's Signature.....

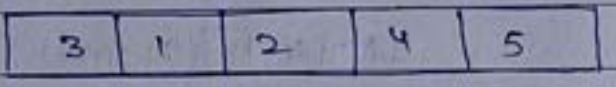
Problem : 6

Reverse First K elements of Queue :

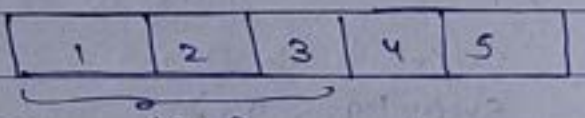


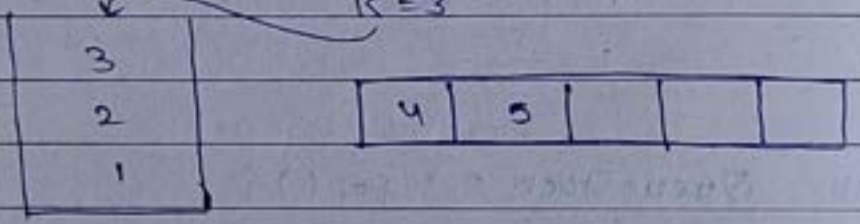
k = 3

output :

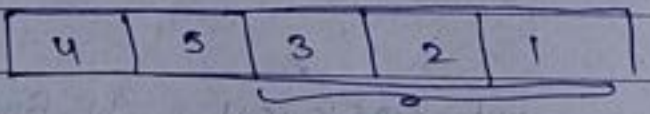


k = 3



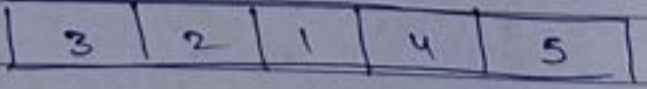


Now Copy in Queue



remaining = 5 - 3 = 2

pop & paste back



output =

Teacher's Signature

Code :

```
queue <int> modifyQueue (queue <int> q, int k) {
    int a = k;
    stack <int> s;

    while (k) {
        s.push (q.front ());
        q.pop ();
        k--;
    }

    while (s.size()) {
        q.push (s.top());
        s.pop();
    }

    int remaining = q.size() - a;

    for (int i = 0; i < remaining; i++) {
        int num = q.front();
        q.pop();
        q.push(num);
    }

    return q;
}
```

Teacher's Signature

Lecture - 39

(Queue problems)

problem : 1 → Circular Tour

→ A circle

→ N petrol pump on the circle

You have two array :

1) Amount of petrol that every petrol pump has.

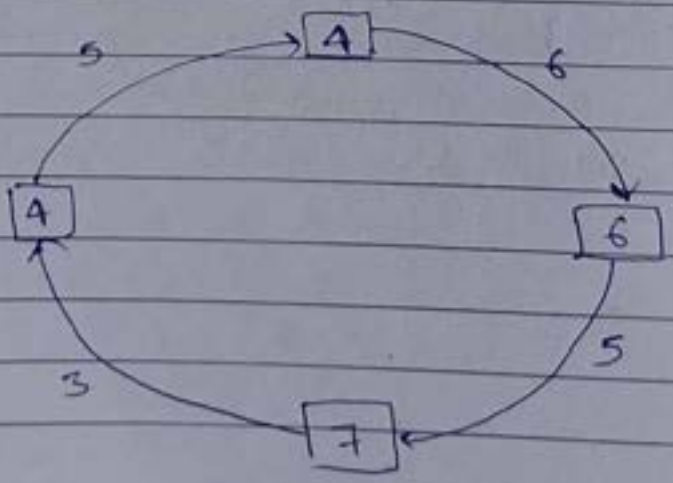
2) Distance from that petrol pump to next petrol pump.

1 litre petrol ⟶ 1 unit of distance

→ N = 4

petrol : 4 6 7 4

Distance : 6 5 3 5

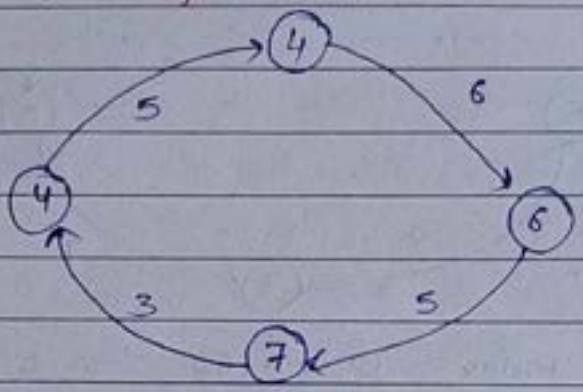


Teacher's Signature.....

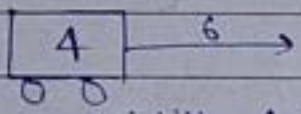
Return the point where it can start and Complete all the petrol pump.

otherwise return -1.

Approach 1 (Brute force Approach)

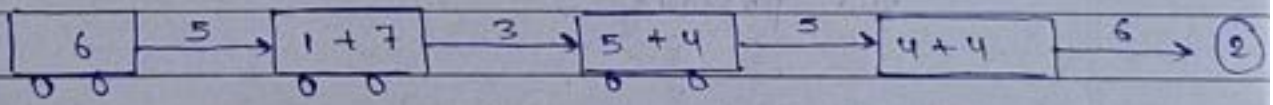


Start with index → 0 (4) ✗



With 4 litere of petrol we can't cover 6 unit distance
Not possible.

Start with index → 1 (6)



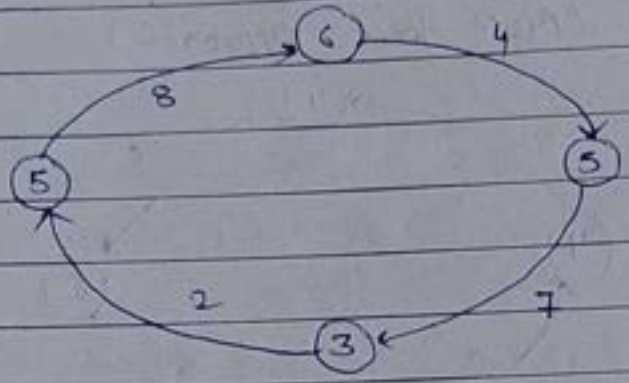
When we start from index 1 where 6 litre of petrol, then we can traverse full circle.

return (1) → index.

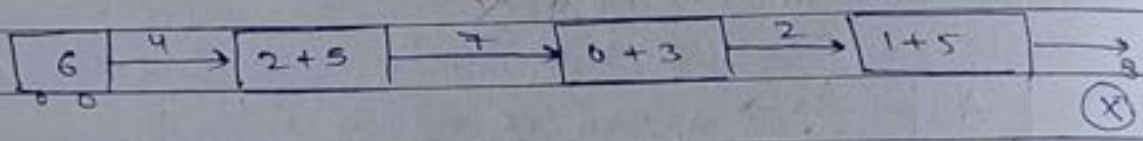
Teacher's Signature

Example-2 :

petrol	6	5	3	5
Distance	4	7	2	8

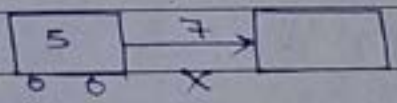


Start With index 0 → (6)



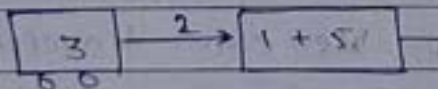
Not possible

Start With index 1 → (3)



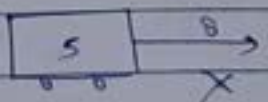
Not possible.

Start With index 2 → (3)



Not possible.

Start With index 3 → (5)



Not possible

return -1

Teacher's Signature.....

In this approach we check from all petrol pump and go to all other petrol pump.

T.C = $O(n^2)$

Optimal solution

Petrol	6	6	5	7	11
Distance	4	7	10	5	9
	6-4 = 2	2+6=7, 8-7 = 1	1+5=6, 6-10 = -4		

If we start from index 0 then not possible.

the next starting index is where we fail. we start one index after the failed index.

start

petrol	6	5	5	7	11
Distance	4	7	10	5	9
				7-5 = 2	2+11 = 13, 13-9 = 4

(pointer = (pointer+1) % N)

6	6	5	7	11
4	7	10	5	9
4+6 = 10, 10-4 = 6	6+6 = 12, 12-7 = 5	5+5 = 10, 10-10 = 0		

Teacher's Signature.....

In this case we reached all petrol pump if we start from index 3.

T.C = O(n)

Implementation:

front ↓ rear ↓

petrol	6	5	5	7	12
Distance	4	7	10	5	9

6-4 = 2

put two pointer front & rear on index 0 of petrol.

~~check~~ ~~petrol[f] + petrol[0~~

→ check petrol[0] - Distance[0] > 0

petrol = 2

→ rear = rear + 1

front ↓ rear ↓

petrol	6	5	5	7	12
Distance	4	7	10	5	9

6-4 = 2 2+5 = 7

7-7 = 0

Petrol = petrol + petrol[rear] - Distance[rear];

Petrol = 0

(0 >= 0)

rear++

Teacher's Signature.....

front ↓ rear ↓

petrol	6	5	5	7	12
Distance	4	7	10	5	9
	2	0	0+5=5		

5-10

= -5

(-5 < 0) → front = rear + 1;

rear = front;

petrol = 0

rear ↓ front ↓

petrol	6	5	5	7	12
distance	4	7	10	5	9

7 - 5

= 2

petrol = (2 > 0) ✓

rear++

front ↓ rear ↓

6	5	5	7	12
4	7	10	5	9
			2	2+12=14

14 - 9 = 5

petrol = (5 > 0)

rear++;

rear = (rear+1)%n;

rear ↓ front ↓

6	5	5	7	12
4	7	10	5	9
5+6=11				5

11 - 4 = 7

petrol = 7 > 0 rear++;

rear = (rear+1)%n;

Teacher's Signature.....

rear ↓ front ↓

6 5 5 7 12

4 7 10 5 9

7 7+5 = 12, 12-7=5 (5>0) rear++;

rear ↓ front ↓

6 5 5 7 12

4 7 10 5 9

5 5+5=10, 10-10 = 0

(0>=0) rear++;

rear ↓ front ↓

6 5 5 7 12

4 7 10 5 9

(front == rear)
return front;

Code:

```
int tour (petrolpump p[], int n) {
    int front = 0, rear = 0;
    int petrol = 0;

    while (front < n) {
    petrol += p[rear].petrol - p[rear].distance;
    if (petrol < 0) {
        if (rear < front)
            return -1;
```

Teacher's Signature.....

```
            petrol = 0;
            front = rear + 1;
            rear = front;
        }
        else{
            rear = (rear + 1) % n;
            if (front == rear)
                return front;
        }
    }
    return -1;
}
```

Appsoach 3:

petrol	6	5	5	7	12
Distance	4	7	10	5	9
	6-4	2+5-7	0+5-10	0+7-5	2+12-9
	= 2	= 0	= -5	= 2	= 5
Defict :			-5		
Balance :	~~2~~	~~0~~	~~-5~~ 0	~~2~~	5

At last we check

balance + deficient >= 0

↳ true

↳ then return start

↳ false

↳ return -1

Teacher's Signature.....

Page No.:
Date: / /

start station पर balance 0 से कम हो जाएगा
उसके एक आगे start को point करें
2)।

Code:

```
int deficit = 0, balance = 0, start = 0;

for (int i = 0; i<n; i++) {
    balance += p[i].petrol - p[i].distance;
        if (balance < 0) {
            deficit += balance;
            balance = 0;
            start = i+1;
        }
    }

    if (balance + deficit >= 0)
        return start;

return -1;
}
```

Teacher's Signature.....

Dequeue

Updated Version of Queue
push and pop from both side.

Dequeue operation:

```
deque<int> d;

d.push_back(9);
d.push_front(11);
d.pop_back();
d.pop_front();
d.front();
d.back();
```

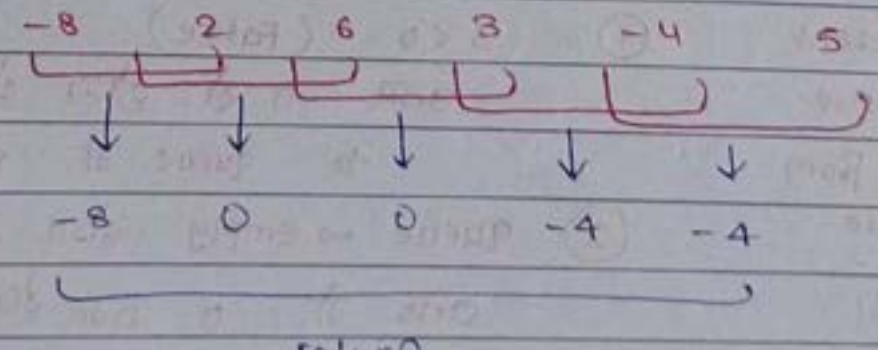
> for getting element.

problem 2: First Negative Integer in every window of Size K.

N = 5

A[] = {-8, 2, 6, 3, -4, 5}

(k=2) ↳ window size



O(n*k)

return

Teacher's Signature.....

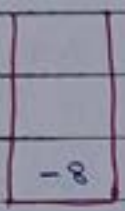
Better Approach

Size = 8

K=3

-8	2	6	3	-7	-11	13	14

from the 1st Window
put all the negative number
in ~~stack~~ queue.



so for 1st Window → 1st Negative is
q.front()
↳ -8

Ans: -8

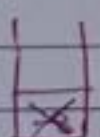
Size of Ans = size of A[] - K + 1
= 8 - 3 + 1
= 6
↳ 6 Window possible.

From 2nd Window:

→

-8	2	6	3	-7	-11	13	14

check -8 is present in queue.
pop it from queue.
q.pop()



यहाँ 3 चीज देखना है।
① 3 < 0 (False)
अगर 0 से छोटा होगा
to queue में push कर देंगे।
② queue → empty होगा तो।
ans में 0 add होगा।

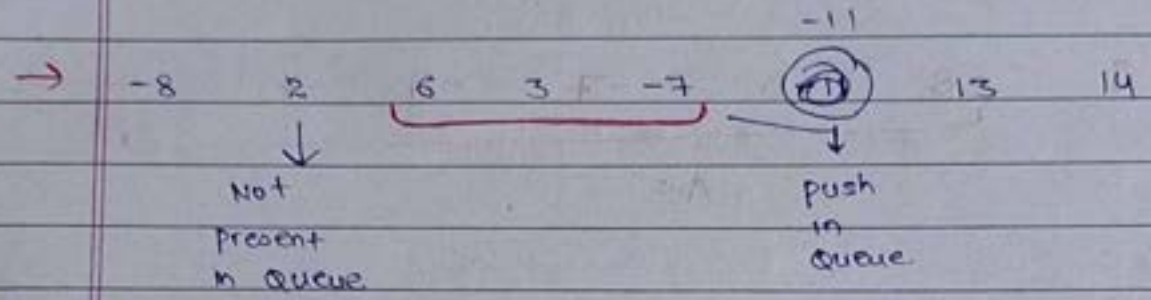
Teacher's Signature

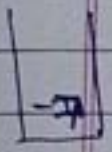
(3) q उसे empty नहीं है तो

ans में ~~q.push_back~~

q.front() और push करो।

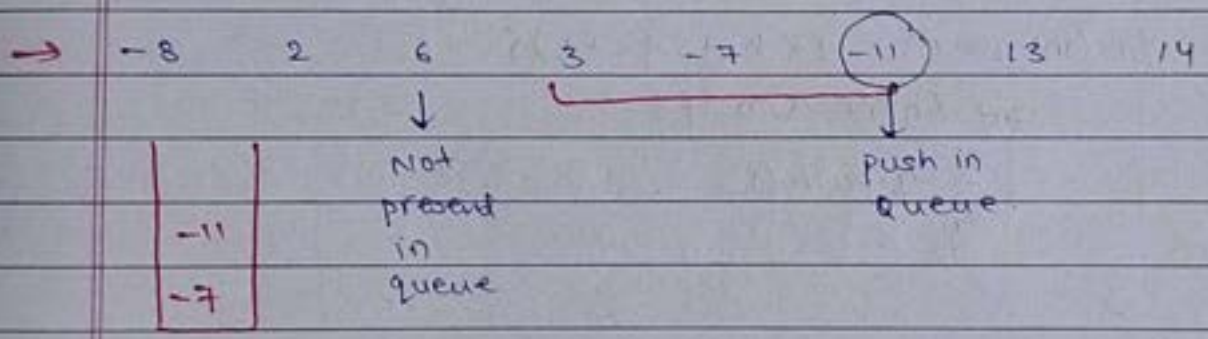
Ans: -8 0



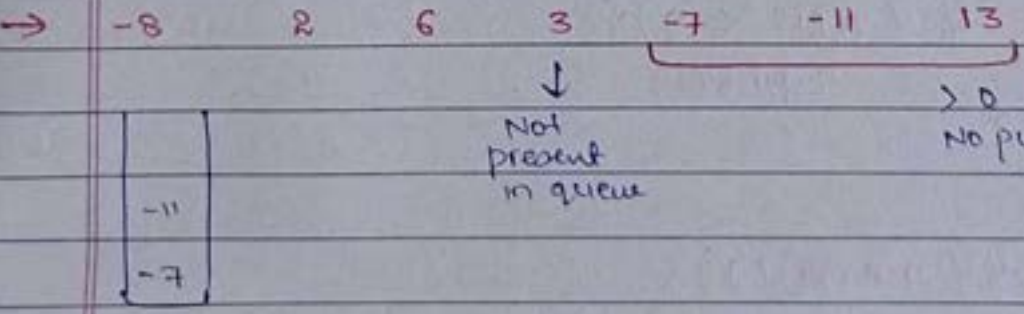


As: -8 0 -7

q.front()



Ans: -8 0 -7 -7



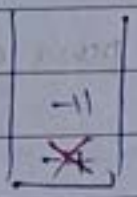
Ans: -8 0 -7 -7 -7

Teacher's Signature

→ -8 2 6 3 -7 -11 13 14

↓ pop from queue

↓ >0 Not push



Ans: -8 6 -7 -7 -7 -11

Ans

Code:

```
vector < long long int > ans;
vector < long long int > q;

for (int i = 0;  i < k-1;  i++){
    if (A[i] < 0){
        q.push(i);
    }
}

for (int i = k-1;  i < N;  i++){
    if (A[i] < 0){
        q.push(i);
    }

if (q.empty()){
    ans.push_back(0);
}
else {
    ans.push_back(A[q.front()]);
}
```

Teacher's Signature

```
        if((!q.empty()) && q.front() <= i-k+1){
            q.pop();
        }
    }
    return ans;
}
```

Problem 3: Sliding Window Maximum (InterviewBit)

3 7 6 8 11 5 -2 4

Find Maximum from Window

k = 3

→ 3 7 6 8 11 5 -2 4

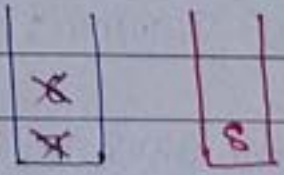
Max = 7

Ans:

→ 3 7 6 8 11 5 -2 4

3 → Not present in Queue

8 → greater than 6 & > 7

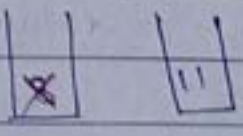


Ans: 7 8

→ 3 7 6 8 11 5 -2 4

7 → Not present

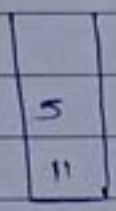
11 → > 8



Ans: 7 8 11

Teacher's Signature

→ 3 7 6 8 11 5 -2 4

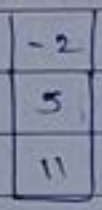


Not present

< 11

Ans: 7 8 11 11

→ 3 7 6 8 11 5 -2 4

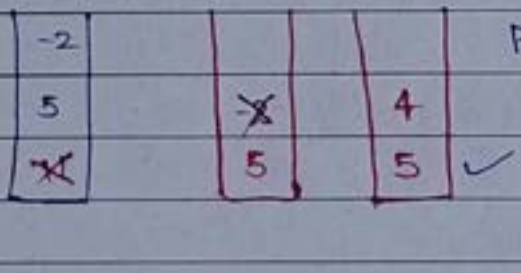


Not present

< 11

Ans: 7 8 11 11 11

→ 3 7 6 8 11 5 -2 4



present

pop()

4 > 2

Ans: 7 8 11 11 11 5

Ans:

Code:

```
Vector<int> Solution::slidingMaximum(const vector<int>&A,
                                     int B){
    Vector<int> v;
    if(A.size()==0)
        return v;

    dequeue<int> d;
```

Teacher's Signature

```
    int i=0, n = A.size();
    for(i=0; i<n && i<B; i++){
        While (d.size() && A[d.back()] < A[i])
            d.pop_back();

        d.push_back(i);
    };

    V.push_back( A[d.front()]);
    if (B >= A.size())
        return V;

    for( ; i<n; i++) {
        if ((i-B) == d.front())
            d.pop_front();

        While ( d.size() && A[d.back()] < A[i])
            d.pop_back();
            d.push_back(i);
        V.push_back (A[d.front()]);
    }

    return V;
}
```

Teacher's Signature

Problem 4: Minimum Number of k Consecutive Bit Flips

Example: nums = [0, 1, 0]

k = 1

0 1 0

1 1 0

1 1 1

Flip required = 2

Example:

num = [0, 1, 0, 0, 1, 0, 1]

k = 3

0 1 0 0 1 0 1

1 0 1 0 1 0 1

1 1 0 1 1 0 1

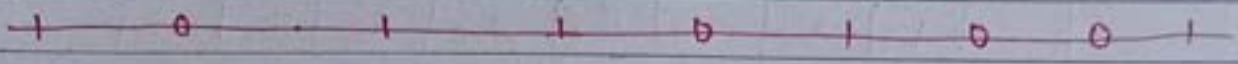
1 1 1 0 0 0 1

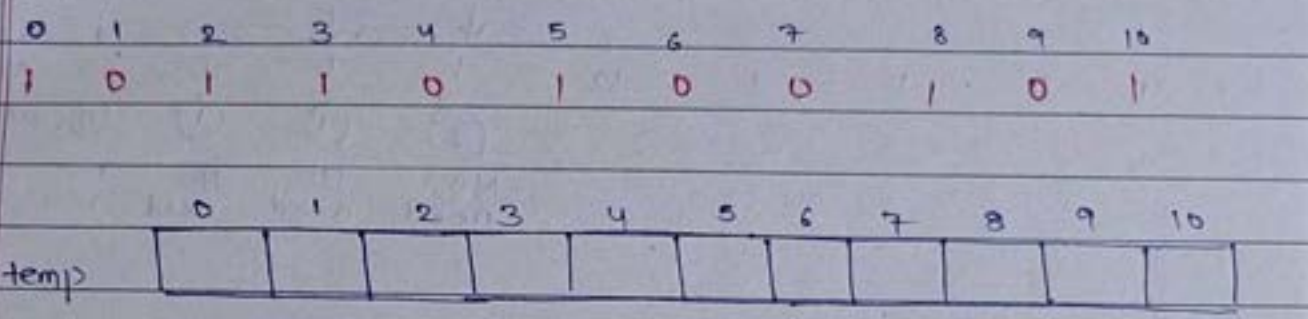
1 1 1 1 1 1 1

Flip required = 4

Teacher's Signature.....

Find zero → पहले zero find करना है फिर flip करना है।



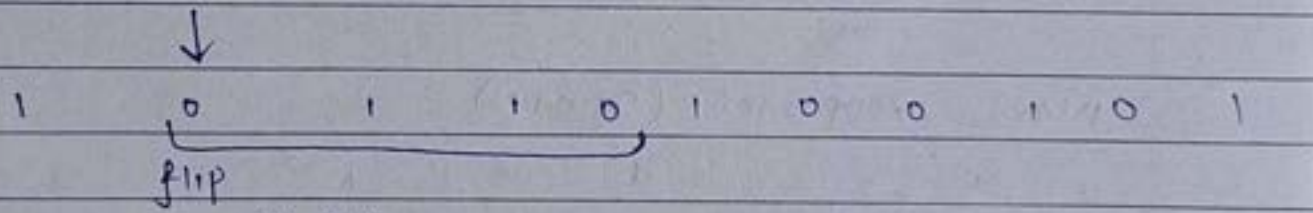


find the zero

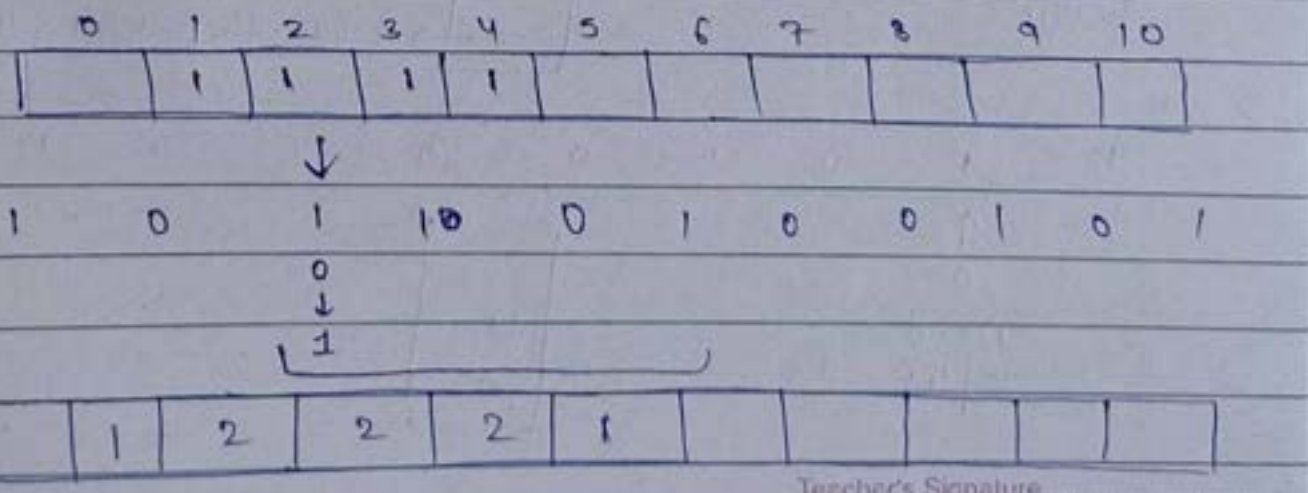
Take Sub array of size 10.

→ k-size subarray return - 1

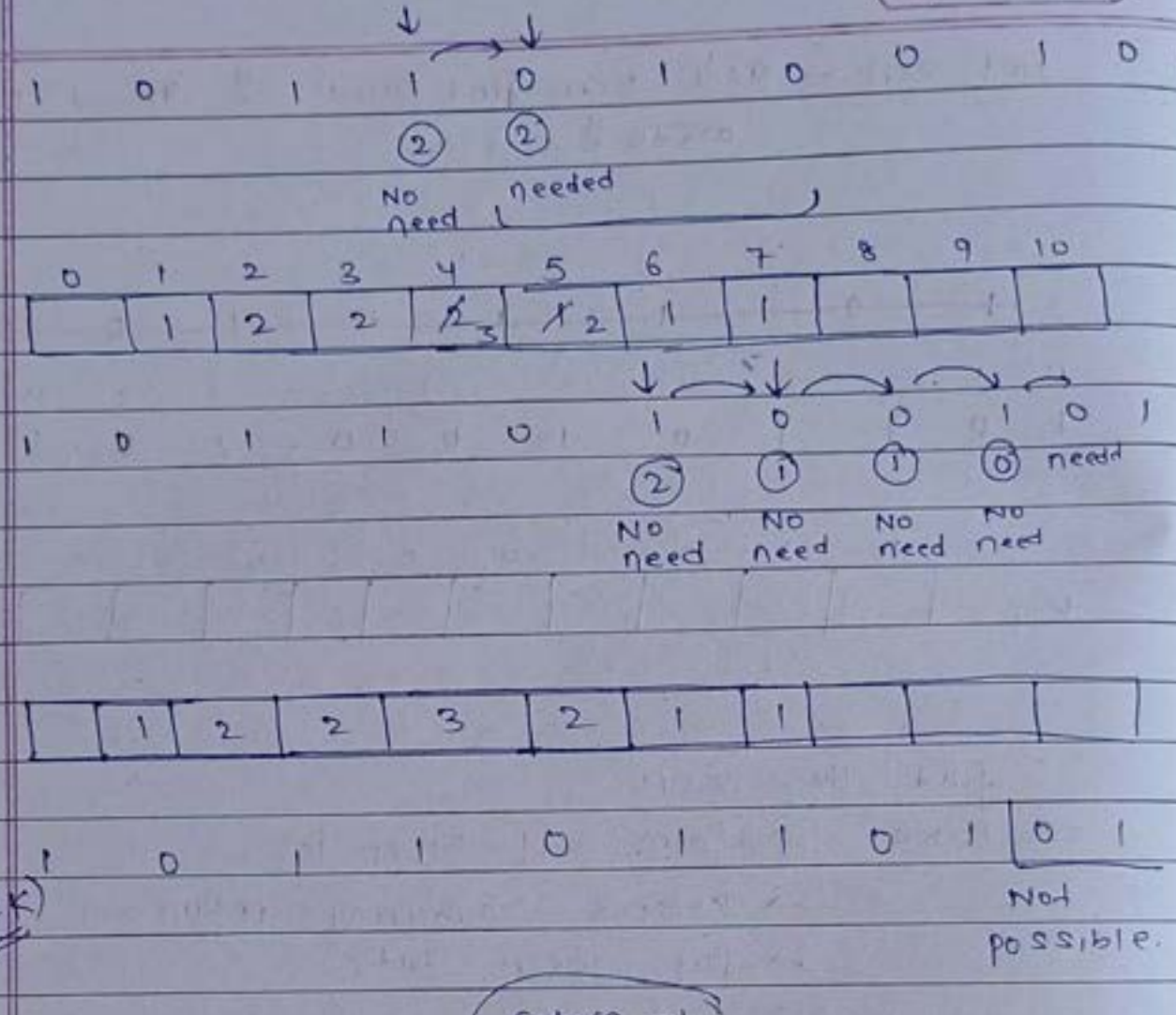
→ flip every bit.



with this whole subarray can flip



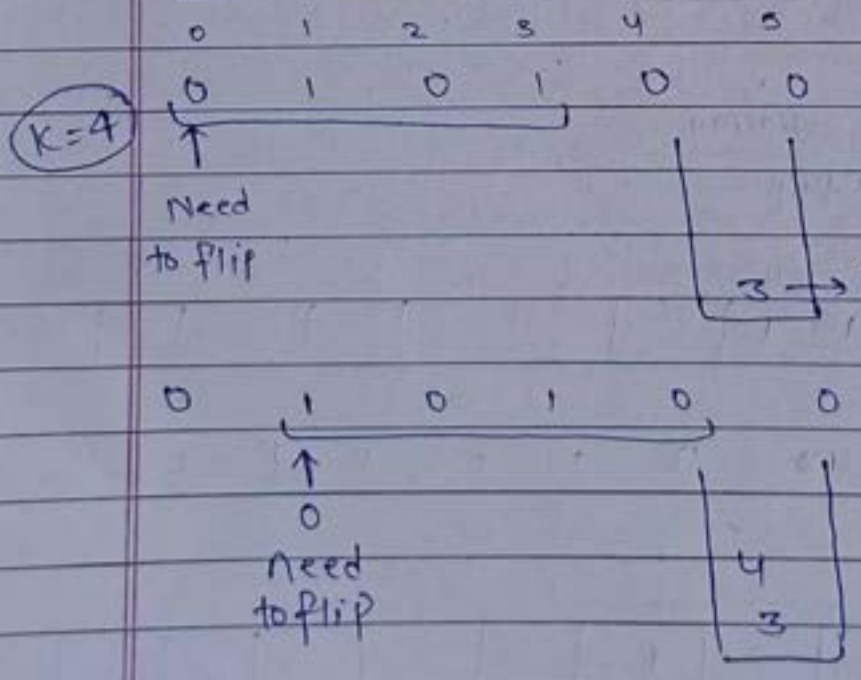
Teacher's Signature.....



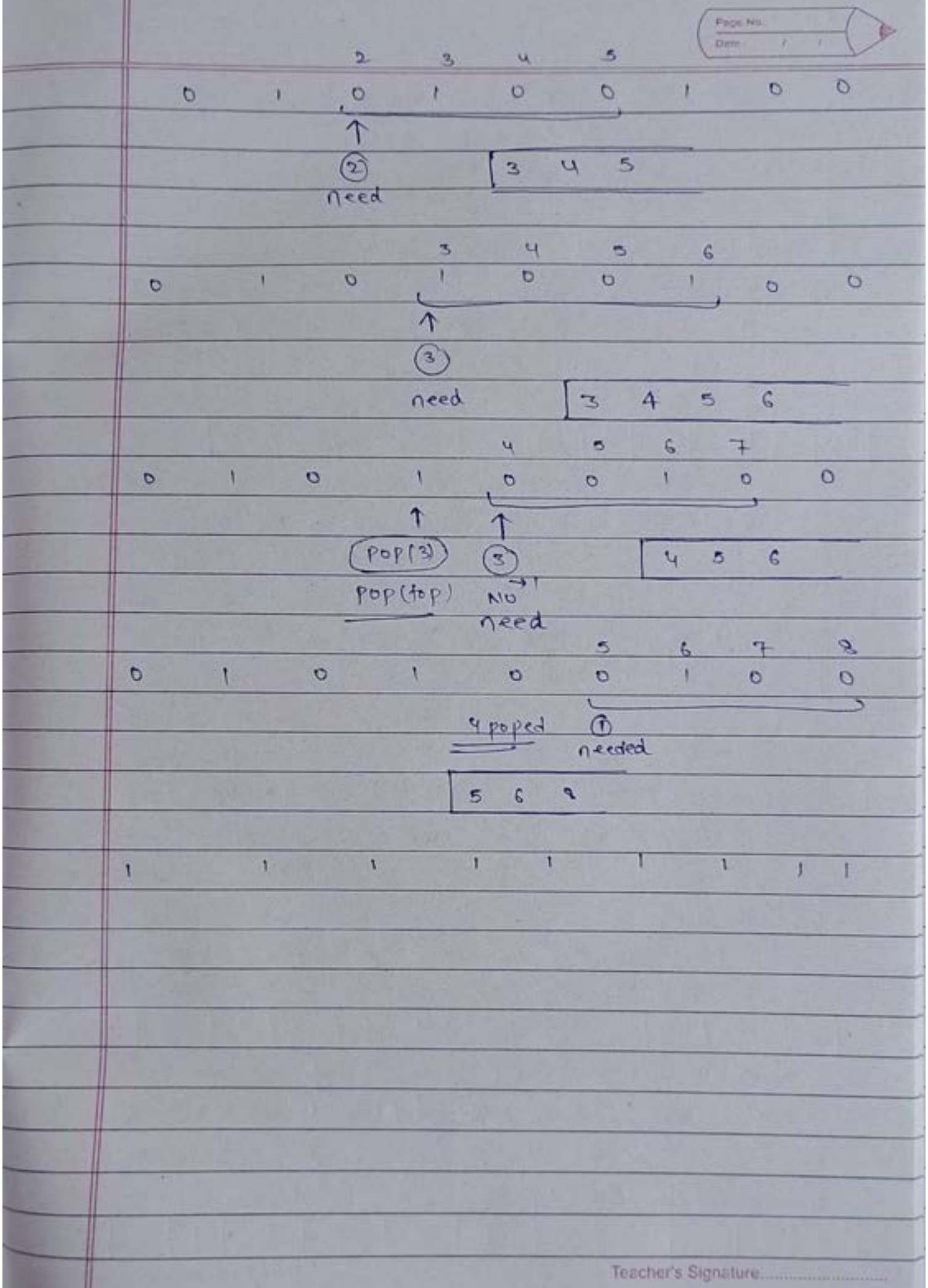
O(n*k)

return -1

Next Approach (Queue)



Teacher's Signature


2 3 4 5
0 1 0 1 0 0 1 0 0
2
need
3 4 5
3 4 5 6
0 1 0 1 0 0 1 0 0
3
need
3 4 5 6
4 5 6 7
0 1 0 1 0 0 1 0 0
pop(3)
pop(top)
3
NO
need
4 5 6
5 6 7 8
0 1 0 1 0 0 1 0 0
4 poped
1
needed
5 6 7
1 1 1 1 1 1 1 1 1

(Lecture - 40)

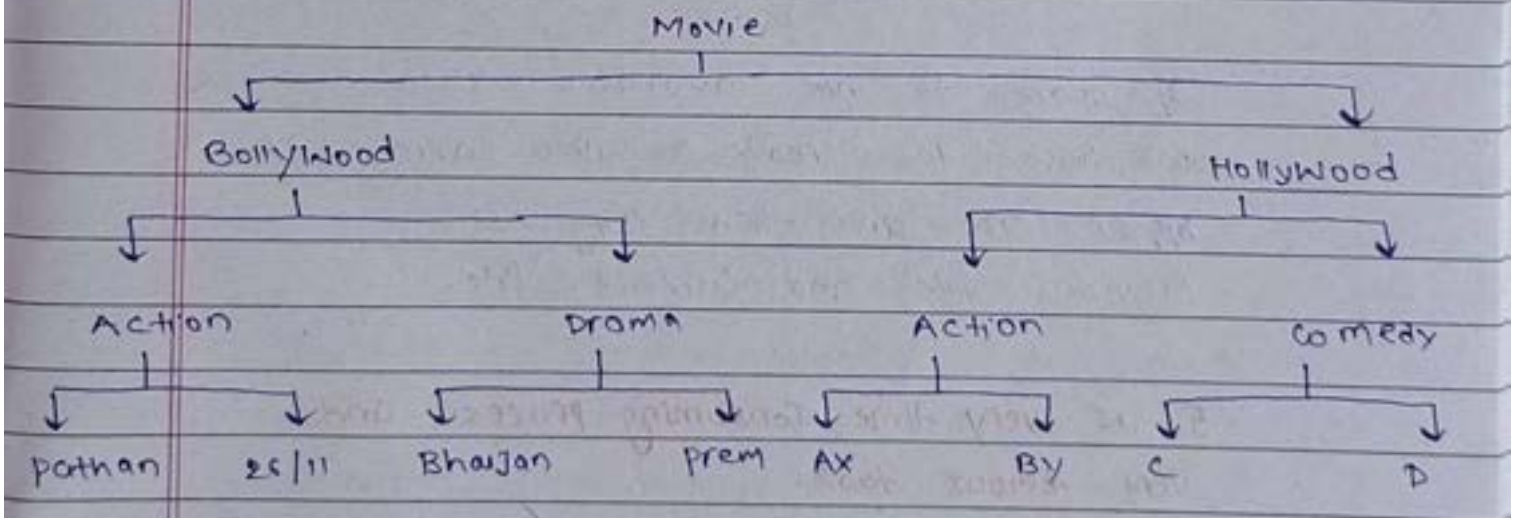
(Tree - Introduction)

Tree is a data structure which is a hierarchi-cal structure that is used to represent and organise data in a way that is easy to navigate and search.

It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.

Real life Example of tree :

Folders in a Computer



In this form, the folder is present in system. This is the real life example of Tree data structure.

For implementation of Tree Data structure which is more suitable for data storage? Array or Unlcedlist

Let's understand it with the example:

If we want to add a folder under the movie named (follywood) and add some movies.

Using Array:

We check that space is available or not then we insert the data (movie).

If space is not available then, we have to create a new array of bigger size and then copy all previous file and current file.

It is very time Consuming process and very tedious task.

Using Unlcedlist:

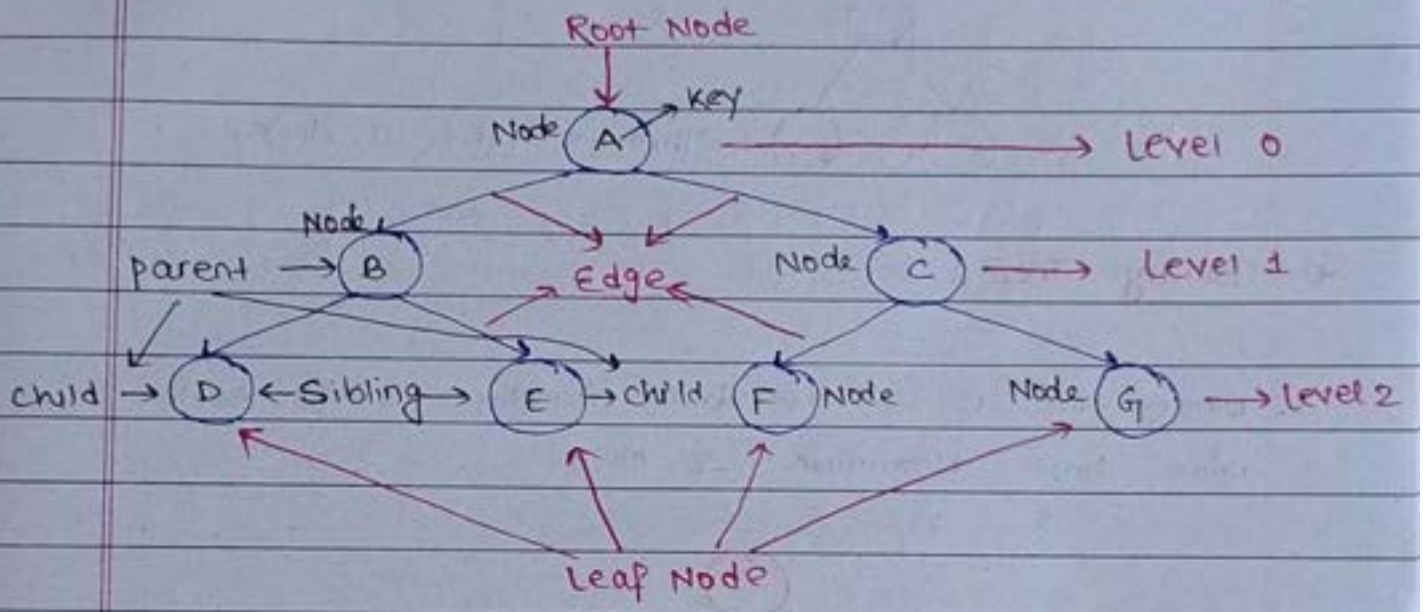
We find a space and insert data in that space and only point the memory with the previous Memory.

Teacher's Signature

Every time we need to store the data in free space and point it with previous data Node.

By linkedlist we can easily store the tree data structure.

We use linked list in tree data structure.



parent Node : The node which is predecessor of a node is called parent node.

B is parent of D & E.

Child Node : The node which is sucessor of a node.

F & G is child of C.

Root Node : The topmost node of a tree.

A is root Node.

Leaf Node : The Node which do not have any child.

D, E, F, G is leaf Node.

Ancestor : Any precedesor node on path to the root.

For D → Ancestors are : A, B

Sibling : Children of same parent nodes.

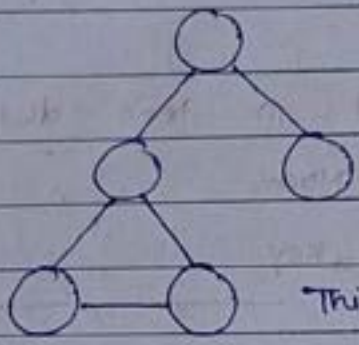
(D & E)

Teacher's Signature.....

No. of Nodes = N

then, No. of edges = N - 1

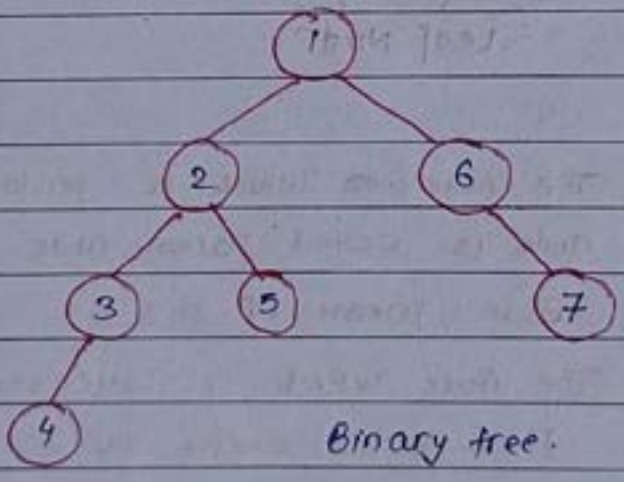
Tree cannot create closed loop



This is not a tree.

* Binary Tree :

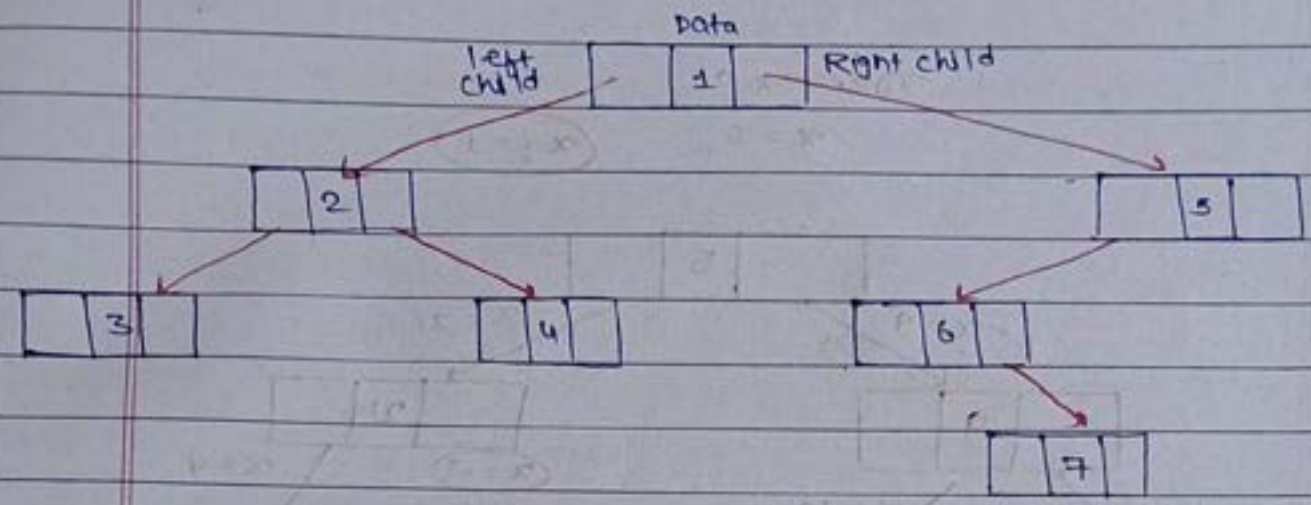
Binary tree is a tree data structure where each node has Maximum 2 child.



Binary tree.

Teacher's Signature

How to represent it in Memory.



Creation of Node:

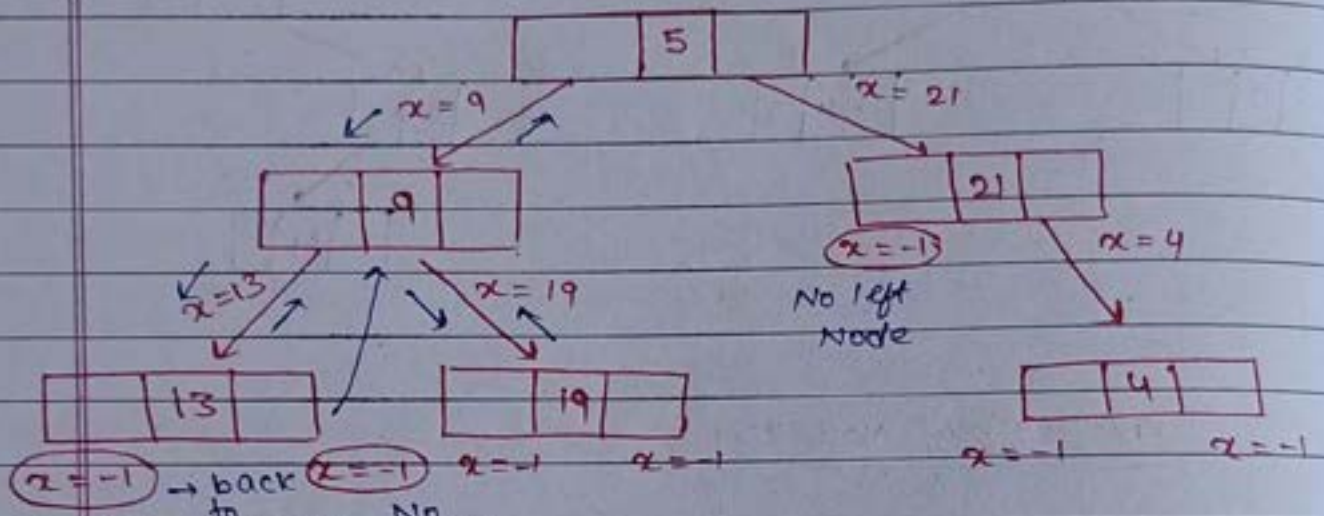
```
class Node {
    public:
    int data;
    Node * left;
    Node * right;

    Node (int value) {
        data = value;
        left = NULL;
        right = NULL;
    }
}
```

Teacher's Signature.....

Creation of Tree :

Take x ?

x = 5 (x != -1)




```
BinaryTree () {
        int x;
        cin >> x;

        if (x == -1)
            return NULL;

        Node * root = new Node(x);

        root -> left = BinaryTree ();
        root -> right = BinaryTree ();

        return root;
}
```

Teacher's Signature.....

Page No.
Date / /

Code :

```
#include <iostream>
using namespace std;

class Node {
    public :
    int data ;
    Node * left;
    Node * right;

    Node (int value) {
        data = value;
        left = NULL;
        right = NULL;
    }
};

Node * BinaryTreeCreation() {
    int x;
    Cout << " Enter value : ";
    cin >> x;

    if (x == -1)
    return NULL;

    Node * root = new Node (x);
    Cout << " Enter left child of " << x << " \n ";
    root -> left = BinaryTreeCreation();
```

Teacher's Signature.....

```
    cout << "Enter the right child of " << x << "\n";
        root -> right = BinaryTreeCreation();

    return root;
}

int main(){

    Node * root = BinaryTreeCreation();

    return 0;
}
```

In this code, we can only create the tree. But how to check that the tree is correctly created or not

For that we need to print the tree.

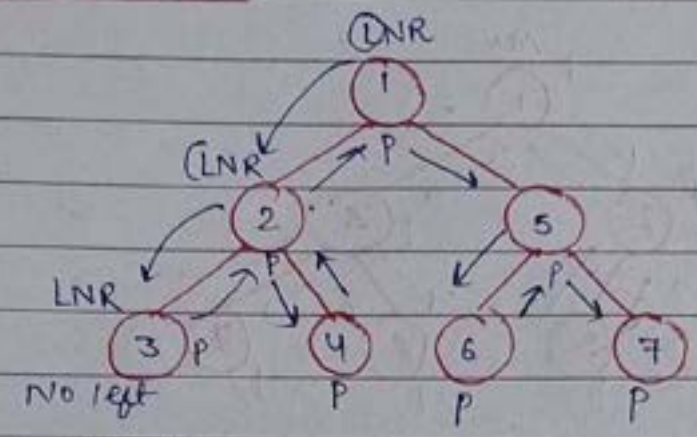
We have 3 traversal types which is used in trees:

1. Inorder Traversal
2. preorder Traversal
3. postorder Traversal

Inorder : L N R
preorder : N L R
postorder : L R N

Teacher's Signature.....

Inorder Traversal



go to Node
& print

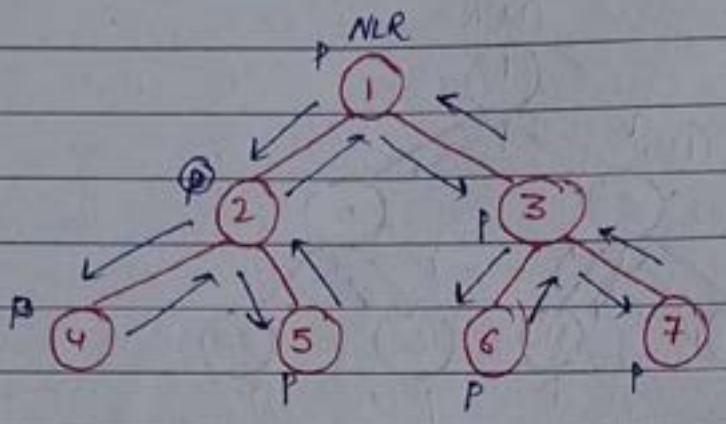
→ 3 2 4 1 6 5 7

Code :

```
Void Inorder ( Node * root ) {

    if (root == NULL)
        return;

    Inorder (root → left);    // L
    Cout << root → data ;     // N
    Inorder (root → right);   // R

}
```

Page No.
Date: / /

Preorder Traversal :

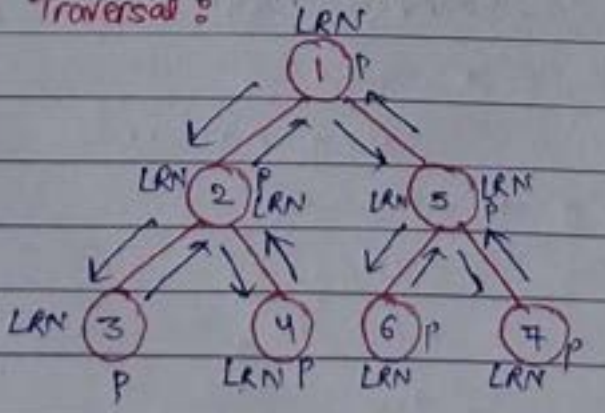


1 2 4 5 3 6 7

Code :

```
Void preorder (Node * root){
    if (root == NULL)
        return;

    Cout << root → data;      // N
    preorder ( root → left );  // L
    preorder (root → right);   // R

}
```

Postorder Traversal :



3 4 2 6 7 5 1

Teacher's Signature.....

Code :

```
Void postorder (Node * root) {
        if (root = NULL) {
            return;
        }

        postorder ( root → left);    // L
        postorder ( root → right);   // R
        Cout << root → data;         // N
    }
```

If we want to store -1 in the tree then first we have to ask :

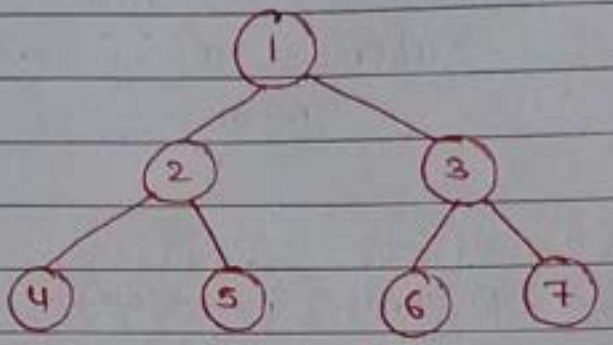
whether you want to create Node or Not.

If yes (1) > boolean.
or No (0)

```
int main () {
        Node * root = BinaryTreeCreation ();
        Cout << " Inorder Traversal : ";
            Inorder ( root);

        Cout << " preorder Traversal : ";
            preorder (root);

        Cout << " postorder Traversal : ";
            postorder (root);

        return 0;
    }
```

Teacher's Signature.....

→ Level order Traversal :



→ 1 2 3 4 5 6 7

We dont print it by recursion.

↳ Because Level order Traversal follow breadth first search (BFS).

↳ And recursion follow the depth first search (DFS).

In level order traversal, it uses Queue for its Traversal.

q = [2, 3, 4, 5, 6, 7

(1) print
↳ said (2, 3) ko print kr do

(2) print
↳ said 4, 5 ko print

(3) print
↳ said 6, 7 ko print

(4) print

(5) print

(6) print

(7) print

Teacher's Signature.....

→ 1 2 3 4 5 6 7

Code :

```
Void LevelorderTraversal (Node * root){

    if (root == NULL)
        return;

    queue <Node *> q;
    q.push(root);

    while ( ! q.empty()) {
    Node * temp = q.front();
    q.pop();
    Cout << temp→data;

    if (temp → left)
      q.push (temp→ left);

    if (temp → right)
    q.push (temp → right);

}
```

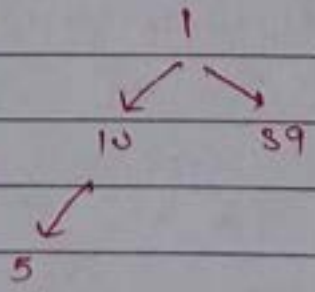
Teacher's Signature.....

problem 1: Inorder Traversal (GFG)

code:

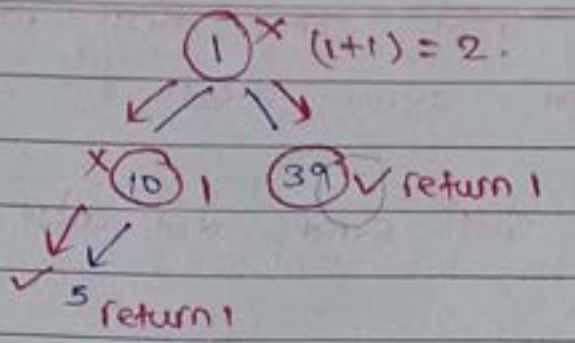
```
class solution {
    public:
    Void inorderTraversal (Node * root, Vector <int> ans){
            if (root == NULL)
                return;

        inorderTraversal (root → left, ans);
         Cout << root → data << " ";
        inorderTraversal (root → right, ans);
    }

    Vector <int> inOrder (Node * root) {
            Vector <int> ans;
            inorderTraversal (root, ans);
        return ans;
    }
};
```

problem 2: Count leaves in Binary tree



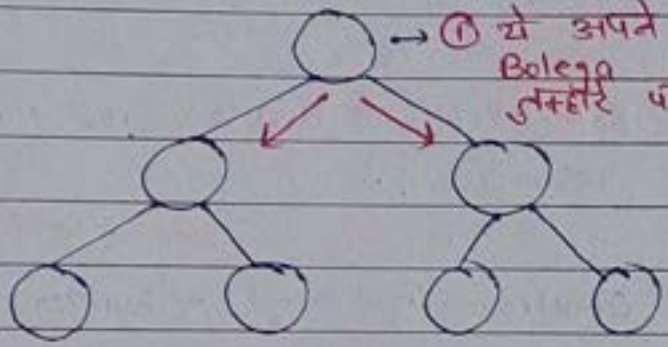
leaves: Node having No child.

Teacher's Signature.....

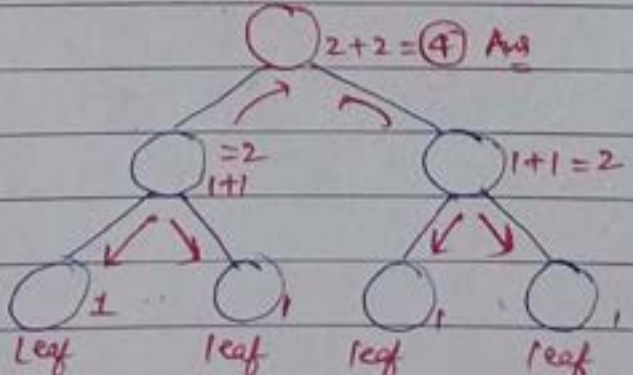



```
root → left == NULL && root → right == NULL
        return 1;
```

return क्या करना है।



② left & right dono check करेगा। क्या
ईन leaf Node है।
अगर होगा तो return 1.
नही तो वो अपने left or right
वाले को बोलेगा कि leaf Node
count kar ke do.



Teacher's Signature

हर बार वो पहले check करेगा कि उसका left or right Exist करता है या नहीं।

left or right दोनों Exist नहीं करेगा तो return 1.

Code :

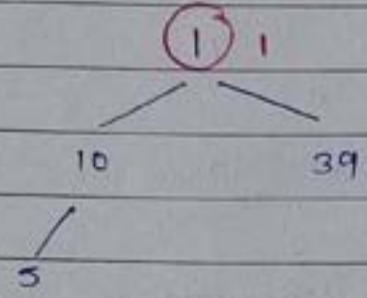
```
int CountLeaves(Node * root) {

    if (root == NULL)
        return 0;

    if (root → left == NULL && root → right == NULL)
        return 1;

    return CountLeaves(root → left) + CountLeaves(root → right);
}
```

Traverse : T.C → O(N)

create : T.C → O(N)
S.C → O(N)

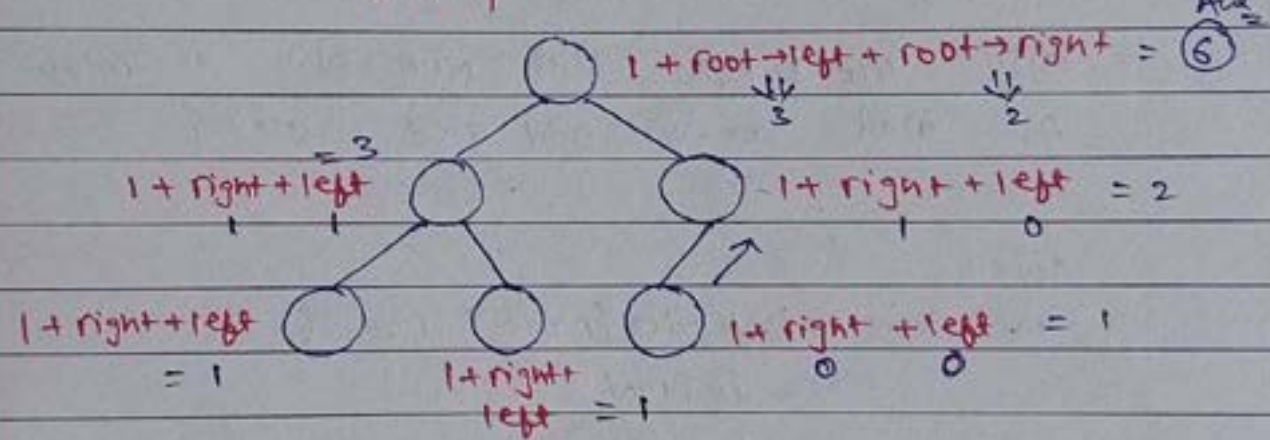
Teacher's Signature.....

* Problem 3 : Size of Binary Tree :



Count No. of Nodes.

① root node बोलेगा अपने left or right से अपने sub tree को count करने को बोलो और उसमें 1 add कर देगा अपना।



Code :

```
if (root == NULL)
    return 0;

return 1 + getSize(root→left) + getSize(root→right);
```

Teacher's Signature.....

Problem 4: Count Non-leaf Nodes in Tree

Approach 1

Step (1) Count total node — (I)

Step (2) Count leaf node — (II)

Step (3) → I - II

Approach 2

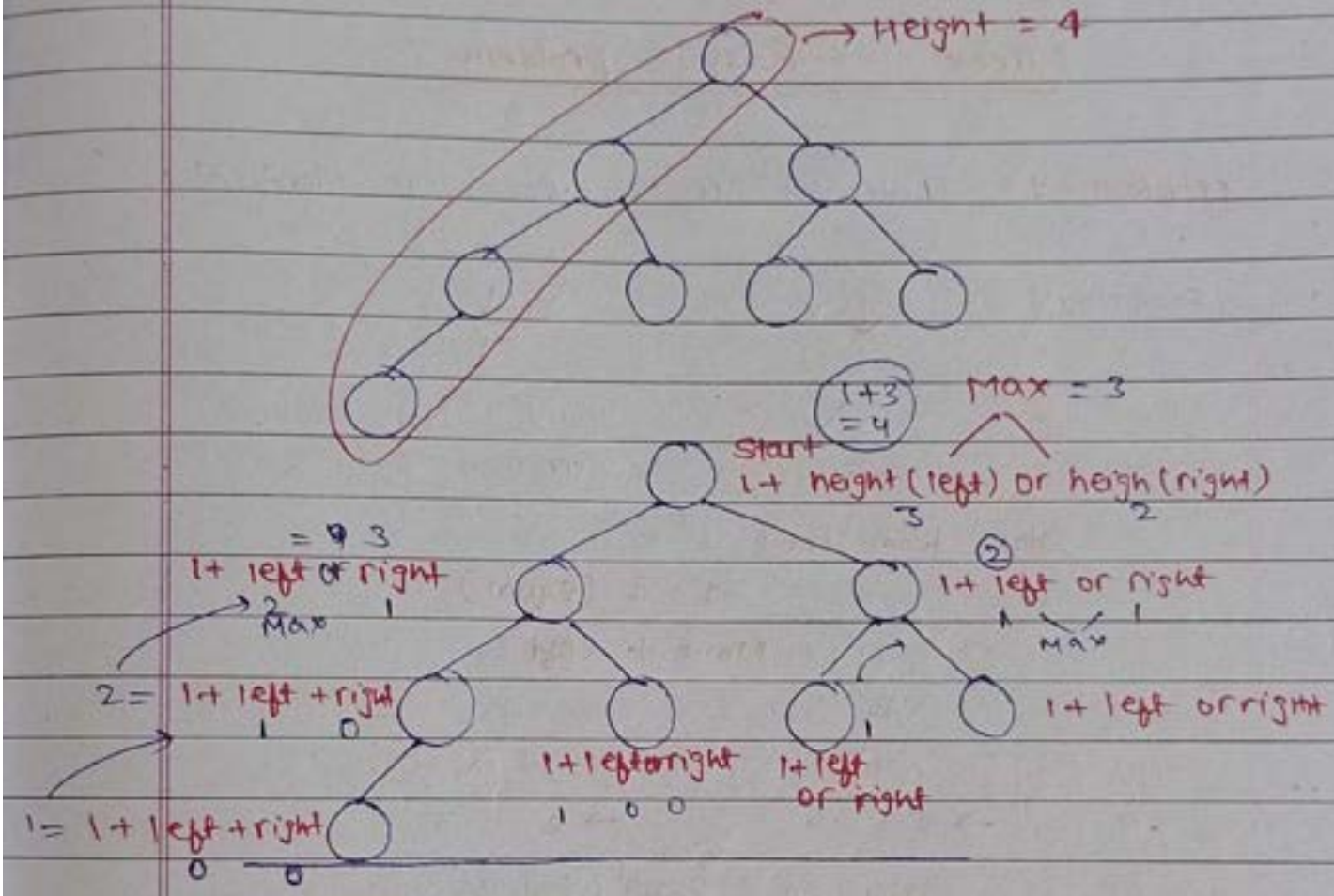
अगर left & right दोनों NULL होगा तो return 0, बाकी case में add करते जाएंगे।

Code:

```
int CountNonLeafNodes (Node * root ){
        if (root == NULL)
            return 0;

        if (root→left == NULL && root→right == NULL)
            return 0;

return 1+ CountNonLeafNodes (root→left) + CountNonLeafNodes
                                          (root→right);
}
```

Teacher's Signature.....

Problem 5 : Height of Binary tree



Code :-

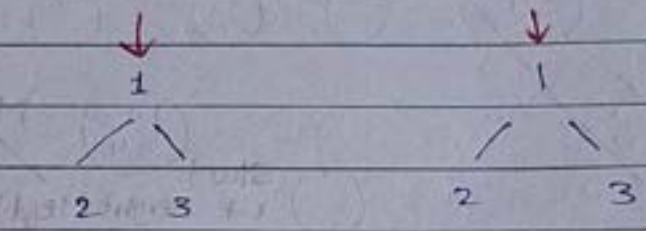
```
int height(struct Node * root) {
    if (root == NULL)
        return 0;

    if (root → left == NULL && root → right == NULL)
        return 1;

    return 1 + max(height(root → left), height(root → right));

}
```

Lecture - 41

Trees - Easy problem

problem - 1 : Determine if Two trees are identical

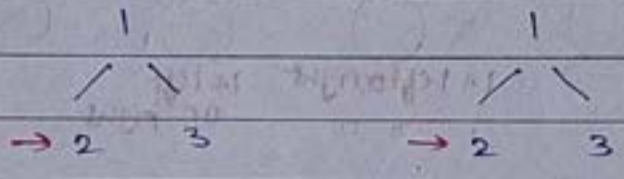
Example 1 :



Start from root

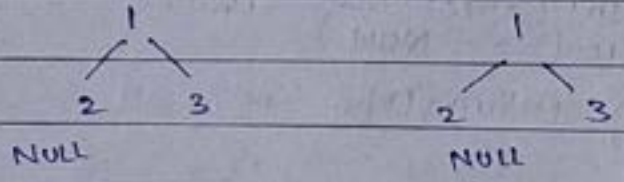
1 = 1 (Equal)

Move to left



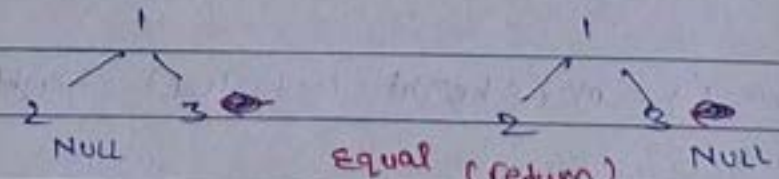
2 = 2 (Equal)

Move to left



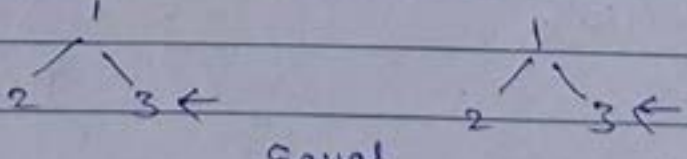
Equal.

Move to right



Equal (return)

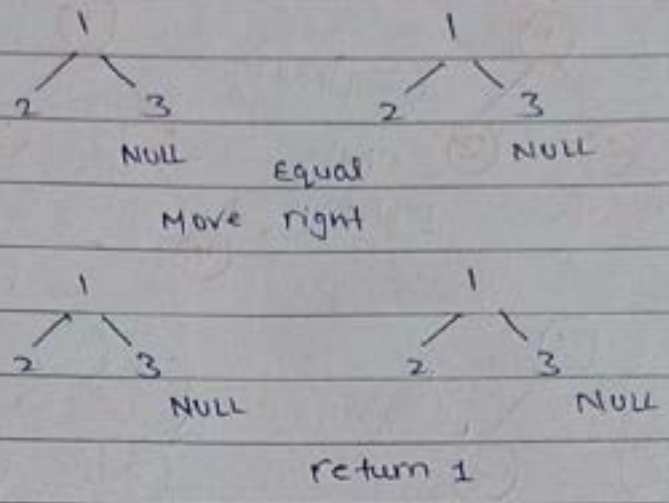
Move right



Equal

Move left.

Teacher's Signature.....



Case:

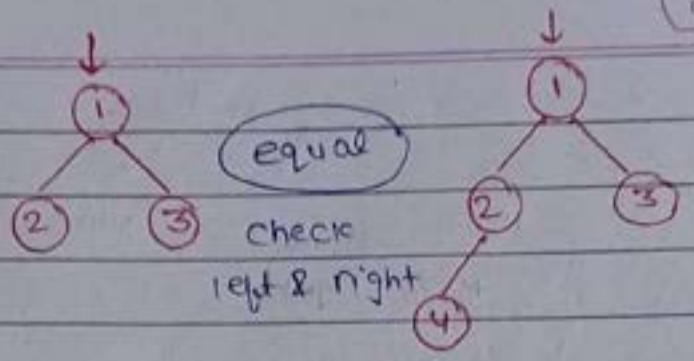
Tree 1	Tree 2		
NULL	Address	→	0
Address	NULL	→	0
NULL	NULL	→	1
Address	Address	→	0/1

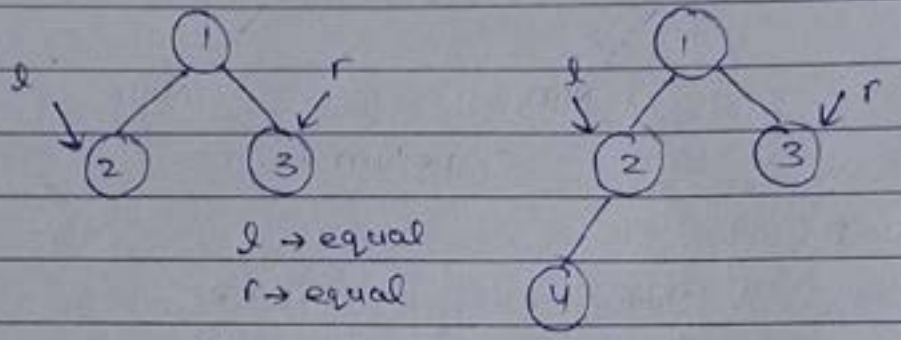
Code:

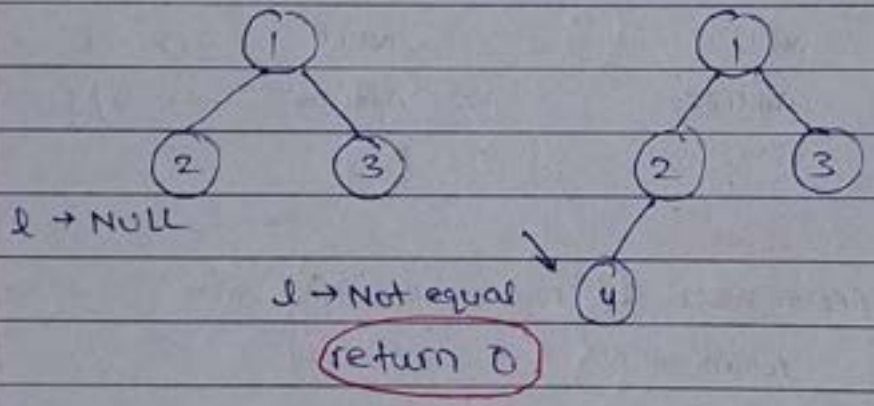
```
if (r1 == NULL && r2 == NULL) {
    return 1;
}
if (r1 && r2 == NULL || r2 && r1 == NULL) {
    return 0;
}
if (r1 -> data != r2 -> data) {
    return 0;
}
return isIdentical(r1->left, r2->left) && isIdentical(r1->right,
                                                      r2->right);
```

Teacher's Signature.....

Example 2

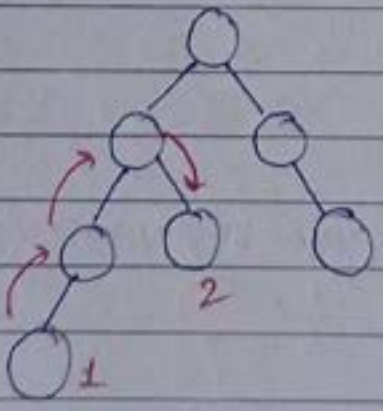






Problem 2 : Diameter of a Binary Tree

How to find diameter :



find diameter b/w 1 & 2 :

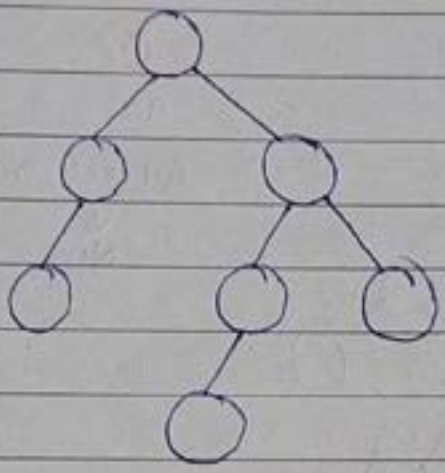
Diameter = 4

Teacher's Signature

Your task is to find the longest diameter present in Binary tree.

Example:

Diameter: 2 Node के बीच Maximum Node include both the two Node.

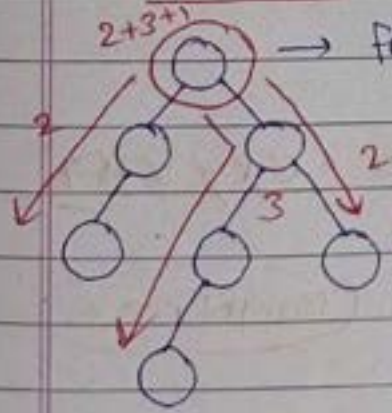


How to find:

diameter always start & end with leaf Nodes.

find the leaf Node:

Approach 1:



→ from this point we find the heigh from left and right.

left height = 2

right height = 3

diameter = 1 + 2 + 3

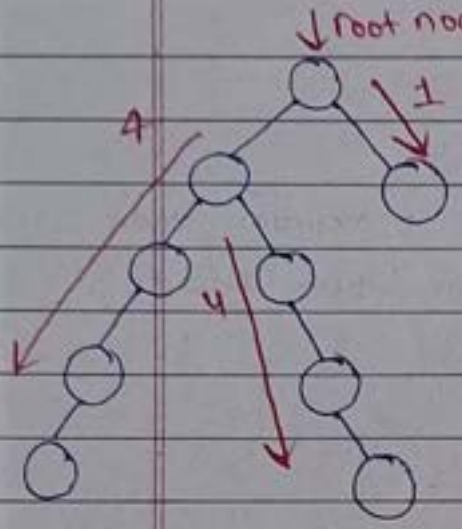
= (6)

↳ Ans.

But can we alway find the correct height from only root.

Teacher's Signature.....

Example 2 :



left height = 4

right height = 1

diameter = 1 + 4 + 1

= (6)

is 6 is the correct answer.

↓

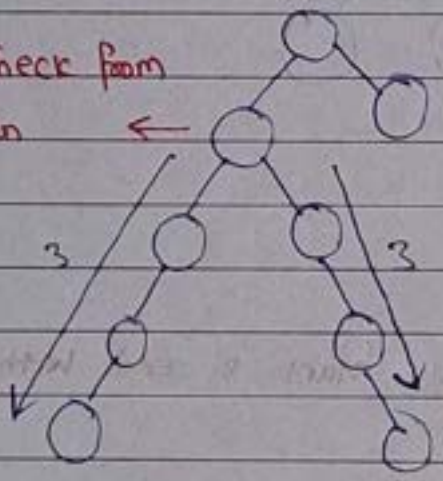
No.

If we check from this point then diameter is

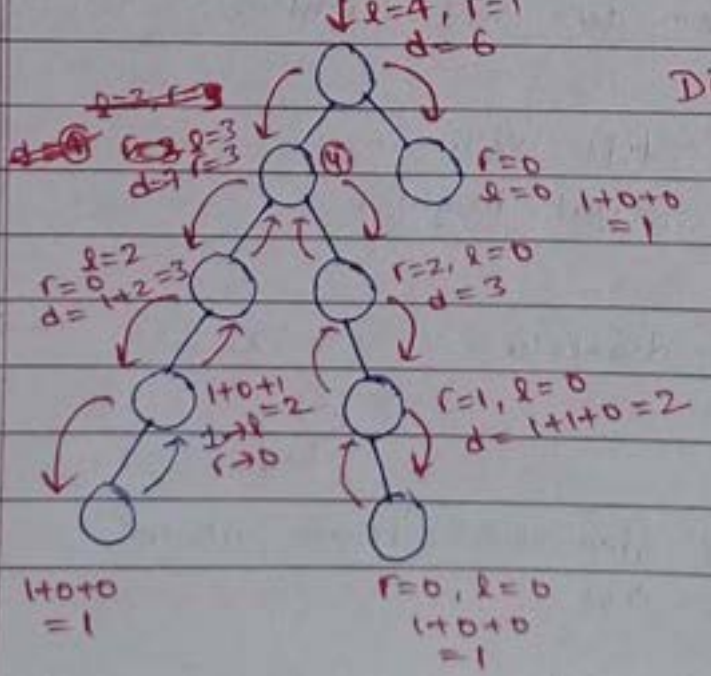
3 + 3 + 1

= (7)

7 > 6



So, we have to check every node for measuring the diameter.



Diameter = ~~0 X 2 3 6~~ 7

Diameter = 7

Teacher's Signature.....

① left height
② right height
③ find max from both and +1 = temp
④ diameter = max (diameter, temp)

base condition :

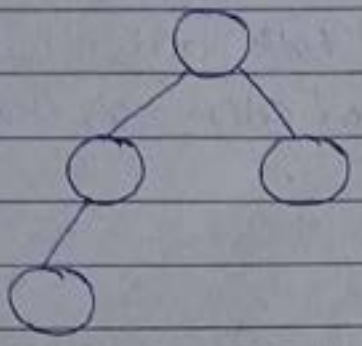
either Node Exist or NULL.

Code :

```
int diameter (Node * root) {
    int ans = 0;
    find (root, ans);
    return ans;
}

int find ( Node * root, int & ans ) {
    if (! root)
        return 0;

    int left = find (root → left, ans);
    int right = find (root → right, ans);

    ans = max ( ans, 1 + left + right );

    return 1 + max (left, right);

}
```

Teacher's Signature.....

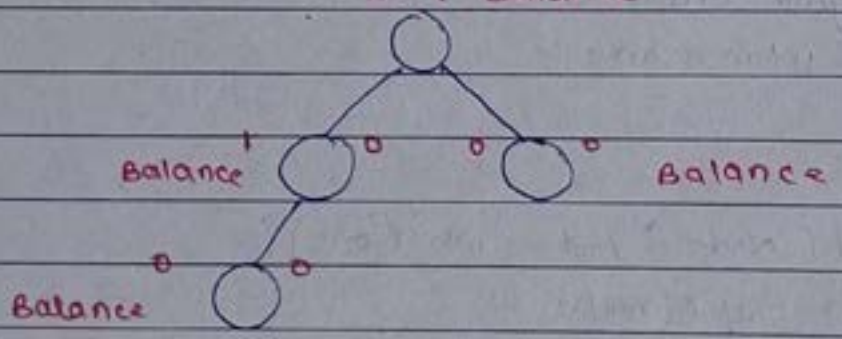
Problem 3 : Check for balanced Tree



difference of left & right subtree not > 1.

Approach :

1. calculate left side height
2. calculate right " "
3. find the diff.
   ↳ if > 1 return 0.



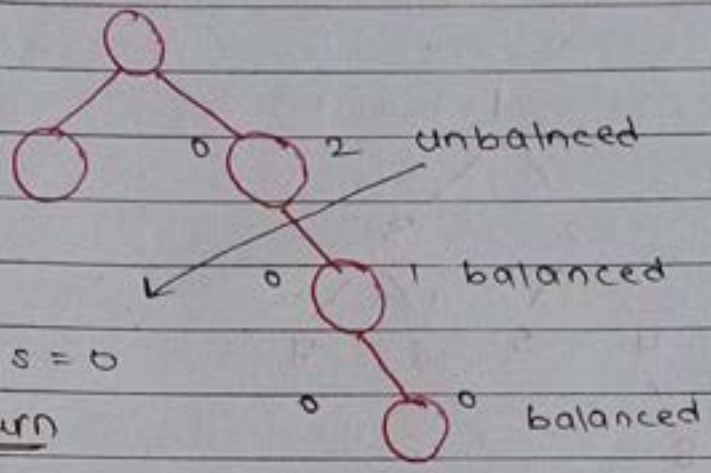
Ek answer variable

~~ans~~ → 0

ans = 1;

अगर एक भी Node balance नहीं रहेगा तो wahi pe turant 0 kar denge or return कर देंगे,

Example : 2

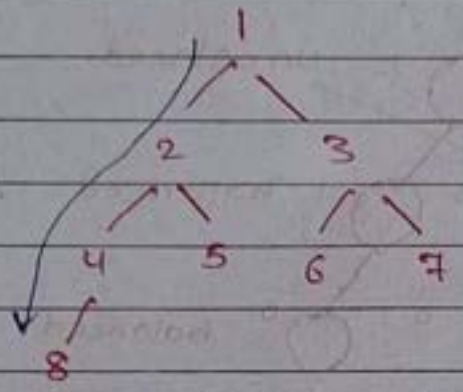


Code :

```
bool isBalanced (Node * root) {
        bool ans = 1;
        findHeight (root, ans);
        return ans;
    }

int findHeight (Node * root, bool &ans) {
        if (root == NULL)
            return 0;

        int left = findHeight (root → left, ans);
        int right = findHeight (root → right, ans);

        if (abs (left - right) > 1)
            ans = 0;

        return 1 + max (left, right);

}
```

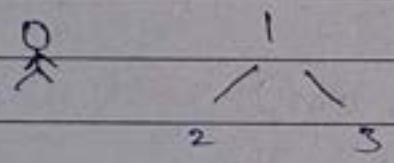
Teacher's Signature

Problem 4: Left View of Binary Tree

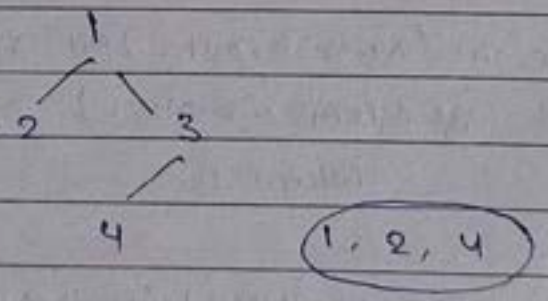


Left View : 1 2 4 8

Visited from left side.



(1, 2) output



How to solve this?

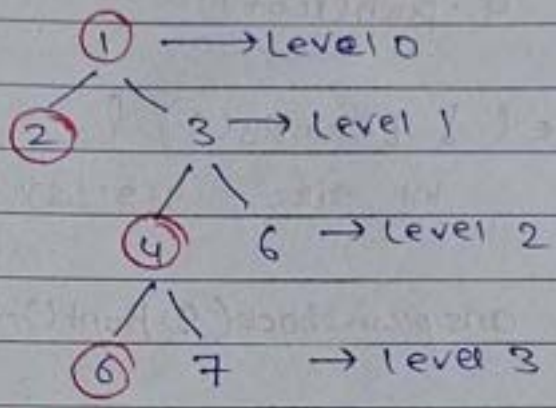
↳ recursion

↳ iterative.

Teacher's Signature.....

(1) Iterative approach

Har level का first Node



1, 2, 4, 6

check all level and return 1st Node of each level from left.

Queue | 2, 3

1 visited — Ans: 1, 2
↳ said 2, 3

only first can be taken in answer. Neglect all other.

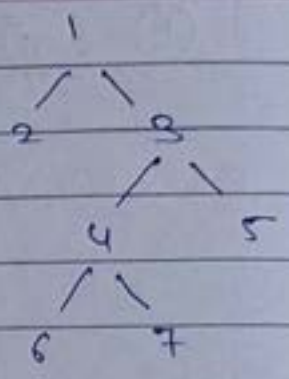
first element → 2
3 can't answer
| 4, 5 — Ans: 1, 2, 4
5 can't answer

| 6, 7 — Ans: 1, 2, 4, 6
7 can't answer

Teacher's Signature.....

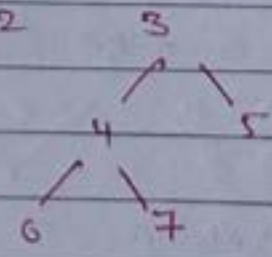
Code:

```
Vector <int> ans;
if (root == NULL) return ans;
queue <Node *> q;
    q.push(root);

while ( ! q.empty()) {
        int size = q.size(); // tells the no. of element present on
ans.push_back( q.front()->data); that level.

while ( size --) {
    Node *temp = q.front();
        q.pop();

    if (temp->left) {
        q.push (temp->left);
    }
    if (temp->right)
        q.push (temp->right);
}
}
return ans;
}
```



Explanation (dry run)

(1) q.push(root) → q [1]



ans = {1}

q.empty() → NO

size = 1

ans.push(q.front()) = (1)

Teacher's Signature

temp = 1 — 1 deleted.

q.pop();

q | 2 3

temp→left = ~~NULL~~ 2

↳ ~~no push~~ push to queue

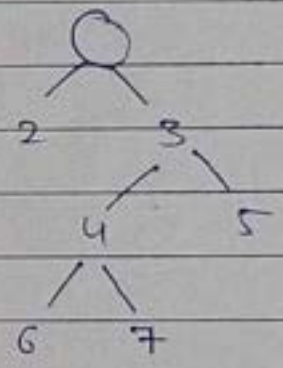
temp→right = 3 push

loop end

again loop start

Size = 2

ans.push(2)



ans = {1, 2 }

temp = 2

2 deleted from q. Size = 1

temp→left NULL

temp→right NULL

temp = 3

3 deleted q | 4 5

temp→left = 4

temp→right = 5

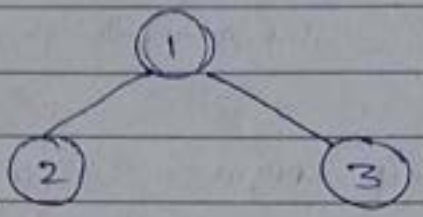
Size = 0

loop end

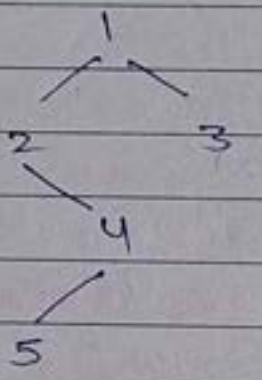
Teacher's Signature

By this process we can Complete the Question.

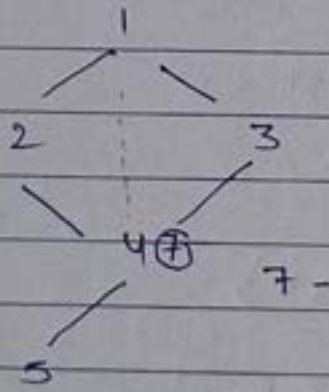
Problem 5: Bottom View of Binary Tree



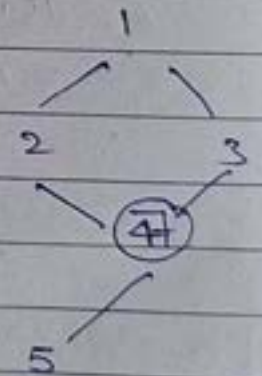
Ans:- 2 1 3



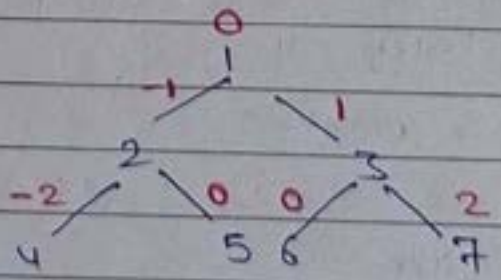
5 4 3



7 → 4 के ऊपर है।



5 7 3

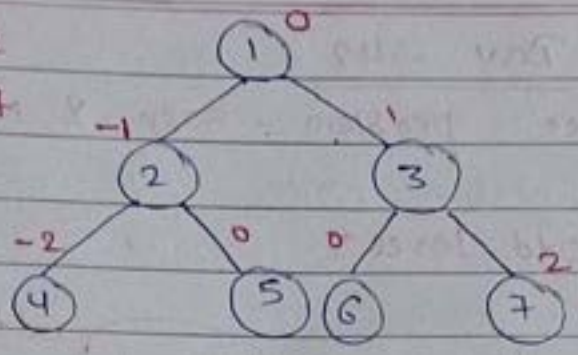


-2 -1 0 1 2

4 2 6 3 7

Teacher's Signature

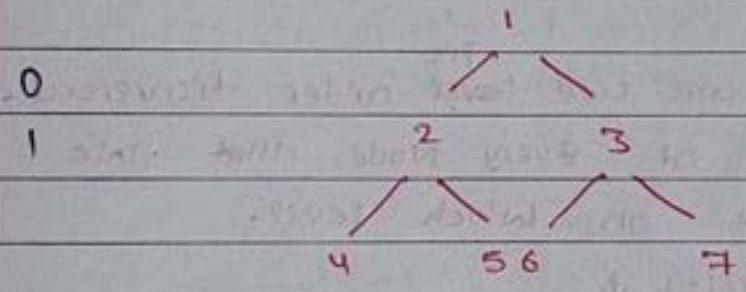
-2	-1	0	1	2
4	2	~~4~~ 5 6	3	7



q : ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~

q : -1 1 -2 0 0 2

Level order.



q : ~~1~~ 2 3

0

q : ~~2~~ 3 4 5

-1 1 -2 0

-1	0
-2	1

q : ~~3~~ 4 5 6 7

1 -2 0 0 2

-1	0	1
2	1	3

q : ~~4~~ ~~5~~ ~~6~~ ~~7~~

-2 0 0 2

-2	-1	0	1	2
4	2	~~4~~ 5 6	3	7

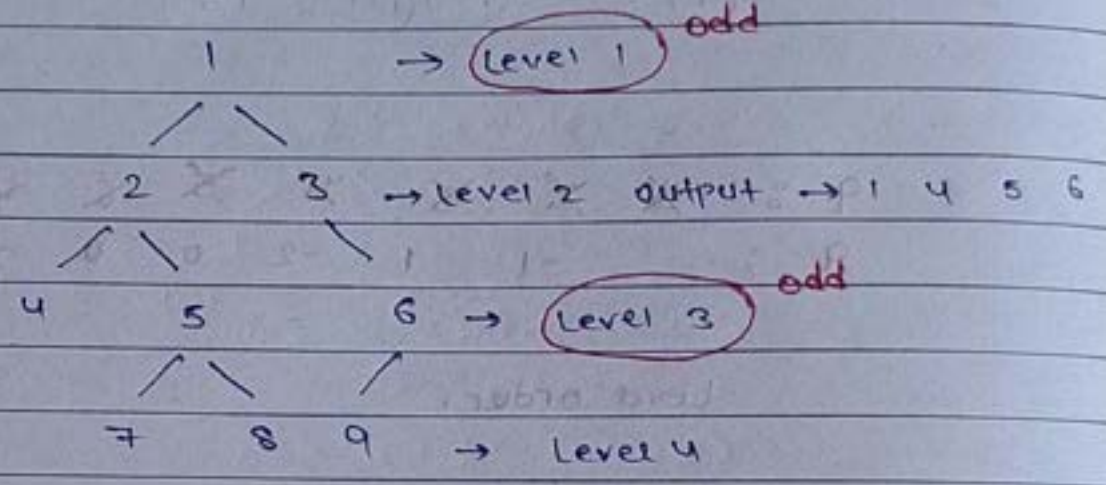
Ans: 4 2 6 3 7

Teacher's Signature

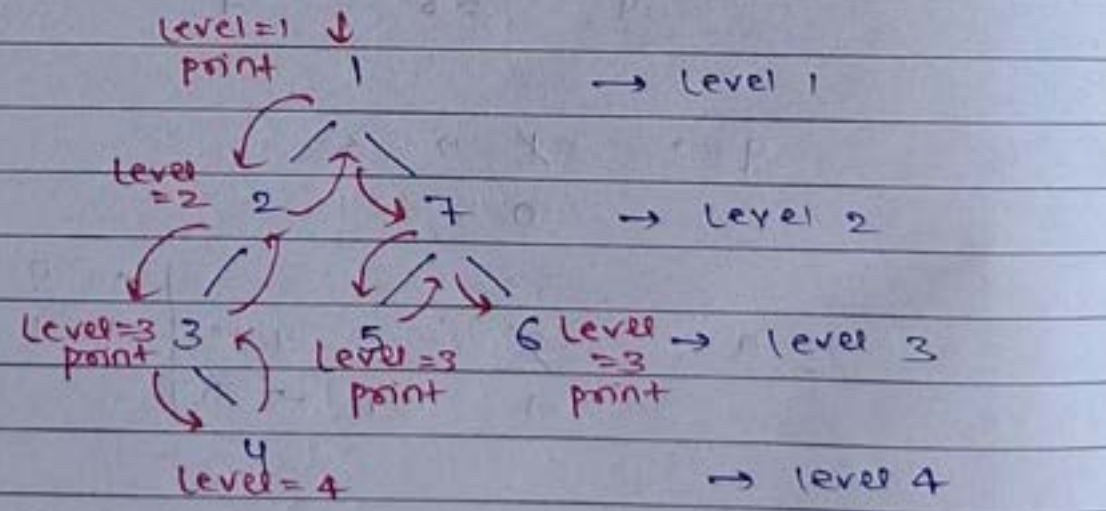
Day - 42

(Tree problem - Easy & Medium)

① Number of odd level :



on this we use ~~level~~ Pre order traversal.
Check at Every Node, that we present on which level.



Output : 1 3 5 6

Teacher's Signature.....

Take a variable label and keep record of pointer that it is at which level.

Code:

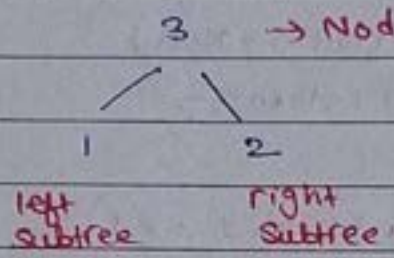
```
Void preOrder(Node* root, vector<Node*>& ans, int level){
        if(root == NULL)
            return;

        if(level % 2 == 1)
         ans.push_back(root);

    preorder(root->left, ans, level+1);
    preorder(root->right, ans, level+1);
}

vector<Node*> nodesAtOddlevels(Node * root){
        vector<Node*> ans;
          int level = 1;
      preorder( root, ans, level);
       return ans;
    }
```

Teacher's Signature

Problem 2 : Sum tree :

Return true, if Node = left subtree + right subtree.
Except leaf Node.

Ex :

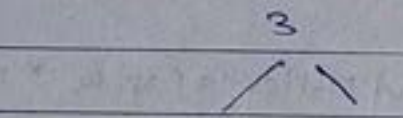


3 = 1 + 2 (True)

return.

Note : Don't care about leaf Node.

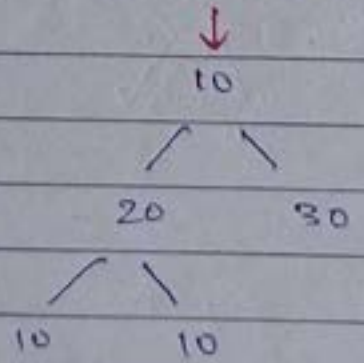
Because it always gives false.



leftSubtree = NULL

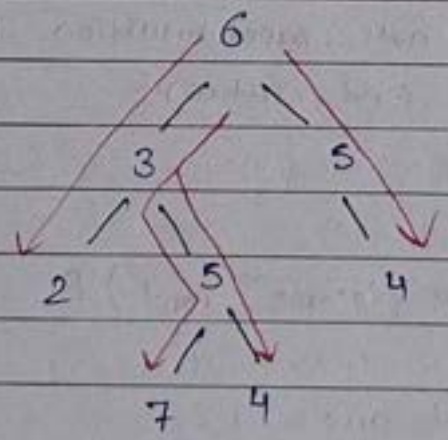
rightSubtree = NULL

NULL + NULL ! = 1 (false).



```
if (root == NULL)
    return 0;

if (root -> left == NULL && root -> right == NULL){
    return root -> data;

int left = sumTree (root -> left, ans);
int right = sumTree (root -> right, ans);

if (left + right != root -> data)
    ans = 0;

return root -> data + left + right;

}
```

Problem 3: Root to leaf paths sum:



4 path from roof to leaf

632 6357 6354 654

ans = 632 + 6357 + 6354 + 654

Teacher's Signature

path:

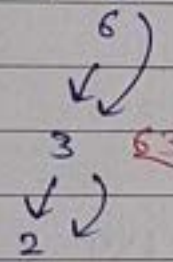
6 → 3 → 2	→	632
6 → 3 → 5 → 7	→	6357
6 → 3 → 5 → 4	→	6354
6 → 5 → 4	→	654

Ans = 632 + 6354 + 6357 + 654

= 13997

How to solve?

path 1 : ↓num = 6



Sum = 0

num = root → data

num = 6

num = num × 10 + root → left

= 6 × 10 + 3

= 63

63 × 10 + 2

= 632

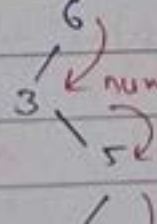
↳ Here we get 1st number from path 1.

add this to sum.

sum += num

sum = 632

path 2 : ↓num = 6

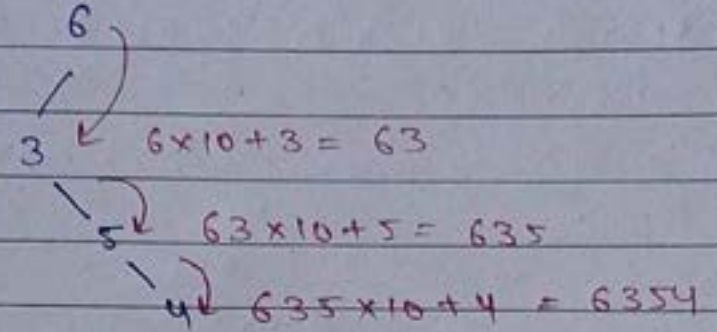


sum = sum + num

= 632 + 6357 = 6989

Teacher's Signature

Path 3: ↓num = 6



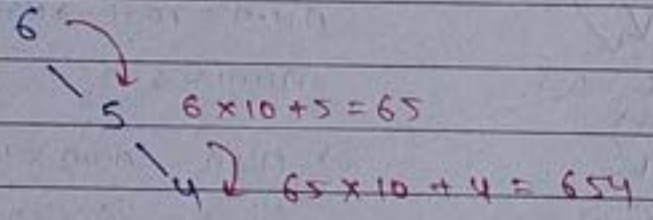
num = 6354

sum = sum + 6354

= 6989 + 6354

= 13343

Path 4: ↓num = 6



num = 654

sum = sum + num

= 13344 + 654

= 13997

We move to all path from root to leaf

then return the sum.

