Lect1 Programming Fundamentals

↳ Thought process to solve a problem

1. Understand the problem
2. input values
3. approach. (find appropriate approach)

```
* min_element(v.begin(), v.end());
* max_element(v.begin(), v.end());
accumulate(v.begin(), v.end());
count(v.begin(), v.end(), 2);
find(v.begin(), v.end(), 3);
reverse(v.begin(), v.end());
```

(begin) map sets

Problem → Solution head → Rough solution / Flowchart / Pseudocode → High level Language (Source code) → [compiler] → Machine Understandable Language

user friendly Language

↳ Using Computer to solve a problem

n = 13
↳ prime or not

13/2 → rem 1
13/3 → rem 1
13/4 → rem 1
13/5 →
13/12 →

Computer remainder 0 नहीं आता तो 13 Prime no है।

Manual

Computer only understands 0/1 machine code which is converted by compiler from source code

Flowcharts

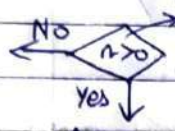
Representation of algorithm into graphical form is known as flowcharts

↳ Components

1. (start/end) Terminator

2. /input/output/ Input/output

3. [a = b+c] Process block

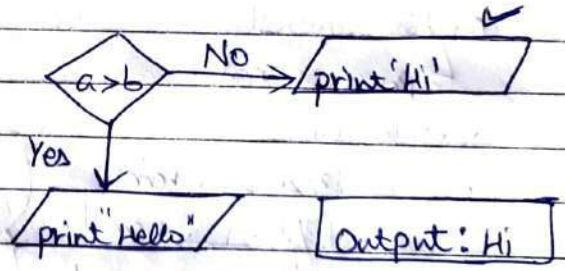
4. 

Decision making block

5. ↓↑ flows

6. (A) Connector

a = 3
b = 5



↳ Pseudo Code [Can have multiple pseudo codes to same question]

ढोंगी code

I can understand the code
friend can understand the code
But computer cannot understand the code.

eg: Start Program
Enter two numbers A, B
Add the numbers together
Print Sum
End Program

ex Give me difference of two numbers

ans Start program
Take two numbers as input from the user
Subtract it and store it in a variable
Print the value stored in the variable
End Program

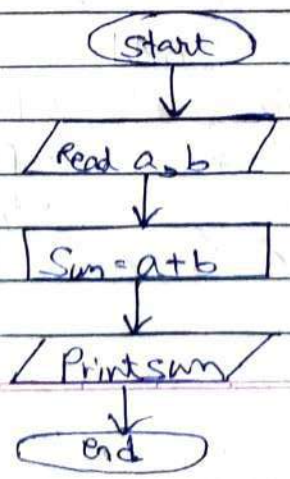
ex Print the multipication of two nos

ans Start program
Take two numbers as input from the user
~~Subtra~~ Multiply it and store it in a variable
Print the value stored in the variable
End the program

ex Print the average of two numbers

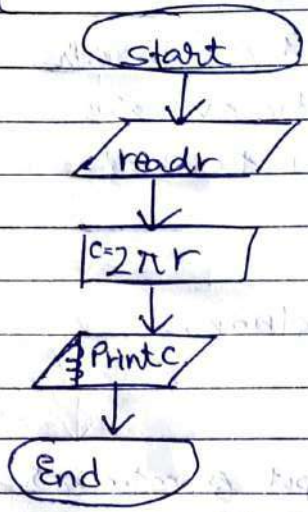
ans. Start program
Take two numbers as input from the user
add the numbers and devide the sum by 2 and store the result in a variable
Print the value stored in the variable.
End the program

ex Add 2 numbers by taking input. [Flowchart]



ex Circumference of circle

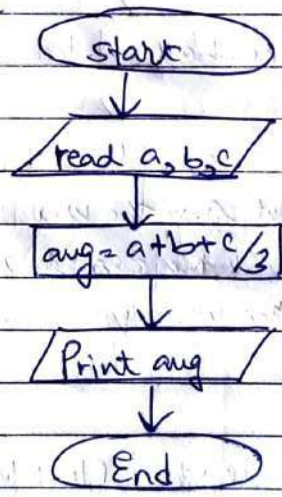
Flowchart



Pseudo code
- start Program
- Take the value of r
- Process it as 2πr
- Show the circumference
- End Program

ex Average of 3 numbers

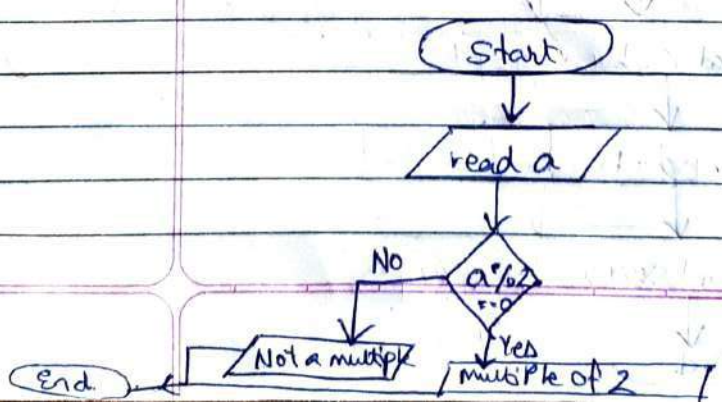
Flowchart



Pseudo code

Start Program
Take three nos. as input
Add 3 nos. and devide the result by 3
Print the average
End Program

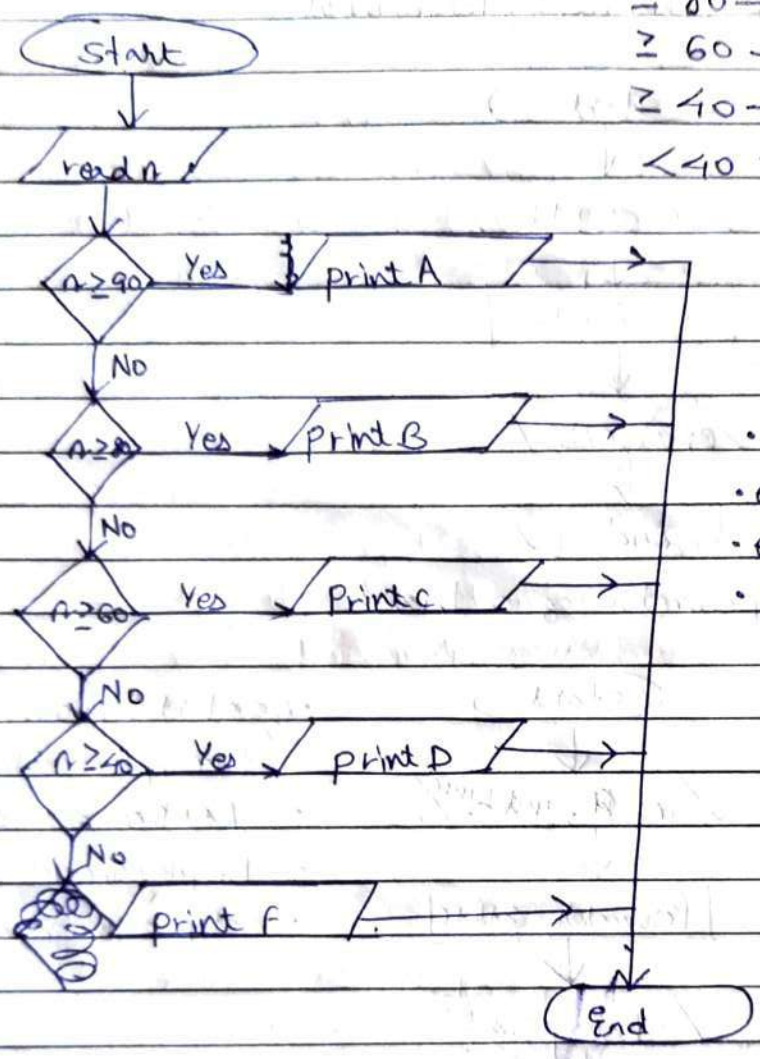
ex Check a no. is even or odd.