At every setep, we move either right or left.

Teacher's Signature

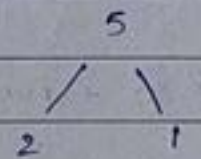
Code :

```
Void findsum (Node * root, long long int &sum,
                                long long int num) {
        if (root == NULL)
            return;

        num = num * 10 + root → data;

    if(root → left == NULL && root → right == NULL){
            sum += Num;
            return;
    }

    findsum (root → left, sum, num);
    findsum (root → right, sum, num);

}

long long int treePathsSum (Node * root) {

        long long int sum = 0;
        long long int num = 0;

    findsum ( root, sum, num);

    return sum;

}
```

Teacher's Signature.....

Problem 4 : Maximum difference between node and its ancestor

Example

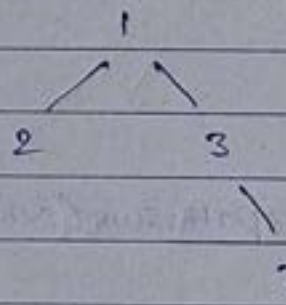


Ancestor of 5 = 2, 1

5 - 2 = 3

5 - 1 = (4) → maximum.

Example



Ancestor of 1 : 2, 3, 7

Ancestor of 3 : 7

There is no ancestor of 2, 7 (leave node)

leaf Node का Acestor नहीं होता है।

1 - 2 = (-1) → maximum (return)

1 - 3 = -2

1 - 7 = -6

3 - 7 = -4

Teacher's Signature.....

Page No.
Date / /

Initially

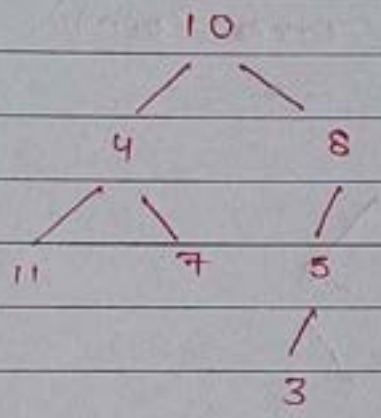
max = Integer.MIN.

A - B

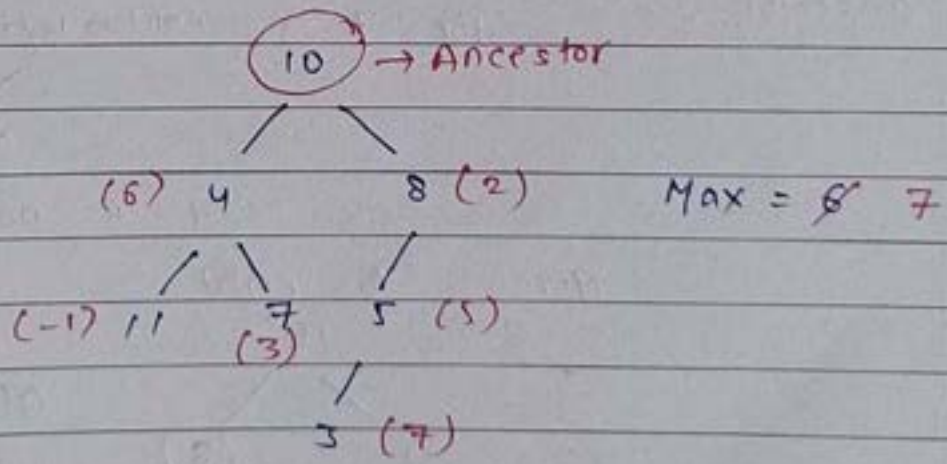
↙ ↘

Acestor Descendant

Example :



Acestor of 10 : 4 11 7 8 5 3

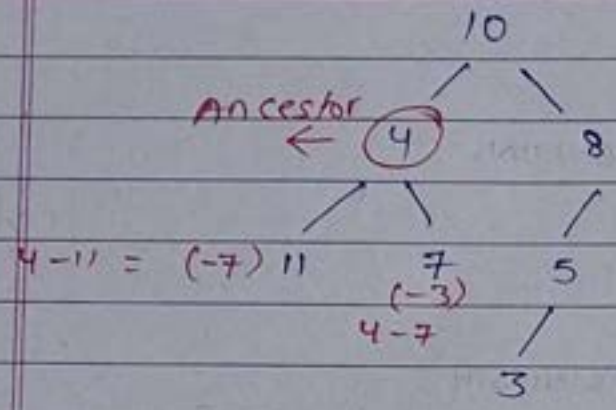


For 10 we check all possible value takes O(n) time.

But we have to check for all other.

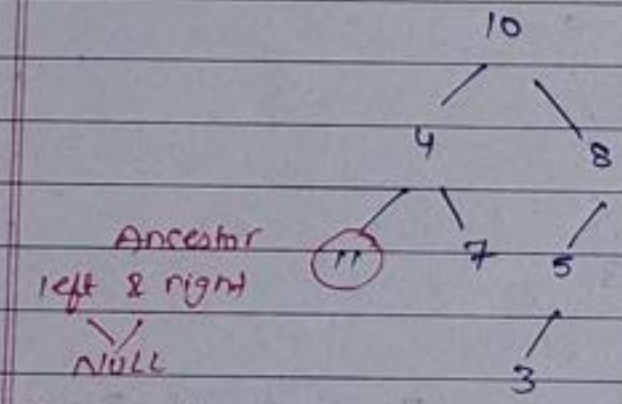
go to left

Teacher's Signature



Ancestor of 4 : 11 , 7

Takes (O(n)).



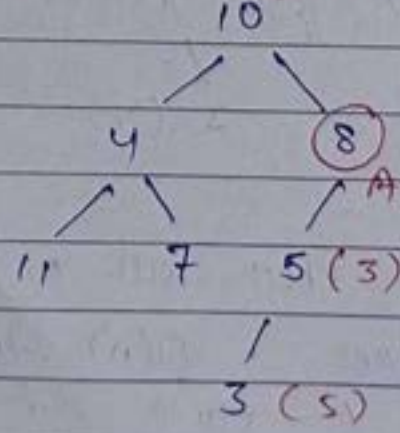
11 is not ansector

For 7 → root → left && right

NULL

7 is not ansector.

Move to right.



O(n)

Max = 7

Teacher's Signature.....

B for 8 = 5, 3

We check for 5 & 3 and Maximum we get:

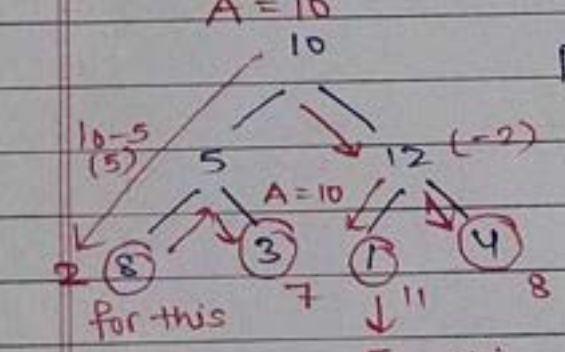
max = 7.

$n * (n-1)$

Time Complexity : $O(n^2)$

↳ We check for every node and for every node we subtract every possible.

* How we solve this in O(N)?



A - B => Max
↳ When A is big number.

for this what it take 5 or 10

for this A = 10 or 12 = 12 (Max)

diff = 5 7 15 Ans

Max (A, root → data)

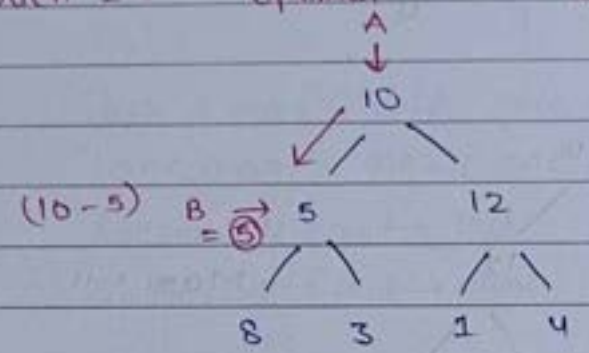
(10 - 8 = 2) → Alway take max.
5 - 8 = -3

By this we can solve in O(N)

Teacher's Signature.....

Problem 4 : Maximum difference between node and its ancestor :

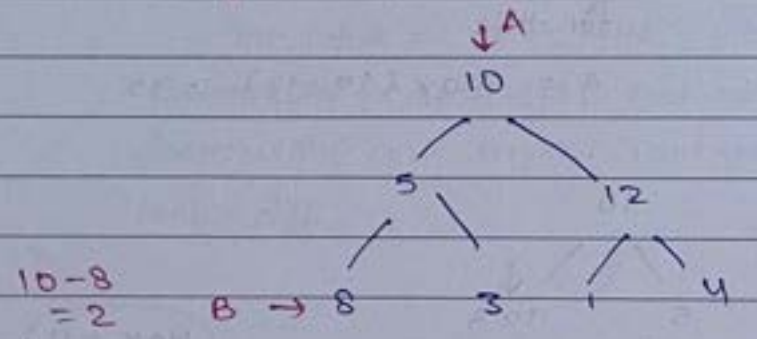
Approach 2 : Optimal O(N)



A - B
↓
Ancestor

Max = ~~Min~~ 5

We use preorder traversal here



Max = 5

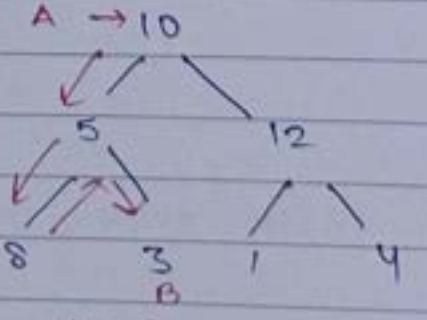
for 8 → which is Ancestor who give maximum value.

10 or 5

Bigger Number gives Max

So, we take max (A, root → data)

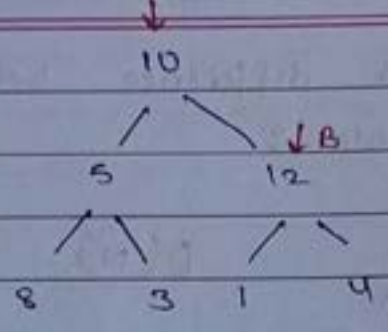
Max = A



Max = ~~5~~ 7

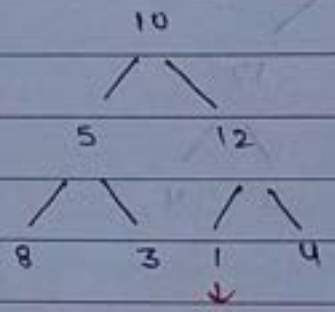
10 - 3

= 7



10 - 12
= - 2

Max = 7

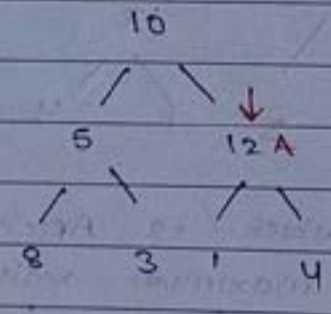


Max = 11

12 - 1 = 11

for this

A = max (10, 12) = 12



Max = 11

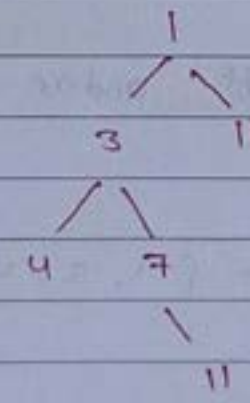
12 - 4 = 8

By this approach, We can solve in O(n) time.

Code:

```
void findMaxDiff ( Node * root, int anc_max, int &diff ) {
        if (root == NULL)
            return ;

        diff = max ( diff, anc_max - root → data );
        anc_max = max ( anc_max, root → data );
        findMaxDiff (root → left, anc_max, diff );
        findMaxDiff (root → right, anc_max, diff );
}

int maxDiff ( Node * root ) {
        int diff = INT_MIN;
        findMaxDiff ( root → left, root → data, diff );
        findMaxDiff ( root → right, root → data, diff );
        return diff;
}
```

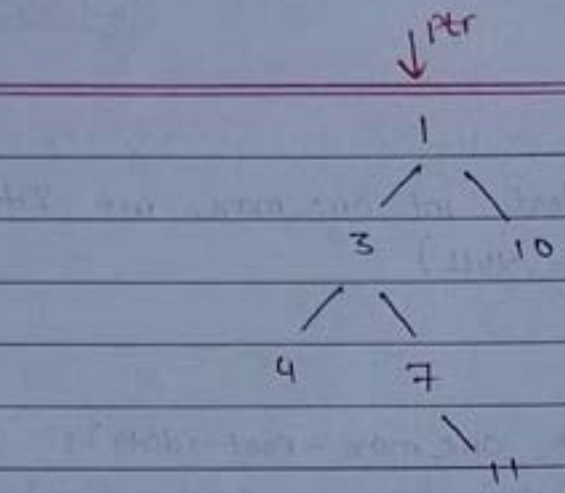
Problem 5 : preorder traversal (iterative)



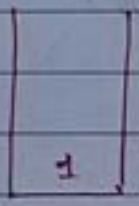
1, 3, 4, 7, 11, 10

We have to solve this without recurion

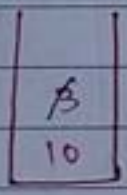
so, we use stack in it.

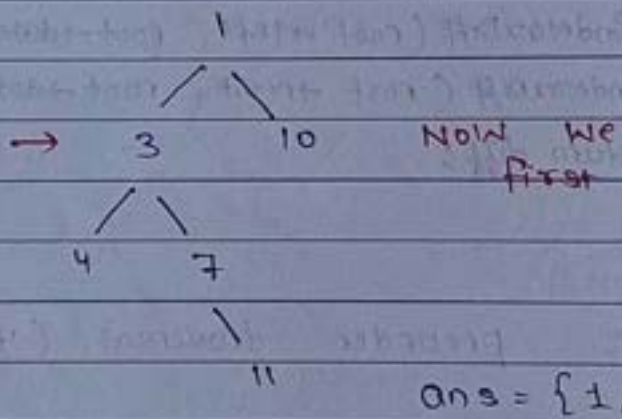


NLR



→ print 1 ans = {1

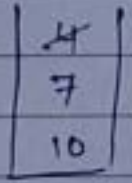




Now we put right ~~first then left~~

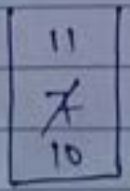
ans = {1, 3

3 का right left aur



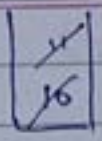
ans = {1, 3, 4

4 का right & left aur



ans = {1, 3, 4, 7

classmate
Date
Page



ans = {1, 3, 4, 7, 11, 10}

(1) root ko daal do stack me

(2) Jab tak stack khali nahi hota.

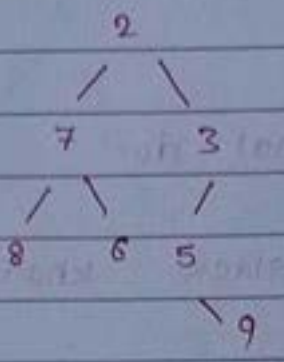
→ top को print kro
→ pop कर दो top को।
→ top ke right को stack Me dalo

→ top ke left को stack Me dalo

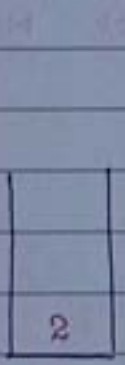
Code :

```
Vector <int> preOrder (Node * root) {
    Vector <int> ans;
    stack <Node*> s;
    s.push (root);
    Node * temp;

    While (! s.empty ()) {
        temp = s.top ();
        s.pop ();
        if (temp → right)
            s.push (temp → right);
        if (temp → left)
            s.push (temp → left);
        ans.push_back (temp → data);
    }
    return ans;
}
```

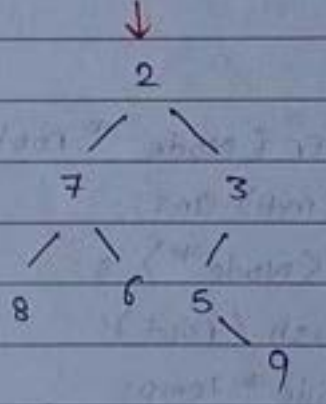
problem 6: postorder Traversal (iterative)



Not by recursion
solve by stack.



LRN



NRL

2 3 5 9 7 6 8

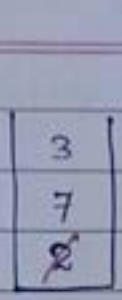
reverse

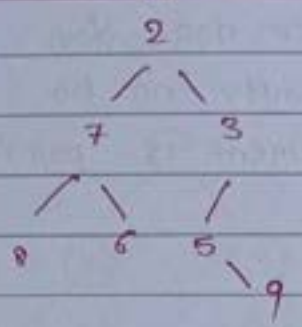
8 6 7 9 5 3 2

Ans

So, here we solve by NRL.
opposite.
We want N First.

classmate
Date
Page

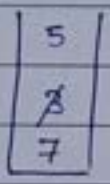


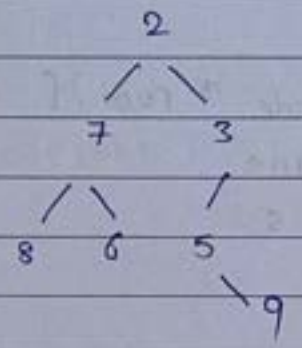


NRL

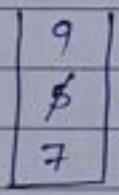
So, in this we put left part first in stack then we put right part.

ans = { 2

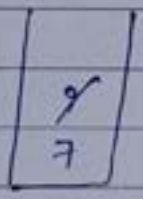




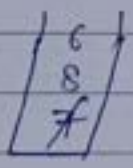
ans = { 2, 3



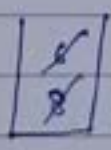
ans = { 2, 3, 5,



ans = { 2, 3, 5, 9



ans = { 2, 3, 5, 9, 7



ans = { 2, 3, 5, 9, 7, 6, 8 }

↓

reverse

8 6 7 9 5 3 2

↳ required answer.

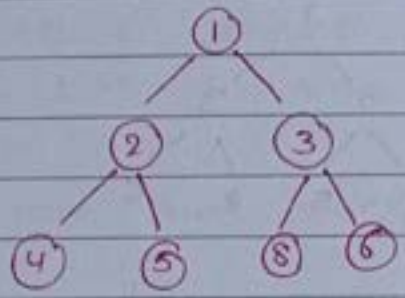
classmate
Date
Page

1. Stack → root ico daal do
2. Jab tak stack empty na ho
   - → print top element & pop it
   - → Left push
   - → Right push.
3. Reverse the answer.

Code :

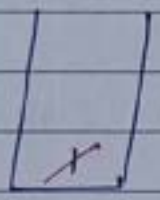
```
vector <int> postorder (Node * root){
        vector <int> ans;
        stack <Node*> s;
        s.push(root);
        Node * temp;

        while (!s.empty()){
            temp = s.top();
            s.pop();

            if (temp → left)
             s.push (temp → left);

            if (temp → right)
             spush (temp → right);

            ans.push_back (temp → data);
        }
    reverse (ans.begin(), ans.end());
    return ans;
}
```

problem 7 : Inorder Traversal (Iterative)

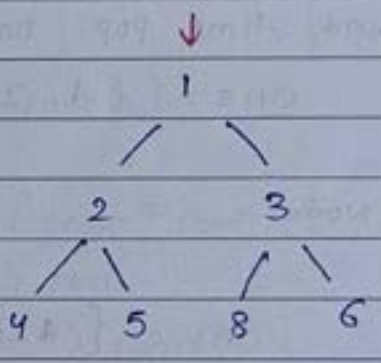


4 2 5 1 8 3 6

Approach

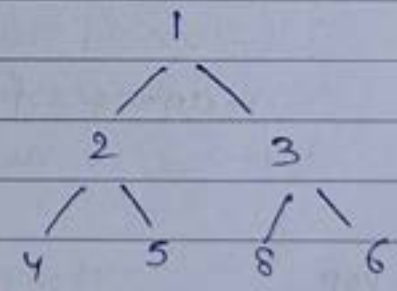


put root



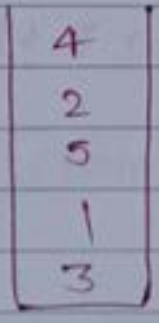
1 को निकालो

उसका Right
तो सुबह
उसका left
} → stack Me dalo



top को निकालो

Right, तो सुबह, left } → stack में डालो।

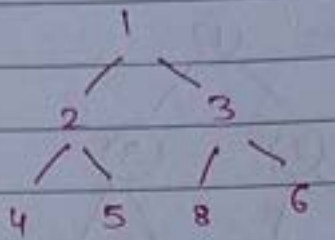


अगर stack का top leaf Nod होगा तो उसको print कर देंगे।

S.top() → right == NULL && S.top() → left == NULL

ans = { 4

~~2~~
5
1
3



अगर stack का top पहले pop हो कर फिर से push हो चुका है तो उसको print कर के pop कर देंगे।

Second time pop nahi karna hai

ans = { 4, 2

5 → leaf Node

~~5~~
1
3

ans = { 4, 2, 5

1 → पहले हो चुका है।

~~1~~
3

ans = { 4, 2, 5, 1,

3 pop

(right, self, left) → push

~~8~~
~~3~~
~~6~~

ans = { 4, 2, 5, 1, 8, 3, 6}

8 → leaf

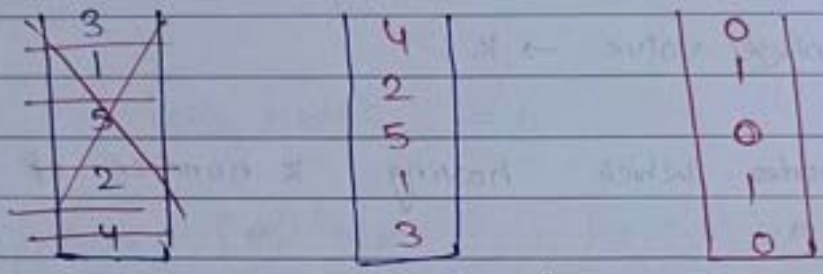
3 → पहले हो चुका है

6 → leaf

ans = {4, 2, 5, 1, 8, 3, 6}

0 → 0 time popped
1 → 1 time popped.

We made here two stack



Code: (By Another Approach)

```
Vector<int> inOrder (Node* root) {
        vector<int> result;
        stack<Node*> nodes;

        Node* Curr = root;
    While (Curr != NULL || !nodes.empty()) {
            While (Curr != NULL) {
                nodes.push(curr);
                Curr = Curr->left;
            }
            Curr = nodes.top();
            nodes.pop();

        result.push_back(Curr->data);
        Curr = Curr->right;
    }
    return result;
}
```

Day - 43

(problems : Tree Traversal)

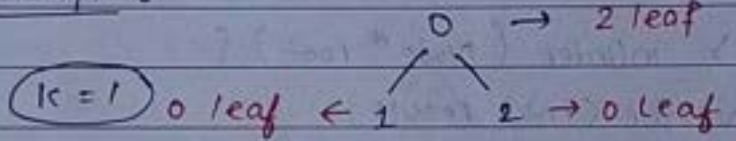
problem 1 : print Nodes having k leaves

Binary tree

integer value → k

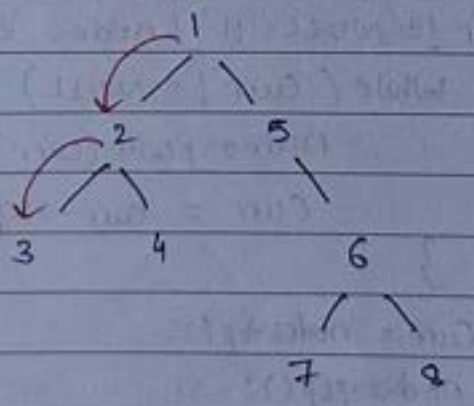
find nodes which having k number of leaves

Example :



return -1

Example :



For 1 → leaf Node = 3, 4, 7, 8 ~~2, 5~~ ⇒ ~~2~~ 4

ans = {~~2~~, 4}

↑

Vector

→ For 2 leaf nodes = 3, 4 → 2 Nodes

K = 2

ans = {[illegible], 2

→ For 3 leaf Nodes = NULL → (0) leaf nodes → Not updated

K = 2

ans = {[illegible], 2,

→ For 4 leaf Node = 0

ans = {[illegible], 2,

→ for 5 leaf Node = 2 = [illegible] {7, 8}

ans = {[illegible], 2, 5

→ For 6 leaf Node = 2 = {7, 8}

ans = {[illegible], 2, 5, 6

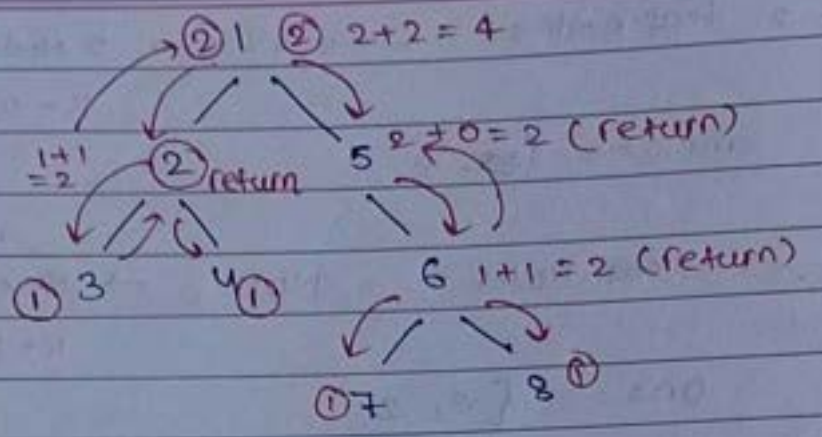
→ For 7 leaf Node = 0

ans = {[illegible], 2, 5, 6,

→ For 8 leaf node = 0

ans = {[illegible], 2, 5, 6}

↳ return this.



Code :

```
int find ( Node * ptr , vector <int> & ans,  int K){
            if (ptr == NULL)
                return 0;

            if (ptr → left == NULL && ptr → right == NULL)
                return 1;

        int left = find (ptr→left, ans, K);
        int right = find ( ptr → right , ans, K);

        return left + right;
    }

Vector <int> btWithKleaves (Node * ptr , int K){
            vector <int> ans;
            find (ptr, ans, K);
            if (ans empty ())
                ans.push_back (-1);

        return ans;

    }
```

Problem 2 : print Binary Tree levels in sorted order :

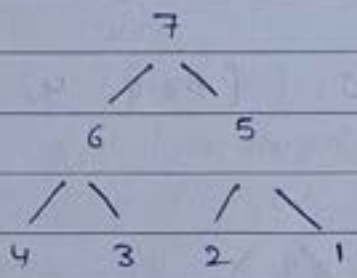
Example : 1

N = 7

arr[] = { 7, 6, 5, 4, 3, 2, 1 }

↓

Binary Tree from this array



levels :

7

6 5

4 3 2 1

sort each levels :

7

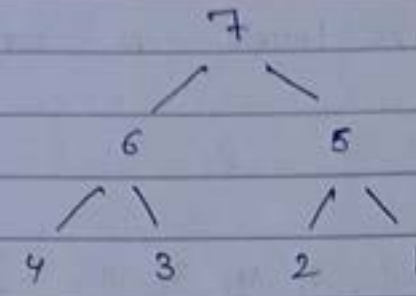
5 6

1 2 3 4

ans = { 7, 5, 6, 1, 2, 3, 4 }

return.

Approach :



Level 0 → $2^0 = 1$

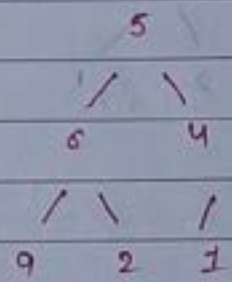
Level 1 = $2^1 = 2$

Level 2 = $2^2 = 4$

At last level, it may be gives less Nodes

Because

arr[] = {5, 6, 4, 9, 2, 1}



At last level only 3 nodes.

We will sort here level wise.

So, we use ~~the~~ Concept of level order traversal.

{ 7 6 5 4 3 2 1 }

At 1st level → only 1 element / Nodes

↳ No need to sort

We will start sorting with level 2.

Approach :

arr = { 1 | 3 2 | 6 7 8 9 | 4 9 }

1 2 3 4

Level 1 → put in ans array. (K = 1)

ans = { 1,

Level 2 → temp[] = { 3, 2 } K = (K * 2) = 2

sort

{ 2, 3 }

↳ stored in ans.

ans = { 1, 2, 3

Level 3 → temp[] = { 6, 7, 8, 9 } K = (K * 2) = 4

sort

{ 6, 7, 8, 9 }

↳ stored in ans.

ans = { 1, 2, 3, 6, 7, 8, 9 K = (K * 2) = 8

Level 4 : temp = { 4, 9 } ↓ If not 8, then जितना है उतना निकाल लो।

sort

{ 4, 9 }

↳ stored in ans.

ans = { 1, 2, 3, 6, 7, 8, 9, 4, 9 }

We have to stored in 2D vector → level wise:

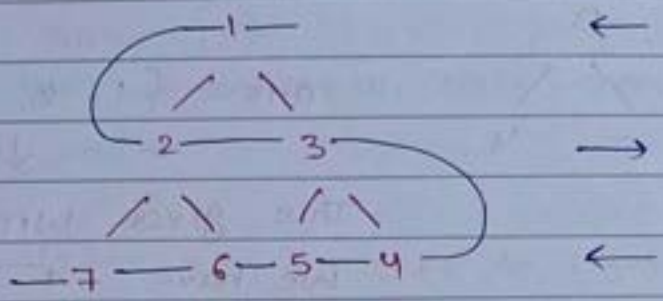
ans = { { 1 }, { 2 3 }, { 6 7 8 9 }, { 4 9 } }

↳ return.

Code :

```
Vector<vector<int>> binTreeSortedLevels (int arr[], int n) {
vector <vector <int>> ans ;

  int K = 1;      // for level 1 : No. of Nodes
  int i = 0;

  while (i < n) {

     int numNodes = min (k, n-i)
                  // No. of nodes at current level.

    Vector <int> temp (arr + i, arr + i + numNodes);

    Sort (temp.begin(), temp.end());

      ans.push_back (temp);

      i = i + numNodes;
       K *= 2;
  }

  return ans;
}
```

problem 3: level order Traversal in spiral form



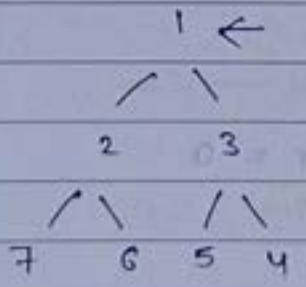
ans = {1, 2, 3, 4, 5, 6, 7}

1st level = Right to left
2nd level = Left to Right
3rd level = right to left

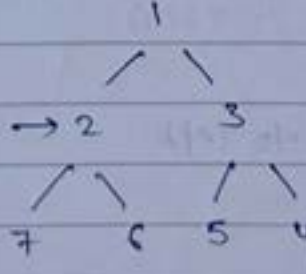
for odd level = Right to left
For even level = left to right

We use Queue to solve this problem:

Queue: ~~1~~ ~~2~~ 3

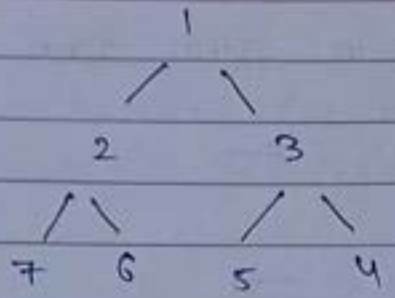


ans = {1



q ~~3~~ 7 6 5 6

ans = {1, 2, 3



q: | ~~1~~ 6 5 4

ans = {1, 2, 3 7

↓

This gives wrong answer
We have 4 after 3
but we get 7

What we have to change:

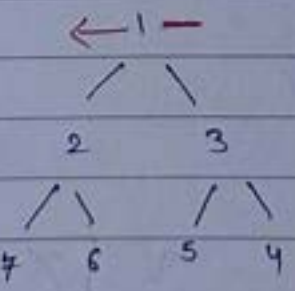
① When we move from Right to left
i.e. odd levels

→ then we put Right Node first in queue then we put left Node

② When we move from left to Right

→ then we put left Node first in queue the we put right node.

Example:



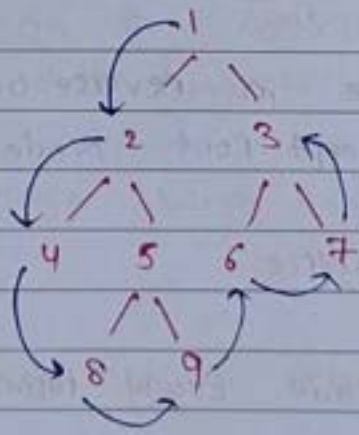
Step 1: Level 1 → Right to left

Code:

```
Vector <int> findSpiral (Node * root) {
    Vector <int> ans;
    if (root == NULL)
      return ans;
    queue < Node *> q;
    stack <Node *> s;

    q.push(root);
    bool dir = 0;

  while ( !q.empty()){
      if (dir == 0){
          int size = q.size();
          while (!q.empty()) {
              Node * temp = q.front ();
              q.pop();

              if (temp → right)
              s.push (temp → right);

              if (temp → left)
              s.push (temp → lef);

              ans.push_back (temp → data);
      }
      dir = 1;
    }
    else {
          int size = q.size();
          while ( !q.empty()) {
              Node * temp = q.front();
```

```
            q.pop();
            if (temp -> left)
            s.push(temp -> left);

            if (temp -> right)
            s.push (temp -> right);

            ans.push_back (temp->data);
        }
        dir = 0;
    }
    while (!s.empty()) {
        q.push(s.top());
        s.pop();
    }
  }
  return ans;
}
```

problem 4: Boundary Traversal of binary Tree



ans = 1 2 4 8 9 6 7 3

The problem divides in 3 parts:

→ Left boundary Nodes:

Root se start karna hai
left side traverse karke print karte
jana hai.
When you find leaf Node, first print it
and the 2nd part comes

→ Leaf Nodes:

jahaan par left bounday khatm hua uske
baad saara leaf Node Traverse karenge
aur print karenge.
jo Node left ya ~~post~~ right boundary
me aata hai usko chor denge.

→ Reverse right most boundries:

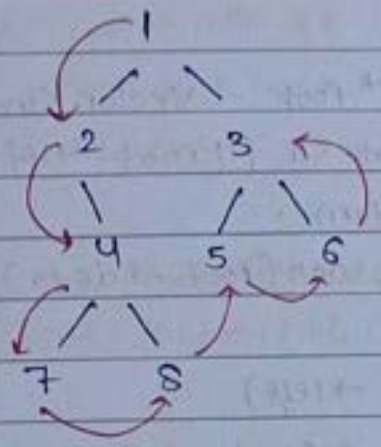
Start from the right most node of root
and it is leaf node:

You can traverse in reverse order and
print all except root Node.

① left subtree

② leaf Node Except leftmost & rightmost

③ right subtree in reverse.



left : {1, 2, 4, 7}
↳ leaf Node then stop

leaf : {*, 8, 5, *}
↓ ↳ right most
left most

right : {6, 3, *}
↳ root

Here we first call the left subtree.
↳ It complete the work.

then, we go to leaf Node,
↳ It complete the work.

then, we go with reverse right subtree.

then at last return the ans.

Code :

```
Void left_sub (Node * root, vector <int> & ans ){
        if ( !root || (!root -> left && !root -> right))
            return ;
        ans.push_back (root -> data);

        if (root -> left)
            left_sub (root -> left, ans);
        else
            left_sub (root -> right, ans);

    return;
}

Void right_sub (Node * root , vector <int> & ans ){
        if (!root || (!root -> left && !root -> right))
            return ;

        if (root -> right)
            right_sub (root -> right, ans);
        else
            right_sub (root -> left, ans);

        ans.push_back (root -> data);

}

Void leaf_sub (Node * root, vector <int> & ans ){
        if ( root == NULL ){
            return;
        }
```

```
    if(!root->left && !root->right){
        ans.push_back(root->data);
        return;
    }
    leaf_sub(root->left, ans);
    leaf_sub(root->right, ans);
}

Vector<int> boundary(Node *root){

    Vector<int> ans;
    ans.push_back(root->data);
    left_sub(root->left, ans);
    if(root->left || root->right)
    leaf_sub(root, ans);
    right_sub(root->right, ans);

    return ans;
}
```

classmate
Date
Page

(Day - 44)

(Tree - Hard problem)

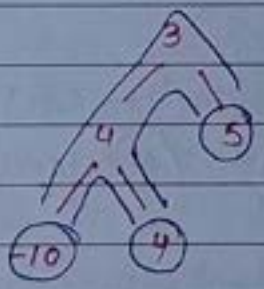
Problem 1 : Maximum path sum between 2 special Nodes

Given a binary tree
↳ each nodes element contain a number

Task : Maximum possible path sum for two special nodes
Special Node is a node which is connected to exactly one different Nodes

दो Node के बीच का sum और दोनो Node सिर्फ एक Node से Connect हो।

Example:



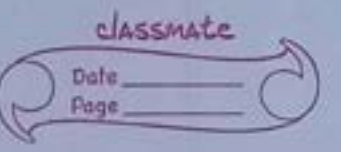
path possible :

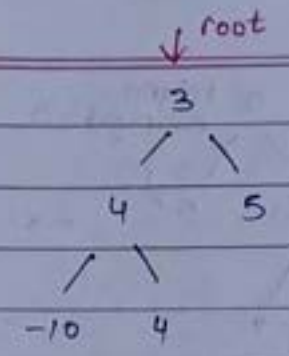
(i) $-10 \rightarrow 4 \rightarrow 4$ Sum = -2

(ii) $-10 \rightarrow 4 \rightarrow 3 \rightarrow 5$ sum = 2

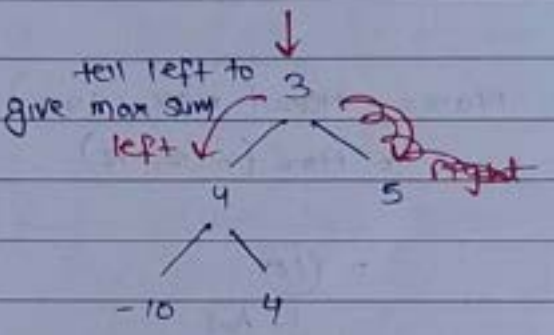
(iii) $4 \rightarrow 4 \rightarrow 3 \rightarrow 5$ sum = (16) ↓ Max

How to approach this to solve :

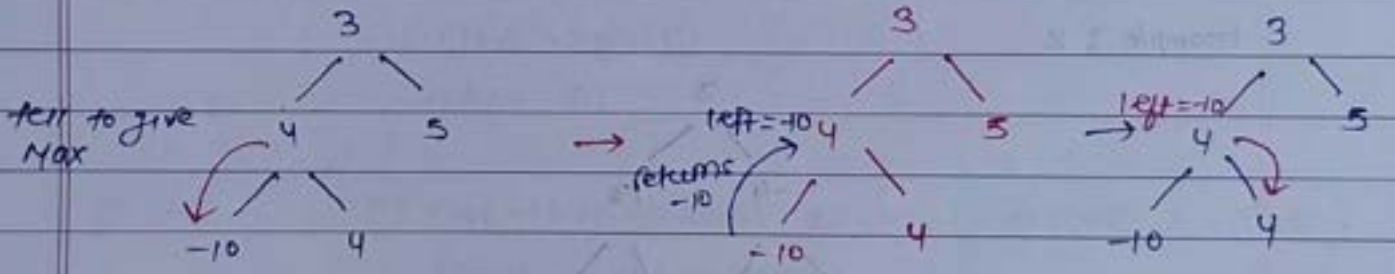


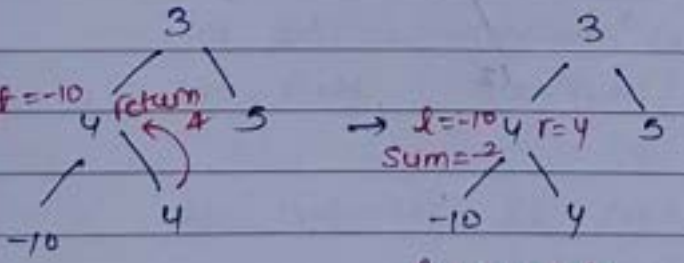


3 बोलेगा दोनों side का Max sum लाकर दो,
और left + right + root → data हो जाएगा।



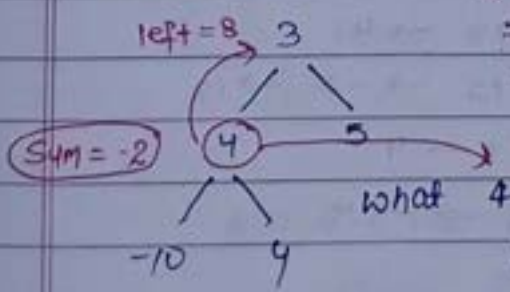
⇒ Max = INT.MIN





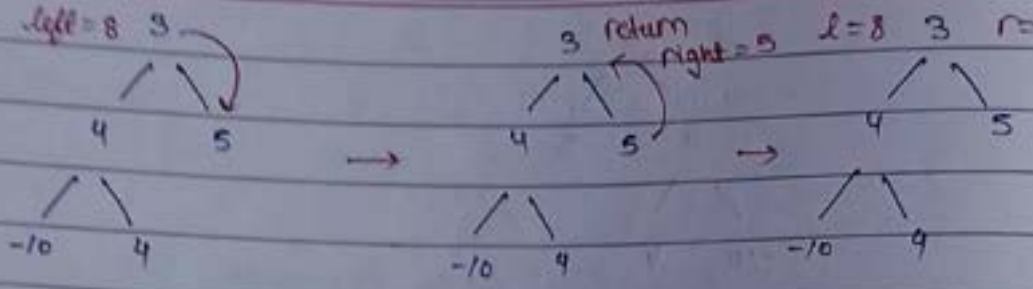
Max = ~~INT.MIN~~
-2

l + r + data
= -10 + 4 + 4
= -2



what 4 return
↳ it has 2 path,
↳ return the Max.
= root → data + max(left, right)

4 + max(-10, 4)
4 + 4
= 8



sum = root → data + left + right

= 3 + 8 + 5

= (16)

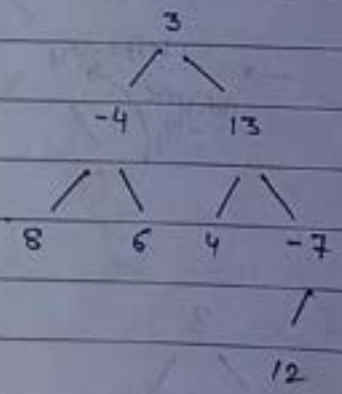
Max = Max (max, sum)

= Max (-12, 16)

= (16)

↳ Ans

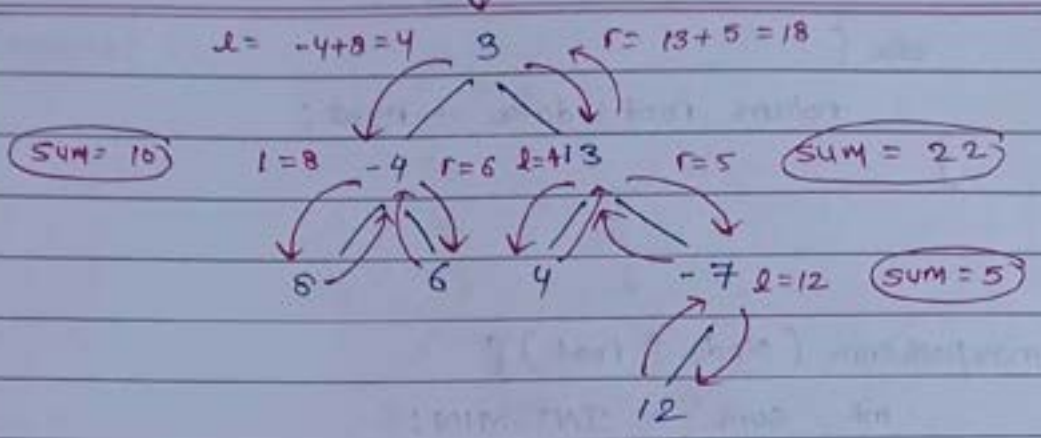
Example : 2



possible path :

10	8 → -4 → 6
24	8 → -4 → 3 → 13 → 4
(25)	8 → -4 → 3 → 13 → -7 → 12
22	6 → -4 → 3 → 13 → 4
23	6 → -4 → 3 → 13 → -7 → 12

Max



Max = ~~MIN~~

~~10~~

~~22~~

(25)

Code :

```
int maxSum (Node * root, int & sum) {
        if (root == NULL)
            return 0;

        if (root → left == NULL && root → right == NULL)
            return root → data;

    int left = maxSum (root → left, sum);
    int right = maxSum (root → right, sum);

    if (root → left && root → right) {
        sum = max (sum, root → data + left + right);
        return root → data + max (left, right);
    }

    else if (root → left && root → right == NULL) {
        return root → data + left;
    }
```

```
        else {
            return root->data + right;
        }
    }
```

```
int maxPathSum (Node * root) {
        int sum = INT_MIN;
        int val = maxSum (root, sum);
        if (root -> left && root -> right)
            return sum;

        return max (val, sum);
    }
};
```

Problem 2 : Burning Tree

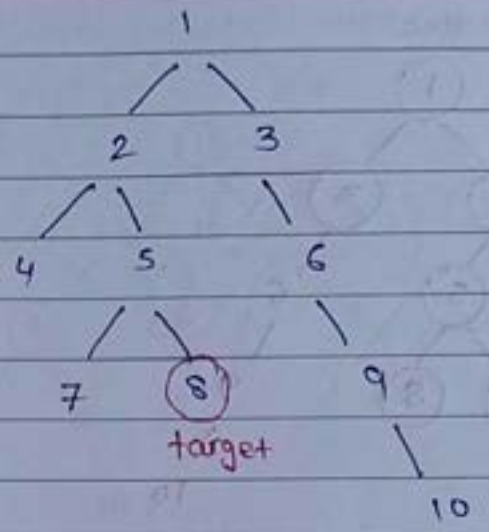
Given a Binary Tree
and a target value.

Task to find minimum time required to burn the complete binary tree, if target is set on fire.

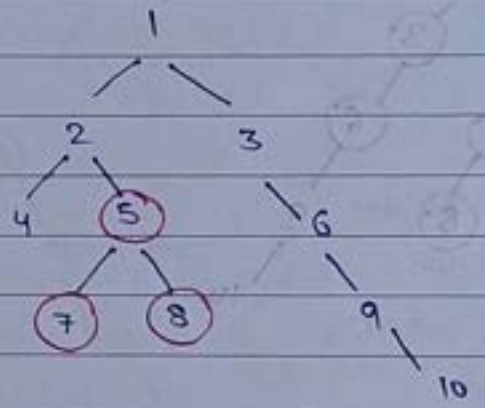
In 1 second all nodes connected to given node get burned.

Note : The tree contains unique values.

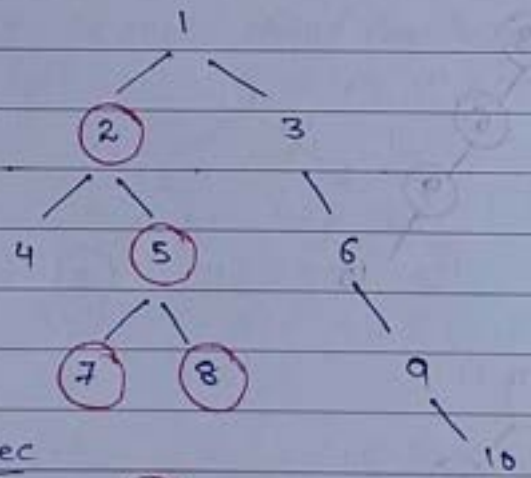
Example:



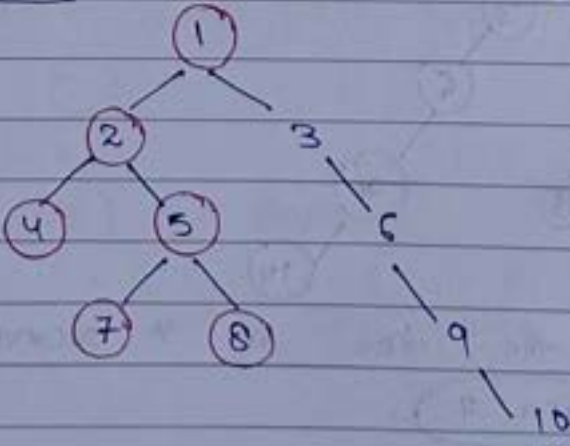
After 1 sec



After 2 sec

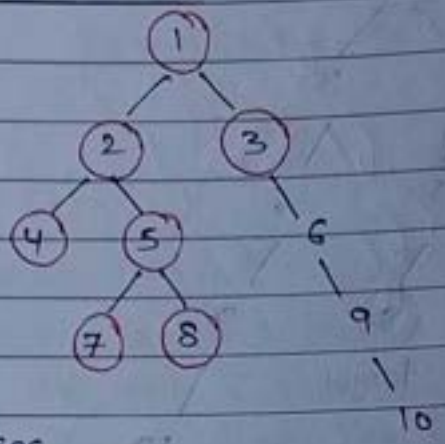


After 3 sec

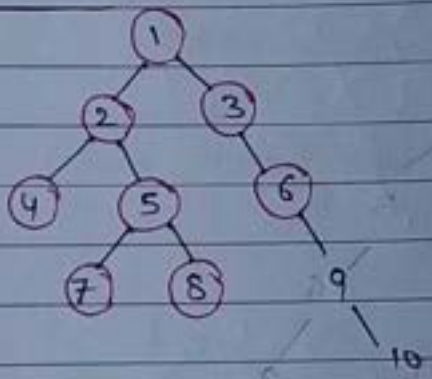


classmate
Date
Page

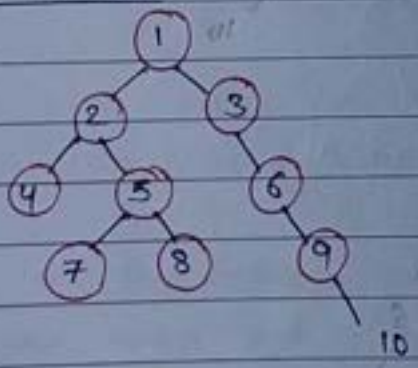
After 4 sec



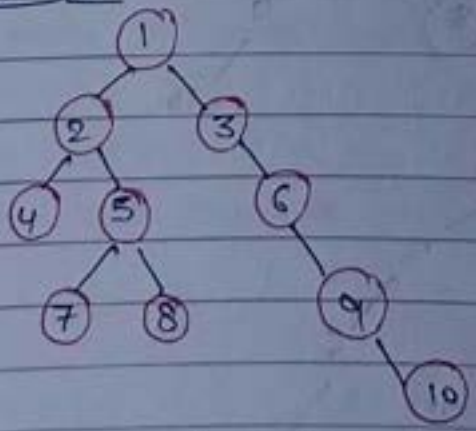
After 5 sec



After 6 sec



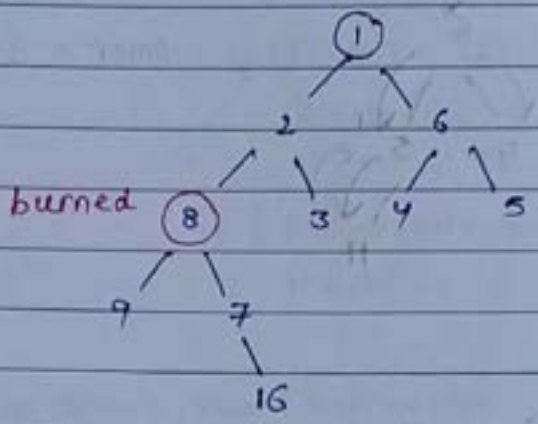
After 7 sec



It burn all the tree in 7 second.

return 7.

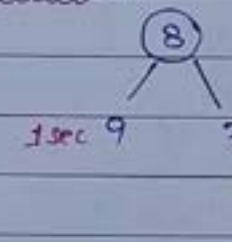
Example 2: If any Middle level Node is Burned.



problem divided in two parts:

lower & upper

For lower:



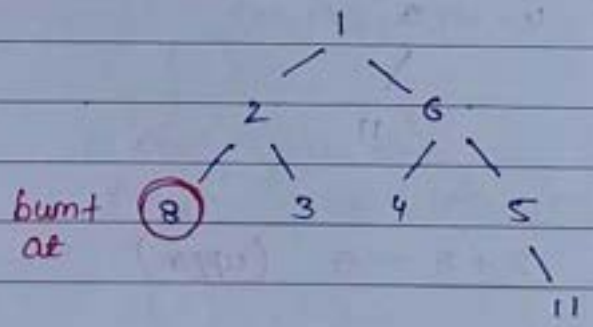
Height = 3

Burned in 2 sec

ans = (Height - 1)

↳ for lower part.

For upper: (Burned Node Considered as leaf Node)

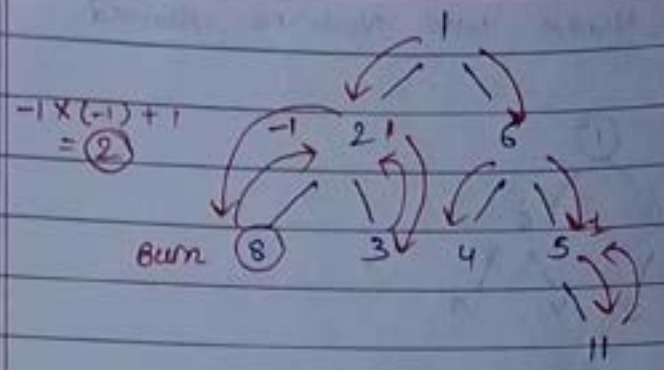


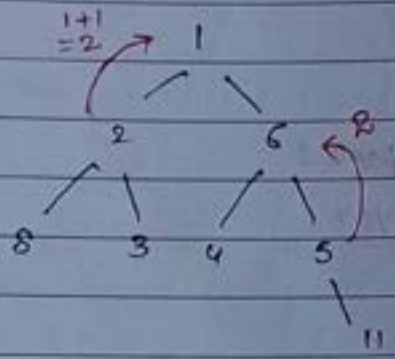
जिस side से burnt हुआ उसको Negative से denote कर देंगे। (-1)

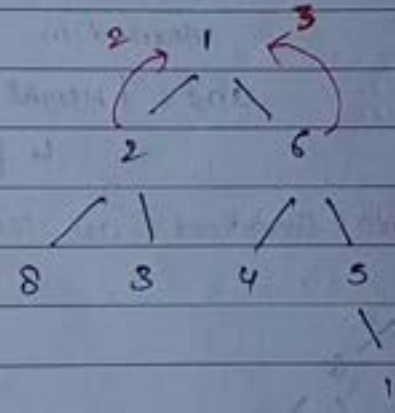
(-ve) → Burn हो गया

और उतना time लगा

classmate
Date
Page







Time = 2 + 3 = 5 (upper)

ans = max (lower_timer, upper_timer)
= max (2, 5)

ans = 5

Code 8

```
int Burn (Node * root, int target, int &timer) {
        if (root == NULL)
            return 0;

        if (root -> data == target)
            return -1;

    int left = Burn (root -> left, target, timer);
    int right = Burn (root -> right, target, timer);
      // left side burnout
      if (left < 0) {
          timer = max (timer, abs (left) + right);
          return left - 1;
      }
      // right side burnout
      if (right < 0) {
          timer = max (timer, abs (right) + left);
          return right - 1;
      }
      // Both positive.
      return max (left, right) + 1;
}

Void find (Node * root, int target, Node * &temp) {
        if (root == NULL)
            return;

        if (root -> data == target) {
            temp = root;
            return;
        }
```

```
        find (root->left, target, temp);
        find (root->right, target, temp);
    }

int height (Node * root){
    if (root == NULL)
        return 0;

    return 1 + max (height (root->left), height (root->right));
}

int minTime (Node * root, int target){

    int timer = 0;
    Burn (root, target, timer);
    Node *temp;
    find (root, target, temp); // find address of burn node
    int num = height (temp) - 1; // find height of below
    return max (timer, num);    part of the tree.

}
```

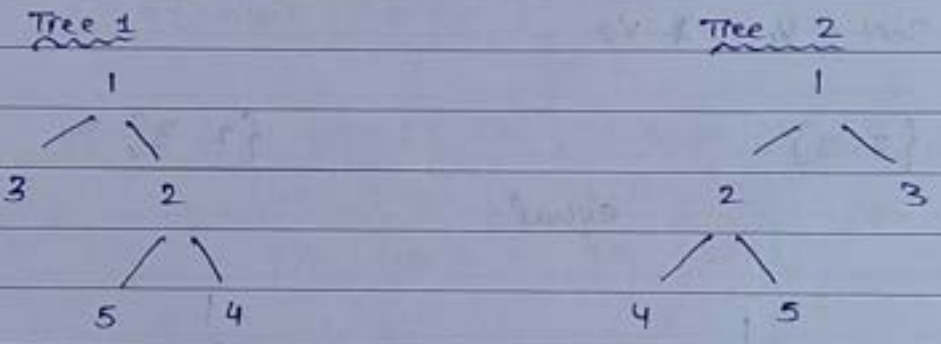
problem 3: Check if all level of two trees are anagram or not

Given two binary trees
↳ having same number of nodes

Task: check at each level that the levels are anagram or not

We use level order traversal here
we Queue for storing the Nodes of each level.

Example:



q1 [X]

q2 [X]

size1 = 1

size2 = 1

same

V1 [1]

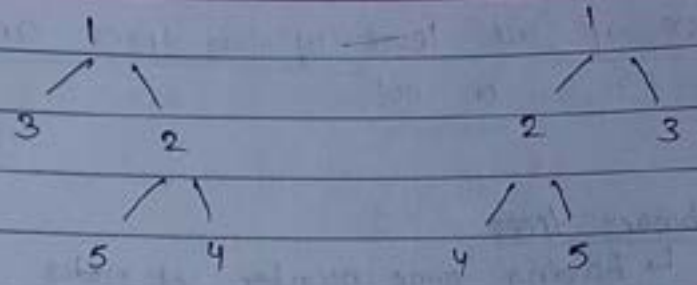
V2 [1]

Sort vector

{1}

{1}

both equal proceed



q_1 | ~~3~~, ~~2~~ | q_2 | ~~2~~, 3

size1 = 2 size2 = 2

$V_1 = \{3, 2\}$ $V_2 = \{2, 3\}$

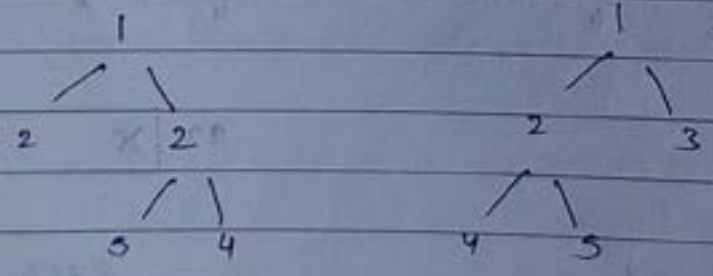
size--

~~size = 0~~

Sort V_1 & V_2

$\{2, 3\}$ $\{2, 3\}$

equal.



q_1 | ~~3~~ ~~4~~ | q_2 | ~~4~~ ~~5~~

$V_1 = \{5, 4\}$ $V_2 = \{4, 5\}$

sort

$\{4, 5\}$ $\{4, 5\}$

equal → return 1

size = 0

4 loop bread

Code:

```
Bool areAnagrams (Node * root1, Node * root2){
    if (root1 == NULL && root2 == NULL){
        return true;
    }

    if (root1 == NULL || root2 == NULL){
        return false;
    }

    queue<Node *> q1;
    queue <Node *> q2;
    q1.push (root1);
    q2.push (root2);

    while(!q1.empty() && !q2.empty()){
        int size1 = q1.size();
        int size2 = q2.size();

        if (size1 != size2){
            return false;
        }

        Vector <int> V1;
        Vector <int> V2;

        while (size1 > 0){
            Node* node1 = q1.front();
            Node* node2 = q2.front();
            q1.pop();
            q2.pop();
```

```
            v1.push_back(node1->data);
            v2.push_back(node2->data);

            if(node1->left)
                q1.push(node1->left);

            if(node1->right)
                q1.push(node1->right);

            if(node2->left)
                q2.push(node2->left);

            if(node2->right)
                q2.push(node2->right);

            size--;
        }

        sort(v1.begin(), v1.end());
        sort(v2.begin(), v2.end());

        if(v1 != v2)
            return false;
    }

    return true;
}
```

Problem 4: Construct tree from Inorder & preorder

Given 2 Array: inorder & preorder

Tree contain duplicate elements.

Task: Construct a tree and print the postorder Traversal.

Inorder: 3 1 4 0 5 2 (LNR)

Preorder: 0 1 3 4 2 5 (NLR)

preorder: 1st Node is always root Node.

root Node.

Tree = { 0

check in Inorder

3 1 4 0 5 2

left side right side.

preorder: 0 1 3 4 2 5

1 Comes left side.

0 Tree = { 0, 1

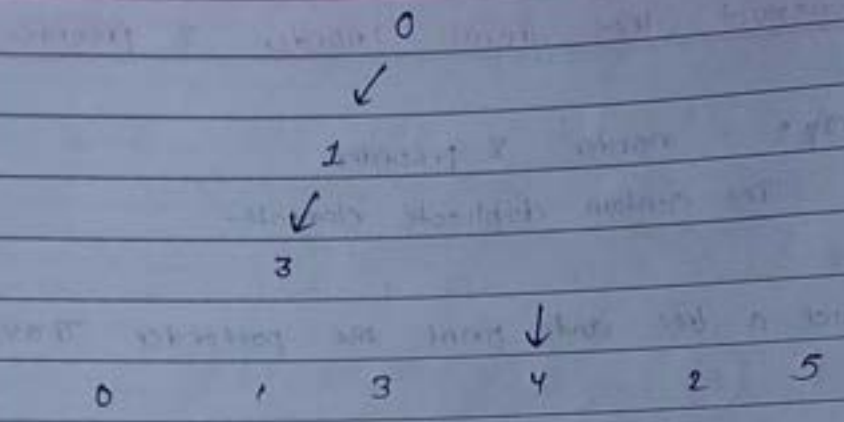
1

3 1 4

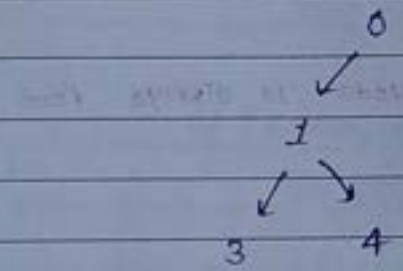
left side right side.

0 1 3 4 2 5

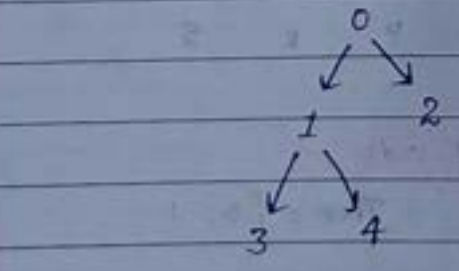
3 Comes left side of 1



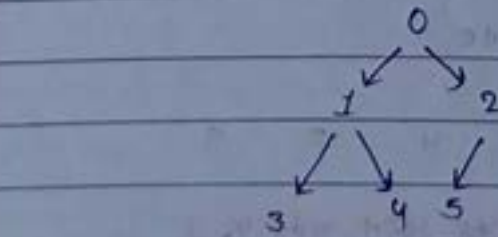
4 comes right of 1.



2 comes right of 0



5 comes left of 2.



Tree

preorder : एक - एक करके traverse करेंगे।

Inorder : direction बता देगा।

problem 5 : Construct tree from postorder & inorder

In this problem

Inorder direction बता देगा।

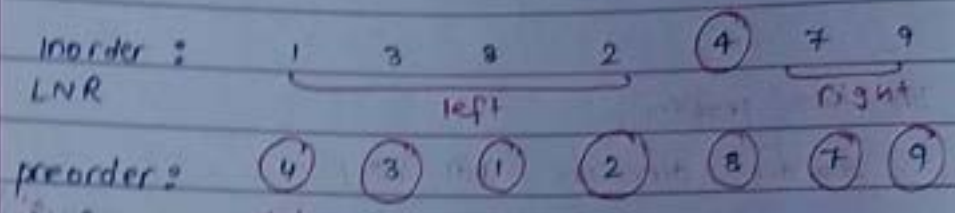
postorder : एक एक करके traverse करेंगे।

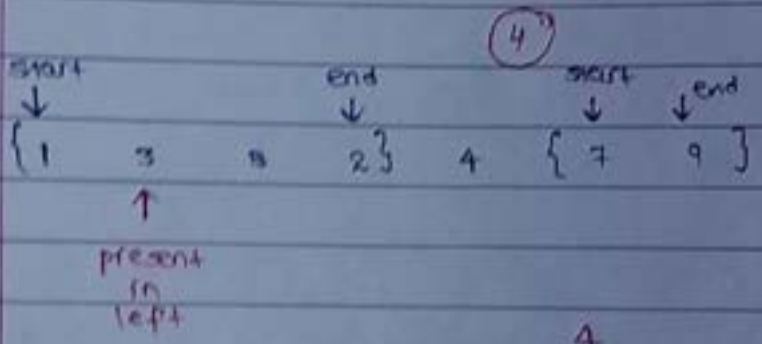
(पीछे से).

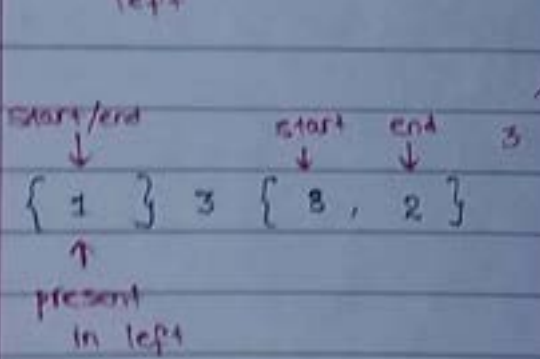
Day - 45

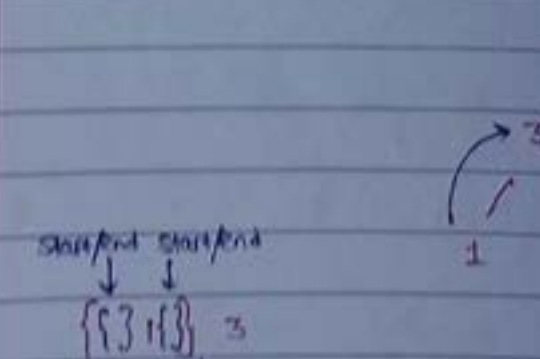
(Binary Tree & Binary Search Tree)

problem 1: Construct tree from inorder & preorder







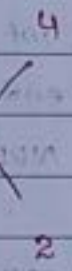


All Null

return.

8 present in right of 3

{1} 3 {8, 2}



start/end

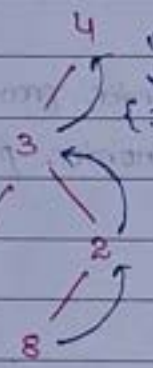
start/end.

{8} 2 { }

left

right

8 present in left of 2



Done

Done

return.

{1} 3 {8, 2}

NULL

NULL

{8} 2 { }

done

return.

NULL

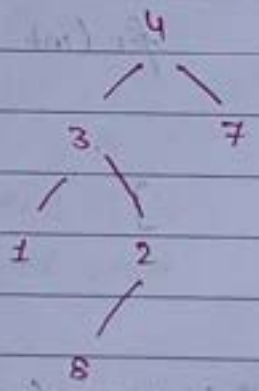
NULL

{ } 8 { }

return.

4 {7, 9}

present right side.

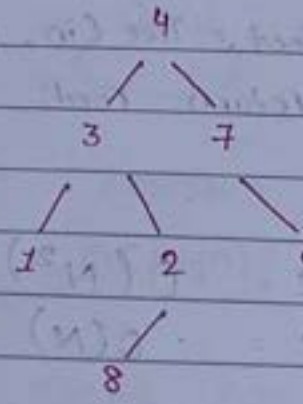


NULL

start/end

{ } 7 {9}

present right side of 7.



required tree:

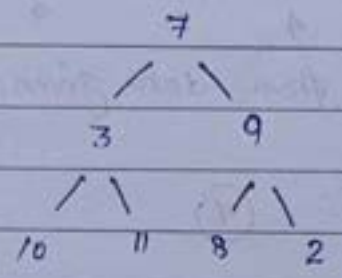
Code :

```
Node* Tree(int inorder[], int preorder[], int start, int end, int index){
        if (start > end)
            return NULL;

    Node * root = new Node (preorder[index]);
    int i = find (inorder, preorder[index], start, end);

root → left = Tree (inorder, preorder, start, i-1, index + 1);
root → right = Tree (inorder, preorder, i+1, end, index + (i - start) + 1);

return root;
}

int find (int inorder[], int num, int start, int end ){
        for (int i = start ; i <= end; i++ ){
            if (inorder [i] == num)
                return i;
        }
}

Node * buildTree (int in[], int pre[], int n){
        Node * root;
        root = Tree (in, pre, 0, n-1, 0);
        return root;
}
```

T. C = $O(N^2)$

S. C = $O(N)$

Binary Search Tree (BST)



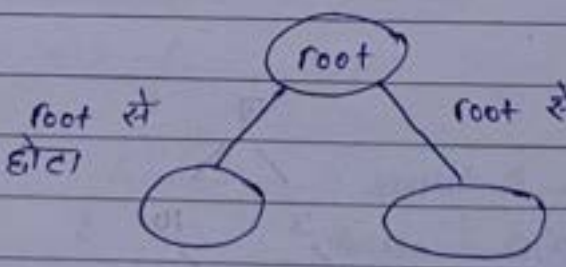
How we search any element ?
(search 2)

root
↓
root → left
↓
root → right

By this we can traverse whole tree.

Time Complexity : O(N)

But, we want to search data faster :
We store the data in sorted order



इस तरह से searching average case में O(log n) ही होगा।

This type of tree is called BST (Binary search Tree).

Let's understand With an Example:

(7) 3 10 8 2 4 11 9 1

Construct a BST from data given

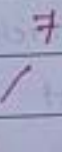
7 → (7)

left = smaller than 7
right = larger than 7

3 →

3 < 7

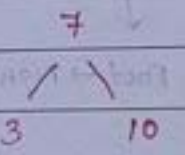
left of 7



10 →

10 > 7

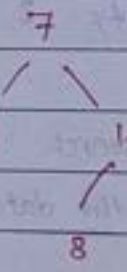
right of 7



8 →

8 > 7

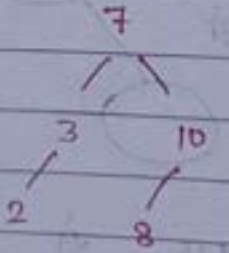
8 < 10



2 →

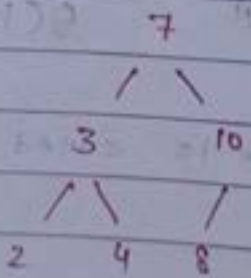
2 < 7

2 < 3



4 →

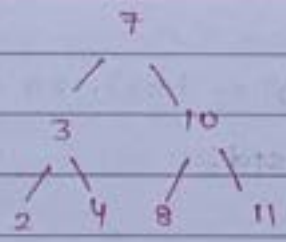
4 > 3



11 →

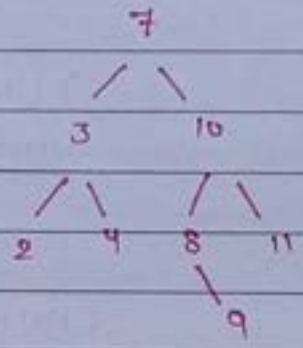
11 > 7

11 > 10



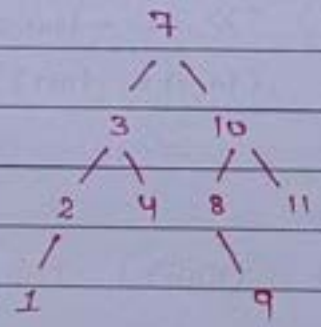
9 →

9 > 8



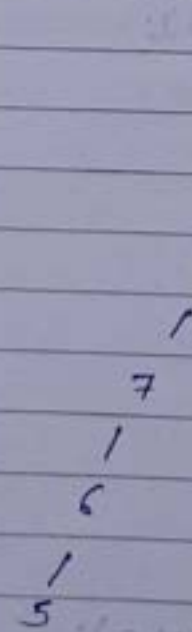
1 →

1 < 2



On Average case searching will take logn time.

For Worst case O(N)



search 5 takes O(N) time.

Code :

```
#include <iostream>
using namespace std;

class Node {
  public:
    int data;
    Node * left;
    Node * right;

    Node (int x) {
        data = x;
        left = NULL;
        right = NULL;
    }
}
int main () {
    int arr[10] = { 10, 13, 4, 8, 11, 19, 2, 7, 18, 23};
                Node * root = NULL;
        for (int i = 0; i < 10; i++) {
                root = BST (root, arr[i];
                    return 0; inorder (root);
                } return 0;
}
Node * BST ( Node * root, int value) {
            if (root == NULL) {
            root = new Node (value);
            return root;
        }

            if (root -> data > value) {
            root -> left = BST (root -> left, value);
                return root;
            }
```

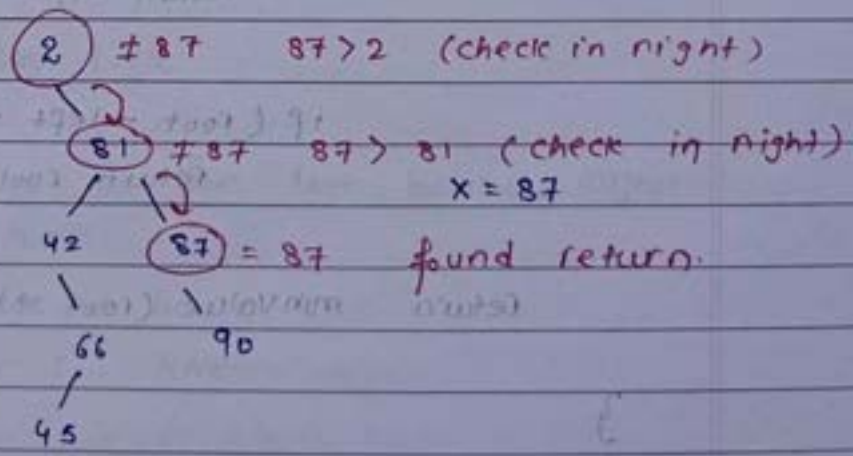
```
else {
    root->right = BST(root->right, value);
    return root;
}
```

```
Void inorder(Node * root) {
    if (root == NULL)
        return;

    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}
```

Problems on BST (Binary Search Tree).

problem 1 : Search a Node in BST.

Example :



Code :

```
bool search (Node *root, int x) {
    if (root == NULL)
        return 0;

    if (root->data == x)
        return 1;

    if (root->data > x)
    return search (root->left, x);

    if (root->data < x)
    return search (root->right, x);
}
```

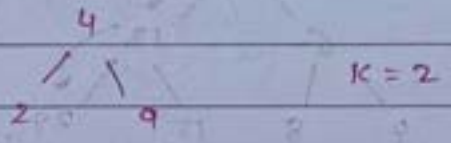
problem 28 Minimum element in BST

Code:

```
int minValue (Node* root) {
    if (root == NULL)
        return -1;

    if (root->left == NULL)
        return root->data;

    return minValue (root->left);
}
```

Problem 3: K^{th} largest Element in BST :

Example :



Ans = 4

Approach 1 :

Store the tree in an array

arr = [2, 4, 9];

K = 2

find total Node -(2)+1

and return.

But here an Extra space for array is required.

Approach 2 :

Inorder : ~~LR~~ LNR

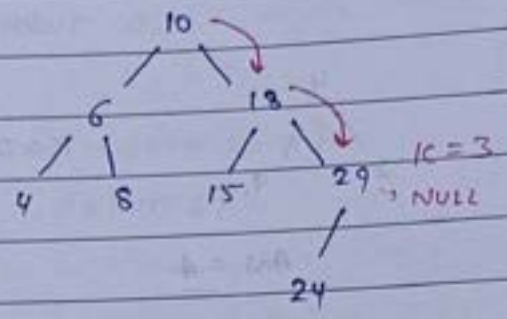
But we have to do from last because largest present at last.

So, we use here

Inorder : RNL

Example:

k = 4



4th largest

RNL

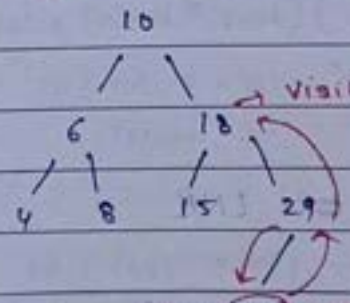
K = 4

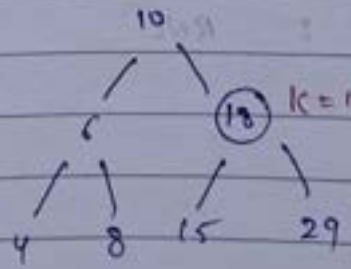
traverse to right until found NULL

1 st larget = 29

k-- => k = 3

then it go to left.





then visit left.

Problem 4 : Check for BST

Approach 1 :

Traverse using inorder

store in array

if sorted in ascending order

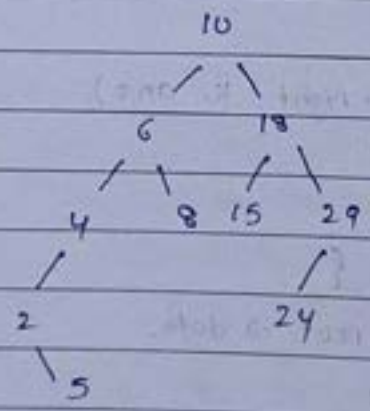
↳ thus it is BST

otherwise

↳ not BST

so, this Extra space required.

Approach 2 :

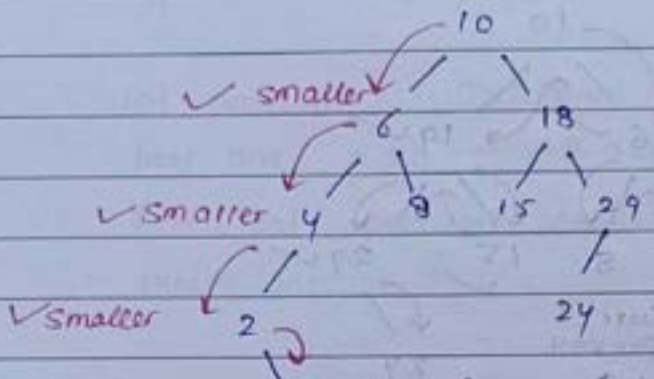


Traverse

↳ if we go left check element is smallest

↳ if we go right check element is larger.

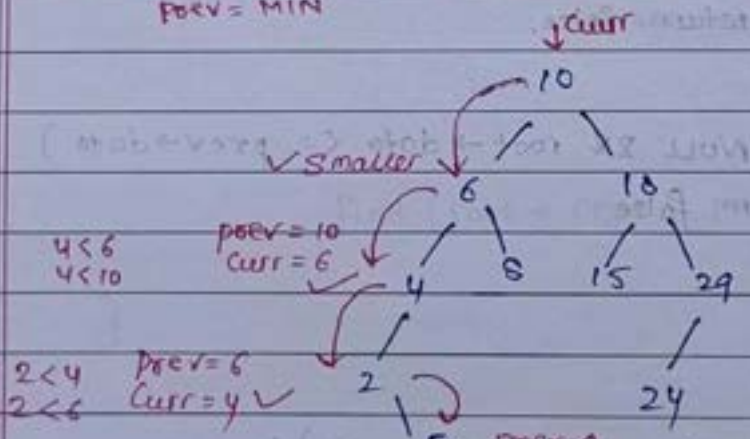
Example:



But also is not BST
because it also less than 4
But here it is greater.

This approach also fails

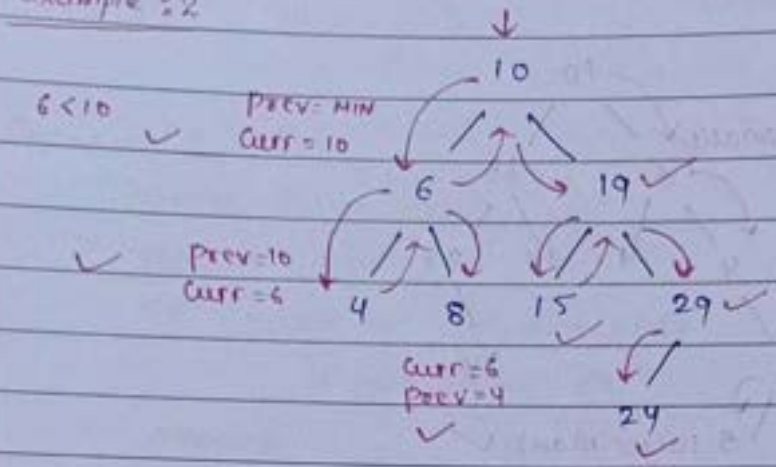
An next Approach we also check with prev also.



In this Example, 5 is greater than 4
but in BST it is not possible.

So, it is not BST.

Example : 2



Code :

```
bool isValidBSTUtil (Node* root, Node * & prev){
        if (root == nullptr)
            return true;

        if (!isValidBSTUtil (root→left, prev))
            return false;

        if ( prev != NULL && root→data <= prev→data )
            return false;

    prev = root;

    return isValidBSTUtil (root→right, prev);
}

bool isBST( Node * root){
        Node * prev = NULL;

    return isValidBSTUtil (root, prev);
}
```

(Day - 46)

(BST & Morris Traversal)

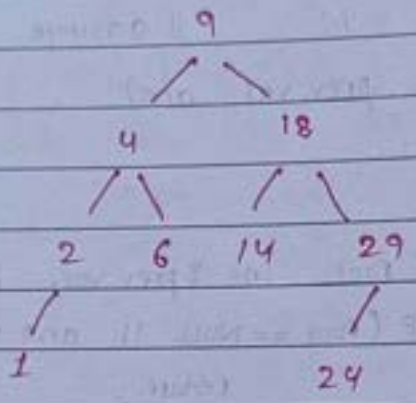
Code : for (check for BST)

```
int prev_val = INT_MIN;
bool ans = 1;          // assume tree is BST
find (root, prev_val, ans);
return ans;

Void find (Node* root, int &prev_val, bool &ans){
        if (root == NULL || ans == 0)
            return;

    find (root->left, prev_val, ans);

    if (prev_val >= root->data) {
        ans = 0;
        return;
    }

    find (root->right, prev_val, ans);

}
```

Problem : sum of K smallest element in BST.

Example

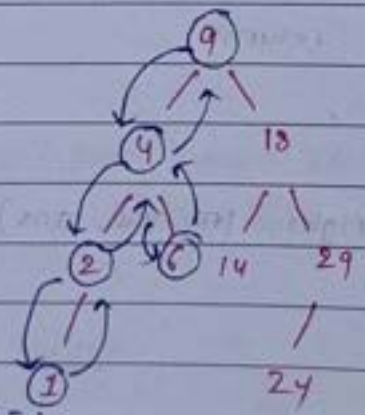
K = 5



In BST, smallest element lies in left side.

Do inorder traversal and find the sum.

Sum = 0



Sum = 1

Sum = 3

sum = 7

Sum = 13

Sum = 22 Count = 0

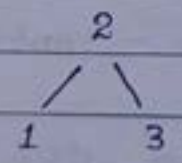
ans = 22

Code :

```
Void find (Node * root, int &K, int &total) {
        if (root == NULL)
            return;

        find (root -> left, K, total);
        K--;
        if (K >= 0)
        total += root -> data;

        find (root -> right, K, total);
}

int sum (Node * root, int K) {
        int total = 0;
        find (root, K, total);
        return total;
}
```

problem : Inorder successor in Binary search tree

Example :



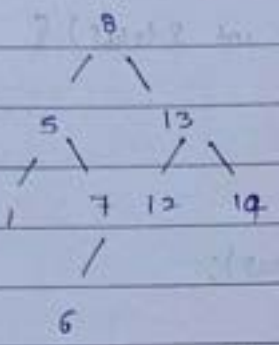
x = 2

Inorder : 1 2 3

successor of 2

ans = 3

Example :



x = 5

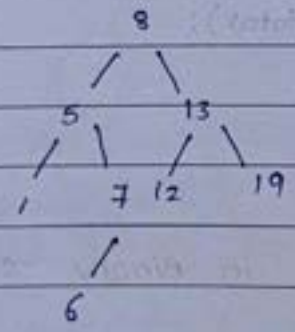
Inorder : 1 5 6 7 8 12 13 19

Successor of 5

ans = 6

Approach 2 :

X = 5



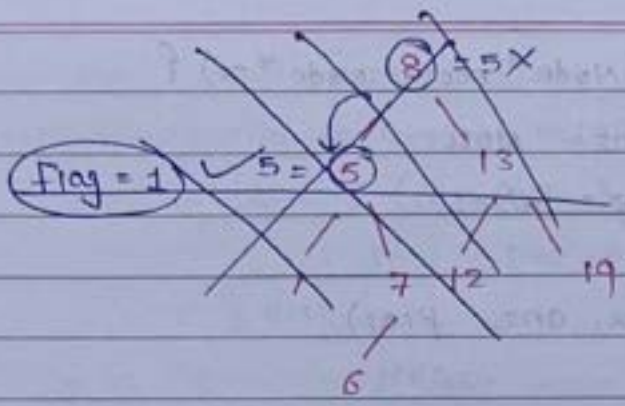
Initial flag = 0

When

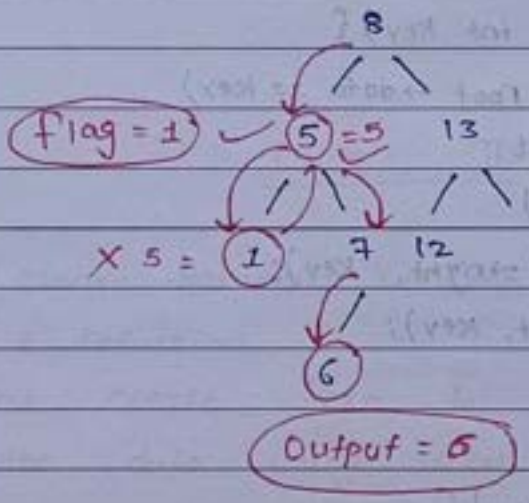
x meets

make flag = 1

and return the next node.



Start with inorder traversal :



Code : (doesn't work) → Not Correct for all cases.

```
Void find (Node * root, Node * x, Node * &ans, bool & flag) {
        if (root == NULL)
            return;

        find (root → left, x, ans, flag);
        if (flag) {
            ans = root;
            return;
        }
        if (root == x)
            flag = true;

        find (root → right, x, ans, flag);
}
```

```
Node * inorderSuccessor (Node * root, Node * x) {
        Node * ans = NULL;
        bool flag = false;

        find (root, x, ans, flag);
        return ans;
    }
```

Optimal Code :

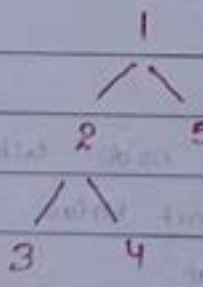
```
Node * find ( Node * root, int key) {
        if (root == NULL || root → data == key)
            return root;
        if (root → data < key)
            return find (root → right, key);
        return find ( root → left, key);
}

Node * findMin (Node * node) {
        if (node == NULL)
            return NULL;
        while (node → left != NULL)
            node = node → left;
        return node;
}

Node * inorderSuccessor (Node * root, Node * x) {
        if (x → right != NULL)
            return findMin (x → right);

        Node * successor = NULL;
        while (root != NULL) {
            if (x → data < root → data) {
```

```
            successor = root;
            root = root → left;
        }else if (x → data > root → data) {
            root = root → right;
        } else {
            break;
        }
    }
    return successor;
}
```

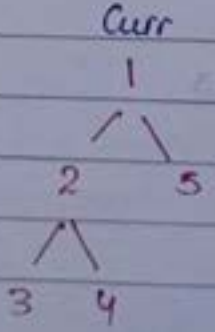
Morris Traversal

using Morris traversal, we can traverse tree without using stack & recursion.
We first create links to Inorder successor and print the data using these links, and finally revert the changes to store original tree.

Algorithm & Example :

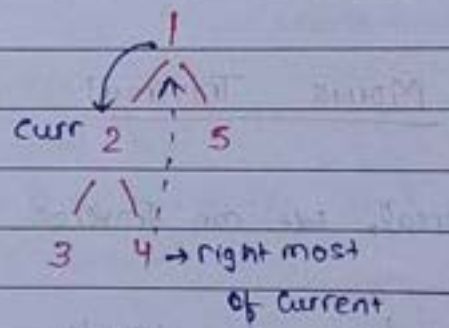


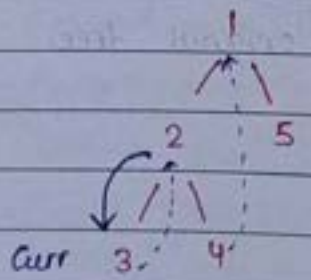
Initialise current node as root



If Current Node does not have any left child
↳ print the Current Node value
↳ then move curr = curr → right

If Current Node have left child,
curr → left Node के right most node को
root से Connect कर दो।
then Current को left में ले आओ।
curr = curr → left

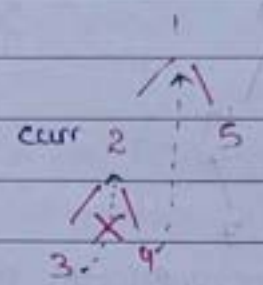




update right child as NULL of node whose right child is node Current and print Current value.
curr = curr → right.

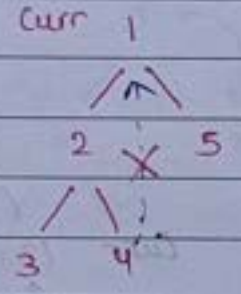
output : 3

Then, we remove the threaded link and current node is updating to the point to the right.

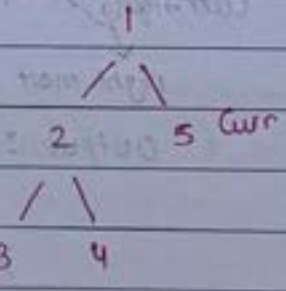


Output : 3, 2

We move to right subtree by pointing current node to right.

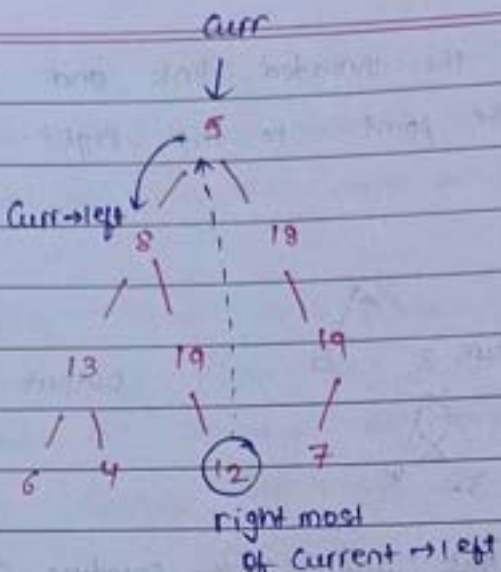


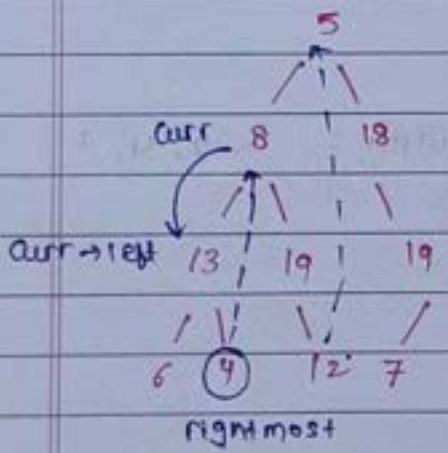
Output : 3, 2, 4, 1

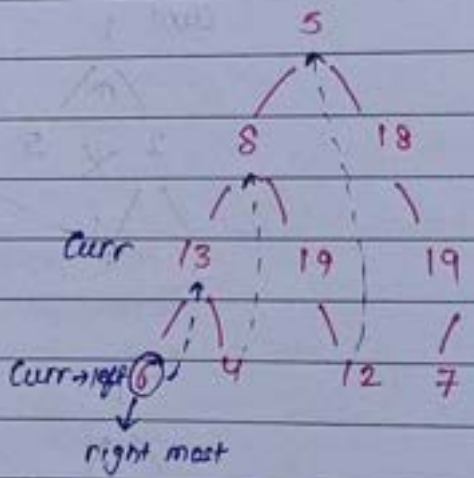


Output : 3, 2, 4, 1, 5

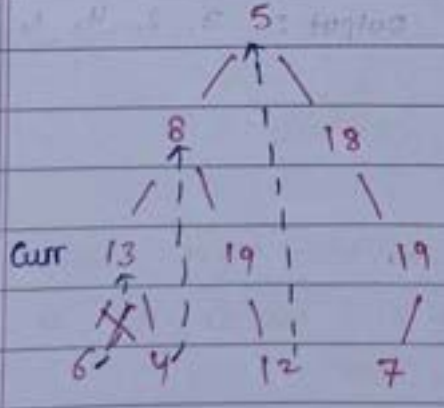
Example :



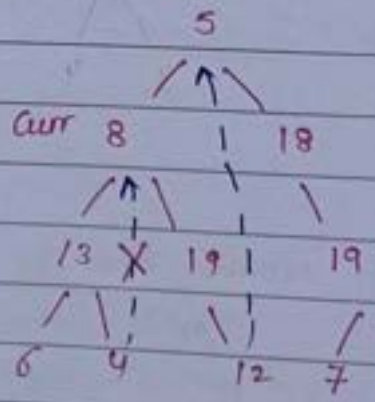




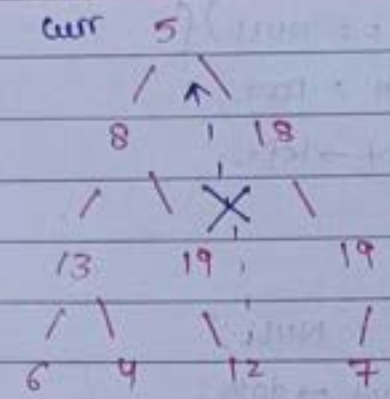
Output : 6



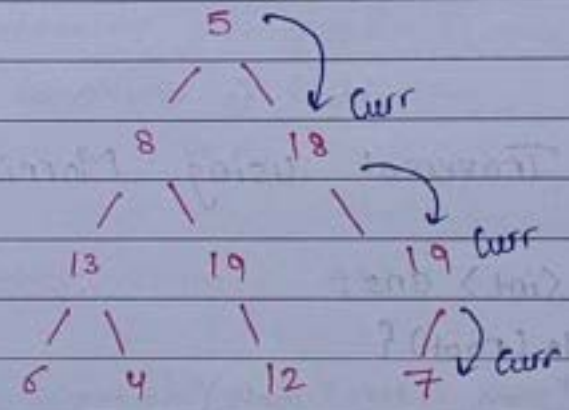
Output : 6 13 4



Output : 6 13 4 8 19 12



Output : 6 13 4 8 19 12 5



Output : 6 13 4 8 19 12 5 18 19 7

Code :

```
While (root) {
    if ( root → left == NULL){
        Cout << root → data;
        root = root → right;
    }
    else {
        Node * curr = root → left;
        While (curr → right) {
            Curr = curr → right;
        }
```

```
        if ( curr → right == NULL ){
            curr → right = root;
            root = root → left;
        }
        else {
            curr → right = NULL;
            cout << root → data;
            root = root → right;
        }
    }
}
```

Inorder Traversal using Morris Traversal :

```
vector <int> ans;
while (root) {

    if ( root → left == NULL ){
        ans.push_back (root → data);
        root = root → right;
    }

    else {
        Node * curr = root → left;

        while ( curr → right && curr → right != root)
            curr = curr → right;

        if (curr → right == NULL) {
            curr → right = root;
            root = root → left;
        }
```

```
        else {
            curr -> right = NULL;
            ans.push_back (root -> data);
            root = root -> right;
        }
    }
}
return ans;
}
```

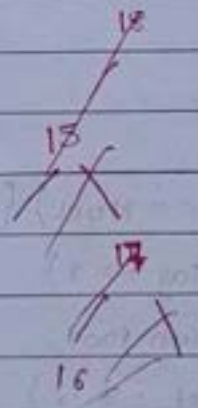
Time Complexity : O(N)

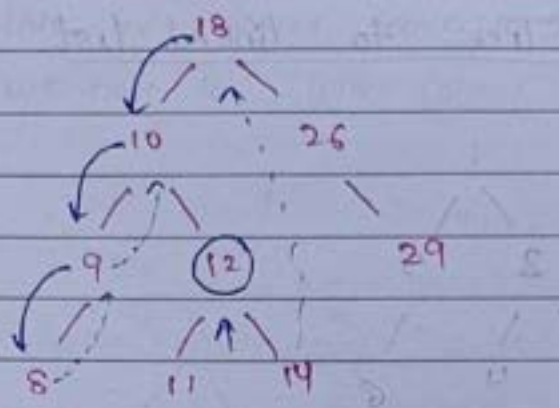
space Complexity : O(1)

* <u>INorder Successor in BST :</u>

```
Node * inorderSuccessor (Node * root, Node * x) {

    bool flag = 0;
    while (root) {
        if ( root -> left == NULL) {
            if ( flag == 1)
                return root;
            if (root == x)
            flag = 1;
            root = root -> right;
        }
        else {
            Node * curr = root -> left;
            while ( curr -> right && curr -> right != root)
                curr = curr -> right;
```

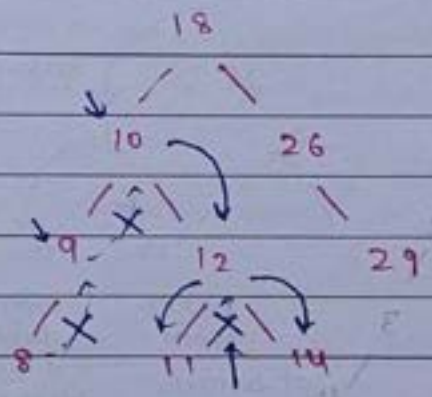
```
        if (!curr → right)
        {
            curr → right = root;
            root = root → left;
        }
        else {
            curr → right = NULL;
            if (flag == 1)
                return root;
            if (root == x)
                flag = 1;
            flag = root → right;
        }
    }
}
```





x = 12

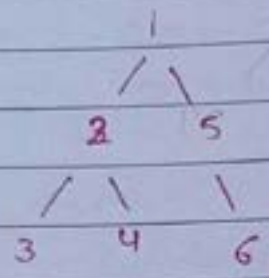
flag = 0



flag = 1

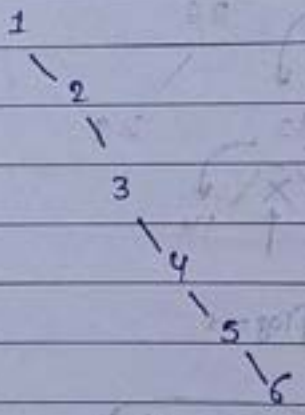
ans = 14

Flatten Binary Tree to linked list



in preorder form

output :



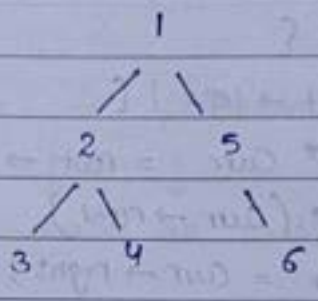
Approach 1 :

1) Stored in Queue
   ↳ Tree in preorder
2) Then print with only right link Node.