Pseudo code
- Read n
- If n%2 = 0 then print even else print odd

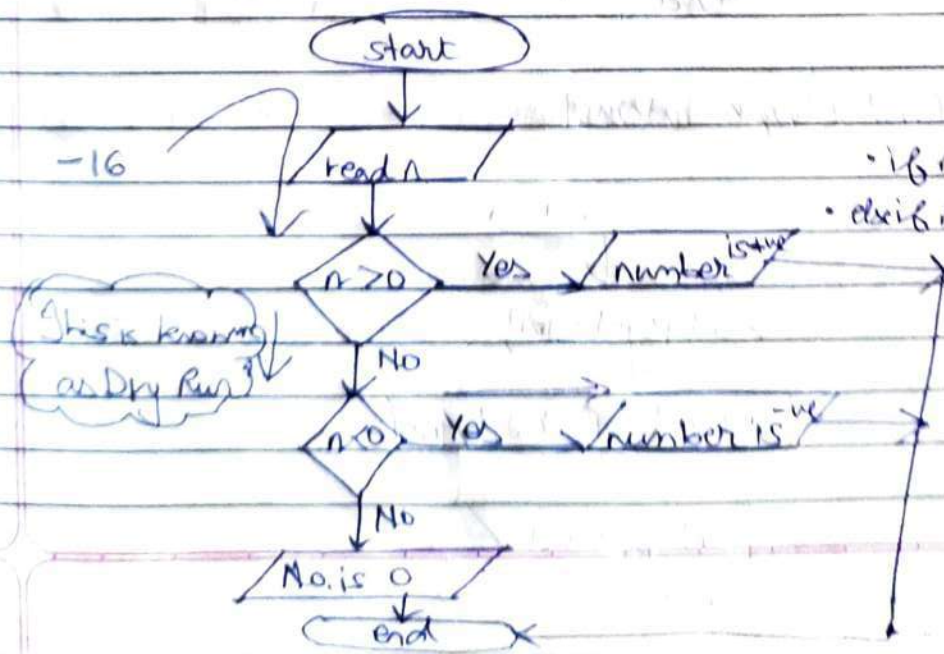
ex Student and Grade Flowchart

Marks	Grade
≥ 90	→ A
≥ 80	→ B
≥ 60	→ C
≥ 40	→ D
< 40	→ F



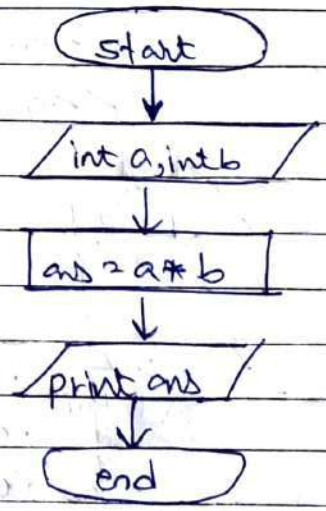
Pseudo code
- read m
- if m >= 90 then print A
- else if m >= 80 then print B
- else if m >= 60 then print C
- else if m >= 40 then print D
- else print F

ex Check whether a number is +ve, -ve or 0.

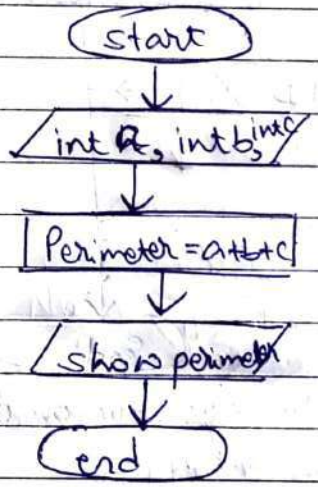


Pseudocode
- read n
- if n > 0 the +ve no
- else if n < 0 then -ve no
- else no. is 0

ex ~~Print counting from 1 to N~~
Multiply two no. by taking input.

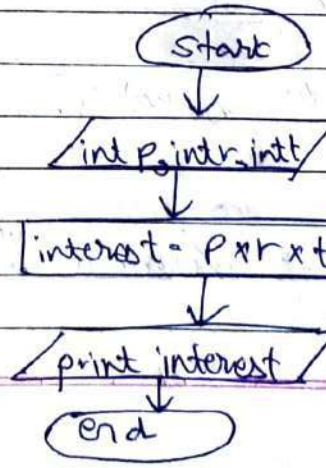


ex Find perimeter of a triangle

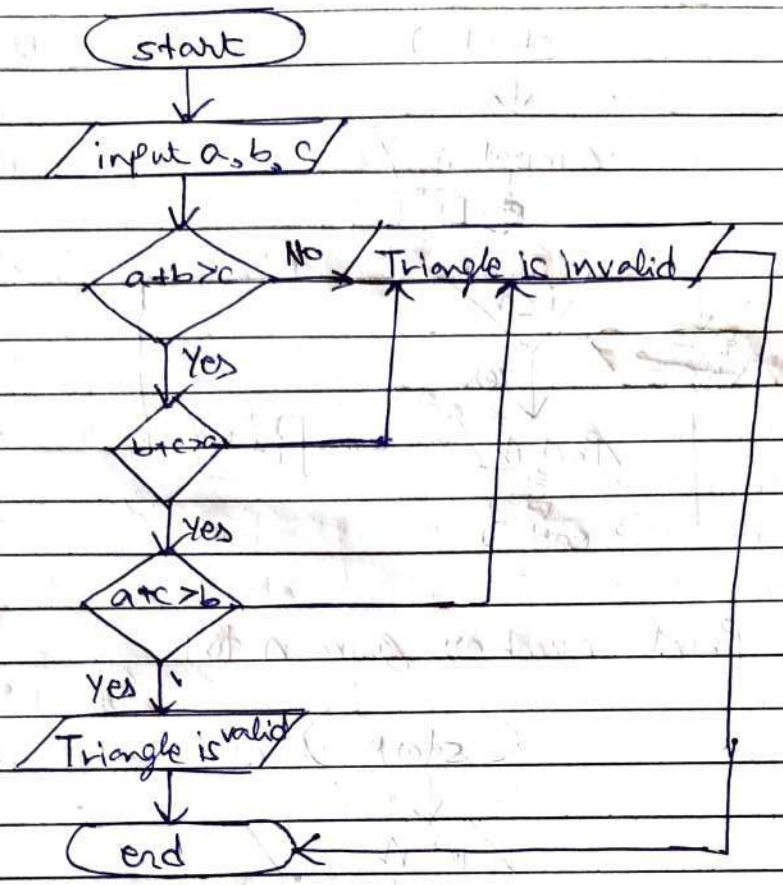


- start a program
- take input of three sides from user
- Add them and store it in a variable
- Display the value of variable
- stop a program