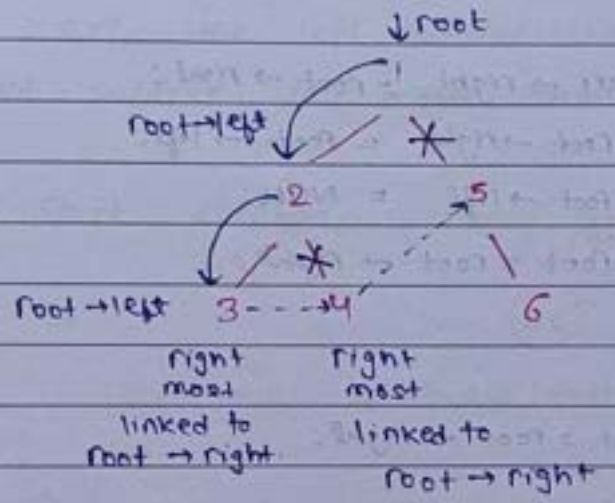
Queue | 1 | 2 | 3 | 4 | 5 | 6 |

We solve this by Morris traversal :

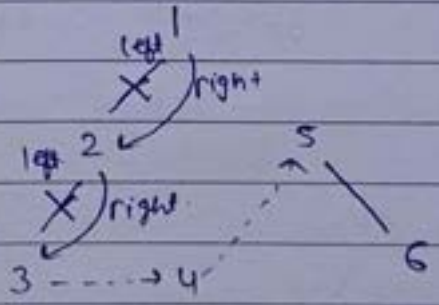
We have to link form.



preorder : N L R



left tod kar
right ort ot
आओ ।



Code :

```
Void flatten (Node * root) {

    While (root) {
        if (root → left) {
            Node * Curr = root → left;
            While (Curr → right)
                Curr = Curr → right;

            Curr → right = root → right;
            root → right = root → left;
            root → left = NULL;
            root = root → right;
        }
        else {
            root = root → right;
        }
    }
}
```

Day - 47

(BST - Hard problems)

problem 1 : preorder to postorder

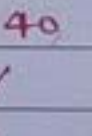
pre : 40 30 35 80 100

In BST, the left subtree is smaller than root and right subtree is bigger than root

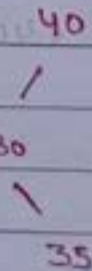
Start creation,

40 → root → (40)

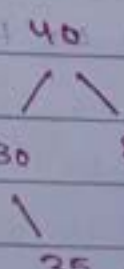
30 → 30 < 40, left side



35 → 35 > 30, 35 < 40, right of 35

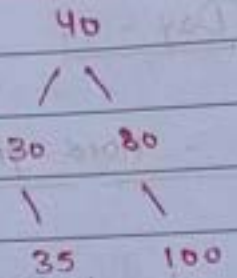


80 → 80 > 40, right of 40



classmate
Date
Page

100 > 80



We approach this problem as range.

MIN = INT_MIN

MAX = INT_MAX

The first Number from preorder:

from start,

If any Node wont to make BST.

MIN < Element < MAX

40 → (40)

Divided in 2 parts.

(40)

Min < Element < 40 40 < Element < MAX

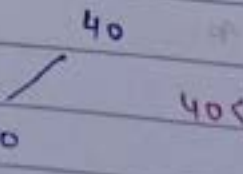
30 →

check on left

30 > MIN ✓

30 < 40 ✓

push on left



40 < Element < MAX

MIN < Element < 30 30 < Element < 40

35 →

check on left

35 > MIN ✓

35 < 30 X Fails

return, check on right

35 > 30 ✓

35 < 40 ✓

Push on right of 30

40

40 < Element < MAX

30

MIN < Element < 30

35

30 < Element < 35 35 < Element < 40

80 →

check left of 35 : 80 > 30 ✓ 80 < 35 X

check right of 35 : 80 > 35 ✓ 80 < 40 X

return check right of 80 : 80 > 40 ✓ 80 < MAX ✓

40

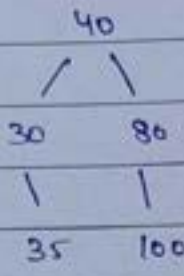
30 80

35

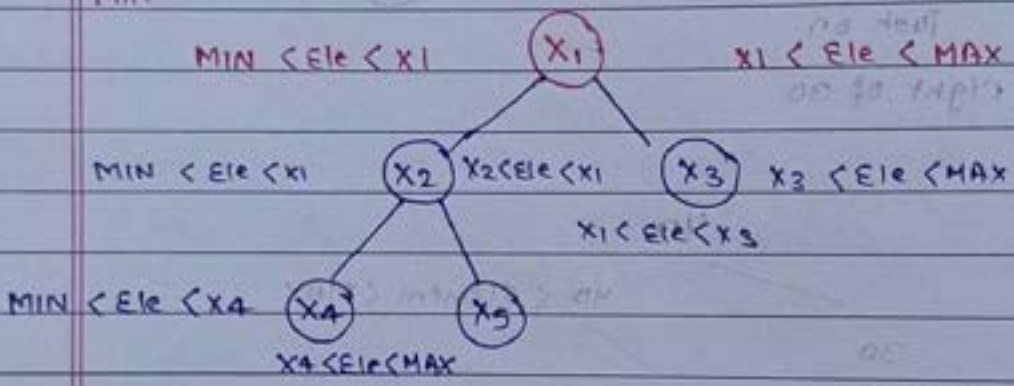
100 :

Check left of 80 : 100 > 40 ✓ 80 100 < 80 X

Check right of 80 : 100 > 80 ✓ 100 < MAX ✓



Required ans.



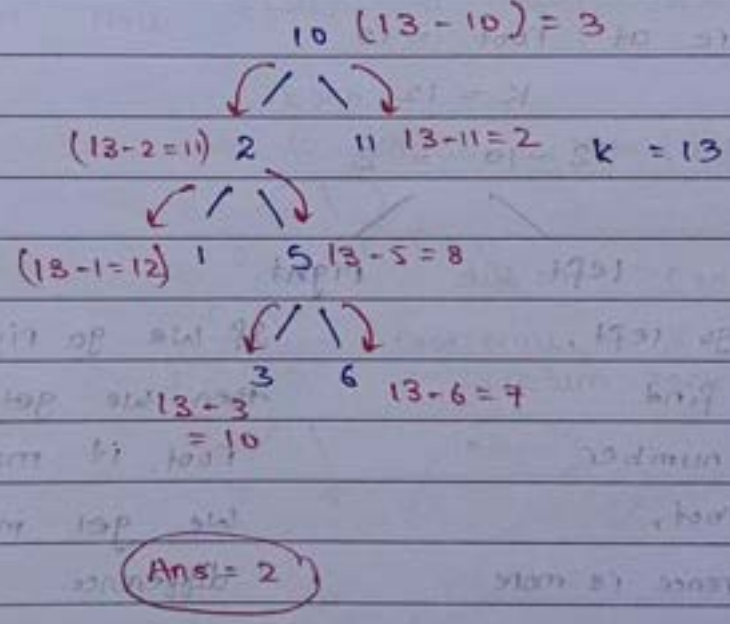
Code :

```
Node* post_order (int pre[], int size) {

  int min = INT_MIN, max = INT_MAX, index = 0;
  return find(pre, min, max, index, size);
}

Node * find (int pre[], int min, int max, int &index, int size ){
      if (index >= size || pre[index] > max || pre[index] < min)
            return NULL;

      Node * root = new Node;
        root → data = pre [index];
        index++;
        if (index == size)
            return root;
```

```
root->left = find(pre, min, root->data, index, size);

root->right = find(pre, root->data, max, index, size);

return root;
}
```

problem 2: find the Closest Element in BST :

Example:

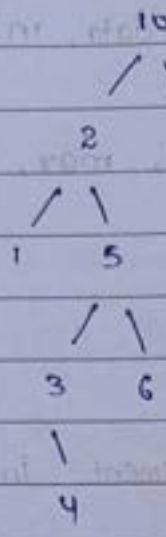


is it necessary to check all Elements

No.

Then, How to solve this

lets understand with Example.



We are at root = 10

K = 12

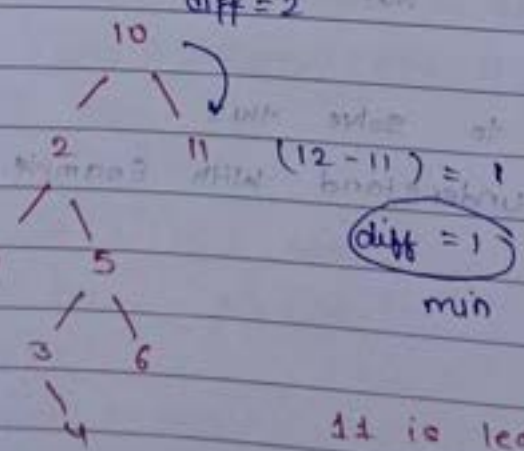
12 - 10 = 2

left right

If we go left, then we find smaller number than root, then difference is more

If we go right then we get bigger than root, it may possible we get more close difference

So, we move right



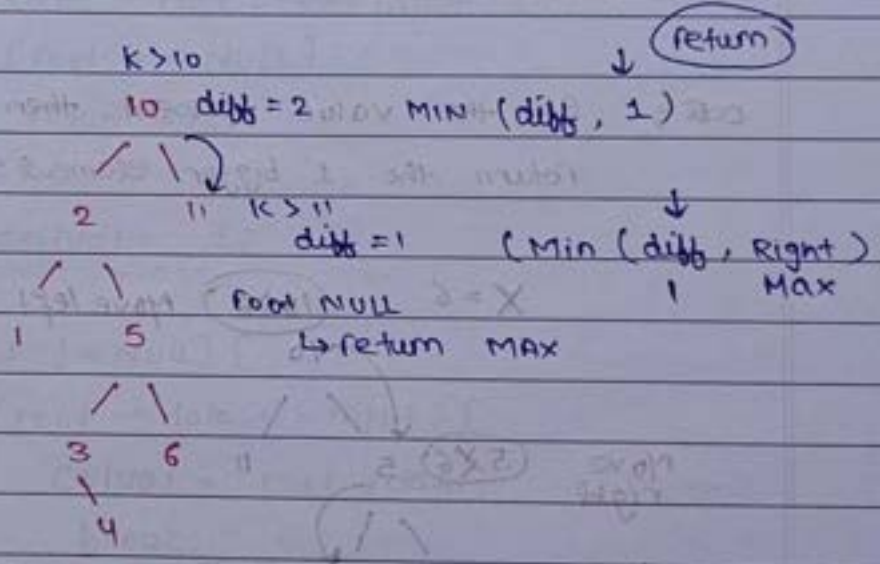
11 is leaf then return.

case 1: K = root → data
return 0;

case 2: K > root → data
↳ go to right side

case 3: K < root → data
↳ go to left side.

अगर root NULL होगा तो क्या करेंगे।

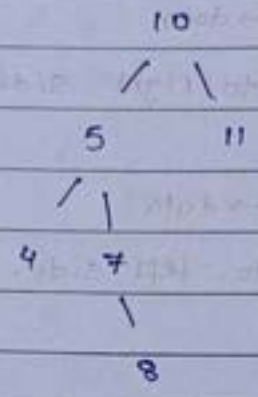


code:

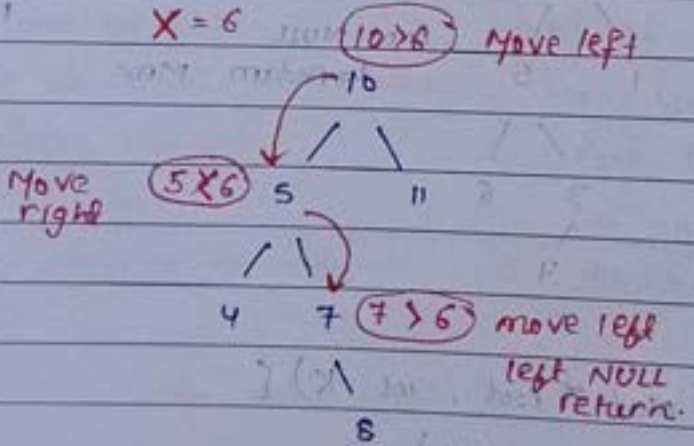
```
int minDiff (Node * root, int K) {
    if (root == NULL)
        return INT_MAX;

    if (K == root → data)
        return 0;

    if (K > root → data)
    return min (K - root → data, minDiff (root → right, K));

    if (K < root → data)
    return min (root → data - K, minDiff (root → left, K));
}
```

Problem 3: Ceil in BST :

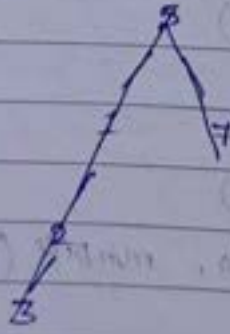
Example 1



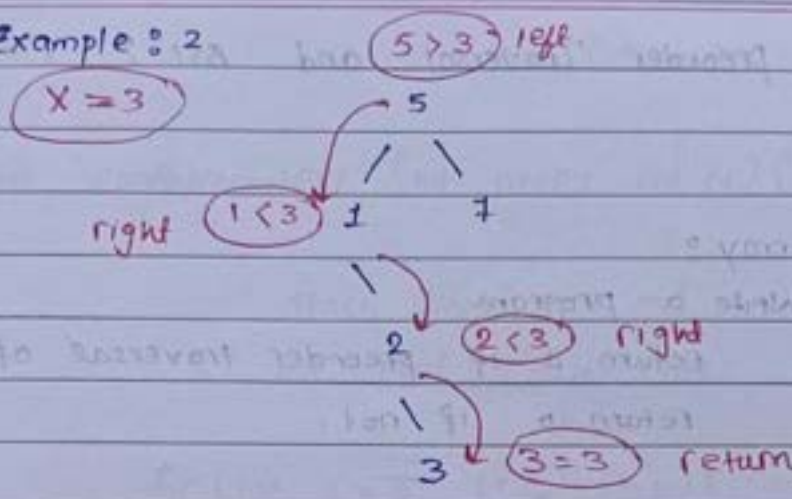
ceil : If the value present then return, otherwise return the 1 bigger element :



Example 2 :



Example : 2



Code :

```
int findCeil (Node * root, int input) {
        if (root == NULL)
            return -1;

        int ceilVal = -1;

        while (root != NULL) {
            if (root → data == input) {
                ceilVal = root → data;
                break;
            } else if (root → data < input) {
                root = root → right;
            } else {
                ceilVal = root → data;
                root = root → left;
            }
        }
        return ceilVal;
}
```

problem 4: preorder Traversal and BST:

Given an array:

↳ Write a program

return 1 if preorder traversal of BST.

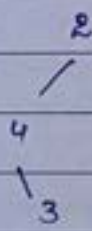
return 0 if not.

Example 1:

N = 3

arr = {2, 4, 3}

output = 1

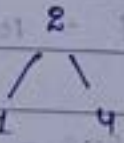


valid preorder.

Example: 2

arr = {2, 4, 1}

NLR



preorder: [2, 1, 4]

arr is not preorder

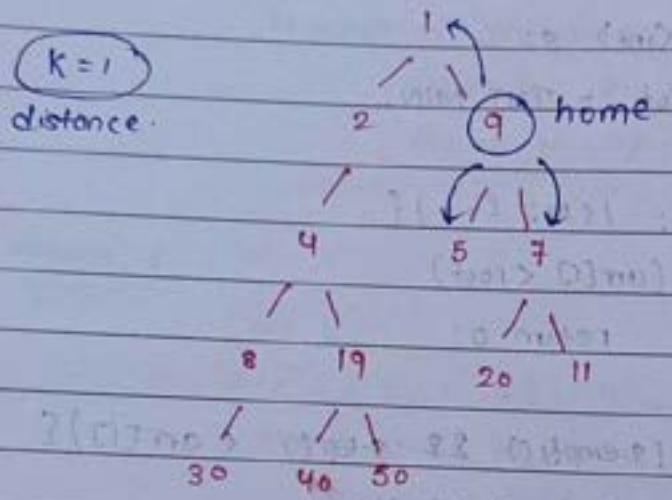
↳ return 0.

code :

```
int canRepresentBST (int arr[], int N){

    stack <int> s;
    int root = INT_MIN;

    for (int i = 0; i<N; i++){
        if (arr[i] < root)
            return 0;

        while ( !s.empty() && s.top() < arr[i]){
            root = s.top();
            s.pop();
        }

        s.push (arr[i]);
    }
    return 1;
}
```

problem 5 : Bheem wants Ladoos :

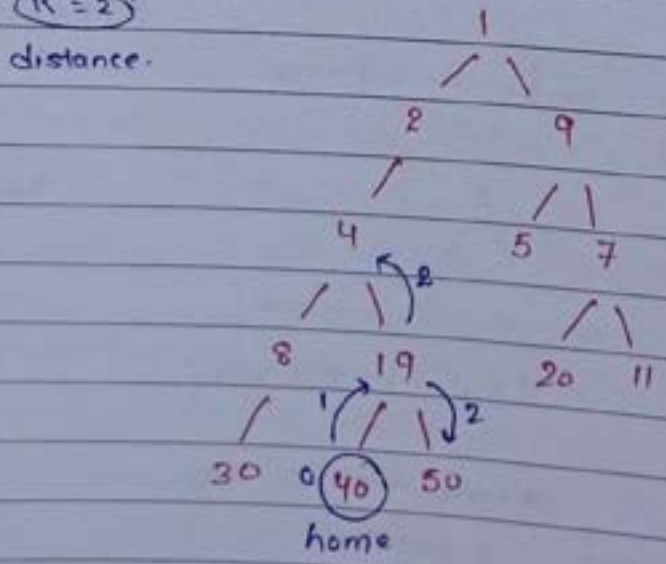
Example 1



9 + 1 + 5 + 7 = (22)

↓ output.

Example 2



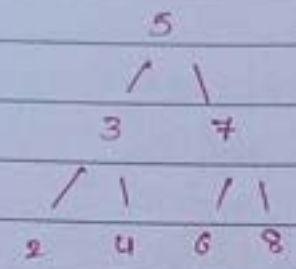
40 + 19 + 4 + 50

= (113)

↳ output.

problem 6:

MINimum Absolute Difference in BST



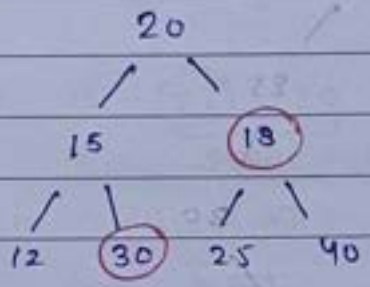
Code:

```
Void inorder (Node* root, int &prev, int &munDiff){
        if (root == NULL)
            return;

    inorder (root→left , prev, munDiff);

        if ( prev != -1)
            munDiff = min (munDiff, root→data - prev);
    prev = root→data;

    inorder ( root→right , prev, munDiff);
}

int absolute_diff (Node * root){
        int prev = -1;
        int munDiff = INT_MAX;
        inorder (root, prev, munDiff);
    return munDiff;
}
```

Problem 7: Fixing two swapped Nodes of a BST :

Example

BST hai lekin 2 node Galat Jagah pe hai wo swapped ❤ Rakha ❤ ! usi ko find karna ❤ !



Inorder : 12 15 30 20 25 18 40

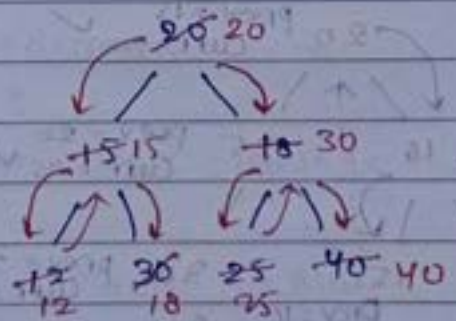
inc inc dec inc dec

Swap(30, 18)

In inorder it always increase.

After swapping :

12 15 18 20 25 30 40.

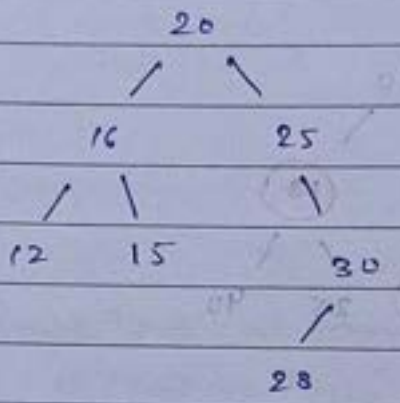


Dusre jaao Put karte jaao !

→ Now We check the edge cases :

अगर सिर्फ एक बार Array में decrease हो रहा है तो :-

Example:



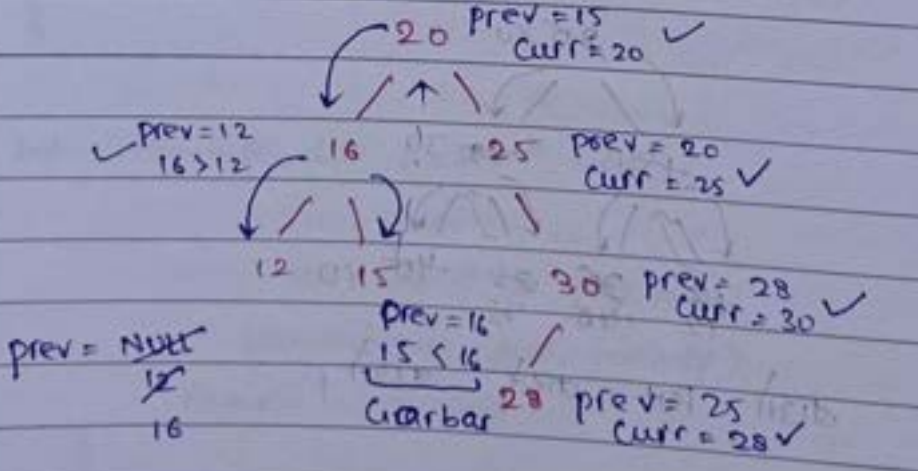
Inorder : 12 16 15 20 25 28 30

inc dec inc inc inc inc

If dec at only one place

then swap the both element.

Without creating array, how we solve :

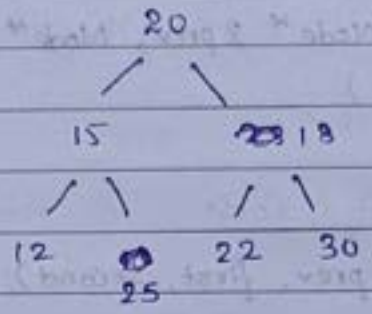


first = prev

second = root

then swap this.

If two decrease :



Inorder : 12 15 25 20 22 18 30

dec dec

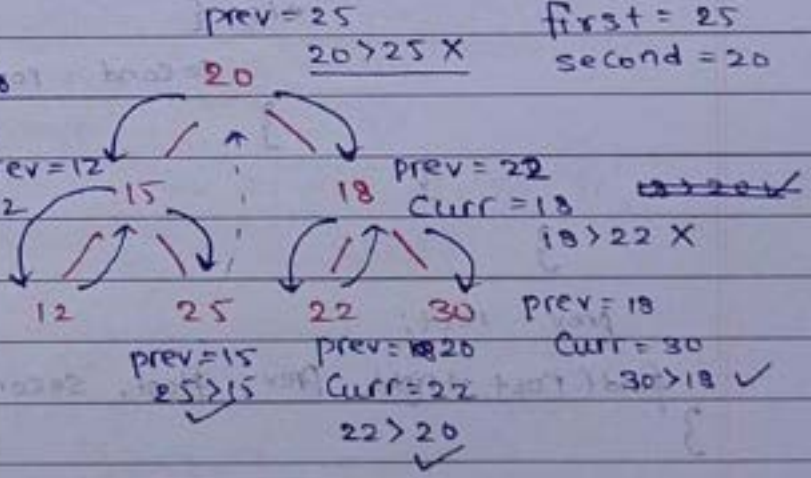
first का first second का second.

Without array :

First = ~~NULL~~ 25

second = ~~NULL~~ ~~20~~ 18

prev = ~~NULL~~ ~~12~~ ~~15~~ 25



When we found decrease 2nd time :

We only change end pointer. (second).

1. prev, first, second → NULL
2. Inorder traversal

prev exist

→ value decrease

↳ 1st time if(first = NULL & second = NULL)

↳ 2nd time if(first != NULL).

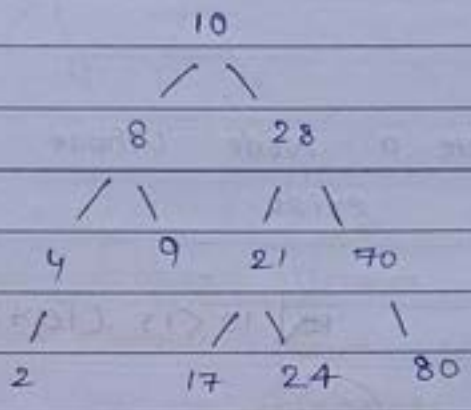
Code :

```
Void find (Node* root, Node* &prev, Node* &first, Node* &second){
        if (!root)
         return;

    find (root->left, prev, first, second);

    if (prev) {
        if (prev->data>root->data){
            if (!first || !second){
                first = prev;
                second = root;
            }
            else{
                second = root;
            }
        }
    }
    prev = root;
 find(root->right, prev, first, second);
}

Struct Node * correctBST (struct Node * root){
    Node* prev = NULL, *first = NULL, *second = NULL;
     find (root, prev, first, second);
     int data = first->data;
     first->data = second->data;
     second->data = data;
     return root;
}
```

Day - 48

(Balanced Binary Search Tree)

problem : Delete a Node from BST :

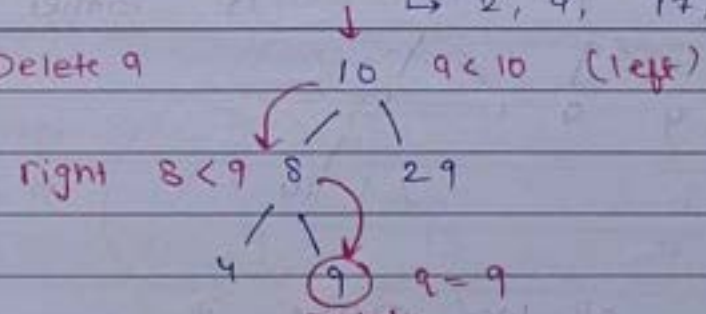


case 1 : leaf Node to be delete

↳ directly delected

↳ 2, 9, 17, 24, 80

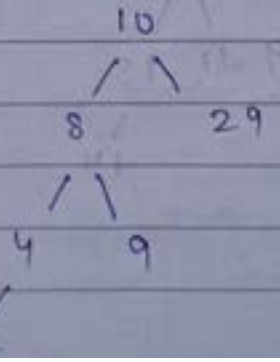
Delete 9

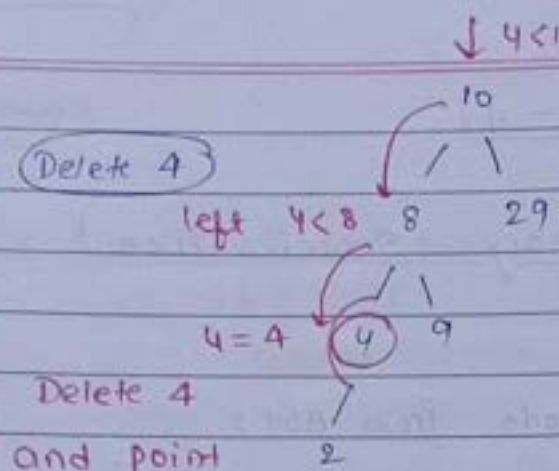


Case 11 : Delete a Node whose 1 child exist

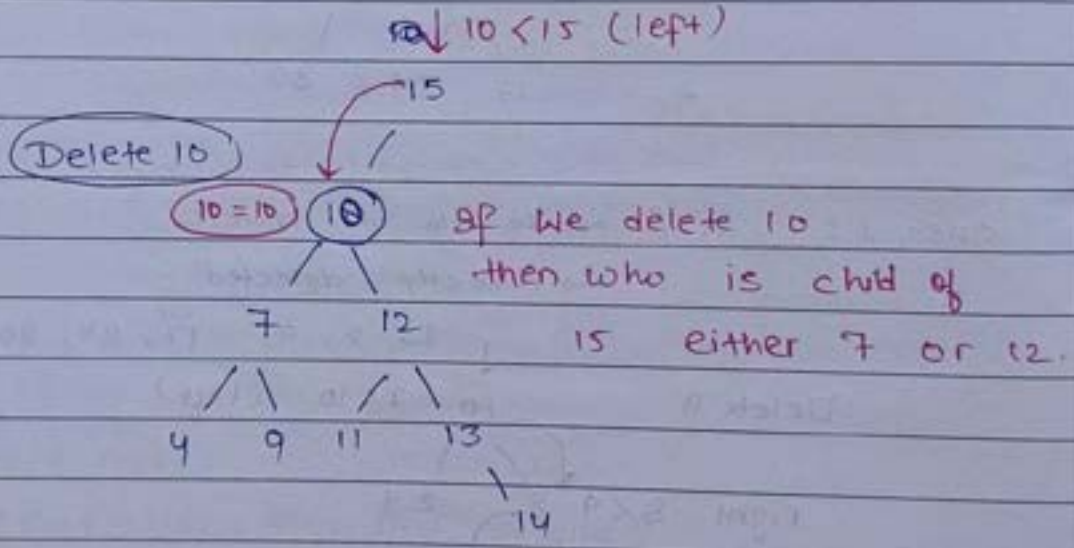
↳ 4, 70

Delete 4



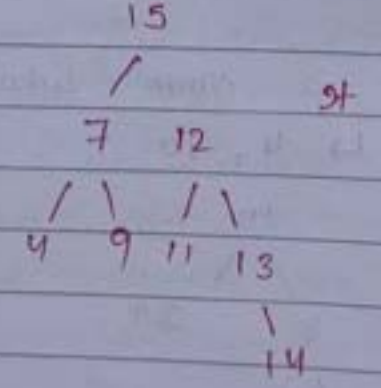


Case III : Delete a Node whose both child exist



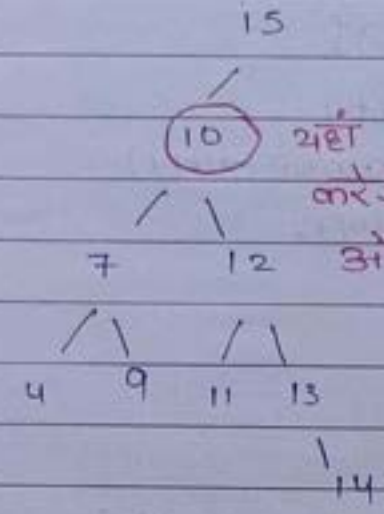
If we delete 10 then who is child of 15 either 7 or 12.

If we make 7



It can't be possible.

So, we do:

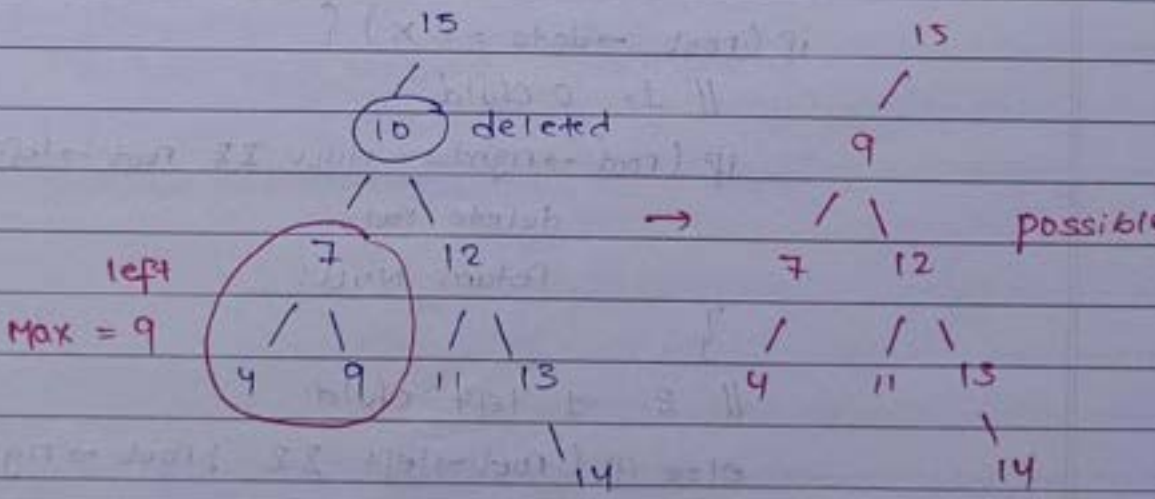


यहाँ पर हम एक ऐसा value put करेंगे जिसका सारा left छोटा हो और सारा right बड़ा हो।

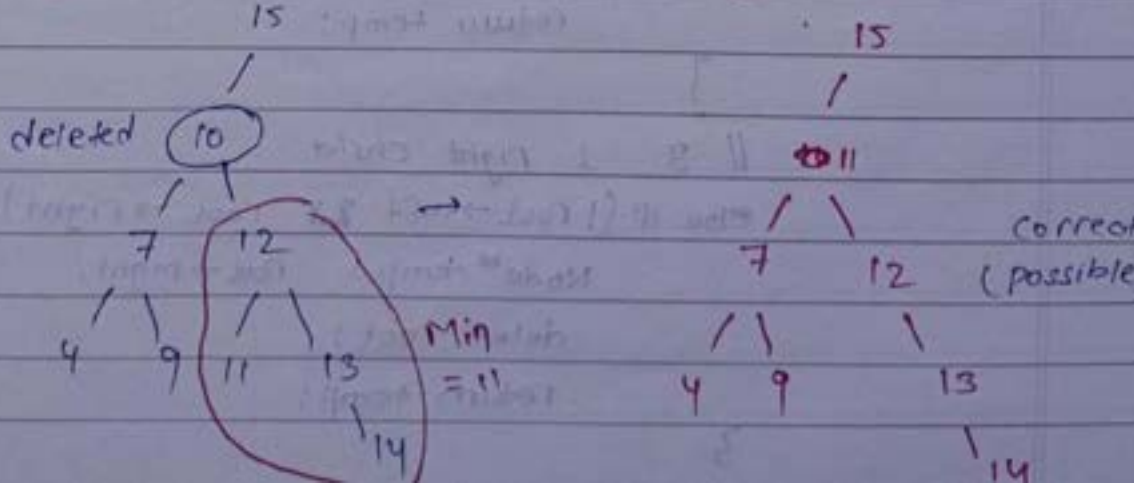
Here, two cases arises

↳ (1) Choose from left subtree

↳ Greatest from left subtree



(2) Choose Minimum from right subtree.

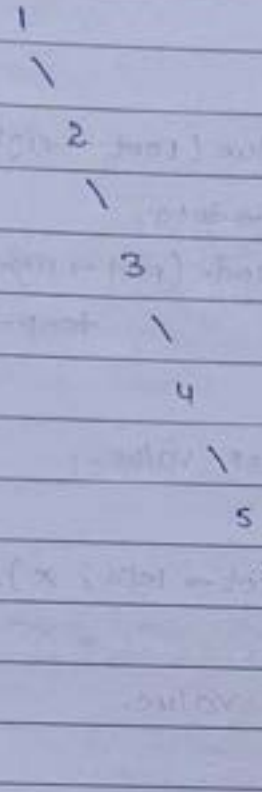


Code :

```
Node* minvalue (Node* root){
    Node* Current = root;
    While (Current && Current->left)
     Current = Current->left;

    return Current;
}

Node *deleteNode (Node *root, int x){
        if (root == NULL)
          return NULL;

  //1. X value is equal to root
        if (root->data == x){
          // 1. 0 child
          if (root->right == NULL && root->left == NULL){
              delete root;
              return NULL;
          }
          // 2. 1 left child
          else if (root->left && !root->right){
              Node* temp = root->left;
              delete root;
              return temp;
          }
          // 3. 1 right child
          Else if (!root->left && root->right){
              Node* temp = root->right;
              delete root;
              return temp;
          }
```

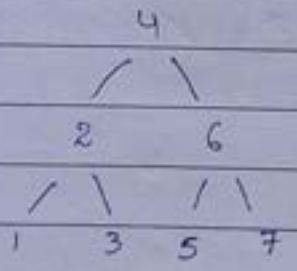
```
// 4. 2 child
    else {
        Node * temp = minValue (root -> right);
        root -> data = temp -> data;
        root -> right = deleteNode (root -> right,
                                    temp -> data);
    }
}
// 2. x value is less than root value.
    else if ( x < root -> data ) {
        root -> left = deleteNode (root -> left, x);
    }
// 3. x value is greater than root value.
    else {
        root -> right = deleteNode (root -> right, x);
    }

    return root;
}
```

classmate
Date
Page



On this if we want to search 7, then it takes O(N) time.

But, after arranging:



On this we can search is O(log N)

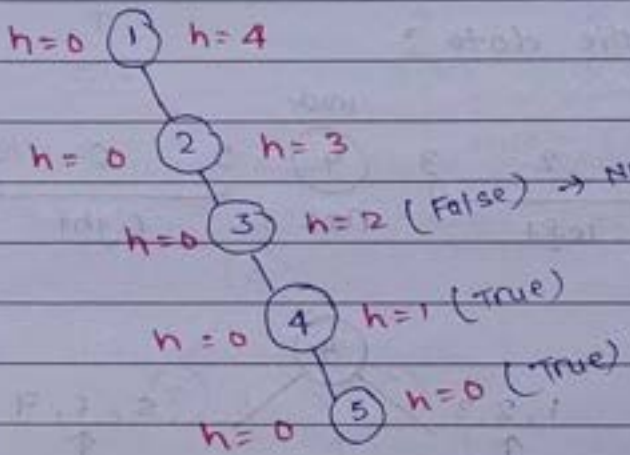
This tree is cared Balanced Binary search tree.

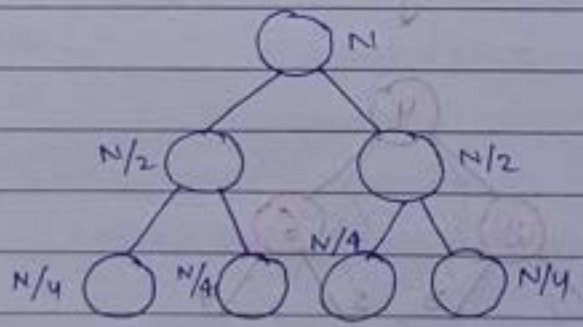
Balanced Binary Search tree

A binary tree is balanced if the height of tree is $O(\log N)$ where n is the number of nodes.

$$-1 \leq (\text{left height} - \text{right height}) \leq 1$$

Then, tree is ~~ppp~~ perfect Balanced tree.





h=2 3 h=3 (2-3) = True

(True) h=0 1 h=1 h=15 h=2 (1-2) = -1 (True)

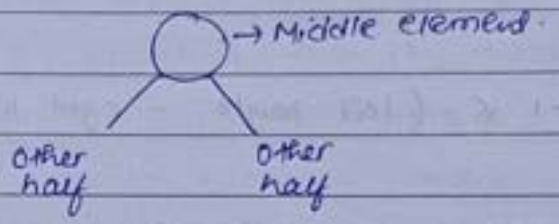
h=0 2 h=0 0 True 4 0 h=0 7 h=1 (0-1) = (-1) True)

(True) 6 h=0 h=0 (True)

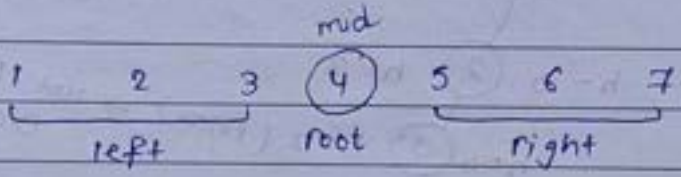
Balanced Binary search tree.

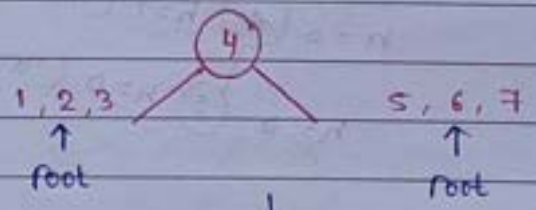
How we Create a Balanced Binary Search tree :

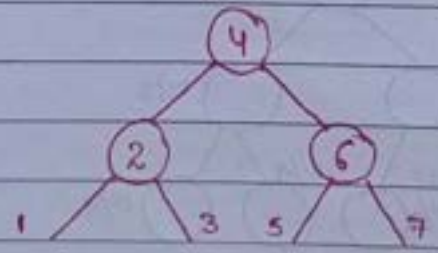
7 4 1 6 5 2 3

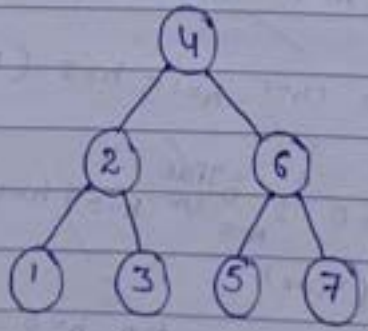


Sort the data :









Takes O(logn) in searching an element.

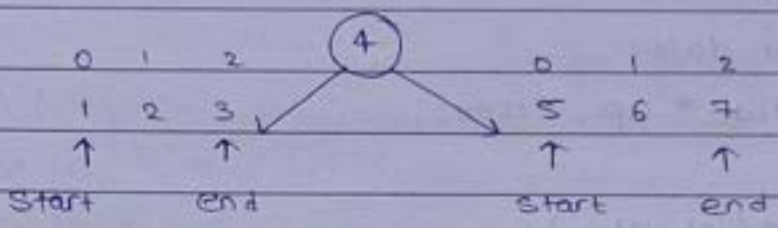
creation : O(nlogn)

classmate
Date
Page

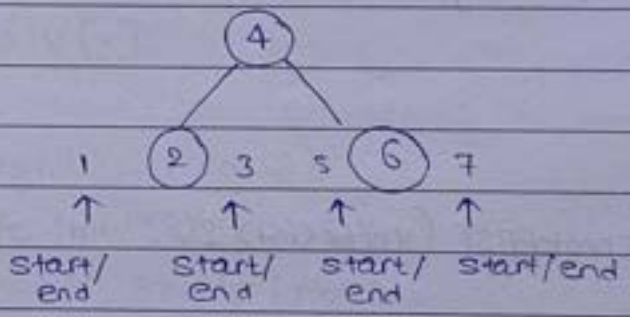
0	1	2	3	4	5	6
1	2	3	4	5	6	7
↑						↑
start						end

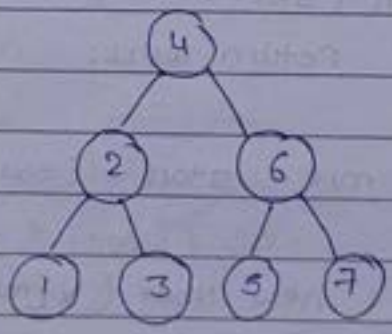
mid = start + (end - start) / 2;

mid = 4



mid = 2 mid = 6





start > end → Stoppage Condition.

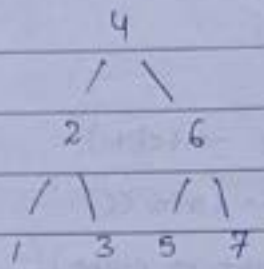
return

Code :

```
#include <iostream>
using namespace std;

~~int main()~~

~~vector~~

Class Node {
    public :
    int data;
    Node * left, *right;

  Node (int value) {
      data = value;
      left = right = NULL;
    }
}

Node* CreateBBST (vector<int> &V, int start, int end) {

        if (start > end)
            return NULL;

        int mid = start + (end - start) / 2;

    Node* root = new Node (V[mid]);
    root->left = CreateBBST ( V, start, mid - 1);
    root->right = CreateBBST (V, mid + 1, end);
    return root;
}
```

```
Void inorder ( Node * root){
        if (!root)
            return;

        inorder (root -> left);
        Cout << root -> data << " ";
        inorder (root -> right);
}

int main () {
    int n;
    cin >> n;
    vector <int> v(n);
   for( int i = 0; i<n; i++)
        cin >> v[i];

   Sort (v.begin(), v.end());
   Node * root = createBBST (v, 0, n-1);
    inorder(root); Preorder(root);
   return 0;
}


Void preorder ( Node * root ){
        if (!root)
            NULL;
        Cout << root -> data << " ";
        preorder( root -> left );
        preorder ( root -> right);
    }
```

preorder : 4 2 1 3 6 5 7



Inorder check about sorted

preorder can check balance of tree

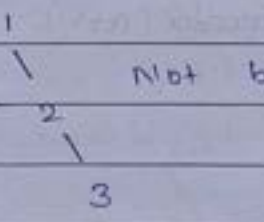
→ In the same problem we have two condition

① you don't have to sort

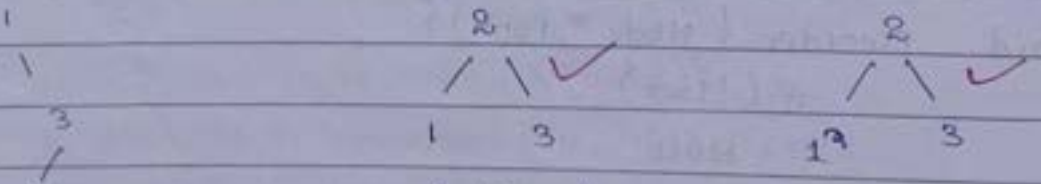
② create at Runtime

Then how to create a Balanced Binary search tree

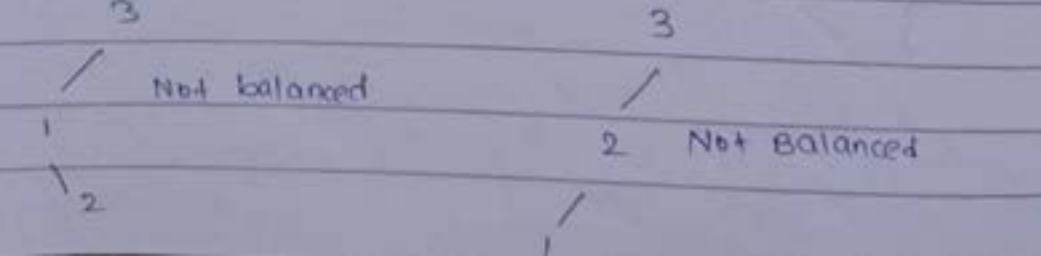
1, 2, 3



1 3 2 2 1 3 2 3 1



3 1 2 3 2 1



How we balance, where we have problem ?

① left side ↓root

1
\R
2

R-R unbalance
↳ then Move Root to left.

⇒ 1 को तोड़ कर नीचे ले आओ

2
/ \
1 3

② ↓root
3 right side fall

2
L/
2

L-L unbalance
↳ the Move root to right

Break. 3 ⇒ put down.

2
/ \
1 3

③ ↓root
3
L/
1
R\
2

⇒

2
/ \
1 3

L-R unbalance
↳ Move root→left to left.
↳ Move root to right.

Break in two step

1 on left

Step 1: 3 / 2 (Break in right) → 1

Step 2: 2 / \ 1 3

④ ↓root
1
\R
3
/L
2

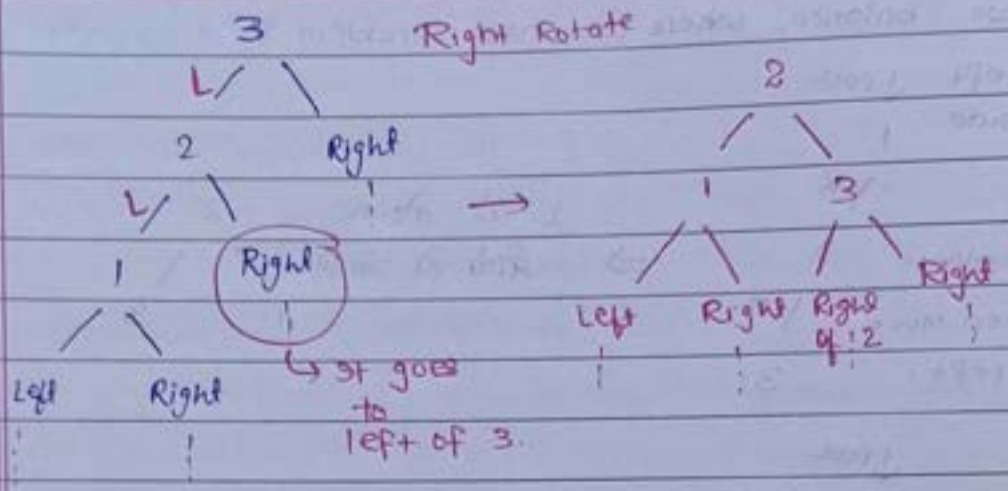
R-L unbalance
↳ Move root ⇒ right to right
↳ Move root to left.

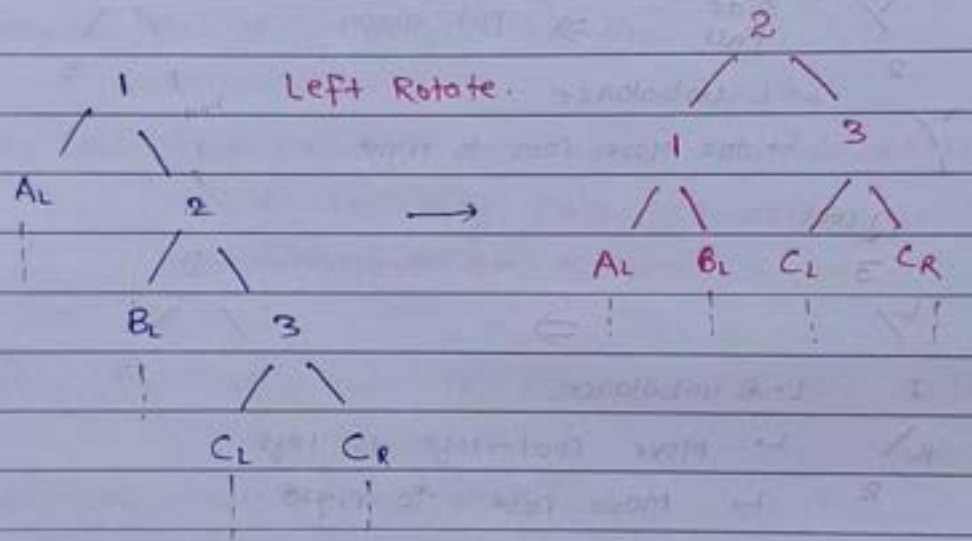
2
/ \
1 3

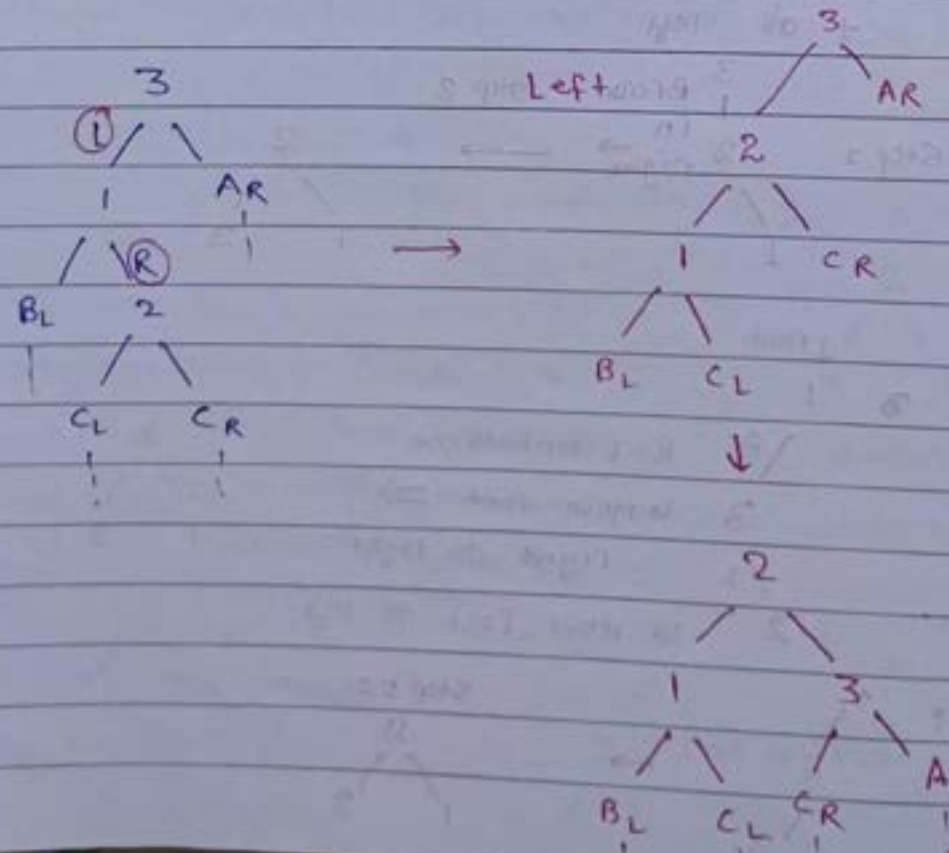
Step 1: 1 \ 2 → 2 / 1

Step 2: 2 / \ 1 3

classmate
Date
Page







classmate
Date
Page

Two function

1) Rotate left
2) Rotate Right

```
Rotate_left (Node * root) {
        Node * temp = root → right;
        root → right = temp → left;
        temp → left = root;
        return temp;
    }
```

```
Rotate_right (Node * root) {
        Node * temp = root → left;
        root → left = temp → right;
        temp → right = root;
        return temp;
    }
```

→ Node creation :

1) Data
2) left
3) right
4) height (for calculation of height on each step).

→ find Element right position ⇒ add
→ Return → check Balance
→ If Not Balance, Balance it.

Code :

```
#include <iostream>
using namespace std;

class Node {
    public:
        int data, height;
        Node * left, * right;

    Node (int value) {
        data = value;
        height = 1;
        left = right = NULL;
    }
};

int getHeight (Node * root) {
    if (!root)
        return 0;

    return root -> height;
}

Node * rotateRight (Node * root) {
    Node * temp = root -> left;
    root -> left = temp -> right;
    temp -> right = root;
    UpdateHeight (root);
    UpdateHeight (temp);
    return temp;
}
```

```
Node * rotateLeft (Node * root) {
        Node* temp = root->right;
        root->right = temp->left;
        temp->left = root;
        updateHeight(root);
        updateHeight(temp);
    return temp;
}

int main() {
    Node* root = NULL;
```

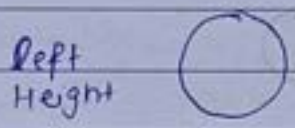
```
Node * insertBST (Node * root, int value) {
        if (!root)
            return new Node(value);

        if (value < root->data)
            root->left = insertBST (root->left, value);

        else
            root->right = insertBST (root->right, value);

    return root;
}

Node * Balance (Node * root) {

        // Complete Yourself

    }

int main() {
        Node * root = NULL;
        int value;
```

```
While(1){
    cin >> value;
    if (value != -1)
        root = insertBST (root , value);
    else
        break;
    };
}

Node * Balance (Node * root){
    if (!root)
        return NULL;
    inorder (root);
    preorder (root);
return 0;
}
```

Lecture - 49

AVL and Heap

Balanced binary search tree have some properties of AVL tree.



abs (Left Height - Right Height) > 1

↳ then unbalance

Then How to know which unbalance is used Rotation :

- → Left - left
- → Left - Right
- → Right - Right
- → Right - left

This is done by :

We can initialise a variable Balance.

Balance = left Height - Right Height.

if Balance > 1

↳ then unbalance is in left side

if Balance < -1

↳ then unbalance is in right side.

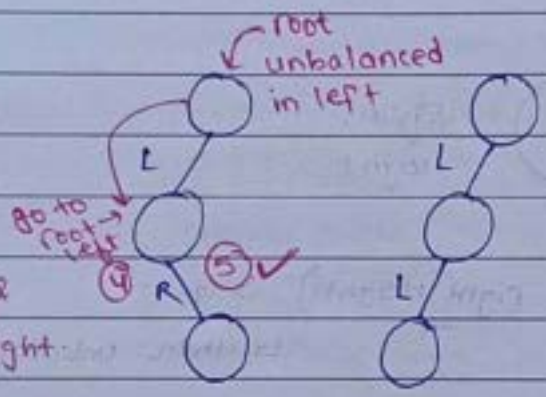
Till now, we know that either left or right balancing happen.

If left then

↳ left - left

↳ left - right

how we know about this.



ask left & right height

Second place find करने के लिए जिसके संतुलन होता उसके ही unbalance होता।

* 17

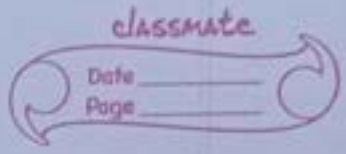
Height = 0 (17) Height = 0

* 15

Height = 1 (17) Height = 0

(15) Balanced.

* 12



Ht = 2 17 Ht = 0

L/ 2 - 0 = 2 (unbalanced)

1 15 0 > 1 (left)

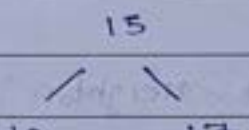
L/

12

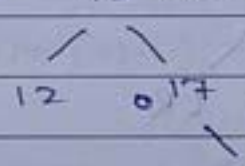
L - L unbalance

↓

So balance this



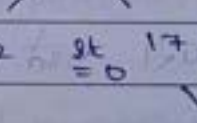
→ 18 1 15 Ht = 3 2 (1 - 2) = -1 (Balanced)



12 0 17 , Ht = 2 (0 - 1) balanced.

18 Ht = 0 + 1

→ 20 lt = 1 15 Ht = 4 Rt = 3 (1 - 3) = -2 (Right unbalance)



12 lt = 0 17 Ht = 3 Rt = 2 (0 - 2) = -2 unbalanced.

R

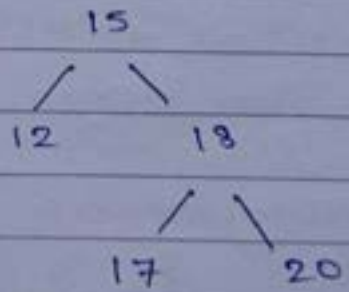
R - R unbalance. lt = 0 18 Ht = 2 Rt = 1 (0 - 1) = -1 balanced

R

20 Ht = 1

lt = 0 Rt = 0

Balance



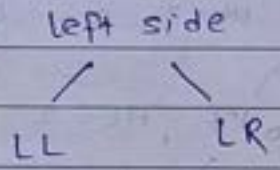
classmate
Date
Page

(1) First Enter Element in our tree.

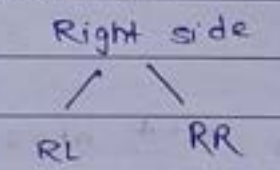
(2) Either element goes in left or right till reach to NULL.

(3) Till it finds NULL, create itself & return.

(4) check the Balancing:
- first update the Height
- Balance = Left - Right
- Balance > 1



- Balance < 1



Balancing इसलोग वापस आते time check करते है।

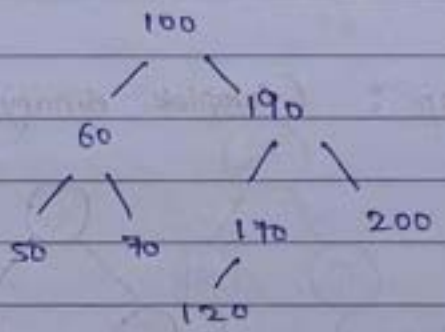
```
Void updateHeight (Node * root){
    int leftHeight = getHeight (root → left);
    int rightHeight = getHeight (root → right);

    root → height = 1 + max (leftHeight, rightHeight);

}

Node* Balance (Node * root) {
        if (!root)
            return NULL;

        updateHeight (root);

    int balance = getHeight (root → left) - getHeight (root → right);
        // Balance > 1 : Left unbalance
        if (balance > 1) {
            // Left Left
            if (getHeight (root → left → left) >= getHeight (
                                    root → left → right) {
                root = rotateRight (root);
            }
            // Left right
            else {
                root → left = rotateLeft (root → left);
                root = rotateRight (root);
            }
        else
        // Balance < -1 : Right unbalance.
        else if (balance < -1) {

            if (getHeight (root → right → right) >= getHeight (root → right
                                    → left)) {
```

```
            root = rotateleft(root);
        }

        else{
            root->right = rotateright(root->right);
            root = rotateright(root);
        }
    }
    return root;
}

Void inorder( Node * root ){
        if(!root)
            return;

        morder(root->left);
        Cout<<root->data<<" ";
        inorder(root->right);
    }

Void preorder( Node * root ){
        if(!root)
            return;

        Cout<< root->data <<" ";
        preorder( root->left);
        preorder(root->right);
    }
```

* Balanced Binary search tree : O(log N)

Insertion :

Insert : 120



Insertion : O(logn)

Deletion : O(logn)

* Binary search tree :

Insertion : O(N)

Deletion : O(N)

* Binary tree :

Insertion : O(N)

Deletion : O(N)

Example :

Flight :

4 Airplanes have to land

Which lands first

	1	2	3	4
Emergency	6	10	8	2

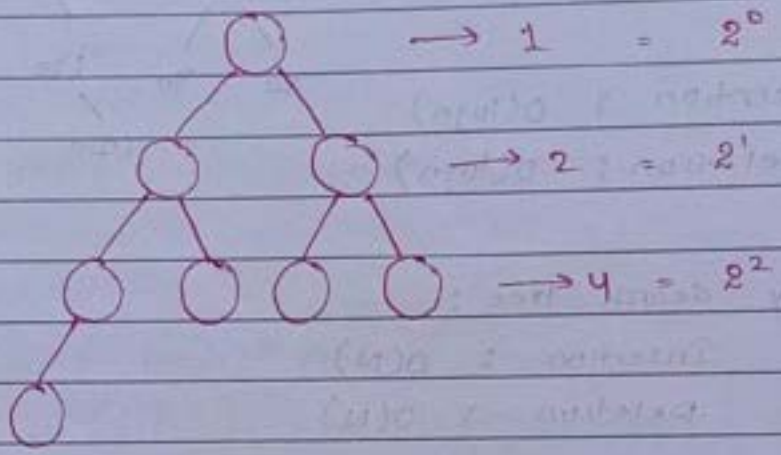
We give permission of landing by the help of Emergency factor.

Airplanes landed : 2 3 1 4

This is done by the help of priority basis.

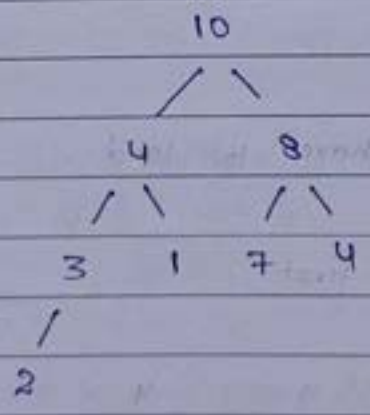
Those have high priority, they have land first.

* Heap : (Complete Binary Tree)

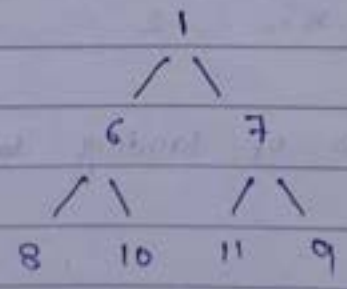


last level सिर्फ property hold Nahi karega.

Max Heap :



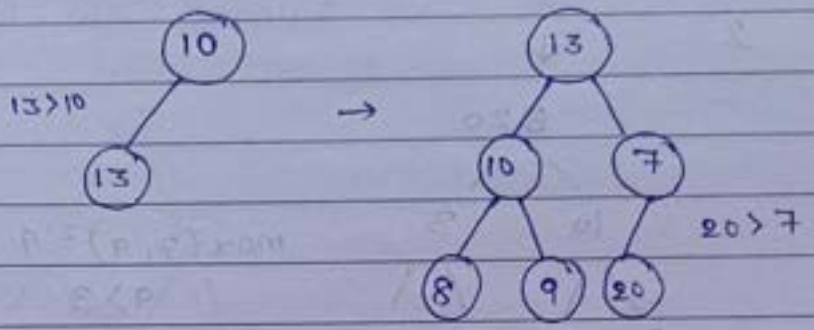
Min Heap :

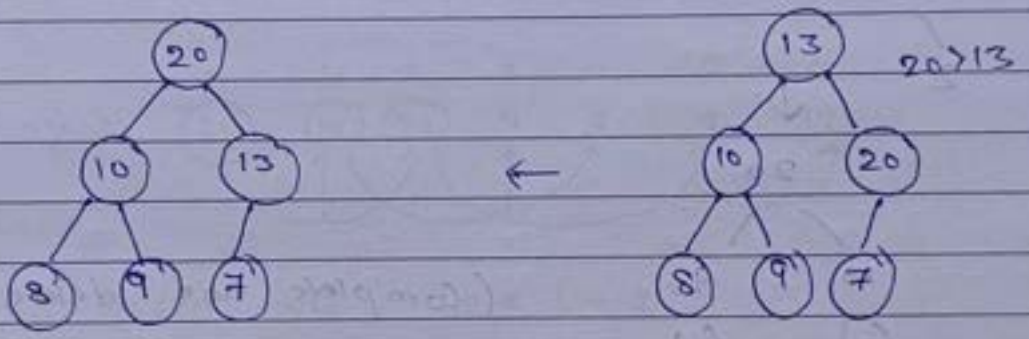


Max - Heap

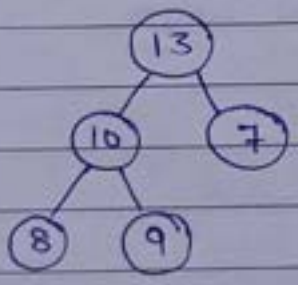
1	2	3	4	5	6
10	13	7	8	9	20

priority :



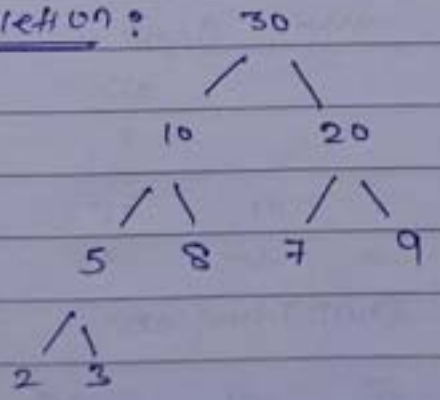


Delete 20



~~→ Not true because this is Not Heap~~

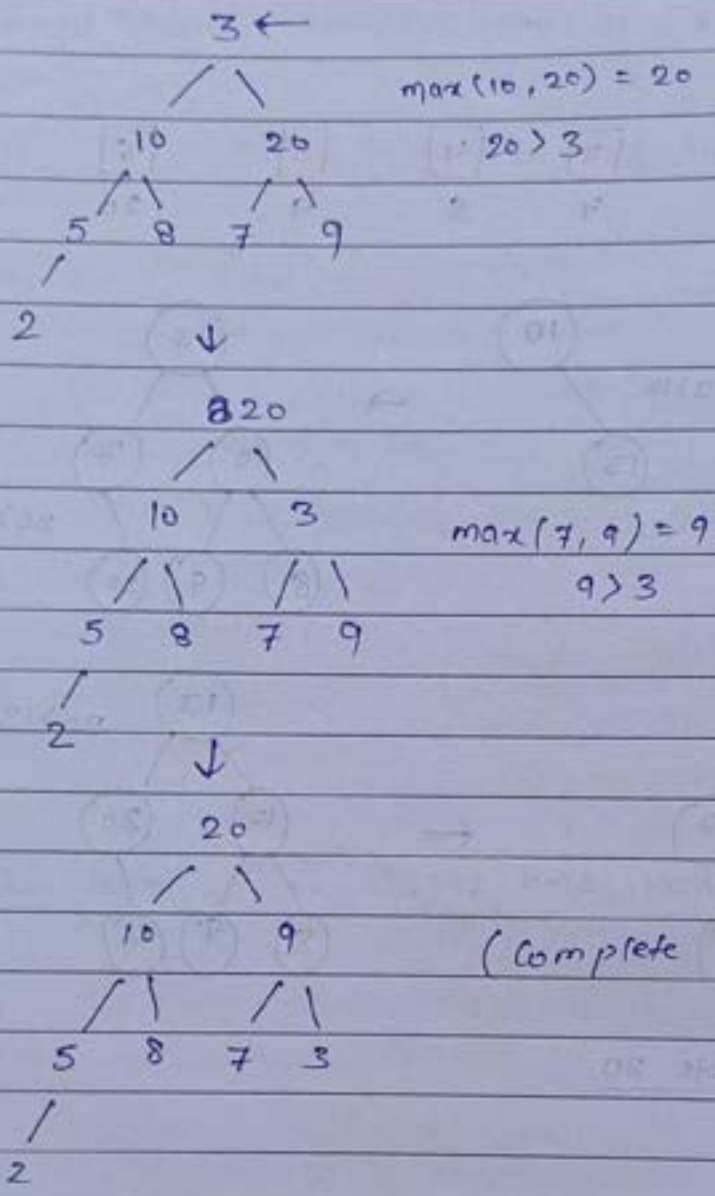
* Deletion :



Delete 30

How to do.

Delete 30 and put the last element of 3.

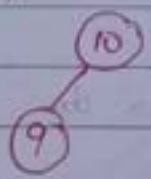


(Complete and deleted).

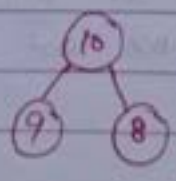
* **Insertion :**

Insert 10

Insert 9



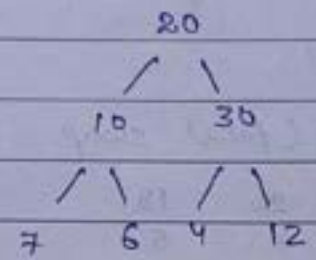
Insert 8

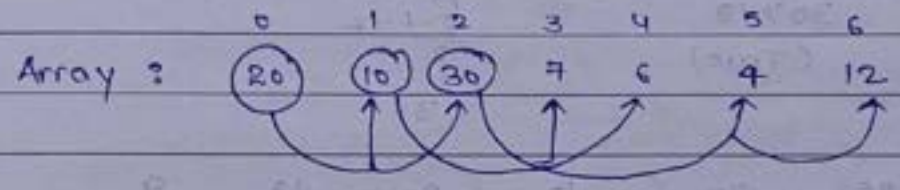


for Insertion we use level order traversal.

→ Better Approach:

Convert it into array:





Left child = 2 * i + 1;

Right child = 2 * i + 2;

parent Node : $\frac{(i-1)}{2}$;

parent Node >= Child Node.

[30 10 70 20 25 18 9 11]

Create Heap

→ 30
0

→ 30 10
0 1

30 >=10 (True)

→ 30 10 70
0 1 2
↓
parent = 30

30 > 70 (false)
swap

→ 70 10 30 20

0 1 2 ↓

parent

||

10

10 > 20 (false) swap

→ 70 20 30 10 25

0 1 2 3 4

20 < 70 (True)

20 > 25 (false) swap

→ 70 25 30 10 20 18

0 1 2 3 4 5

30 > 18 (True)

parent

$\frac{5-1}{2} = 2$

||

30

→ 70 25 30 10 20 18 9

0 1 2 3 4 5 6

30 > 9 (True) $\frac{6-1}{2} = 2 = 30$

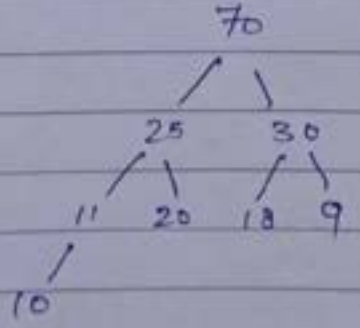
→ 70 25 30 10 20 18 9 11

0 1 2 3 4 5 6 7

10 > 11 (false) $\frac{7-1}{2} = 3 = (10)$

swap

→ 70 25 30 11 20 18 9 10



Time Complexity : $O(N \lg N)$

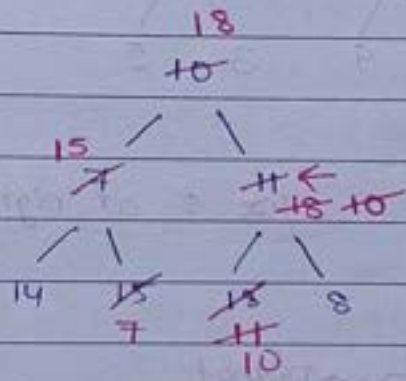
Lecture - 50

(Heap)

Heap sort : (sorting)

Max Heap
Min Heap

Heapify : Correct position
O(N) create.



Max Heap

for sorting :

1) Take Element from top.

2) put the last element in place of top.

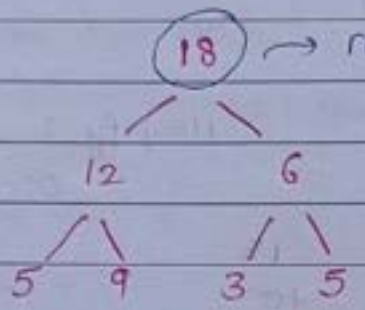
3) top element find its correct position and fix it.

4) Apply these 3 steps repeatedly till Heap is empty.

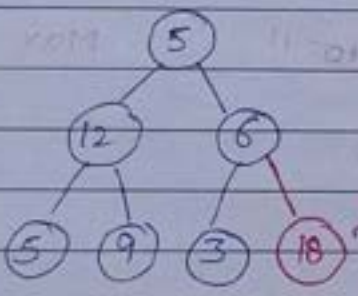
0	1	2	3	4	5	6
18	12	6	5	9	3	5

Sort these

Make a tree :



→ remove and put here bottom right most element



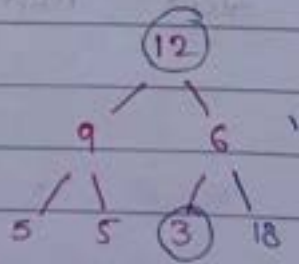
fix 5 at right place.

→ Didnot Consider this

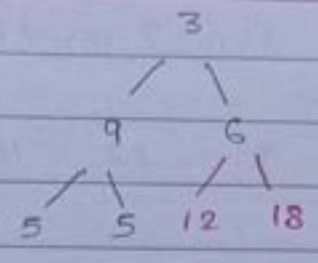
5 | 12 | 6 | 5 | 9 | 3 | 18

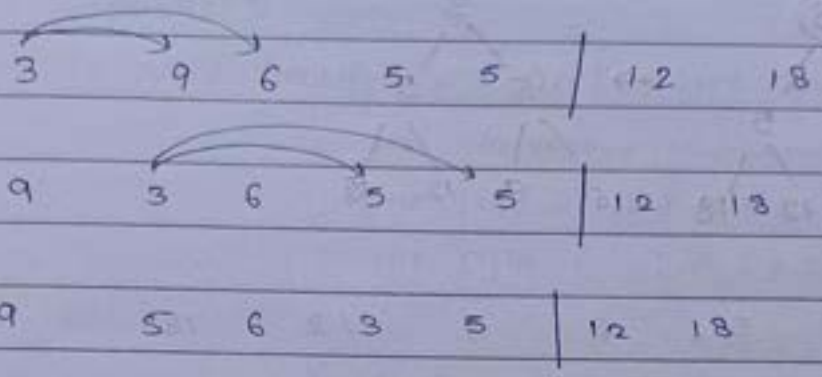
12 | 5 | 6 | 5 | 9 | 3 | 18

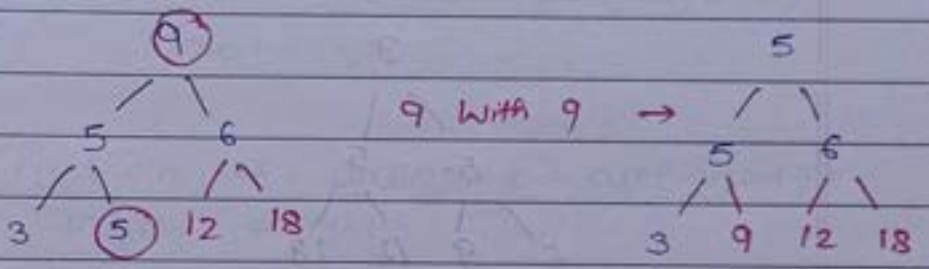
12 | 9 | 6 | 5 | 5 | 3 | 18

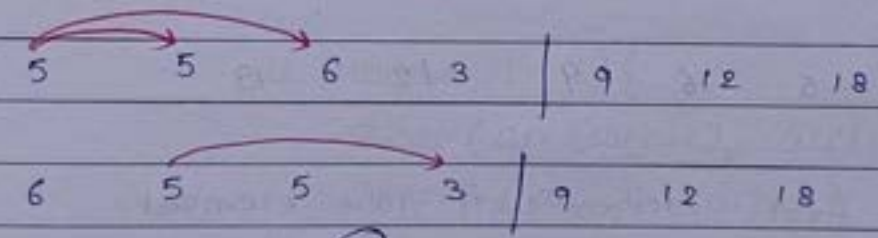


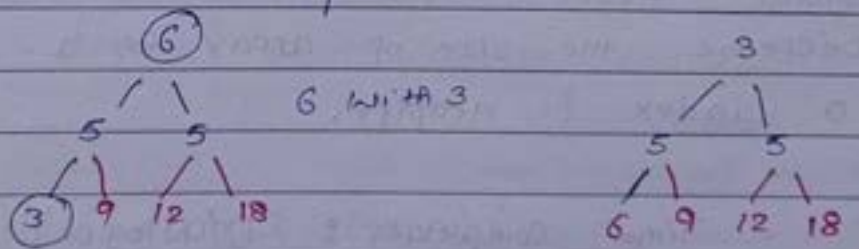
12 with 3

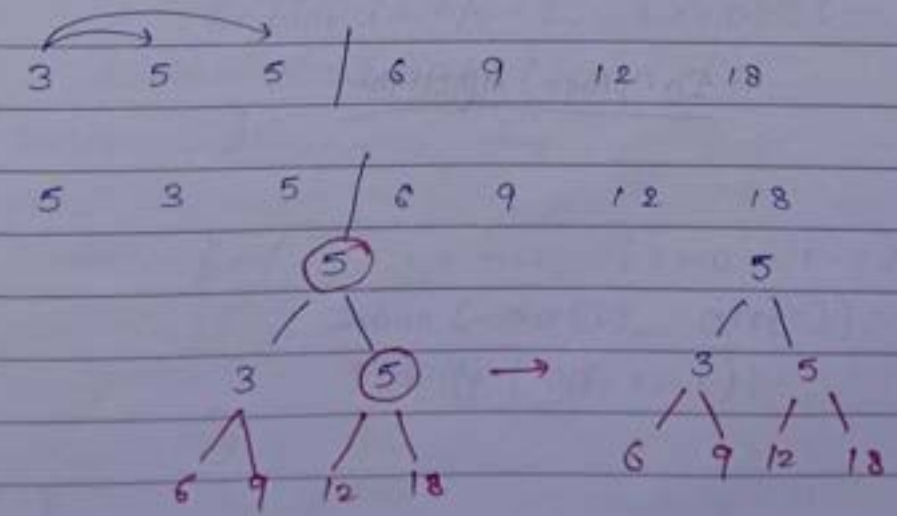

3
9 6
5 5 12 18


3 9 6 5 5 | 12 18
9 3 6 5 5 | 12 18
9 5 6 3 5 | 12 18


9
5 6
3 5 12 18
9 with 9 →
5
5 6
3 9 12 18


5 5 6 3 | 9 12 18
6 5 5 3 | 9 12 18

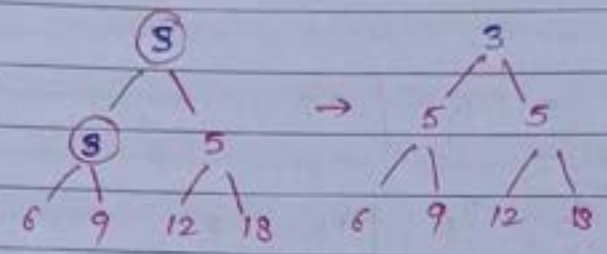

6
5 5
3 9 12 18
6 with 3
3
5 5
6 9 12 18


3 5 5 | 6 9 12 18
5 3 5 | 6 9 12 18
5
3 5
6 9 12 18
→
5
3 5
6 9 12 18

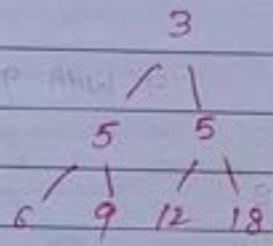
classmate
Date
Page

5 3 | 5 6 9 12 18

5 3 | 5 6 9 12 18



3 | 5 5 6 9 12 18



3 5 5 6 9 12 18

1. Swap first element with last element.
2. Decrease the size of array by 1.
3. 0 index ! Heapify.

Time Complexity : $O(N \log N)$

In-place Algorithm

Problem - 1 : Heap Sort

Code :

```
class solution {
    public :
        void heapify (int arr[], int n, int i) {
            int largest = i;
            int left = 2 * i + 1;
            int right = 2 * i + 2;

            if (left < n && arr[left] > arr[largest])
                largest = left;

            if (right < n && arr[right] > arr[largest])
                largest = right;

            if (largest != i){
                swap (arr[largest], arr[i]);
                heapify (arr, n, largest);
            }
    }

void heapsort (int arr[], int n){
        for (int i = n/2 - 1; i >= 0; i--){
            heapify (arr, n, i);
        }

        for (int i = n-1; i >= 0; i--){
            swap ( arr[i], arr[0]);
            heapify (arr, i, 0);
        }
}
```

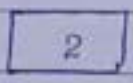
~~Heap~~

Heap (Code using STL)

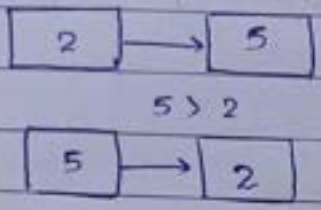
```
#include <iostream>
#include <queue>        // STL for Heap
Using namespace std;

int main(){
    // max heap
    priority_queue<int> p;
    p.push(4);
    p.push(7);
    p.push(2);
    p.push(6);

    cout<<p.top();
    p.pop();
    cout<<p.top();

    cout<<p.empty();    // 0 because heap is not empty
    // min Heap
return 0;
    priority_queue<int, vector<int>, greater<int>> p;
}
```

Heap creation by linked list :

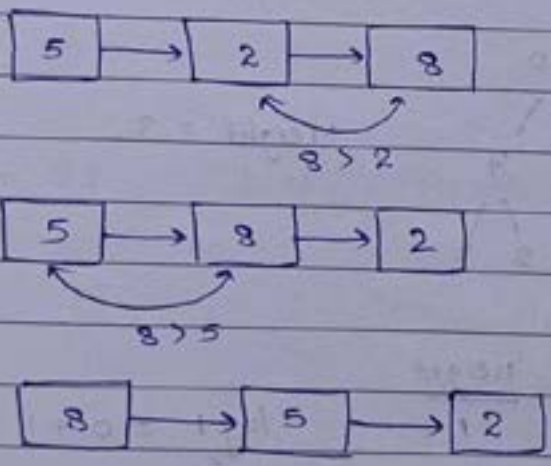
→ 2



→ 5



→ 8



run Heap :

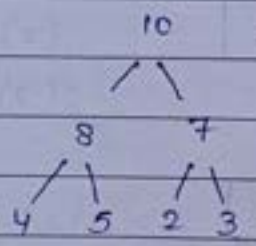
```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

int main (){
    // run Heap
    priority_queue <int, vector <int>, greater <int> > p;
        p.push (9);
        p.push (6);
        p.push(3);
        p.push(2);
```

```
cout<<p.top();
p.pop();
cout<<p.top();
return 0;
}
```

* problem 2 : ~~Height of a tree~~ (Height of Heap)

Tree in form of array

arr : 10 8 7 4 5 2 3



Height = 3

Element	Height	
1	1	$\log_2 1 = 0 + 1 = 1$
2	2	$\log_2 2 = 1 + 1 = 2$
3	2	$\log_2 3 = 1 + 1 = 2$
(4 - 7)	3	$\log_2 4 = 2 + 1 = 3$
(8 - 15)	4	$\log_2 9 = 3 + 1 = 4$

find log with base 2 and then add 1

$\log_2 15$: $\frac{15}{2} \rightarrow \frac{7}{2} \rightarrow \frac{3}{2} \rightarrow 1$

$3 + 1 = 4$

classmate
Date
Page

An question, they ask 0 based height, so the answer is only $\log_2 n$.

Code:

```
int heapHeight ( int N, int arr[]) {
        int count = 0;
        while ( N != 0) {
            N = N/2;
            count++;
        }
        return count -1;
}
```

* problem 3: profix Maximization (interview bit)

Given : Array A
↓
represents seats in each row of stadium.

sell Ticket to B people.

Seat Cost = No. of Vacant seats in row it belong to

Maximize the profit.

0 1
A = {2, 3}
B = 3

Row 0 _ _ cost = 2
Row 1 B_1 _ _ (cost = 3) Max

(B = 3) → people.
people --
B = 2

2 Row 0 _ _ price = 2

1 Row 1 _ _ price = 2 Max

B = ~~2~~ 1

profit = 3 + 2 = 5

Row 0 = 2 → Max

Row 1 = 1

B = ~~1~~ 0

profit = 3 + 2 + 2 = 7.

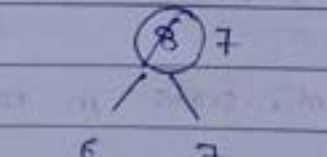
* Example 2 :

Array : [6, (8), 4, 7]

B = ~~5~~ 4 ↑ find Max

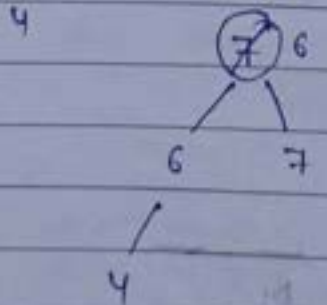
profit = 8

Solve by Max-heap :



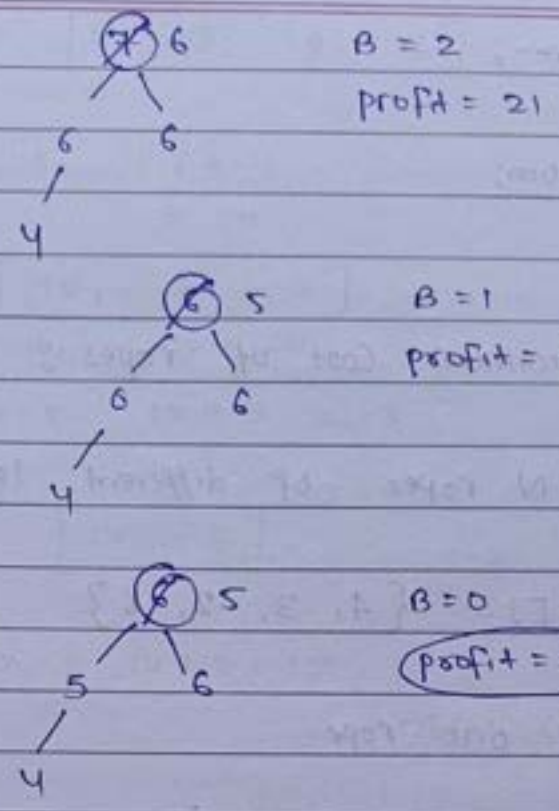
B = ~~5~~ 4

profit = ~~6~~ 8



B = ~~4~~ 3

profit = 15



Time Complexity : O(NlogN).

Code :

```
int Solution::Solve(vector<int> &A, int B){
    if(B==0)
        return 0;

    int sum=0;
    priority_queue<int> p;
    for(int i=0; i<A.size(); i++){
        if(A[i])
        p.push(A[i]);
    }

    While(B!=0 && p.size()!=0){
        sum+= p.top();
        if(p.top()>1)
        p.push(p.top()-1);
        p.pop();
```

```
        B--;
    }
    return sum;
}
```

* Problem 4 : Minimum Cost of ropes :

Given : N ropes of different length

arr[] = {4, 3, 2, 6}

Connect in one rope.

Cost of Connect = sum of both ropes

find minimum Cost of Connect all ropes.

Example

[4, 3, 2, 6]

Choose Minimum :

3 & 2

5

[4, 5, 6]

Minimum :

4 + 5 = 9

[9, 6]

Minimum

9 + 6 = 15.

Total Cost = 15

5
9
15
29 (Min)

[4, 3, 2, 6]

Max : 6 + 4
= 10

[10, 3, 2]

Max : 10 + 3 = 13

[13, 2]

Max : 13 + 2 = 15

Total Cost = 15

10
13
15
38 (Max)

By finding from Minimum we get the correct answer.

Every time we find 2 smallest size rope.

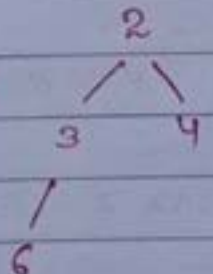
By array Everytime it takes O(N)

Total time : $O(N^2)$

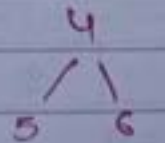
So, we use Min Heap.

Then we solve easily.

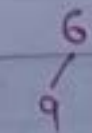
Time Complexity : O(NlogN)



2 + 3 = 5



4 + 5 = 9



6 + 9 = 15

5 + 9 + 15 = 29

Code :

```
long long minCost ( long long arr[], long long n ) {

        long long cost = 0;
priority_queue <long long, vector <long long> , greater <long long>> p;

    for (~~int~~ long long i = 0; i < n; i++) {
            p.push (arr[i]);
    }

    long long first, second;
    while ( p.size ( ) != 1 ) {
            first = p.top ( );
            p.pop ( );
            second = p.top ( )
            p.pop ( );
            first += second;
            cost += first;
            p.push ( first );
    }
    return cost;
}
```

problem 5 : Magician and chocolate (interview BIT)

Given N Bags, Each Bags Contains some chocolate.

On one unit of time, Kid choose a random Bag i, eats Bi chocolates, then magicians fills the bags with Bi/2 chocolates.

Find maximum number of chocolates that kid can eat in A unit of time.

may be Concept of Heap.

A = 5 (time)

[2, 4, 6, 8, 10] (Maximum chocolate)

sum = 10

Boy picks 10 chocalate

[2 4 6 8 5] sum = 10+8 = 18

A = 4

Boy pick bag of 8 Chocolate

[2 4 6 4 5]

A = 3 sum = 18+6 = 24

He pick bag of 6 chocolate

[2 4 3 4 5] sum = 24+5 = 29

A = 2

He pick 5 chocolate.

[2 4 3 4 2] sum = 29+4 = 33

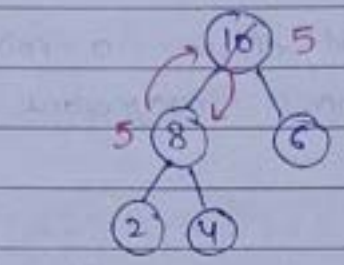
A = 1

He pick 4 chocolate

(A = 0) Time ends.

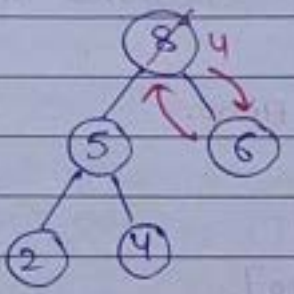
Boy picks maximum of 33 chocolate

we can solve this question by Max Heap.



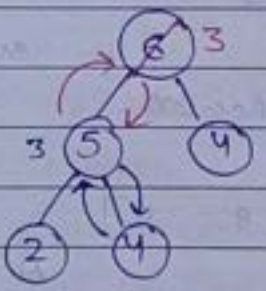
A = ~~5~~ 4

sum = ~~0~~ 10



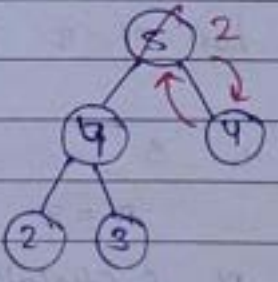
A = ~~4~~ 3

sum = ~~10~~ 18



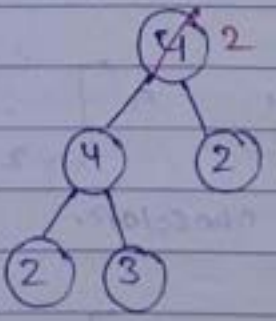
A = ~~3~~ 2

sum = ~~18~~ 24



A = ~~2~~ 1

sum = ~~24~~ 29



A = ~~1~~ 0

sum = ~~29~~ 33

final sum = 33

Code :

```
int solution :: nchoc (int A, vector <int> &B){
        priority_queue <int> p;
        long long int total = 0;

    for (int i = 0; i < B.size(); i++)
        p.push(B[i]);

    while (A != 0 && p.size() != 0){
        total = (total + p.top()) % 1000000007;
        if ( p.top() /2)
            p.push ( p.top() /2);
        p.pop();
        A--;
    }
    return total;
}
```

problem 6 : k largest Element :

* Arr[] = { 10, 20, 15, 18, 25, 19 }

(1) Sort → N log N

(2) K Element ko bahar Nikal do

Total time = K + N log N.
= N log N

*

Max Heap : creation : N log N ⇒ N

Then K element ant निकाल दो
↳ K log N.

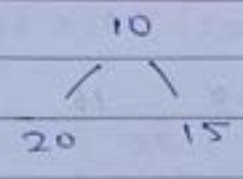
N + K log N.

* k - largest Element
↳ then create Min Heap

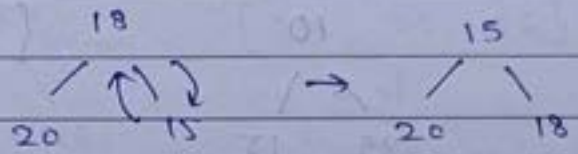
k - smallest Element
↳ then create Max Heap.

For this problem :
Create MIN Heap
of size K.
↳ K log K.

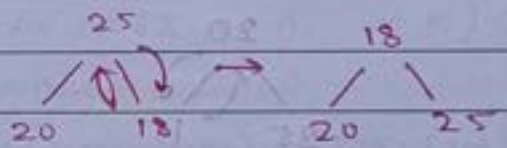
Min heap



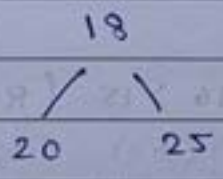
Now, 18 → 18 > 10



Now, 25 → 25 > 15



Now 17 → 17 < 18



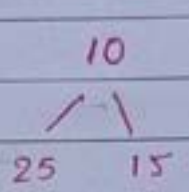
Largest Element : 18, 20, 25

Example 2 :

10 25 15 20 16 18 26

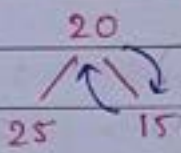
k = 3

Mean Heap of 3 elements

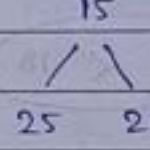


{10, 25, 15}

20 → 20 > 10 (Remove 10)

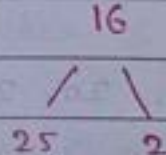


{20, 25, 15}



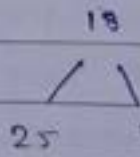
{15, 25, 20}

16 → 16 > 15 (Remove 15)



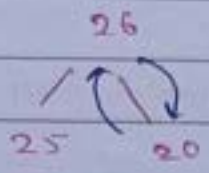
{16, 25, 20}

18 → 18 > 16 (Remove 16)

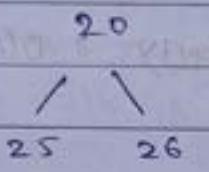


{18, 25, 20}

26 → 26 > 18 (Remove 18)



{25, 25, 20}



{20, 25, 26}

↓

Ans

Code :

```
vector<int> klargest(int arr[], int n, int K){
        vector<int> ans;
    priority_queue<int, vector<int>, greater<int>> p;

    for(int i=0; i<K; i++){
            p.push(arr[i]);
        }

    for(int i=K; i<n; i++){
        if(p.top() < arr[i]){
            p.pop();
            p.push(arr[i]);
        }
    }
    while(!p.empty()){
        ans.push_back(p.top());
        p.pop();
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

Lecture 51

(Heap)

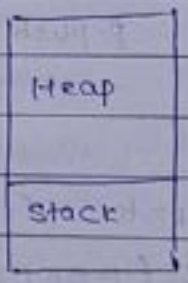
By help of Recursion
↳ gives segmentation fault
Space Complexity : O(N)

Stack :
↳ Solved and run properly
Space Complexity : O(N)

Segmentation fault : We want to access that memory which does not exist

Recursion works in Stack Memory

Heap ka space jyada hota hai.



Stack works in heap. On heap, only STL library Command Works.

Heap → Max Heap
↳ Min Heap
} priority Queue.

classmate
Date
Page

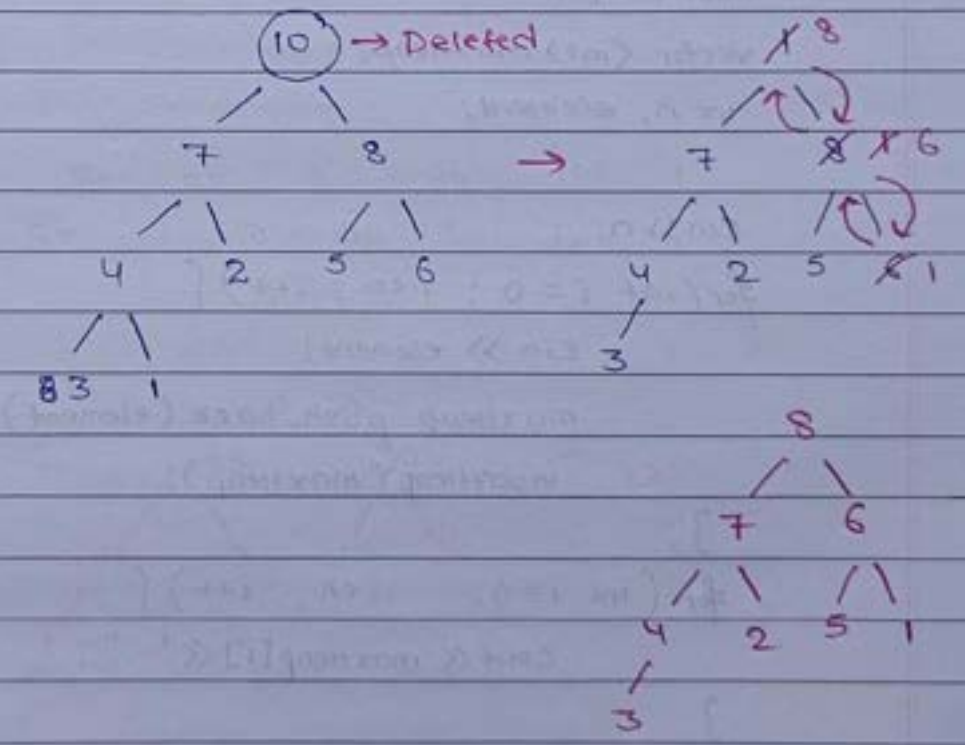
In Max Heap,

parent Node >= child Node

Max Heap & Min Heap can be represented in tree and Array both.

In Heap, creation always starts from top.
In Heap, Deletion always starts from top.

After Deleting the top Node, replace it from the last Node and then swap accordingly.



For creation, we have to use level order traversal in tree.

In Array, we easily access the Nodes and know about the blank space.

parent = (i-1)/2

i = child.

By tree, find the last Node takes time.

But in array, we know about the last space.

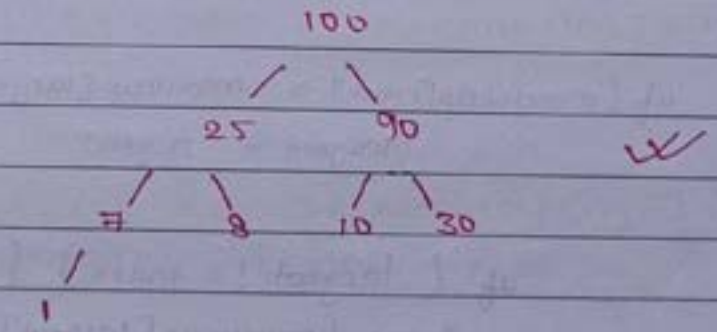
Code :

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector <int> maxHeap;
    int n, element;

    cin >> n;
    for(int i = 0; i<n ; i++) {
        cin >> element;
        maxHeap.push_back(element);
        insertHeap(maxHeap);
    };
    for (int i=0;    i<n ; i++) {
        cout << maxHeap[i] << " ";
    }
return 0;
}
Void insertHeap( vector <int> &maxHeap ) {
    int index = maxHeap.size() -1;
    int parent;
    while ( index > 0) {
        parent = (index -1)/2;
    // parent is small
```

```
        if (maxHeap[parent] < maxHeap[index]) {
            swap(maxHeap[parent], maxHeap[index]);
            index = parent;
        }
        // parent is big
        else
            break;
    }
}
```

output :

```
8
10  7  30  25  8  90  100  1
100  25  90  7  8  10  30  1
```

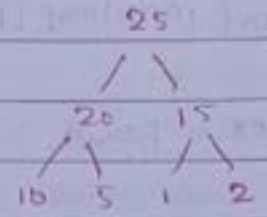


Time Complexity : N log N

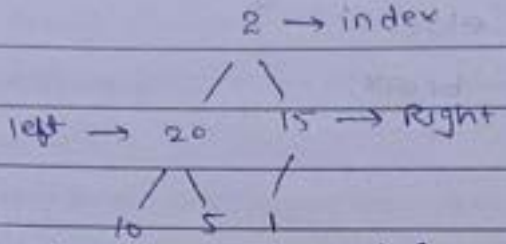
N → N element creation

log N → Every Element Swap to the Max of tree height.