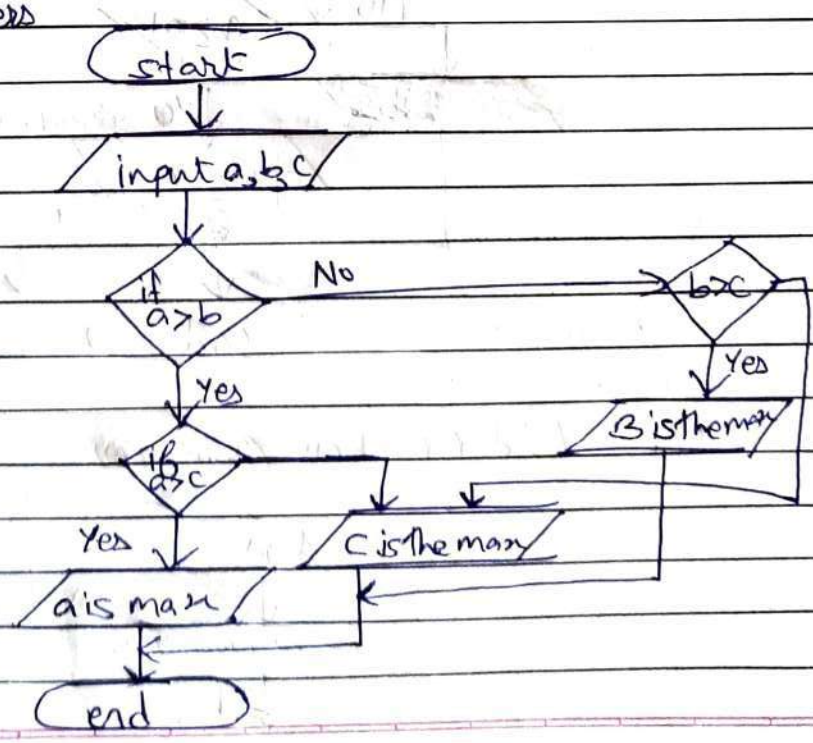
ex Find simple interest



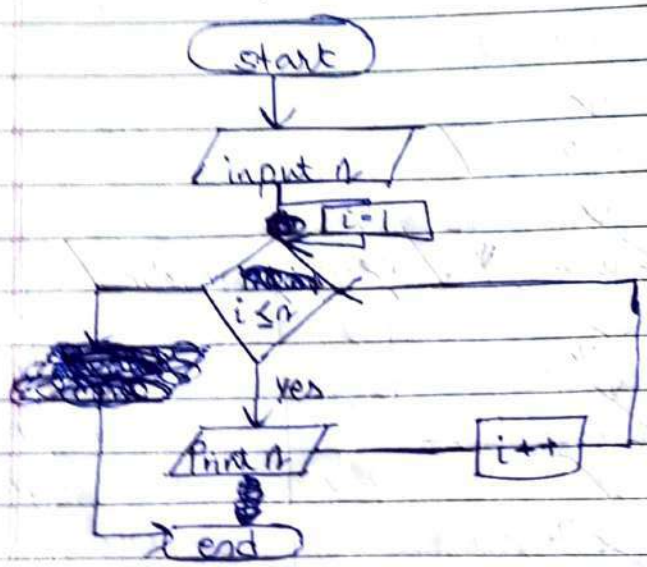
ex Valid triangle or not.



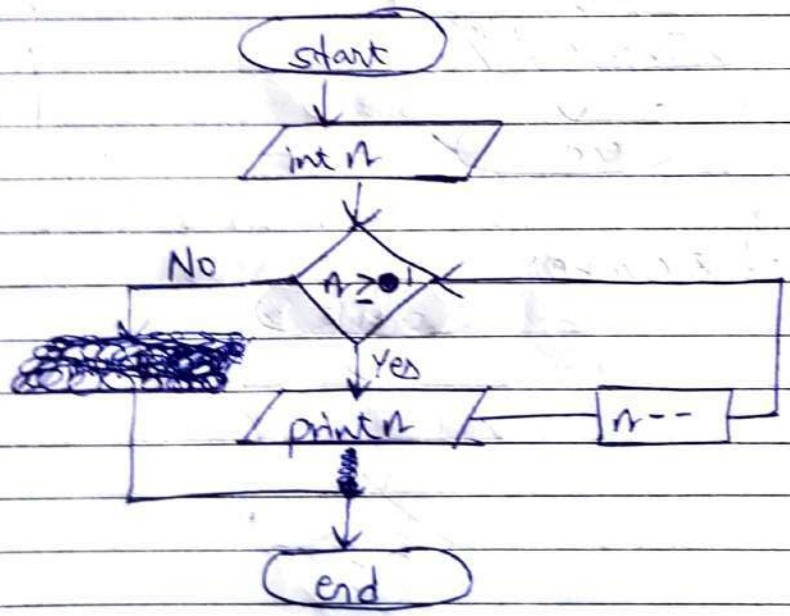
ex Max of 3 numbers



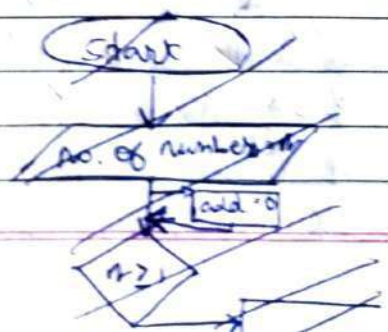
ex Print counting from 1 to n

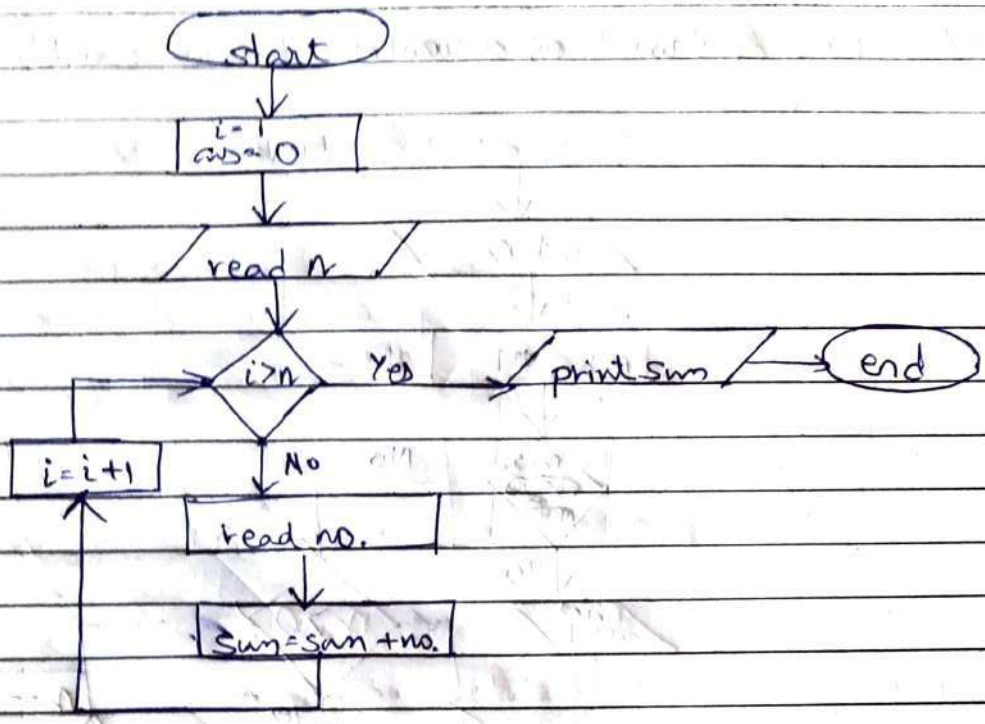


ex Print numbers from n to 1

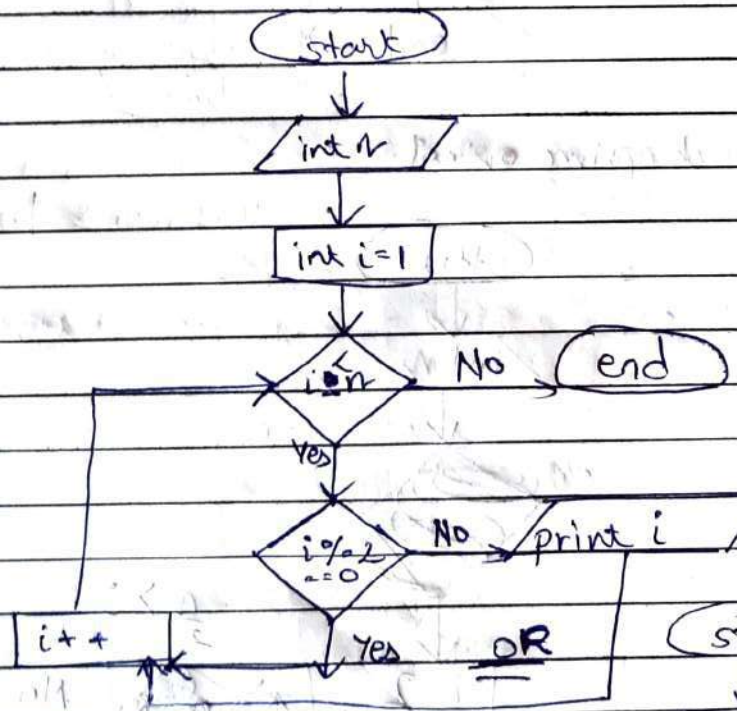


ex Add n numbers from user

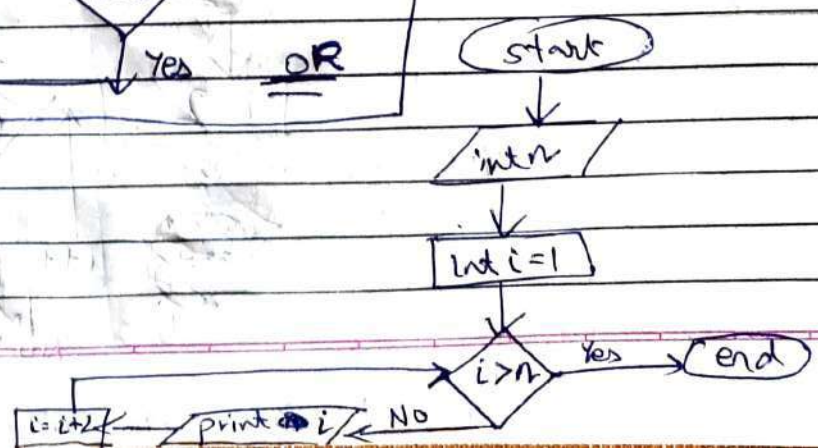




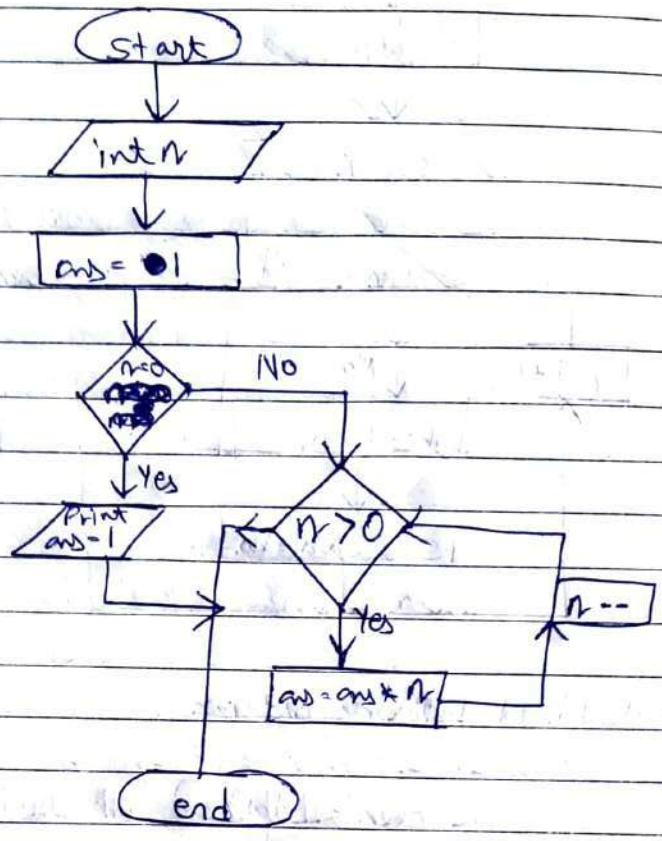
ex Print 1 to N but only odd nos



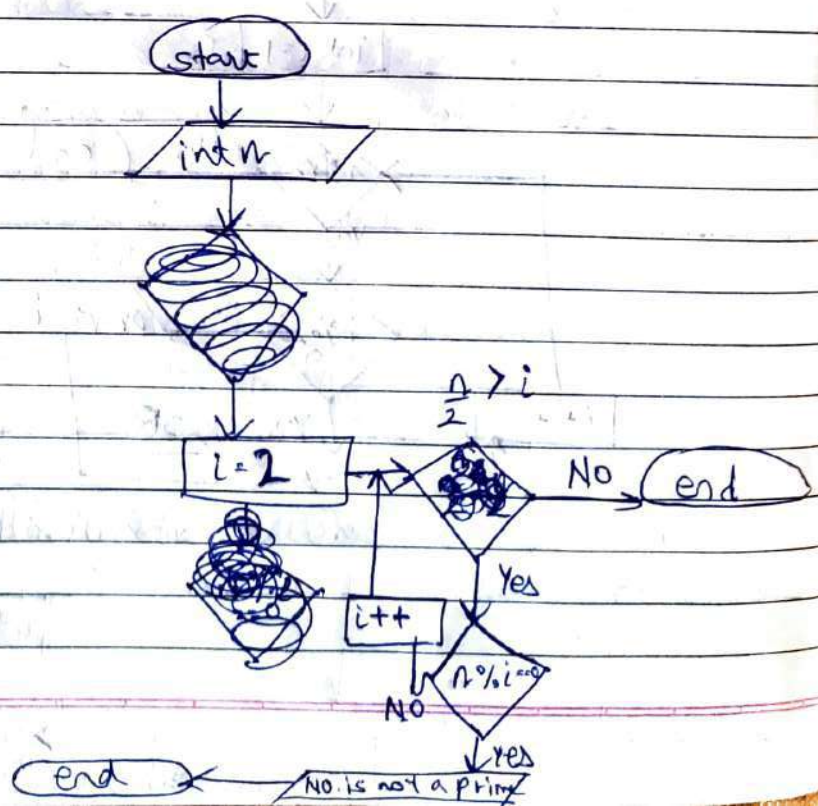
OR



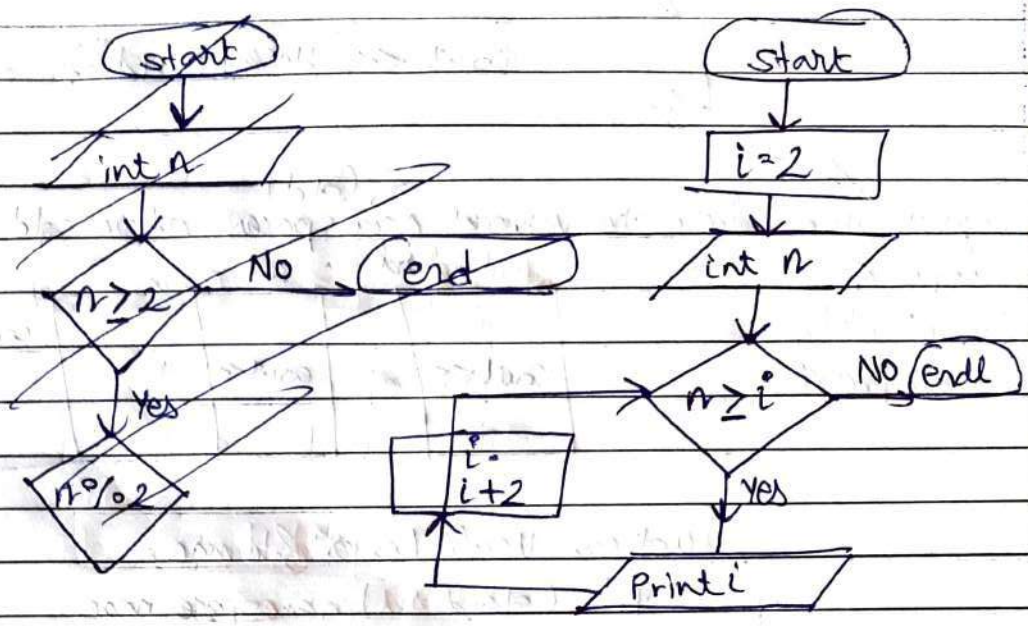
ex Find factorial of a number



ex Check prime or not.

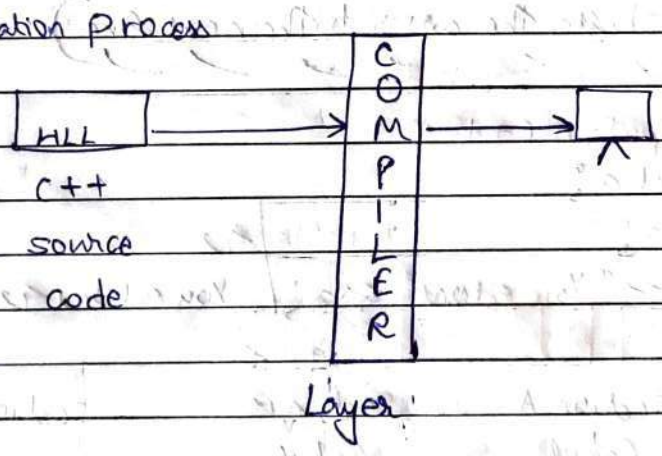


ex Print 1 to n but even numbers



Lect 2 Creating your First Program