* Deletion



Delete




```
Heapify (maxheap, largest) {
    largest = index;
    left = 2 * index + 1;
    right = 2 * index + 2;

if (maxHeap[left] > maxHeap[largest])
        largest = left;

if (maxHeap[right] > maxHeap[largest])
        largest = right;

    if ( largest != index) {
        Swap (maxHeap[largest], maxHeap[index]);
        Heapify (maxheap, largest);
    }
```

```
DeleteHeap(maxHeap);      // function call

Void DeleteHeap(vector<int> &maxHeap){
    // Replace first element with last element
    maxHeap[0] = maxHeap[maxHeap.size()-1];
    // Delete the last element
    maxHeap.pop_back();
    // Correct position pe le ke jao
    Heapify(maxHeap, 0);
}

Void Heapify(vector<int> &maxHeap, int index){
    int largest = index;
    int left = 2*index+1;
    int right = 2*index+2;
    int size = maxHeap.size();
    // check left side.
    if(left < size && maxHeap[left] > maxHeap[largest])
        largest = left;
    // check for right side
    if(right < size && maxHeap[right] > maxHeap[largest])
        largest = right;

    // swap
    if(largest != index){
        swap(maxHeap[largest], maxHeap[index]);
        Heapify(maxHeap, largest);
    }
    return;
}
```

Time Complexity : O(log N)
Space Complexity : O(log N)

→ Max Heap :

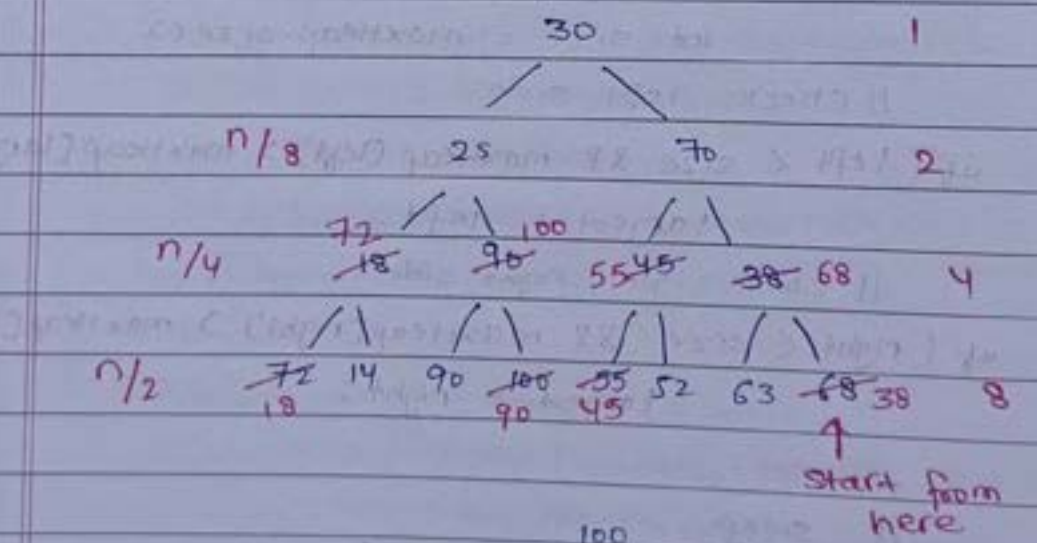
↳ Time Complexity : N log N.

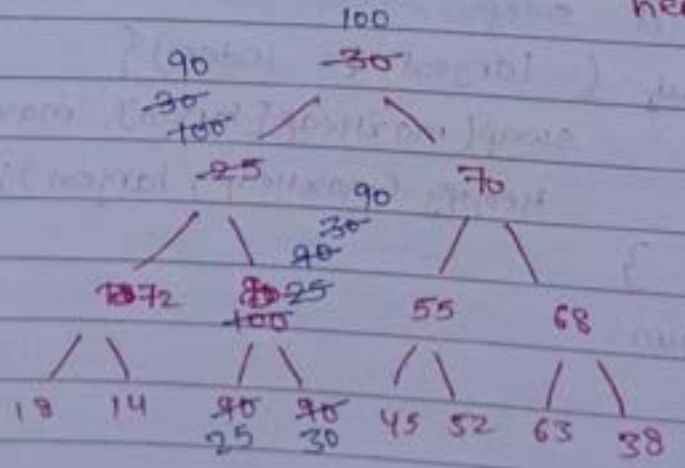
↓

Then we can use sorting instead of using Max Heap.

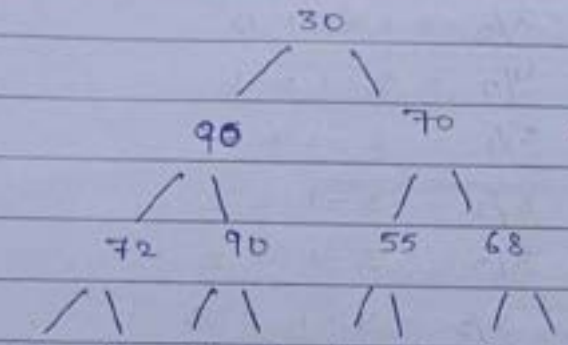
But, for Max Heap we can optimize N log N to N.

lets see how :-





classmate
Date
Page



→ $n/8$ → max two step down

→ $n/4$ → max one step down.

→ $n/2$ → 0 step.

$$\left[\frac{n}{4} \times 1 + \frac{n}{8} \times 2 + \frac{n}{16} \times 3 + \frac{n}{32} \times 4 \cdots \right]$$

$$= n \left\{ \underbrace{\frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \frac{4}{2^5} + \cdots}_{\hookrightarrow \text{sum nearly 1}} \right\}$$

$$= O(n \times x)$$

↳ Constant

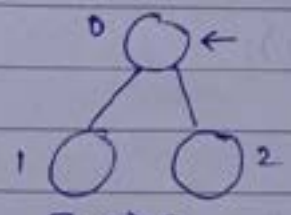
$$O(n)$$

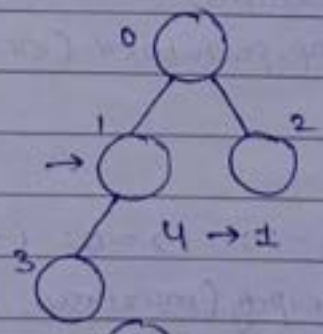
↳ Time Complexity

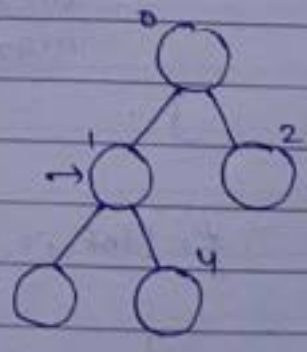
Space Complexity : $O(\log n)$.

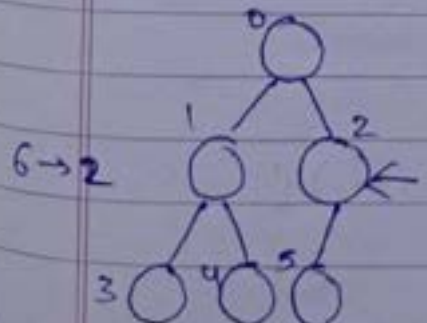
How to solve : How to find Starting Node.

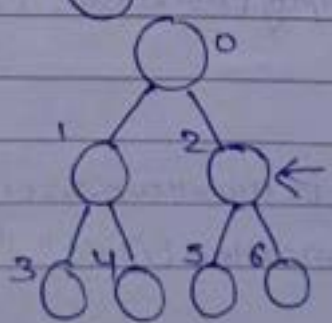
→











8 → 3

3 → 0 ⇒ 3/2 = 1 - 1 = 0

4 → 1 ⇒ 4/2 = 2 - 1 = 1

5 → 1 ⇒ 5/2 = 2 - 1 = 1

6 → 2 ⇒ 6/2 = 3 - 1 = 2

7 → 2 ⇒ 7/2 = 3 - 1 = 2

8 → 3 ⇒ 8/2 = 4 - 1 = 3

→
```
for (int i= size/2 -1 ;  i>=0; i--) {
        Heapify (maxHeap, i);
    }
```

* Code:

```
#include <iostream>
using namespace std;

int main () {
    vector <int> maxHeap;
    int n, element;
    cin >> n;
    for (int i=0; i<n; i++){
        cin >> element;
        maxHeap.push_back (element);
    }

    for (int i= n/2 -1 ; i>=0; i--) {
        Heapify (maxHeap, i);
    }

    for (int i=0; i<maxHeap.size() ; i++ ) {
        cout << maxHeap[i] << " ";
    }
```

```
    return 0;
}

Void Heapify (vector <int> &maxHeap, int index) {
        int largest = index;
        int left = 2 * index + 1;
        int right = 2 * index + 2;
        int size = maxHeap.size();

    if (left < size && maxHeap[left] > maxHeap[largest])
        largest = left;

    if (right < size && maxHeap[right] > maxHeap[largest])
        largest = right;

    if (largest != index) {
        Swap (maxHeap[largest], maxHeap[index]);
        Heapify (maxHeap, largest);
    }
    return ;
}
```

Lecture 52

Heap Hard

* **pair in STL :**

Two Element : int, char

①
```
class Node {
    int a;
    char b;
}
```

(int, char)
↑ ↑
first second

②
```
pair <int, char> p;
p.first = 10;
p.second = 'd';
```

②
```
pair <int, char> p;
p = make_pair (10, 'd');
```

③ for 3 Element :

```
pair <int, int, int> p;
```
↳ Wrong format

classmate
Date
Page

we use nested pair :

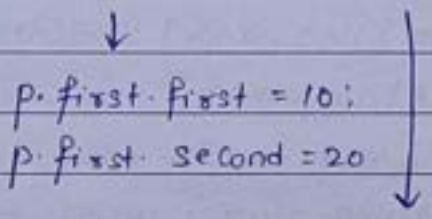
```
pair <int, pair <int, int>> p;
```

(4) for 4 element :

```
pair <int, pair <int, pair <int, int>>> p;
```

(5) for 3 Element :

```
pair < pair <int , int> , int > p;
```




```
p.second = 30;
```

```
p = make_pair (make_pair (10, 20), 30);
```

Code :

```
#include <bits/stdc++.h>
#include <vector>
using namespace std;

int main(){

    pair <int, int> p;
      p.first = 10;
      p.second = 20;

  //  p = make_pair (10, 20);
cout <<p.first << " " <<p.second <<" ";
```

```
pair<int, pair<int, int>> q;

    q.first = 1;
    q.second.first = 10;
    q.second.second = 20;

cout << q.first <<endl;
cout <<q.second.first <<endl;
cout << q.second.second <<endl;

return 0;
}
```

→

```
int main() {
    pair<int, int> p;
    p= make_pair (10, 20);

    pair<int, int> p2;
      p2 = p;                    // Copying.

  cout << p2.first << " " << p2.second ;
return 0;
}
```

Example

Sort on the basis of first : ascending

10 , 20

8 , 7

4 , 9

4 , 7

9 , 15

sort according to the first :

```
int main () {
  Vector <pair <int, int>> v;

    v.push_back (make_pair (10, 20));
    v.push_back (make_pair (8, 7));
    v.push_back (make_pair (4, 9));
    v.push_back (make_pair (4, 7));
    v.push_back (make_pair (9, 15));

  sort ( v.begin (), v.end ());

  for (int i=0;  i<5; i++)
    cout << v[i].first << " " << v[i].second << endl;

  return 0;
}
```

For descending :

```
sort (v.rbegin (),  v.rend ());
```

* Ascending order on the basis of second :

4, 8		11, 7	11, 7
11, 7	Ascending on	4, 8	3, 8
12, 13	⟶	3, 8	4, 8
3, 8	basis of second.	5, 9	5, 9
5, 9		12, 13	12, 13

↑

If second is same then check for first

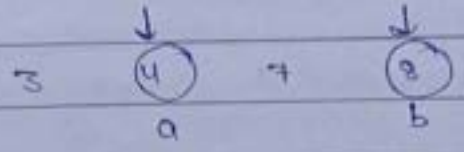
```
sort(v.begin(), v.end(), sortbysec);
```

→
```
bool sortbysec(int a, int b){
    return a > b;
}
```

→
```
bool sortbysec(pair<int, int>a, pair<int, int>b){
    return a.second > b.second;   // a.second < b.second;
}
```
ascending / descending

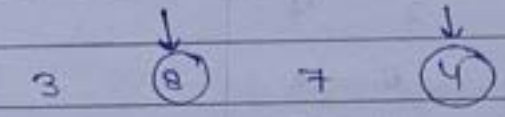
It works on the basis of second.

ascending order :



a < b, (return 1)

No change.



8 < 4 (return 0)

so, print 4 first, then 8.

* If second is same then check in first :

10	7
11	8
9	7

a.second < b.second

(10, 7) (11, 8)

7 < 8 return 1

(10, 7) (9, 7)

7 < 7 return 1

(9, 7), (10, 7), (11, 8).

```
bool sortbysec(pair<int, int> a, pair<int, int> b) {
    return a.second < b.second || (a.second == b.second &&
                                   a.first < b.first);
}
```

All Concepts are defined in STL in sort function and vector.

* Merge K sorted Array (Interview Bit)

1	2	4
2	6	8
3	4	9
4	11	12

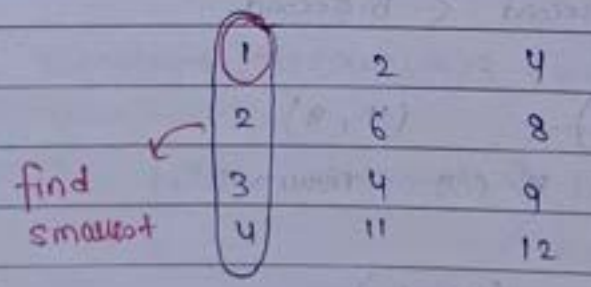
Method 1 :

1 2 4 2 6 8 3 4 9 4 11 12

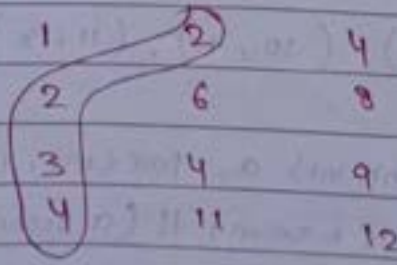
↳ sort these all

Time Complexity : $k * n \log(k * n)$.

Method 2 :



1



1, 2

for one element we do k comparison.

Total Element = $k * N$

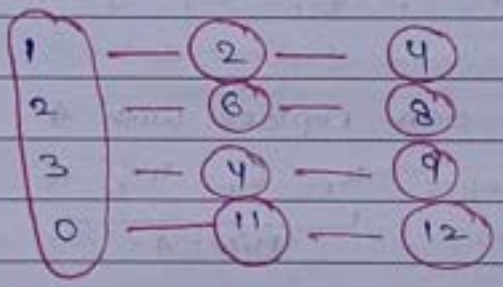
Time Complexity = $k * k * N$

$= k^2 * N$.

Method 3:

We want every time Min Element.

So, we use Min Heap.



Heap : 1 2 3 0

Min = 0

Ans : 0,

0 can replace with 11

Heap : 1 2 3 11

Min = 1

Ans : 0 , 1

1 can replace with 2

Heap : 2 2 3 11

Min = 2

Ans = 0, 1, 2

2 can replace with 4

Heap = 4 2 3 11

Min = 2

Ans = 0, 1, 2, 2

2 can replace with 6

Heap = 4 6 3 11

Min = 3

Ans = 0, 1, 2, 2, 3

3 can replace with 4

Heap = 4, 6, 4, 11

Min = 4

By this :

Ans = 0, 1, 2, 2, 3, 4, 4, 6, 8, 9, 11, 12

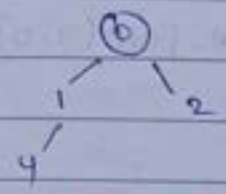
For 1 element : log k

Total Element = n k

Time Complexity = n · k · log k.

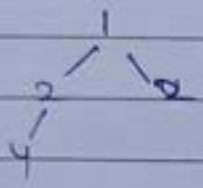
Example :

1 → 2 4
2 3 6
0 → 2 5
4 5 10



Heap : 1 2 (0) 4 (Min = 0)
2

Ans : 0



Heap = (1) 2 2 4 (Min = 1)
2

Ans : 0, 1

Heap = (2) 2 2 4 (Min = 2)

Ans = 0, 1, 2

By this we can solve,
we can give the next element also in the pair
(0, 2)
↑ on heap ← if 0 is go to ans, then replaced with 2.

Code :

```
Vector <int> Solution :: Solve (vector <vector <int>> &A) {
        Vector <int> ans;
        int row = A.size();
        int col = A[0].size();

    // Min Heap
    /// pair <int, pair <int, int>> p;
priority_queue <pair <int, pair <int, int>>>, vector <pair <int,
        pair <int, int>>>, greater <pair <int, pair <int, int>>>>
                                                    minheap;
```

```
// insert first column into minheap
for (int i=0; i<row; i++)
minheap.push(make_pair(A[i][0], make_pair(i,0)));

pair<int, pair<int,int>> p;

while(!minheap.empty()){
// Get minimum element, top
    p=minheap.top();
// Insert the value in answer vector
    ans.push_back(p.first);
    row = p.second.first;
    col = p.second.second;
    minheap.pop();
// Insert the next element of that row, first check
// whether that row is exhausted or not.
    if(col<A[0].size()-1)
    minheap.push(make_pair(A[row][col+1],
                           make_pair(row, col+1)));
}
return ans;
}
```

* Huffman coding :

String s : A B B C D A B D - - - -

50 character

A = 20

B = 4

C = 15

D = 19

E = 2

We send the ASCII Code of character.

A = 65

01000001

8 bit

50 character

↳ 50 x 8 bit = 400 bit.

We have to show only 5 character

इतनों को दूसरे से नाम देना है।

8 x 5 = 40 3 x 5 = 15 = 55

01000001

A → 000

B → 001

C → 010

D → 011

E → 100

50 character

↳ 50 x 3 = 150 bits.

table : 15 + 40 = 55

Total = 150 + 55 = 205 bits.

Minimized.

A	= 20	0	→ 20 × 1 = 20
B	= 4	1	→ 4 × 1 = 4
C	= 15	00	→ 15 × 2 = 30
D	= 9	01	→ 9 × 2 = 18
E	= 2	10	→ 2 × 2 = 4
			76

We can't use this; this some gives error

Table = 40 + 8 = 48

76 + 48

= 124

↳ Minimized.

Can we can minimize more ?

Yes

↳ How

If frequency is high, assign it with min-m no. of bit.

A	= 20	→ 0	=> 20
BC	= 15	→ 1	=> 15
D	= 9	→ 00	=> 18
B	= 4	→ 01	=> 8
E	= 2	→ 10	→ 4
			65

We can't use this create some error

Table = 40 + 8 = 48

65 + 48

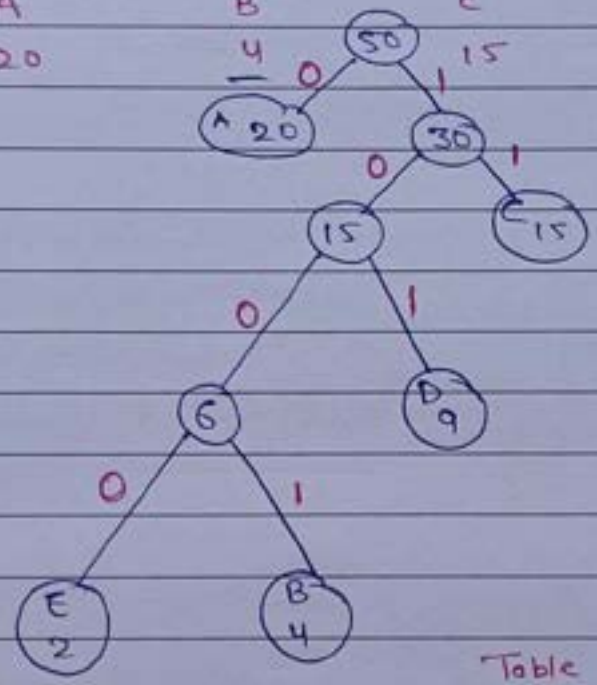
= 113

↳ minimum.

जिसका frequency सबसे ज्यादा उसको हम सबसे कम bit से represent करेंगे।

Freq ↑ = less bit
Freq ↓ = more bit.

A	B	C	D	E
20	4	15	9	2



A = 0 => 20
C = 11 => 30
B = 1001 => 16
D = 101 => 27
E = 1000 => 8

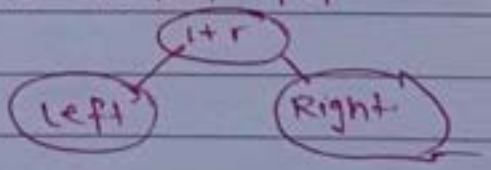
101

Table size = 40 + 14
= 54

Total = 155

Min Heap : Insert all element.

2 element pop



Lecture 53

Huffman Encoding (Algorithm)

5 9 12 13 16 45

Real life Example : file Compression

{ 5	9	12	13	16	45 }
a	b	c	d	e	f

Message : aabcdeaaabbeeccddaf- - - -

			Ascii	
a	→	5	98	01100001
b	→	9	99	01100010
c	→	12	100	01100011
d	→	13	101	⋮
e	→	16	102	⋮
f	→	45	103	⋮

Total = 100

R o h i t

Total bit = 100 x 8 = 800 bit

01001001 01101010 10101110 01001000 01

10001001

Ascii → 8 character

classmate
Date
Page

We have 6 character :

a, b, c, d, e, f

We can show this in 3 byte.

8 bits	a	→	000	→ 5
"	b		001	→ 9
"	c		010	→ 12
"	d		011	→ 13
"	e		100	→ 16
"	f		101	→ 45
				100

Here bit = $100 \times 3 = 300$ bits.

And for table :

$8 \times 6 + 3 \times 6 = 48 + 18$

$= 66$

Total $= 300 + 66$

$= 366$ bits.

In previous we use 800 bits

Now, only 366 bits.

$366 < 800$

		bits	
a	5	101	→ 5 × 3 = 45
b	9	100	→ 9 × 3 = 27
c	12	11	→ 2 × 12 = 24
d	13	10	→ 2 × 13 = 26
e	16	1	→ 16 × 1 = 16
f	45	0	→ 45 × 1 = 45
↳ max			183

But we can't use this Method

Table = 8 × 6 + 12 = 60

Total = 243

→ Reason :

Receiver →

100 → b

100 → df

It creates problem when we use the size of message is different.

We always use the size of each character with same number of bits.

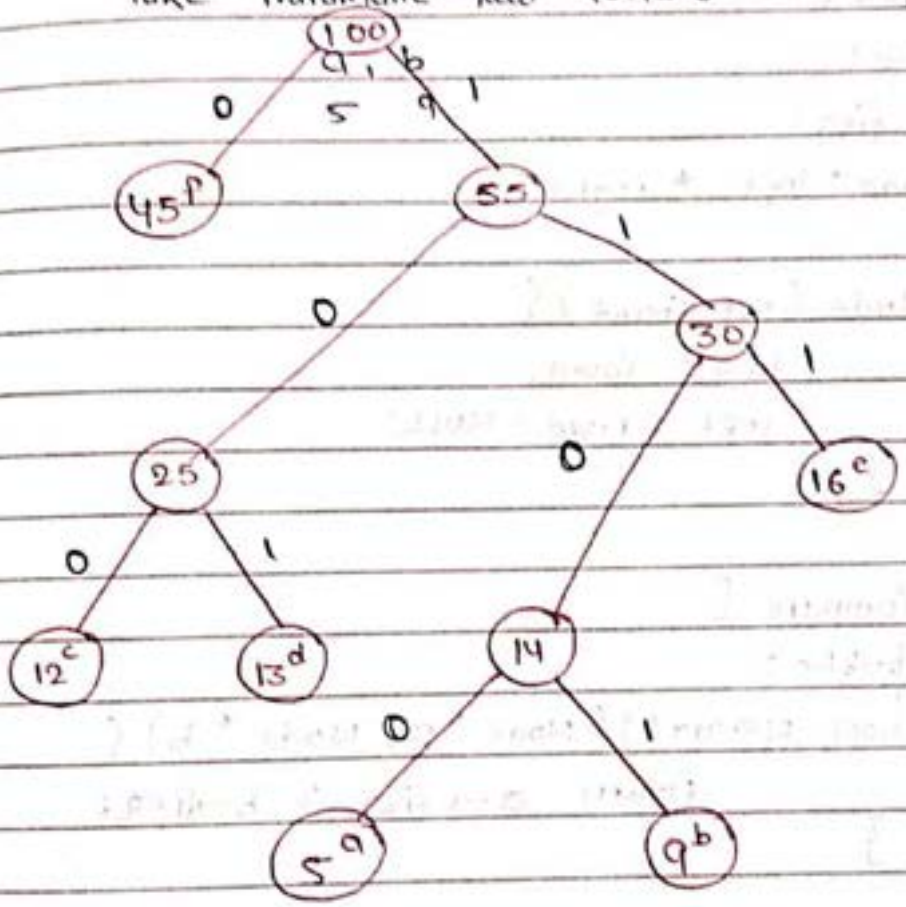
→ So, we use Huffman Encoding Here :

↓

Takes variable length of each character. But it take care of matching of two letter.

a	5	→	1 1 0 0
b	9	→	1 1 0 1
c	12	→	1 0 0
d	13	→	1 0 1
e	16	→	1 1 1
f	45	→	0

Take minimum two letter :



We create a Min Heap for all the character

1. Min Heap
2. push all the char in frequence.
3. Jab take Mera heap ka size ! = 1
   pop 2 element
   Combine
   push into heap.

Code :

Tree
↳
Node
↳ freq : char a, left, right.

```
Public :
Class Node {
    public :
    int freq;
    Node * left, * right ;

    Node (int Count) {
        freq = Count;
        left = right = NULL;
    }
};
Class Compare {
    public :
    bool operator() ( Node * a, Node * b) {
        return a → freq > b → freq;
    }
}
```

Problem : Find Median in a Stream :

Input :

5 15 1 3

This is wrong Approach we first sort the given array.

Even length = 4

$\frac{15+1}{2} = 8$

Input : 1 2 3 4 5

odd length

Median = 3.

Example : 1

5 15 1 3

Approach 1 :

sort the array

1 3 5 15

for n = 1 : {1}

Median = 1

for n : 2 : {1, 3}

Median : $\frac{1+3}{2} = 2$

for n = 3 : {1, 3, 5}

Median : 3

for n=4 {1, 3, 5, 15}

median = $\frac{5+3}{2} = 4$

Approach 1 is Wrong, Because have to sort after each step.

Example : 5 15 1 3

Approach 2 :

for n = 1 : {5}

sort : 5

Median = 5

for n = 2 : {5, 15}

sort = 5, 15

Median : $\frac{15+5}{2} = 10$

for n = 3 : {5, 15, 1}

Sort : 1, 5, 15

Median = 5

for n = 4 : {5, 15, 1, 3}

sort : 1, 3, 5, 15

Median = $\frac{5+3}{2} = 4$

Time Complexity : $N \cdot N \log N$

$= N^2 \log N$.

Approach 3:

Example -1

2 4 6 7 8 9

↑ Median

$\frac{6+7}{2} = 6.5$

Case 1: left = right → Median = 6.5

Case 2: left > Right → Median = 6

If 1 Comes

on left 4 element, in right 3 element

Case 3: Right > left. → Median = 7.

If 10 Comes

on left 3 Comes, on right 4 element.

Example 2)

2 4 10 ↓ 16 20 25

Median = 13

Now, 6 Comes.

2 4 6 10 16 20 25

↑

Median = 10

If 30 comes

2 4 10 16 20 25 30

↑

Median.

Lecture - 54

(Hashing)

Searching :

1. Array
   - ↳ sorted : $O(\log n)$ Binary search
   - ↳ unsorted : $O(n)$ Linear search
2. Linked list : $O(n)$
3. Binary Tree : $O(n)$
4. Binary Search Tree : $O(n)$
5. Balanced Binary search tree : $O(\log n)$
6. priority Queue (Min / Max) Heap : $O(n)$
7. stack : $O(n)$
8. Queue : $O(n)$

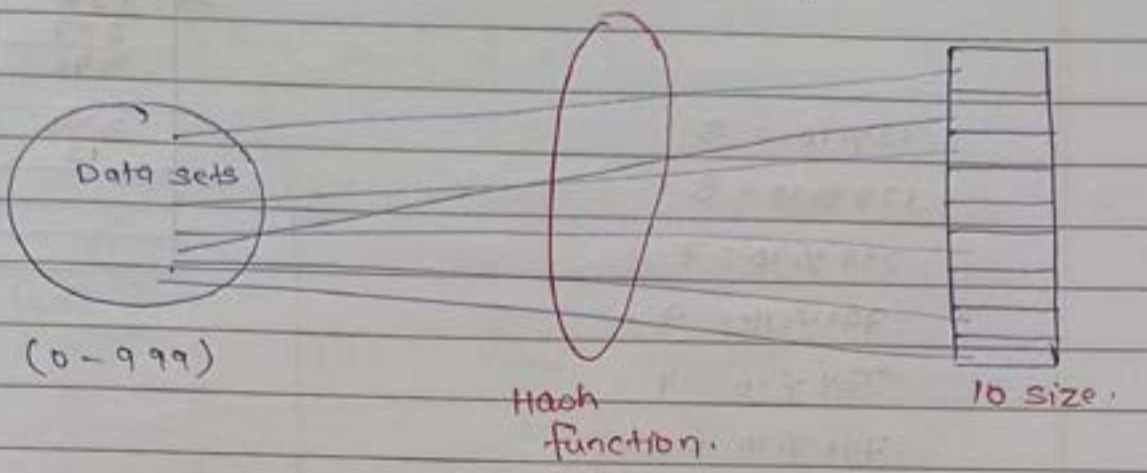
In all searching can take minimum of $O(\log n)$ time.

Can We reduce the time Complexity for searching.

Yes,

Hashing can search in $O(1)$ time.

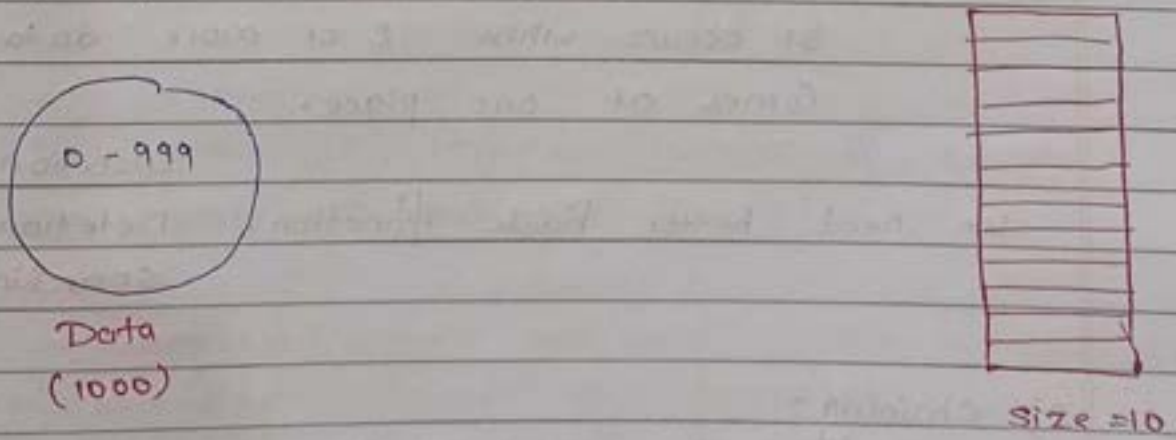
Hashing : O(1) searching



Hash function helps to search any element in O(1) time.

If we have a data set of 1000 data, but we write a function for search in O(1) time for only some data. like 10 data.

We can store only 10 data for searching in O(1) time.



We have 1000 data, but we can satore important data only which is 6.

{13, 128, 257, 999, 334, 776}

We will do num % 10 ;

Size = 10

999	9
128	8
257	7
776	6
	5
334	4
13	3
	2
	1
	0

13 % 10 = 3

128 % 10 = 8

257 % 10 = 7

999 % 10 = 9

334 % 10 = 4

776 % 10 = 6

Now,

search 128

128 % 10 = 8

↓

Go to

Searching in O(1).

→ Problems :

1) Collision :

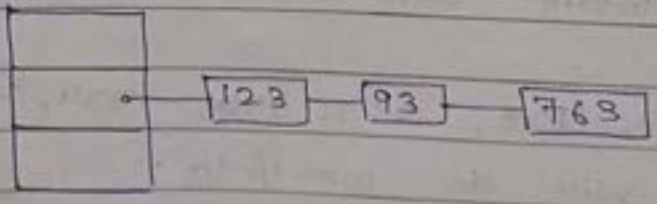
It occurs when 2 or more data comes at one place.

We need better hash function.

Insertion : O(1)
Deletion : O(1)
searching : O(1)

2) Chaining :

If two or more element come, then we do chaining.



Insertion : O(1)
Deletion : O(N)
searching O(N)

classmate
Date
Page

⇒ $\frac{n}{m}$ → Total No. of Element.
m → Size of array

$$\alpha = \frac{n}{m}$$

↓
load factor.

Let Total Element = 30
Array size = 10

$$\alpha = \frac{30}{10} = 3$$

We, know that on an average at one place 3 element occur.

Then,

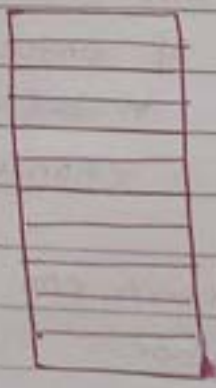
Searching : $\theta(1 + n/m)$

↓

$O(1)$

We can tell, Maximum number of element a place Contain.

③ Open Addressing :



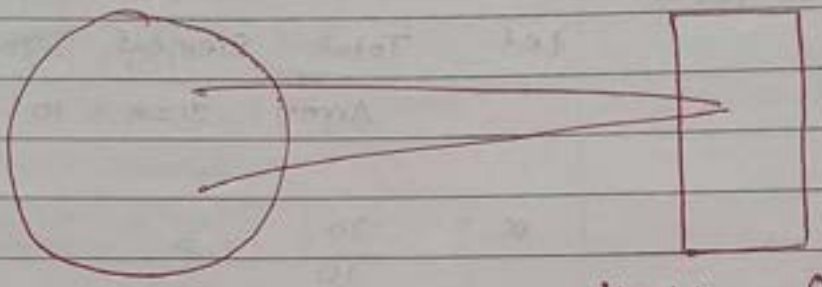
If collision occurs, then find another place for store.

For finding another place, we use various Method.

1) Linear probing.

* Hash code : Blockchain, Message transfer

SHA-256, SHA-512 . . .



Hashing O(1) ⇒ time

If you send any Message
↳ It will ~~first~~ encoded into some other code.
↳ If anybody in b/w has add / delete something to your Message.
↳ How you can check that the Message you send can received as you send.

By the help of hash code you can achieve it.

↔ You will send a Hash code with your Message.
If anyone do something with your Message in between, the Hash code should be changed.

The Receiver can detect easily with the help of Hash code.

→ your whole Message can be Converted into 256 bit: either your message is too big or too small.

Hashing can find a unique place for the data.

→ If you change a single bit, it can change the whole hash code.

* Database Leak :

↳ your all data leaked, but your password never leaked.

↓

Why ?

↓

because they store the hascode of the password.

Here, also collision occur but the probability is very less.

password से Hash Code generate होता है।

But reverse true नहीं होगा।

↳ one way

↳ If Database Hack

↳ then password can be Changed easily.

↳ If Data leak

↳ then, Not possible to changed.

classmate
Date
Page

* Example :

Satyam = 7
paridhi = 8
Prashant = 2
Mohan = 6
Rohan = 9

How we store this ? So we access in O(1) time.

Earlier, in Array :

A[0]	A[1]	A[2]	A[3]	A[4]
7	8	2	6	9

B[0]	B[1]	B[2]	B[3]	B[4]
satyam	paridhi	prashant	6	9

↳ Take O(N)

On new Method :

A[satyam] = 7
A[paridhi] = 8
A[prashant] = 2 O(1)
A[Mohan] = 6
A[Rohan] = 9

We use map to store this.

map <int, int> m;

key — string

value — string

If there are two paridhi, then how we use this,

We can't able to use.

We only use where there is only one paridhi means, it has to be unique id.

* MULTIMAP :

Map : Balanced Binary search tree

Search (O(log n))

Insert O(log n)

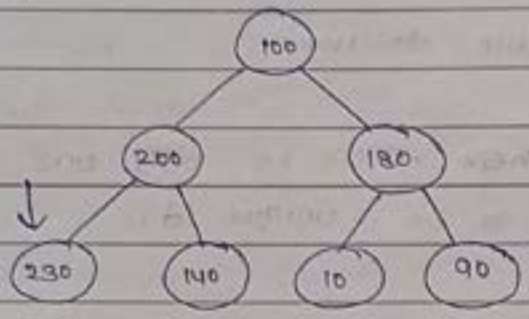
Delete O(log n).

It print data in sorted order.

Code :

```
#include <map>
#include <iostream>
using namespace std;

int main() {
    map <int, int> m;

    m[3] = 84;
    m[2] = 8;
    m[9] = 15;
    m[10] = 43;
```

```
for (auto i = m.begin(); i != m.end(); i++) {
    cout << i->first << " " << i->second << endl;
}
cout << m.size() << endl;
return 0;
}
```



itr = m.begin()

itr = 230;

if we write m.end() = 90

itr < m.end();

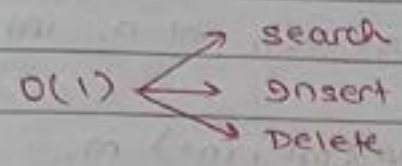
230 < 90

↳ is it valid.

So, we write !=

m.end() is the address of the next address of the last address.

* unordered map :



It cannot give data in sorted order.

Code changed :

```
#include <unordered_map>

unordered_map <int, int> m;
```

* problem : First Element to occur k times.

1 7 4 3 4 8 7

K = 2

num	Count
1	1
7	1
~~4~~	~~1~~
3	1
4	2

map : $\log n$ for 1 element
for N element : $N \log N$.

unordered-map : $O(1)$ for 1 element
for N element : $O(N)$

So, We use unordered_map.

code :

```
int firstElementKTime(int a[], int n, int K){

    unordered_map<int, int> m;
    for(int i=0; i<n; i++){

        if(m.count(a[i])==0){
            m[a[i]] = 1;
        }
        else{
            m[a[i]]++;
        }
        if(m[a[i]]==K)
            return a[i];
    }
    return -1;
}
```

* problem : Largest subarray with 0 sum.

A[] = {15, -2, 2, -8, 1, 7, 10, 23}

Approach 1 :

find all the subarray and find the largest subarray

```
for(      ){
    for(      ){

    }
}
```

$O(n^2)$

Approach 2 :

A[] : { 15, -2, 2, -8, 1, 7, 10, 23 }

sum of Array

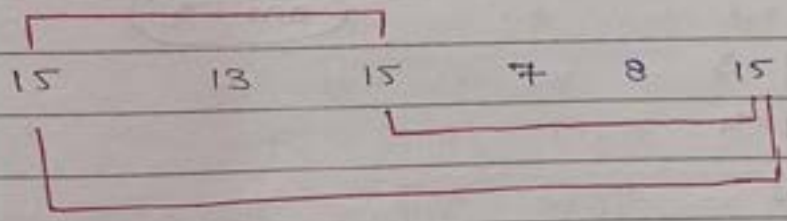
{ 15, 13, 15, 7, 8, 15, 25, 48 }

(A) 1 1 1 1 (B)

20 +5 -7 +9 -7 20

*Traverse sum array by two pointer :

if value of two index is same then the sum is 0.



O(n) time.

Array 1 : 15, [-2, 2]

sum = 0

Array 2 : 2 [-8 1 7]

sum = 0

O(n²)

Array 3 : 15 [-2 2 -8 1 7]

↑ sum = 0

longest subarray.

- Time Complexity : O(N²).

We have to check in O(1) time that this element is present previously or not.

We can't update more than one times.

0	1	2	3	4	5	6
15	-2	2	-8	1	7	-8

आगे तक आया है ?

0	1	2	3	4	5	6
No	No	Yes	No	No	Yes	Yes

then m[13] = 1

m[15] = 0

Length = 5

m[15] = 0
m[13] = 1
m[7] = 3
m[8] = 4

ans = 5

Edge case :

When sum = 0

then we do len = i + 1

Example :-

{ 15 -2 2 -8 1 7 -8 }

len = 0

15 Not Come before.

then,

m[15] = 0

→ 15 -2 2 -8 1 7 -8

len = 0

sum = 13

13 Not Come before

m[13] = 1

→ 15 -2 2 -8 1 7 -8

sum = 15

15 Comes before

find length

2 - 0 = 2 len = 2

→ 15 -2 2 -8 1 7 -8

sum = 7

7 Not Come before

m[7] = 3

→ 15 -2 2 -8 1 7 -8

sum = 8

8 not Come before

m[8] = 4

classmate
Date
Page

→	15	-2	2	-8	1	7	-8

sum = 15

It come before

len = 5 — length = 5 - 0 = 5

→	15	-2	2	-8	1	7	-8

sum = 7

Come before

length = 6 - 3 = 3

(3 < 5)

So len = 5.

Traverse the array,
and find the sum.

If sum is come before then it means the sum from that place to current index is 0.

So, find the length and return.

and this process continues till last

The Maximum length can be returned as a answer.

code :

```
int maxLen ( vector <int> &A, int n){

        int len = 0;
      // number, index
      unordered_map <int, int> m;
      int sum = 0;
      for (int i=0; i<n; i++){
            sum += A[i];
          // Exist
          if (sum == 0)
            len = i+1;
          if (m.count(sum)) {
              len = max (len, i-m[sum]);
            }
          // Not exist
            else {
              m[sum] = i;
            }
        }
      if (sum == 0)
        return n;
      return len;
    }
};
```

Lecture - 55

Hashing With Maps

problem 1 : Smallest Range in K lists :

Given K sorted list of integers,

Task : find smallest range that includes at least one element from each of the K lists.

If more than 1 found, then return first such range.

Example :

arr = {{ }}

1	3	5	7	9
0	2	4	6	8
2	3	5	7	11

Row wise sorted

1	3	5	7	9
0	2	4	6	8
2	3	5	7	11

(min, Max)
(A B)

(A <= B)

{ 1 , 0 , 2 }

mini = 0 (0, 2)
max = 2

classmate
Date
Page

(0, 2) → length = 2

Either we increase 0, or we decrease 2.

if we decrease 2,

↳ No option, because no any element present in left of 3rd array.

If we increase 0

↳ 2 Comes in limit

{1, 2, 2}

(1, 2) → range

↳ length = 1

Increase smallest value :

Increase 1

↳ 3 Comes

{3, 2, 2}

(2, 3)

↳ length = 1

2 कि जगह 4 :

{3, 4, 2}

(2, 4) → length = 2

⋮

जब तक 1 row end नहीं हो जाए।

Min Heap

range के element का Row or Column भी मालूम होना चाहिए।

Min heap :

pair<int, pair<int, int>> p;
value row col

सबसे छोटा वाला length = 1 आया था।
length 1 में जो सबसे पहले आया है
उसको return करेंगे।

start & end. → we store the value of range here, and the min-m value is stored.

If second time come with min-m value then can't change.

① if (end - start > maxi - mini) {
    end = maxi
    start = mini
② if (col = n-1)
    break
③ else {
    maxi = max(maxi, arr[row][col+1]);
    minheap.push(make_pair(arr[row][col+1],
        make_pair(row, col+1));

Code :

pair<int,int> findSmallestRange(int KSortedArray[][N], int n, int K){

priority_queue<pair<int, pair<int,int>>, Vector<pair<int, pair<int,int>>>, greater<pair<int, pair<int, int>>>> heap;

int mini = INT_MAX;

```
        int maxi = INT_MIN;
        int row, col;
    for (int i = 0; i < k; i++) {
            row = i;
            col = 0;
        heap.push (make_pair (ksortedArray[row][col], make_pair (row, col)));
        mini = min (mini, ksortedArray[row][col]);      // 1 (3)  (1,0,2)
        maxi = max (maxi, ksortedArray[row][col]);      // 1 (2)
    }
        int start = mini;
        int end = maxi;
    pair <int, pair <int, int>> temp;
        while (!heap.empty()) {
            temp = heap.top();
            heap.pop();
            row = temp.second.first;
            col = temp.second.second;
            mini = temp.first;
        if (end - start > maxi - mini) {
                end = maxi;
                start = mini; }
        else if (col == n-1)
                break;
            else {
                maxi = max (maxi, ksortedArray[row][col+1]);
        heap.push(make_pair(ksortedArray[row][col+1], make_pair
                (row, col+1)));
            }
        }
    return {start, end};
}
};
```

Problem 2: 2 Sum (interview Bit)

[2, 7, 11, 8] sum = 9

find two index whose sum = 9.

Approach 1: (Brute force)

```
for ( int i=0;  i<n;  i++ ) {
    for ( int j=0;  j<n;  j++ ) {
        if (arr[i] + arr[j] == sum) {
            first = i+1;
            second = j+1;
        }
    }
}
```

$O(n^2)$

By this we can find.

But, what if there is more than one possible answer:

{ 5, 7, 8, 3, 2 }

sum = 10

possible = (7, 3)
(8, 2)

Approach 2:

{5, 7, 8, 3, 2}

Sum = 10

By Heap & Map

m[5] = 0 → 5 और चाहिए | Not seen before

(1, 3)

- m[7] = 1 → 3 more req Not seen before
- m[8] = 2 → 2 more. Not seen before
- m[3] → 7 more, yes seen before

prefer small index value.

T.C O(n)

S.C = O(n)

{5, 7, 8, 7, 3, 2}

Approach :

Make a unordered map;

Make a vector to store the answer.

↓

5, 7, 8, 7, 3, 2 Sum = 10

10 - A[i] = 5 (is Exist)

↓ NO

then,

A[i] = 5 → पहले से Exist कर रहा है ?

↓ NO

m[5] = 0

i++;

classmate
Date
Page

10 - 7 = 3 → isExist

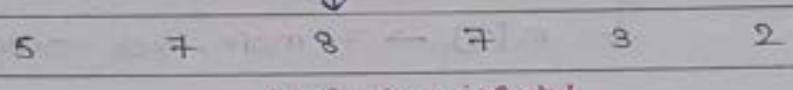
↓

No.

A[i] = 7 → isExist

↓ No

m[7] = 1



10 - 8 = 2 → isExist

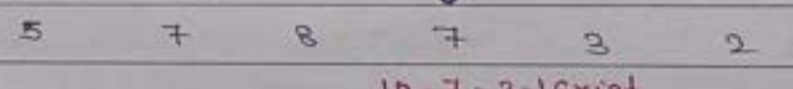
↓

No

A[i] = 8 → isExist

↓ No

m[8] = 2



10 - 7 = 3 → !Exist

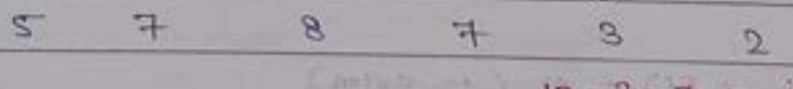
↓

No

A[i] = 7 → isExist

↓ yes

i++



10 - 3 = 7 = isExist

↓

yes

first = ans = index of 7

= 1

second = ans = i = 4

according to 0 based indexing

If 1 based indexing:

ans + 1

ans = {2, 5}.

code:

```
unorder_map<int, int> m;
vector<int> ans;

for (int i=0; i< A.size(); i++) {
    if (m.count(B-A[i])) { // if seen before
        ans.push_back (m[B-A[i]] + 1);
        ans.push_back (i+1);
        return ans;
    }

    else { // if does not seen before
        if (m.count(A[i]) == 0)
            m[A[i]] = i;
    }
}
return ans;
}
```

classmate
Date
Page

Problem 3: Largest subarray of 0's and 1's :

Example $A = \{0, 1, 0, 1\}$

Taske find largest subarray with equal No. of 0's & 1's.

Brute force :

```
Count_0 = 0;
Cout_1 = 0;
```

Cout all 0's & 1's.
find Maxm possible :

0 1 0 1
0 1 1 0 0 1
0 1 0 1 0 1
✓ 0 1 0 1
↳ Maxm

Subarray Sum :

$A = [0, 1, 0, 1]$
0 1 1 2
↳ Sum equal but size = 4 (for largest)

So, this Approach don't Work.

After Modifying :

Count_0 - Count_1 = 0

↳ then answer Comes.

Next Approach :

Replace 0 With (-1)

A = [0, 1, 0, 1]

[-1, 1, -1, 1]

-1 0 -1 0

Sum = 0

↳ When (1) on Sum Array you find 0.

(2) Sum is again Comming same in Array.

-1 0 -1 0

Example :

B = { 0, 0, 1, 0, 1, 1, 1 }

Sum = { -1, -2, -1, -2, -1, 0, 1 }

↑ m[-1] = 0

पहले को -1 आ

↳ No, it is Not answer

↓

{ -1, -2, -1, -2, -1, 0, 1 }

-2, Not found previously

↳ it is not answer.

m[-2] = 1

classmate
Date
Page

→ {-1, -2, -1, -2, -1, 0, 1}

found before

len = 2-0

len = 2

remain same

m[-1] = 0

→ {-1, -2, -1, -2, -1, 0, 1}

found before

2-1 = 2

len = 2

remain same

m[-2] = 1

→ {-1, -2, -1, -2, -1, 0, 1}

found before

len = 4

4-0 = 4

remain same

m[-1] = 0

→ {-1, -2, -1, -2, -1, 0, 1}

sum = 0

then

len = index + 1

len = 6

= 6

→ {-1, -2, -1, -2, -1, 0, 1}

not found before

m[1] = 6

unorder_map <int, int> m;

sum index.

Code:

```
int maxLen(int arr[], int N){
    int len = 0;
    unordered_map<int, int> m;
    int sum = 0;

    for(int i=0; i<N; i++){

        if(arr[i]==1){
            sum++;
        }else{
            sum--;
        }

        if(sum==0)
            len = i+1;

        else if(m.count(sum)){
            len = max(len, i-m[sum]);
        }
        else{
            m[sum] = i;
        }
    }
    return len;
}
};
```

problem 1: Count distinct elements in every window.

$A = \{1, 2, 1, 3, 4, 2, 3\}$

$N = 7$ $\quad K = 4$

Count of distinct element in every window of size K.

$A = \{1, 2, 1, 3, 4, 2, 3\}$

Window 1:

1, 2, 1, 3

distinct = 3

Window 2:

2 1 3 4

distinct = 4

Window 3:

1 3 4 2

distinct = 4

Window 4: 3 4 2 3

distinct = 3.

Approach :

Window 1 : 1, 2, 1, 3

m[1] = 0 X 2
m[2] = 1
m[3] = 0 1

3

Count = ∅ X ∅ 3

Window 2 :

1 2 1 3 4

Excludent

m[1] = 0 X
m[2] = 0 1
m[1] = 0 1
m[3] = 0 1
m[4] = 0 1

4

Count = ∅ X ∅ ∅ 4

Window 3 :

2 1 3 4 2

m[2] = 0 X
m[1] = 0 1
m[3] = 1
m[4] = 0 1
m[2] = 0 1

4

Count = ∅ X ∅ ∅ 4

Window 4 :

1 3 4 2 3

m[1] = 0 X
m[3] = 0 2
m[4] = 1
m[2] = 0 0

3

Count = ∅ X ∅ 3

Ans : 3 4 4 3

A = {1, 2, 1, 3, 4, 2, 3}

distinct_count = ~~0~~ X ~~2~~ 3

m[1] = X 2
m[2] = 1
m[3] = 1

ans: {3,

{1, 2, 1, 3, 4, 2, 3}

distinct_count = ~~3~~ 4
→ because first removed

m[1] = ~~2~~ 1
m[2] = 1
m[3] = 1
m[4] = 1

ans: {3, 4,

{1, 2, 1, 3, 4, 2, 3}

distinct_count = ~~4~~ ~~3~~ 4

m[1] = 1
m[2] = ~~2~~ ~~0~~ 1
m[3] = 1
m[4] = 1

ans = {3, 4, 3, 4

{1, 2, 1, 3, 4, 2, 3}

distinct count = 3

m[1] = 0

m[2] = 1

m[3] = 2

m[4] = 1

ans = {3, 4, 4, 3}

code:

```
unordered_map <int , int> m;
  int distinct_count = 0;
  for ( int i=0;  i< k; i++) {
        m[A[i]++;
        if (m[A[i]] == 1)
          distinct_count ++;
    }
    vector <int> ans;
    ans.push_back(distinct_count);
for (int i= k;  i <n;  i++) {
        m[A[i-K]] --;
        if (m[A[i-K]] == 0)
          distinct_count - -;

        m[A[i]] ++;
        if (m[A[i]] == 1)
          distinct_count ++;

      ans.push_back (distinct_ count);
    }
    return ans;
  }
};
```

Lecture - 56

Hashings and Maps

Problem 1 : First Repeating Element (Interview Bit)

Given : Array A
size N

find first repeating element.

A = [10, 5, 3, 4, 3, 5, 6]

first character that repeat

Brute for Approach : $O(n^2)$

10 5 3 4 3 5 6

Check for all 10 is present or not

NO

10 5 3 4 3 5 6

5 present or not

yes

return 5.

2nd Approach :-

0	1	2	3	4	5	6
10	5	3	4	3	5	6

↑

10 पहले आया है

↳ No

m[10] = 0

10	5	3	4	3	5	6

↑

5 पहले आया है

↳ No

m[5] = 1

10	5	3	4	3	5	6

↑

3 पहले आया है

↳ No

m[3] = 2

10	5	3	4	3	5	6

↑

4 पहले आया है

↳ No

m[4] = 3

10	5	3	4	3	5	6

↑

3 पहले आया है

↳ yes

index = 2

↳ पहले का index

classmate
Date
Page

10 5 3 4 3 5 6

↑

5 पहले आया है

↳ yes

Index = 2

1 < 2 (yes)

Index = 1

10 5 3 4 3 5 6

↑

6 पहले आया है

↳ No

m[6] : 6

At last return -1 (if Not any index stored)

If stored return index.

Another Approach :

We can traverse the reverse,
So we can't have access the index.

10 5 3 4 3 5 6

↑

ans = -1

Not Come before

m[6] = 1

classmate
Date
Page

10 5 3 4 3 5 6

↑

Not come before

m[5] = 1

10 5 3 4 3 5 6

↑

Not come before

m[3] = 1

10 5 3 4 3 5 6

↑ Not come before

m[4] = 1

10 5 3 4 3 5 6

↑ Come before

ans = 3

10 5 3 4 3 5 6

↑ Come before

ans = 5

10 5 3 4 3 5 6

↑ Not come before

m[10] = 1

Code :

```
int ans = -1;
unordered_map <int, bool> m;

for(int i= A.size()-1; i>=0; i--) {
        if (m.count(A[i]))
            ans = A[i];

        m[A[i]] = 1;
    }
return ans;
}
```

problem 8: Subarrays with equal 1s and 0s

Task: Total no. of subarray with equal number of 1 and 0.

A[] = {1, 0, 0, 1, 0, 1, 1}

Approach 1:

0	1	2	3	4	5	6
1	0	0	1	0	1	1

(0, 1), (0, 3), (0, 5) ~~(0, 6)~~ (1, 6), (2, 3) (2, 5) (3, 4) (4, 5)

Brute force

find all subarray:

(0, 0) (1, 1) ...
(0, 1) (1, 2)
(0, 2) ⋮ $O(n^2)$
(0, 3) ⋮
⋮
⋮

count all the subarray for equal 0 & 1.

$O(n)$

Total time: $O(n^3)$

classmate
Date
Page

Approach 2:

find all subarray

Count_0 = 0

Count_1 = 0

1, 0, 0, 1, 0, 1, 1

When we traverse this for finding subarray at same time, we count 0 & 1.

and when ever we find the equal value of count0 and count1.

We can do Count++.

Time Complexity = $O(N^2)$

Approach 3:

1 0 0 1 0 1 1

We use sumarray

1 0 0 1 0 1 1

from 0 we can't find any soln

we make 0 as -1 for solve the problem.

	1	0	0	1	0	1	1
Sum =	1	0	-1	0	-1	0	(1)
		(1-1)	(0-1)	(-1+1)	(0-1)	(-1+1)	0+1

classmate
Date
Page

Sum : 1 0 -1 0 -1 0 1

↑

1 not present m[1] = 1

count = 0

if 0

↳ (count of 0) + 1

1 0 -1 0 -1 0 1

↑

0 not present

m[0] = 1 (0 comes means equal no. of 0 and 1)

count = 1

1 0 -1 0 -1 0 1

↑

-1 not present

m[-1] = 1

1 0 -1 0 -1 0 1

↑

0 present

m[0] = 2

Count = 3

1 0 -1 0 -1 0 1

↑

-1 present

m[-1] = 2

Count ++

Count = 4

1 0 -1 0 -1 0 1

↑

0 present

m[0] = 3

Count = 7

classmate
Date
Page

1 0 -1 0 -1 0 1

↑ present

count++

count = 8

m[0] = 1

↳ because if 0 comes we add one more.

0 → 1 जोड़ा add होगा।

1, -1 → जितना value है उतना ही add होगा।

Code :

```
int sum = 0;
long long int final = 0;
unordered_map <int , int> m;

m[0] = 1;

for (int i = 0; i < n; i++) {
        if (arr[i])
          sum++;
        else
          sum--;

        final += m[sum];
        m[sum]++;
    }
return final;
}
```

problem 5: Subarray with B odd numbers: (Interview Bit)

Given: an Array
A integer B

Task: the total number of subarrays having exactly B odd numbers.

Example:

A = {4, 3, 2, 3, 4}

B = 2

Brute force

4 3 2 3 4

(4)	(3)	(2)	(3)	(4)
(4, 3, 2)	(3, 2)	(2, 3)	(3, 4)	
(4, 3, 2, 3) ✓	(3, 2, 3) ✓	(2, 3, 4)		
(4, 3, 2, 3, 4) ✓	(3, 2, 3, 4) ✓			

Count = 4

$O(N^2)$

2nd Approach

K = 2

Index:	0	1	2	3	4	5	6
Array:	1	2	4	5	6	8	9
odd:	1	1	1	2	2	2	3

m[0] = 1

5 6 8 9

4 5 6 8 9

2 4 5 6 8 9

} 3

Exclude 1 2 4 5 6 8 9

include 5 6 8 9.

m[1] = 3

m[3] = 1

m[3-2]

m[1].

*

1	2	4	5	6	8	9	10
1	1	1	2	2	2	3	3

(3 - 2) = 1

1 वाला 3 hai

↑

include कर लो।

1	2	4	5	6	8	9	10	11
1	1	1	2	2	2	3	3	4

4 - 2 = 2

↓

3 बार

ans में 3 include कर लो।

* Example:

↓					
5	6	7	8	9	(B = 2)
1					

	↓			
5	6	7	8	9
1	1			

		↓		
5	6	7	8	9
1	1	2		

Ans = 1

5	6	7	8	9
1	1	2	2	

Ans = 1+1 = 2

5	6	7	8	9
1	1	2	2	3

3 - 2 = 1

Count of 1 = 2

Ans = 2 + 2 = 4
↳ final.

Example:

5	6	7	8	9
1	1	2	2	3

B = 3

Ans = 1

Edge case: B = 0

	0	1	2	3	4	5	6
Count	2						
odd :	0						

m[0] : ✗ 2 ans = 2-1 = 1

↑
initially

*

	1	4	6	7	8	10	11
odd	1	1	1	2	2	2	3

m[0] = 1 m[3] = 1 ans = ✗ ✗ ✗ 6

m[1] = ✗ ✗ 3
m[2] = ✗ 3

code:

```
unordered_map <int, int> m;
// total = Count of odd,  sum = No. of subarray
int total = 0;
int sum = 0;

m[0] = 1;

for (int i=0; i< A.size(); i++) {
    // Count No. of odd.
    if (A[i] % 2)
        total++;
    // Map ko increase kr do
    m[total]++;
    // B == 0
    if(B == 0)
        sum += (m[total - B] - 1);
    // sum me include karo.
    else if (m.count(total - B))
        sum += m[total - B];
}

return sum;
```

Problem 4: Equal 0, 1 and 2.

Example: 0 1 0 2 0 1 0

Brute force:

find total subarray and Count 0, 1, 2

if they are equal return count.

$O(N^2)$

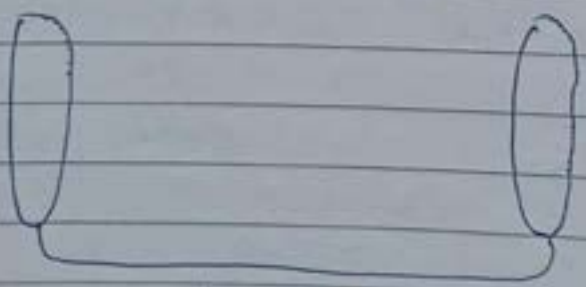
Another: (Approach)

	0	1	0	2	0	1	0
Count 0:	1	1	2	2	3	3	4
Count 1:	0	1	1	1	1	2	2
Count 2:	0	0	0	1	1	1	1

On this problem, all are positive so it is not possible that to find answer

So, we need some modification.

0 1 0 2 0 1 0



1 case

but it is always increasing so, not possible in this.

We want this equal to find the answer.

classmate
Date
Page

0 1 0 2 0 1 0

Count 0 - Count 1

Count 0 - Count 2

Count 1 - Count 2

Take only two

Count 0 - Count 1 = A

Count 0 - Count 2 = B

Count 1 - Count 2 = C

↳ generated by B - A

A - B = Count 2 - Count 1

B - A = Count 1 - Count 2

= C

	0	1	0	2	0	1	0
Count 0 :	1	1	2	2	3	3	4
Count 1 :	0	1	1	1	1	2	2
Count 2 :	0	0	0	1	1	1	1
Count 0 - Count 1 :	1	0	1	1	2	1	2
Count 0 - Count 2 :	1	1	2	1	2	2	3

How to Create in map

पहले आया था

Ans = 1

पहले आया था

Ans = 2

< int , int , Count >

Count 0 - Count 1

Count 0, - Count 2

done kitne baar आए थे

value | frequency

unordered_map <int, unordered_map <int, int>> m;

Count 0 - Count 1

Count 0 - Count 2

m[1][2] = 4

(Array)

classmate
Date
Page

m[3][2] = 5;

unordered_map <int, unordered_map <int, int>> m;

Code :

```
long long getSubStringWithEqual012 (String str) {

    unordered_map <int, unorderd_map <int, int>> m;

        m[0][0] = 1;

    int Count_0 = 0;
    int Count_1 = 0;
    int Count_2 = 0;

    long long sum = 0;

    int first, second;

    for (int i = 0; i < str.size(); i++) {
        if (str[i] == '0')
            Count_0++;

        else if (str[i] == '1')
            Count_1++;

        else
            Count_2++;

        first = Count_0 - Count_1;
```

classmate
Date
Page

```
        second = count_b - count_a;
        sum += m[first][second];
        m[first][second]++;
    }
    return sum;
}
```

Graph

Lecture - 57

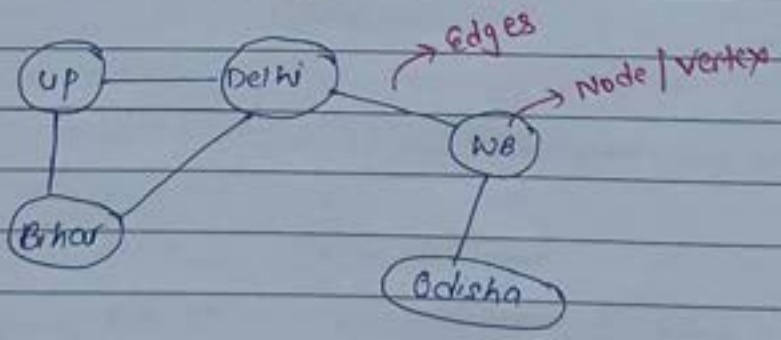
Till Now We Covered

1) Array
2) Linked list
3) Queue
4) stack
5) Tree
6) BST
7) BBST
8) Heap
9) Hashing

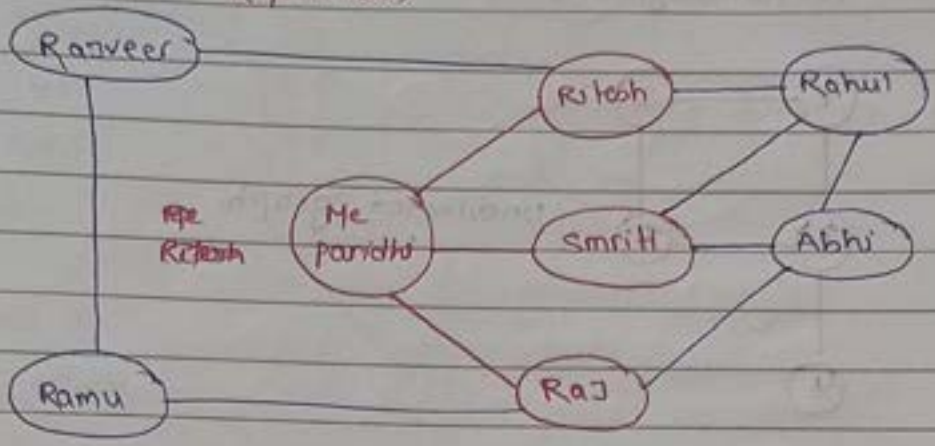
Now we start Graph.

Example : city state :

Graph : Node / vertex
→ Edges



* Social Media : (Friends)

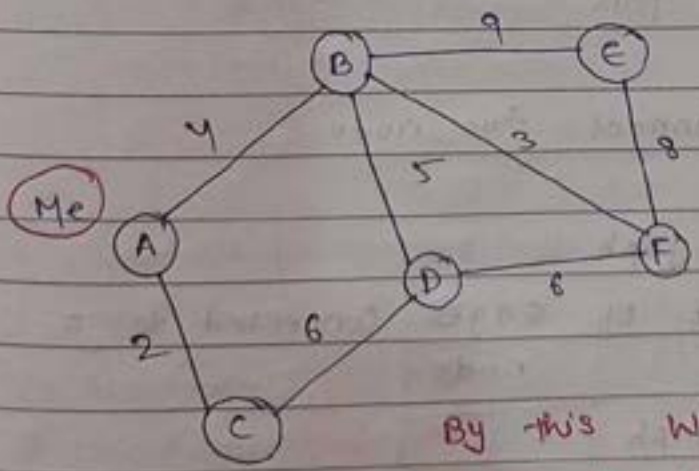


My Friend : Ritesh, Smriti, Raj
Smriti Friend : Rahul, Abhi
Ritesh Friend : Rajveer, Rahul.

on my LinkedIn, on my people you may know section :
I have suggestion of :
Rahul, Abhi, Rajveer.

This is done by Map.

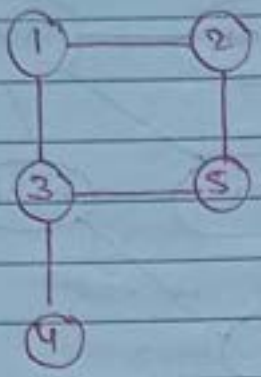
* Google Map :



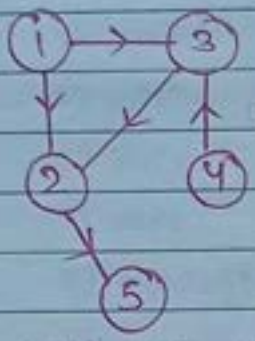
By this we can find shortest time to reach any place.
shortest distance.

classmate
Date
Page

* Terminologies :



undirected graph.



Directed graph.

→ Vertex : (1) (2) (3) (4) (5)

These are vertex / Node.

→ unweighted group : The weight are not given on path.

→ Weighted graph : The weight are mentioned on path.

→ Edge : Which Connect two node.

→ Degree : → undirected gaph

↳ No. of edges Connected to a node.

→ Directed graph

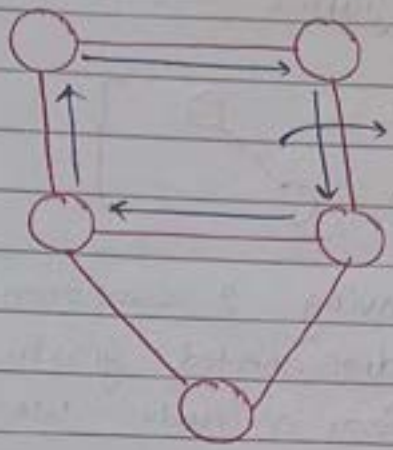
↳ In degree : No. of Edges Coming inward to the Node.

↳ out degree : No. of edges going Outward from the Node.

→ Cycle :

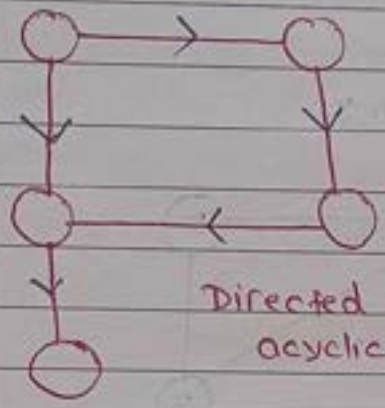
No edge or vertex should be repeat.

start = end
↳ for cycle.



This form a cycle.

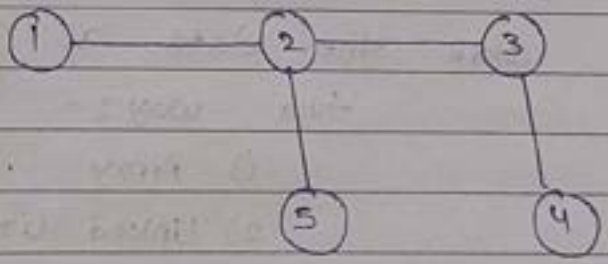
→ Path :



Directed acyclic graph.

→ path :

Node | edge Me dubare nahi jao.



1 → 2 → 3 → 4

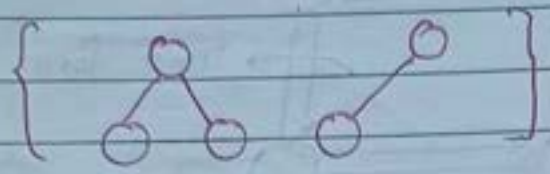
1 → 2 → 5

1 → 2 → 3

→ Connected graph :

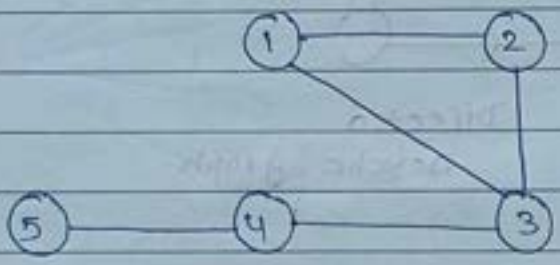
from one node you can visit any Node.

→ Disconnected graph :



graph having 2 component
This is disconnected graph.
because from a node we can't travel all other nodes.

* undirected graph :



store the data of graph in two way :-

1) Array
2) Linked list

1) Array 2D Array

	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	0	0
3	1	1	0	1	0
4	0	0	1	0	1
5	0	0	0	1	0

searching : O(1)
Delection : O(1)
Insertion : O(1)

$O(N^2)$

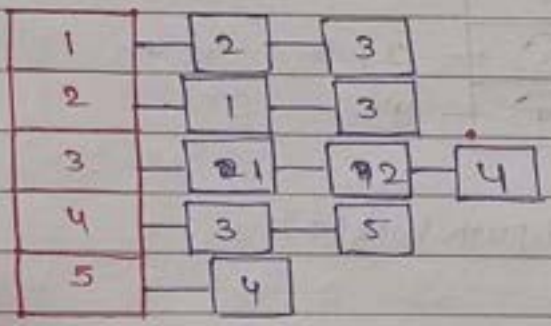
A[V][V] $O(V^2)$

Vertex = No. of vertex

Space Complexity : $O(V^2)$.

2) Another Way to store graph.

① → 2, 3 1 is Connected to 2 & 3.
② → 1, 3
③ → 2, 4
④ → 3, 5
⑤ → 4



Edge include → 2 times

Vertex / Node includ → 1 time.

$2 * E + V$

$O(2E + V)$

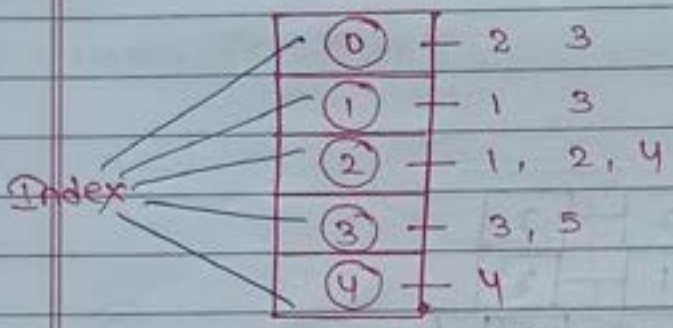
Searching : $O(V)$

Deletion : $O(V)$

Insertion : $O(1)$.

3) Another way to store graph: (Adjacency List)

Vector <int> arr[5];



arr[0].push_back(2);

* If you have Dense graph:
  ↳ use adjacency Matix.

Dense graph: Every node is Connected almost With every other node.
↳ nearly equal to.
If V x V = E
↳ use adjacency Matrix.

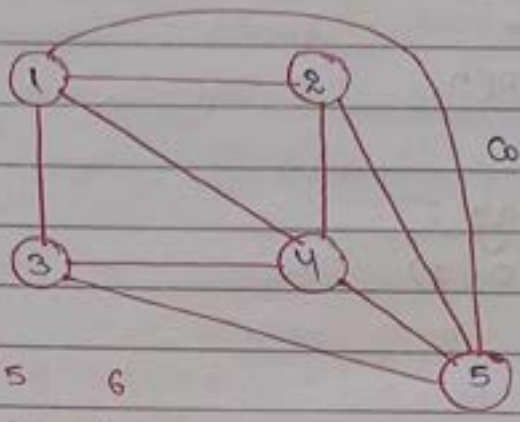
* If you have Sparse graph:
  ↳ use Vector to store.

Sparse graph: Every node is not Connected to other Maximum Node.

* Complete graph :

↳ All Node present in graph are connected with all other Nodes.

Ex :



Complete graph

use adjacency Matrix to store.

	1	2	3	4	5	6
1	0	1	1	1	1	1
2	1	0	1	1	1	1
3	1	1	0	1	1	1
4	1	1	1	0	1	1
5	1	1	1	1	0	1
6	1	1	1	1	1	0

$O(V^2)$

If we want to make vector for this graph.

↳ size is same in both.

$O((2(E)+V))$ ↳ searching is taking more time here.

So, for Complete graph Adjacency Matrix is good to store the graph.

$$\frac{n*(n+1)}{2} \text{ edges}$$

Total Node Compination : $n C_2$

$$\text{Total No. of Edges} = \frac{n!}{(n-2)\,2!}$$

classmate
Date
Page

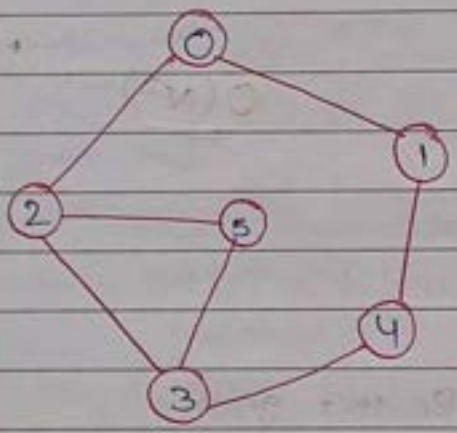
Adjacency Matrix Complete Graph Adjacency List

S.Comp : V^2 $E = V^2$ S.Comp : $2E + V$

Dense Graph sparse Graph

Searching : O(1) Searching : O(N)

*



Node = 6

Edges / Vertex = 7

A[v][v]

	0	1	2	3	4	5
0	0	1	1	0	0	0
1	1	0	0	0	1	0
2	1	0	0	1	0	1
3	0	0	1	0	1	1
4	0	1	0	1	0	0
5	0	0	1	1	0	0

Code :

```
int main() {

    int v, e;

    cin >> v >> e;
     int a, b;
     int A[v][v];

    for(int i=0; i<e; i++) {
        cin >> a >> b;

        A[a][b] = 1;
         A[b][a] = 1;

    }

    return 0;

}
```

classmate
Date
Page

* Adjacency list :

0	1, 2
1	0, 4
2	0, 3, 5
3	2, 4, 5
4	1, 3
5	2, 3

```
Vector <int> arr[v];
```

Code :

```
int main () {

    int v, e;
    cin >> v >> e;

    Vector <int> A[v];
    int a, b;

    for (int i=0; i<e; i++) {
        cin >> a >> b;

        A[a].push_back(b);
        A[b].push_back(a);

    }
return 0;
```

Example : Social Medion
(Face Book)

adjacency Matrix — adjacency list

What it use

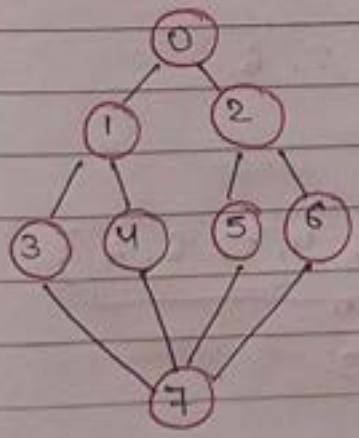
Total user : 1 Million

1 person know Average of 1000.

Is it sufficient to make a adjacency Matrix to use store data in FB?
↳ No

on FaceBook we use adjacency list.
because very few people can be known by a single user.

* Graph Traversal :



BFS : 0 1 2 3 4 5 6 7
(Breadth first search)

0 → 1, 2
1 → 0, 3, 4
2 → 0, 5, 6
3 → 1, 7
4 → 1, 7
5 → 2, 7
6 → 2, 7
7 → 3, 4, 5, 6

We use Queue here:

0

↑ point it

ans : {0

1, 2

ans : {0, 1

2, 3, 4

ans : {0, 1, 2

3, 4, 5, 6

ans : {0, 1, 2, 3,

4, 5, 6, 7

ans : {0, 1, 2, 3, 4

5, 6, 7

ans : {0, 1, 2, 3, 4, 5

6, 7

ans : {0, 1, 2, 3, 4, 5, 6

7

ans : {0, 1, 2, 3, 4, 5, 6, 7}

BFS.

We have take care of which node we have visited, if we visited a node then we cannot visit again to that node.

We use a visited list to track which node we have visit ?

If the limit of Node,

Let V nodes are there

and values are from 0 to V-1

↳ then we use here list.

If value of Nodes are not fix.

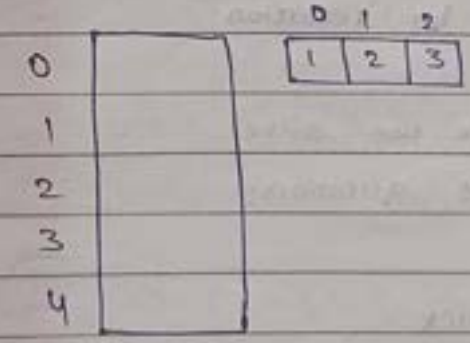
↳ then we use map.

* If my pen is lost on floor having tiles and there is dark,

then we search first 1st tiles in all side if Not found.

We search which having distance 2 and so on.

* problem : BFS ~~traversal~~ of Graph.



~~adj[0][0] = 1~~ adj[0][0] = 1

~~adj[0][2] = 3~~ adj[0][2] = 3

Code :

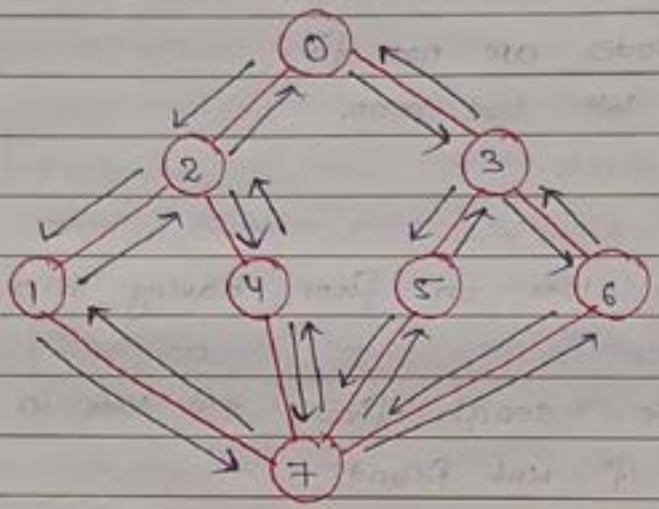
```
vector<int> ans;
bool visited[V] = {0};
queue<int> q;
q.push(0);
visited[0] = 1;
int node;
while (!q.empty()) {
    node = q.front();
    q.pop();
    ans.push_back(node);
    for (int i=0; i<adj[node].size(); i++) {
        if (!visited[adj[node][i]]) {
            q.push(adj[node][i]);
```

```
            visited[adj][Node][i] = 1;
        }
    }
}
return ans;
```

Tree is alway open.

Graph is may be open or closed.

* DFS : (Depth first search):



DFS : {0, 2, 1, 7, 4, 3, 5, 6}

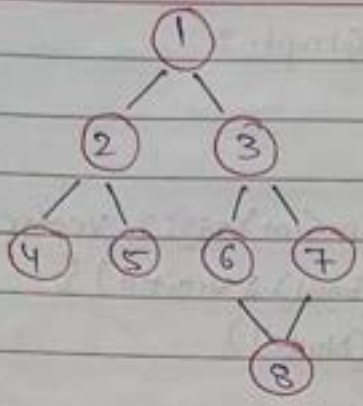
	0	1	2	3	4	5	6	7
Visited	0	0	0	0	0	0	0	0

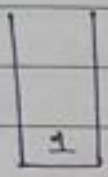
We can solve this by recursion

By stack we solve this question:

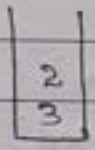
DFS : Stack

BFS : Queue.

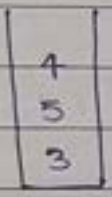




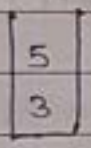
ans = {1,



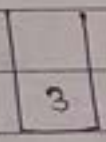
ans = {1, 2,



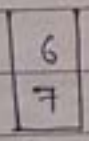
ans = {1, 2, 4,



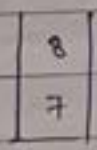
ans = {1, 2, 4, 5,



ans = {1, 2, 4, 5, 3



ans = {1, 2, 4, 5, 3, 6



ans = {1, 2, 4, 5, 3, 6, 8

ans = {1, 2, 4, 5, 3, 6, 8, 7}

problem 2 : DFS of Graph :

```
Void DFS (int Node, vector <int> adj[], vector<int> &ans,
                vector <bool> &visited) {
        if (visited[Node])
            return;

        visited[Node] = 1;
        ans.push_back (Node);

    for (int i= 0; i< adj[Node].size() ; i++ ) {
        DFS (adj[Node][i], adj , ans, visited );
    }
}

vector <int> dfsOfGraph (int V, vector <int> adj[]) {

        vector <bool> visited (V, 0);
        vector <int> ans;

        DFS ( 0, adj, ans, visited );

    return ans;

}
```

Lecture 58

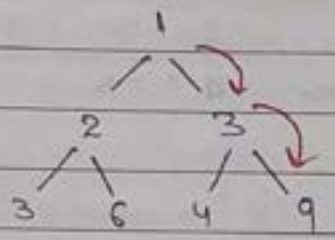
Graph

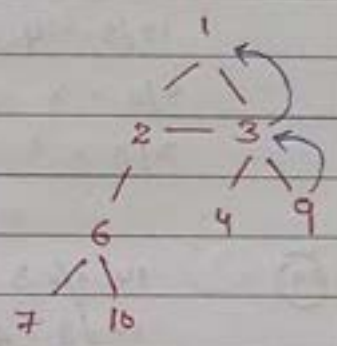
* problem 1: Shortest path from 1 to n.

$J = i + 1$

$J = 3 * i$

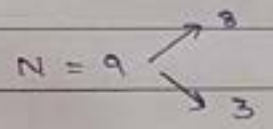
N = 9



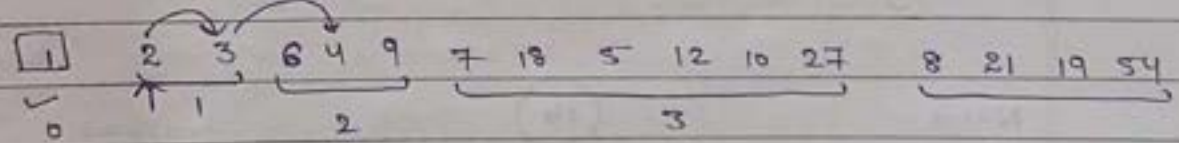


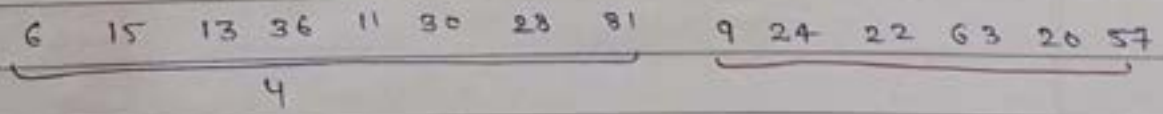
{Edge = 2}

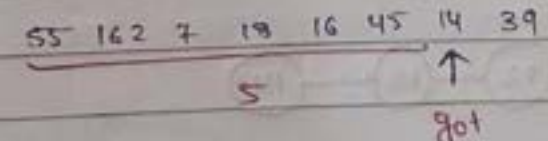
1 2 3
{1}, {2, 3}, {3, 6, 54, 9}, {- - - -}



* N = 14







{1 ← 3 ← 4 ← 12 ← 13 ← 14}

for reverse :

i+1 is Converted to i-1

i*3 is Converted to i%3

(14) → 14/3 → X then 14-1 = 13 Count = 1

13/3 → X then 13-1 = 12 Count = 2

12/3 = 4 Count = 3

4/3 = X then 4-1 = 3 Count = 4

3/3 = 1 ✓ Count = 5

(14) 14 %3 = 2 Count = 2

14/3 = 4 Count = 3

4%3 = 1 Count = 4

4/3 = 1 Count = 5

```
CountEdges = 0
{ CountEdges += N%3;
  N/=3;
  CountEdges++;
}
```

N=14 (14)

CountEdge = 0

14%3 = 2

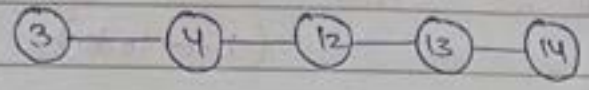
CountEdge = 2 (12) — (13) — (14)

14/3 = 4

CountEdge = 3 (4) — (12) — (13) — (14)

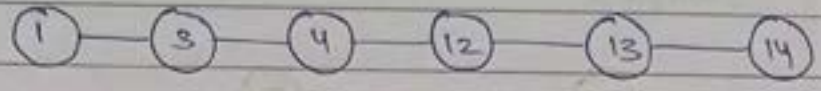
4 % 3 = 1

CountEdge = 4



4 / 3 = 1

CountEdge = 5



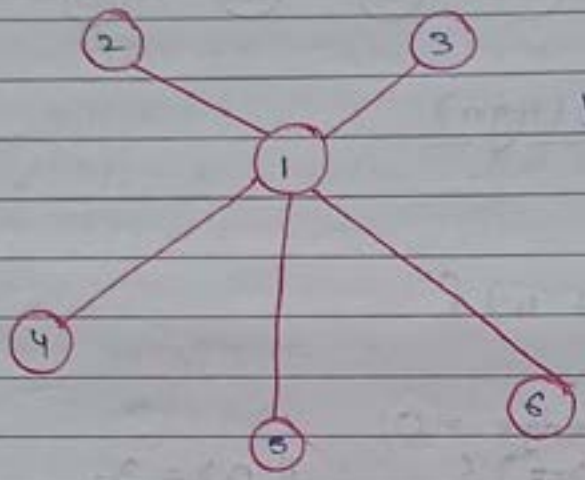
T.C : O(log n)

Code:

```
int minimumStep( int n ){

    int Count_edge = 0;
    while ( n >= 3 ) {          || n >= 3.
        Count_edge += n % 3;
        n /= 3;
        Count_edge++;
    };

    Count_edge += n;
    Count_edge--;

    return Count_edge;

}
```

* Problem 8: Find center of star Graph. (Leetcode)

Example:



We have to find center of start: 1.

→ undirected graph
→ n nodes (1 to n)
→ 1 centre node & n-1 edges connected to centre

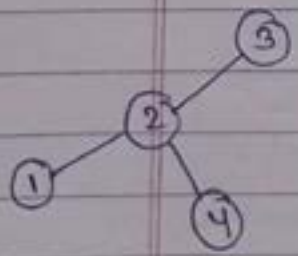
Given:

↳ 2D Array (Edges)
edges[i] = $[u_i, v_i]$
↳ indicates an edge b/w nodes u_i & v_i.

Return center.

Example 1:

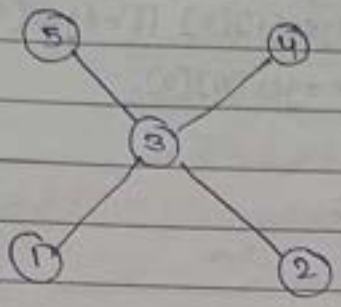
edges: $[[1,2],[2,3],[4,2]]$



find Common in any two and Common is the answer.

Example 2:

[1, 3] [3, 2] [4, 3] [3, 5]



find two edges

[1, 3] [3, 2]

find Common

↳ 3 → return.

Time Complexity : O(1)

[1, 3], [3, 2] [4, 3] [3, 5]

E[0][0] E[0][1] E[1][0] E[1][1]

if (E[0][0] == E[1][0] || E[0][0] == E[1][1])

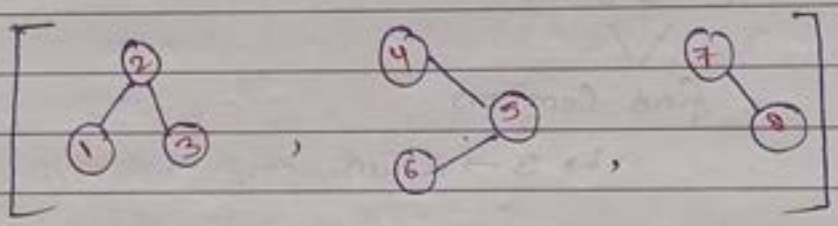
return E[0][0]

return E[0][1]

Code:

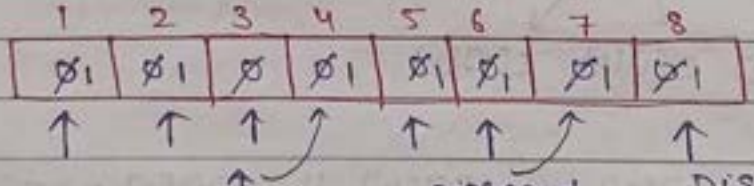
```
public int findCenter (int [][] edges) {
    if (edges[0][0]== edges[1][0] || edges[0][0] == edges[1][1])
            return edges[0][0];

    return edges[0][1];

}
```

* problem 3: Number of provinces :



3 provinces

return 3:



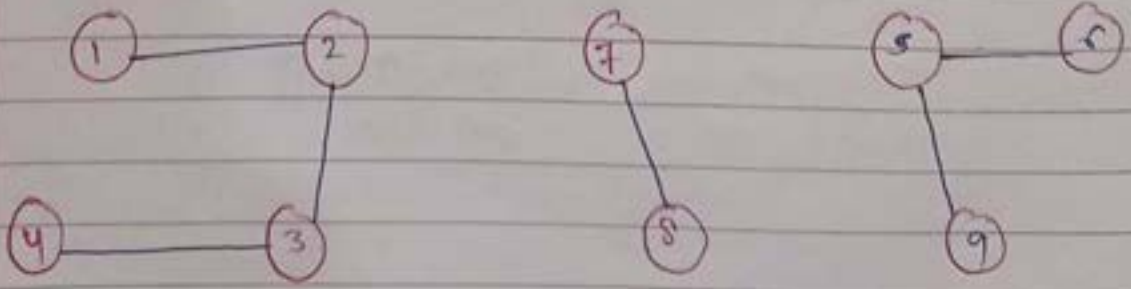
provices = 1

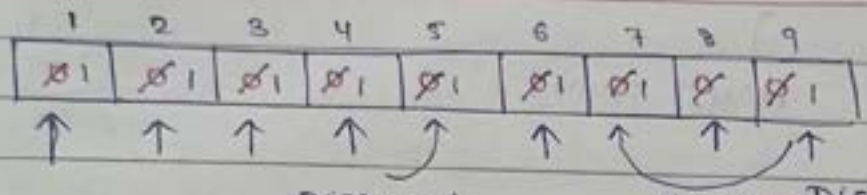
provices = 2

provinces = 3

↳ return.

* check cities are connected or not, if not how many breaks are there.





return Count;

code :

```
Void DFS (vector<vector<int>>& adj, Vector<bool>& visit,
                                    int node) {
    if (visit[node])
        return;

    visit[node] = 1;

    for (int i=0; i< adj[node].size(); i++) {
        if (adj[Node][i])
            DFS (adj, visit, i);
    }

    return;
}

int numprovinces (vector<vector<int>> adj, int v) {
    int count = 0;
    Vector<bool> visit (v, 0);

    for (int i=0; i< v; i++) {
        if (!visit[i]) {
            count++;
            DFS (adj, visit, i);
        }
```

```
        }
        return count;
    }
}
```

* Problem 4 : Two Teams ○ (interview bit)

A people
↳ numbered from 1 to A.

Make two team
↳ Not necessary, size is not same.

2D-Array → B
↳ size M x 2

B[i][0] & B[i][1]
↳
Dushman
not be in 1 team

Example :

A = 4

B = 1 4

Team 1	Team 2
1	4
	3

3 1
4 3
2 1

(1,4) साथ नहीं

पहले से चले गए

↳ return 0
Not possible.

Example 2:

A = 5

B = 1, 2
2, 3
1, 5
2, 4

Team 1	Team 2
1	2
3	
	5
4	

All fix

return 1.

Edge cases :

		Team 1	Team 2
	1 2 →	1	2
	2 3 →	3	
Now (random)	5 8 →	5	8
	9 5 →		9
	9 8		

↳ एक team से चला गया

but we have option.

↳ If we put 8 in team 1 and 5 in team 2, when we put randomly,

then team is distributed.

Randomly distribute करने से output wrong हो सकता है।

Map → create problem.

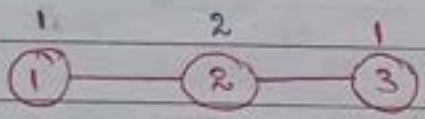
We can use Graph here.

1	2
2	3
5	8
6	4
5	6

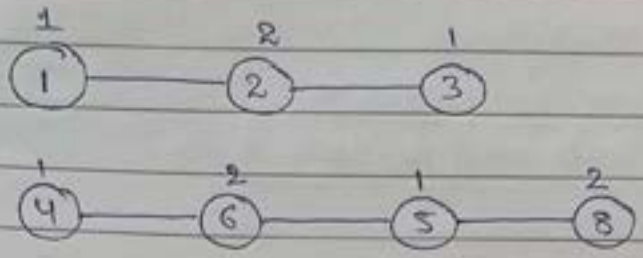
By Map:

	Team 1	Team 2
	1	2
	3	
But if we put	5	8
6 in team 2 and	6	4
4 in team 1,	Not be in	
then true.	1 team	

By Graph:

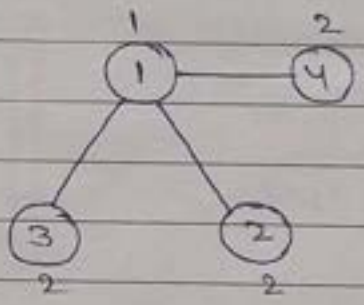


player		Dushman.
1	→	2
2	→	1, 3
3	→	2,
4	→	6
5	→	8, 6
6	→	4, 5
8	→	5



Example:

1	4
3	1
4	3
2	1



1 → 4, 3, 2

2 → 1

3 → 1, 4

4 → 1, 3

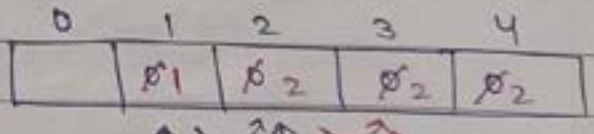
Not possible return 0.

* 1 4 4 player

3 1

4 3 Make an array

2 1



1 → 2, 3, 4

↑ ↑

Team 1 team 2

Start from here (because name of player start from 1)

Enemy 1 (different)

Enemy 1, 4 (4 same team) return 0.

2 → 1

3 → 1, 4

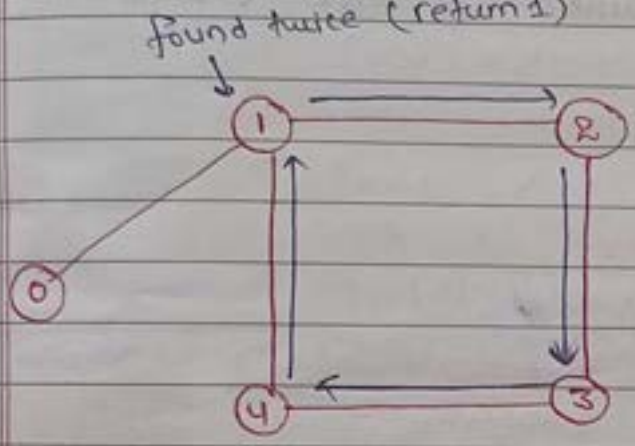
classmate
Date
Page

* problem 5 : Detect cycle in an undirected graph :

Check cycle is present or not

V = 5 E = 5

adj = {{1}, {0, 2, 4}, {1, 3}, {2, 4}, {1, 3}}.

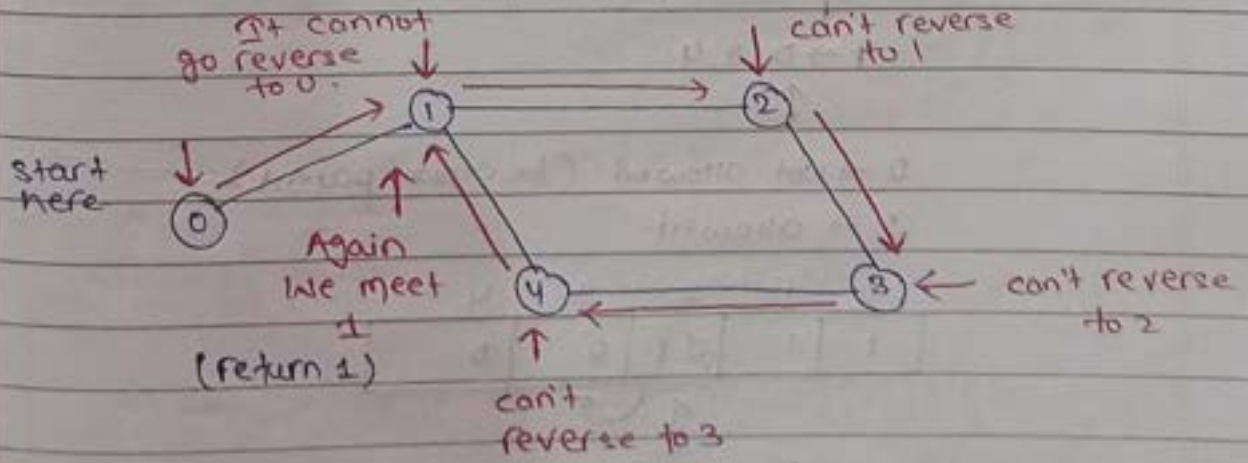


एक Node को दुबारा मिल जाए तो cyclic graph.

→ But Backward नहीं जाना है।

Example:

0 → 1
1 → 0, 2, 4
2 → 1, 3
3 → 2, 4
4 → 1 3



Track of visitor :

0	1	2	3	4
✗1	✗1	✗1	✗1	✗1

Koi bhi Node Apne parent pe nahi jayega !

Visited : to check where we have visited

0 → 1
1 → 0, 2, 4
2 → 1, 3
3 → 2, 4
4 → 1, 3

0	1	2	3	4
✗1	0	0	0	0

0 → 1

0	1	2	3	4
1	✗1	0	0	0

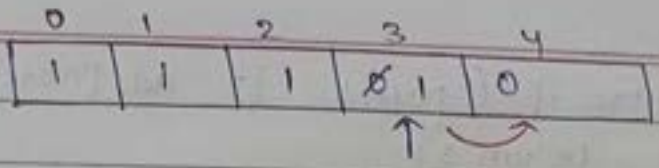
1 → 0, 2, 4

0 → Not allowed (because parent).
2 → allowed.

0	1	2	3	4
1	1	✗1	0	0

2 → 1, 3

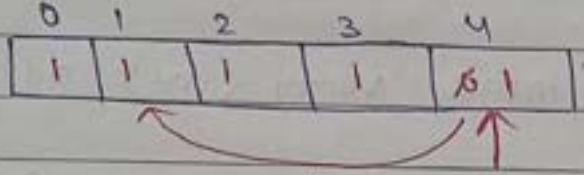
1 → Not allowerd (parent)
3 → allowed



3 → 2, 4

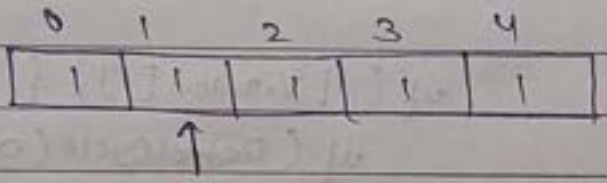
2 → Not allowed (parent)

4 → allowed



4 → 1, 3

1 → allowed



Already visited

So, it create cycle

return 1.

code :

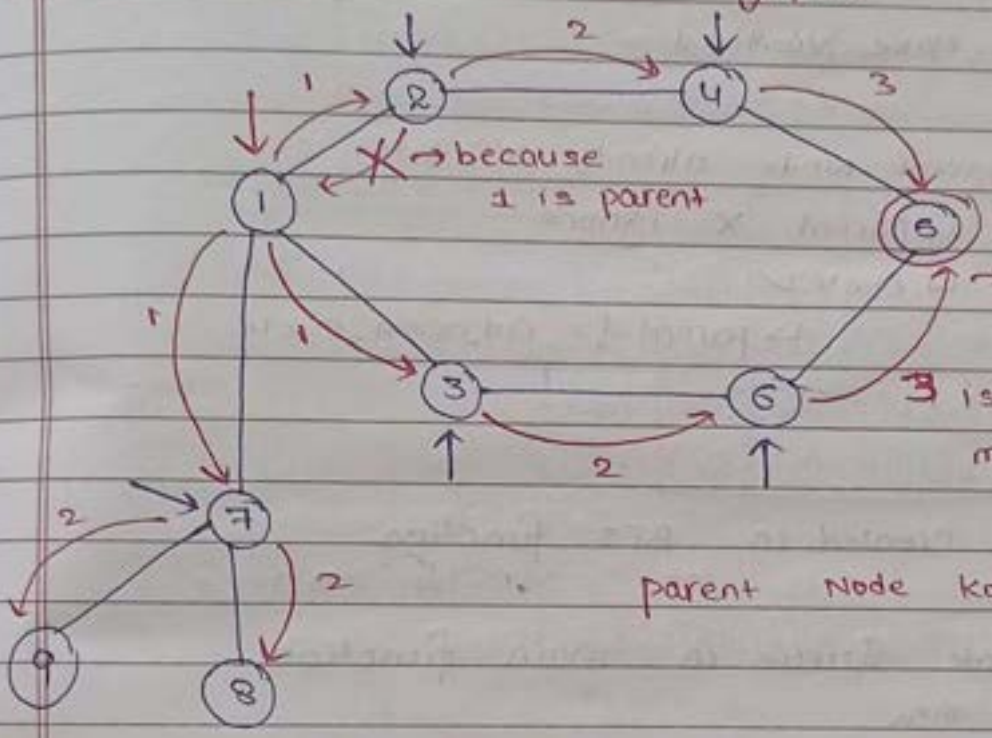
```
bool DetectCycle (vector <int> adj[], int node, int parent,
                  vector <bool> & visited ) {

    visited[node] = 1;

    for ( int i = 0;  i < adj[node].size();  i++ ) {

        if ( !visited [adj[node][i]]) {
            if (DetectCycle (adj, adj[node][i], node,
                             visited));
            return 1;
        }
```

```
            else if (parent != adj[node][i])
                return 1;
        }
    return 0;
}

bool isCycle (int V, vector<int> adj[]){

        vector<bool> visited (V, 0);

        for(int i= 0; i<V; i++) {

            if( !visited[i]) {
                if ( DetectCycle(adj, i, -1, visited))
                    return 1;
            }
        }
    return 0;
}
```

Lecture 59

Detect Cycles in Graph

* Detect cycle in a undirected graph :

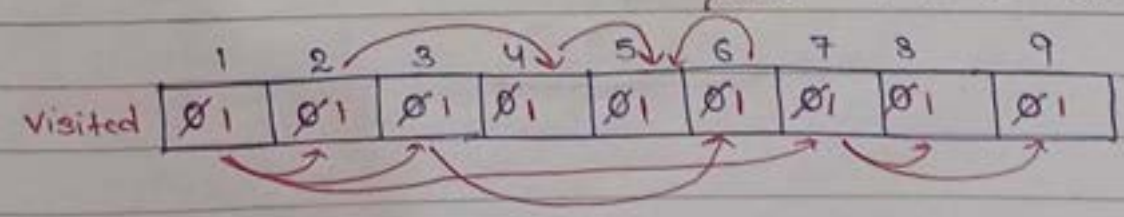


parent Node ko ignore kyu karega ?

↳ Kyuki Wo pice Jayega to Node to visited Mil gya to cycle ho Jayega, lekin real Me Vo cycle nahi hoga.

① Visit kar liye hai (data)

② Ignore parent



Two condition for 1 :

① parent Node (ignore)

② cycle ✓✓

If Adjacent Node Not Visited

Make Visit = 1

else Adjacent Node already visited

↳ parent ✗ ignore

↳ cycle ✓

↳ parent != adjacent_cycle

Queue is created in BFS function.

If we create Queue in main function

↳ then

queue<1, -1>

↳ Bahut baar main function me create karne pe garbage value aa jati hai, or wo garbage value flush nahi ho pati hai.

Code:

```
bool BFS (int node, vector <int> adj[], vector <bool> & visit ) {
        queue <pair < int, int >> q;
        q.push ( {node, -1});
        visit[node] = 1;
        while (!q.empty()) {
          int child  = q.front().first;
          int parent = q.front().second.
          q.pop();

        for(int i=0; i<adj[child].size(); i++) {
            if (visit[adj[child][i]] == 0) {
                visit[adj[child][i]] = 1;
                q.push({adj[child][i], child});
              }
            else {
                if (adj[child][i] != parent)
                  return 1;
          }
        }
      }
      return 0;
  }
bool  isCycle (int V, vector <int> adj[]) {
        vector <bool> visited(V, 0);
        for (int i=0;  i<V; i++){
            if (!visited[i]) {
                bool ans = BFS (i, adj, visited);
                if (ans == 1)
                  return 1;
              }
        }
      return 0;
}
```

Problem 2 : Rotten Oranges

Given → Grid of dimension : n x m

0 → cell empty
1 → fresh orange
2 → karab orange

0	1	1	0	2
2	1	2	2	0
1	0	2	2	0
0	0	1	0	1
1	2	1	1	0

karab orange apne ass pass ke orange ko 1 unit time me karab kar deya.

Example :

0	(2)	1
1	0	1
1	1	(2)

After 1 unit time :

0	(2)	(1)
1	0	(1)
1	(1)	(2)

After 2 unit time :

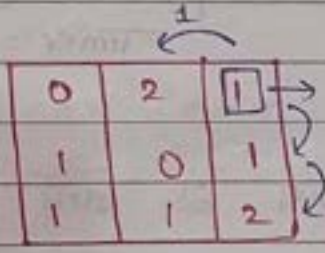
0	2	2
1	0	2
1	2	2

After 3 unit time :

0	2	2
1	0	2
2	2	2

After 3 unit time all orange rotten.

We use BFS here.



kitna der me karab hoga.

Min (1, 2)

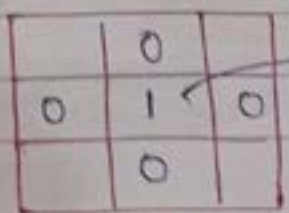
↳ (1)

Min (2)

↳ (2)

* If any orange left without rotten

↳ return -1

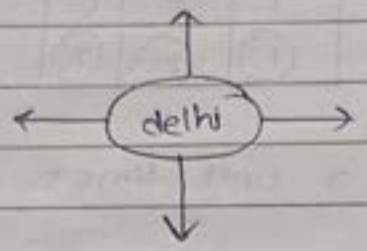


This orange never get rotten.

Example 8 Covid :

↓

first Come in delhi



* Approach

	0	1	2
0	0	2	1
1	1	0	1
2	1	1	2

Rotten row col timer

↳ (0,1) — 0

↳ (2,2) — 0

Good oranges = 5

Timer = 0

एक बार 2 का काम हो जाए तो उसको 0 बना दो।

↳ isse visited vector Alag से नहीं बनाना पड़ेगा।

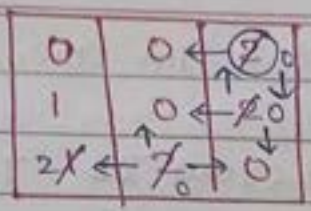
	0	1	2
0	0	2	2
1	1	0	2
2	1	2	2

Timer = 1

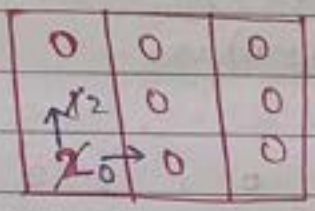
(0,2) → 1

(1,2) → 1

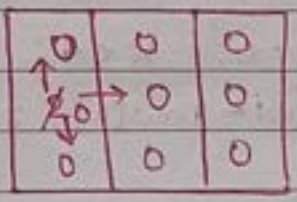
(2,1) → 1



(0,2) ✗
(1,2) ✗
(2,1)
↳ (2,0) – 2



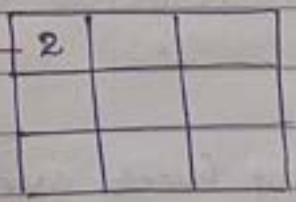
(2,0)
↳ (1,0) – 3



Total time = 3

Ye bhi check karha hai ki kisi location ki access है या नहीं।

Ye अगर upar से check करेगा तो error deya.



isliye ये पहले check karega ki memory access hai ya nhi, iska matlab ki wo grid ke andar present है या नहीं।
↳ done by check function.

bool check (int i, int j, int row, int col)

return i > -1 && i < row && j > -1 && j < col;

→ up : (i - 1, j) -1 0
down : (i + 1, j) 1 0
Left : (i, j - 1) 0 -1
Right (i, j + 1) 0 0 1

0 1 2 3
int row[4] = {-1, 1, 0, 0}

int col[4] = {0, 0, -1, -1}

for (int k = 0; k < 4; k++)
(i + row[k], j + col[k])

Code:

```
bool check (int i, int j, int row, int col) {
    return i > -1 && i < row && j > -1 && j < col;
}

int orangesRotting (vector<vector<int>>& grid) {

    int n = grid.size();
    int m = grid[0].size();

    int goodOranges = 0;
```

```
queue < pair<pair<int, int>, int>> q;

for (int i=0; i<n; i++){
    for (int j=0; j<m; j++){

        if (grid[i][j]==1)
            goodoranges ++;

        else if (grid[i][j]==2)
            q.push({{i, j}, 0});

    }
}

int i, j, timer = 0;
while (!q.empty()){
    i = q.front().first.first;
    j = q.front().first.second;
    timer = q.front().second;
    q.pop();

    if (check(i-1, j, n, m) && grid[i-1][j]==1){
        grid[i-1][j] = 0;
        goodoranges--;
        q.push({{i-1, j}, timer+1});
    }

    if (check(i+1, j, n, m) && grid[i+1][j]==1){
        grid[i+1][j] = 0;
        goodoranges--;
        q.push({{i+1, j}, timer+1});
    }
```

```
if (check(i, j-1, n, m) && grid[i][j-1] == 1){
        grid[i][j-1] = 0;
        goodOranges--;
        q.push({{i, j-1}, timer+1});
}

if(check(i, j+1, n, m) && grid[i][j+1] == 1){
        grid[i][j+1] = 0;
        goodOranges--;
        q.push({{i, j+1}, timer+1});

}

    if(goodOranges)
        return -1;
    else
        return timer;
}
```

or

We will change some code for optimal :

```
int row[4] = {-1, 1, 0, 0};
int col[4] = {0, 0, -1, 1};

for(int k=0; k<4; k++)
    if(check(i+row[k], j+col[k], n, m) &&
            grid[i+row[k]][j+col[k]] == 1){
        grid[i+row[k]][j+col[k]] = 0;
        goodOranges--;
        q.push({{i+row[k], j+col[k]}, timer+1});
    }
```

problem 3: Distance of nearest cell having 1.

Minimum Distance
↳ BFS

Binary grid : n x m

distance of nearest 1 from each cell.

0	0	1	1
0	0	0	1
1	1	0	0
1	0	0	1

distance is calculated from

$$|i_1 - i_2| + |j_1 - j_2|$$

Jaha par bhi 1 है, उस जगह पर distance 0 होगा।.

Brute force :

1) pick one element
2) Traverse the array
3) If one found, find the distance and store a minimum length and store in a new 2D array at some index.

Time Complexity : $O((n \times m) * (n * m))$

$= O(n^2 * m^2)$

0 →	0	1	1
0 ↓	0	0	1
1	1	0	0
1	0	0	1

We start from 1

↳ 1 वाला सारा zero

↳ 1 के बगल वाला सारा zero से 1 distance.

0 (change in answer)

0	0 ←	1 →	1 → 1 (don't do anything)
0	0	0 ↓	1
1	1	0	0
1	0	0	1

0	$\emptyset_1$	0	0
$\emptyset_1$	$\emptyset_1$	$\emptyset_1$	0
0	0	$\emptyset_1$	$\emptyset_1$
0	$1\emptyset$	$\emptyset_1$	0

Ans

↓

0	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

$\emptyset_2$	1	0	0
1	1	1	0
0	0	1	1
0	1	1	0

Example 2:

	0	1	2
0	1	0	0
1	0	0	0
2	0	0	1

Answer:

0	$\emptyset_1$	$\emptyset_2$
$\emptyset_1$	$\emptyset_2$	$\emptyset_1$
$\emptyset_2$	$\emptyset_1$	$0\emptyset_2$

classmate
Date
Page

stack : {(0,0) (2,2) (0,1),(1,0), (1,2), (2,1)

Ans :

0	1	2
1	2	1
2	1	0

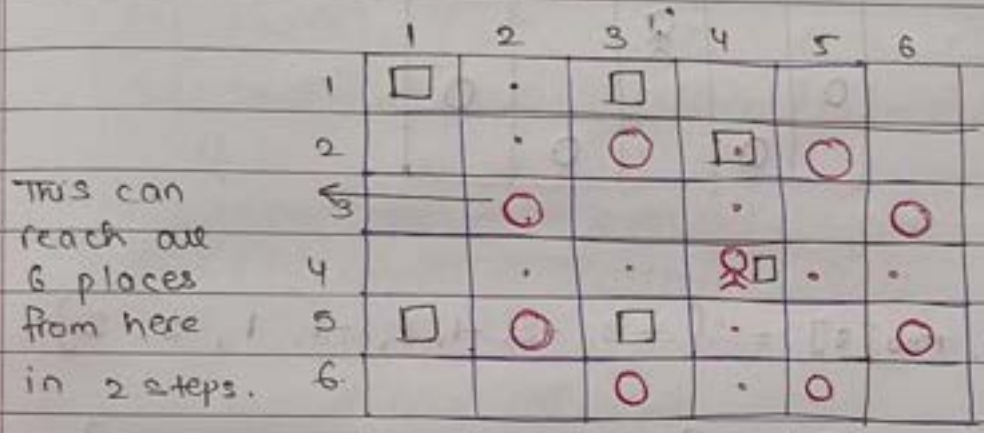
classmate
Date
Page

Lecture 60

BFS in Advance

problem : steps by Knight
↳ घोड़ा in chess
↳ 2 & half step.

Minimum time (BFS)



This can reach all 6 places from here in 2 steps.

2.5 steps.

In one chance, he can visit any of the 8 places. ○

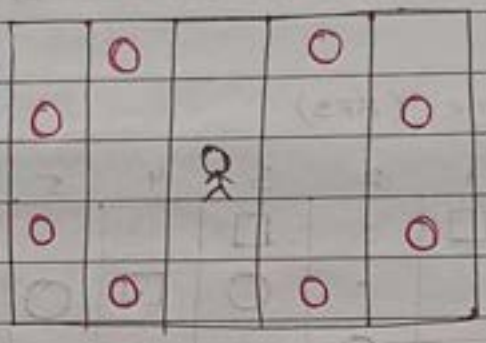
If I at (4, 4)
↳ I go to 8 places in 1 step.

stored is stack { (2,3), (2,5), (3,2), (3,6), (5,2), (5,6), (6,3), (6,5) }

Again we check for (2,3)
↳ { (1,1), (1,5), ~~(0,4)~~, (4,2), (3,1), (3,5), }
↓
Invalid because visited

Queue <row, col, step>

<2, 3, 1>
<5, 2, 1>
<1, 1, 2>
↳ got the answer.
return 2.



```
int row[8] = {-2, -2, -1, 1, -1, 1, 2, 2}

int col[8] = {-1, 1, 2, 2, -2, -2, -1, 1}

bool check (int i, int j, int n) {
    return i>0 && i<n && j>0 && j<n;
}
```

Code:

```
bool check (int i, int j, int n) {
        return i>=0 && i<=n && j>0 && j<=n;
}

int minStepToReachTarget (vector<int>& knightpos, vector<int>& Targetpos,
                          int N)
{

    int row[8] = {2, 2, -2, -2, 1, 1, -1, -1};
    int col[8] = {1, -1, 1, -1, 2, -2, 2, -2};
      // row, col, step
    queue<pair<pair<int, int>, int>> q;
    q.push ({{knightpos[0], knightpos[1]}, 0});
    // Jis position me ja chuke hai
    // Waha dubara na Jaye.
    vector<vector<bool>> visit (N+1, vector<bool>(N+1, 0));

    visit[knightpos[0]][knightpos[1]] = 1;

    int i, j, step;

    while ( !q.empty()) {
        i = q.front().first.first;
        j = q.front().first.second;

        step = q.front().second;

        if ( i == Targetpos[0] && j == Targetpos[1])
            return step;
```

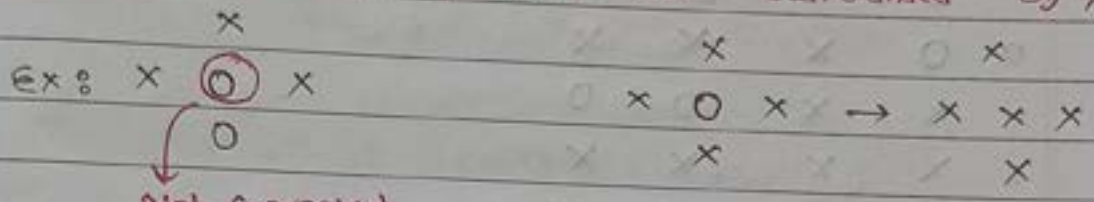
```
        step = q.front().second;
        q.pop();

        for(int k=0; k<8; k++) {

            if (check(i+row[k], j+col[k], N) &&
                    !visit[i+row[k]][j+col[k]]) {

                visit[i+row[k]][j+col[k]] = 1;

                q.push({{i+row[k], j+col[k]}, step+1});

            }

        }

    }
    return -1;

}
```

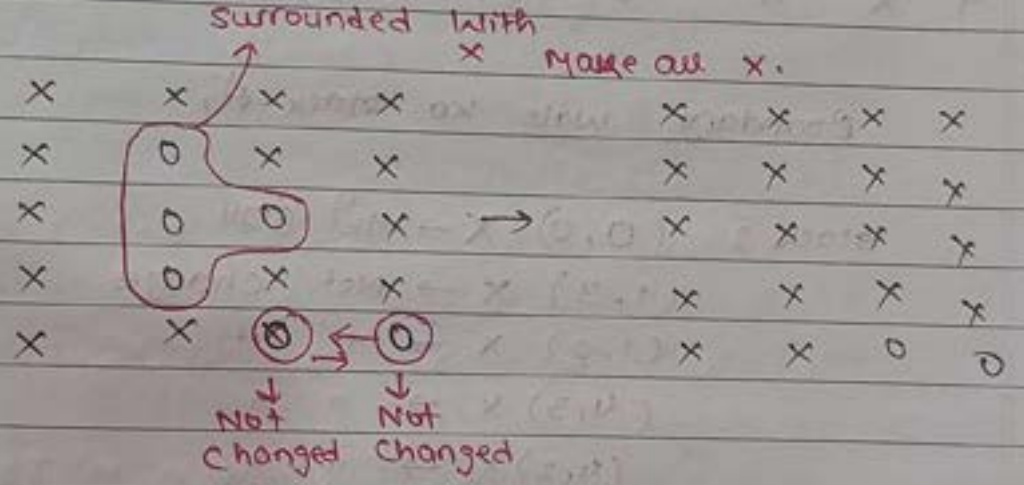
problem : Replace O's with X's

Matrix mat
↳ size N x M
↳ Every element is either O or X.

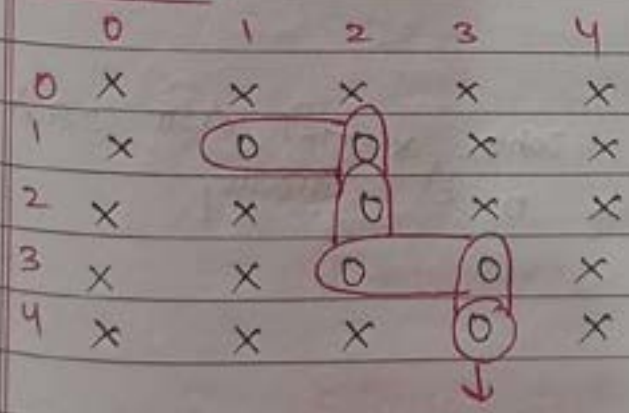
Replace all O with X that are surrounded by X.

Ex :





Example 2 :



start with boundary.

stack → (4,3) X
(3,3) X
(3,2) X
(2,2) X
(1,2) X
(1,1) X

No any O changed to X.

If O or Group of O touch the boundary, then it cannot be Converted into X.

Baaki Jitna O bacha hai sab ko X बना देंगे।

	0	1	2	3
0	O	X	X	X
1	X	X	O	O
2	X	X	X	X
3	O	X	X	X
4	X	X	O	O

Boundary Wale ko stack में,

stack : (0,0) X → नहीं होगा
(1,3) X → Not change
(1,2) X → " "
(4,3) X → " "
(4,2) X → " "
(3,0) X → " "

	0	1	2	3
0	XO	X	X	X
1	X	X	XO	XO
2	X	X	X	X
3	XO	X	X	X
4	X	X	XO	XO

Jaha X नहीं होगा उसको O हो जाएगा।

Boundary Wale se start hoga.

Start with boundary
↳ if 0, store in stack
↳ if X, neglect
↳ Apply BFS, if '0' found

boundary visit ke baad
↳ stack / queue से निकालते जाएंगे और four side dekhte jayenge.

Code:

```
bool check(int i, int j, int n, int m) {
    return i>=0 && i<n && j>=0 && j<m;
}

vector<vector<char>>fill(int n, int m, vector<vector<char>>mat) {
    int row[4] = {1, -1, 0, 0};
    int col[4] = {0, 0, 1, -1};
  queue<pair<int, int>> q;
 vector<vector<char>> ans(n, vector<char>(m, 'X'));
 vector<vector<bool>>visit(n, vector<bool>(m, false));
    for(int j=0; j<m; j++){
        if(mat[0][j]=='0'){
            q.push({0, j});
            ans[0][j] = '0';
            visit[0][j] = true;
        }
    }
    for(int j=0; j<m; j++){
        if(mat[n-1][j]=='0'){
            q.push({n-1, j});
            ans[n-1][j] = '0';
            visit[n-1][j] = true;
        }
    }
    for(int i=1; i<n-1; i++){
        if(mat[i][m-1]=='0'){
            q.push({i, m-1});
```

```
            ans[i][m-1] = '0';
            visit[i][m-1] = true;
        }
    }
    for(int i=1; i<n-1; i++){
        if(mat[i][0] == '0'){
            q.push({i, 0});
            ans[i][0] = '0';
            visit[i][0] = true;
        }
    }
    int u, v;
    while(!q.empty()){
        u = q.front().first;
        v = q.front().second;
        q.pop();

        for(int k=0; k<4; k++){
            int newU = u + row[k];
            int newV = v + col[k];

            if(check(newU, newV, n, m) &&
                    !visit[newU][newV]){
                visit[newU][newV] = true;
                if(mat[newU][newV] == '0'){
                    ans[newU][newV] = '0';
                    q.push({newU, newV});
                }
            }
        }
    }
    return ans;
}
```

problem : Find the number of islands

Given a grid size : m x m

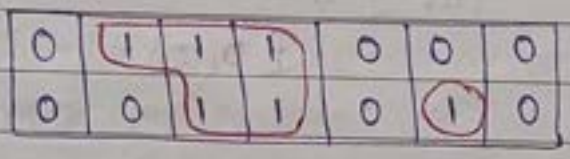
Contains 0's & 1's

0's → Water

1's → Land

An island is either surrounded by water or boundary of grid. It make in all 8 direction.

Example

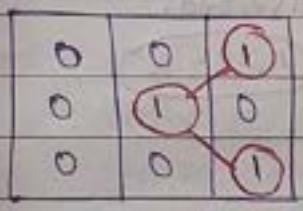


0 → water

1 → Island.

Count = 1

Count = 2.

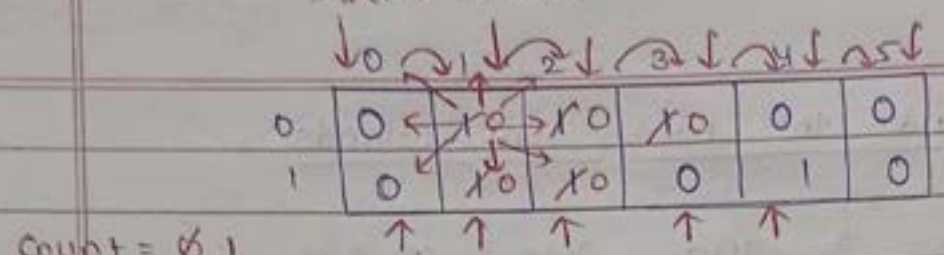


Count = 1

Solving By BFS.

```
int row[8] = {-1, -1, -1, 0, 0, 1, 1, 1}
int col[8] = {0, -1, 1, -1, 1, -1, 0, 1}
```

= 1(No) = 1(yes)



Count = ✗ 1

queue = {0, 1} → index of 1

1 visit ho gya usko 0 bna do.

queue = {0, 2}
{1, 1}
{1, 2}

{0, 1} deleted

{0, 3}

Equal to 1

Count = 2 {1, 4}

All traversed.

Ans = 2

→ traverse
↳ if 1 found
↳ store the index (row, col) in queue
↳ Make 0 ↳ Count ++
↳ check in 8 direction
↳ stack empty

↳ check all element of array
↳ repeat same steps.

Code :

```
bool check (int i, int j, int row, int col)
    {
        return i > -1 && i < row && j > -1 && j < col;
    }

Void BFS ( vector <vector <char>> &grid, int i, int j) {

        int row[8] = { 1, 1, 1, -1, -1, -1, 0, 0};
        int col [8] = {-1, 0, 1, -1, 0, 1, -1, 1 };

        int n = grid.size();
        int m = grid[0].size();

      queue <pair<int , int>> q;

       q.push ({i, j});

      while ( ! q.empty()) {
            i = q.front().first ;
            j = q.front(). second;
            q.pop();

        for (int k= 0; k < 8; k++ ) {
            if ( check (i+row[k], j+col[k], n, m) &&
                    grid [i + row[k]][j+col[k]] == '1' ) {

                grid [i+ row[k]][j+col[k]] = '0';
                q.push ( {i+row[k], j+col[k]});
            }
        }
    }
```

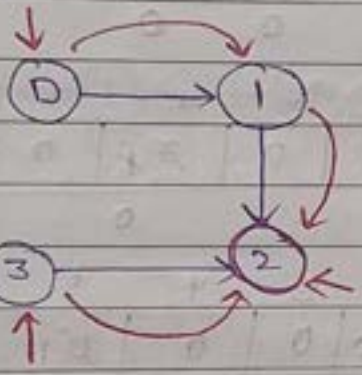
```
int numIslands(vector<vector<char>>& grid){

    int n = grid.size();
    int m = grid[0].size();

    int count = 0;

    for(int i = 0; i<n; i++){

        for(int j=0; j<m; j++){

            if(grid[i][j] == '1'){
                grid[i][j] = '0';
                count++;
                BFS(grid, i, j);
            }
        }
    }

    return count;
}
```

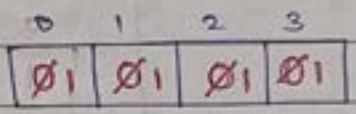
Lecture 61

Directed Graph cycle + Topological Sort

problem : Detect cycle in a directed graph.

Example:



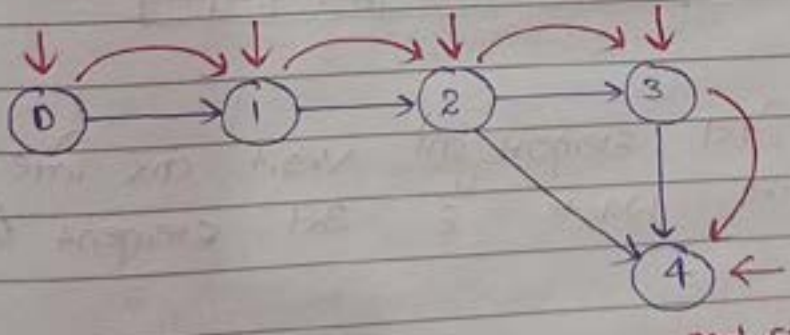


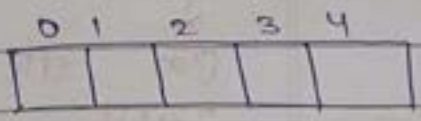
3 think 2 is visited then it form cycle

Try logic of undirected graph.

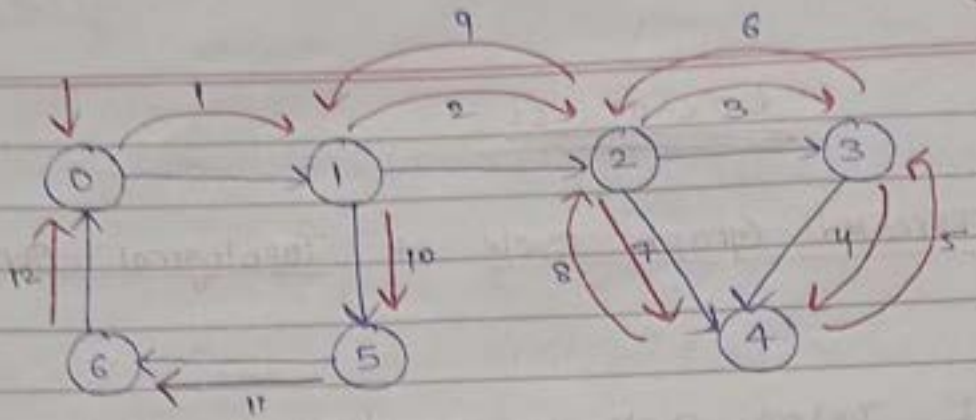
But this Approach fails here.

Approach 2 :





Not cycle



Step 1 : path

0	1	2	3	4	5	6
~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ x	~~0~~ ~~x~~	0	0
			0	0		

0	1	2	3	4	5	6
1	1	1	0	~~0~~ x	0	0
				0		

0	1	2	3	4	5	6
1	1	~~x~~ 0	0	0	~~0~~ 1	~~0~~ 1

0	1	2	3	4	5	6
1	1	0	0	0	1	1

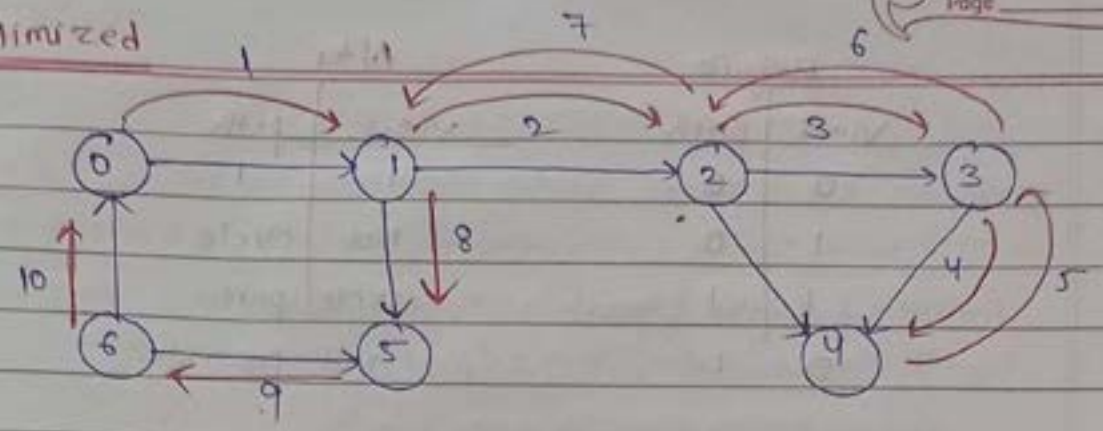
visited :

0	1	2	3	4	5	6
~~0~~ 1	~~0~~ 1	~~0~~ ~~0~~ 1	~~0~~ ~~0~~ 1	~~0~~ 1	~~0~~ 1	~~0~~ 1

↓

एक बार जिस element को visit कर लिए तो दुबारा जाना, waste है उस element पे।

Optimized



path :

0	1	2	3	4	5	6
$\emptyset_1$	$\emptyset_1$	$\emptyset_1$	$\emptyset_1$	$\emptyset_1$	0	0

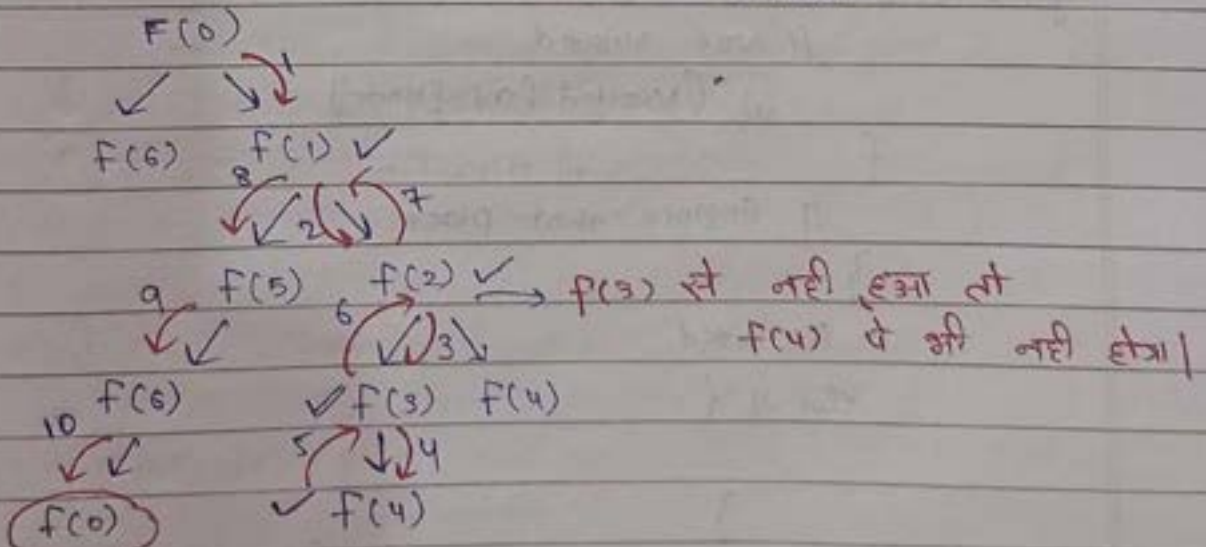
0	1	2	3	4	5	6
1	1	$\cancel{0}_0$	$\cancel{0}_0$	$\cancel{0}_0$	0	0

0	1	2	3	4	5	6
1	1	0	0	0	$\emptyset_5$	$\emptyset_1$

Visit :

0	1	2	3	4	5	6
$\emptyset_1$	$\emptyset_1$	$\emptyset_1$	$\emptyset_1$	$\emptyset_1$	$\emptyset_1$	$\emptyset_1$

0 visited then
↳ Circle.

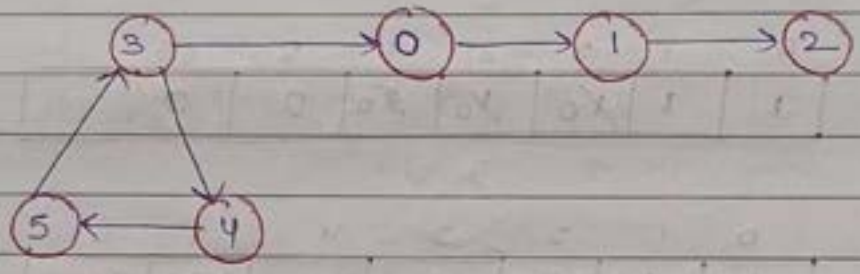


formed circle

return 1.

Before		After	
Visit	path	visit	path
0	0	1	1
1	0	Not	cycle
1	1	cycle	part
0	1	Not possible	

Example:




```
path (Node) = 1
Visit (node) = 1;

for (int i=0  i<adj [node] size ; i++) {
        // Not Visited
          if (visited (adj [Node]
    {
      // explore that place
    }
     // visited
     else if {

        }.
```

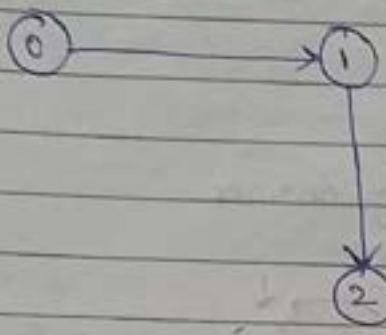
classmate
Date
Page

Code :

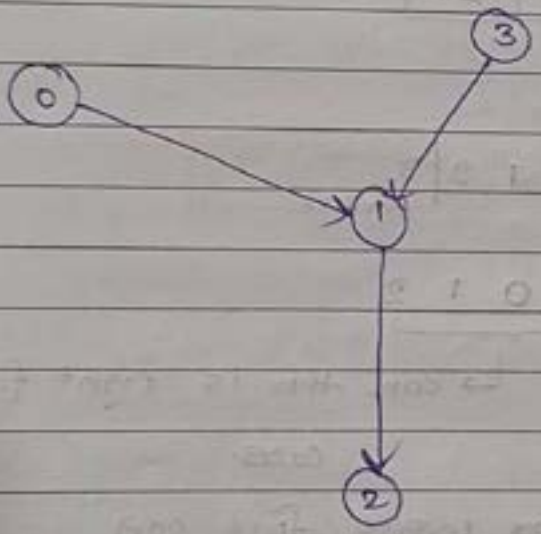
```
bool DFS (vector<int> adj[], vector<bool> &visited,
          vector<bool> &path, int node) {

    visited[node] = 1;
    path[node] = 1;

    for(int i=0; i< adj[node].size(); i++) {

        if (visited[adj[node][i]] == 0) {

            if (DFS(adj, visited, path, adj[node][i]))

                return 1;
        }

        else {

            if (path[adj[node][i]])
                return 1;

        }
    }

    path[node] = 0;

    return 0;

}
```

```
bool isCyclic (int v, vector<int> adj[]) {

    vector<bool> visited (v, 0);

    vector<bool> path (v, 0);

    for (int i = 0; i < v; i++) {

        if (!visited[i]) {

            if (DFS (adj, visited, path, i))

                return 1;

        }

    }

    return 0;

}
```

problem 8: Topological sort



0 1 2 ✓ 1 0 2 ✗



u → v

uv

u पहले v बाद

0 3 1 2 ✓

3 0 1 2 ✓

→ If Graph is cycle, then topological sort is not possible.

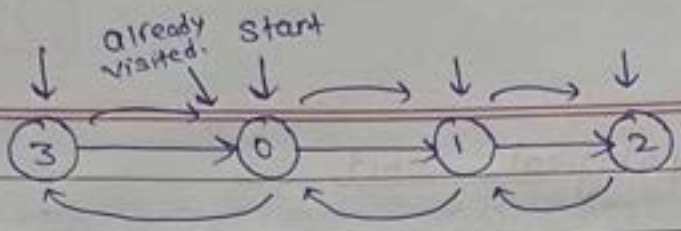
→ Topological sort is only possible for acyclic graph with direction. (DAG) Directed Acyclic Graph.

↓

→ (0) → (1) → (2)

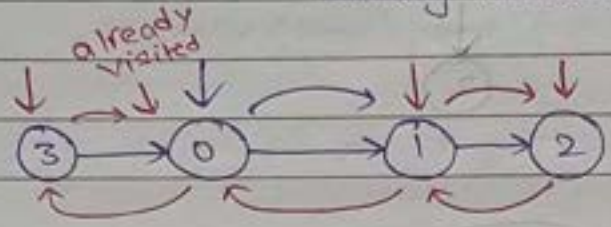
(1) DFS

0 1 2 ✓



0 1 2 3

↳ Wrong answer



DFS

0	1	2	3
✗	✗	✗	✗

0 1 2 | 3

reverse

3 0 1 2

↳ can this is right for all cases.

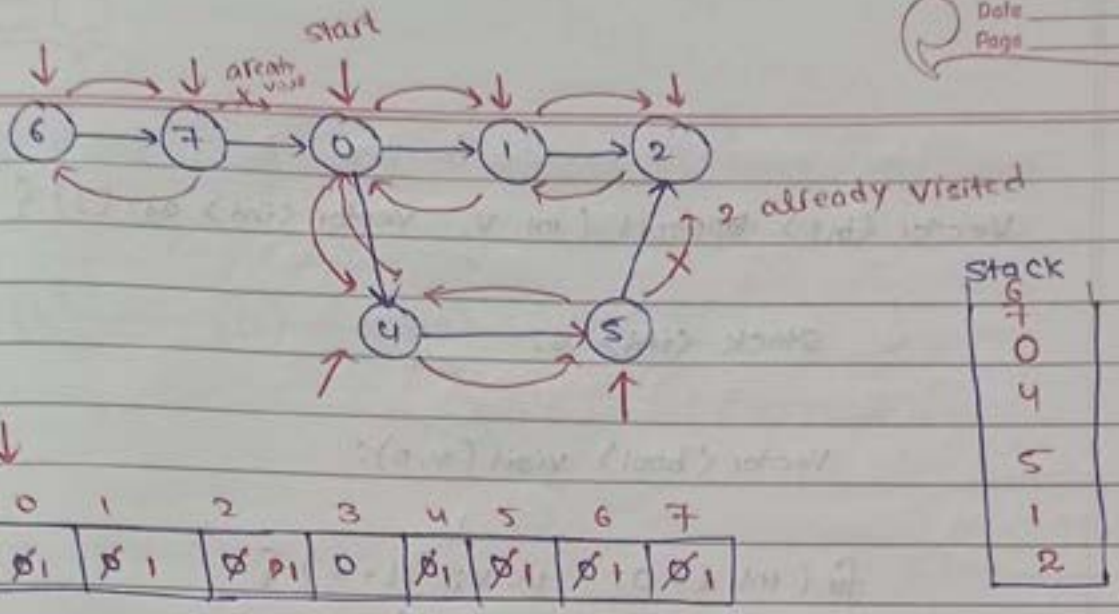
We can reverse first part

0 1 2 | 3

2 1 0 3

Then we reverse whole

3 0 1 2



Store the answer in reverse order.

use stack to store reverse order.

6	7	0	4	5	1	2

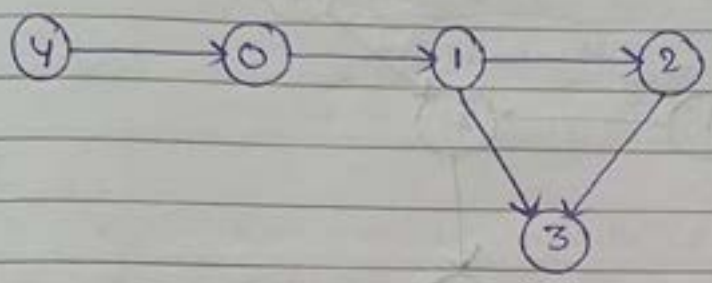
↳ Topological sort.

Code :

```
Void DFS (vector <int> adj[], stack[int> &s, vector <bool> &
          visit, int node) {

    visit[node] = 1;
    for(int i= 0;  i< adj[node].size();  i++) {
        if ( !visit[adj[Node][i]] )
            DFS (adj, s, visit , adj[node][i]);
    }
    S.push (node);

    return ;

}
```

```
Vector <int> toposort (int V, Vector <int> adj [J) {

    stack <int> s;

    Vector <bool> visit (v, 0);

    for ( int i = 0;  i < v;  i++ ) {
        if ( !visit [i] )
        DFS ( adj,  s,  visit,  i);
    }

    vector <int> ans;

    while ( !s.empty() ) {

        ans.push_back ( s.top() );
        s.pop ();

    }
    return  ans;

}
```

Kahn's Algorithm (Topological sort)



4	0	1	2	3

Index

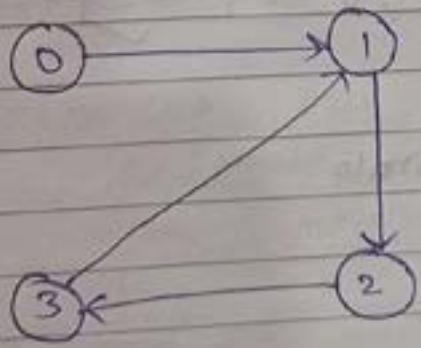
0	1	2	3	4
1	1	1	2	0

queue =

queue में हमेशा वही आ रहा है जिसका Indegree 0 हो।

अगर सारा traverse होने के बाद Array में किसी Index पे 1 हो तो graph cyclic होगा।

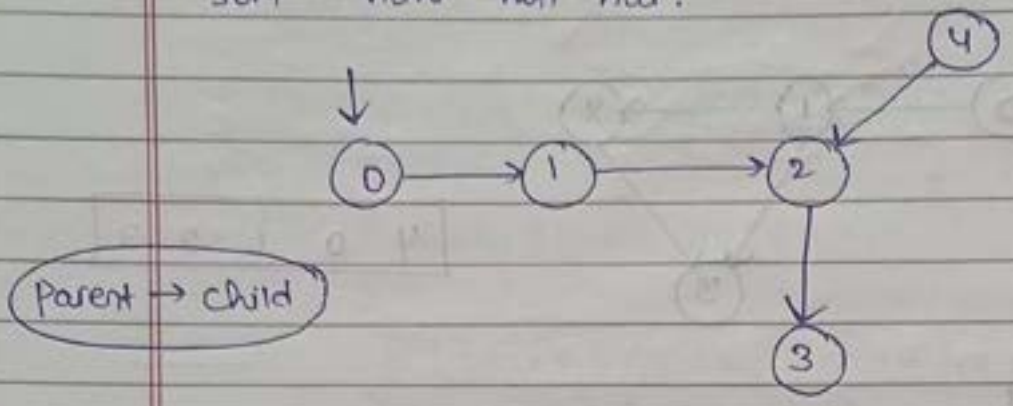
Ex



queue = {

0	1	2	3
0	2	1	1

Jab bhi Meri cycle bahti hai to Topological sort nahi hoti hai.



→ Jiska parent Nahi hai 4 हलें उसको stack Me dalo

q: {0, 4}

↓ ↓

q: {0, 1, 4, 2, 3}

is this correct for all cases.

→ Jiska indegree 0 होगा उससे start होगा।

	0	1	2	3	4
Indegree	0	~~0~~	~~2~~ ~~1~~ 0	~~0~~	0

4 ko bahar Nikalo

result : 4

4 goes to 2

↳ decrease indegree

0 ko bahar Nikalo

result : 4 0

0 goes to 1.

↳ decrease.

1 ko bahar Nikalo

result = 4 0 1

1 goes to 2

↳ decrease.

2 ko bahar Nikalo

result = 4 0 1 2

2 goes to 3

↳ decrease.

3 ko bahar Nikalo

result = 4 0 1 2 3

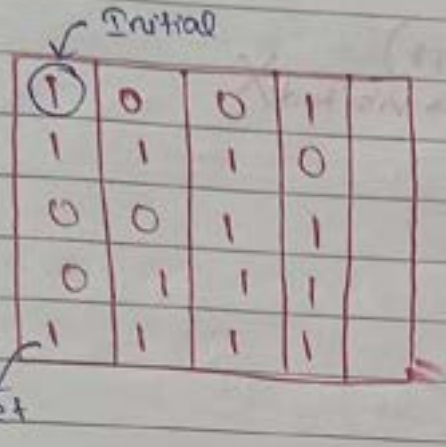
↳ final answer.

Code:

```
vector <int> toposort (int v, vector <int> adj[]) {

    vector <int> indeg (v, 0);

    for (int i=0; i<v; i++) {
        for (int j=0; j<adj[i].size(); j++) {
            indeg[adj[i][j]] ++;
        }
    }

    queue <int> q;
```

```
for (int i = 0; i < v; i++) {

        if (!indeg[i])
            q.push(i);

}

Vector<int> ans;
int node;

while (!q.empty()) {

    node = q.front();
    q.pop();

    ans.push_back(node);

    for (int i = 0; i < adj[node].size(); i++){
        indeg[adj[node][i]]--;
        if(indeg[adj[node][i]] == 0)
            q.push (adj[node][i]);
    }
}
return ans;
}
```

Lecture - 62



1	0	0	1	
1	1	1	0	
0	0	1	1	
0	1	1	1	
1	1	1	1	

path only possible on 1.
We cannot travel in 0.

You have to go to Target from Initial and find the minm path.

Use BFS.

	0	1	2	3
0	1	0	0	1
1	1	1	1	0
2	0	0	1	1
3	0	1	1	1
4	1	1	1	1

queue: {((1,0) 1)} ((row, col), step)

↓

{(1, 1), 2}

↓

((1, 2), 3)

↓

(2, 2), 4)

↓

((3,2), 5) ((2,3), 5)

↓ →

((3,1),6), ((4,2),6), ((3,3),6) ((3,3),6)

↳ already present.

((4,1),7) ((4,3),7)

↓ ↓

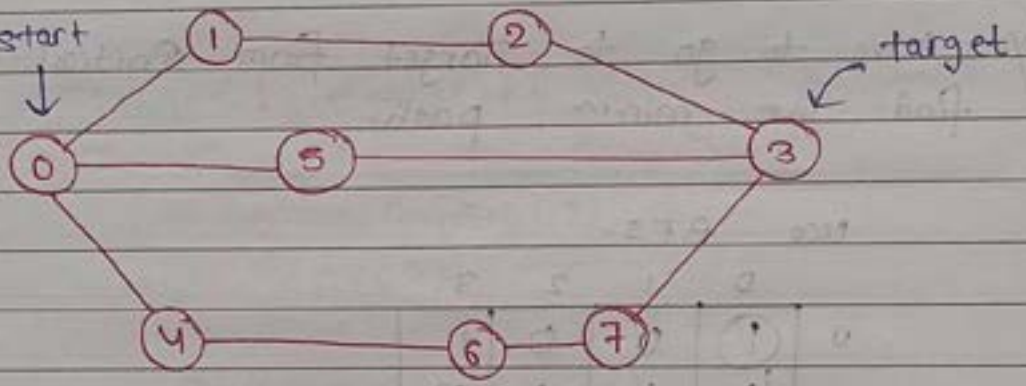
((4,0),8) ((4,2),8)

↳ Visited ✗

↓

(reached)

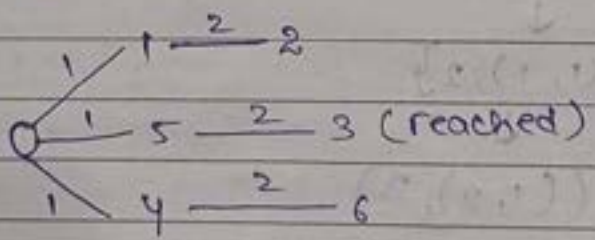
Minimum = 8

* 

→ All path cost 1.

→ find minimum cost to reach target.

BFS :



We take all path from min-m distance

$O(E+V)$

DFS:

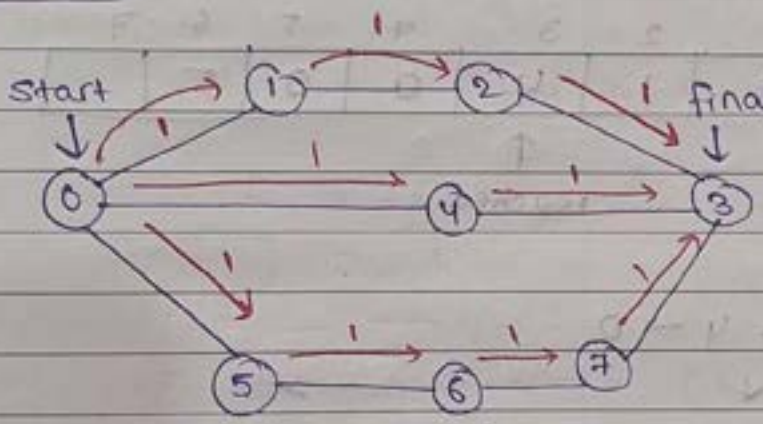
$0 \rightarrow 1 \rightarrow 2 \rightarrow 3$ (3) Cost = ~~7~~ 2

$0 \rightarrow 5 \rightarrow 3$ (2)

$0 \rightarrow 4 \rightarrow 6 \rightarrow 7 \rightarrow 3$ (4)

Ans = 2

Approach 3:



0	1	2	3	4	5	6	7
0	~~1~~	~~2~~	~~2~~	~~1~~	~~1~~	~~2~~	~~3~~

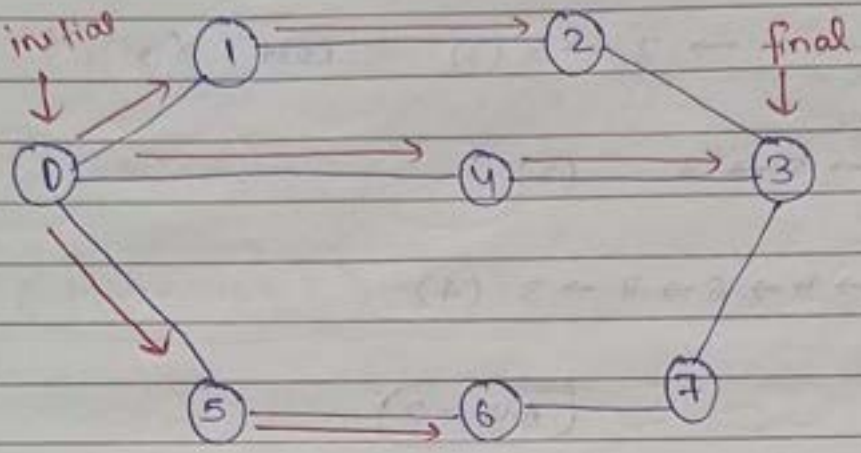
min(2, ∞)

min(2, ∞) 0 → 4 → 3
min(2, 3) 0 → 1 → 2 → 3
min(2, 4) 0 → 5 → 6 → 7 → 3

Agar Node ek baar visited hai to usko dubara visit hi nahi karna hai (only in BFS).

classmate
Date
Page

If you have to return path also :



0	1	2	3	4	5	6	7
0	0	1	4	0	0	5	

↑ reached.

$3 \rightarrow 4 \rightarrow 0$

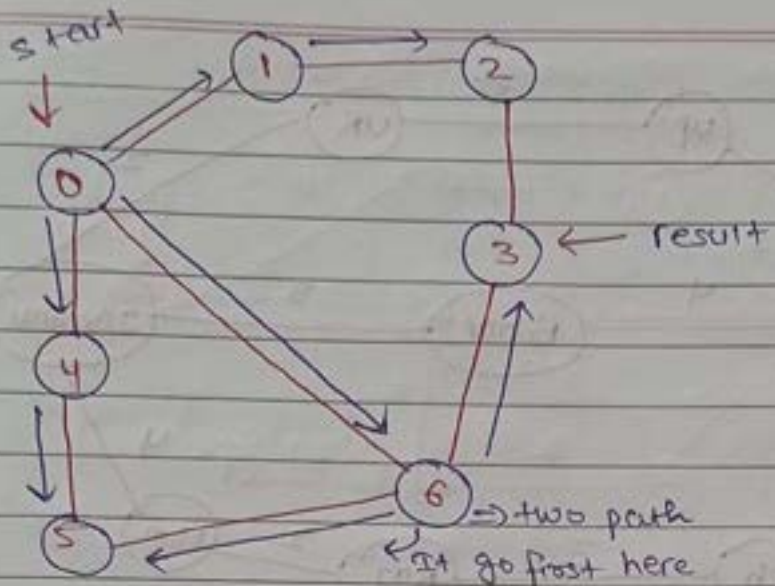
↓

reverse

$0 \rightarrow 4 \rightarrow 3 \rightarrow$ Ans

Shortest path :

We can store the parent Node in the array. and when we reach onee to the final then we stop the process and find the path.



0	1	2	3	4	5	6
0	0	1	6	0	6	0

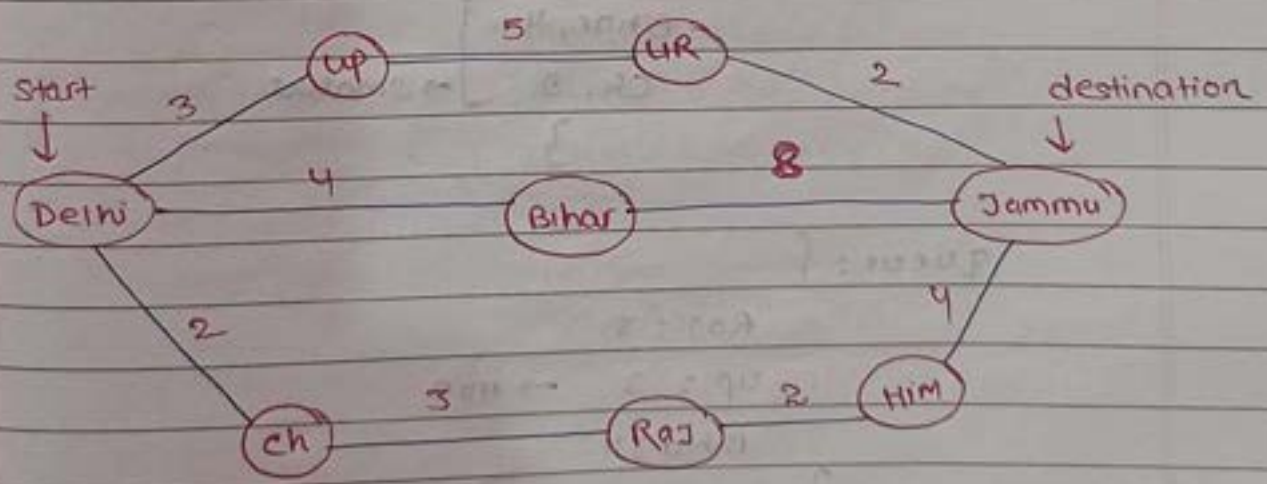
We store the parent in the array.

$3 \rightarrow 6 \rightarrow 0$

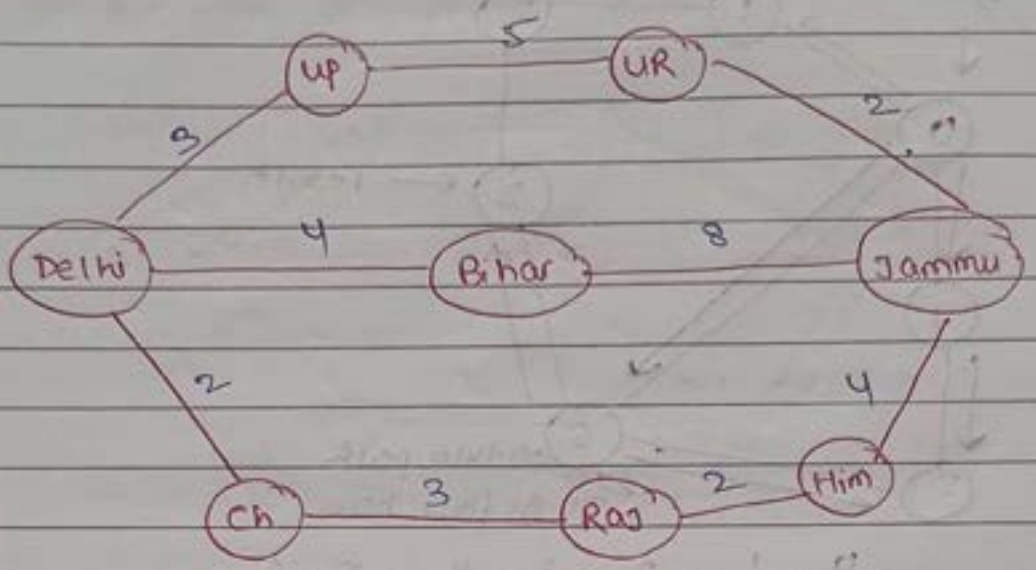
↓

reverse

$0 \rightarrow 6 \rightarrow 3$ (Answer).



Shortest path.



Queue = {
Delhi, 0
}

With help of priority Queue.

Queue : {
Delhi, 0 Removed
up, 3
Bihar, 4
Ch, 2 → smallest
}

Queue : {
Raj : 5
up : 3 → min
Bihar : 4
}

Queue : {
UR : 8
Bihar : 4 → min.
Raj : 5
}

classmate
Date
Page

queue : {
Jammu : 12
UR : 8
Raj : 5 → min.
}

queue : {
Jammu : 12
~~Jammu = 10~~
Him : 7 → min.
UR : 8
}

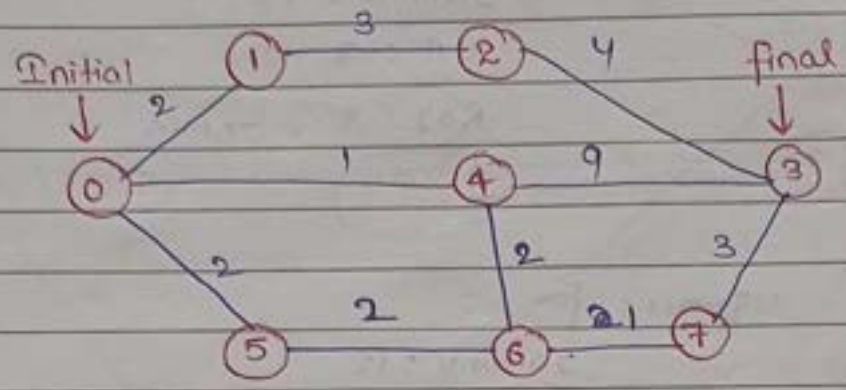
queue {
Jammu : 12
Jammu : 11
UR : 8 → min
}

queue {
Jammu : 12
Jammu : 11
Jammu : 10 → min
}

Cost(Min) = 10

classmate
Date
Page

* Dijkstra Algorithm:



Shortest path.

0 से सारे का Distance.

0	1	2	3	4	5	6	7
0	∞ 2	∞ 5	∞ 16	∞ 1	∞ 2	∞ 3	∞ 4
				+			

Node ↓ distance ↓

q : (1, 2)
(4, 1) → smallest distance
(5, 2)

q : (1, 2) → smallest
(5, 2)
(3, 10)
(6, 3)

q : (5, 2) → smallest
(3, 10)
(6, 3)
(2, 5)

q : (3, 10)
(6, 3) → min
(2, 5)
(6, 4)

classmate
Date
Page

q: (3,10)
(2,5)
(6,4) → min
(7,4)

(6,3)
(5,5) (7,4)

removed because all path are greater.

not done because we have reach 5 in 2.

q: (3,10)
(2,5)
(7,4) → min.

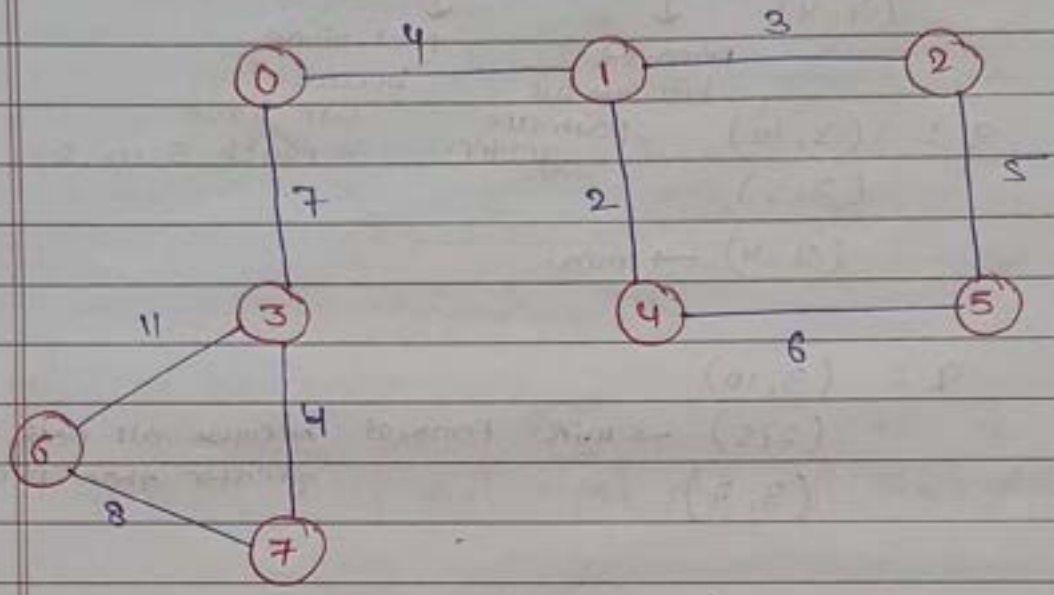
q: (3,10)
(2,5) → min removed because all path are greater than previous.
(3,7)

q: (3,10)
(3,7) → min (removed
↳ answer final

q: (3,10) → min
removed.

path: (shortest)
3 in 7 (cost)

MST : (Minimum spanning Tree)



Convert in into tree.

→ there is 8 Node

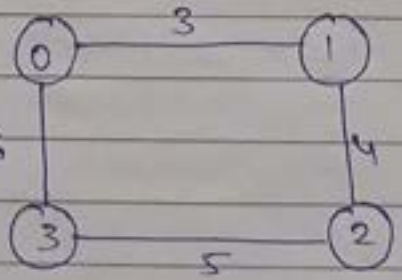
↳ so arrange in that we have only 7 Edges.

↳ and all Node are Connected to each other.

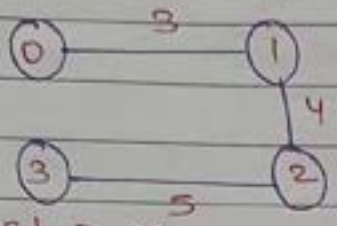
↳ and the Cost is also minimum.

Example :

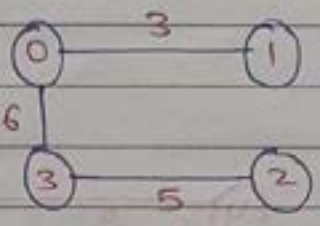
Graph



All possible tree from this graph :-

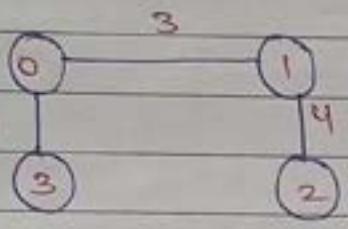


Cost = 12

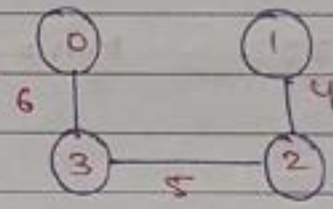


Cost = 14

Minimum

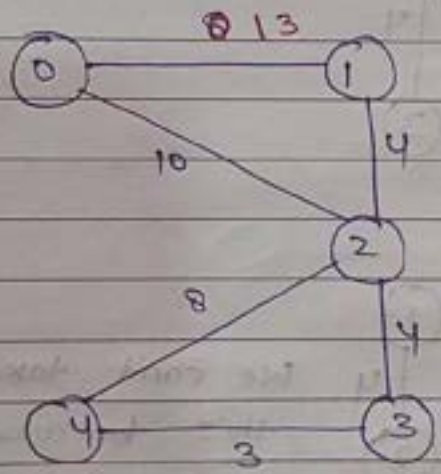


Cost = 13



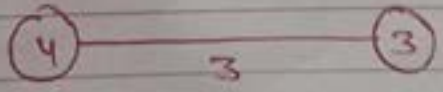
Cost = 15

So, this is MST.

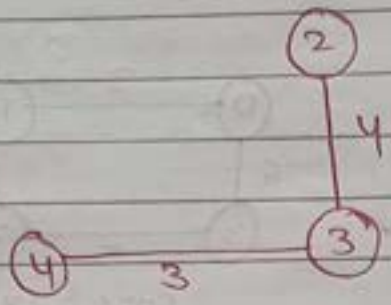


We start choosing from the min-m cost.
↳ 5 Node
↳ so we have to choose 4 edges.
Which connect all the ~~Edge~~ Node
↳ cycle nahi banne dena hai

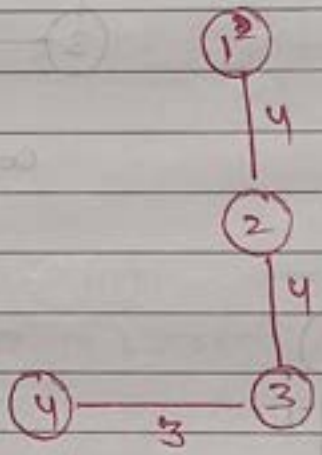
Step 1 : Min = 3



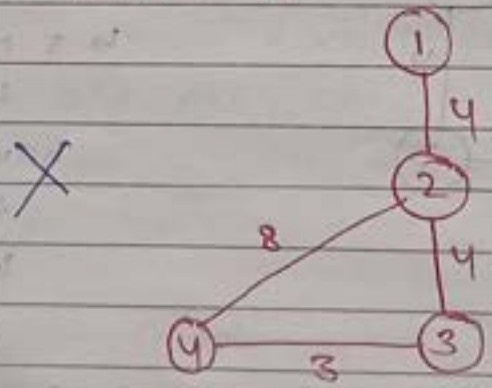
Step 2 : Min = 4



Step 3 : Min = 4

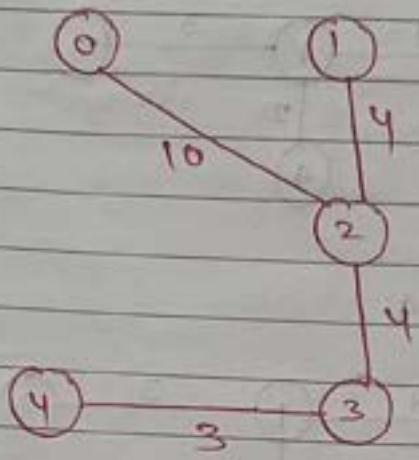


Step 4 : Min = 8



We can't take this because it can take and make a cycle

Step 5 : Min = 10

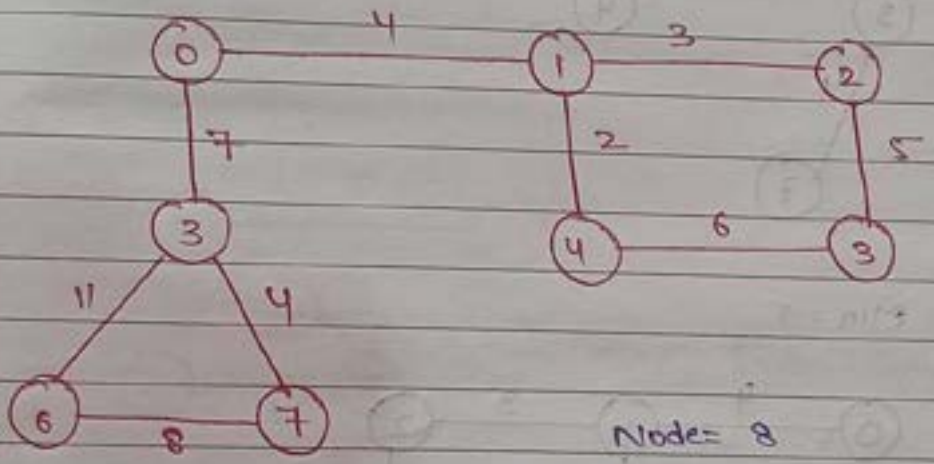


This is the min-m cost.

Final

This is known as kruskal Algorithm.

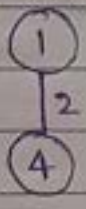
*



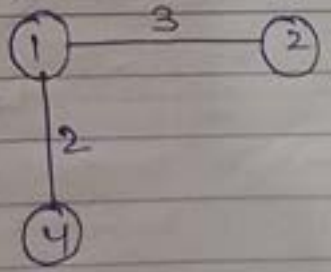
Node = 8

So, edge = 8-1 = 7

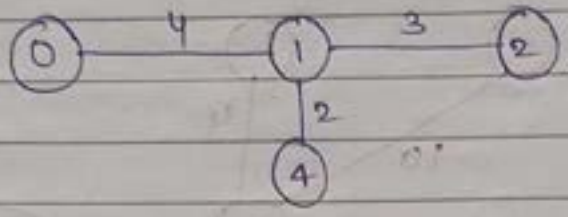
Step 1 : Min = 2



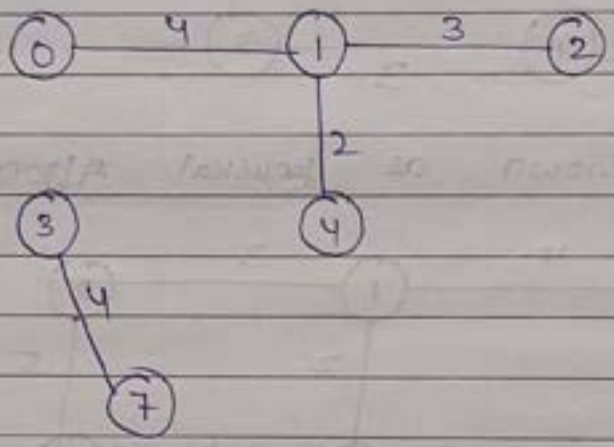
Step 2 : Min = 3



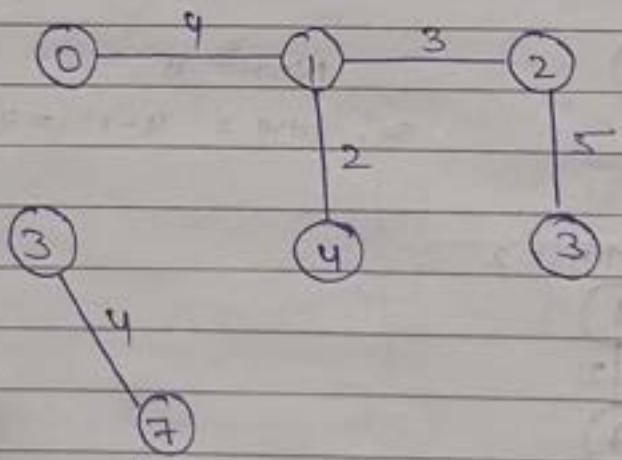
Step 3 : Min = 4



Step 4 : Min = 4



Step 5 : Min = 5

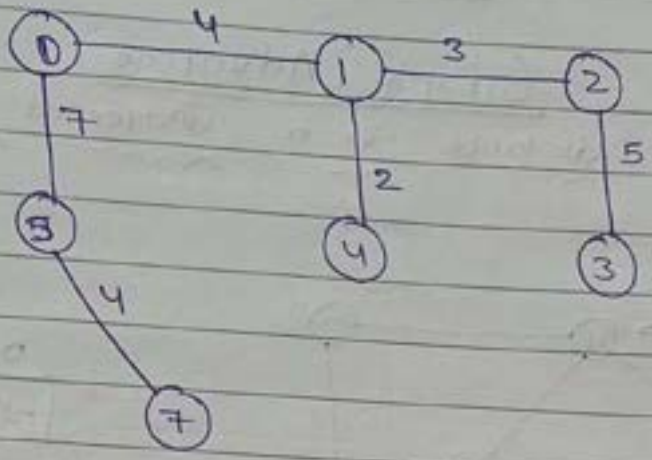


Step 6 : Min = 6

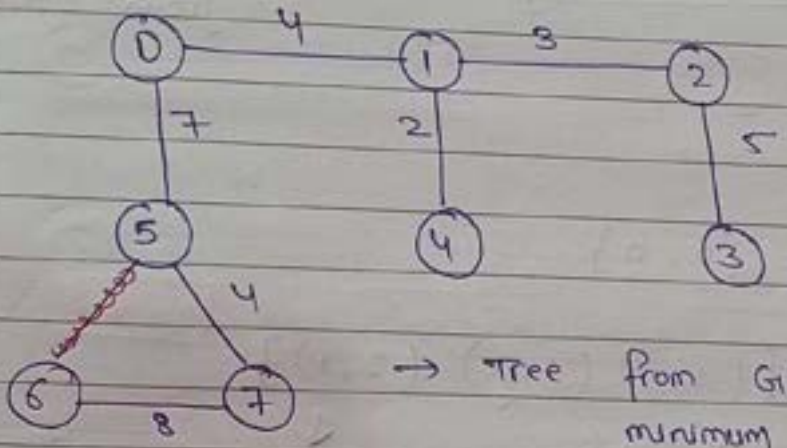
↳ can't take

because it can create a loop.

Step 7 : Min = 7



Step 8 : Min = 8



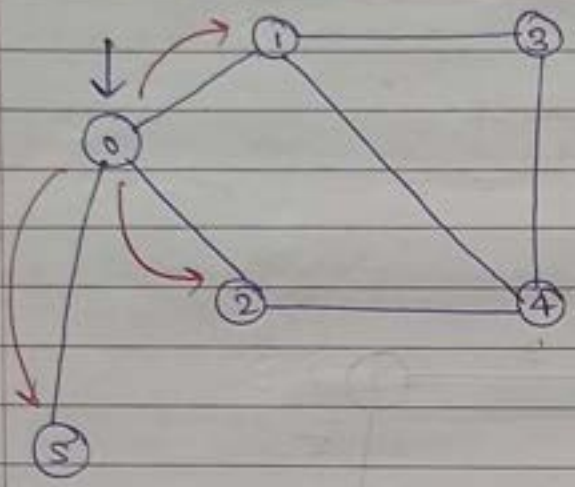
→ Tree from Graph with minimum cost.

Lecture 63

Graph Advance

* Shortest distance in a undirected graph

BFS



0	1	2	3	4	5
~~0~~	~~1~~	~~1~~	~~2~~	~~2~~	~~1~~

↓

This distance is from start (0).

$q = \{0, 0\}$

$q = \{(5,1)\ (1,1)\ (2,1)\}$

$q = \{\ \{(3,2),(4,2)\},\ \{(4,2)\}\}$

↓

ignore because पहले से हो चुका है।

→ Source Node ko queue Me push kro

→ Jab Tak queue empty Na ho

↳ queue ka front ko pop karo

↳ Front ke sare neighbour ko dekho

↳ if neighour is visted

↳ don't do Anything

↳ else neighbour is new

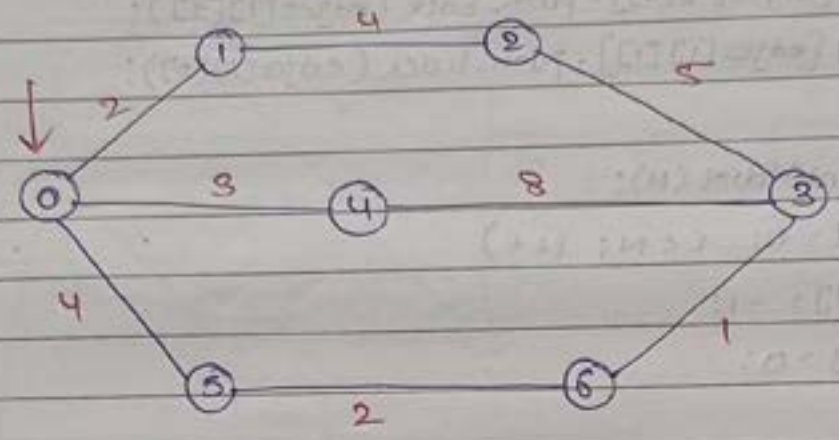
↳ distance ++.

↳ push that neighbour into queue

code:

```
vector<int> adj[N];
for (int i=0; i<M; i++) {
    adj[edges[i][0]].push_back(edges[i][1]);
    adj[edges[i][1]].push_back(edge[i][0]);
}
vector<int> dist(N);
for (int i=0; i<N; i++)
    dist[i] = -1;
dist[src] = 0;

queue<pair<int, int>> q;
q.push({src, 0});
int i, step;
while (!q.empty()) {
    i = q.front().first;
    step = q.front().second;
    q.pop();

    for (int k=0; k<adj[i].size(); k++) {
        if (dist[adj[i][k]] == -1) {
            q.push({adj[i][k], step+1});
            dist[adj[i][k]] = step+1;
        }
    }
}
return dist;
}
```

* Implementing Dijkastra's Algorithm



BFS:

0	1	2	3	4	5	6
0	~~Max~~ 2	~~Max~~ 6	~~Max~~ 11	~~Max~~ 3	~~Max~~ 4	~~Max~~ 6

(Min(6, Max) ↓ 7

q → {(0, 0)}

↓

q → {(1, 2), (4, 3), (5, 4)}

↓ ↓ ↓

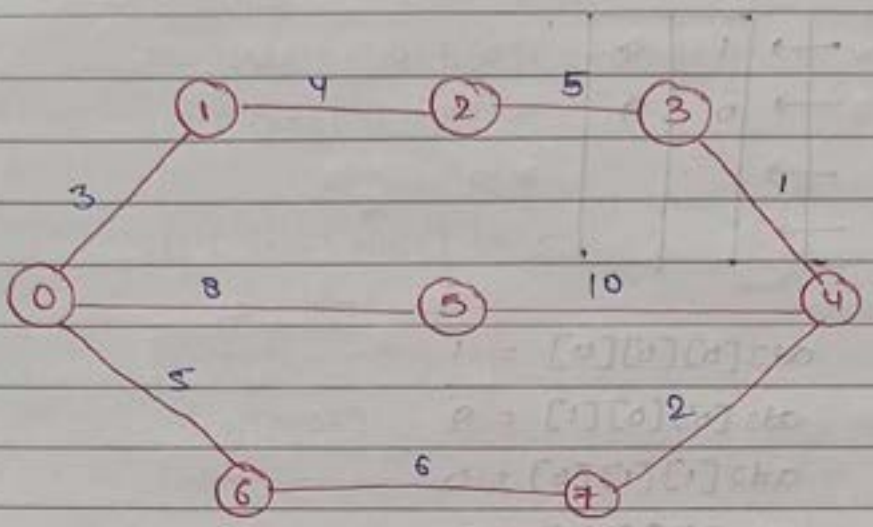
q → {(2, 6)} {(3, 11)} {(6, 6)}

↓ ↓ ↓

(3, 11) × (3, 7)

(3,6) & (3,8) → अगर एक साथ आ जाए।

→



0	1	2	3	4	5	6	7
0 ~~∞~~	~~∞~~ 3	~~∞~~ 7	~~∞~~ 12	~~∞~~ 13	~~∞~~ 8	~~∞~~ 5	~~∞~~ 11

q : {(0,0)}

↓

q : {(1,3), (5,8), (6,5)}

↳ value not update in vector

↳ value updated when it pop from queue

q : {(2,7)}, (4,12), (7,11)}

↓

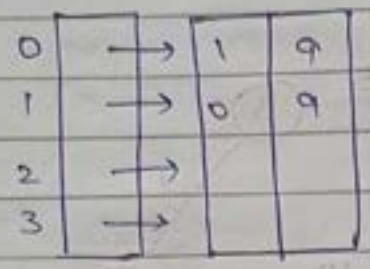
q : {(3,12),

↓

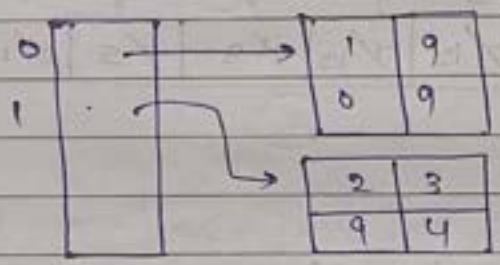
(4,13)

~~(4,13)~~ (4,13)}

Code :



adj[0][0][0] = 1
adj[0][0][1] = 9
adj[1][1][0] = 0
adj[1][1][1] = 9



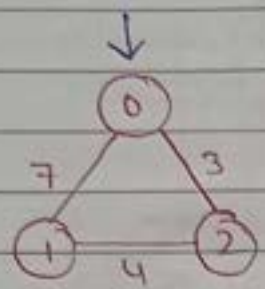
Code :

```
vector <int> dijkstra (int V, vector<vector<int>> adj[], int S){

    vector <int> dist(V);
    for (int i = 0; i < V; i++){
            dist[i] = -1;
    }

priority_queue <pair<int, int>, vector <pair <int, int>>,
        greater <pair<int, int>>> q;

    q.push ({0, s});

    int Node, i, j, step;
```

```
While (!q.empty()){
    step = q.top().first;
    Node = q.top().second;
    q.pop();

  if (dist[Node] != -1)
     continue;

  dist[Node] = step;

 for(int j=0; j<adj[Node].size(); j++) {
    if (dist[adj[Node][j][0]] == -1) {
       q.push({ step + adj[Node][j][1], adj[Node][j][0]});

    }
  }
 }
 return dist;
}

};
```

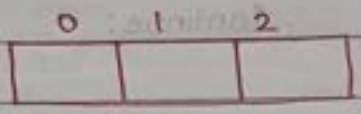
classmate
Date
Page

* Minimum spanning tree :



N Nodes

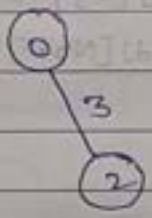
N-1 Edges.



0 → Node → dist

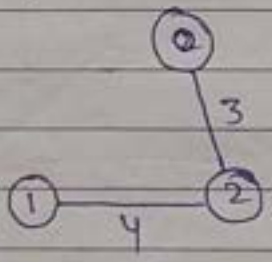
(1, 7)

(2, 3) Min ✓



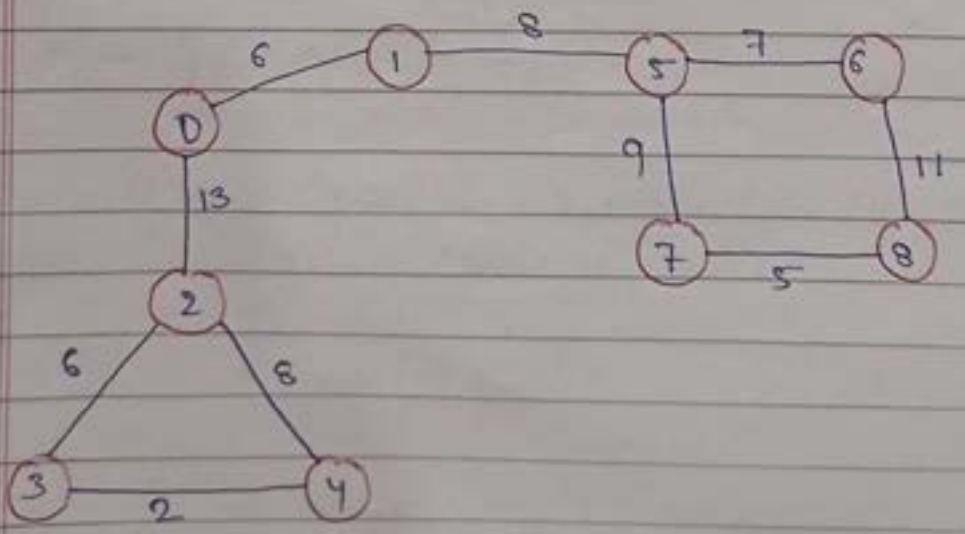
(1, 7)

(1, 4) Min ✓



3 + 4 = 7 Ans

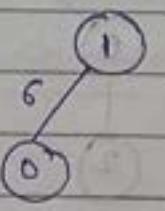
*



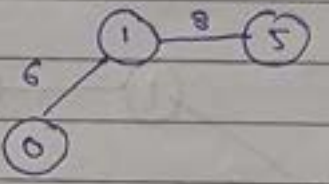
classmate
Date
Page

start with 1 :

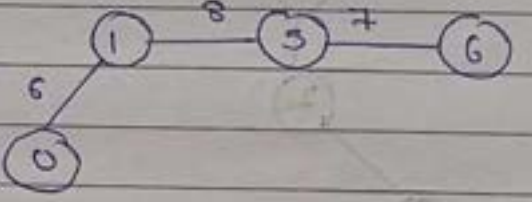
(0, 6) ✓ min
(5, 8)



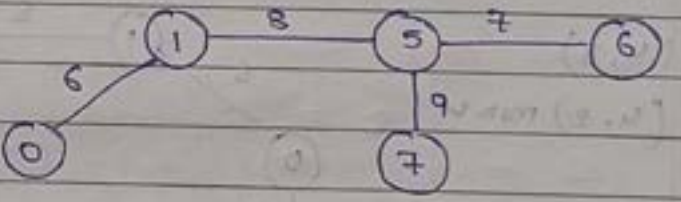
(5, 8) ✓ min
(2, 13)



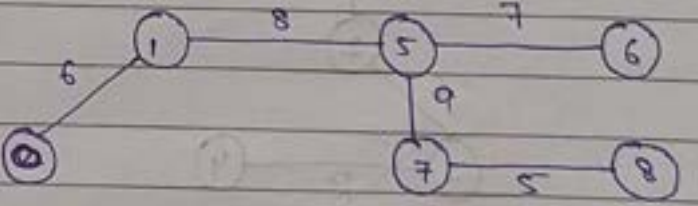
(7, 9)
(2, 13)
~~(3, 6)~~
~~(4, 8)~~
(6, 7) ✓ min



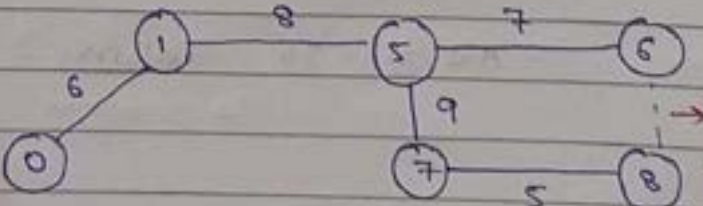
(2, 13)
(7, 9) ✓ min
(8, 11)



(2, 13)
(8, 11)
(8, 5) ✓ min

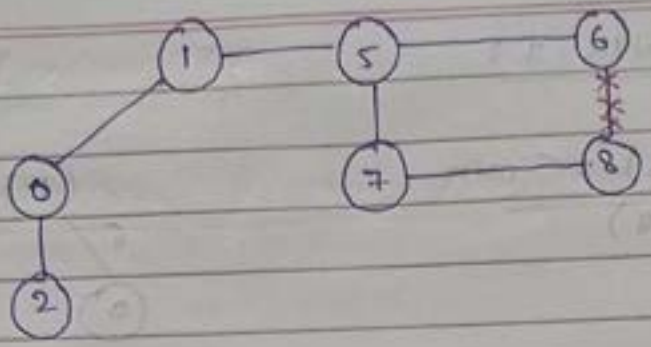


(2, 13)
(8, 11) Min ✓



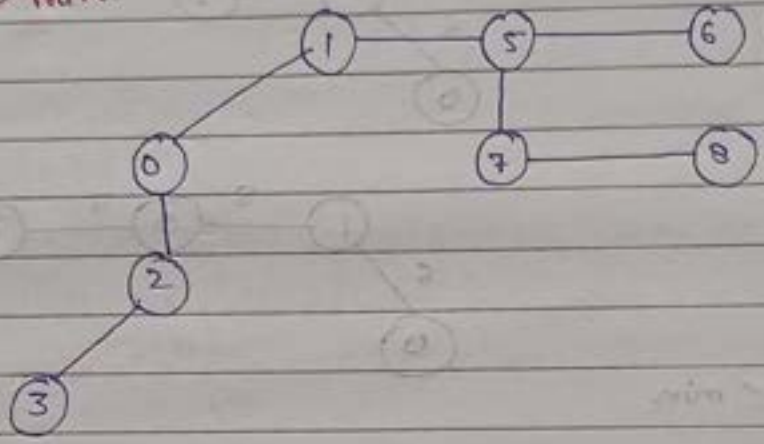
→ by joining this it create a cycle, so we discard it.

(2, 13) ✓ min



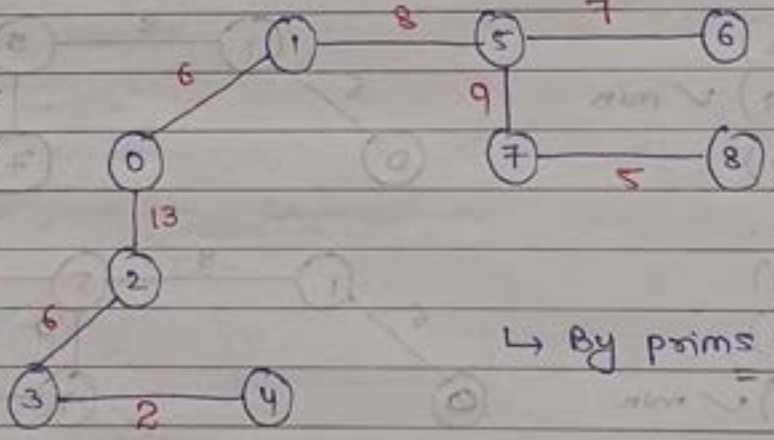
(3, 6) ✓ min.

(4, 8)



(4, 8)

(4, 2) min ✓



↳ By prims

Ans = 56 return.

→ Node liya

→ Jo uske sabse pass wala Node hai

↳ If cycle → discard

↳ on not cycle

↳ Select it and choose Min.

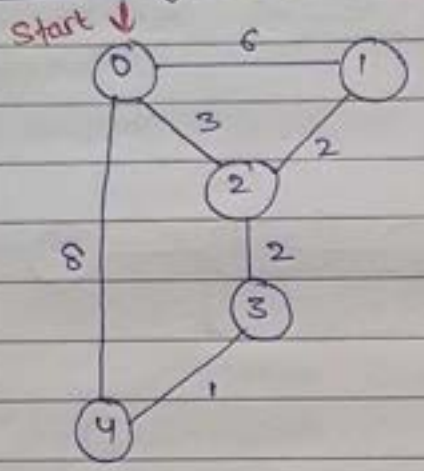
→ Kruskal

↳ connects Minm Edge.

Lecture 64

Prims and Kruskal Algorithm

* **Prims Algorithm :**

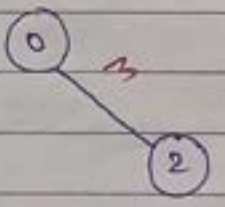


choose any Node :

Node ↑ Weight ↑

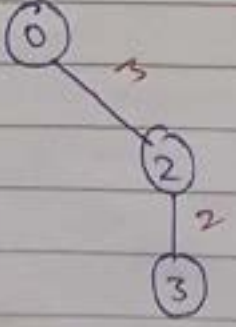
0 → {(4, 8), (2, 3) (1, 6)}

↳ minimum

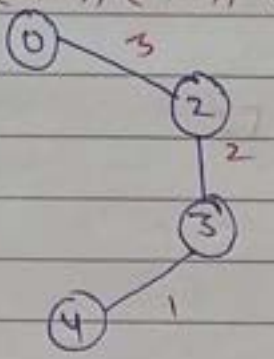


0 → {(4,8), (1,6), (3,2), (1, 2)}

↳ minimum



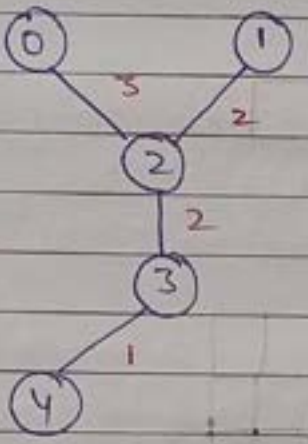
{(4,8), (1,6), (1,2), (4,1)}



↳ min-H

{(4,8), (1,6), (1,2)}

↳ min-H



Make a Weigth Array:

	0	1	2	3	4
Weight :	~~max~~ 0	~~max~~ ~~8~~	~~max~~ 3	~~max~~ 2	~~max~~ ~~8~~ 1

	0	1	2	3	4
parent :	-1	~~-1~~ ~~8~~ 2	~~-1~~ 0	~~-1~~ 2	~~-1~~ ~~8~~ 3

	0	1	2	3	4
visited :	~~8~~ 1	~~8~~ 1	~~8~~ 1	~~8~~ 1	~~8~~ 1

→ Node Bahar Nikalo
  ↳ check visited or not
    ↳ if visited skip the step
    ↳ else visited[node] = 1
  ↳ look at adjacent Node

→ Adjacent node is not visited && wt > curr_wt
  ↳ then push in queue.

3D Array :

```
vector<vector<int>> arr[]
```

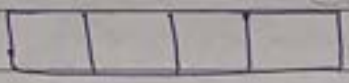
2D Array :

```
vector<int> arr[]
```

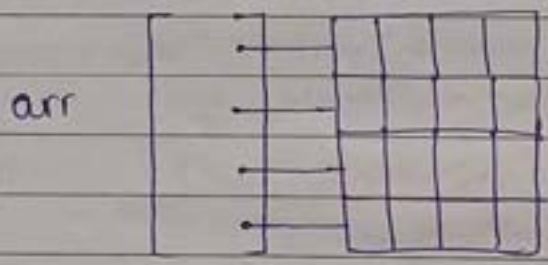
1D Array :

```
arr[]
```

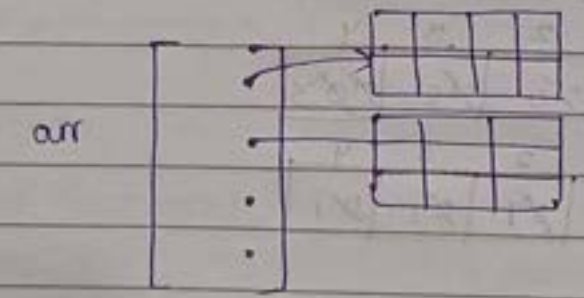
1D :



2D :



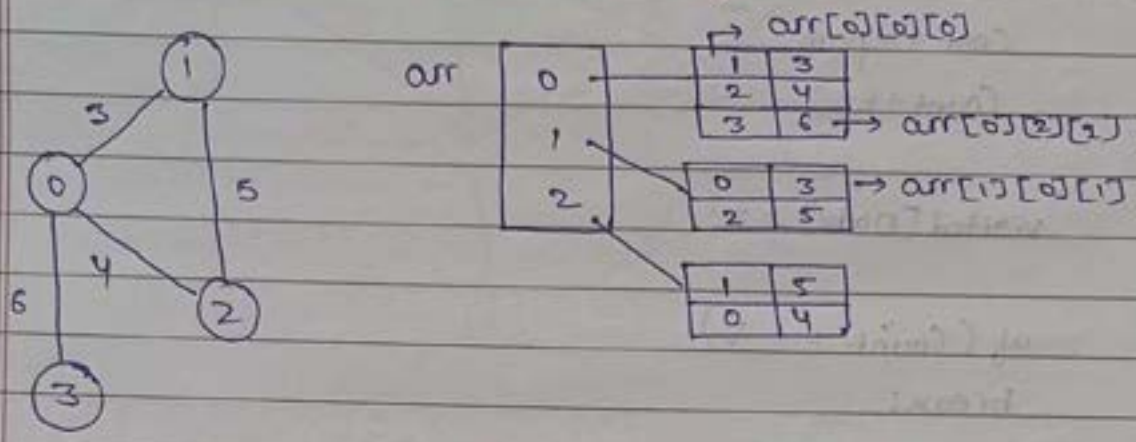
3D :



Code : (Min-m spanning Tree)

↓

By Prims Algorithm.




```
int spanningTree (int V, vector<vector<int>> adj[]) {

    vector<int> weight (V, INT_MAX);
    vector<int> parent (V, -1);
    vector<bool> visited (V, 0);
    int count = 0;
    int cost = 0;

priority_queue <pair<int, int>, vector<pair<int, int>>,
                greater<pair<int, int>>> q;

q.push ({0, 0});

int node, path, adjnode, adjpath;

while (!q.empty()) {
    path = q.top().first;
    node = q.top().second;
    q.pop();
```

```
        if (visited[node] == 1)
            continue;

        cost += path;
        Count++;

        visited[node] = 1;

        if (Count == V)
            break;

        for (int i = 0; i < adj[node].size(); i++) {
            adjnode = adj[node][i][0];
            adjpath = adj[node][i][1];

            if (!visited[adjnode] && adjpath < weigth[adjnode]) {
                q.push({adjpath, adjnode});
                parent[adjnode] = node;
                weight[adjnode] = adjpath;
            }
        }
    }

    return cost;
}
```

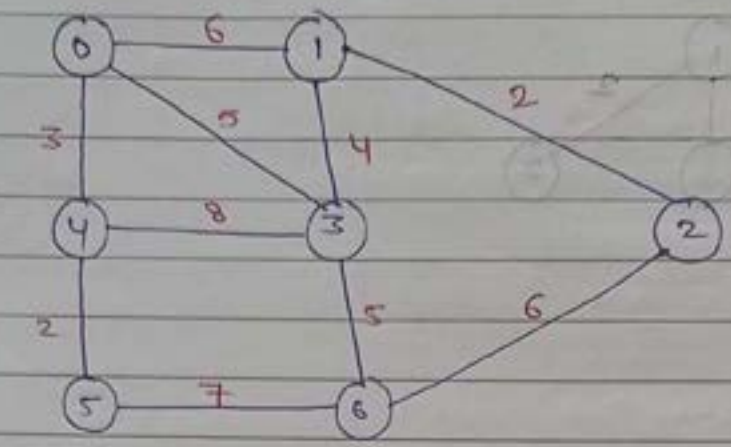
Time Complexity : $E * \log E$

$E \log V$

Space Complexity : V^2

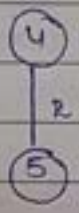
classmate
Date
Page

* Kruskal Algorithm

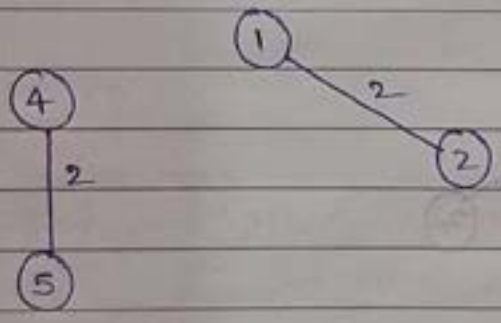


Find Min-m (Edge)

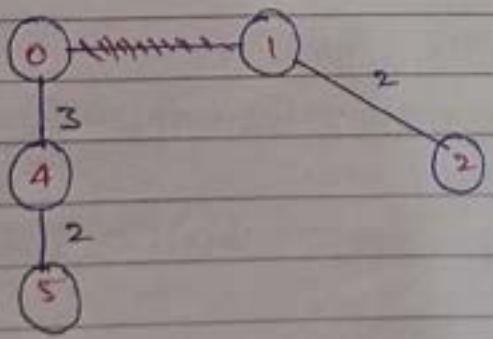
→ 2



→ Min = 2

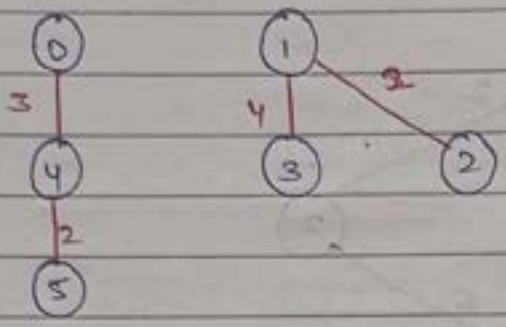


→ Min = 2

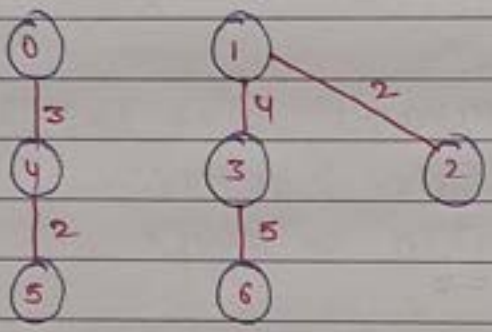


→ Min = 3

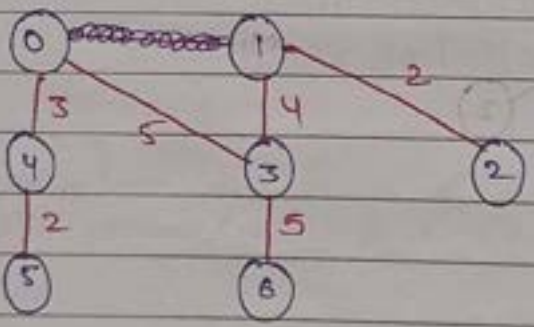
Min = 4



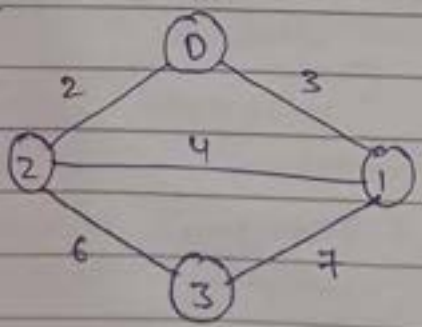
Min = 5



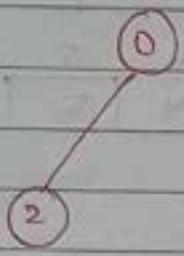
Min = 5, 6



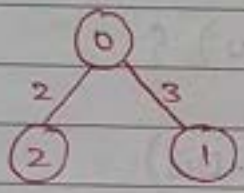
Example 2:



Min = 2



Min = 3

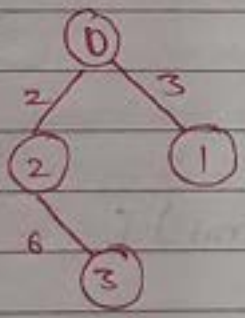


Min = 4

(Create cycle)

↳ ignore

Min = 6



Kruskal

↳ Minimum Spanning tree.

→ Insert Every Edge into min heap
→ check for cycle . pop edge one by one
→ if cycle is not forming Add it
→ else discard

classmate
Date
Page

(1) rank
size

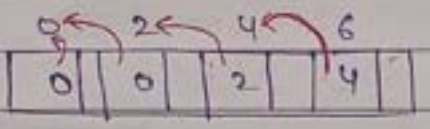
	0	1	2	3	4	5	6
rank	∅1	0	0	0	0	0	0

	0	1	2	3	4	5	6
parent:	0	~~0~~	~~0~~	3	4	5	6

(0, 1)
(1, 2)
(3, 4)
(6, 5)
(2, 3)
(1, 6)

find parent

```
int find_parent ( int u) {

        if ( u == parent[u])
            return u;

        return  parent[u] = find_parent (u);

}
```



Code:

```
int find_parent (int u, vector<int>&parent ) {
        if ( u == parent[u])
            return u;

        return  parent[u] = find_parent ( parent[u], parent);

}

bool unionset (int u, int v, vector<int> &ranic, vector<int> & parent)
            int pu = find_parent (u, parent);
            int pv = find_parent ( v, parent);
```

```
        if ( pu == pv)
            return 1;

        if ( rank[pu] == rank[pv] ) {
                rank[pu]++;
                parent[pv] = pu;
        }
        else if ( rank[pu] < rank[pv]) {
                parent[pu] = pv;
        }
        else {
                parent[pv] = pu;
        }
        return 0;
}

int spanningTree ( int V, vector<vector<int>> adj[]) {

        vector<int> parent(V);
        vector<int> parent (V, 0);
        for ( int i=0 ; i<V; i++)
                parent[i] = i;

priority_queue <pair<pair<int, int>, int> , vector<pair<pair<int, int>int>>
        greater<pair<pair<int, int>, int>>> q;

    int u, v, w;

  for ( int i=0;  i < V; i++ ) {
        for ( int j=0;  j<adj[i].size(); j++) {
                u = i;
                v = adj[i][j][0];
                w = adj[i][j][1];
```

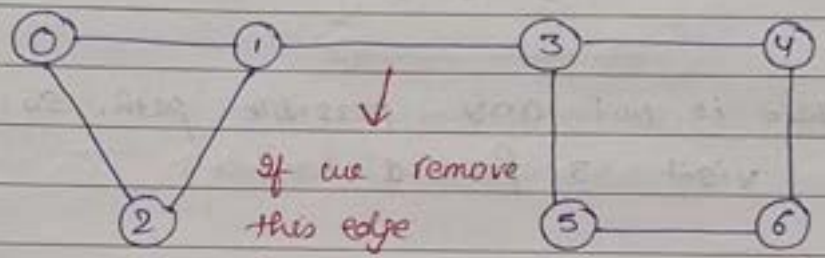
```
            q.push({{w, u}, v});
        }
    }

    int count_edge = 0, cost = 0;

    while (!q.empty()) {
        if (count_edge == V-1)
            break;

        w = q.top().first.first;
        u = q.top().first.second;
        v = q.top().second;
        q.pop();

        if (unionset(u, v, rank, parent))
            continue;

        count_edge++;
        cost += w;
    }
    return cost;
}
```

Lecture 65

Strongly Connected Component + Bellmonford

* **Bridge :**

Agar Edge tut jaye to hum kabhi bhi dusre side nahi ja payenga.



So this edge is bridge.

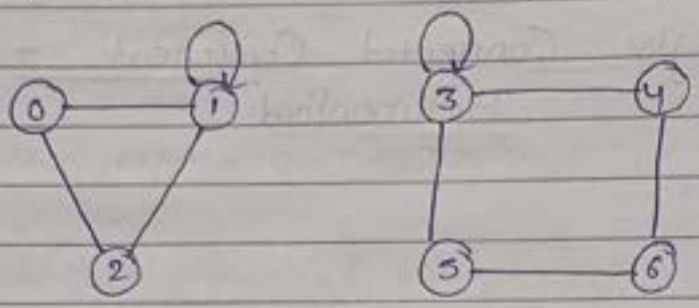
Adj: (List)

0 : 1 2
1 : 0 2 3
2 : 0 1
3 : 1 4 5
4 : 3 6
5 : 3 6
6 : 4 5

Edge : U V
1 3

Check this edge is bridge or not.

→ Make u & v to the self loop



→ Now we want to go v from v.

1 to 3

There is Not any possible path. So we can visit 3 from 1.

So, (u,v) → (1,3) is a bridge.

Code :

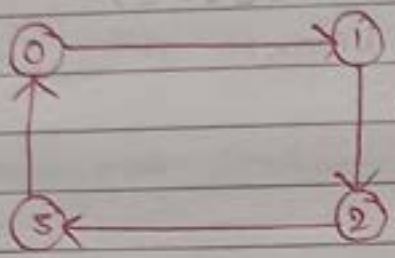
```
bool DFS (int node, int target, vector<int> adj[],
          Vector<bool> & visited) {

    if (node == target)
        return 1;

    visited[node] = 1;

    for (int i=0; i<adj[node].size(); i++) {
        if (!visited[adj[node][i]]) {
            if (DFS(adj[Node][i], target, adj, visited)
                return 1;
        }
    }
```

```
    return 0;
}

int isBridge (int V, vector<int> adj[], int c, int d) {

    for(int i=0; i< adj[c].size(); i++) {
        if (adj[c][i] == d) {
            adj[c][i] = c;
            break;
        }
    }

    for(int i=0; i< adj[d].size(); i++) {
        if (adj[d][i] == c) {
            adj[d][i] = d;
            break;
        }
    }

    vector<bool> visited (V, 0);

    return !DFS (c, d, adj, visited);
    // 1: target Node found, → bridge nahi hai → return 0
    // 0: target Node not found → bridge hai → return 1

}
```

* Strongly Connected Components (Kosaraju's Algo)

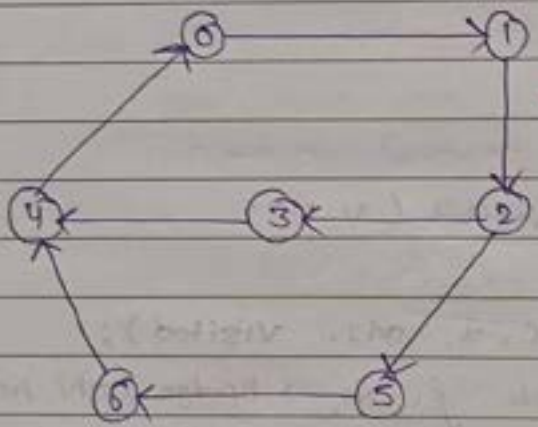
(Directed graph)

from one vertex of a graph, we can reach any other vertex.

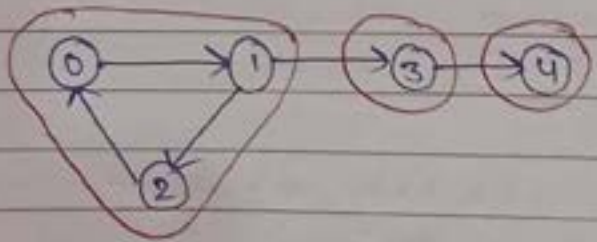


→ Strongly Connected Component.

cycle is always a strongly connected component



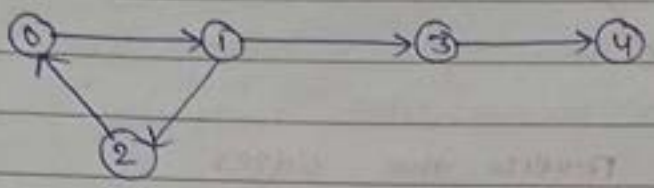
→ This is also a strongly connected component.



How many strongly connected component?

↳ Ans : 3

How to find :



Start from 4

↳ 4 cannot visit further

Count = 1

Start from 3

↳ 3 cannot proceed further

Count = 2

Start from 0

0 → 1 → 2

Count = 3

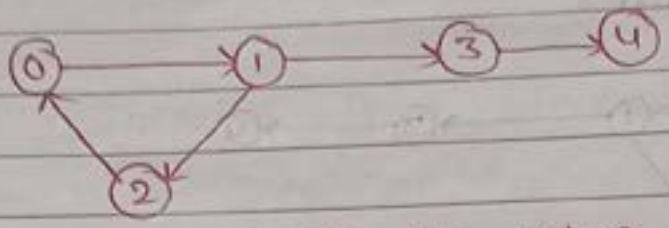
return 3

We can reverse the arraw.

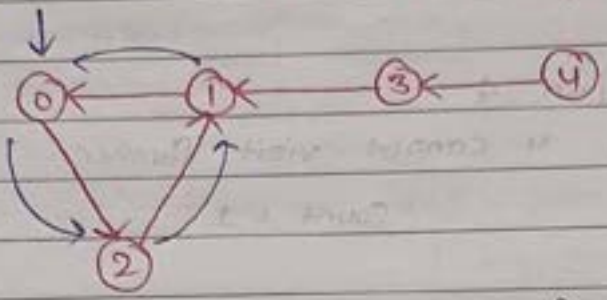
Reversing arrow in cycle cannot affect in counting of strongly connected components.

We can reverse all the edges

then we see what will happen.



reverse the edges



Now start traversing from 0.

0 → 2 (Not visited)

Make 2 visited

2 → 1 (Not visited)

Make 1 visited

1 → 0 (visited)

↓

Count = 1

The we go to 3

3 → 1 (visited)

Count = 2

Then we goto 4

4 → 3 (visited)

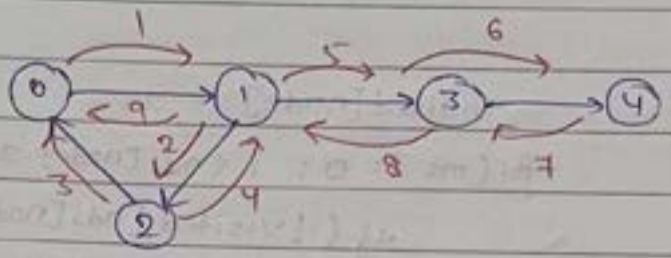
Count = 3

Return Count

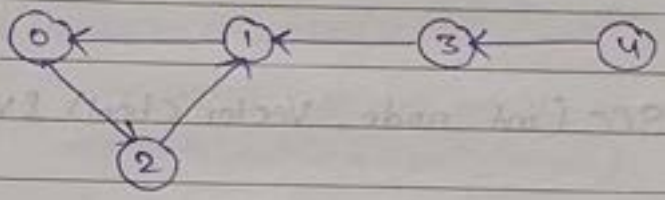
classmate
Date
Page

① Topological sort
② Graph Transpose
③ DFS logo do.

0
1
3
4 3
2



Graph transpose



0 pop → visits : 0, 1, 2

Count = 1

1 pop → visited

3 pop → visits : 1 (visited)

Count = 2

4 pop → visits : 3 (visited)

Count = 3

2 pop → visited

Final Answer : 3

Code :

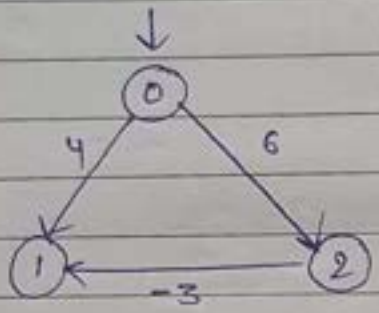
```
Void DFS (int node, Vector<vector<int>> &adj, stack<int> &s,
          Vector<bool> &visited) {

    visited[node] = 1;
    for(int i=0; i<adj[node].size(); i++) {
        if (!visited[adj[node][i]])
            DFS(adj[node][i], adj, s, visited);
    }
    s.push(node);
}

Void SCC (int node, Vector<bool> &visited, vector<vector<int>>&adj
{
    visited[node] = 1;
    for (int i=0; i<adj[node].size(); i++) {
        if (!visited[adj[node][i]])
            SCC (adj[node][i], visited, adj);
    }
}

int Kosaraju (int V, vector<vector<int>> & adj) {

    stack<int> s;
    Vector<bool> visited (V, 0);

    for(int i=0; i<V; i++) {
        if (!visited[i])
            DFS(i, adj, s, visited);
    }
```

```
vector<vector<int>> trans(V);
for(int i=0; i<V; i++){
    for(int j=0; j<adj[i].size(); j++){
        trans[adj[i][j]].push_back(i);
    }
}
int count = 0;
for(int i=0; i<V; i++)
    visited[i] = 0;

while(!s.empty()){

    if(!visited[s.top()]){
        count++;
        scc(s.top(), visited, trans);
    }
    s.top();
}
return count;
}
```

* Disadvantages of Dijkastra :
  ↳ doesn't work properly for the negative path

We need an algorithm that work on Negative edges.

* **Bellman - Ford Algorithm**



0 → 1 (2 rupees)
2 → 1 (-3 rupress)
0 → 2 (6 rupees)

Initially at 0

→

	0	1	2
distance	0	Max	Max

(0 → 0)

→

0	1	2
0	4	Max

(0 → 1)

→

0	1	2
0	4	Max-3

(~~0 → 2~~)

2 → 1

→

0	1	2
0	4	6

0 → 2

One we traverse all the path.

We again traverse all the path.

→ (0 → 1)

0	1	2
0	4	6

dist[i] = min (dist[i], New value)

→ (2 → 1)

0	1	2
0	3	6

(2 → 1) = 0 → 2 → 1

2 → 1 → given (-3)

0 → 2 = 6

6 - 3 = 3

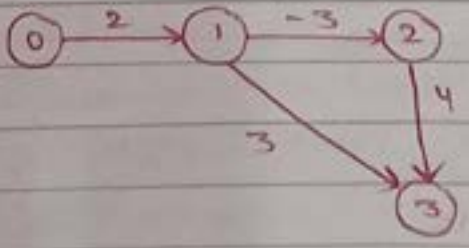
min (4, 3) = 3

→ 0 → 2

0	1	2
0	3	6

↳ Answer

Example 2:



classmate
Date
Page

$0 \rightarrow 1 = 2$

$1 \rightarrow 2 = -3$

$2 \rightarrow 3 = 4$

$1 \rightarrow 3 = 3$

Initially.

	0	1	2	3
distance from 0	0	Max	Max	Max

$(0 \rightarrow 1)$

0	1	2	3
0	2	Max	Max

$(1 \rightarrow 2)$

0	1	2	3
0	2	-1	Max

$1 \rightarrow 2$

$0 \rightarrow 1 \rightarrow 2$

$2 - 3 = -1$

$(1 \rightarrow 3)$

0	1	2	3
0	2	-1	5

$1 \rightarrow 3$

$0 \rightarrow 1 \rightarrow 3$

$2 + 3 = 5$

Next Cycle

0 → 1

0	1	2	3
0	2	-1	5

1 → 2

0	1	2	3
0	2	-1	5

0 → 1 → 2

2 - 3 = -1

2 → 3

0	2	-1	3

2 → 3

0 → 1 → 2 → 3

2 + (-3) + 4

= 3

0	1	2	3
0	2	-1	3

↳ Final Answer.

dist[v] = min(dist[v], dist[u] + w);

→ Bellman ford does not work on Negative cycle.

↳ Works on undirected graph but all edges should be positive.

→ Bellmon ford → catches negative cycle.

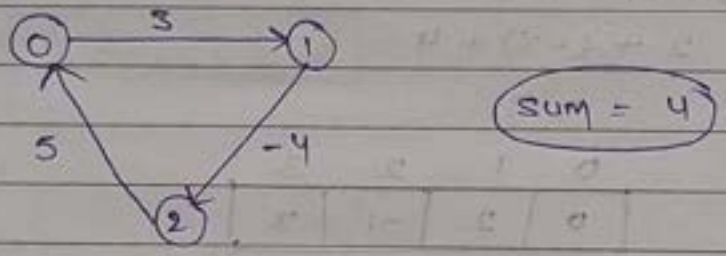
(n-1) → we get final answer cycle

→ After (n-1) cycle, the answer will changes then it contain negative cycle.

Lecture 66

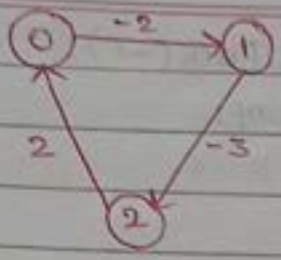
Bellman Ford Algorithm

Negative weight cycle:



If Sum > 0
↳ then it is not Negative Weight cycle

If Sum < 0
↳ then H is Negative Weight cycle



Sum = -3

Sum < 0

↳ Negative Weight cycle.

Code :

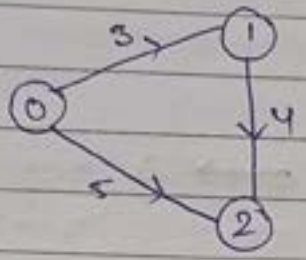
```
bool Bellmon (vector <vector <int>> & edges, vector <int> & dist) {
        bool Count = 1;
    for (int i = 0 ; i < edges.size(); i++ ) {
        if (dist [edges[i][1]] > dist [edges [i][0]] + edges [i][2]) {
            dist [edges [i][1]] = dist [edges[i][0]] + edges [i][2];
            Count = 0;
        }
    }
    return Count;
}
Vector <int> bellman_ford (int V, vector<vector <int>> & edges, int S){

    Vector <int> dist (V, 1e8);
     dist[S] = 0;

    for (int i = 0; i < V; i++) {
        if (Bellmon (edges, dist))
            return dist;
        }

    if (! Bellmon (edges, dist)) {
        vector <int> temp;
```

```
        temp.push_back(-1);
        return temp;
    }

    return dist;
  }
};
```

Lecture - 67

Graph Advance Euler

* **Floyd Warshall**



we have to find shortest distance for

0 → 1
0 → 2

Bellman Ford

1 → 0
1 → 2

Time Complexity : $(V \times E)$

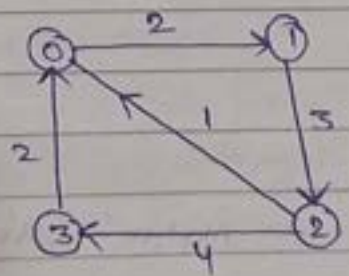
2 → 0
2 → 1

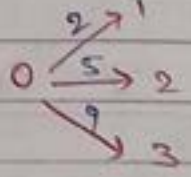
Floyd Warshall

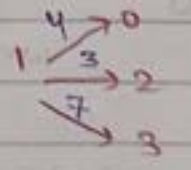
↳ V^3 time

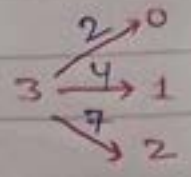
↳ It also detect -ve cycle and tell that it can't able to find answer for -ve cycle.

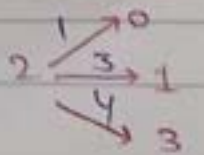
Example:











We have to return Matrix :

Given : A^{-1}

	0	1	2	3
0	0	2	-1	-1
1	-1	0	3	-1
2	1	-1	0	4
3	2	-1	-1	0

Let travel : $0 \longrightarrow 3$

① $0 \rightarrow 3$

② $0 \rightarrow 1 \rightarrow 3$

③ $0 \rightarrow 2 \rightarrow 3$

~~④ $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$~~

* 0 1 2 3 4 5

$0 \rightarrow 4$

$0 \rightarrow 1 \rightarrow 4$

$0 \rightarrow 2 \rightarrow 4$

$0 \rightarrow 3 \rightarrow 4$

$0 \rightarrow 5 \rightarrow 4$

$(0 \rightarrow 2)$ means shorter path from 0 to 2.

pick 1 node :

0

We create a Matrix A_0

$1 \rightarrow 0 \rightarrow 4$

classmate
Date
Page

A_0

	0	1	2	3
0	0	2	-1	-1
1	-1	0	3	-1
2	1	3	0	4
3	2	4	-1	0

* $0 \to 1$
  ↳ $0 \to 0 \to 1$
  $0 + 2 = 2$

* $0 \to 2$
  ↳ $0 \to 0 \to 2$
  $0 - 1 = -1$

* $1 \to 0$
  $1 \to 0 \to 0 \Rightarrow -1 + 0 = -1$

* $2 \to 0$
  $2 \to 0 \to 0 \Rightarrow 1 + 0 = 1$

* $3 \to 0$
  $3 \to 0 \to 0 = 2 + 0 = 2$

* $1 \to 2$
  $1 \to 0 \to 2 = -1 + -1 =$ Not possible.

* $2 \to 1$
  $2 \to 0 \to 1 = 1 + 2 = 3$

A_1

	0	1	2	3
0	0	2	5	-1
1	-1	0	3	-1
2	-1	3	0	-1
3	-1	4	7	0

* $0 \to 2 \Rightarrow 0 \to 1 \to 2 = 5$
* $0 \to 3 \Rightarrow 0 \to 1 \to 3$ -1
* $2 \to 0 \Rightarrow 2 \to 1 \to 0 = -1$
* $2 \to 3 \Rightarrow 2 \to 1 \to 3 = -1$
* $3 \to 0 \Rightarrow 3 \to 1 \to 0 = -1$
* $3 \to 2 \Rightarrow 3 \to 1 \to 2 = 7$

A_2

	0	1	2	3
0	0	2	5	-1
1	-1	0	3	-1
2	-1	3	0	-1
3	-1	4	7	0

* $0 \to 1 \Rightarrow 0 \to 2 \to 1 \Rightarrow 8$
* $0 \to 3 \Rightarrow 0 \to 2 \to 3 \Rightarrow -1$
* $1 \to 0 \Rightarrow 1 \to 2 \to 0 \Rightarrow -1$
* $1 \to 3 \Rightarrow 1 \to 2 \to 3 \Rightarrow -1$
* $3 \to 0 \Rightarrow 3 \to 2 \to 0 \Rightarrow -1$
* $3 \to 1 \Rightarrow 3 \to 2 \to 1 \Rightarrow 10$

A_3

	0	1	2	3
0	0			-1
1		0		-1
2			0	-1
3	-1	4	7	0

* $0 \to 1 \Rightarrow 0 \to 3 \to 1$
* $0 \to 2 \Rightarrow 0 \to 3 \to 2$
* $1 \to 0 \Rightarrow 1 \to 3 \to 0$
* $1 \to 2 \Rightarrow 1 \to 3 \to 2$
* $2 \to 0 \Rightarrow 2 \to 3 \to 0$
* $2 \to 1 \Rightarrow 2 \to 3 \to 1$

Time Complexity: V^3

Space Complexity: V^2

Auxiliary space: $O(1)$

2nd Approach

	0	1	2	3
0	0	2	-1	-1
1	-1	0	3	-1
2	1	-1	0	4
3	2	-1	-1	0

Make -1 as Max

	0	1	2	3
0	0	2	Max	Max
1	Max	0	3	Max
2	1	Max	0	4
3	2	Max	Max	0

matrix[i][j] = min (matrix[i][j], matrix[i][k] + matrix[k][j])

Code :

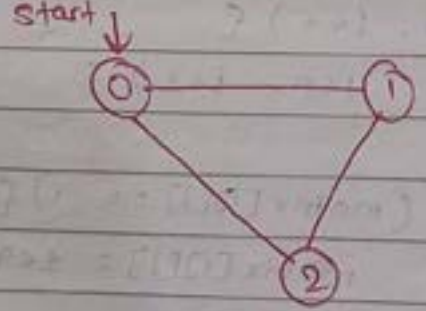
```
Void Shortest_distance (vector<vector<int>>& matrix) {

    int n = matrix.size();

for (int i = 0;  i<n; i++) {
        for (int j=0; j<n; j++) {

            if (matrix[i][j] == -1) {
                matrix[i][j] = 1e9;
            }
        }
    }
```

```
for (int k=0; k<n; k++) {
    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            matrix[i][j] = min(matrix[i][j], matrix[i][k]
                                + matrix[k][j]);
        }
    }
}
```

```
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
        if (matrix[i][j] == 1e9)
            matrix[i][j] = -1;
    }
}
```

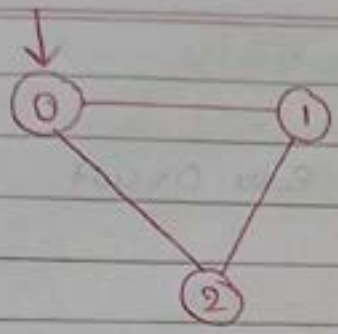
* Euler Circuit (undirected graph)

① start node = end node



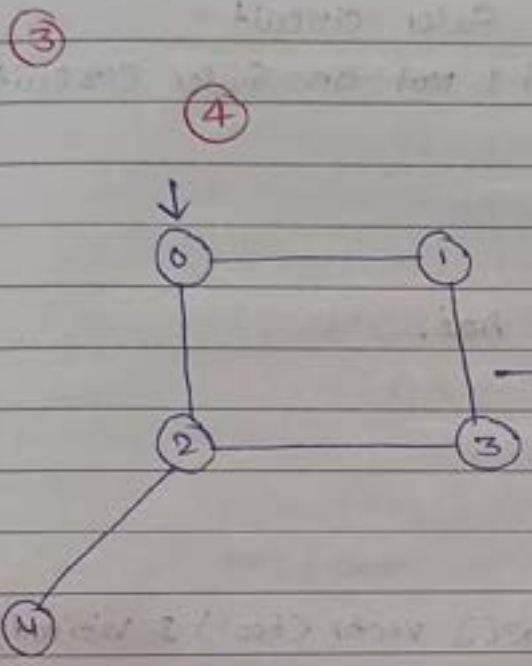
② sare Edge ko traverse karna hai

③ Ek Edge ko sirf Ek baar cover karenge.



सारे Edge पे जाना Compulsary है।

→ Euler circuit.

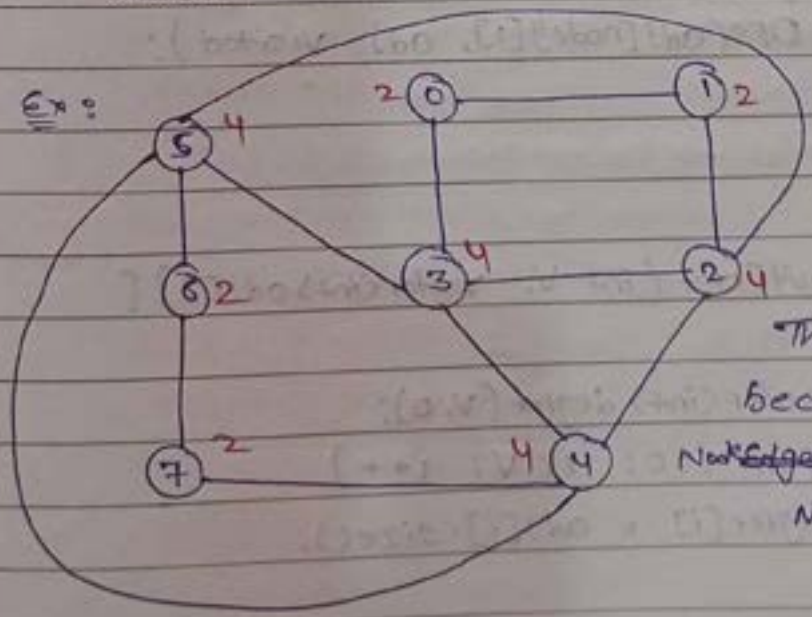


→ Not Euler

Even No. of Edge hone chaiye ek Node pe.

If odd no. of edges, then it can't create Euler Circuit.

Ex:



This is Euler circuit because from every Node ~~Edge~~ it has even No. of Edge.

→ find degree
→ If degree odd → Not an Euler Circuit
→ If all have even degree
→ Traversal the circuit
→ Degree exist : then visited hone chaiye
↳ if visited : Euler circuit
↳ if not visited : Not an Euler Circuit

* Euler path :

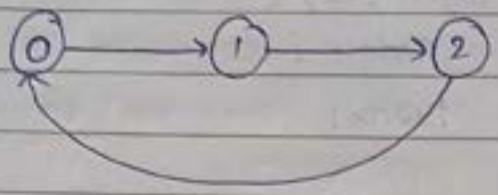
Sare Edge ko cover karna hai.
start != end

Code:

```
Void DFS (int node, Vector<int> adj[], vector <bool> & visited) {
        visited[node] = 1;

    for (int i = 0;  i < adj[node].size() ; i++ ) {
            if (! visited [adj[node][i]] )
            DFS(adj[node][i], adj, visited );
        }
}

bool isEulerCircuitExist (int V, vector<int> adj[]) {

        Vector<int> degree(V, 0);
    for (int i=0;  i< V;  i++)
        degree[i] = adj[i].size();
```

```
        int node = -1;
    for (int i=0; i<V; i++) {
            if (degree[i]) {
                node = i;
                break;
            }
    }
    if (node == -1)
        return 1;

    for (int i=0; i<V; i++)
        if (degree[i]%2)
            return 0;

    vector<bool> visited (V, 0);
    DFS (node, adj, visited);

    for (int i=0; i<V; i++) {
        if (!visited[i] && degree[i])
            return 0;
    }

    return 1;
}
```

Euler Circuit for directed graph:



- indegree of any node = out degree of same node.
- graph is Connected

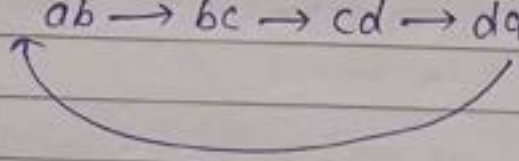
→ then Euler Circuit.

* **Circle of string:**

Ex:

ab, bc, da, cd

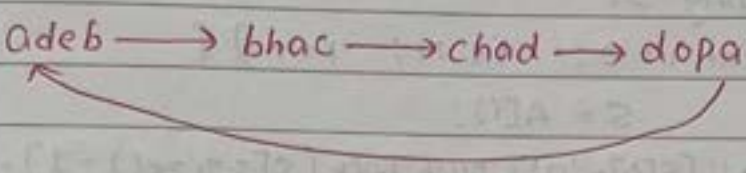
It can combine only if the last character is equal to first character of any other string.



Euler Circuit

Ex:

adeb bhac dopa chad



We only care for first & last character. We don't care about the middle character.

If Euler circuit is created, then it is circle of string.

Approach:

a	b
b	c
c	d
d	a

Code:

```
Void DFS (int node, vector<int> adj[], vector<bool> & visited ) {
        visited[node] = 1;

    for (int i=0; i< adj[node].size() ; i++){
        if ( !visited[adj[node][i]])
        DFS(adj[node][i] , adj, visited);
    }
}

int isCircle (int N, vector<string> A) {

        Vector <int> adj [26];
```

```
vector<int> in(26, 0);
vector<int> out(26, 0);

string s;
for(int i=0; i<N; i++){
    s = A[i];
    adj[s[0]-'a'].push_back(s[s.size()-1]-'a');
    out[s[s.size()-1]-'a']++;
    in[s[0]-'a']++;
}

int node = -1;
for(int i=0; i<26; i++){
    if(in[i] != out[i])
        return 0;
    if(in[i])
        node = i;
}

vector<bool> visited(26, 0);
DFS(node, adj, visited);

for(int i=0; i<26; i++){
    if(in[i] && !visited[i])
        return 0;
}
return 1;
}
};
```

classmate
Date
Page

* Alien Dictionary :

dict : { baa, abcd, abca, cab, cad }

K = 4

baa abcd

b → a

abca cab

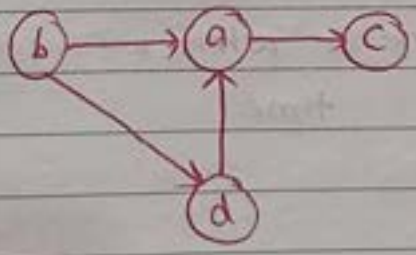
d → a

abca cab

a → c

caba cadd

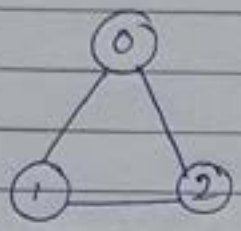
b → d



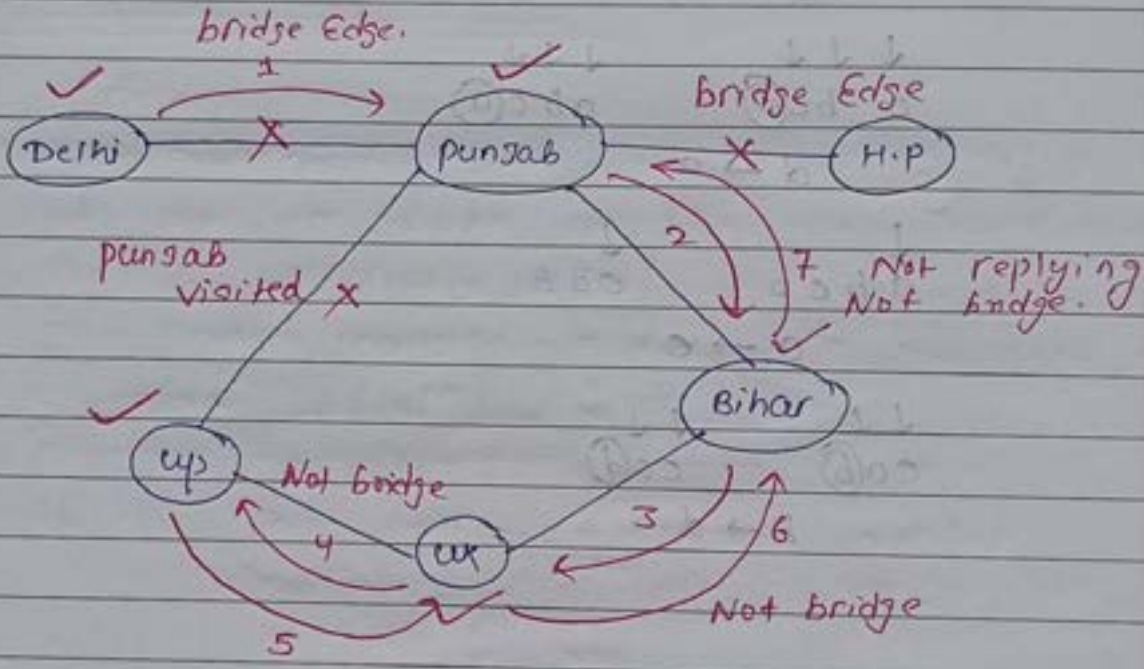
bdac return.

* Critical Connections:

Bridge find karna hai



No any Edge is bridge

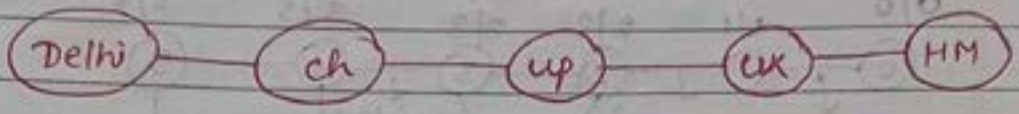


discovered time : when we reach there.

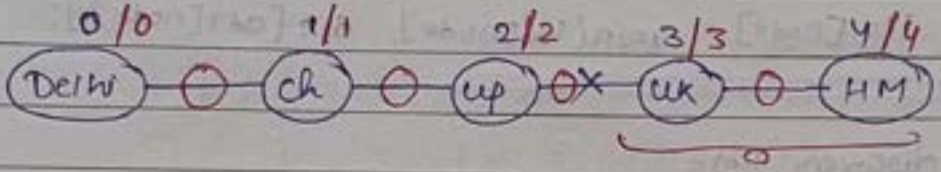
low time : time

Lecture 68

Articulation point



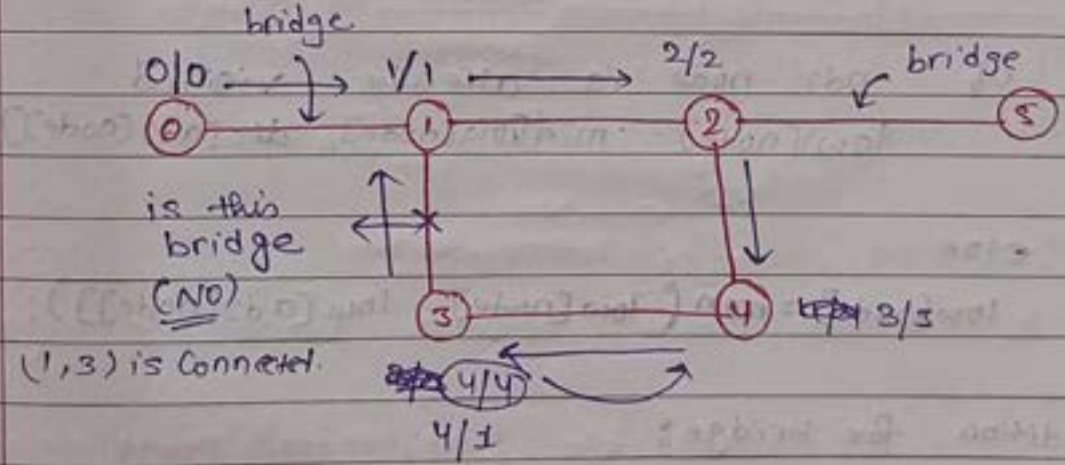
Ek bhi Edge Jo Mere side aa rahi ho
↳ then Not Bridge.



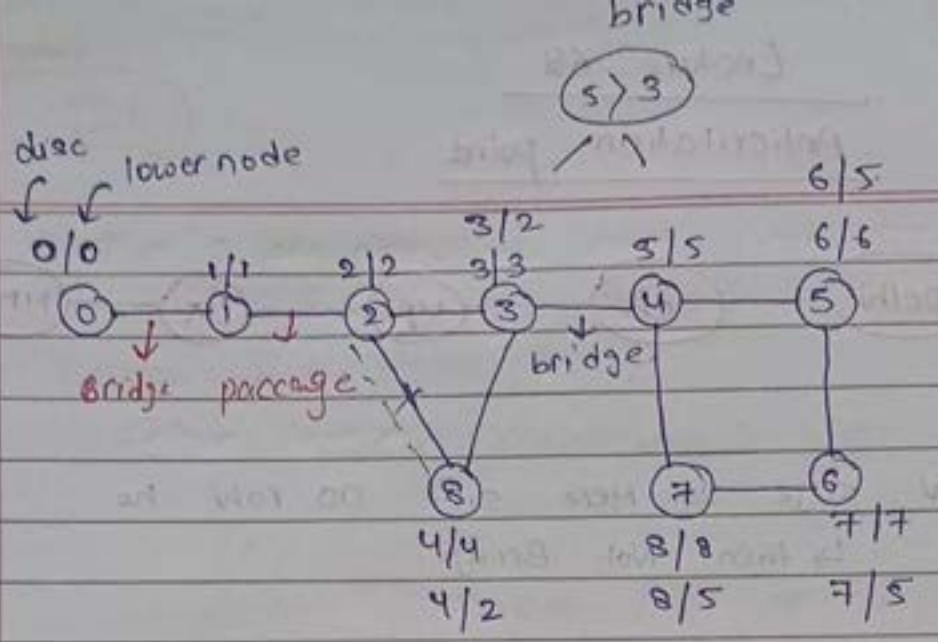
कोई भी Edge 2 से कम है।
edge <= 2

Back edge

There are 4 bridges present.



low[adj[node]] > disc[node].



low[node] = min(low[node], dis[adj[node]];

discovery time

low time.

Initialise discovery & low with (-1).

Then traverse and change discovery & low both to value.

if adj node is already visited

low[node] = min(low[node], dis[adj[node]]);

else

low[node] := min(low[node], low[adj[node]]);

condition for bridge:

if (discovery[node] < low[adj[node]])

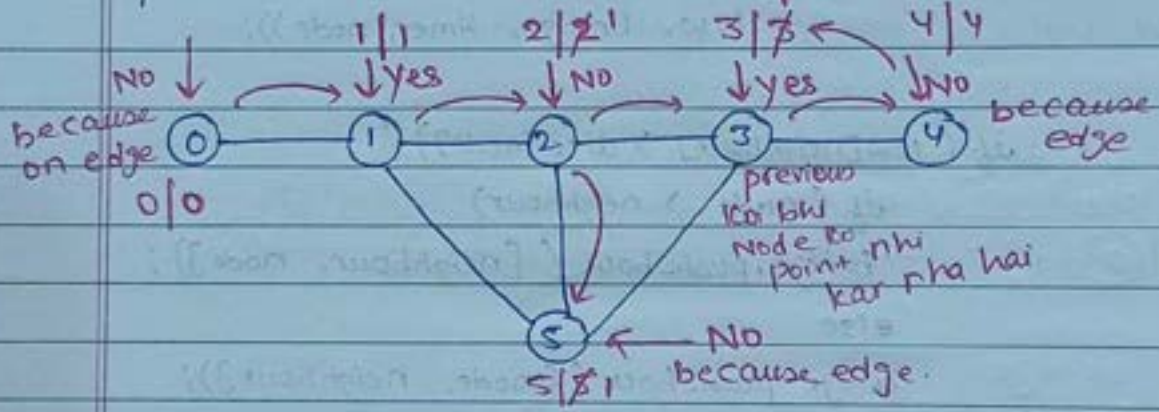
Code:

```
int DFS (int node, Vector<int> adj[], Vector<int> & disc, vector<int> & low,
        Vector <vector<int>> & bridge, int & timer, int & parent) {
            if ( disc[node] != -1)
              return disc[node];

            timer++;
            disc[node] = low[node] = timer;

          for(int i = 0; i < adj[node].size(); i++) {
              int neighbour = adj[node][i];
              if (neighbour == parent)
                Continue;
             low[node] = min(low[node], DFS ( neighbour, adj, disc,
                    low, bridge, timer, node ));

           if ( low[neighbour] > disc[node]){
              if ( node > neighbour)
               bridge.push_back ( {neighbour, node});
              else
              bridge.push_back ({node, neighbour});
          }
    }
  return low[node];
}

Vector<vector<int>> CriticalConnections (int V, vector<int> adj[]){

      vector< Vector < int>> bridge;
      vector<int> disc ( V, -1);
      vector<int> low ( V, -1);
      int timer = -1, parent = -1;
```

```
    DFS(0, adj, disc, low, bridge, timer, parent);
    sort(bridge.begin(), bridge.end());

    return bridge;

  }
};
```

* Articulation point - 1

Ek Node ko hatana hai taki graph 2 ya 2 se jyada part me divide ho jaye, usko Articulation point bolte hai.



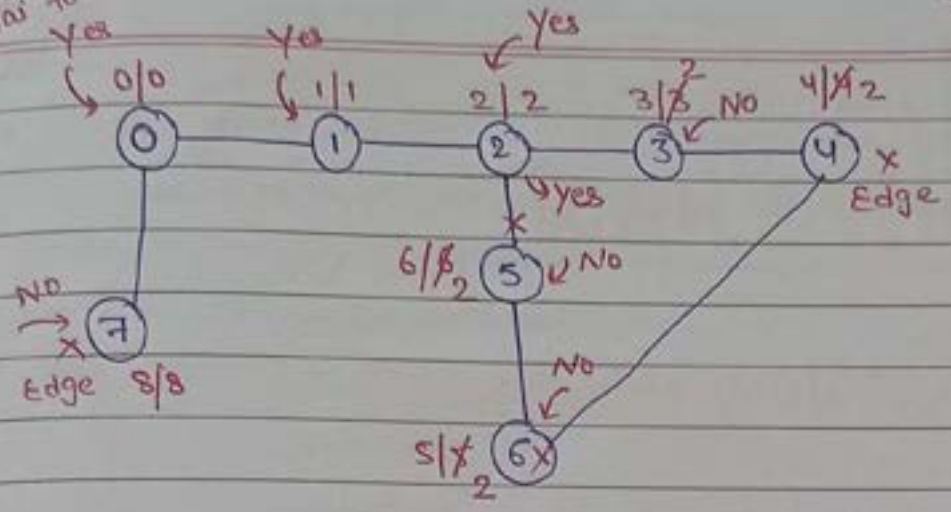
Articulation point hamesa beech me hoga.

Side me kabhi bhi Articulation point nahi hoga.

Articulation point: {1, 3}.

classmate
Date
Page

2 children connected nahi hai to articulation



Lecture : 69

Dynamic programming

→ Jo kaam Ek baar kar chuke hai usko yaad rakho.

→ Fibonacci series :

$f(n)$: 0, 1, 1, 2, 3, 5, 8, 13, 21, - - - ..

→ Dynamic programming problem can solved in two ways :

① Top down Approach + Memoization (Recursion)

② Tabulation (Bottom up approach) (loop)

space optimization.

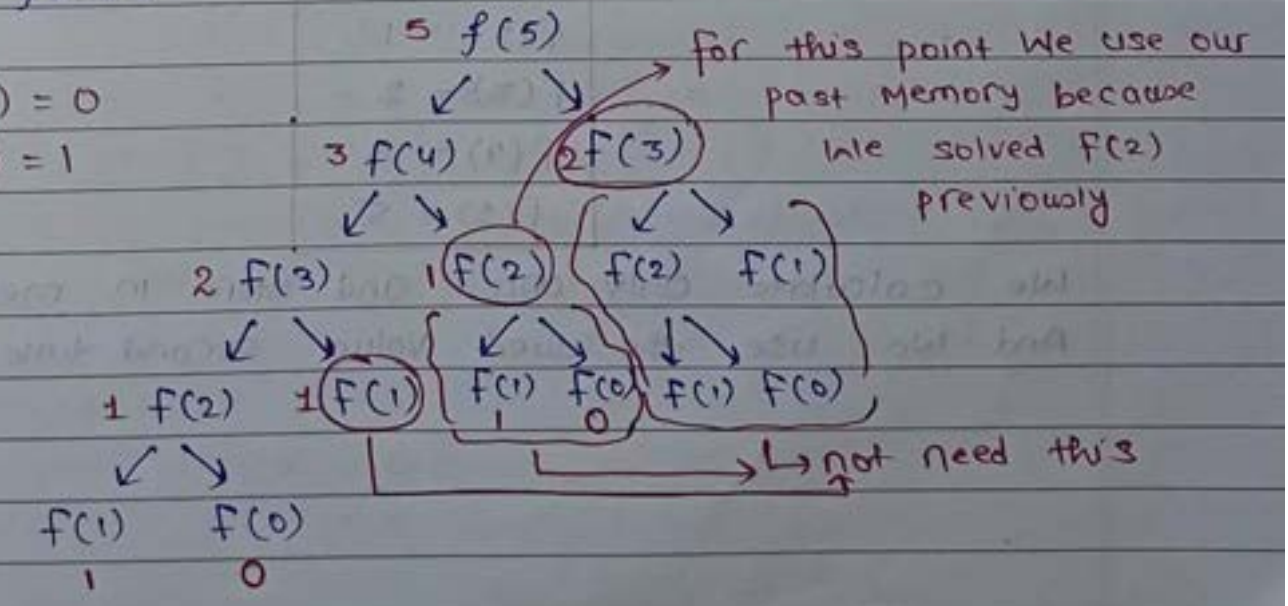
→ n^{th} fibonacci number :

$$f(n) = f(n-1) + f(n-2)$$

$f(5)$:

$f(0) = 0$

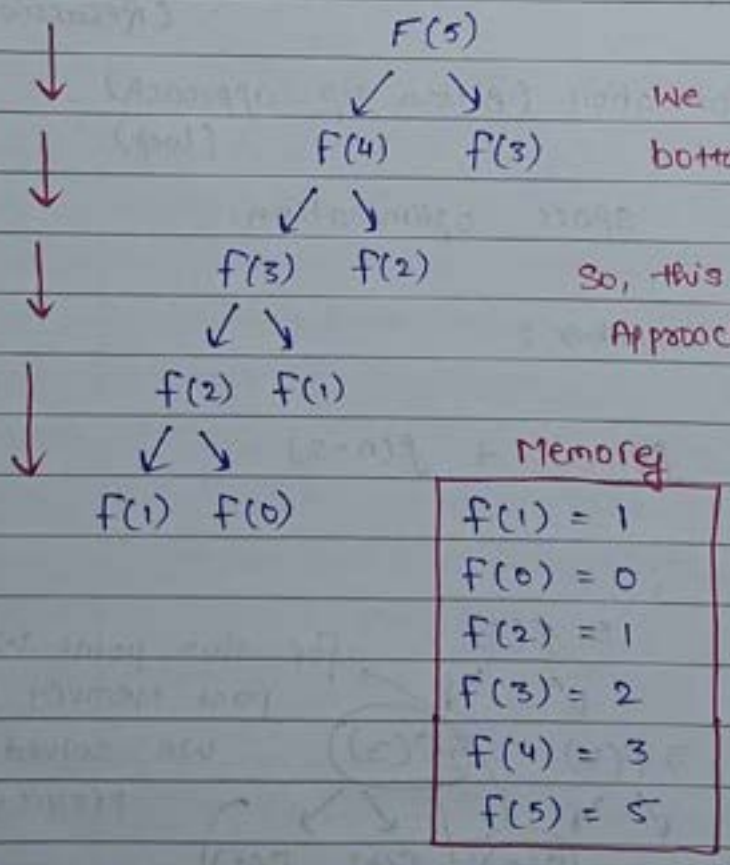
$f(1) = 1$



* previously, We solve all the value
So, Time Complexity
↳ ~~O(n)~~ $O(2^n)$

* Now, With the help of Dynamic progamming one value can solved only one time and store the value in memory and used again in O(1) time.

So, Time Complexity :
↳ O(n)



We go from top to bottom.

So, this is top-down Approach.

Memory

f(1) = 1
f(0) = 0
f(2) = 1
f(3) = 2
f(4) = 3
f(5) = 5

We calculate only once, and store in memory. And We use the used value second time.

→ Divide and Conquer

↳ DP ke problem ko pahle divide karte jao or uske baad usko ek ek kar ke jodte jao.

→ Overlapping subproblem

↳ Jo problem ka part solve kar chuke hai usko dubaro solve nahi karo, Memory me store karwa kar usko direct use kar lo.

Before DP	After DP
2^n	n

For $n = 10$

$2^{10} = 1024$

For $n = 10$

10

See the difference of both

How to store the value which we calculated previously?

Make an array of size $(n+1)$

F(5)

↳ arr[6]

0	1	2	3	4	5
-1	-1	-1	-1	-1	-1

Initially all -1

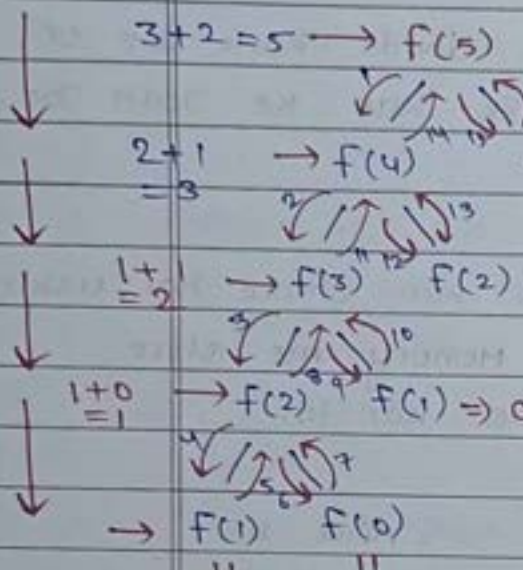
Initially

F(0) = 0

F(1) = 1

classmate
Date
Page

0	1	2	3	4	5
0	1	-X1	-X2	-X3	-X5



Top down Approach:

Code:

```
int find (int n, vector<int> &dp) {
        if ( n <= 1)
            return n;

        if (dp[n] != -1)
            return dp[n];

        dp[n] = find(n-1, dp) + find((n-2), dp);
        return dp[n];
}
int nthFibonacci (int n){

        vector <int> dp (n+1, -1);
        return find (n, dp);

}
```

Space Complexity : O(n) + O(n) ⇒ O(n)

↓ ↓

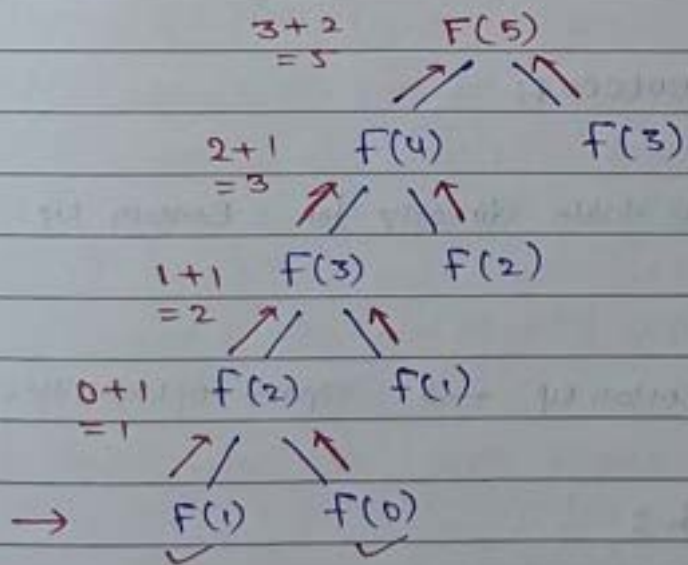
dP Recursive call

Bottom up Approach :

Jitna bhi boottom ke result hai usko solve kar lo.

Make a table of size n+1.

0	1	2	3	4	5
0	1	1	2	3	5



Sabse pahle bottom walo ka → f(0), f(1)

↳ Next f(2) → f(1) + f(0) = 1

↳ Next f(3) → f(2) + f(1) = 2

↳ Next F(4) → f(3) + f(2) = 3

↳ Next f(5) → f(4) + f(3) = 5

→ 0 & 1 ke liye value assign kar denge.

→ 2 se n tak ke liye loop laga denge.

Tabulation Method: (Bottom up Approach)

→ code:

```
int nthFibonacci (int n) {

    vector <int> dp(n+1);
    dp[0] = 0;
    dp[1] = 1;

  for(int i= 2; i<=n; i++){
      dp[i] = (dp[i-1] + dp[i-2])% 1000000007;
    }
  return dp[n]%1000000007;
}
```

We can't create the table directly for Bottom up Approach.

Top down ⟶ Bottom up ⟶ Space optimality.

⇒ Optimise the Approach:

$$f(n) = f(n-1) + f(n-2)$$

f(0)	f(1)	f(2)
0	1	1

↑

for this we required f(0) & f(1)

f(0)	f(1)	f(2)	f(3)
0	1	1	2

↓

for this
we require
f(1) & f(2)
We don't require f(0). so we delete.

We only require 2 previous value. So, we delete the first value in each step.

Code: (Running)

```
int first  = 0;
int second = 1;
int result;
int third;
for (int i = 2;  i<=n;  i++) {
        third  = (first + second) % 1000000007;
        first  = second;
        second = third;
}
result = third % 1000000007;
return result;
```

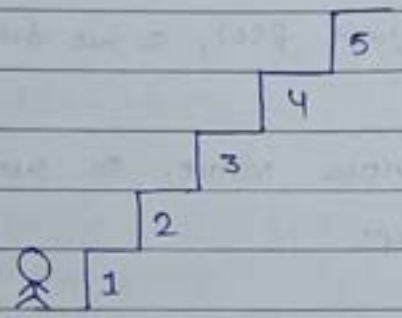
→ on this we only store the two previous value.

Space Complexity : O(1)
Time Complexity : O(N)

* Climbing stairs : (Leetcode)

Example :

N = 5



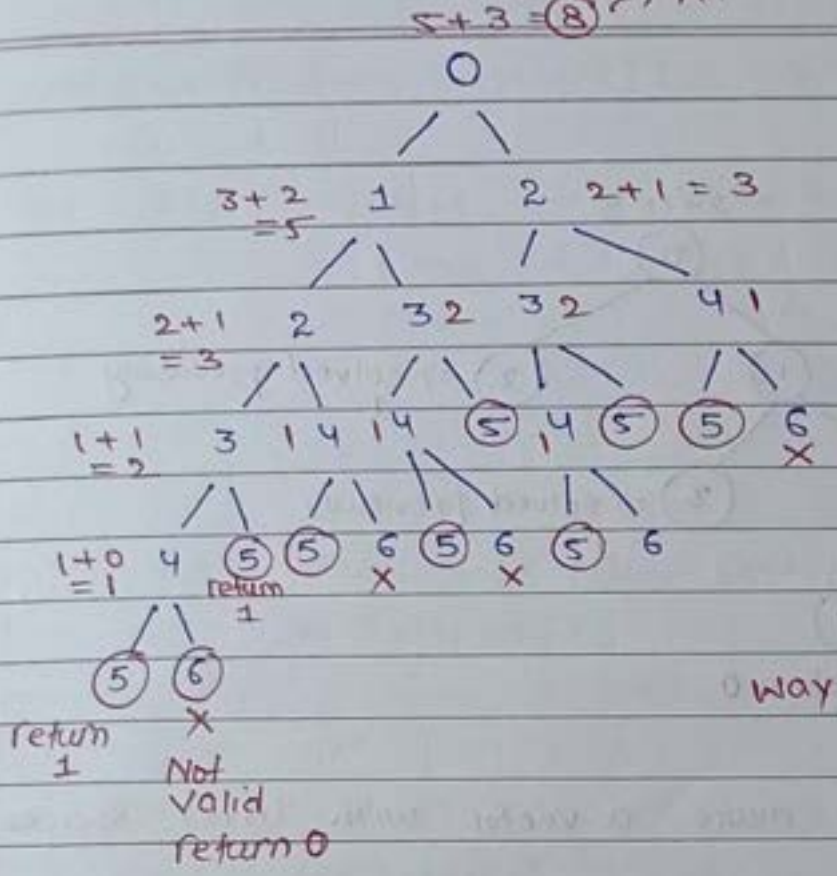
⇒ At 1 time take 1 step or 2 step

⇒ Find total No. of ways to reach on top ?

→ 1 step + 1 step + 1 step + 1 step + 1 step
→ 1 step + 1 step + 1 step + 2 step
→ 1 step + 1 step + 2 step + 1 step
→ 1 step + 2 step + 1 step + 1 step
→ 2 step + 1 step + 1 step + 1 step
→ 1 step + 2 sep + 2 step
→ 2 step + 1 step + 2 step
→ 2 step + 2 step + 1 step

Total = 8 ways.

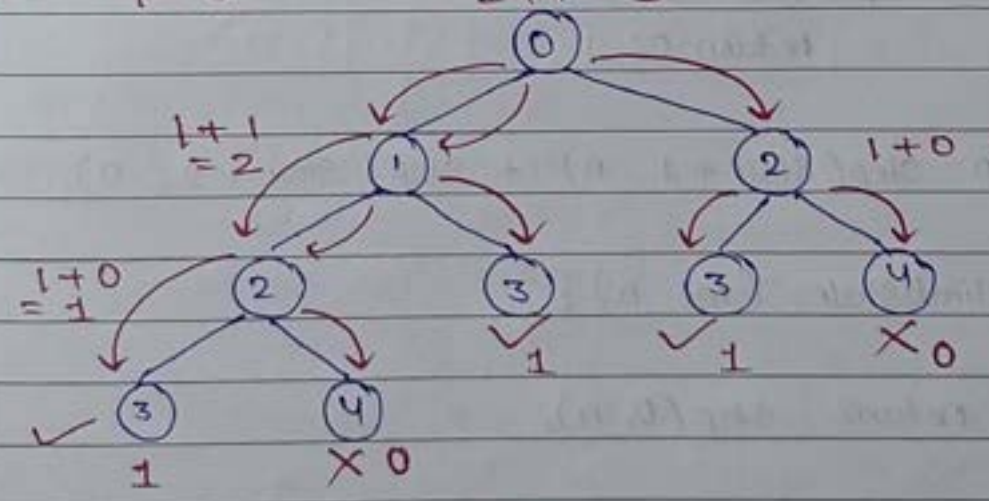
We can understand this using graph of stairs.



Ways = 8

```
n == 5
return 1;
n > 5
return 0;
```

Let $n = 3$



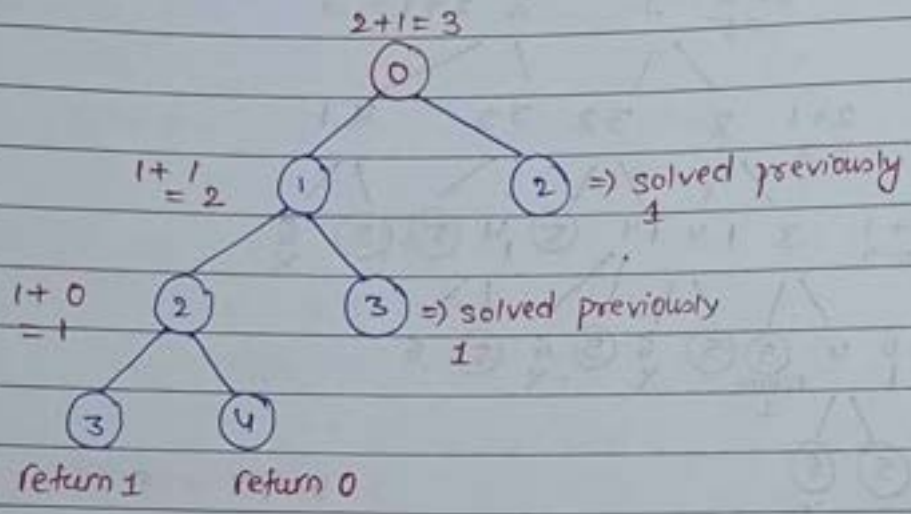
$$f(n) = f(n+1) + f(n+2)$$

$n \rightarrow$ return 1

$> n \rightarrow$ return 0

Optimise :

n = 3



In this we make a vector with (n+1) spaces.

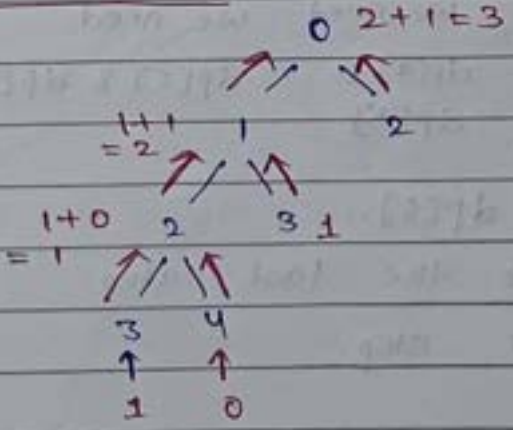
Code (By recursion) :

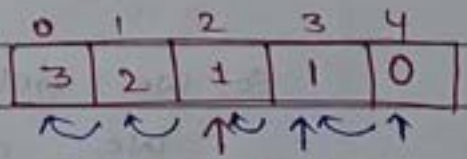
```
int step (int stair, int n) {
        if (stair == n)
            return 1;
        if (stair > n)
            return 0;

    return step(stair + 1, n) + step(stair + 2, n);
}
int climbStairs (int n) {

    return step(0, n);
}
```

Code (By Top down Approach) :

```
int step (int stair, int n, vector<int> &dp) {
        if (stair == n)
            return 1;

        if (stair > n)
            return 0;
    // Already calculated result return
        if (dp[stair] != -1)
            return dp[stair];

    dp[stair] = step(stair+1, n, dp) + step(stair+2, n, dp);
     return dp[stair];
}

int climbStairs (int n) {
      vector <int> dp (n+2, -1);
      dp[n] = 1,  dp[n+1] = 0;
      return   step (0, n, dp);
  }
```

Bottom up Approach :





f(3)+f(4)
= 1

f(1) = f(2) + f(3)
= 2

f(0) = f(1) + f(2) = 3

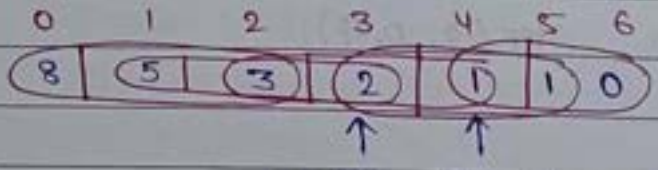
O(N) → space Complexity

Code (Bottom up Approach):

```
int climbStairs (int n) {
        Vector <int> dp(n+2);
        dp[n] = 1;
        dp[n+1] = 0;

    for (int i = n-1; i >= 0; i--)
        dp[i] = dp[i+1] + dp[i+2];

    return dp[0];

}
```

Time Complexity : O(N)
Space Complexity : O(N).

* optimise space Complexity :



For this we need dp[4] & dp[5]

For this we need dp[5] & dp[6]

So, we delete dp[6].
We delete the last step in every step.

classmate
Date
Page

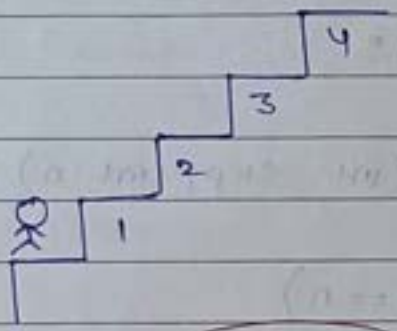
* Count Number Of Hops :

At one frog can jump either one, two or three steps.

1 or 2 or 3

Example:

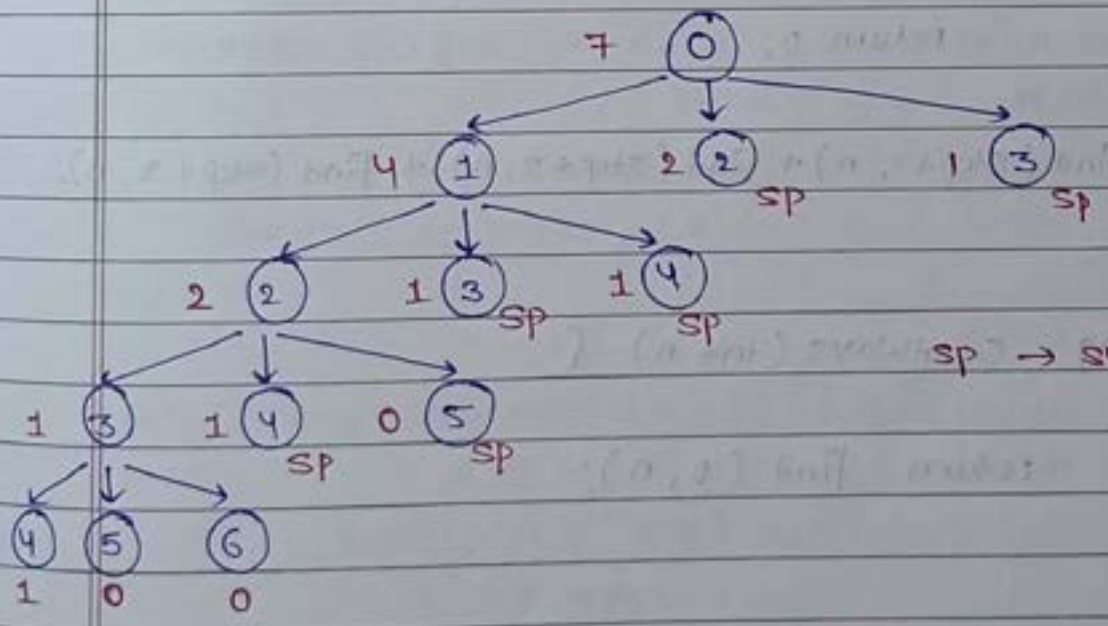
N = 4



Total = 7

possible :

1 + 1 + 1 + 1
1 + 1 + 2
1 + 2 + 1
2 + 1 + 1
2 + 2
1 + 3
3 + 1



sp → solved previously.

For n → return 1
For n+1 → return 0
For n+2 → return 0

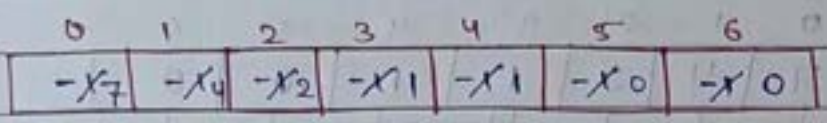
classmate
Date
Page

```
if (step == n)
    return 1

if (step > n)
    return 0;

return For 1 step + For 2 step + For 3 step;
```

Code (by recursion):

```
long long find (int step, int n) {

    if ( step == n)
        return 1;

    if ( step > n)
        return 0;

return find (step+1, n) + find (step+2, n) + find (step+3, n);
}

long long countWays (int n) {

    return find (0, n);
}
```

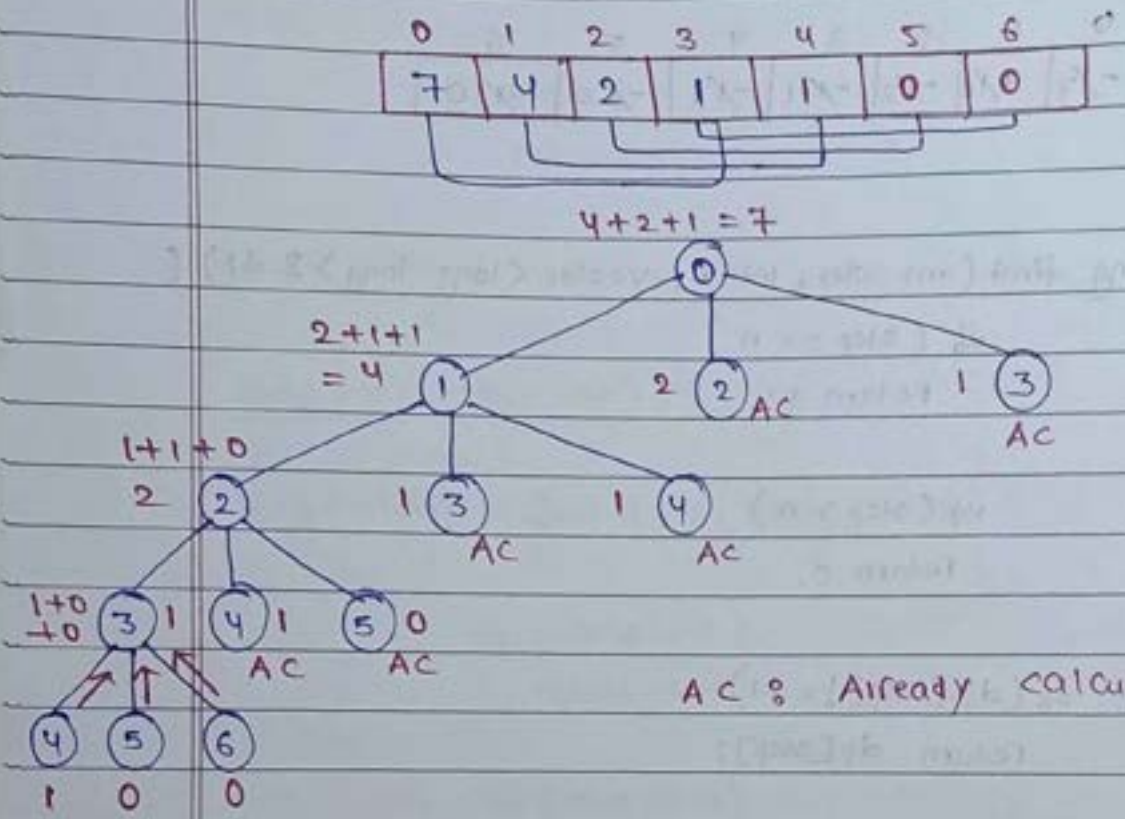
Code (by top down Approach) :



code :

```
long long find (int step, int n, vector <long long> & dp) {
        if (step == n)
            return 1;

        if (step > n)
            return 0;

        if (dp[step] != -1)
            return dp[step];

dp[step] = find (step+1, n, dp) + find (step+2, n, dp) +
                                  find (step+3, n, dp);
   dp[step] %= 1000000007;
   return dp[step];
}

long long countWay (int n) {

        vector <long long> dp (n+3, -1);
        dp[n] = 1;
        dp[n+1] = 0;
        dp[n+2] = 0;

        return find (0, n, dp);

}
```

Bottom up Approach:



AC : Already calculated.

code :

```
long long countWays(int n){

    vector<long long> dp(n+3);
    dp[n] = 1;
    dp[n+1] = 0;
    dp[n+2] = 0;

for(int i= n-1; i>=0; i--){
    dp[i] = (dp[i+1] + dp[i+2] + dp[i+3]) % 1000000007;
}
return dp[0];
}
```

Lecture - 70

House Robber Coin Change

* House Robber (Leetcode)

You don't enter into two adjacent house, if you enter the alarm starts ringing.

You have an array "nums" representing the amount of money at each house.

Return the maximum amount of money you rob tonight without ringing the alarm.

nums = { 1, 2, 3, 1 }

possible Combination :

1 + 3 = 4 (Max) return
1 + 1 = 2
2 + 1 = 3

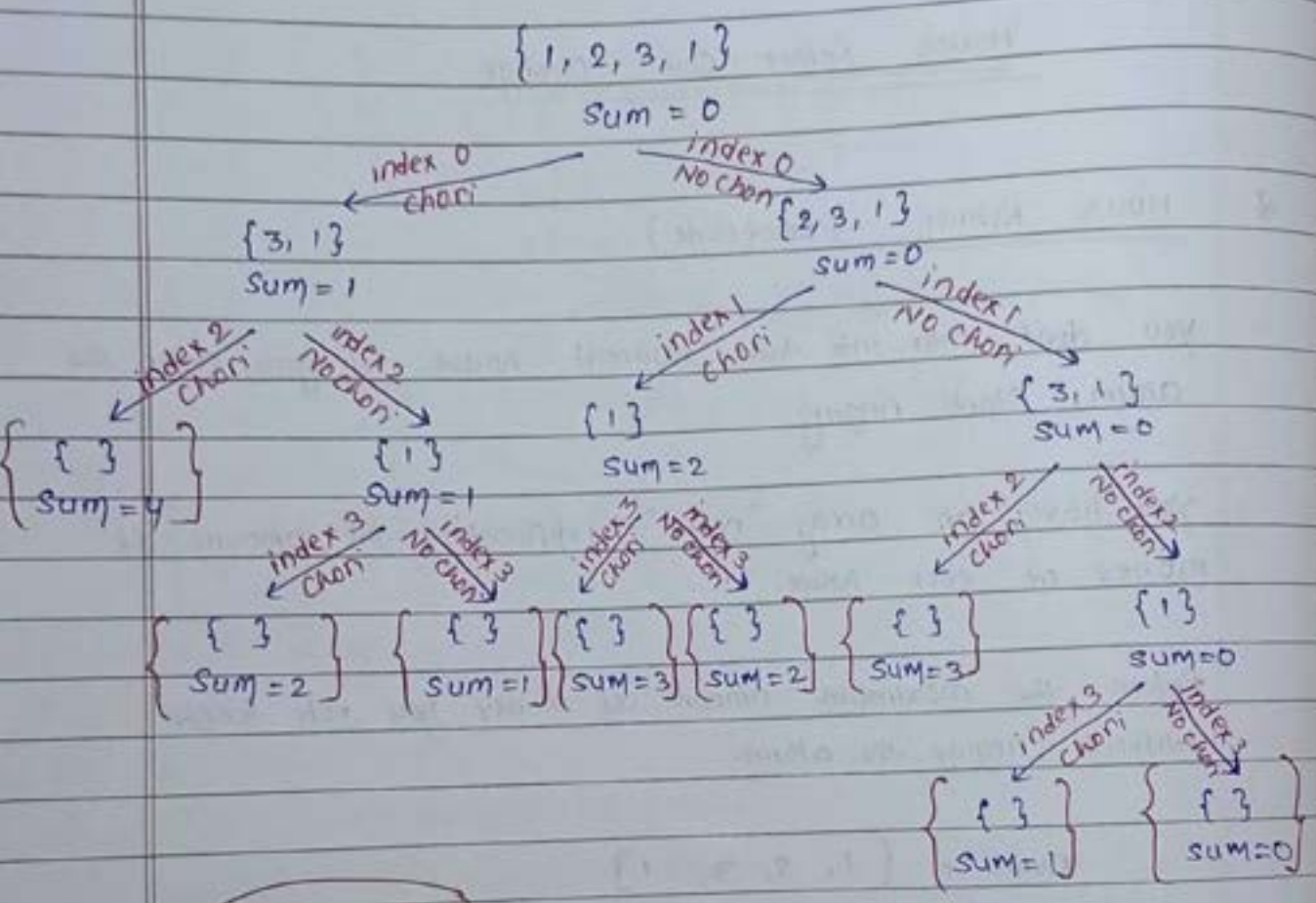
We can understand this with help of tree.

We start with all houses.

→ 1st house se chori kar liye to uske baad wala hata denge option se.

By Recursion : House :

0	1	2	3
1	2	3	1



Max = 4

return 4.

nums = {1, 2, 3, 1}

House = {1, 2, 3, 1}

find (index + 1)

↳ Agar chori nahi kiye to

(House[index] + find(index + 2))

↳ Agar chori kar liye to

Dono me se jo max ayega wo answer hoga.

Base case :

if (index == n)

return 0;

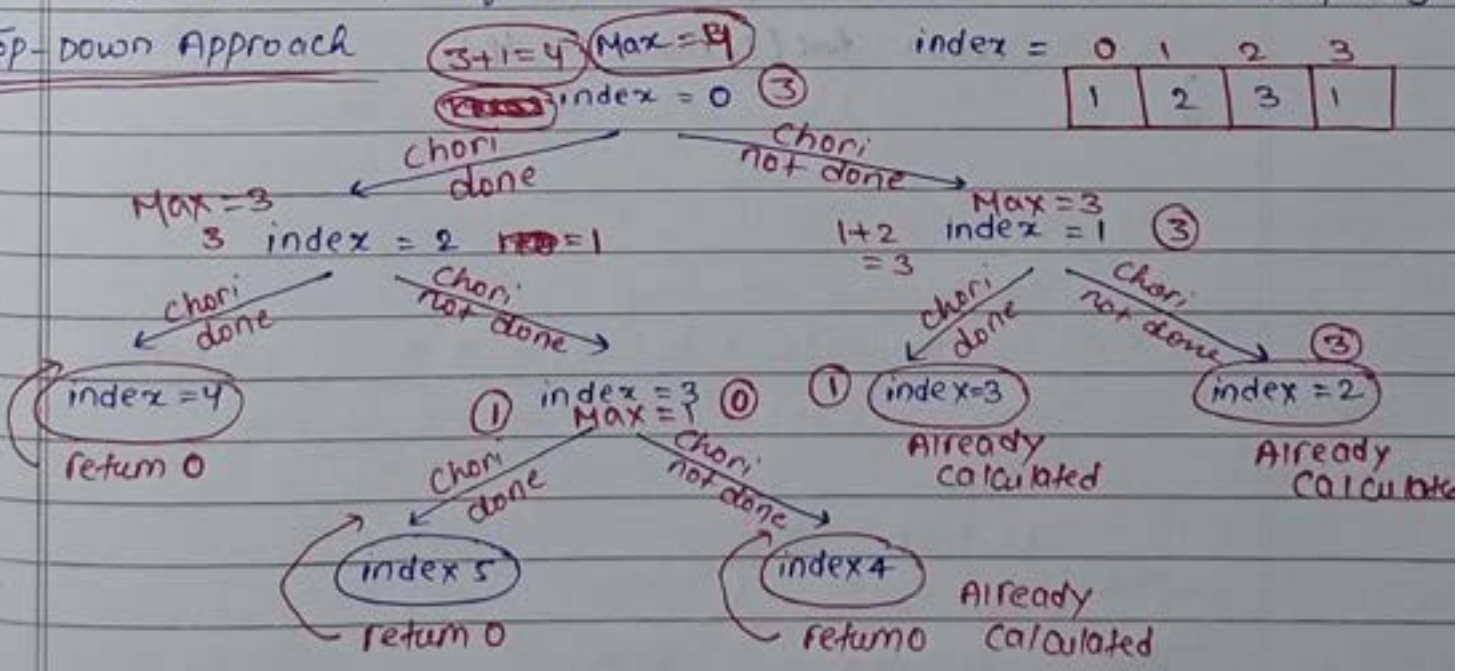
Code :

```
int find (int index, vector<int> &nums, int n) {

        if (index >= n)
            return 0;

    return max (nums[index] + find (index+2, nums, n),
                find (index+1, nums, n));

}

int rob (vector<int> &nums) {
        int n = nums.size();
        return find (0, nums, n);
}
```

Time Complexity : $O(2^n)$ (TLE)

By the help of DP we can decrease the Time Complexity.

Top-Down Approach



Code : (Top down Approach)

```
int find(int index, vector<int>& nums, int n, vector<int>& dp){

        if(index >= n)
            return 0;

        if(dp[index] != -1)
            return dp[index];

return dp[index] = max(nums[index] + find(index+2, nums, n, dp),
                       find(index+1, nums, n, dp));

}

int rob(vector<int>& nums){
        int n = nums.size();
        vector<int> dp(n+2, -1);
        return find(0, nums, n, dp);
}
```

Bottom up Approach:

	0	1	2	3
House :	1	2	3	1

	0	1	2	3	4	5
Dp :	4	3	3	1	0	0

$Dp[n] = 0$

$Dp[n+1] = 0$

For $Dp[3]$:

$House[3] + Dp[5] = 1$ Max

$Dp[4] = 0$

For $Dp[2]$

$House[2] + Dp[4] = 3$ Max

$Dp[3] = 1$

For $Dp[1]$

$House[1] + Dp[3] = 3$ Max

$Dp[2] = 3$

For $Dp[0]$

$House[0] + Dp[2] = 1 + 3 = 4$ Max

$Dp[1] = 3 = 3$

return $Dp[0]$

Time Complexity : $O(N)$

space Complexity : $O(N)$

classmate
Date
Page

Code :

```
int rob (vector <int> & nums) {
        int n = nums.size();
        vector <int> dp(n+2);
        dp[n] = dp[n+1] = 0;
        for (int i = n-1;  i >= 0;  i--)
          dp[i] = max (nums[i] + dp[i+2] , dp[i+1]);

    return dp[0];
}
```

* More optimised in space complexity :
We doesn't require full size of vector dp.
We only need two value in each step.

So, we don't make any dp vector.

Goal : space complexity : O(1)

dp[n] → First
dp[n+1] → second

```
ans = max (nums[i] + second, first);
        second = first;
        first = ans;
```

Time Complexity : O(N)

Space Complexity : O(1)

Code :

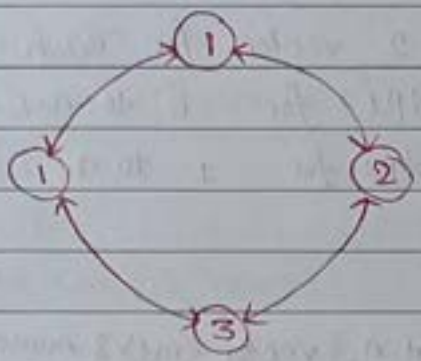
```
int rob(vector<int>& nums){
        int n = nums.size();

        int first = 0, second = 0;
        int ans;

      for(int i=n-1; i>=0; i--){
          ans = max(nums[i]+second, first);
           second = first;
           first = ans;
      }

   return ans;
}
```

classmate
Date
Page

* House Robber II (Leetcode)

The last house is connected with first house in circular form.

0	1	2	3
1	2	3	1



On this, we give an extra condition here:

Ya to 0 se n-1 tak kar lo ya 1 se n tak

code: (Recursion)

```
int find (int index, int n, vector <int> & nums) {
        if (index >= n)
            return 0;

    return max (nums [index] + find (index + 2, n, nums),
                find (index + 1, n, nums));
}
int rob (vector <int> & nums) {
        int n = nums.size();
    return max (find (0, n-1, nums), find (1, n, nums));
}

{ if (n == 1)
    return nums[0];
```

Top Down Approach :

We find in 2 ways :

1. 0 to n-1
2. 1 to n

So, We make 2 vector for each.
1 vector dp1 for 0 to n-1
2nd Vector dp2 for 1 to n.

Code :

```
int find(int index, int n, vector<int>& nums, vector<int>& dp){
        if (index >= n)
            return 0;

        if (dp[index] != -1)
            return dp[index];

return dp[index] = max(num[index] + find(index+2, n, nums, dp),
                       find(index+1, n, nums, dp));
}
int rob(vector<int>& nums){
        int n = nums.size();
        if(n==1)
            return nums[0];

        vector<int> dp1 (n+2, -1);
        vector<int> dp2 (n+2, -1);

return max(find(0, n-1, nums, dp1), find(1, n, nums, dp2));
}
```

Bottom up Approach:

	0	1	2	3
nums :	1	2	3	1

	0	1	2	3	4	5
dp1 :	4	3	3	0	0	-1

	0	1	2	3	4	5
dp2 :	-1	3	3	1	0	0

dp1[0], dp2[1]

(4, 3) → Max = 4 Ans

Code :

```
int rob(vector<int>& nums){
    int n = nums.size();
    if(n==1)
    return nums[0];

    vector<int> dp1(n+2, -1);
    vector<int> dp2(n+2), -1);

dp1[n-1] = dp1[n] = dp2[n] = dp2[n+1] = 0;

for(int i=n-2; i>=0 -1; i--)
 dp1[i] = max(nums[i] + dp[i+2], dp[i+1]);

for(int i=n-1; i>0; i--)
 dp2[i] = max(nums[i] + dp[i+2], dp[i+1]);

return max(dp1[0], dp2[1]);
}
```

T.C : O(N)
S.C : O(N)

* Optimise the Space Complexity:

dp1[n-1] → first1
dp1[n] → second1

dp2[n] → first2
dp2[n+1] → second2

ans1 = max(nums[i] + second1, first1);

ans2 = max(nums[i] + second2, first2);

second1 = first1
first1 = ans1

second2 = first2
first2 = ans2.

In this, we don't require any vector and by this we can minimise our space Complexity.

In top-down approach and bottom up Approach we use vector.
But we need only 2 values each time from vector.

So, we can't use vector, in place of vector we use 2 variable.

Space Complexity: O(1)

Code :

```
int rob(vector<int>& nums) {
        int n = nums.size();
        if(n==1)
          return nums[0];

        int first1 = 0;
        int second1 = 0;
        int first2 = 0;
        int second2 = 0;
        int ans1, ans2;

    for(int i=n-2; i>-1; i--){
        ans1 = max(nums[i] + second1, first1);
            second1 = first1;
            first1 = ans1;
    }

    for(int i=n-1; i>0; i--){
        ans2 = max(nums[i] + second2, first2);
                second2 = first2;
                first2 = ans2;
    }

    return max(ans1, ans2);
}
```

classmate
Date
Page

* ## Coin change II (Leetcode)

Given : Integer array
↳ Coins of different denomination

Amount
↳ Total amount of money

Return : Number of Combination that make up that amount.

If not possible → return 0.

Assume you have infinite Number of each kind of Coin.

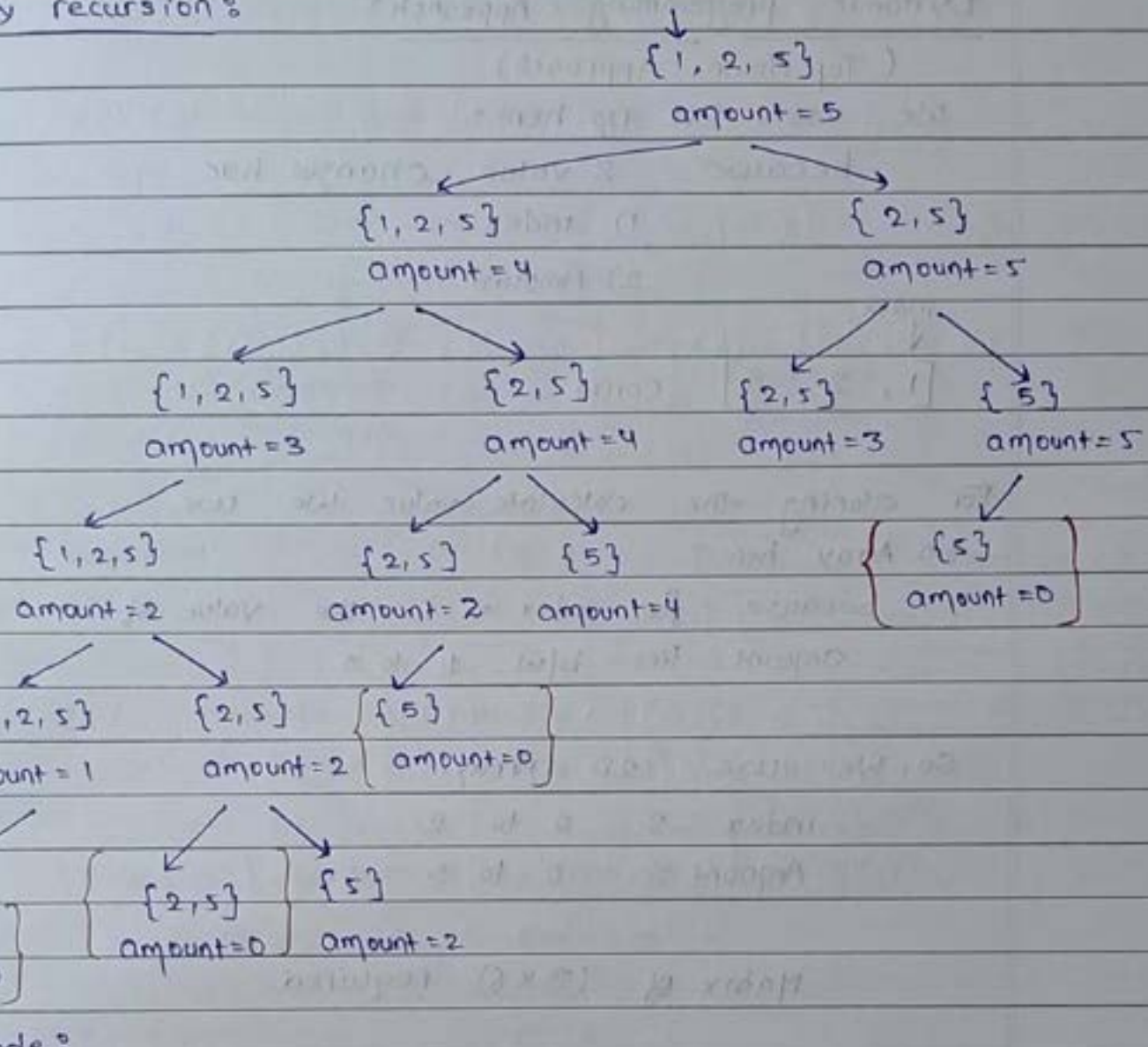
Example 1 :

Coins : {1, 2, 5}
Amount : 5

possible : 5 = 5
5 = 2 + 2 + 1
5 = 2 + 1 + 1 + 1
5 = 1 + 1 + 1 + 1 + 1

Total = 4 possibilities.

By recursion:



Code:

```
int find (int index, int amount, vector <int>& coins, int n) {
        if (amount == 0)
            return 1;
        if (amount < 0 || index >= n)
            return 0;
return find (index, amount - coins[index], coins, n) + find (index + 1,
            amount, coins , n);
}
int change (int amount, vector <int>& coins) {
        int n = coins.size();
    return find( 0, amount, coins, n);
}
```

classmate
Date
Page

Dynamic programming Approach:

(Top down Approach)

We use 2D Dp here:

because 2 value changes here

1) index
2) Amount

index ↓

[1, 2, 5] coins

For storing the calculate value we use 2D Array here:

because for index = i, the value of amount lies b/w 1 to 5.

So, we use 2D Array.

index : 0 to 2

Amount : 0 to 5

Matrix of (3 x 6) required.

Index : 0 - n => (n+1)

Amount : amount + 1

2D Dp of size : (n+1) * (amount+1)

We understand with an Example:

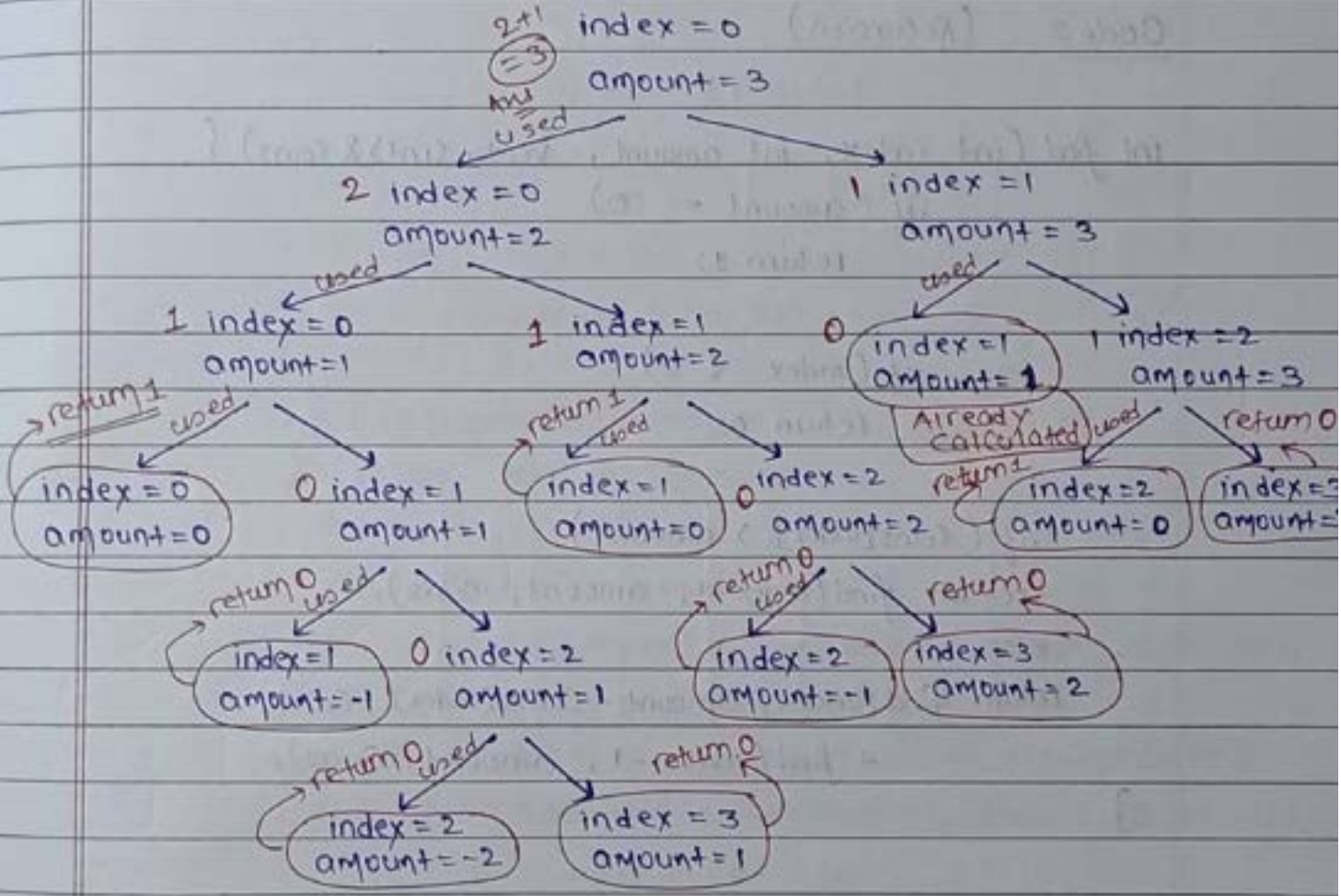
Coins : [1, 2, 3]

amount = 5

classmate
Date
Page

0	1	2
1	2	3

amount = 3



• Time Complexity : $O(n+1)(amount+1)$

→ If the Coins array is unsorted, Make it sorted
↳
then try to solve the problem
↳
otherwise, it gives ~~error~~ (orrect answer

Note:

* You can solve problem with both sorted & unsorted.

classmate
Date
Page

Lecture - 71

Coin Change II

Code: (Recursion)

```
int find (int index, int amount, vector<int>& coins) {
        if (amount == 0)
            return 1;

        if (index < 0)
            return 0;

        if (coins[index] > amount)
        return find(index-1, amount, coins);
      else
      return find(index, amount - coins[index], coins)
                + find(index-1, amount, coins);
}

int change (int amount, vector<int>& coins) {
        int n = coins.size();
        return find(n-1, amount, coins);
    }
```

classmate
Date
Page

Code : (Top down Approach)

```
int find (int index, int amount, vector<int> &Coins,
                vector<vector<int>> &dp) {

        if (amount == 0)
            return 1;

        if (index < 0)
            return 0;

        if (dp[index][amount] != -1)
            return dp[index][amount];

        if ( Coins [index] > amount )
        return dp[index][amount] = find (index -1, amount, Coins, dp);
        else
        return dp[index][amount] = find (index, amount-Coins[index],
                Coins, dp) + find (index -1, amount, Coins, dp);

}

int change ( int amount, vector <int> & Coins) {
        int n = Coins.size();

 vector <vector<int>> dp (n+1, vector<int> (amount +1, -1));
    return find ( n-1, amount, Coins, dp);
}
```

Bottom up Approach :

→ check first Base cases

```
if (amount == 0)
    return 1;
```

0	1	2
1	2	5

Amount = 5

dp[n+1][amount+1]

amount

coins	0	1	2	3	4	5
0	1	0	0	0	0	0
1	1	1	1	1	1	1
2	1	1	2	2	3	3
3	1	1	2	2	3	4

```
if (Coins[index - 1] > amount)
    find (index - 1, amount, Coins)

else
    find (index, amount - Coin[index - 1], Coins)
        + find (index - 1, amount, Coins);
```

Size : (n+1) * (amount+1)

classmate
Date
Page

Lecture 72

Coin Change II :
(Bottom up approach)

(i) \ (j)	0	1	2	3	4	5
(coins) 0	1	0	0	0	0	0
(1) 1	1	1	1	1	1	1
(2) 2	1	1	2	2	3	3
(5) 3	1	1	2	2	3	4

(amount)

```
if (index == 0)
    return 0;
```

```
if (Coins[i-1] > amount)
    dp[i][j] = dp[i-1][amount];
else
    dp[i][j] = dp[i][j - Coin[i-1]] + dp[i-1][amount];
```

→ i = 1, j = 1 (j = amount)

0 1 2
{1, 2, 5}

Coins[1-1] > amount

1 > 1 (False)

else

↳ dp[1][1] = dp[1][0] + dp[0][1] = 1 + 0 = 1

→ i = 2, 1, j = 2

Coins[1-1] > amount

1 > 2 (false)

dp[1][2] = dp[1][1] + dp[0][2] = 1

By this, we solve all value.

Code : Bottom up Approach

```
int change (int amount, vector<int>& coins) {

    int n = coins.size();

    vector<vector<int>> dp(n+1, vector<int>(amount + 1, 0));

    for (int i=0; i<=n; i++) {
        dp[i][0] = 1;

    for (int i=1; i<=n; i++) {
        for (int j=1; j<=amount; j++) {

            if (coins[i-1] > j)
            dp[i][j] = dp[i-1][j];
            else
            dp[i][j] = dp[i][j] - coins[i-1] + dp[i-1][j];
        }
    }
    return dp[n][amount];
}
```

Time Complexity : O(n+1)(amount+1)

Space Complexity : O(n+1)(amount+1)

Optimise the space Complexity :

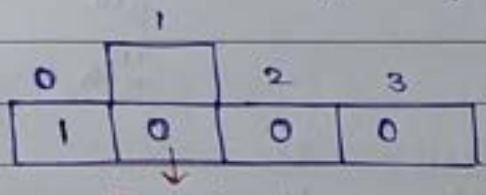
We have to fill the 2D array and we need the last value as the answer.

For finding any value of 2D DP. We require only two rows for finding value.

① previous row
② Current row

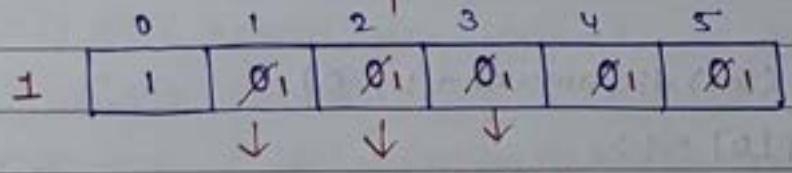
We want to optimize more :

In previous row, We only need one value which is upper of my value (Current).



For this we require that row and a upper block.

(2-1) = ① → 1 wala add kar do

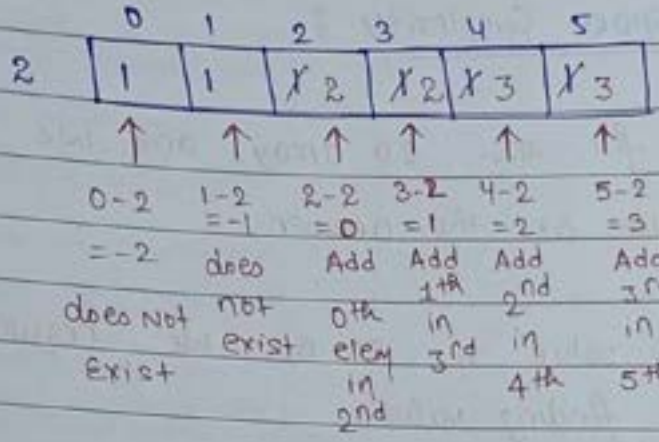


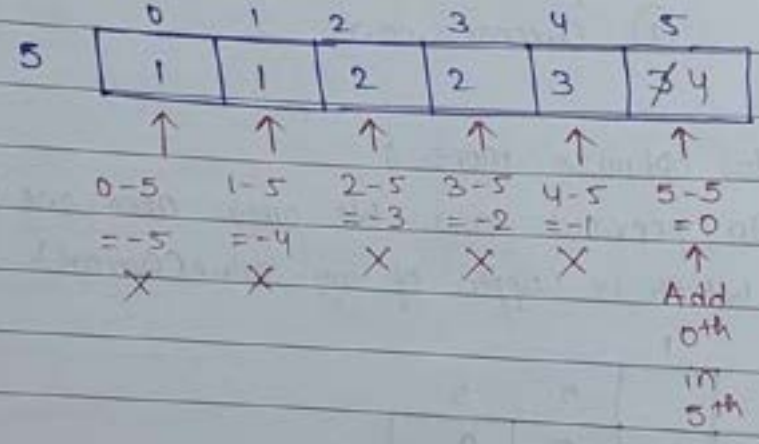
1 rupee ke Coin se 1 rupya bnnaha kitna possible ways hai

(3-1) = 2 → 2 wala add kar do

3 - 1 = (2)
↓ ↓
rupee Coin
↳ rupee (value)
↳ add in 3 Col.

classmate
Date
Page





Final answer = 4

code :

```
int change (int amount, vector<int> & Coins) {
        int n = Coins.size();

    vector <int> dp (amount + 1, 0);
        dp[0] = 1;

    for (int i = 1; i<=n; i++) {
        for (int j = Coins[i-1]; j<=amount; j++) {
            dp[j] += dp[j - Coins[i-1]];
        }
    }

    return dp[amount];
}
```

* Count Ways to N'th Stair (Order does not matter)

→ There are N stairs.

→ person can climb either 1 stair or 2 stair at a time.

→ Here {1, 2, 1}, {2, 1, 1}, {1, 1, 2} are considered as same.
Because ofrequency of 1 & 2 are same in all.

Code: Recursion

```
int find(int index, int n, int step[]) {
        if (n == 0)
          return 1;

        if (index == 0)
          return 0;

        if (step[index - 1] > n)
        return find(index - 1, n, step);
        else
        return find(index, n - step[index - 1], step) +
                find(index - 1, n, step);
}
int nthStair(int n) {

        int step[2] = {1, 2};
        return find(2, n, step);

}
```

On this problem

↳ 2 values changes

↳ index & n

So, 2D DP required here.

Top Down Approach (code) :

```
int find (int index, int n, int step[], vector<vector<int>>& dp) {
        if (n == 0)
        return 1;

        if (index == 0)
        return 0;

        if (dp[index][n] != -1)
        return dp[index][n];

        if (step[index-1] > n)
        return dp[index][n] = find (index-1, n, step, dp);
        else
        return dp[index][n] = find (index, n-step[index-1],
            step, dp) + find (index-1, n, step, dp);
    }
    int nthStair (int n) {
        int step[2] = {1, 2};
        vector<vector<int>> dp (3, vector<int> (n+1, -1));
        return find (2, n, step, dp);
    }
```

classmate
Date
Page

Bottom up Approach : (code)

```
int nthStair (int n) {

    int step[2] = {1, 2};
    vector<vector<int>>dp (3, vector<int>(n+1, 0));

    for (int i = 0; i < 3; i++) {
        dp[i][0] = 1;
    }

    for(int i = 1; i <= 2; i++) {
        for (int j = 1; j <= n; j++) {
            if (step[i-1] > j)
            dp[i][j] = dp[i-1][j];
            else
            dp[i][j] = dp[i][j-step[i-1]] + dp[i-1][j];
        }
    }
    return dp[2][n];
}
```

→ Optimised Code

```
return 1 + n/2;
```

step	ways.
1	1
2	2
3	2
4	3
5	3
6	4
7	4
8	5
9	5

classmate
Date
Page

* Fractional knapsack :

Given : Weight and values of N items

Task : put the items in knapsack to get Maximum total value.

0/1 knapsack means, item will be keep totally.

You ~~dosnt~~ allow to break the item.

Example : 1

N = 3 W = 50

Values [] = { 60, 100, 120 }
Weight [] = { 10, 20, 30 }.

Valuesperunit [] = { 6, 5, 4 }

Take first the maximum from valuesperunit

(1) 6

Weight = 10

10 < 50

ans = 6 x 10 = 60

W = 50 - 10 = 40

(2) Next larger = 5

Weight = 20

20 < 40

ans = 60 + 20 x 5

= 160

$W = 40 - 20 = 20$

(3) next larger = 4

weight = 30

$30 > 20$

We take only 20 kg

ans = $160 + 4 \times 20$

$= 240.00.$

↳ ans.

* Knapsack 0-1

We allowed to take either full weight of an item or we cannot take that item.

put the item in knapsack to get maximum total values.

$N = 3$ $W = 50$

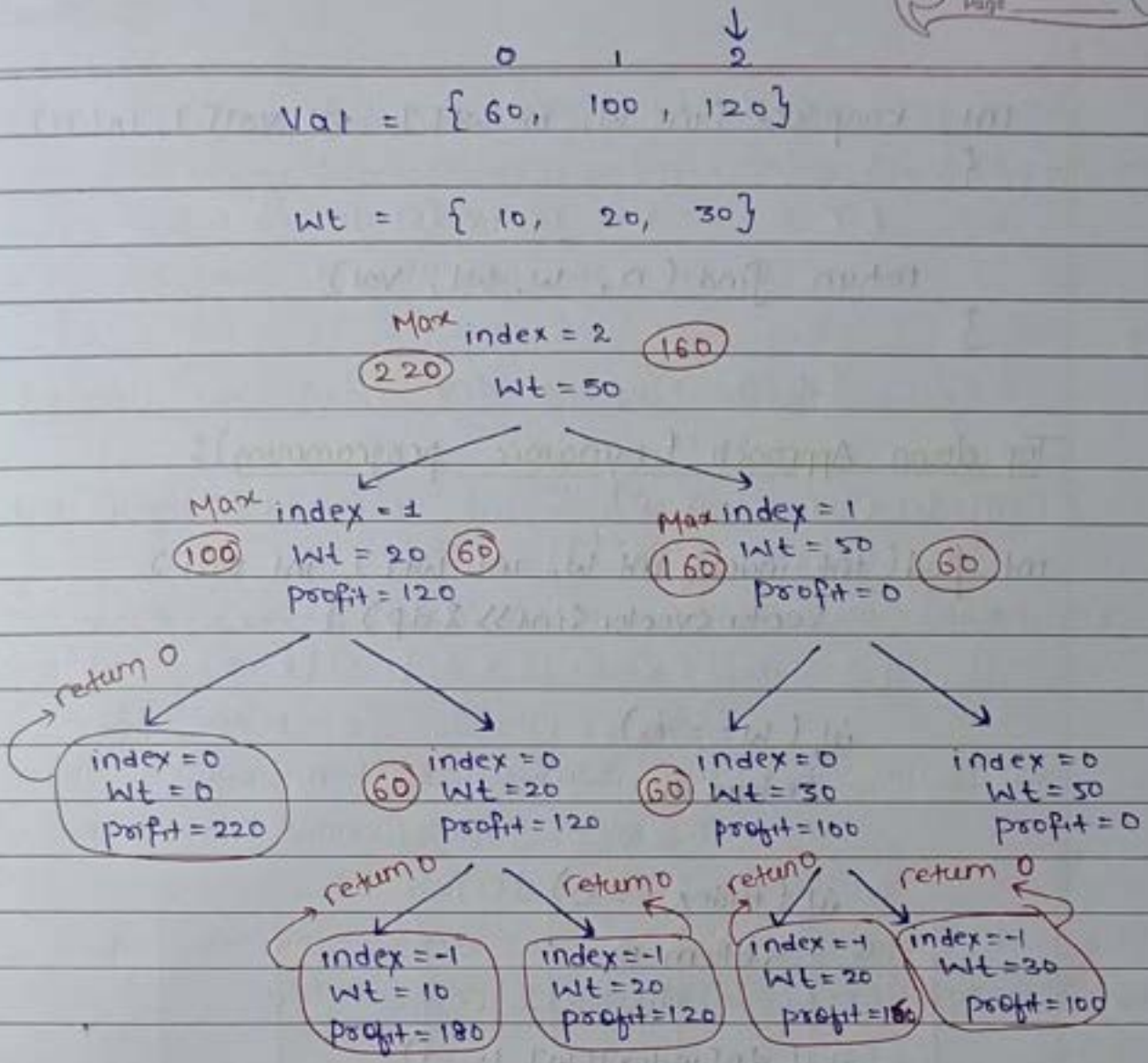
values[] = {60, 100, 120}

weight[] = {10, 20, 30}

In this we solve by recursion tree.

Either we take the full item or we don't take it.

classmate
Date
Page



Top down Approach: (Recursion)

```
int find (int index, int W, int Wt[], int val[]) {
        if (W == 0)
        return 0;

        if (index == 0)
          return 0;

        if( Wt[index - 1] > W)
        return find (index - 1, W, Wt, val);

        else
        return max (val[index-1] + find(index-1, W-Wt[index-1],
                Wt, val), find(index-1, W, Wt, val);
}
```

```
int knapsack (int W, int wt[], int val[], int n)
 {

    return find (n, W, wt, val);
 }
```

Top down Approach (Dynamic programming):

```
int find( int index, int W, int wt[], int val[],
          vector<vector<int>> &dp) {

      if (W == 0)
        return 0;

      if (index == 0)
        return 0;

      if (dp[index][W] != -1)
        return dp[index][W];

     if (wt[index] > W)
     return dp[index][W] = find ( index -1, W, wt, val, dp);

     else
     return dp[index][W] = max (val[index-1] +
       find( index-1, W-wt[index-1], wt, val, dp),
       find ( index-1, W, wt, val, dp));

 }
```

```
int knapsack(int W, int wt[], int val[], int n) {
    vector<vector<int>> dp(n+1, vector<int>(W+1, -1));

    return find(n, W, wt, val, dp);
}
```

Bottom up Approach (Dynamic programming) :

```
int knapsack(int W, int wt[], int val[], int n) {

    vector<vector<int>> dp(n+1, vector<int>(W+1, 0));

    for(int i=1; i<=n; i++) {
        for(int j=1; j<=W; j++) {
            if(wt[i-1] > j)
                dp[i][j] = dp[i-1][j];
            else
                dp[i][j] = max(val[i-1] + dp[i-1][j-wt[i-1]],
                               dp[i-1][j]);
        }
    }
    return dp[n][W];
}
```

* Optimise the Space Complexity :

```
int knapsack(int W, int wt[], int val[], int n) {
    vector<int> dp(W+1, 0);

    for(int i=1; i<=n; i++) {
        for(int j=W; j>=wt[i-1]; j--) {
            dp[j] = max(dp[j], val[i-1] + dp[j-wt[i-1]]);
        }
    }
    return dp[W];
}
```

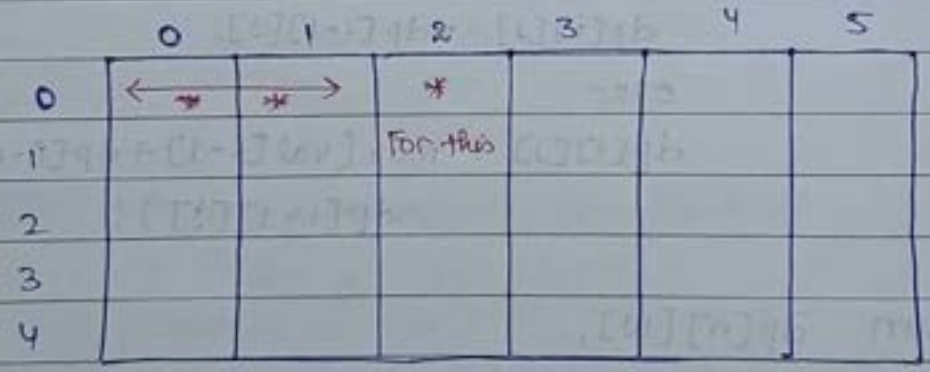
classmate
Date
Page

Lecture - 73

LCS + pallindrome

* Space Complexity optimised
(0-1 knapsack problem)

```
if (wt[i-1] > j)
  dp[i][j] = dp[i-1][j];
else
dp[i][j] = max(val[i-1] + dp[i-1][j - wt[i-1]],
                dp[i-1][j]);
```

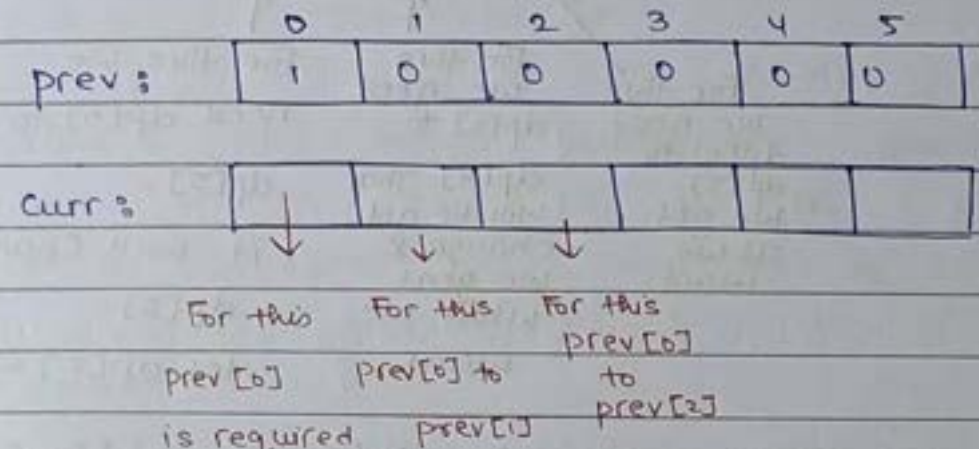


2D DP → 1D DP

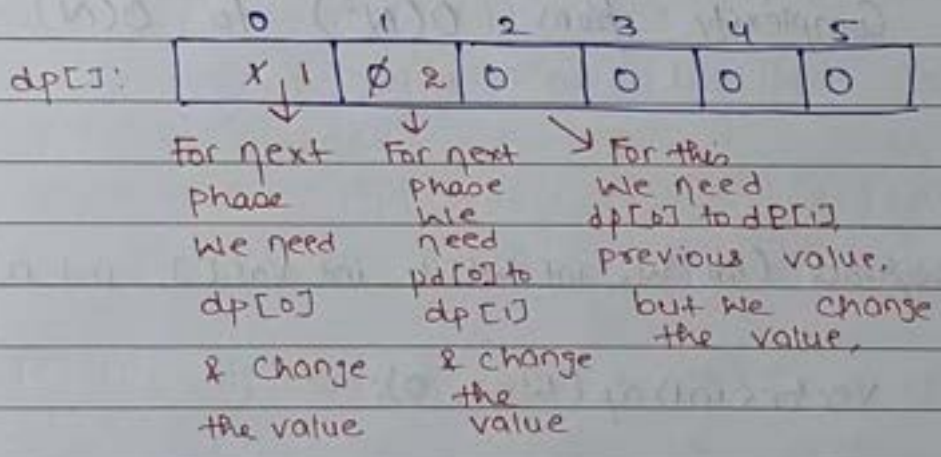
for solving dp[1][2]
we need.
↳ dp[0][2]
↳ dp[0][0] to dp[0][1]

these all are from upper row.

So, insted of whole 2D-Dp we only need 1 row for solving any index.



But we have to solve with one 1-D Dp.
Lets try to solve:

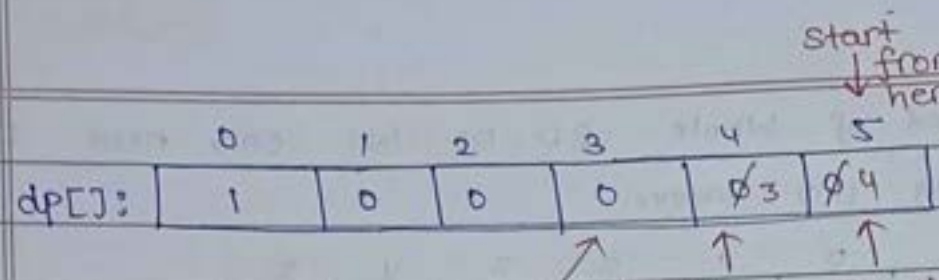


So, by this process, it is not possible to Solve this and decrease the space Complexity.

* If we do this process from the last element of the array, then the required element is not changed for any element

Let's understand with an example.

classmate
Date
Page



→ This is the right approach when we start solving from last.

→ So, by this method we can optimise the space Complexity from $O(N^2)$ to $O(N)$.

Code :

```
int Knapsack (int W, int wt[], int val[], int n){

        Vector<int> dp(W+1, 0);

    for(int i=1; i<=n; i++){
        for(int j=W; j>0; j--){
            if(wt[i-1] <= j)
            dp[j] = max(val[i-1]+dp[j-wt[i-1]], dp[j])
        }
    }
    return dp[W];
}
```

* Longest Common Subsequence :

Given : Two string

Task : find longest subsequence present in both.

Let us understant with the help of Example

Example :

Str1 : A B C D G H

Str2 : A E D F H R

Domo string me se sabe bada sub string find karna hai

↳ String jaruri nhi hai ki lagatar hona chaiye.

(A) B C (D) G (H)

(A) E (D) F (H) R

(Size : 3)

Example 2 (A) (C) D

D E (A) P (C)

(Size : 2)

Real life Example : (DNA)

We have to find
↳ the longest pattern.

Approach :

```
                ↓
Str1 :  A   C   D

Str2 :  D   E   A   P   C
                        ↑
```

Start from last

D != C

Break in two part

↳ from 1st part Remove last element of str1.

↳ on 2nd part, Remove last character of str2

If Matched :

C == C

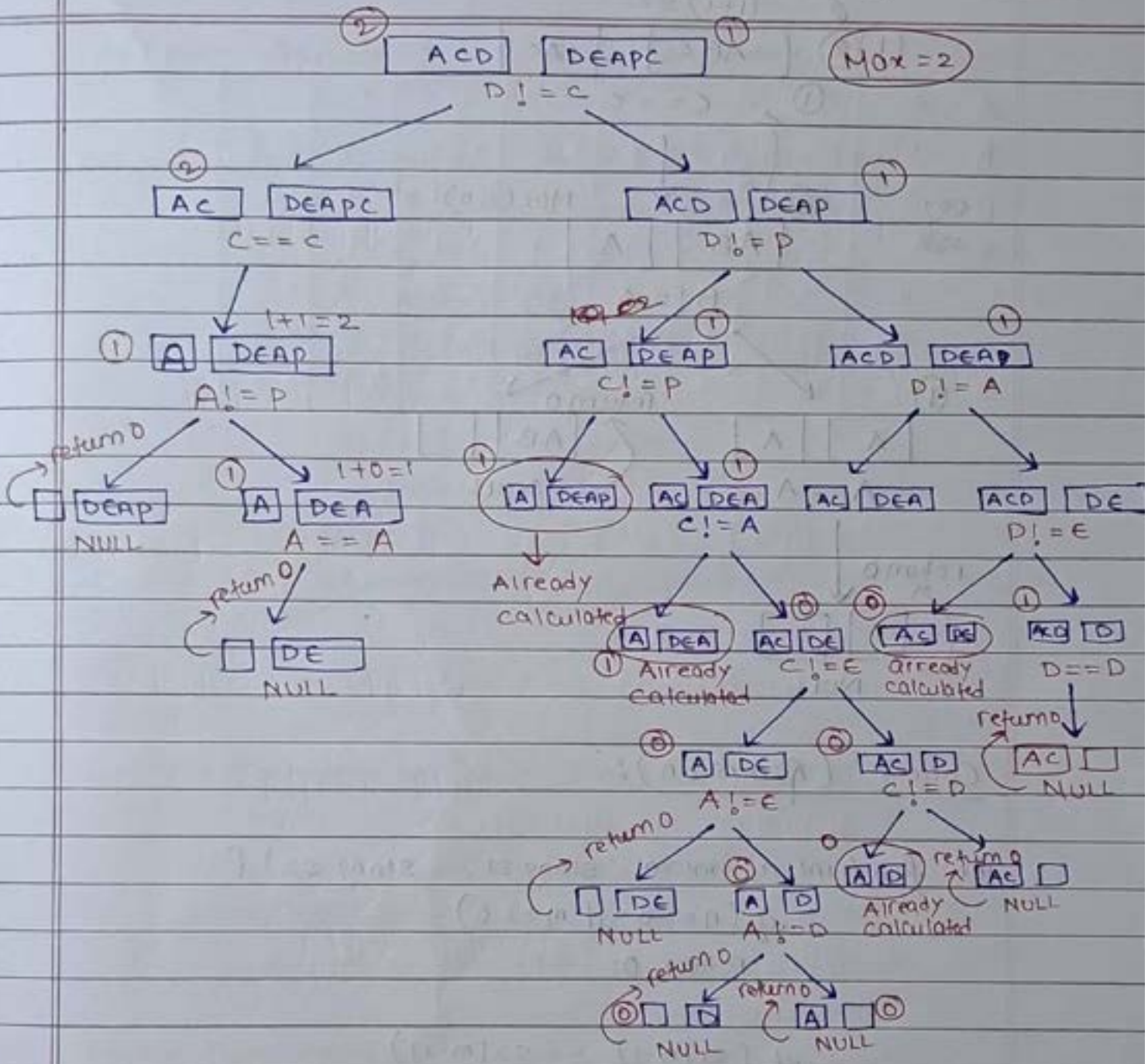
↳ then remove last alphabet from both the strng.

↳ on this Condition, we can't break in two part. we only create one part.

If one string is empty :

↳ then return 0.

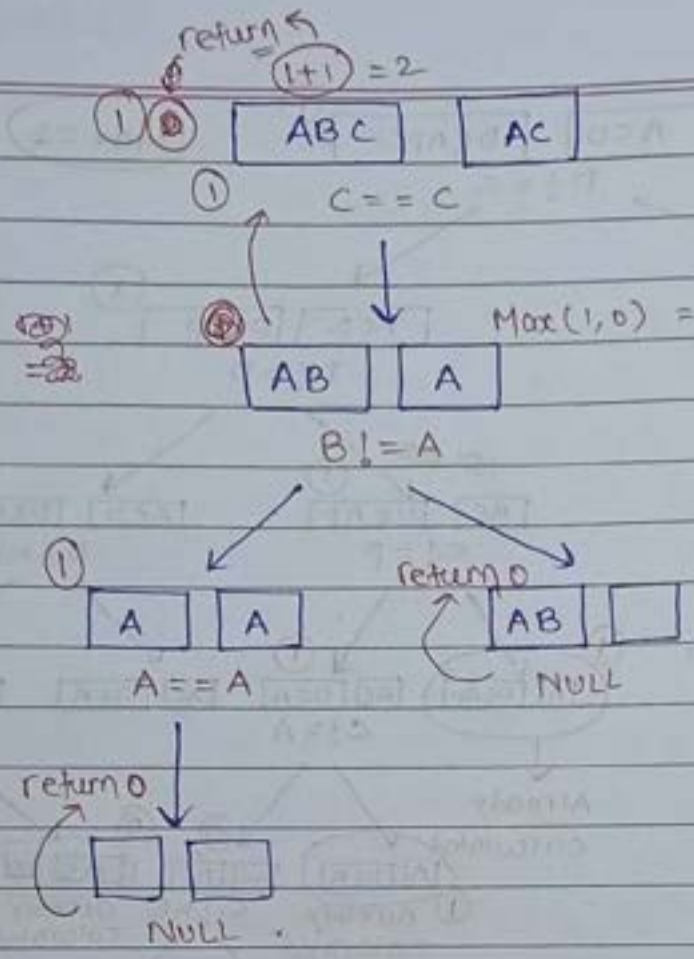
```
str1[i] == str2[j];
return 1 + find(i-1, j-1);

return max(find(i-1, j), find(i, j-1));
```



Let us Clear this Concept with a Small Example :

Str 1 : A B C

Str 2 : A C



Code: (Recursion):

```
int find (int n, int m, string s1, string s2) {
        if (n==0 || m==0)
          return 0;

        if (s1[n-1] == s2[m-1])
        return 1 + find (n-1, m-1, s1, s2);

        else
        return max (find (n-1, m, s1, s2),
                          find (n, m-1, s1, s2));
}
int lcs (int n, int m, string s1, string s2) {

        return find (n, m, s1, s2);
}
```

Top down Approach : Dynamic programming :

```
int find (int n, int m, String &s1, String &s2,
                    vector<vector<int>>& dp ) {
        if ( n == 0 || m == 0)
            return 0;
        if ( dp[n][m] != -1)
            return dp[n][m];
        if ( s1[n-1] == s2[m-1])
            return dp[n][m] = 1 + find ( n-1, m-1, s1, s2, dp);
        else
        return dp[n][m] = max (find (n-1, m, s1, s2, dp),
                               find (n, m-1, s1, s2, dp));
}

int lcs ( int n, int m, string s1, string s2) {

    vector <vector< int>> dp (n+1, vector (int) (m+1, -1));
    return find( n, m, s1, s2, dp);
}
```

Bottom up Approach :

String 1 = A B C D G H

String 2 = A E D F H R

We make a 2D Dp with size of str1.length + 1 & str2.length + 1.

dp[7][7];

classmate
Date
Page

		A	E	D	F	H	R
		0	1	2	3	4	5	6
	0	0	0	0	0	0	0	0
A	1	0	1 (A=A)	1	1	1	1	1
B	2	0	1	1	1	1	1	1
C	3	0	1	1	1	1	1	1
D	4	0	1	1	2	2	2	2
G	5	0	1	1	2	2	2	2
H	6	0	1	1	2	2	3	3

```
if (n==0 || m==0)
    return 0;
```

In row 0 and col 0 we put 0.

```
if(MATCH)
dp[i][j]= 1+ dp[i-1][j-1];

else
dp[i][j] = max(dp[i-1][j], dp[i][j-0]);
```

Common String: ADH

Code:

```
int lcs (int n, int m, String s1, String s2) {

    vector<vector<int>> dp (n+1, vector<int> (m+1, 0));
    for (int i = 1; i<=n ; i++) {
      for (int j=1; j<=m; j++) {
          if(s1[i-1] == s2[j-0])
          dp[i][j] = 1 + dp[i-1][i-1];
```

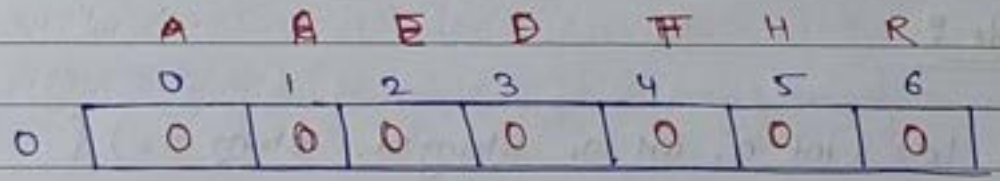
```
            else
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }
    return dp[n][m];
}
```

Space Complexity : $O(N+1)(M+1)$

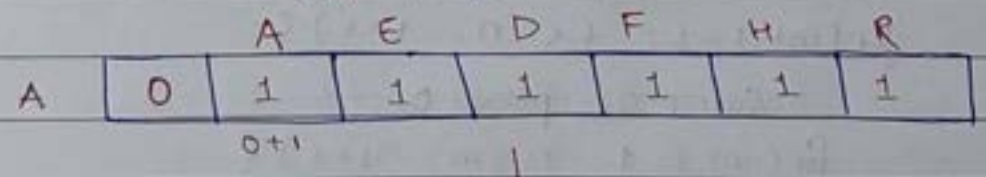
If we want to decrease the space Complexity it only depend on 3 value :

dp[i-1][j-1]
dp[i-1][j]
dp[i][j-1]

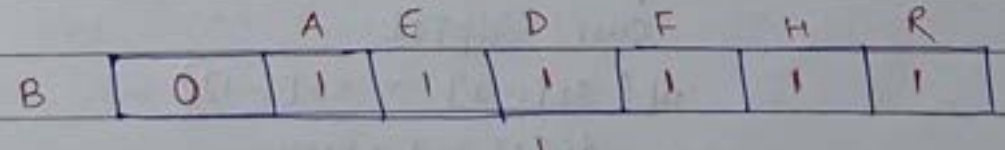
Instead of using 2D DP we will solve this only from 1D DP.



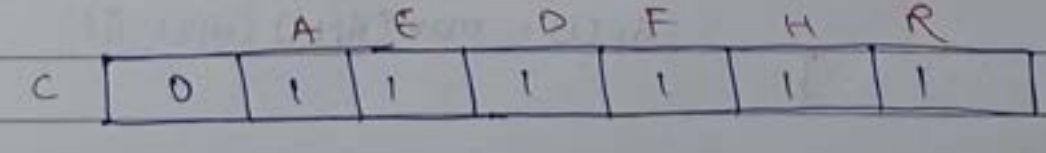
For i = 1



for i = 2



For i = 3



For i = 4

		A	E	D	F	H	R
D	0	1	1	2	2	2	2

for i = 5

		A	E	D	F	H	R
G	0	1	1	2	2	2	2

for i = 6

		A	E	D	F	H	R
H	0	1	1	2	2	3	3

return 3

Space Complexity : O(N).

Code :

```
int lcs (int n, int m, string s1, string s2) {
    vector <int> dp(m+1, 0);
    int curr, prev;
    for(int i = 1; i <= n; i++) {
        curr = 0, prev = 0;
        for(int j = 1; j <= m; j++) {
            prev = curr;
            curr = dp[j];
            if( s1[i-1] = s2[j-1])
                dp[j] = 1 + prev;
            else
                dp[j] = max(dp[j], dp[j-1]);
        }
```

}

return dp[m];

* Longest palindromic Subsequence :

b b a b c b c a b

↳ We want longest palindromic subsequence.

palindrome :

bb

bab

cbc

We can also delete some element if we want.

LCS : b b a b c b c a b

reverse : b a c b c b a b b

b a b c b a b

Find the LCS of string and the reverse of the string.

The Concept is same as Longest Common Subsequence.

only we have to create a new string B, which is reverse of A.

Recursion (code):

```
int find(int n, int m, string &s1, string &s2) {
        if (n==0 || m==0)
            return 0;

        if (s1[n-1] == s2[m-1])
        return 1 + find(n-1, m-1, s1, s2);
        else
        return max(find(n, m-1, s1, s2), find(n-1, m, s1, s2));
}

int longestpalinsubseq (string A) {

        string B = A;
        reverse (B.begin(), B.end());

    return find(A.size(), B.size(), A, B);
}
```

Top Down Approach (Dynamic programming):

```
int find ( int n, int m, string &s1, string &s2,
                vector <vector <int>> &dp) {

        if (n == 0 || m == 0)
            return 0;

        if ( dp[n][m] != -1)
          return dp[n][m];

        if ( s1[n-1] == s2[m-1])
        return dp[n][m] = 1 + find (n-1, m-1, s1, s2, dp);

    return dp[n][m] = max (find(n, m-1, s1, s2, dp),
                           find (m, n-1, s1, s2, dp));
}
int longestpalinSubseq ( string A) {
            String  B = A;
            reverse ( B.begin(); B.end());

vector <vector <int>> dp ( A.size()+1, vector <int> (B.size()+1,-

    return find (A.size() , B.size() , A, B, dp);

}
```

Bottom up Approach : (Dynamic programming)

```
int longestpalinsubseq (string A) {

    String B = A;
    reverse (B.begin(), B.end());

    int n = A.size();
    int m = B.size();

Vector <vector <int>> dp (A.size() + 1, vector <int> (B.size() +1, 0));

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (A[i-1] == B[j-1])
            dp[i][j] = 1 + dp[i-1][j-1];
            else
            dp[i][j] = max (dp[i-1][j], dp[i][j-1]);
        }
    }
    return dp[n][m];
}
```

Space Complexity : $O(N+1)(M+1)$

We can decrease the space complexity and write the code gain.

optimal code (space Complexity decrease)

```
int longestPalinSubseq (String A){

    String B = A;
    reverse (B.begin(), B.end());
    int n = A.size();
    int m = B.size();

    Vector<int>dp(m+1, 0);
    int curr, prev;

    for(int i=1; i<=n; i++) {
        curr = 0, prev = 0;
        for(int j=1; j<=m; j++) {
            prev = curr;
            curr = dp[j];

            if (A[i-1] = B[j-1])
                dp[j] = 1 + prev;
            else
                dp[j] = max(dp[j], dp[j-1]);
        }
    }
    return dp[m];
}
```

classmate
Date
Page

* Longest Repeating Subsequence :

Given : str

Task : length of longest repeating subsequence.

Example:

str : a x x z x y

0	1	2	3	4	5
a	x	x	z	x	y

1 - 2 2 - 4

2

str : a b c a b

length = 2

str : a x x z x y

Approach:

→ Copy str in str1

→ start traversing from end.

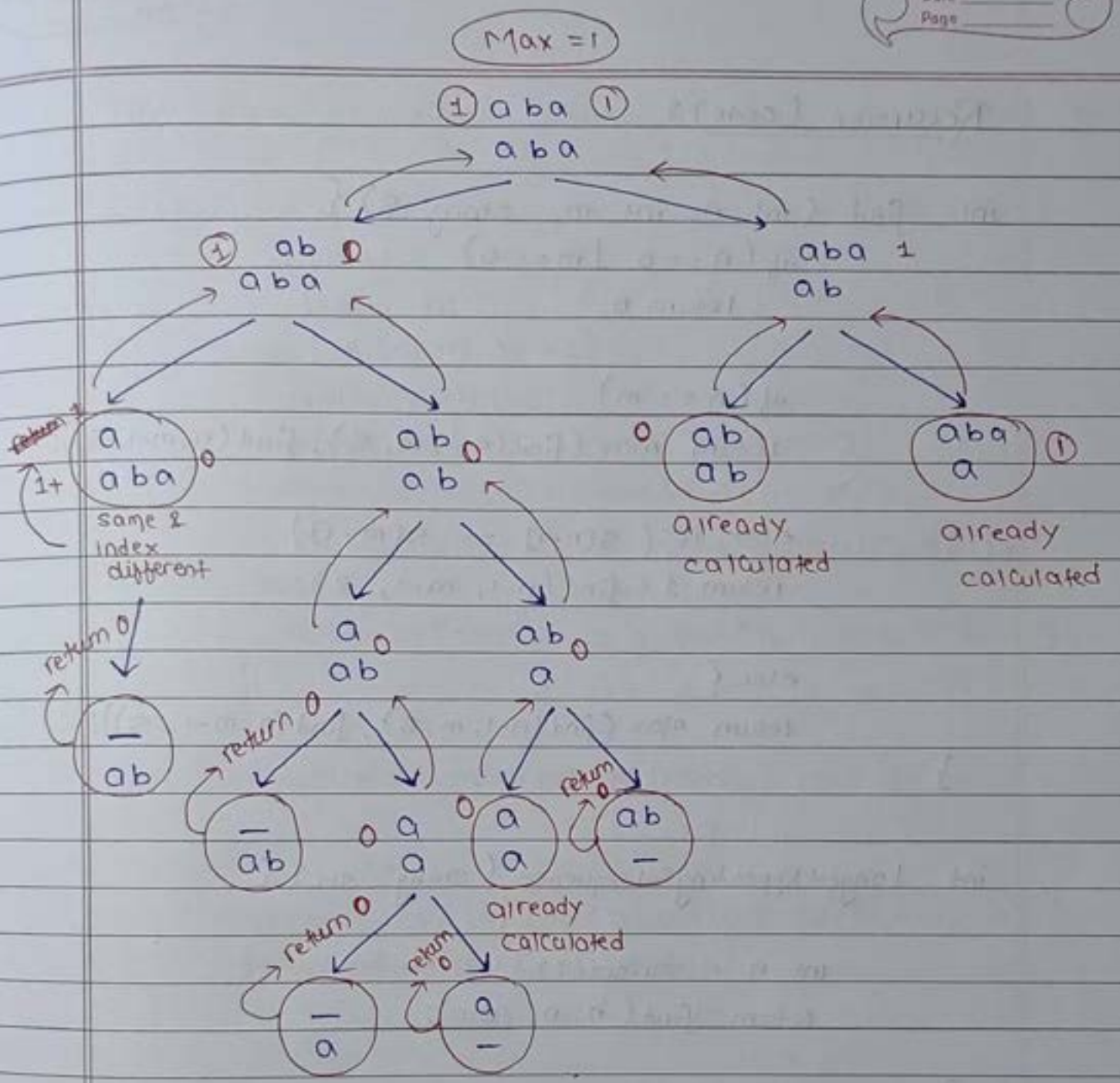
→ if matched || not matched

take two case

↳ remove 1 character from last from one string.

→ if matched && index different

↳ return 1 & remove from both.




```
if (n == m)
    return (find(n-1, m), find(n, m-1))
                                ↳ max.
else if ( s1[n-1] == s2[m-1])
    return  1 + find(n-1, m-1)

else
    return max (find(n-1, m), find(m, n-1))
```

Recursion (code):

```
int find (int n, int m, string s) {
        if (n == 0 | m == 0)
            return 0;

        if (n == m)
        return max (find (n-1, m, s), find (n, m-1, s));

        else if ( s[n-1] == s[m-1])
        return 1 + find (n-1, m-1, s);

        else {
        return max (find (n-1, m, s), find (n, m-1, s));
}

int LongestRepeatingSubsequence (string str) {

        int n = str.size();
        return find ( n, n, str);

}
```

Top down Approach (Dynamic programming):

```
int find (int n, int m, string &s, vector<vector<int>> &dp) {
        if (n==0 || m==0)
          return 0;
        if ( dp[n][m] != -1)
         return dp[n][m];
        if ( n==m || s[n-1] != s[m-1])
        return dp[n][m] = max (find (n-1, m, s, dp),
                               find(n, m-1, s, dp));
        else
         return dp[n][m] = 1 + find (n-1, m-1, s, dp);
}

int LongestRepeatingSubsequence (string str) {

        int n = str.size();
      vector <vector <int>> dp (n+1, vector <int> (n+1, -1));
      return find (n, n, str, dp);
}
```

Bottom - up Approach (Dynamic programming):

```
int LongestRepeatingSubsequence (string str) {
        int n = str.size();
      vector <vector<int>> dp (n+1, vector<int>(n+1, 0));

      for (int i = 1;  i<=n;  i++) {
            for (int j=1;  j<=n; j++) {

          if( i==j || str[i-1] != str[j-1])
            dp[i][j] = max (dp[i-1][j], dp[i][j-1]);
```

```
            else
                dp[i][j] = 1 + dp[i-1][j-1];
        }
    }
    return dp[n][n];
}
```

Optimising space Complexity (code):

```
int longestRepeatingSubsequence ( string str) {

    int n = str.size();
    vector <int> dp (n+1, 0);

    for(int i=1; i<=n; i++) {
        int prev = 0;
        for(int j=1; j<=n; j++) {
            int temp = dp[j];
            if ( i != j && str[i-1] == str[j-1]) {
                dp[j] = 1 + prev;
            } else
                dp[j] = max(dp[j], dp[j-1]);
            prev = temp;
        }
    }
    return dp[n];
}
```

→ When we start solving question on Dynamic programming, we analyze all the approach in written form then we will able to understand and write the code.

→ After solving certain number of problem, we will analyze the approach without paper and able to write the code.

Specially for the Conversion of
- ↳ Recursion to Top down Approach
- ↳ Top down Approach to Bottom up Approach
- ↳ Bottom up Approach to optimise the space Complexity.

* Minimum number of deletions and insertions :

Given : two string str1 & str2

Task : Total No. of operation performed.
↳ For Making str2 from str1.

Example: str1 = heap str2 = pea

Approach:

Find LCS : ea

Rest Element = 2 (from str1)

Deletion = 2

For str2 → size = 3

3 - size of LCS

3 - 2 = 1

Insertion = 1.

deletion = str1.size - LCS

insertion = str2.size - LCS

This is the same problem as LCS.
only we have to perform some operation after finding the LCS.

Recursion (code):

```
int find(int n, int m, string &s1, string &s2) {
        if (n==0 || m==0)
            return 0;

        if (s1[n-1] == s2[m-1])
        return 1 + find(n-1, m-1, s1, s2);

        else
        return max(find(n-1, m, s1, s2),
                   find(n, m-1, s1, s2));

}
int minOperations(string str1, string str2) {

        int n = str1.size();
        int m = str2.size();

        vector<vector<int>> dp(n+1, vector<int>(m+1, -1));
        int size_lcs = find(n, m, str1, str2, dp);

        int insertion = m - size_lcs;
        int deletion = n - size_lcs;
```

```
    return insertion + deletion;
}
```

Top Down Approach (Dynamic programming):

```
int find(int n, int m, string &s1, string &s2, vector<vector<int>> &dp){
        if(n==0 || m==0)
            return 0;

        if(dp[n][m] != -1)
            return dp[n][m];

        if(s1[n-1] == s2[m-1])
            return dp[n][m] = 1 + find(n-1, m-1, s1, s2, dp);

        else
            return dp[n][m] = max(find(n-1, m, s1, s2, dp),
                                  find(n, m-1, s1, s2, dp));
}
int minOperations(string str1, string str2){
        int n = str1.size();
        int m = str2.size();

    vector<vector<int>> dp(n+1, vector<int>(m+1, -1));
        int size_lcs = find(n, m, str1, str2, dp);

        int insertion = m - size_lcs;
        int deletion = n - size_lcs;

    return insertion + deletion;
}
```

* Bottom up Approach: (Dynamic programming)

```
int minOperations ( string str1, string str2) {

    int n = str1.size();
    int m = str2.size();

vector<vector<int>> dp (n+1, vector<int>(m+1, 0));
    for (int i= 1; i<=n; i++) {
       for (int j= 1; j<=m; j++) {
            if (str1[i-1] == str2[i-1])
              dp[i][j] = 1 + dp[i-1][j-1];
            else
            dp[i][j] = max (dp[i-1][j], dp[i][j-1]);
          }
    }
    int size_lcs = dp[n][m];

    int insertion = m - size_lcs;
    int deletion = n - size_lcs;

    return insertion + deletion;
}
```

* Optimise space Complexity :

```
int minOperations (string str1, string str2) {

        int n = str1.size();
        int m = str2.size();

    vector<int> dp(m+1, 0);
      int curr, prev;

    for (int i=1; i<=n; i++) {
            curr = 0, prev = 0;
        for (int j=1; j<=m; j++) {
                prev = curr;
                curr = dp[j];
            if (str1[i-1] == str2[j-1])
                dp[j] = 1 + prev;
            else
            dp[j] = max (dp[j], dp[j-1]);
        }
    }
    int size_lcs = dp[m];

    int insertion = m - size_lcs;
    int deletion = n - size_lcs;

     return insertion + deletion;
}
```

Lecture - 74

DP Longest Increasing Subsequence || LCS || LAP

* Longest Common Substring
  ↓
  Contigous

In this problem, sobse bada Common Substring Nikalna hai.

Subsequence Me hum beech beech Me chor kar bhi bana Sakte the.

Lekin Contigous chaiye hota hai Substring Me.

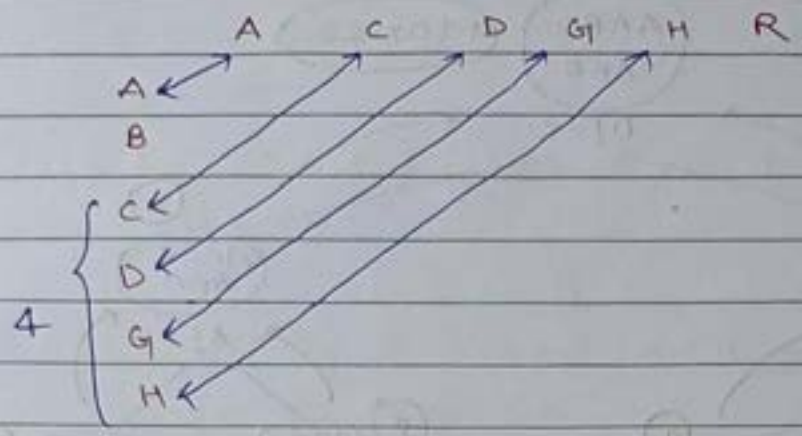
Ex: Str1 : A B C D G H

Str2 : A C D G H R

		A	C	D	G	H	R
		1	0	0	0	0	0
B	0	0	0	0	0	0	
C	0	1+0	0	0	0	0	
D	0	0	1+1	0	0	0	
G	0	0	0	1+2	0	0	
H	0	0	0	0	1+3	0	

⇓

4




```
if ( s1[i-1] == s2[j-1])
    dp[i][j] = 1 + dp[i-1][j-1];

    ans = max (ans, dp[i][j])
```

Recursion Method :

We can start matching both string from last.

If both element does not Match, the We do 2 case.

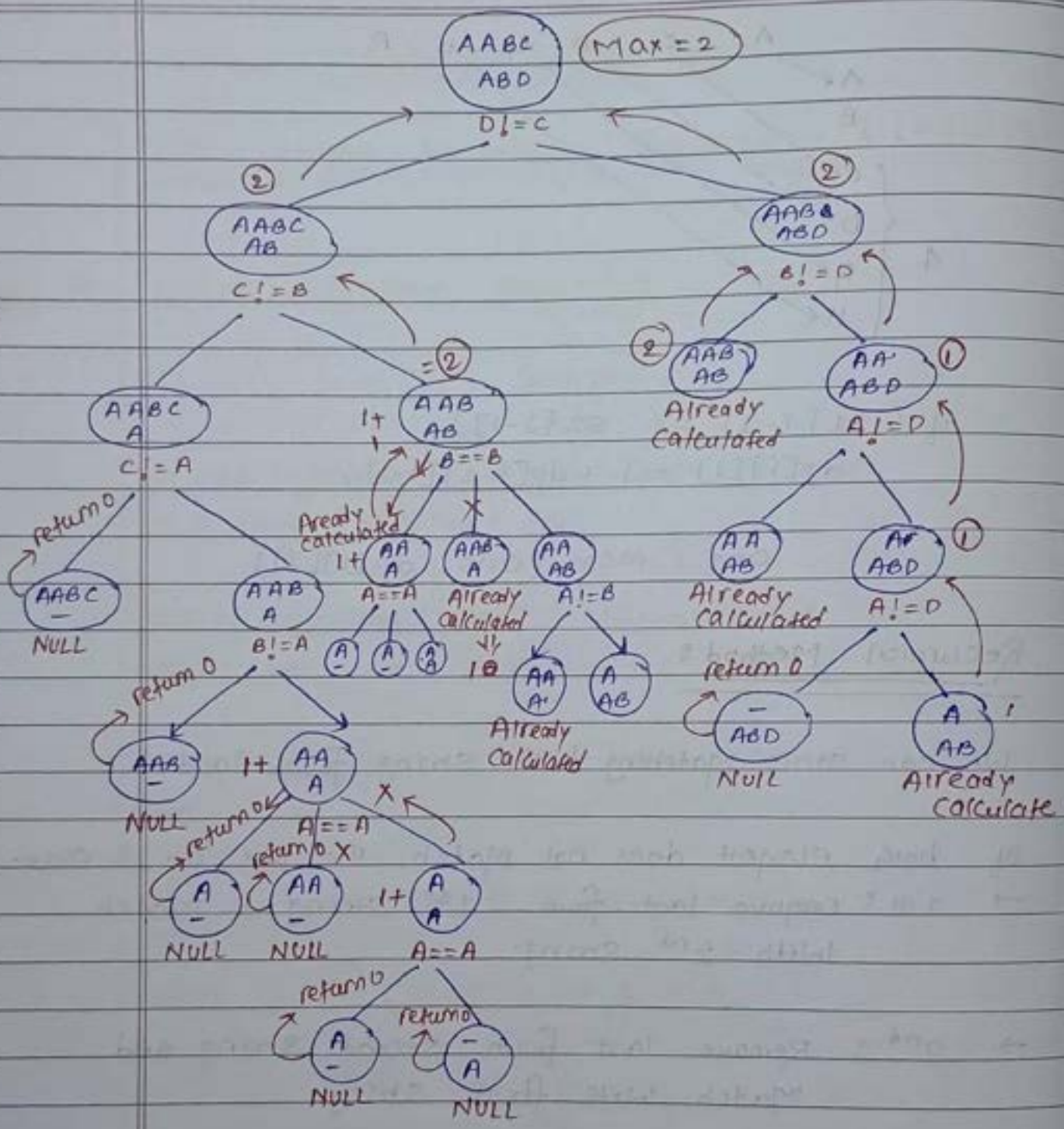
→ 1st : remove last from 1st string & match with 2nd string.

→ 2nd : Remove last from second string and Match with first string.

If both element match, then it divide into 3 parts
- ↳ remove last from both
- ↳ 2 case come from the cases when both element dont Match.

Example : Str1 : A A B C

Str2 : A B D




```
if (n==0 || m==0)
    return 0;

if (S1[n-1] == S2[m-1])
    len = 1 + find(n-1, m-1);

find(n-1, m)
find(m, m-1)
```

```
        ans = max (ans, len);
        return len;
```

Recursion (code) :

```
int find (int n, int m, string &s1, string &s2, int &ans){
        if (n==0 || m==0)
            return 0;

        int len = 0;
        if (s1[n-1] == s2[m-1]) {
            len = 1 + find (n-1, m-1, s1, s2, ans);
            ans = max ( ans, len);
        }

        find ( n-1, m, s1, s2, ans);
        find (n, m-1, s1, s2, ans);
    return len;
}
int longestCommonSubstr (string s1, string s2, int n, int m) {
        int ans = 0;
        find (n, m, s1, s2, ans);
        return ans;

}
```

Top down Approach (Dynamic programming):

```
int find (int n, int m, string &s1, string &s2,
          int &ans, vector<vector<int>> &dp) {
    if (n==0 || m==0)
        return 0;

    if (dp[n][m] != -1)
        return dp[n][m];

    int len = 0;
    if (s1[n-1] == s2[m-1]) {
        len = 1 + find (n-1, m-1, s1, s2, ans, dp);
        ans = max (ans, len);
    }

    find (n-1, m, s1, s2, ans, dp);
    find (n, m-1, s1, s2, ans, dp);

    return dp[n][m] = len;
}
int longestCommonSubstr (string s1, string s2, int n, int m){

    int ans = 0;
    vector<vector<int>> dp (n+1, vector<int>(m+1, -1));
    find (n, m, s1, s2, ans, dp);

    return ans;
}
```

Bottom up Approach :

```
int longestCommonSubstr (string s1, string s2, int n, int m) {

    int ans = 0;

Vector <vector < int >> dp (n+1, vector<int>(m+1, 0));

    for (int i= 1; i<= n; i++) {
        for (int j= 1; j<= m; j++) {
            if (s1[i-1] == s2[j-1]) {
                dp[i][j] = 1 + dp[i-1][j-1];
                ans= max (ans, dp[i][j]);
            }
        }
    }
    return ans;
}
```

Example:

		A	C	D	G	H	R
		1	0	0	0	0	0
B	0	0	0	0	0	0	
C	0	0+1	0	0	0	0	
D	0	0	1+1	0	0	0	
G	0	0	0	2+1	0	0	
H	0	0	0	0	3+1	0	

⇓

4

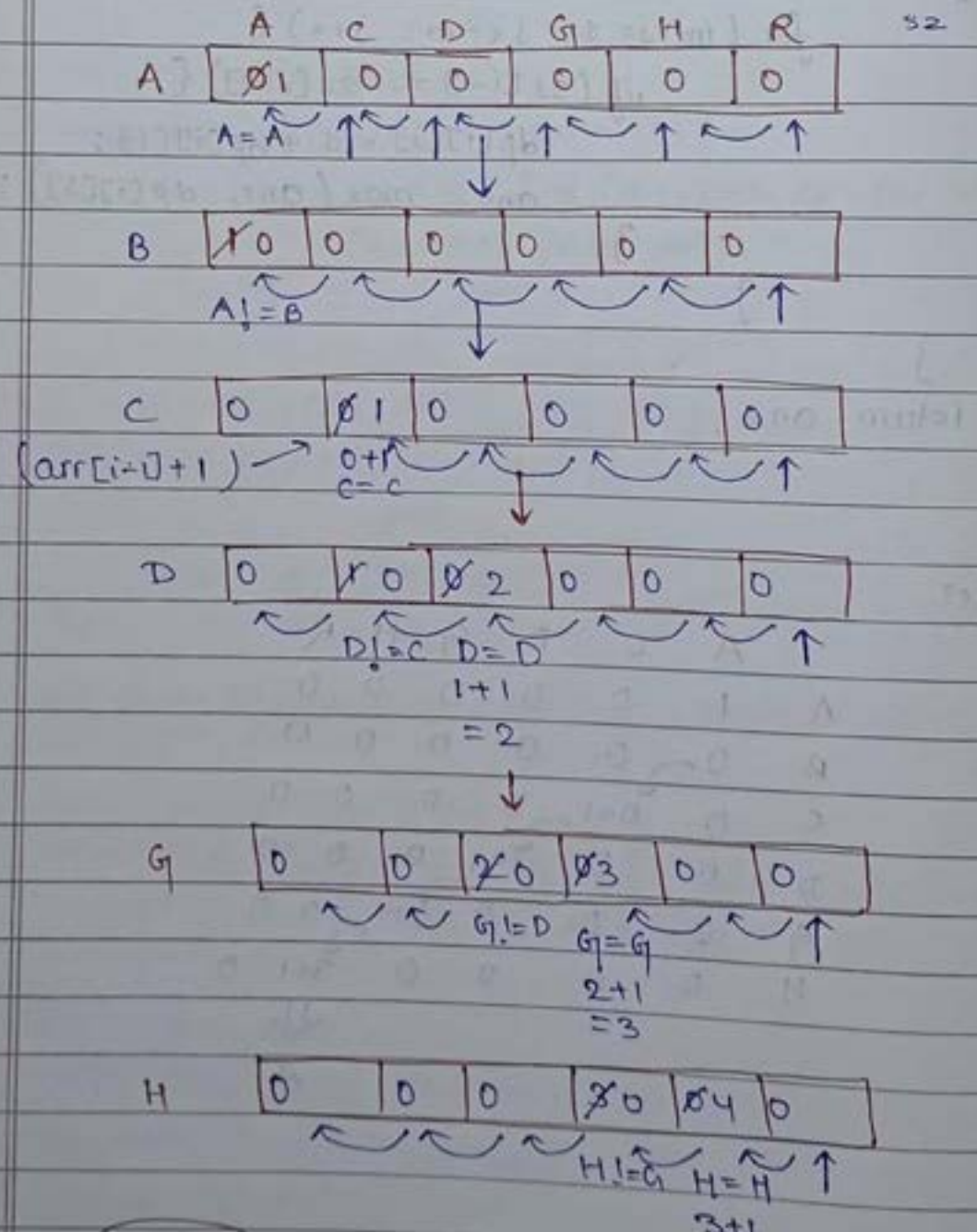
If We Want to optimise our space Complexity then,

the ans index or any index depend on only.

↳ one - D array

We can calculate all the values from only one - D Array.

We can start solving from last



Max = 4

Ans

Code (space Complexity optimused) :

```
int longestCommonSubstr (string s1, string s2, int n, int m) {

    int ans = 0;
    vector <int> dp (m+1, 0);

    for (int i=1; i<=n; i++) {
        for (int j= m; j>=1 ; j--) {
            if (S1[i-1] == s2[j-1]) {
                dp[j] = dp[j-1] + 1;
                ans = max (ans, dp[j]);
            }
            else
                dp[j] = 0;
        }
    }
    return ans;
}
```

classmate
Date
Page

* Longest increasing Subsequence:

Given: An array of integers.

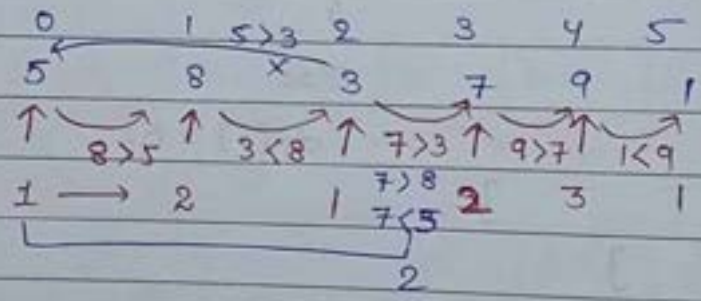
Task: length of longest increasing subsequence

Example:

N = 6

{ 5, 8, 3, 7, 9, 1 }
0 1 2 3 4 5

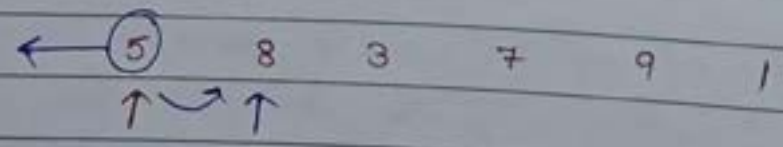
Approach:



Ans = 3

N = 6

Detailed Approach:

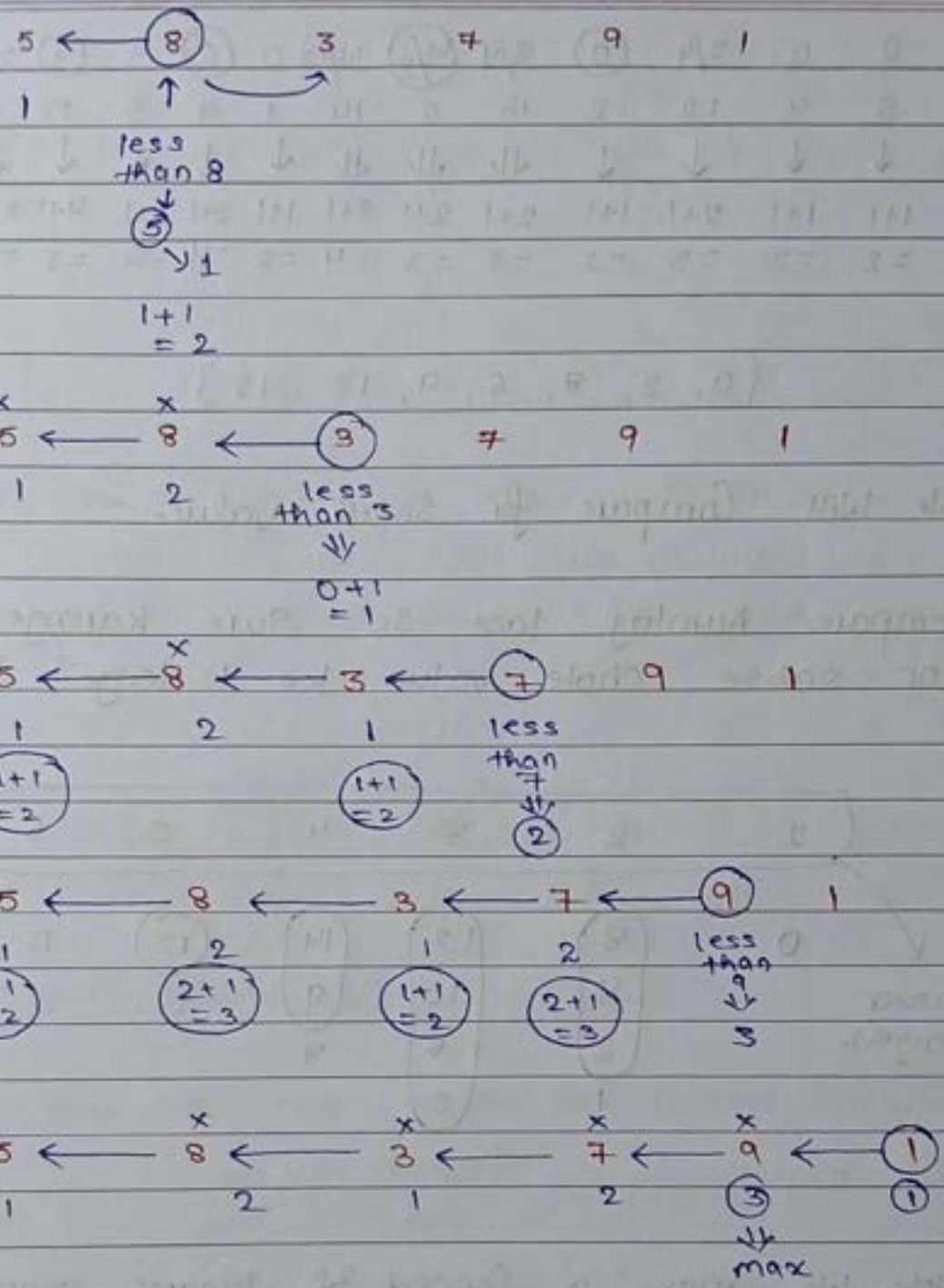


less than

5 = 0

height = 1

self



Ans = 3

Time Complexity = $O(N^2)$

space Complexity = $O(N)$

We have to convert time complexity to $O(\log N)$

Example ⊖ 0 0 8/4 ⓪ 8/4 4/2 10/12 0 ⑥ 4 ⑨ 2 9 6 ⑬

0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15

↓ ↓ ↓ ↓ ↓ ↓↓ ↓↓ ↓↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

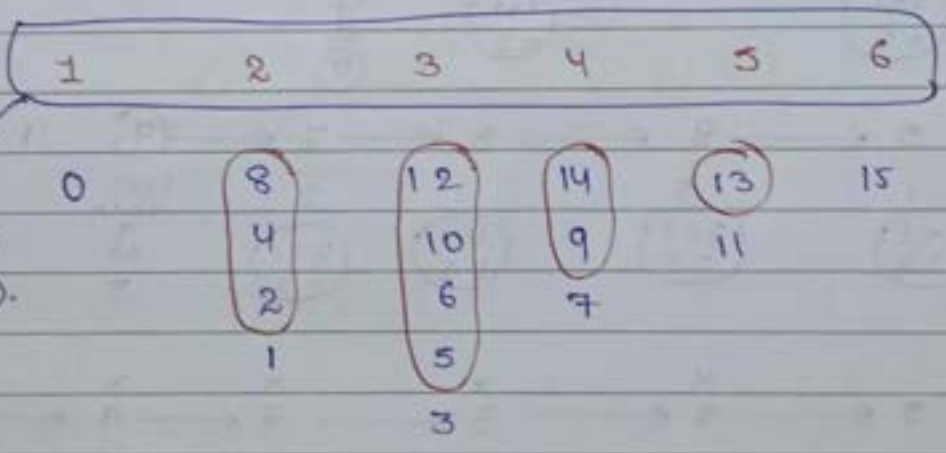
1 1+1 1+1 2+1 1+1 2+1 2+1 3+1 1+1 3+1 2+1 4+1 2+1 4+1 3+1 5+1

=2 =2 =3 =2 =3 =3 =4 =2 =4 =3 =5 =3 =5 =4 =6

{0, 2, 4, 6, 9, 13, 15}

→ We will Compare for Small value.

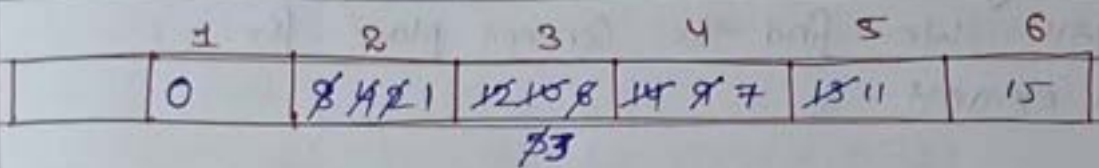
→ Compare humlog last se Sure karenge or sabse chote value ko le lenge.

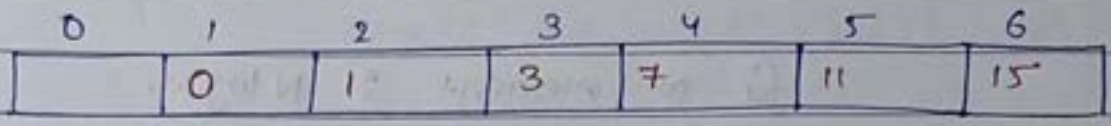


Now, we make a Concept of Binary search.

Jab humlog Sabse small value ko dhundte hai to kyu nahi hum Sife Ek hi value ko store kare.

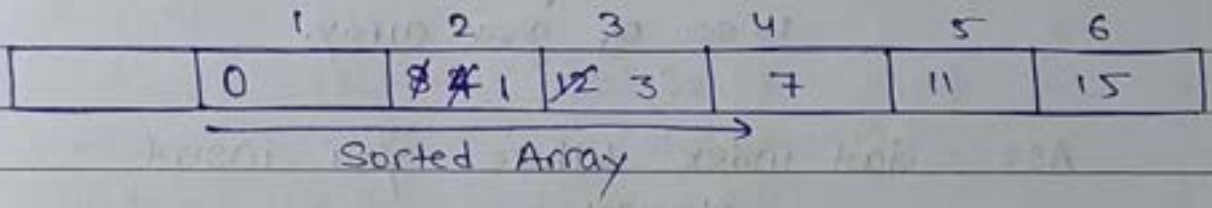
classmate
Date
Page





Lets see how we use Binary search :

0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15



0 → fits at 1

8 → 8 can search its correct position

start = 1

end = 1

mid = 1

arr[mid] < 8

↳ 8 goes to mid + 1

4 → start = 1

end = 2 (mid = 1)

arr[mid] < 4

↳ 4 goes to (mid + 1, end)

12 → start = 1, end = 2, mid = 1

arr[mid] < 12 (mid + 1, end)

(mid = 2) arr[mid] < 12 → (mid + 1)

By solving by binary search for each element we find the correct place for each element.

For one element : log N

for N element : N log N.

Jo bhi answer array ka length aayega wahi answer ho jayega.

Array ka size define karenge
↳ size of given array.

As : last index where you insert element.

Code :

```
int longestSubsequence (int n, int a[]) {

    int size = 0, start, end;
    int mid, index;

    vector<int> LIS (n);
    LIS [0] = a[0];

    for (int i = 1; i<n; i++) {
        start = 0;
        end = size;
        index = size+1;
```

```
While ( start <= end)
    mid = start + (end - start) / 2;

    if (Lis[mid]< a[i])
        start = mid+1;
    else if ( Lis[mid == a[i]){
        index = mid;
        break;
    }
    else {
        index = mid;
        end = mid-1;
    }
}
Lis[index] = a[i];
size = max(size ,index);
}
return size + 1;
}
```

* Longest Arithmetic progression

Given : An Array
↳ having No Repeated Elements

Task : Find longest AP

0	1	2	3	4	5
1	7	10	13	14	19

(difference, length)

0	1	2	3	4	5
1	7	10	13	14	19
	(6, 2) ↓ With 1	(3, 2) With 7	(3, 3) With 10	(1, 2) With 13	(5, 2) With 14
		(9, 2) With 1	(6, 3) With 7	(4, 2) With 10	(6, 4) With 13
			(12, 2) With 1	(7, 2) With 7	(9, 3) With 10
				(13, 2) With 1	(12, 2) With 7
					(13, 2) With 1

Ans = Maximum length.

Time Complexity : $O(N^3)$

If we make unordered map, then we can access it in O(1) time.

↳ then time Complexity = $O(N^2)$.

```
unordered_map <int, int> m[6];
```

But, after using unordered map it gives time limit exceed.

Because unorded map does not take exactly O(1), it take some more than O(1).

Let we write code with unorded Map :

```
int lengthOfLongestAp (int A[], int n) {

    if (n <= 2)
      return n;

    unordered_map <int, int> m[n];
      int d;
    int ans = 2;
    for (int i = 1; i<n; i++) {
        for (int j = i-1; j >=0 ; j--) {
            d = A[i] - A[j] ;
          if ( m[j].count(d)) {
            m[i][d] = max (m[i][d], 1 + m[j][d]);
            ans = max (ans, m[i][d]) ;
          }
          else {
            if ( !m[i].count(d)) {
                m[i][d] = 2;
            }
```

```
            }
        }
    }
    return ans;
}
```

This gives time limit Exceed

→ So we have to use Something else for the Solving in place of unordered map.

Make an array of size : (Maximum difference)

⇓

last element - first element

2D Array

1 7 10 13 14 19

19 - 1 = 18

Make a two D array of 18 x 18.

	1	7	10	13	14	19
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	2				
7						
8						
:						
18						

Let we can understand in detail :-

	1	7	10	13	14	19
0	0	0	0	0	0	0
1	0	0	0	0	~~0~~ 2	0
2	0	0	0	0	0	0
3	0	0	~~0~~ 2	~~0~~ 3	0	0
4	0	0	0	0	~~0~~ 2	0
5	0	0	0	0	0	~~0~~ 2
6	0	~~0~~ 2	0	~~0~~ 3	0	~~0~~ 4
7	0	0	0	0	~~0~~ 2	0
8	0	0	0	0	0	0
9	0	0	~~0~~ 2	0	0	~~0~~ 3
10	0	0	0	0	0	0
11	0	0	0	0	0	0
12	0	0	0	~~0~~ 2	0	~~0~~ 3
13	0	0	0	0	~~0~~ 2	0
14	0	0	0	0	0	0
15	0	0	0	0	0	0
16	0	0	0	0	0	0
17	0	0	0	0	0	0
18	0	0	0	0	0	~~0~~ 2

Max = 4

T.C = $O(N^2)$

S.C = $O(N^2)$

Lecture 75

Grid + Stock

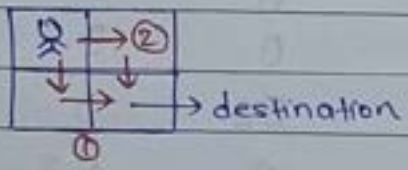
* Number of unique paths:

Grid : mxnx

Ek baar me app ya to right move kar sakte ho ya down move kar sakte ho.

↳ Right

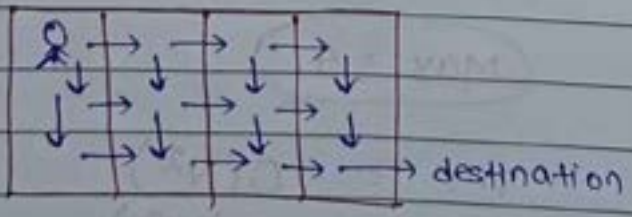
↳ Down



from source to destination, How many ways to reach?

Ans = 2

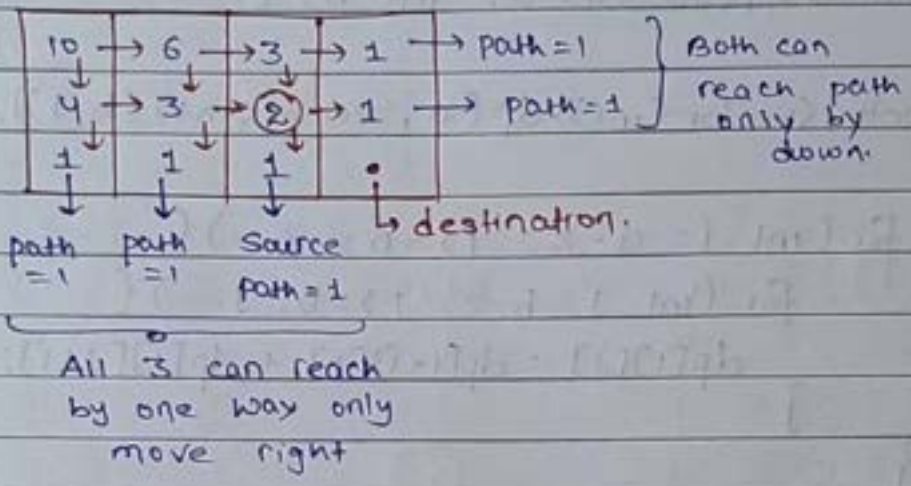
Example : 2



Total : 10 ways.

classmate
Date
Page

* . How to approach ?



Sum of the down and right index.

→ Make the last row & last col = 1.

→ Start with blank index which is empty.

Code understanding :

Assign last row & col to 1

			1
			1
1	1	1	1

Add all the rest index with

a[i][j] = a[i+1][j] + a[i][j+1];

→ Answer.

16	6	3	1
4	3	2	1
1	1	1	1

Code :

```
int NumberOfPaths (int a, int b){
    Vector<vector<int>> dp(a, vector<int>(b,1));
                    :
    for(int i= a-2; i>=0; i--){
        for (int j = b-2; j>=0; j--){
            dp[i][j] = dp[i+1][j] + dp[i][j+1];
        }
    }
    return dp[0][0];
}
```

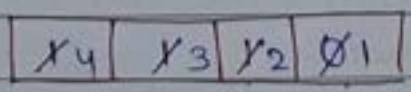
We can optimise this Code :

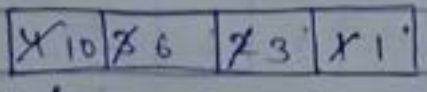
10	6	3	1
4	3	2	1
1	1	1	

→ This row is depend only on next row.

So, we can only solve this by 1 D array.

Approach :





Ans = 10

optimised code:

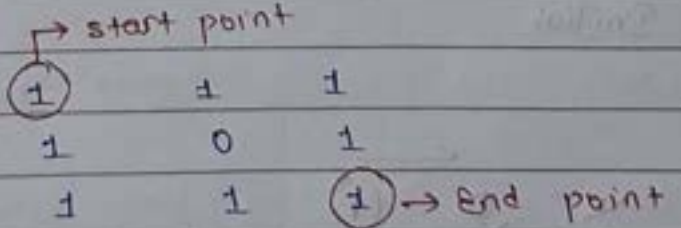
```
int NumberOfPath (int a, int b) {

    Vector <int> dp ( b, 1);

    for (int i = a-2;  i>=0;  i--) {
        for (int j = b-2;  j>=0;  j--) {
            dp[j] += dp[j+1];
        }
    }
    return dp[0];
}
```

* Unique path in a Grid

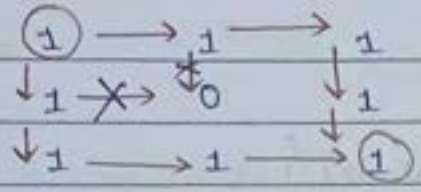
Given: A Grid
↳ size : n x m
↳ Consists 0 and 1
↳ 1 means you are allowed to enter.
↳ 0 means you are not allowed to enter.

Example:

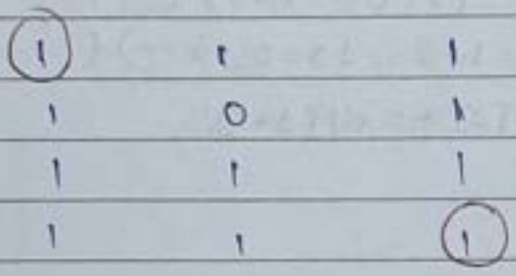
n = 3 m = 3



Find number of possible path to reach end point from start point.



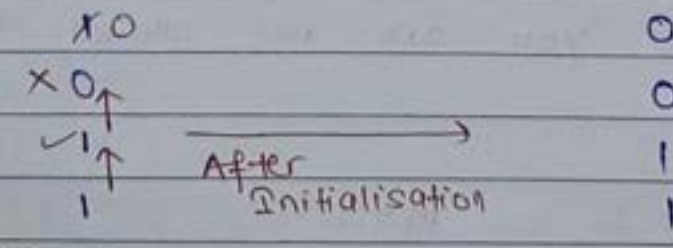
Ans = 2

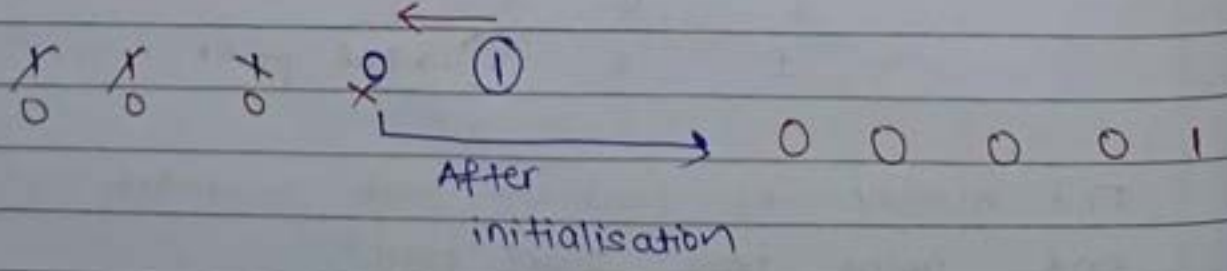


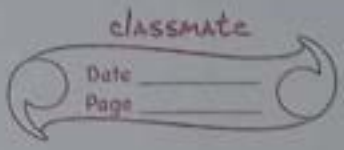
Initialise :

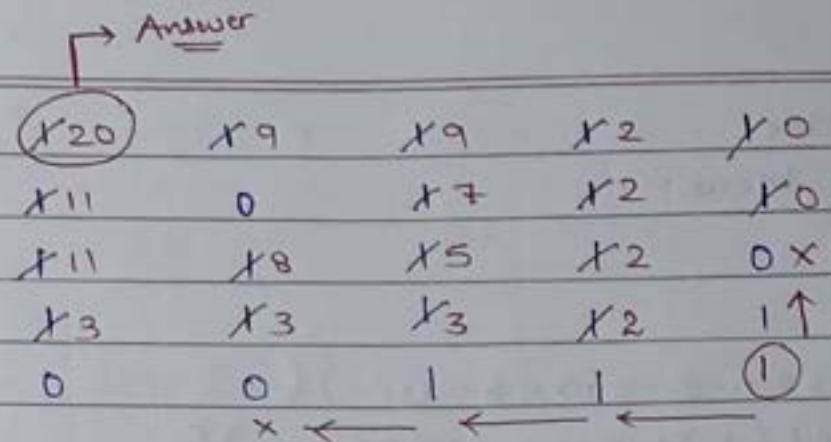
↳ Start from right most bottom
  ↳ check for left and up
    ↳ if any 0 found
      ↳ All upper or left
        . are initialised as 0.

Ex :









→ First we initialise last row & last colum.

→ then we check other element and perform.

arr[i][j] = arr[i][j+1] + arr[i+1][j]

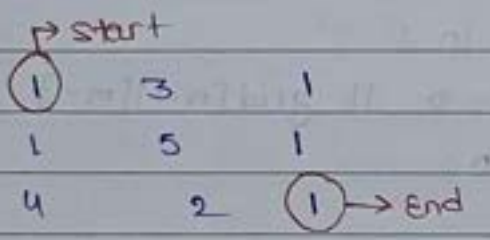
Code :

```
int uniquepaths (int n, int m, vector <vector <int>> &grid ) {
    // for first and last
    if ( grid [0][0] == 0 || grid [n-1][m-1] == 0)
            return 0;

    Vector <vector <int>> dp ( n, vector <int>(m, 0));

for (int j = m-2;  j >= 0;  j --) {
        if (grid [n-1][j] == 1)
            dp[n-1][j] = 1;
        else
            break;
    }

for (int i = n-2;  i >=0 ;  i--) {
        if (grid [i][m-1] == 1)
            dp[i][m-1] = 1;
```

```
            else
                break;
            }

        for(int i = n-2;  i>=0;  i--){
            for(int j = m-2; j>=0; j--){
                if (grid[i][j])
                dp[i][j] = (dp[i+1][j] + dp[i][j+1]) % 1000000007;
            }
        }
        return dp[0][0];
    }
```

* **Minimum Path Sum (LeetCode)**

Example:



Find the path with minimimum cost from start to end.

At each, you can go either Right or Down.

→ First update last row & colum from last

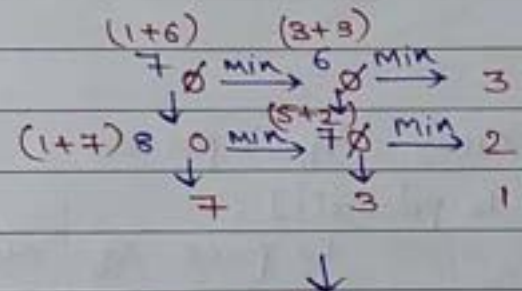
→ Then start for remaining and choose the Min-m path.

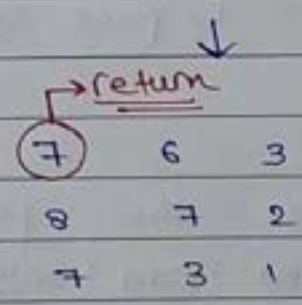
1	3	1
1	5	1
4	2	1

update last row & last column

0	0	∅3 (2+1)
0	0	∅2 (1+1)
∅7 (4+3)	∅3 (1+2)	∅1

For remaining choose Minm from right or down





Code:

```
int minPathSum ( vector<vector<int>> &grid ) {

    int n = grid.size();
    int m = grid[0].size();

for (int j= m-2;    j>=0;  j--){
    grid$[n-1][j] +=grid$[n-1][j+1];
}
```

```
for (int i= n-2;  i >= 0;  i--){
    grid[i][m-1] += grid[i+1][m-1];
}

for ( int i = n-2;  i >= 0;  i--) {
    for ( int j = m-2;  j >= 0;  j--){
    grid[i][j] += min ( grid[i+1][j] , grid[i][j+1]);
    }
}
return grid[0][0];
}
```

* **Best time to Buy and Sell Stock**

Given : Array of prices
↳ prices[i]
↳ price of stock on ith day.

→ you want to maximize the profit.
one day you can only buy the stock or sell the stock.

you are allowed to skip the day.

↳ Return Maximum profit.

↳ If No profit made
↳ return 0

Days	1	2	3	4	5	6
	7	1	5	3	6	4

profit (Condition)

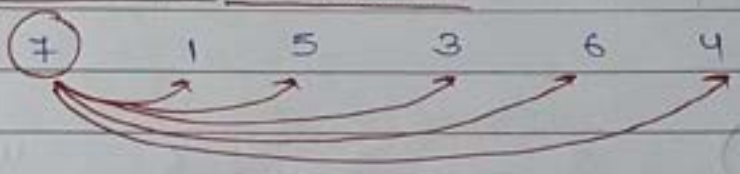
↳ Buy on less price

↳ sell on More price

→ Minimise the Buy day

↳ First we have to buy.

BRUTE FORCE APPROACH



profit = 0

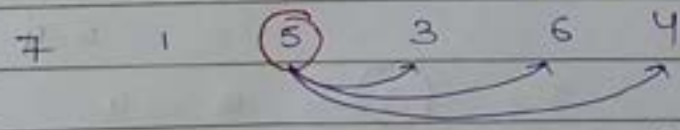
profit -6 -2 -4 -1 -3

No profit

profit = 5

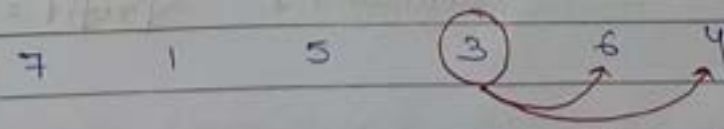
Profit : 4 2 5 3

↳ Max



profit = 5

profit 2 -1 1



profit = 5

profit 3 1

Time Complexity = N^2

Approach 2 :

7 1 5 3 6 4

Find smallest and largest and we get the Maximum profit

largest always comes after smallest.

(7) 1 5 3 6 4

↑

smallest = 7 profit = 0

7 (1) 5 3 6 4

1 > 7

false profit = 0

smallest = 1

7 1 (5) 3 6 4

5 > 1

smallest = 1 profit = max (0, 5-1)

= 4

7 1 5 (3) 6 4

3 > 1

smallest = 1 profit = max (4, 3-1)

= 4

7 1 5 3 (6) 4

6 > 1

smallest = 1

profit = max (4, 6-1)

= 5

7 1 5 3 6 (4)

4 > 1

smallest = 1

profit = max (5, 4-1)

= 5

- Time Complexity : O(N)

Code :

```
int maxprofit ( vector <int> & prices){
        int profit = 0;
        int n = prices.size();
        int stock = prices[0];

    for (int i= 1; i<n; i++){
        profit += max ( profit , prices[i] - stock);
        stock = min (stock, prices[i]);
    }
    return profit;
}
```

* Best time to Buy and sell Stock II

An Integer Array with prices[i] of stock on ith day.

↳ Find maximum profit.

You can buy and sell the stock any number of times.

Condition :
↳ you can first buy the stock.
↳ ~~After~~ 2
↳ Before 2nd time buy the stock you have to sell the previous stock.

Example:

7 1 5 3 6 4

→ Unlimited time buy sell.

Condition :
↳ Buy → sell → Buy → sell ✓

↳ Buy → Buy → sell ✗

↳ Buy & sell on same day ✓
But profit 0.

7 1 5 3 6 4

1 → 5
5 → 6

profit = 5

1 5
3 6

profit = 7

Approach :

7 1 5 3 6 4

Buy at 7

7 1 5 3 6 4

1 < 7

cancel at 7 — Buy at 1

7 1 5 3 6 4

5 > 1

profit

sell at 5 → 5 - 1 = 4 profit

Then, Buy at 5.

7 1 5 3 6 4

3 < 5

cancel at 5 — Buy at 3

7 1 5 3 6 4

6 > 3

sell at 6 → 6 - 3 = 3

3 + 4 = 7

Buy at 6

7 1 5 3 6 4

4 < 6

cancel at 6 — Buy at 4

Total profit = 7

→ first Day में stock को buy kar lenge.

↳ अगर Next day stock का price ज्यादा है तो sell कर देंगे और profit को add कर देंगे।

↳ phir usi din stock ko usi price pe buy kar lenge.

↳ फिर से check करेंगे।

↳ अगर Next day stock का price कम है तो buy price का change कर के current day के price को assign कर देंगे।

* Approach 2: (More optimal)

(2) (7) 11 3 8

7 - 2

= 5 (+ve)

profit = 5

2 (7) (11) 3 8

11 - 7

= 4 (+ve)

profit = 5 + 4 = 9

2 7 (11) (3) 8

3 - 11

= -8 (-ve)

✗

classmate
Date
Page

2 7 11 (3) (8)

8 - 3 = 5

(+ve)

profit = 9 + 5 = 14

Example 2:

(7) (1) 5 3 6 4

1 - 7

= -6 (-ve) X

7 (1) (5) 3 6 4

5 - 1

= 4 (+ve)

profit = 4

7 1 (5) (3) 6 4

3 - 5 = -2

-ve X

7 1 5 (3) (6) 4

6 - 3 = 3

+ve ✓

profit = 4 + 3 = 7

7 1 5 3 (6) (4)

4 - 6 = -2

-ve X

profit = 7

Code :

```
int maxprofit (vector <int> & prices) {

        int profit = 0;
        int n = prices.size();

    for (int i=1; i<n; i++){
        if( prices[i] > prices[i-1])
          profit += prices[i] - prices[i-1];
    }
    return profit;
```

classmate
Date
Page

Lecture 76

Stock + How to define row and column in dP

* **Subset sum problem**

Given : An array (of non negative integers)
Sum value

Task : Is there any subset exist whose sum is equal to given sum value.

Example : 3 34 4 12 5 2

Subset : Example
Subset of {1, 3} = {}, {1}, {3}, {1, 3}

$$\text{Subset} = 2^n$$

We can understand this with help of recursion tree.

Sum_Required = 9

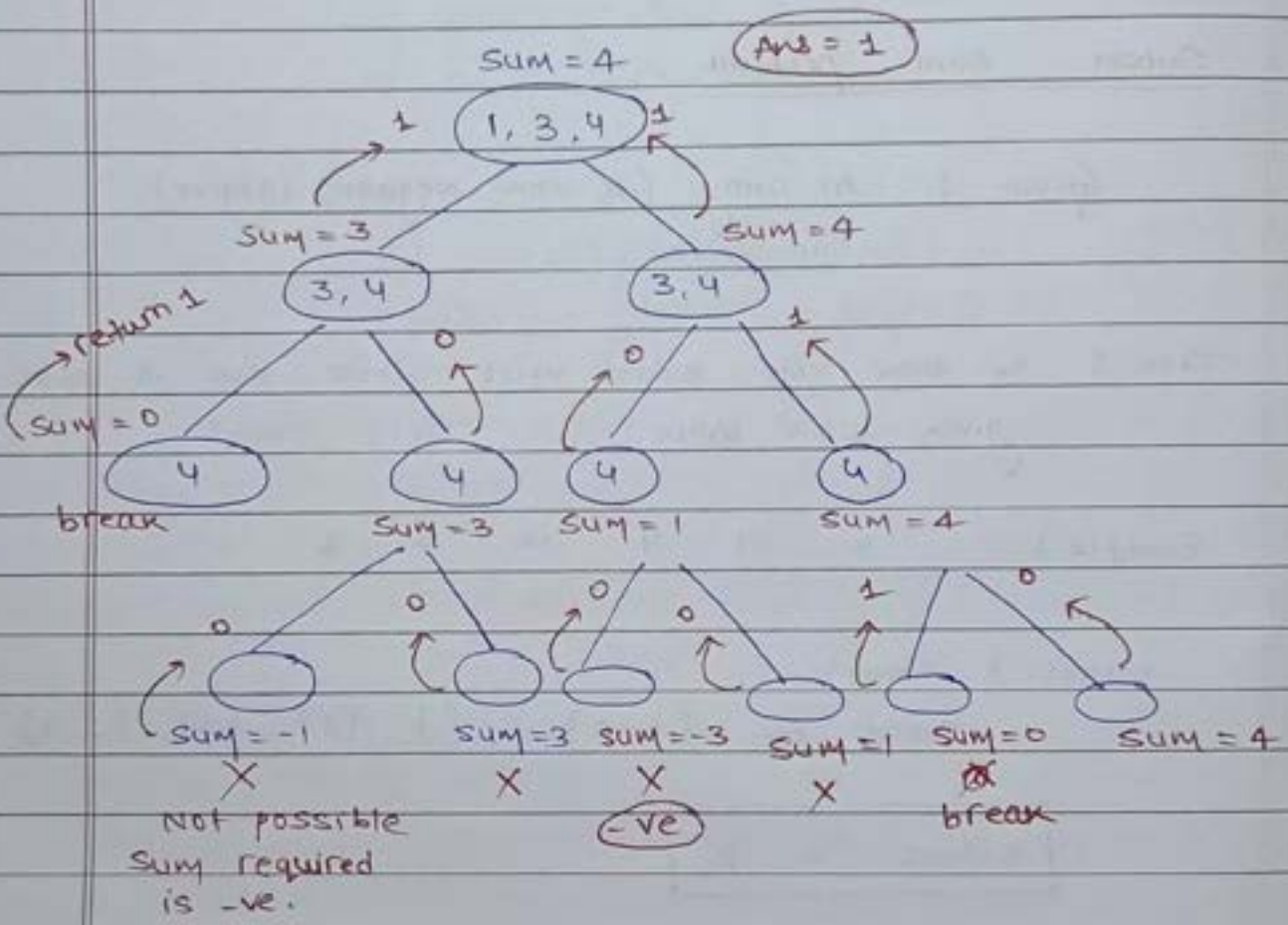
Here, we will decrease the sum_Required in each step and make sure the sum never be negative.

Check for all possible cases.

Understand with small Example :-

arr = {1, 3, 4}

Sum = 4



First we write the code of Recursion :

Recursion :

```
bool find (int n, int sum, vector<int> & arr) {

    if (sum == 0)
        return 1;

    if (n == 0)
        return 0;

    if (sum < arr[n-1])
        return find (n-1, sum, arr);

    else
        return find (n-1, sum - arr[n-1], arr) || find (n-1,
                                                  sum, arr);
}
bool isSubsetSum (vector<int> arr, int sum) {

        int n = arr.size();
    return find (n, sum, arr);
}
```

By dynamic programming: (Top Down Approach)

```
bool find (int n, int sum, vector<int>& arr, vector<vector<int>>& dp){
        if (sum == 0)
            return 1;

        if (n == 0)
            return 0;

        if (dp[n][sum] != -1)
            return dp[n][sum];

        if (sum < arr[n-1])
        return dp[n][sum] = find(n-1, sum, arr, dp);

        else
        return dp[n][sum] = find(n-1, sum - arr[n-1],
                    arr, dp) || find(n-1, sum, arr, dp);
}
bool isSubsetSum (vector<int> arr, int sum) {

        int n = arr.size();
        vector<vector<int>> dp(n+1, vector<int>(sum+1, -1));
        return find(n, sum, arr, dp);
}
```

Bottom up Approach:

```
if (sum == 0)
    return 1;
if (N == 0)
    return 0;
```

sum →

↓N	0	1	2	3	4	
0	1	0	0	0	0	→ N == 0 {return 0}
1	1	1	0	0	0	
2	1	1	0	1	1	
3	1	1	0	1	1	→ return this

sum == 0
return 1

Example:

{1, 3, 4}

sum = 4

In this we can fill in reverse order

Col : sum → 0 1

row : N → 0 1

loop : { 1 → N
       { 1 → sum

↓

Because we fill from down.

Code: Bottom up Approach:

```
bool isSubsetSum (vector <int> arr , int sum) {

        int n = arr.size();
    vector<vector<int> dp (n+1, vector<int> (sum+1, 0));

      for (int j= 0; j<= sum; j++)
            dp[0][j] = 0;

      for (int j=0;  j<= n;  j++)
            dp[j][0] = 1;

      for ( int i = 1; i <=n; i++) {
          for ( int j=1; j<= sum; j++ ) {
              if ( j < arr[i-1]) {
                  dp[i][j] = dp[i-1][j] ;
                }
            else{
             dp[i][j] = dp[i-1][j-arr[i-1]] || dp[i-1][j];
            }
        }
    }
    return dp[n][sum];
}
```

Next logic :

Recursion :

```
bool find (int index, int sum, vector<int>&arr, int n){
        if (sum == 0)
            return 1;
        if (index == N)
            return 0;
        if (sum < arr[index])
            return find (index + 1, sum, arr, n);
        else
        return find (index+1, sum - arr[index], arr, n) ||
                find (index + 1, sum, arr, n);
}
  bool isSubsetSum ( vector<int>arr, int sum) {

        int n = arr.size();
      return find (0, sum, arr, n);
}
```

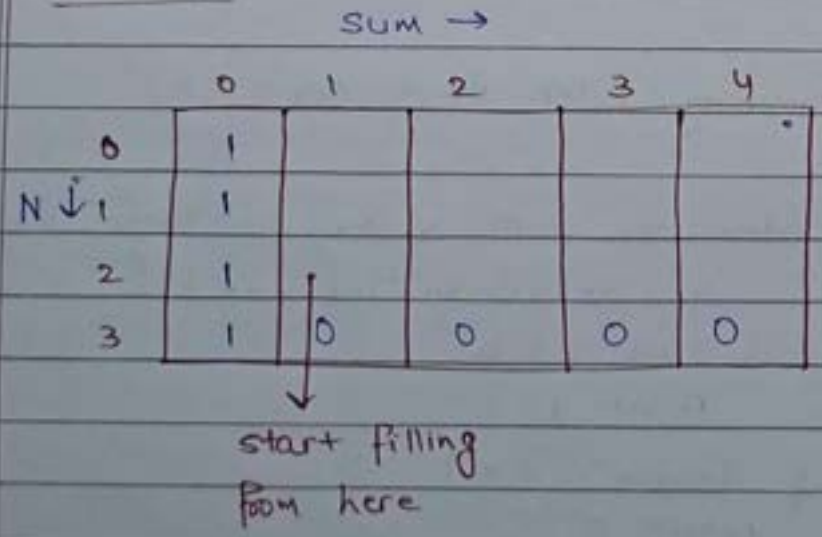
Dynamic programming (Top down Approach)

```
bool find (int index, int sum, vector<int>& arr, int n,
          vector <vector <int>> dp) {
        if (sum == 0)
            return 1;
        if (index == n)
            return 0;
        if (dp[index][sum] != -1)
            return dp[index][sum];
```

```
if ( sum < arr[index])
return dp[index][sum] = find (index+1, sum, arr,
                              n, dp);

else
return dp[index][sum] = find (index+1, sum - arr[index]
      arr, n, dp) || find (index+1, sum, arr, n, dp);

}
bool issubsetSum (vector<int>arr, int sum) {

    int n = arr.size();
vector <vector <int>> dp (n+1, vector<int> (sum+1, -1));
return find (0, sum, arr, n, dp);

}
```

Bottom up Approach :

Example :



classmate
Date
Page

~~↳ solved~~

index : 0 to N

sum : sum to 0

↳ solved.

index : 0 to N-1

sum : sum to 1

⇓

reverse

index : N-1 to 0

sum : 1 to sum

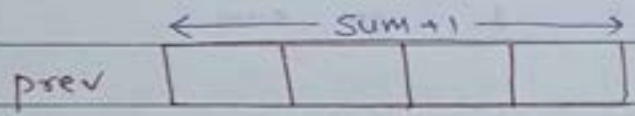
Code:

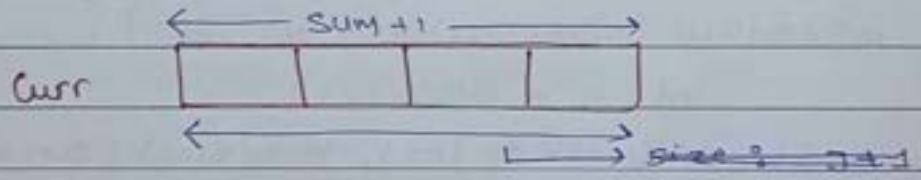
```
bool isSubsetSum (vector<int> arr , int sum) {
        int n = arr.size();
    vector< vector <int>> dp (n+1, vector<int> ( sum + 1 , 0));
    for (int j=0; j<= sum; j++)
      dp[n][j] = 0;

    for (int i=0; i<=n; i++)
      dp[i][0] = 1;

    for (int i = n-1; i >=0 ; i--) {
      for (int j = 1 ; j <= sum; j ++) {
          if (j < arr[i])
            dp[i][j] = dp[i+1][j];
          else
          dp[i][j]= dp[i+1][j-arr[i]] || dp[i+1][j];
      }
    }

    return dp[0][sum];
}
```

* Conversion of 2D DP to 1D DP :

```
if ( j < arr[i-1])
    dp[i][j] = dp[i-1][j]
else
dp[i][j] = dp[i-1][j-arr[i-1]] || dp[i-1][j]
```

curr — prev — prev





We use 2 row here only.

```
Vector <int> dp ( sum + 1);   // only prev dp initialised.
Vector <int> curr (sum+1);
```

code : (optimised)

```
bool isSubsetSum (vector<int> arr, int sum) {

    int n = arr.size(),
    vector <int> prev ( sum + 1, 0);
    Vector <int> curr ( sum + 1);

    prev[0] = 1;
```

```
        for (int i = 1;  i<=n;  i++) {
                for (int j= 1;  j<= sum ; j++) {
                    if (j< arr[i-1])
                        curr[j] = prev[j];
                    else
                    curr[j] = prev[j- arr[i-1]] || prev[j];
                }
                prev = curr;  Swap (curr, prev);
            }
        return curr[sum];
}
```

Here we use 2 row after optimisation.

→ Now we can optimise more to 1 row only.

We will fill from last, then we can done with only 1 row.

Code : (Highly optimised)

```
bool isSubsetSum ( vector<int> arr, int sum ) {
        int n = arr.size();
    vector <int> dp (sum + 1, 0);
    dp[0] = 1;
  for (int i = 0; i<n; i++)
    for (int j = sum; j>= arr[i] ; j--) {
        dp[j] = dp[j] || dp[j - arr[i]];
    }
  return dp[sum];
}
```

Best time to Buy and Sell Stock III

Given : An Array
↳ prices

prices[i] is the prices of a given stock on the i^{th} day.

Task : Maximum profit

We can do atmost two transaction.

Example :

{ 3 3 5 0 0 3 1 4 }

Buy → sell → Buy → sell

↓

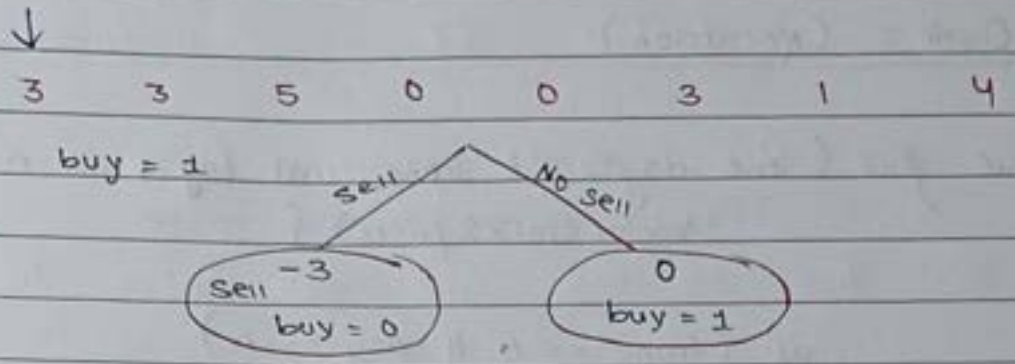
At most

2 transaction

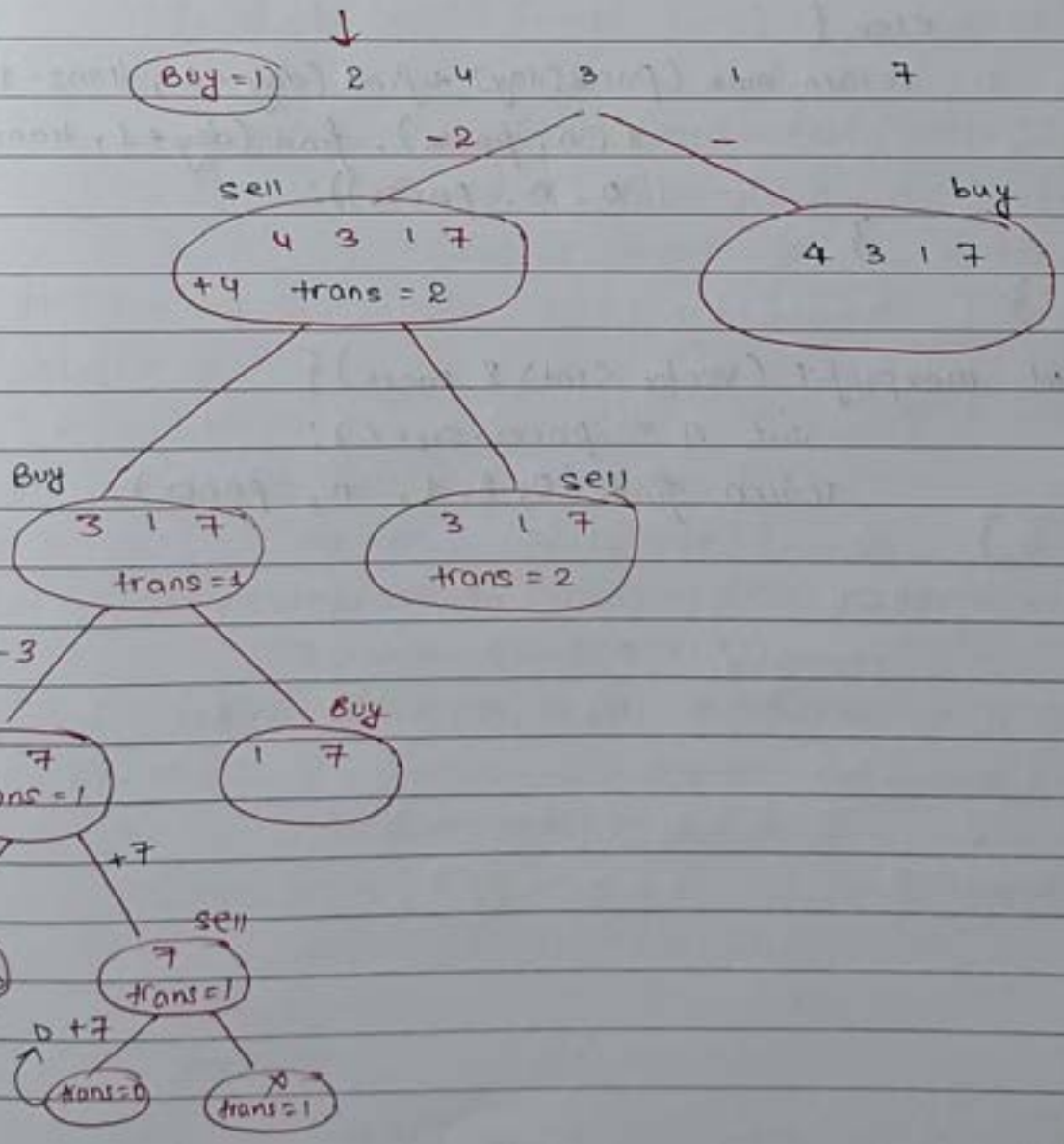
↓

{2 time}

buy = 1 → खरीदना है।
buy = 0 → बेचना है।




```
if (buy == 1)
max (-prices[i] + find(i+1, 0, Transaction),
                 find(i+1, 1, Transaction));
else
( prices[i] + find(i+1, 1, Transaction-1),
              find(i+1, 0, Transaction))
```



Code : (Recursion)

```
int find ( int day, int trans, int buy, int n,
            vector<int>& prices) {

    if (trans == 0 || day == n)
        return 0;

    if ( buy) {
    return max(-prices[day] + find (day + 1, trans, 0,
            n, prices), find ( day + 1, trans, 1, n,
            prices));
    }
    else {
    return max (prices[day] + find (day + 1, trans - 1,
            1, n, prices), find (day + 1, trans,
            0, n, prices));
    }
}
int maxprofit (vector <int> & prices) {
        int n = prices.size();
        return find (0, 2, 1, n, prices);
}
```

Code : Dynamic programming (Top down Approach)

```
int find (int day, int trans, int buy, int n, vector<int>& prices,
          vector<vector<vector<int>>>& dp){

    if (trans == 0 || day == n)
        return 0;
    if (dp[day][trans][buy] != -1)
        return dp[day][trans][buy];
    if (buy){
     return dp[day][trans][buy] = max(-prices[day] +
       find(day+1, trans, 0, n, prices, dp), find(
         day+1, trans, 1, n, prices, dp));
    }
    else {
     return dp[day][trans][buy] = max(prices[day] +
         find(day+1, trans-1, 1, n, prices, dp),
         find(day+1, trans, 0, n, prices, dp));
    }
}

int maxprofit (vector<int>& prices){
        int n = prices.size();
    vector<vector<vector<int>>> dp(n+1, vector<vector<int>>(
        3, vector<int>(2, -1)));
    return find(0, 2, 1, n, prices, dp);
}
```

Lecture - 77

Stock and player Game

* Best time to Buy and sell stock III

3 3 5 0 0 3 1 4

→ only 2 transaction possible

Example :

3	3	5	0	0	3	1	4
(3-3)	(3-3)	(5-3)	(prev)	(prev)	(3-0)	(prev)	(4-0)
profit on 1 transaction 0	0	2	2	2	3	3	4

profit on 2 transaction :

3 3 5 0 0 3 1 4

→ 0 0

0+0

3 3 5 0 0 3 1 4

1 transaction

↓ 5 - 3 = 2 (1 transaction)

value = 0 2 + 0 = (2)

classmate
Date
Page

3 3 5 0 0 3 1 4

2 transaction

↓ (0-5) = -ve (No transaction)

profit = 2

Total profit = 2

3 3 5 0 0 3 1 4

1 transaction

↓ 0-0 = 0 (1 transaction)

profit = 2

Total profit = 2+0 = 2

0 0 -2 3

3 3 5 0 0 3 1 4

(3-0)

1 transaction

3-0 = 3 (1 transaction)

profit = 2

Total profit = 3+2 = 5

-2 -2 -4 1 1 -2

3 3 5 0 0 3 1 4

Max

1 transaction

1-0 = 1 (1 transaction

profit = 3

Total profit = 4

1 1 -1 Ans = 5 4 1 3

4

3 3 5 0 0 3 1 4

Max

1 transaction

4-0 = 4 (1 transaction)

profit = 2

Total profit = 6

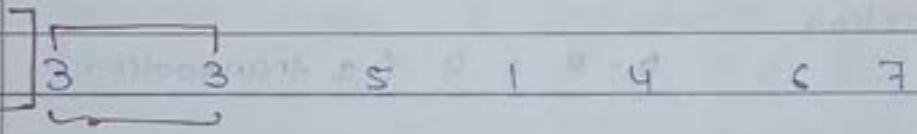
↳ Ans

Example 2:

3 3 5 1 4 6 7

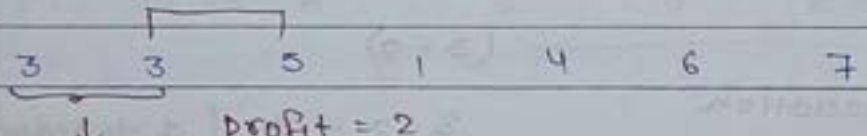
1 trans: 0 0 2 2 3 5 6

For 2nd transaction:



0 profit = 0

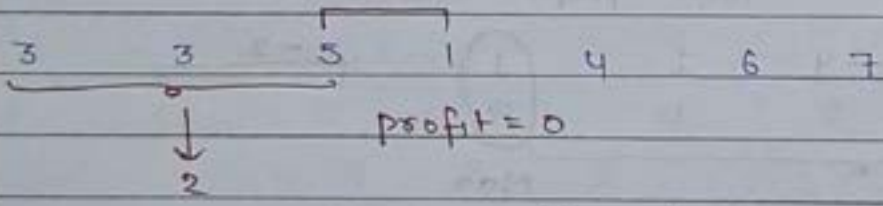
Total profit = 0 + 0 = 0



profit = 2

0

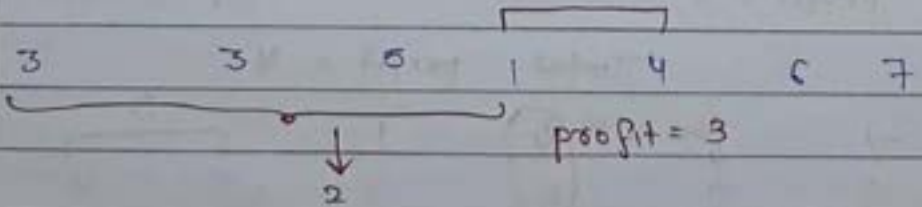
Total profit = 2



profit = 0

2

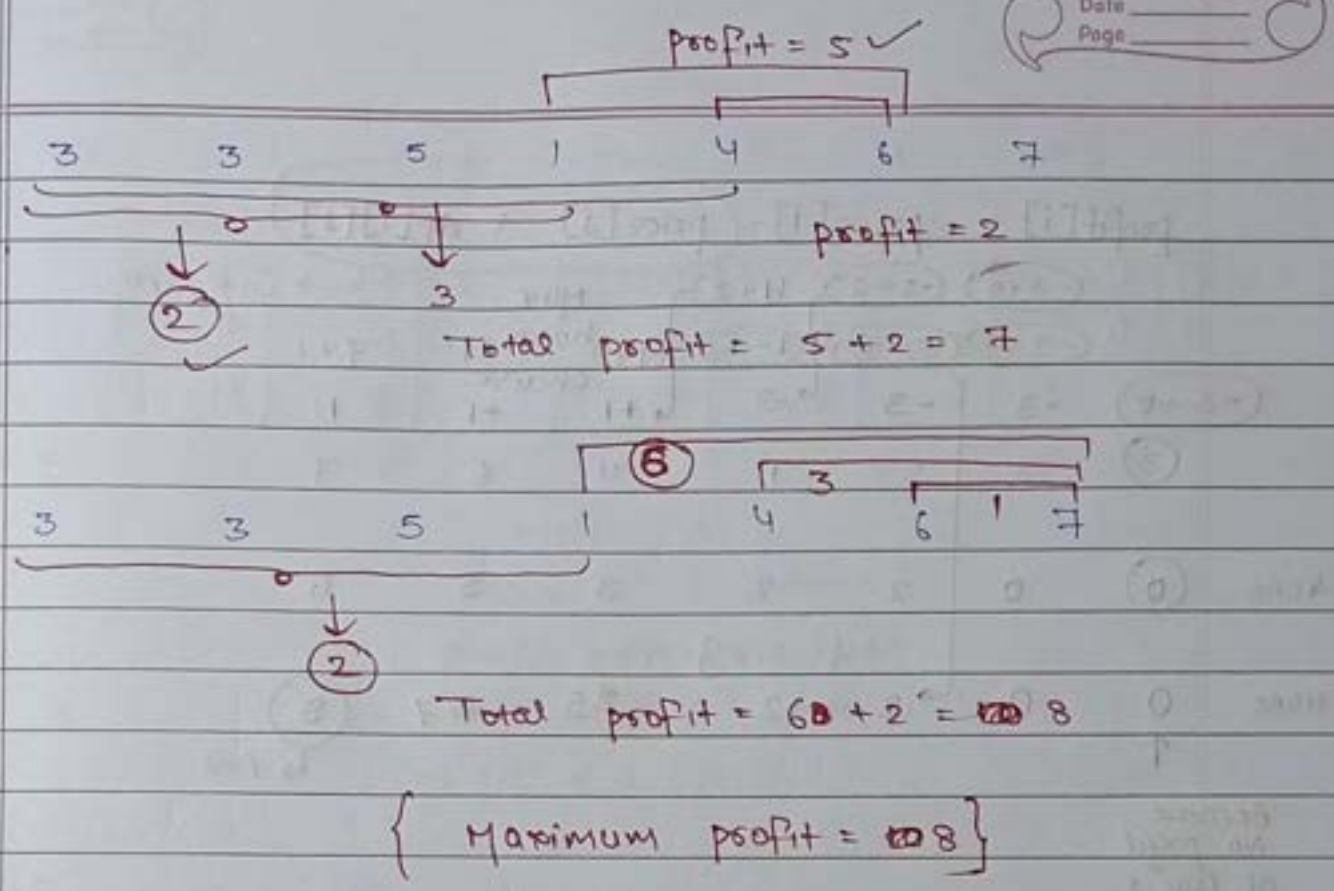
Total profit = 2



profit = 3

2

Total profit = 5



{ Maximum profit = 8 }

Time Complexity : $N^2 \times 2$

$\Downarrow\Downarrow$

$2N$

↳ How to achieve this.

For the new transaction, we compare with all the previous day. We have to minimise that comparison.

Humlog phir se minimum choose karte jayenge 2 transaction ke liye.

				j	i↓		
	3	3	5	1	4	6	7
1	0	0	2	2	3	5	6
2	0	0					

profit[i] = price[i] - (price[j] + dp[i][j]) → Compare.

(-3+0) (-5+2) (4+2) Max hona chaiye

(-3+3) (5-3) (1-3) (4+1) 6+1 7+1

	(-3+0)	-3	-3	-3	+1	+1	1
	(3)	3	5	1	4	6	7
1 trans	(0)	0	2	2	3	5	6
2 trans	0 ↑	0	{(-2,2)} 2 → 2		~~5~~	~~7~~	(8) → Ans.

Because No profit at day 1.

We can understand this in detail.

	3	3	5	1	4	6	7
1 trans:	0	0	2	2	3	5	6

2 trans:

(3) 3 5 1 4 6 7

-3+0
= -3

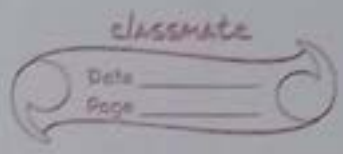
profit = 0

3 (3) 5 1 4 6 7

-3

-3+3 = 0 (profit)

-3+0 = -3 (आगे जाएगा)



3 3 (5) 1 4 6 7

-3

-3+5 = 2 (profit)

-5+2 = -3 (आगे जायेगा)

3 3 5 (1) 4 6 7

-3

-3+1 = -2 (Negative)

profit → previous wala

profit = 2

-1+2 = 1 (proceed)

3 3 5 1 (4) 6 7

1

1+4 = 5 (profit)

-4+5 = 1 (आगे जाएगा)

3 3 5 1 4 (6) 7

1

1+6 = 7 (profit)

-6+7 = 1 (आगे जाएगा)

3 3 5 1 4 6 (7)

1

7+1 = 8 (profit)

-7+8 = 1

{ Maximum profit = 8 }

	3	3	5	1	4	6	7
0	0	0	0	0	0	0	0
1	0	0	~~0~~ 2	~~0~~ 2	~~0~~ 3	~~0~~ 5	~~0~~ 6
2	0	0	~~0~~ 2	~~0~~ 2	~~0~~ 5	~~0~~ 7	~~0~~ 8

↓ Answer.

Code: (Bottom up Approach)

```
int maxprofit (vector<int>& prices) {

    int n = prices.size();
    vector < vector <int>> dp (3, vector <int>(n+1, 0));

    int total;
    for (int i = 1; i <= 2; i++) {
        total = -prices[0];

        for (int j = 2; j <= n; j++) {
            dp[i][j] = max (dp[i][j-1], prices[j-1] + total);
            total = max (total, -prices[j-1] + dp[i-1][j]);
        }
    }
    return dp[2][n];
}
```

* Best time to Buy and sell stock IV

In this problem, we have to find Maximum profit after doing K transaction.

K is given in question.

Code:

```
int maxProfit ( int K, Vector<int>& prices) {
        int n = prices.size();
    Vector<vector<int>> dp(K+1, vector<int>(n+1, 0));

     int total = 0;

    for ( int i = 1;  i <= K;  i++) {

            total = -prices[0];
        for(int j = 2;  j <= n;  j++) {
            dp[i][j] = max (dp[i][j-1], prices[j-1] + total);
            total = max (total, -prices[j-1] + dp[i-1][j]);
    }
}
    return dp[K][n];
}
```

* player With Max Score :

Given : An array (Non-Negative)
↳ Size : N

2 players → plays Game.

player 1 start the Game.
↳ can choose one element from end and the element will be added to the score. Then player 2 can choose.

Task : check that player 1 will win or not

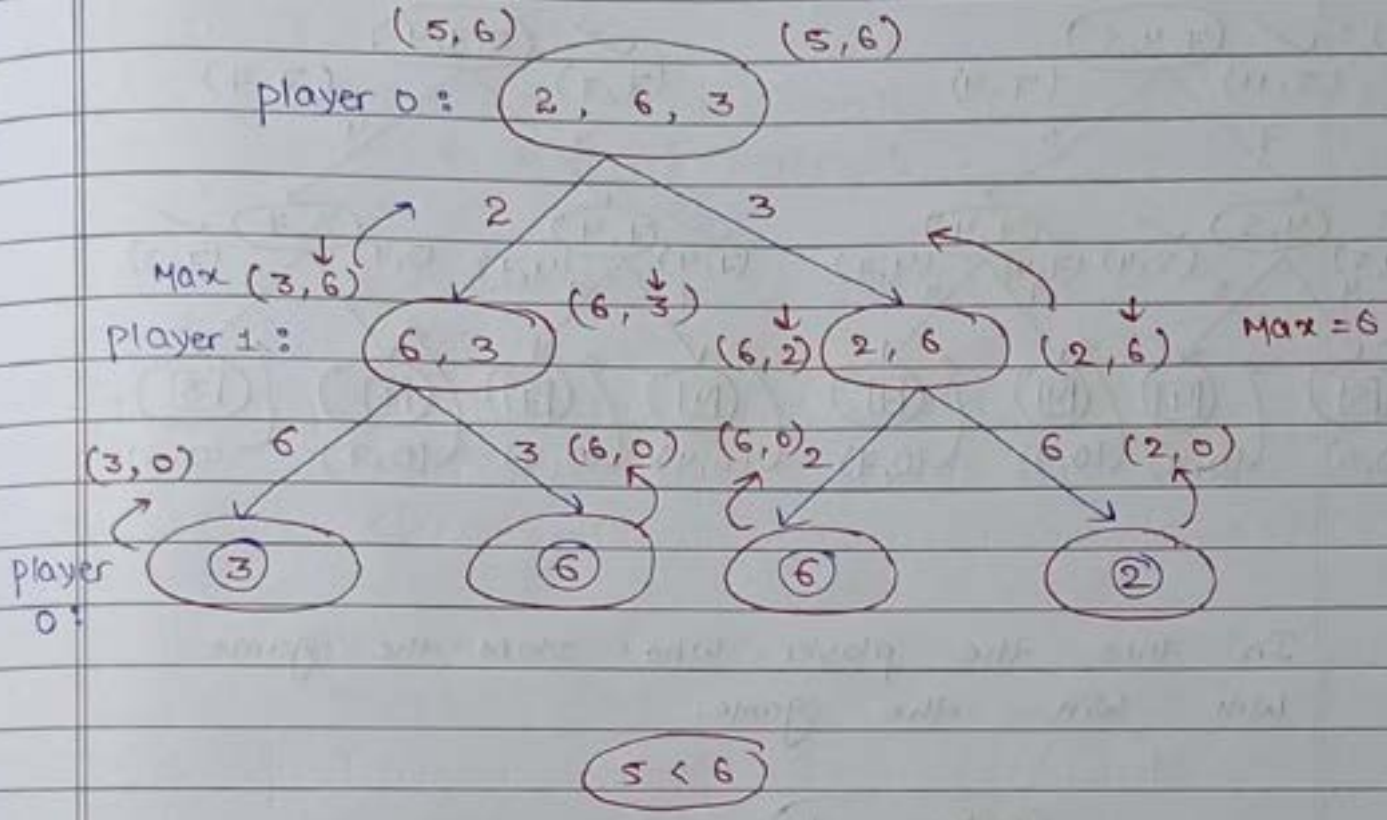
Example:

2 ~~8~~ ~~2~~ ~~4~~ ~~6~~

player 1 (start)	player 2
(1) Max (2, 5) = 5	(2) Max (2, 4) = 4
(3) Max (2, 3) = 3	(4) Max (2, 6) = 6
(5) Max (2) = 2	
Total = 10	Total = 10

Match Tie
return 0

But the approach is wrong. We can understand this by recursion tree.



5 < 6

There is No possibility to win player 1 if both play optimally.

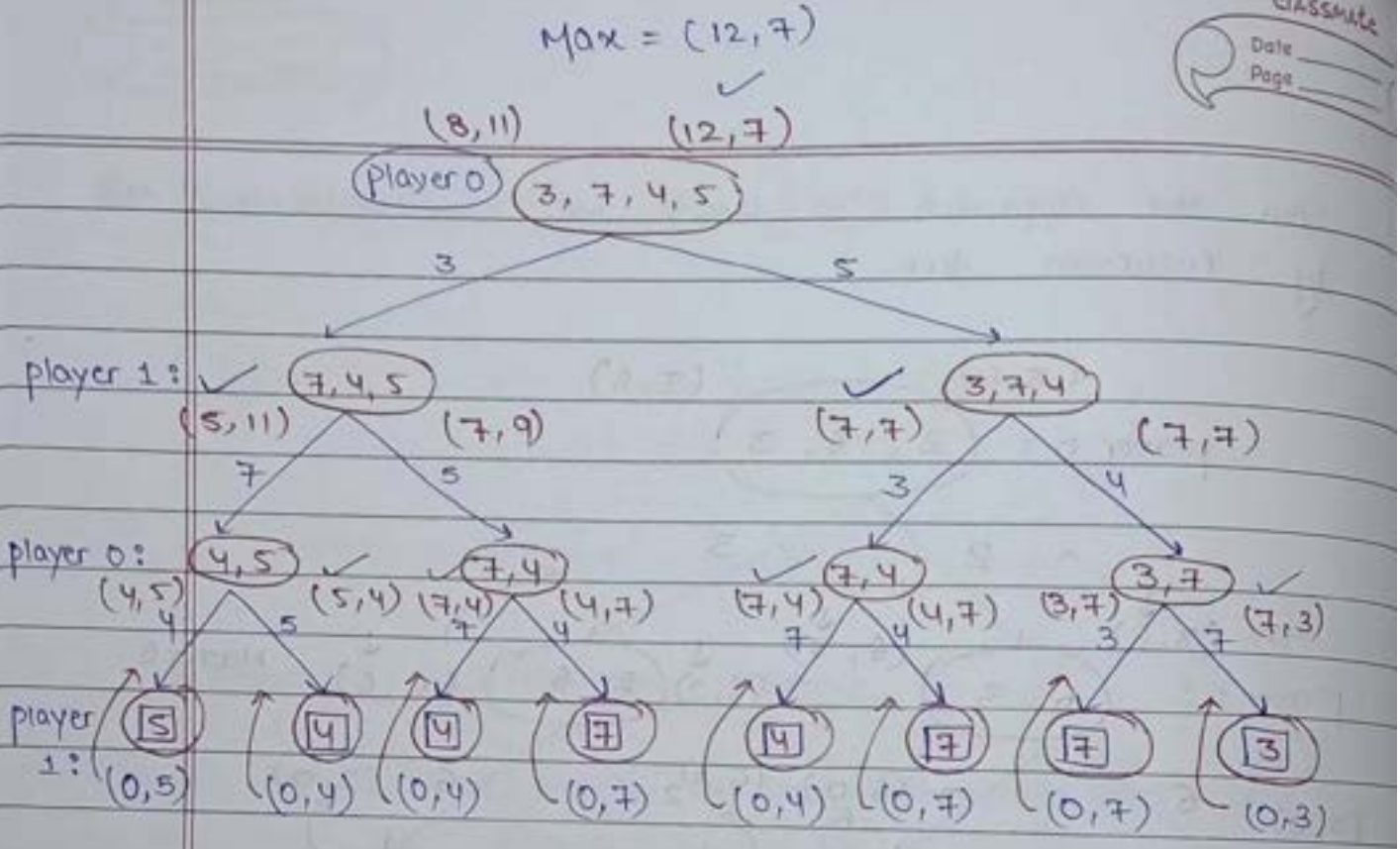
Example 2:

{ 3 7 4 5 }

We will understand this problem with this Example.

player 0 & player 1 are 2 player

player 0 can start the game.



In this the player who start the game will win the game.

return 1

Recursion code:

```
pair<int, int> find(int start, int end, int player,
                    int arr[]){
    if (start == end){
        if (player == 0)
            return {arr[start], 0};
        else
            return {0, arr[start]};
    }

pair<int, int> score1, score2;
```

```
    if (player == 0) {
        score1 = find (start + 1, end, 1, arr);
        score2 = find (start, end - 1, 1, arr);

        score1.first += arr[start];
        score2.first += arr[end];

        if (score1.first > score2.first)
            return score1;
        else
            return score2;
    }
    else {
        score1 = find (start + 1, end, 0, arr);
        score2 = find (start, end - 1, 0, arr);

        score1.second += arr[start];
        score2.second += arr[end];

        if (score1.second > score2.second)
            return score1;
        else
            return score2;
    }
}
bool is1Winner (int N, int arr[]) {
    pair <int, int> score;
    score = find (0, N - 1, 0, arr);
    return score.first > score.second;
}
```

Dynamic programming : (Top down Approach)

```
pair<int, int> find (int start, int end, int player,
        int arr[], vector<vector<vector<pair<int, int>>>> dp){

    if (start == end){
        if (player == 0)
            return {arr[start], 0};
        else
            return {0, arr[end]};
    }

    if (dp[start][end][player].first != -1)
        return dp[start][end][player];

    pair<int, int> score1, score2;

    if (player == 0)
        score1 = find (start+1, end, 1, arr, dp);
        score2 = find (start, end-1, 1, arr, dp);

        score1.first += arr[start];
        score2.first += arr[end];

        if (score1.first > score2.first)
            return dp[start][end][player] = score1;
        else
            return dp[start][end][player] = score2;
    }
    else {
        score1 = find (start+1, end, 0, arr, dp);
        score2 = find (start, end-1, 0, arr, dp);
```

```
            score1.second += arr[start];
            score2.second += arr[end];

        if (score1.second > score2.second)
            return dp[start][end][player] = score1;
        else
            return dp[start][end][player] = score2;
    }
}
bool is1Winner(int N, int arr[]) {

    pair<int, int> score;
vector<vector<vector<pair<int, int>>>> dp(N, vector<vector<pair<int,
    int>>>(N, vector<pair<int, int>>(2, {-1, -1})));

    score = find(0, N-1, 0, arr, dp);

    return score.first > score.second;
}
```

Lecture 78

Dp Advance Question

* **Egg Dropping puzzle :**

Given : → You have N eggs
→ K - Floored building from 1 to K.

There exist a floor f where $0 <= f <= k$ such that any egg dropped a floor higher than f will break.

Any egg dropped at or below floor f will not break.

Rules :

→ An egg that survives a fall can be used again.
→ A broken egg must be discarded.
→ The effect of a fall is same for all egg.
→ If egg doesn't break at a certain floor, it will not break at any floor below.
→ If the eggs break at certain floor, it will break at any floor above.

Return minimum number of moves that you need to determine with certainty what the value of f is.

Example :

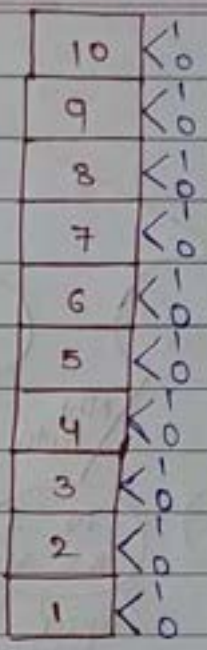
N = Egg, K = floor

N = 2 , K = 10

→ Agr Egg fut gya to dubara use nahi hoga.

nahi

→ Agar Egg ∧ tuta to same egg ko hm phir se use kar sakte hai.



Case 1 : N = 1 , K = 10

```
if (N == 1)
    return K;
```

→ Ek-Ek kar sare floor pe check karenge jab tak anda na fut jaye ya last take na pahunch jaye.

case 2 : N = 2, K = 10

1 : Nahi tuta

0 : fut gya

Jisme Anda fut jaya to :

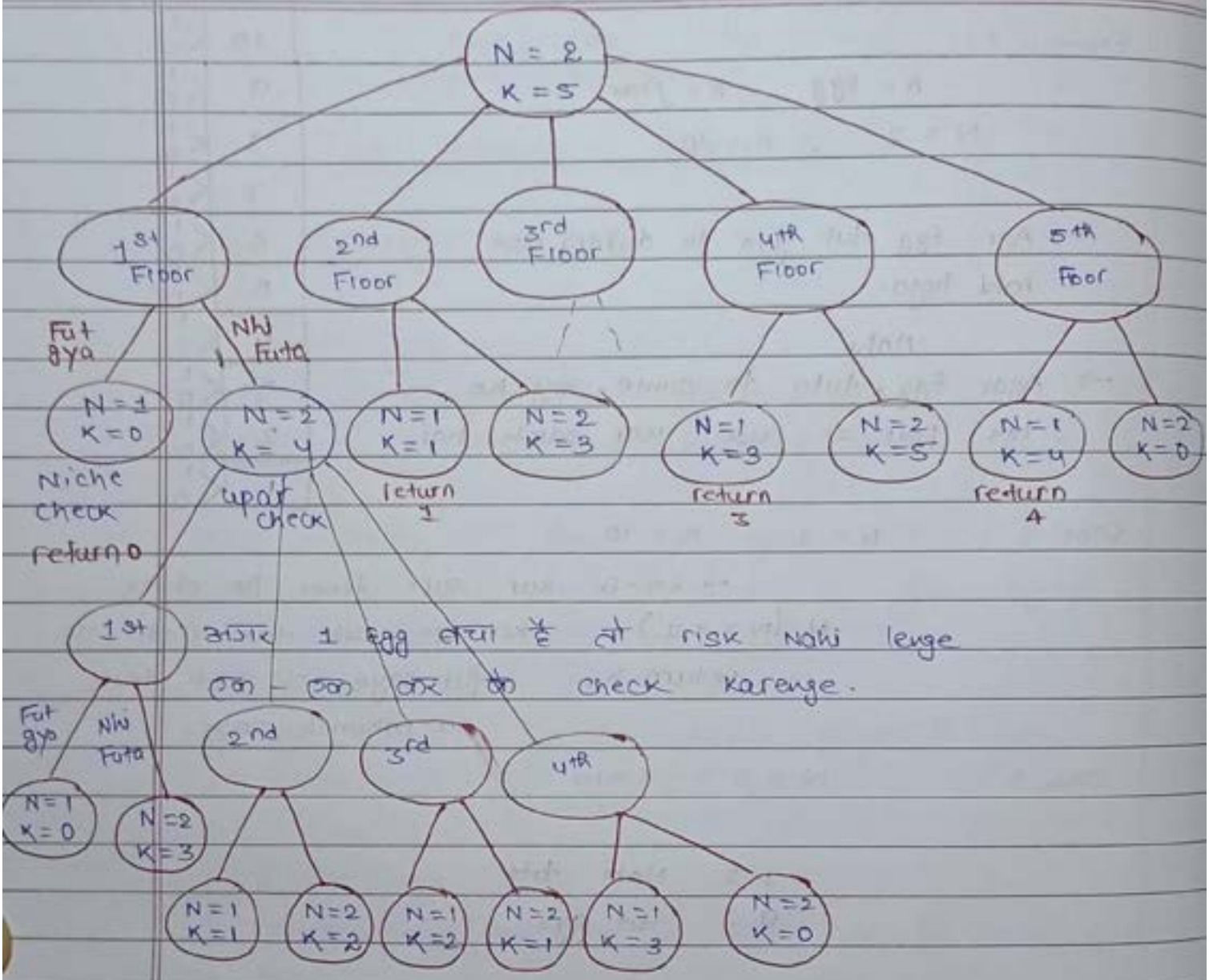
1 + find (N-1, K-1)

Jisme Anda Nahi tuta to :

1 + find (N, above floor)

2 Egg, (1-5) buildings

N = 2, K = 5



अगर 1 egg बचा है तो risk Nahi lenge
एक - एक कर के check karenge.

```
if (N == 1)
    return K;

if (K == 0)
    return 0;
```

→ अगर tut gya तो नीचे जा कर check karo
→ अगर नहीं tuta तो upar जा कर check करो।

Recursion (code) :

```
int find (int eggs, int floors){vector<vector<int>> ~~dp~~
    if (eggs == 1)
        return floors;

    if (floors == 0)
        return 0;

    ~~if (dp[eggs][floors] != -1)~~
    ~~return dp[eggs][floors];~~

    int ans = INT_MAX;
    int temp;
    for(int i = 1; i <= floors; i++) {
        temp = 1 + max (find (eggs-1, i-1, ~~dp~~), find (eggs,
                    floors - i, ~~dp~~));
        ans = min (ans, temp);
    }
    return ans;
}

int eggDrop (int n, int k) {

    return find (n, k);
}
```

Top Down Approach (Dynamic programming):

```
int find (int eggs, int floors, vector <vector <int>> & dp) {

        if (eggs == 1)
            return floors;

        if (floors == 0)
            return 0;

        if (dp[eggs][floors] != -1)
            return dp[eggs][floors];

        int ans = INT_MAX;
        int temp;

    for (int i = 1; i <= floors; i++) {
        temp = 1 + max (find (eggs - 1, i - 1, dp),
                        find (eggs, floors - i, dp));
        ans = min (ans, temp);
    }
    return dp[eggs][floors] = ans;
}
  int eggDrop (int n, int k) {

    vector <vector <int>> dp (n+1, vector <int> (k+1, -1));
     return find ( n, k, dp);
}
```

classmate
Date
Page

Bottom up Approach (Dynamic programming):

```
int eggDrop (int n, int k) {

    vector<vector<int>> dp (n+1, vector<int> (k+1, 0));

    for (int j = 0; j<=k; j++) {
        dp[1][j] = j;
    }

    for (int i = 2; i<=n; i++) {
        for (int j = 1; j<=k; j++) {

            int ans = INT_MAX;
            int temp;
            for (int a = 1; a<=j; a++) {
                temp = 1 + max (dp[i-1][a-1], dp[i][j-a]);
                ans = min (ans, temp);
            }
            dp[i][j] = ans;
        }
    }
    return dp[n][k];
}
```

Longest Increasing Path in a Matrix

Matrix with n rows and m Columns.

Find the length of longest increasing path in Matrix

Increasing path means the value in specified path increases.

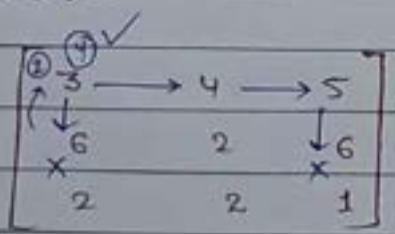
Example:

if a path of length k has value $a_1, a_2, a_3 \ldots a_k$, then for every i from {2 - K} must hold $a_i > a_{i-1}$.

From each cell, you can move in four direction.

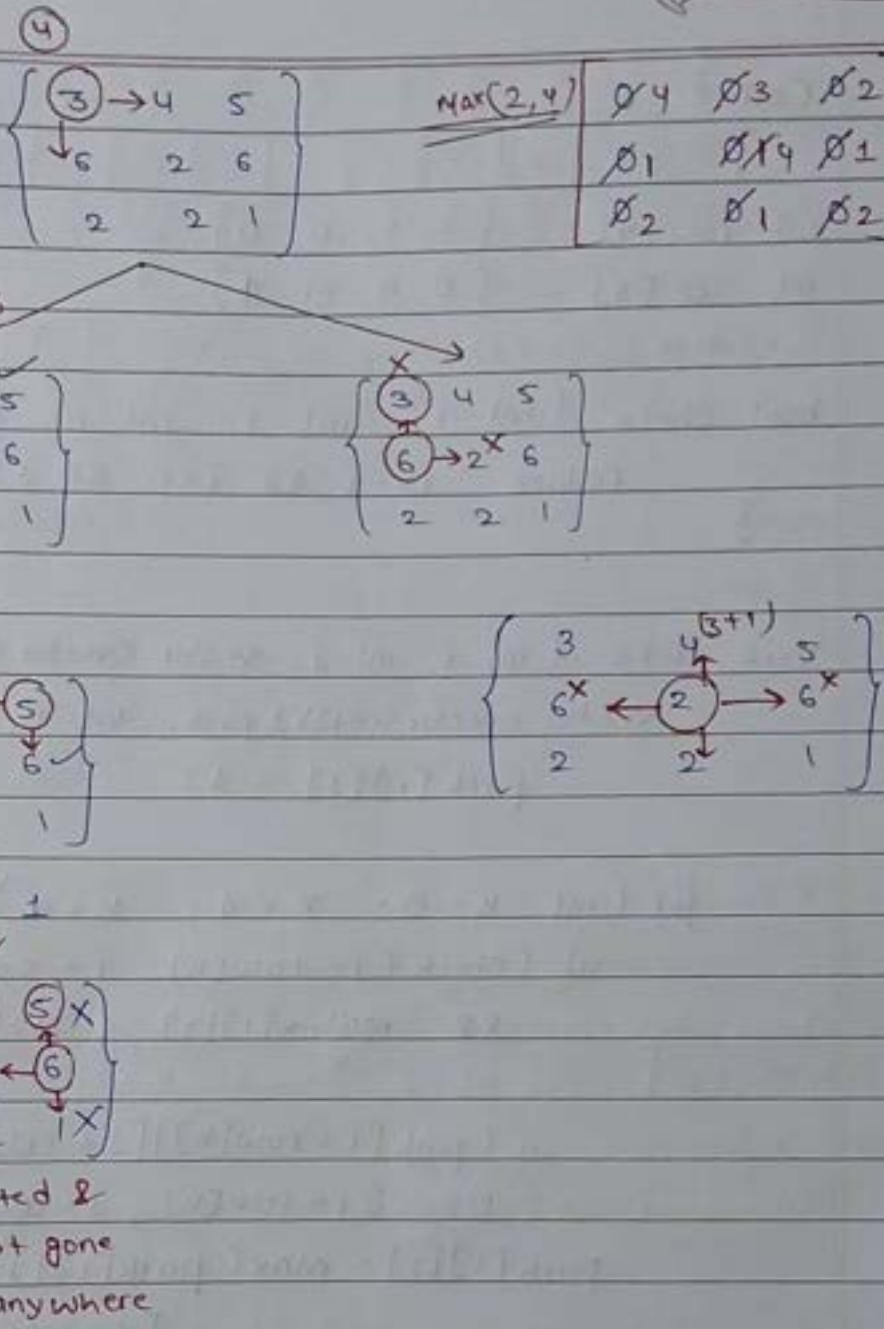
Not allowed to move dragonally or move outside the Boundary.

Example :-



path

0	0	0
0	0	0
0	0	0



We can make an extra array for storing the value of longest path in Matrix.

We can choose the answer which is maximum from all the array. (matrix).

Code:

```
int row[4] = {-1, 1, 0, 0};
int col[4] = {0, 0, -1, 0};

bool check (int i, int j, int n, int m) {
        return i>-1 && i<n && j>-1 && j<m;
    }

void DFS (int i, int j, vector <vector <int>> & matrix,
        vector <vector<int>> & path, int n, int m) {
            path[i][j] = 1;

    for (int k=0; k<4; k++) {
        if (check (i+row[k], j+col[k], n, m)
            && matrix[i][j] < matrix[i+row[k]][j+col[k]]
        {
            if (path[i+row[k]][j+col[k]] == 0)
                DFS (i+row[k], j+col[k], matrix, path, n, m)
            path[i][j] = max (path[i][j], 1+path[i+row[k]]
                                [j+col[k]]);
        }
    }
}

int longestIncreasingPath (vector <vector <int>> & matrix,
                    int n, int m) {

    Vector <vector <int>> path (n, vector<int> (m, 0));
                int total = 0;
```

```
for(int i = 0; i<n; i++){
    for(int j=0; j<m; j++){
        if(path[i][j] == 0)
        DFS(i, j, matrix, path, n, m);
        total = max(total, path[i][j]);
    }
}
return total;
}
```

* **Word Break:** (leetcode)

Given: → A String s
→ Dictionary of strings WordDict

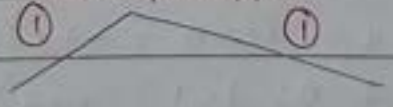
Task: return true if s can be segmented into a space separated sequence of one or more dictionary word.

Example:

dict: { apple, pen }

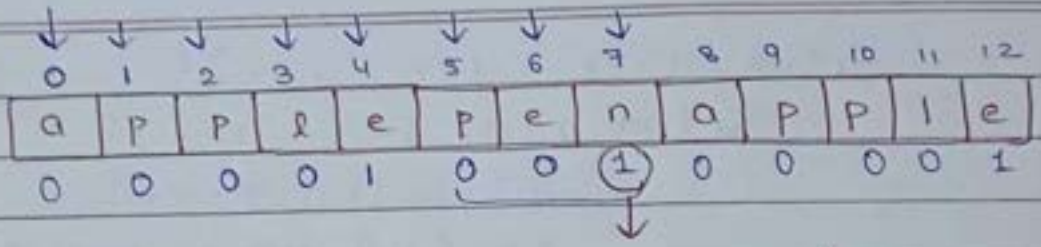
① true.

applepenapple



apple penapple ①

pen apple



0 → Exist (No)
1 → Exist (Yes)

Yaha 1 tab hoga jab us Word ke pahle wala Word bhi Exist karta ho.

pen → (5-7)
7 pe 1 tab ho jab 4 pe 1 ho.

Code:

```
bool WordBreak(string s, Vector<string>& WordDict){
        int n = s.length();

unordered_set<string> Wordset(WordDict.begin(), WordDict.end());

    Vector<bool> dp(n+1, false);

        dp[0] = true;

for(int i=1; i<=n; ++i){
    for(int j=0; j<i; ++j){
        if(dp[j] && Wordset.find(s.substr(j, i-j)) !=
                                    WordSet.end()){
            dp[i] = true;
            break;
        }
}
```

classmate
Date
Page

```
    }
    return dp[n];
}
```

```
    }
    return dp[n];
}
```

Lecture 79

Trie and it's implementation

Trie Data Structure :

String : {"apple", "appex", "almond", "money", "mon"}

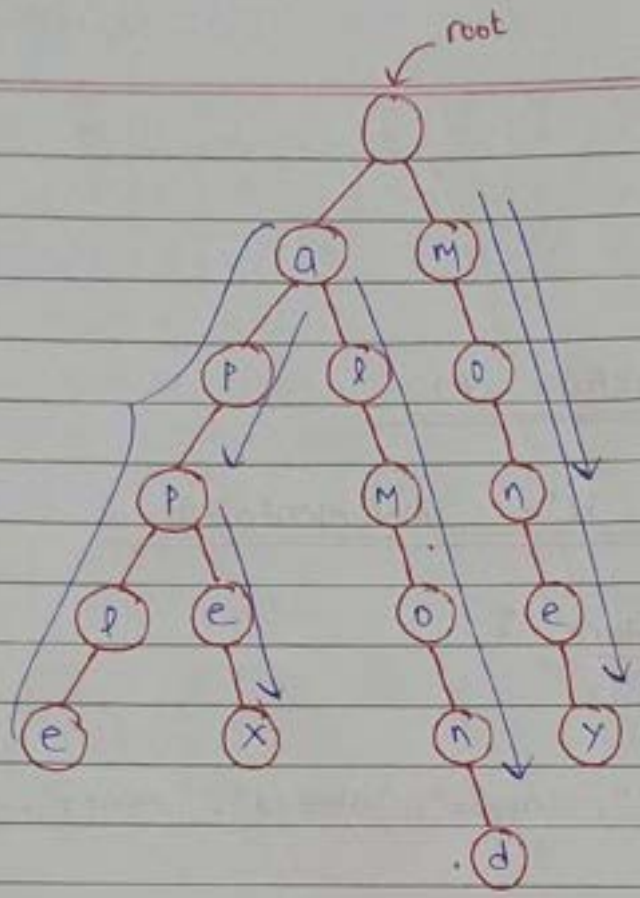
↳ If we use Hashmap
  ↳ We can fetch, delete in O(1) time.
  ↳ But in worst case it may also take O(N) time.

Trie :
↳ prefix Matching
↳ Retrieval

string : {"apple", "appex", "almond", "money", "mon"}

Let we can see how trie can store the data.

We can see all the string present in array using trie data structure.



When we use Hashmap, it take more memory because we can store all the elements or words uniquely in different - different location.

But, in Trie we can use reused memory we can Match the prefix, If prefix is same then we can't store that element uniquely.

Real life Example:

1) Contacts in sim card:

In Sim Card memory is very less. So, Saving the name of person it can use trie.

Time Complexity :

Insertion : $O(l)$ $l \rightarrow$ length of string

Searching :

Search : ① Mon
- ↳ search in diagram
  - ↳ True
- ↳ search in array
  - ↳ True

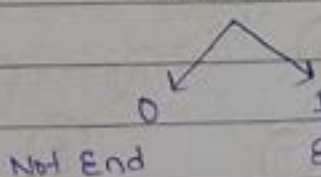
② appo
- ↳ search in diagram
  - ↳ False
- ↳ search in array
  - ↳ False

③ Mone

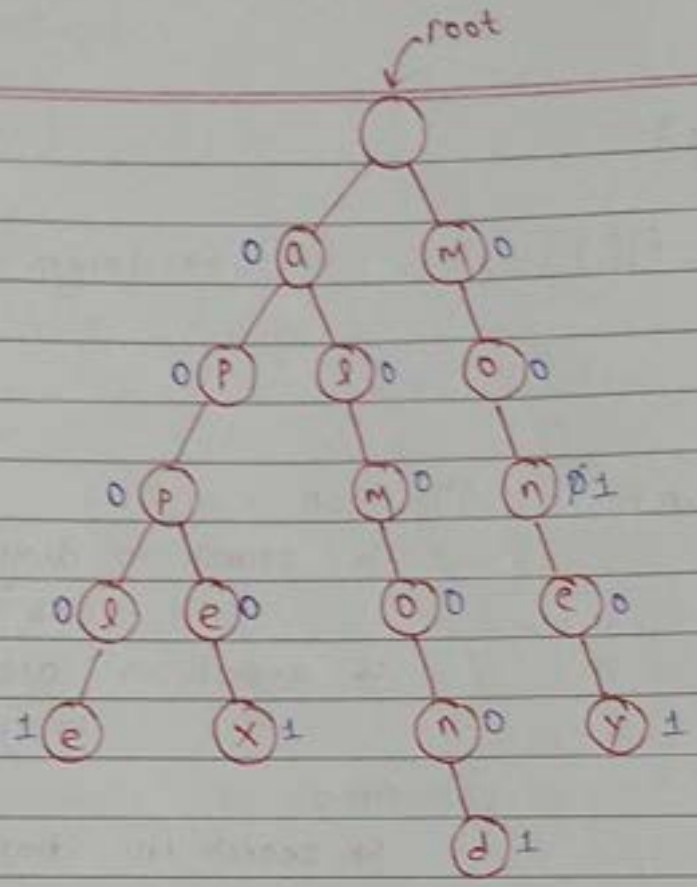
Here, it gives Wrong answer while searching in diagram.
- ↳ search in diagram
  - ↳ True
- ↳ search in array
  - ↳ False

So, we have to update something

↳ isEndOfWords



Lets recreate the diagram.



Search : (1) mone

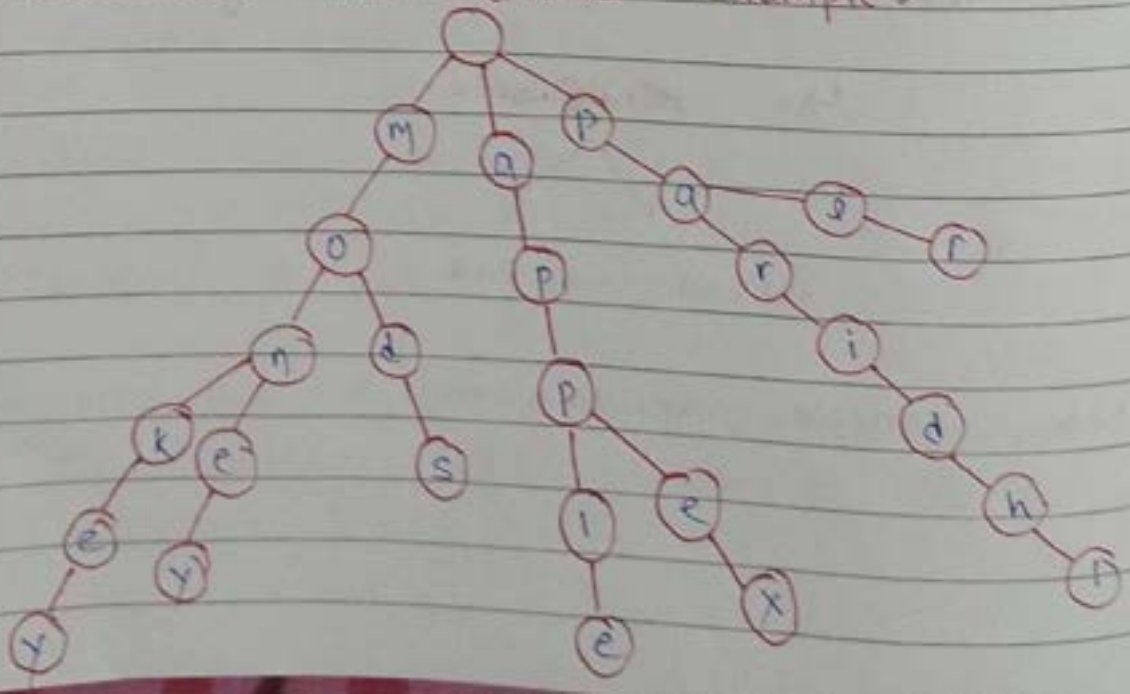
↳ diagram

↳ True & 1 present at last

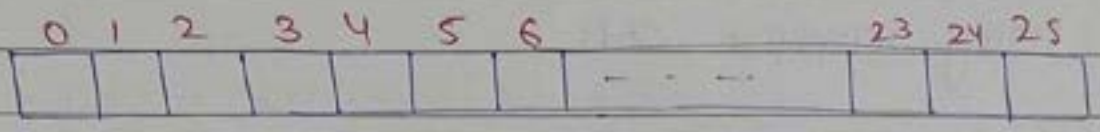
↳ false

By this approach we can get correct answer.

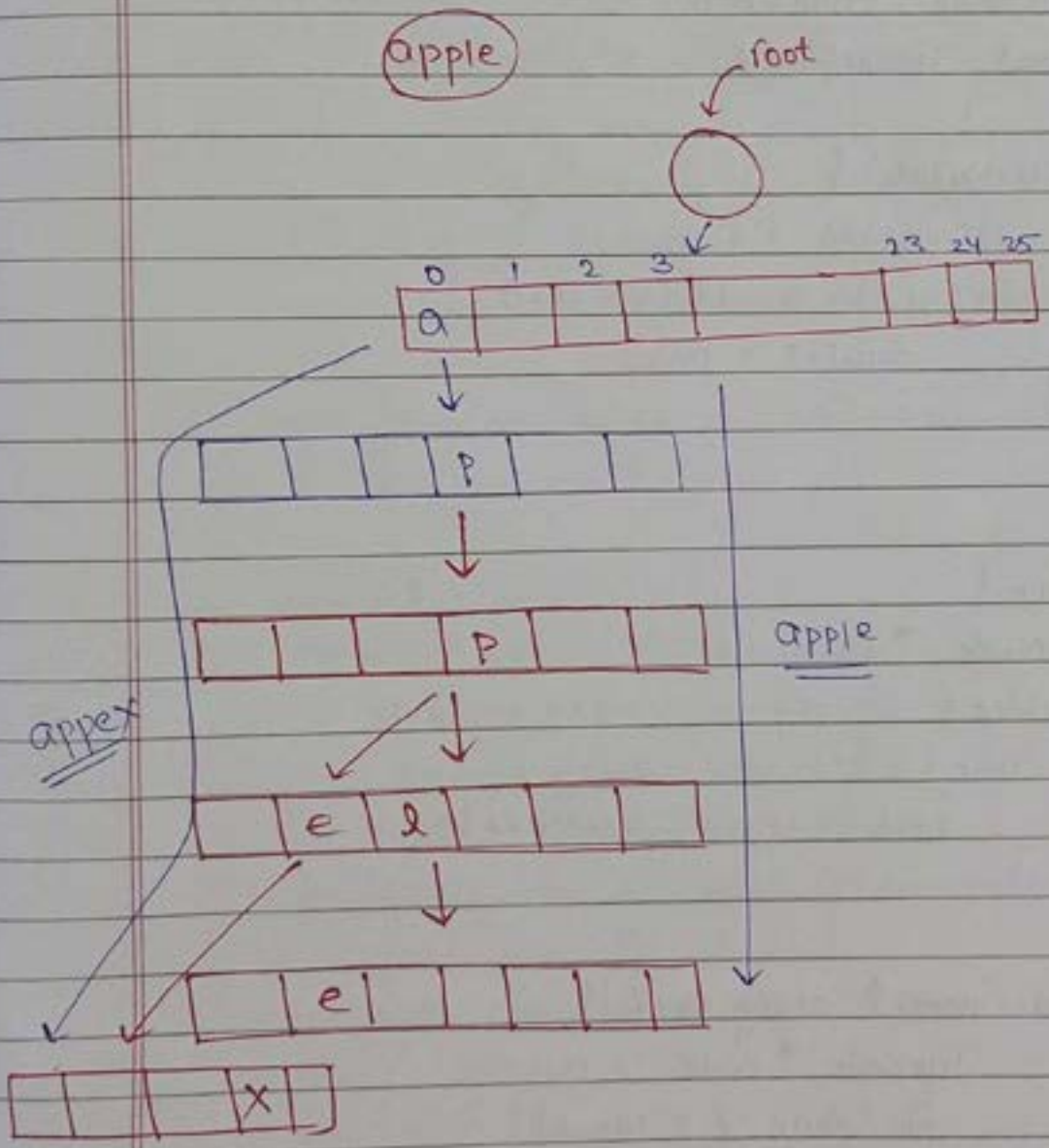
→ understand With another Example :



At first level, there is possibility of 26 letter.
So, we make a array of size 26 of 1st level.



Example:

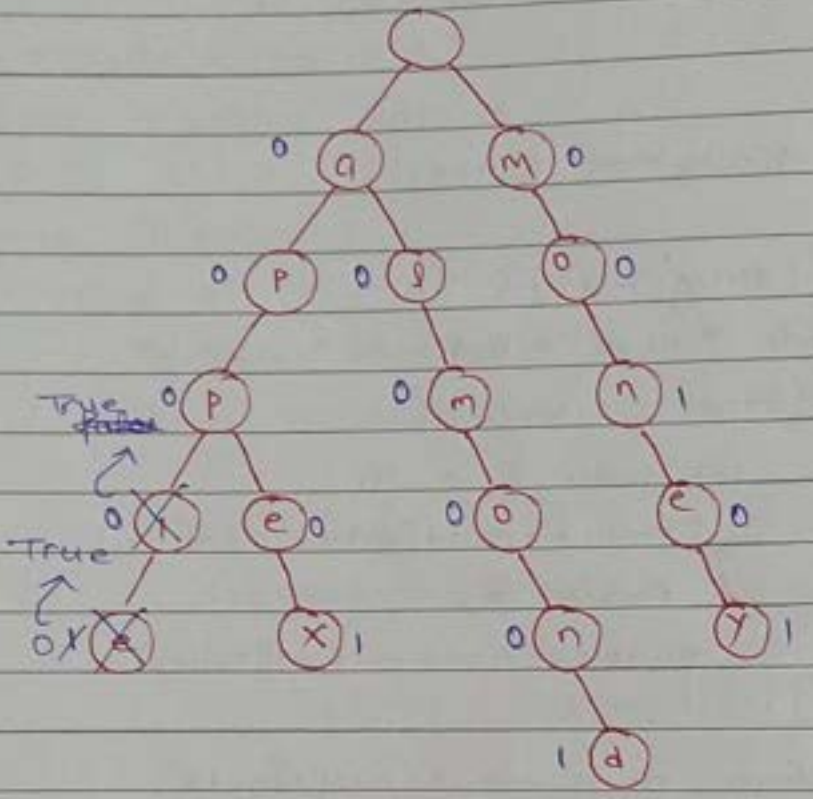


Code :

```
# include <iostream>
Using namespace std;

class TrieNode {
    public :
        TrieNode * child[26];
        bool isEndOfWord;

        TrieNode() {
            isEndOfWord = false;
            for(int i = 0; i<26; i++)
                child[i] = NULL;
        }
};

Class Trie {
    TrieNode * root;
    public;
        Trie () {
            root = new TrieNode ();.
        };

    Void insert ( string word ){
        TrieNode * node = root;
        for (char c : Word) {
            int index = c - 'a';
        if ( node → child [index] == NULL) {
            node → child [index] = new TrieNode();
            node = node → child [index];
        }
```

```
        else {
            node = node->child[index];
        }
    }
    node->isEndofWord = true;
}
bool search(string Word) {
    TrieNode *node = root;
    for(char c: Word) {
        int index = c - 'a';
        if(node->child[index] == NULL)
            return 0;
        node = node->child[index];
    }
    return node->isEndofWords;
};
};
int main() {
    Trie *tree = new Trie();
    tree->insert("apple");
    tree->insert("appex");
    tree->insert("almond");
    Cout<<"Does it exist" << tree->search("apple") <<endl;

    Cout<<"Does it exist" << tree->search("app") <<endl;

    return 0;
}
```

Delete:



Delete "apple"

① Search apple
- ↳ make the last e from 1 to 0
- ↳ अब 0 बन गया उसके बाद check करेंगे उसका child है या नहीं
  - ↳ If yes, we cannot delete.
  - ↳ If No, the delete last letter, and reverse and check in every step that child is present or not or root element is 0.

* Delete "Mon"

→ search M → found
search O → found
search n → found
→ check at n, it is 1.
↳ yes, then it means it is end of Word
↳ Make it 0.
↳ check any child is present or not for n.
↳ yes, return false.
↳ No Deletion performed.

Code :

```
bool isEmpty(TrieNode * node){
    for(int i=0; i<26; i++){
        if(node -> child[i])
            return false;
    }
    return true;
};

bool Delete(TrieNode * node, String Word, int depth){
        if(depth == word.size()){
        // It is not end of Word
            if(node -> isEndofWord == 0){
                return false;
            }
        // It is end of Word
        node -> isEndofWord = 0;
        // Child exist or not
        return isEmpty(node);
    }
```

```
        int index = Word[depth] - 'a';
    // char does not exist
        if (node -> child[index] == NULL)
            return false;

    // char exist
// recursive call for next element deletion in tree
        bool shouldDeleteChild = Delete(node->child[index], Word,
                                        depth + 1);

            if (shouldDeleteChild) {
                delete node -> child[index];
                node -> child[index] = NULL;

            return (node -> isEndOfWord && isEmpty(node));
        }

    return false;
};

Void DeleteWords (string Word) {
        Delete (root, Word, 0);
    };
};

In main:

    tree -> DeleteWords ("apple");
```

* Trie | (insert and search) || GFG

Code :

```
Void insert (struct TrieNode *root, string key) {

    TrieNode *node = root;
    for (char c : key) {
        int index = c - 'a';
        if (node -> children[index] == NULL) {
            node -> children[index] = new TrieNode();
            node = node -> children[index];
        }
        else {
            node = node -> children[index];
        }
    }
    node -> isLeaf = true;
}

bool search (struct TrieNode * root, string key) {

    TrieNode * node = root;
    for (char c : key) {
        int index = c - 'a';
        if (node -> children[index] == NULL)
            return 0;
        node = node -> children[index];
    }
    return node -> isLeaf;
}
```

phone directory

Given:

list of Contact : contact[]
↳ length = n.

Each Contact is in string
↳ Exist in phone directory and
a query string.

Task:

Implement a search query for phone directory.
↳ Run a search query for each prefix p
of query string s (from index 1 to |s|)
↳ prints all distinct Contact which
have same prefix as p in
lexicographical increasing order.

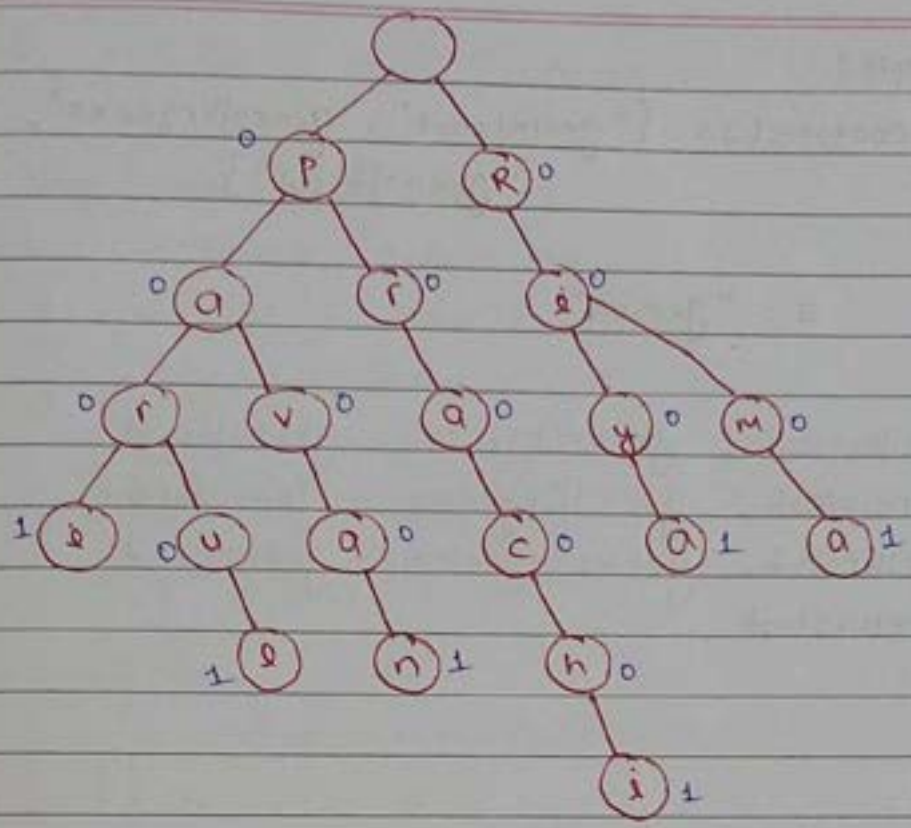
If there is no match b/w query and Contacts, print "0".

Explanation:

When we search Contacts in our phone.

→ Assume we type "R".
then it show all Contact which
start with R.
→ Then we type 'a'.
then it show Contact which
start with "Ra".

This is done with the help of Trie.



When we search 'p'

It show:

pari

parul

pavan

prachi

Now, "pa"

pari

parul

pavan

Now, "par"

pari

parul

Now, "pari"

Pari

Example:

contacts[] = {"geeikistest", "geeksforgeeks",
"geeksforcoa"}

s = "geeips"

g: geeikistest, geeksforgeeks, geeksforcoa
ge: geeikistest, geeksforgeeks, geeksforcoa
gee: geeikistest, geeksforgeeks, geeksforcoa
geei: geeikistest
geeip: 0
geeips: 0

Code:

```
class solution {
public:
class TrieNode {
  public:
    bool isEndofWord;
    TrieNode * child[26];

    TrieNode() {
        isEndofWor = 0;
        for (int i=0; i<26; i++) {
            child[i] = NULL;
        }
    }
};
class Trie {
    TrieNode * root;
    public:
    Trie() {
```

```
        root = new TrieNode();
    }
    Void insert (string word){
        TrieNode * node = root;
        for (char c : word){
            int index = c - 'a';
            if (node → child[index] == NULL){
                node → child[index] = new TrieNode();
                node = node → child[index];
            }
            else{
                node = node → child[index];
            }
    }
    node → isEndofWords = true;
}
Void findContact (string prefix, TrieNode * node, vector<string> &Contact)
{
    if (node → isEndofWord)
        Contact.push_back (prefix);

    for (char c = 'a'; c <= 'z'; c++){
        int index = c - 'a';
        if (node → child[index])
            findContact (prefix + c, node → child[index],
                contact);
    }
}
Vector<string> searchContact (string prefix){
    TrieNode * node = root;
    for (char c : prefix){
        int index = c - 'a';
```

```
            if(node->child[index] == NULL)
                return {"0"};
            node = node->child[index];
        }
        vector<string> Contact;
        findContact(prefix, node, Contact);
        return Contact;
    }
};

vector<vector<string>> displayContacts(int n, string Contact[],
                                       string s) {
    Trie tree;
    for(int i=0; i<n; i++)
        tree.insert(Contact[i]);

    vector<vector<string>> result;
    string prefix = "";

    for(int i=0; i<s.size(); i++){
        prefix += s[i];
        vector<string> Contact = tree.searchContact(prefix);

        result.push_back(Contact);
    }
    return result;
}
};
```

Lecture 80

Segment Tree

→ Range type

	0	1	2	3	4	5	6
Array :	3,	6,	4,	25,	5,	13,	8

Sum of index (2-5) → 4+25+5+13 = 47

Range sum (2-5)

Normal Approach

Time Complexity : O(N)

→ But, Segment tree can solve this problem in O(log N)

→ From Normal approach find sum of q queries it take O(n * q) time.

→ For creating a segment tree it take : O(N)

↳ For find sum of 1 query : O(log n)

For q query : O(q · log N)

Total time Complexity : O(N + q · log N)

→ one more approach :

Make an sum array → O(N)

Finding sum → O(1)

↳ For q query → O(q)

Total time Complexity : O(N + q)

→ But, if we want to find the maximum of an array then, Normal Approach can take O(N) time.

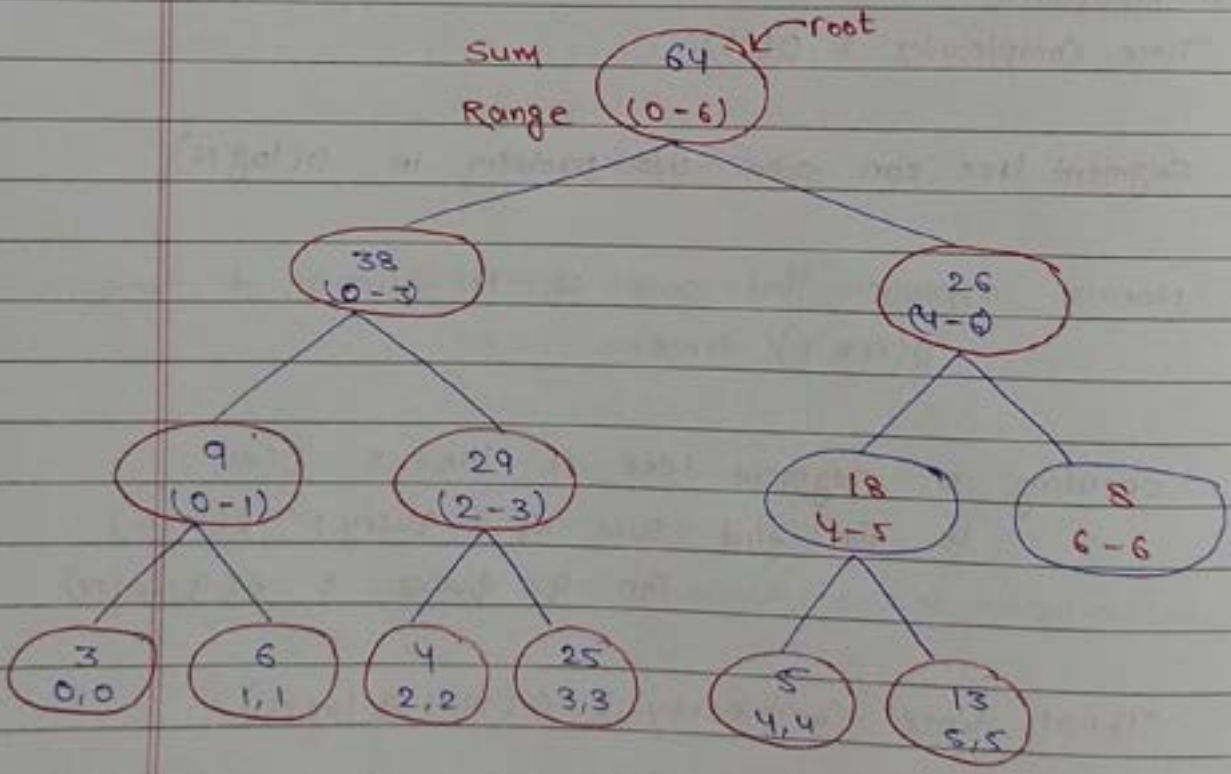
Segment tree can find in O(logN) time.

So, We use segment tree.

→ In sum array, We don't use segment tree.

→ Sum - ronge query, update

{3, 6, 4, 25, 5, 13, 8}
0 1 2 3 4 5 6



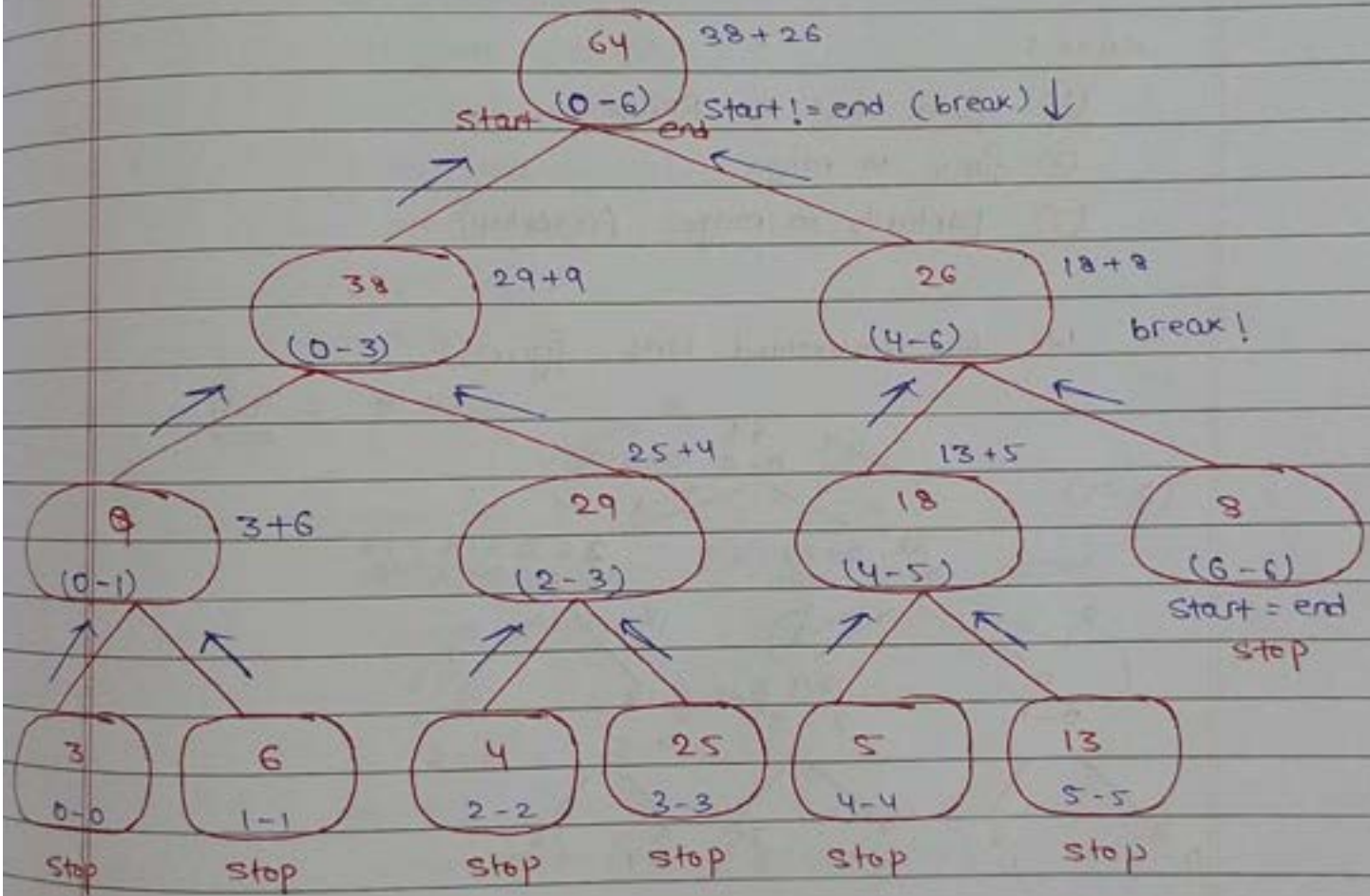
*Tree creation:

$n = 7$

Total Node: $\{2n-1\}$

We can create the tree from down.
(Bottom to up)

→ First Break the tree.



Now, We can fill the sum from bottom.

Now, till Now We create the Sum.

How We find the range sum.

* Sum of Range (2, 5)

(2, 5)
left ↵ ↳ right

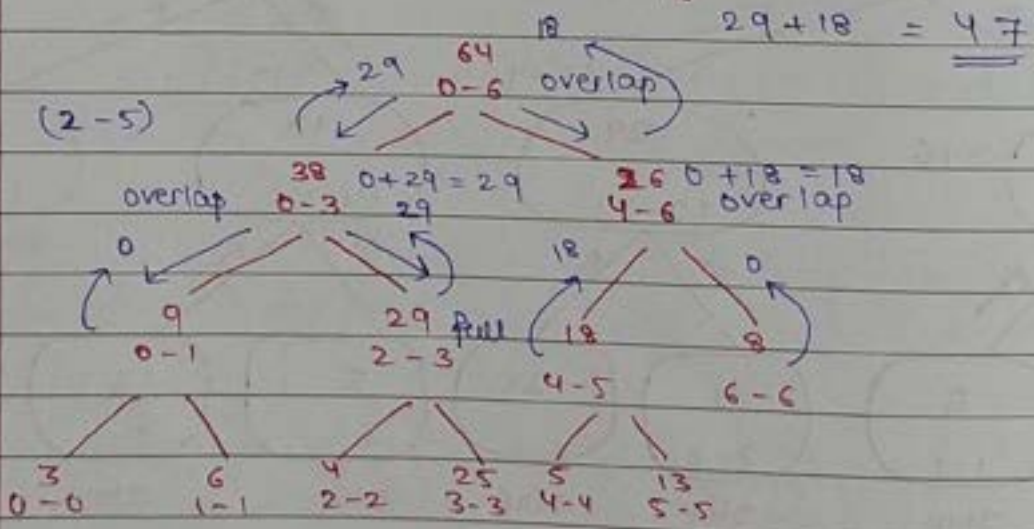
start end
0 6

left right
2 5

case :

1. fully out of Range
2. fully in range
3. partially in range. (overlap)

↳ We will understand with figure :



code :

```
#include <iostream>
#include <vector>

using namespace std;
int main() {
        Vector <int> arr;
        arr.push_back(3);
        arr.push_back(6);
        arr.push_back(4);
        arr.push_back(25);
        arr.push_back(5);
        arr.push_back(13);
        arr.push_back(8);
        return 0;
    }

class segmentTree {
        vector<int> tree;
        vector <int> arr;
        int n;

        public:
        segmentTree (vector <int> input) {
                n = input.size();
                arr = input;
                tree.resize(4*n);
                build(0, 0, n-1);
                                // node index, start, end;
        }
```

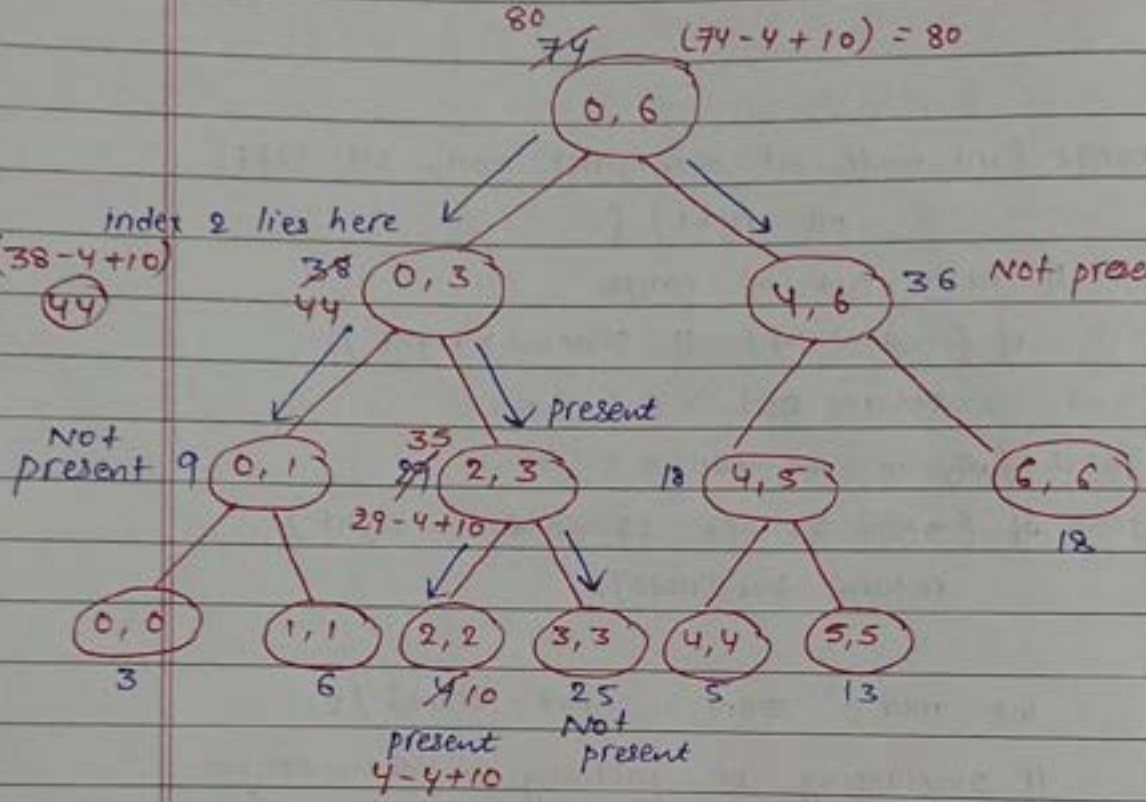
~~in main:~~

```
void build (int node, int start, int end) {

        if (start == end) {
            tree[node] = arr[start];
            return;
        };

        int mid = start + (end - start)/2;
        build (2 * node + 1, start, mid);
        build (2 * node + 2, mid+1, end);
        tree[node] = tree[2*node+1] + tree[2*node+2];
    }
};

int main() {
    vector <int> arr;
    arr.push_back(3);
    arr.push_back(6);
    arr.push_back(4);
    arr.push_back(25);
    arr.push_back(5);
    arr.push_back(10);
    arr.push_back(8);

    SegmentTree *Tree = new SegmentTree(arr);
        cout << Tree->query(2,5) <<endl;
        Tree->update(2,10);
        cout << Tree->query(2,5) <<endl;

    return 0;
}
```

```
int query (int left, int right) {
    return range (0, 0, n-1, left, right);
    // Root node index (seg), start, end, left right;
}

int range (int node, int start, int end, int left,
                int right) {
    // fully out of range
    if (end < left || start > right)
        return 0;
    // fully in the range
    if (start >= left && end <= right)
        return tree [node];

    int mid = start + (end - start) / 2;
    // overlapping or partially in the range
    // left child + right child

return range (2 * node + 1, start, mid, left, right) +
        range ( 2* node + 2 , mid + 1, end, left, right);
}
```

Updation:

{ 3, 6, 4, 25, 5, 13, 18 }
  0 1 2 3 4 5 6



⇒ update index 2

↳ present : 4

change it to 10

* Code :

```
Void update (int idx, int val) {
    return updating (0, 0, n-1, idx, val);
        // node, start, end, index, value
}

Void updating (int node, int start, int end, int idx, int val)
{
    // Base Condition
    if((idx > end) || (idex < start))
        return;
```

```
if (start == end) {
    tree[node] = val;
    return;
}

tree[node] += val - arr[idx];
int mid = start + (end - start) / 2;

// left child
updating (2 * node + 1, start, mid, idx, val);
// right child
updating (2 * node + 2, mid + 1, end, idx, value);

}
```

Real life Application of segment tree:

Stock Market
↳ used to predict the price of stock on specific day.

Range = { Max, Min, sum, product }

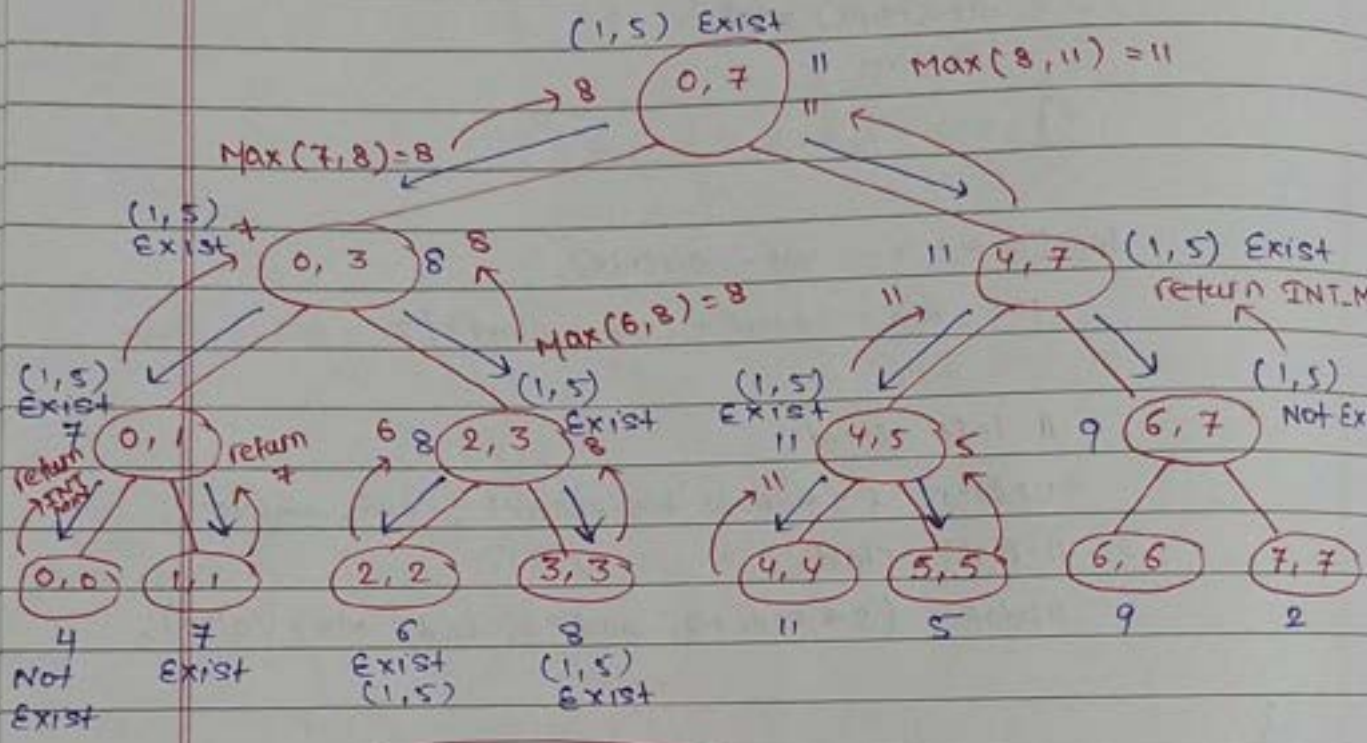
	0	1	2	3	4	5	6	7
(Max)	4,	7,	6,	8,	11,	5,	6,	2

We have to find the Maximum in specific Range.

We have to make a tree and solve this.
Let understand with an Example.

classmate
Date
Page

0	1	2	3	4	5	6	7
4	7	6	8	11	5	9	2



Max (1,5) : 11

```
Void build (int node, int start, int end) {

    if (start == end) {
        tree[node] = arr[start];
        return;
    }

    int mid = start + (end - start) / 2;
    build (2 * node + 1, start, mid);
    build (2 * node + 2, mid + 1, end);

    tree[node] = Max(tree[2 * node + 1], tree[2 * node + 2]);

}
```

```
int range (int node, int start, int end, int left,
                    int right) {
    // fully out of range
    if (end < left || start > right)
            return INT_MIN;
    // fully in the range
    if (start >= left && end <= right)
            return tree[node];

    int mid = start + (end - start) / 2;
    // overlopping or partially in the range
    // Max (left child, right child)

return max (range (2 * node + 1, start, mid, left, right)
        + range (2 * node + 2, mid + 1, end, left,
                right));

}

void updating (int node, int start, int end, int idx, int value) {
        // Base Condition
        if (idx > end || idx < start)
                return;

        if (start == end) {
                tree[node] = val;
                return;
        }

        int mid = start + (end - start) / 2;
        // left child
        updating (2 * node + 1 , start, mid, idx, val);
```

```
        // right child
        updating (2*node + 2, mid+1, end, idx, val);

    tree[node] = max(tree[2*node+1], tree[2*node+2]);

}

Void update(int idx, int val){

    return updating(0, 0, n-1, idx, val);
        arr[idx] = val;

}
```

Real life Example :

→ Stock Marketing

→ Networking

→ Weather Forecasting