Compilation Process



Converts
HLL to machine
understandable code (0/1)

Compiler
↓
At one go

Interpretor [line by line]

↳ Repl.it

Start

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World";
}
```

regions where scope of different identifiers are present

There are various namespaces (std) where std is defined

iostream vaali header file mae
std (standard) namespace mae

file where the implementation of i/o stream is present

endl shifts the cursor to the next line

↳ Taking input in C++

```
int a;
cin>> a;
cout << "You entered" << a;
```

<42
You entered 42

Section A
Rahul

Section B
Rahul

Section A::Rahul;

C++ Datatypes

- Built in / Primitive
  - int
    - int
    - char
  - floating
    - float
    - double
  - void
- Derived
  - arrays
  - pointers
  - references
- User defined
  - structures
  - unions
  - classes
  - enumerations

variable

int sum = 12;

datatype value

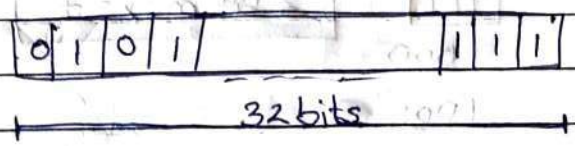
return

int main()
{
}

return

void main()
{
}

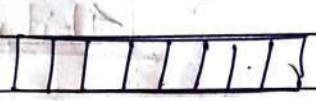
int → 4 byte → 32 bits



2^{32} combinations

charactor → 1 byte → 8 bits

ch = 'a';



$2^8 = 256$ combinations

Char is mapped to → ASCII value

bool flag = true;

cout << " value of flag is " << flag << endl;

Value of flag is 1

True ⟷ 1

False ⟷ 0

bool takes 1 byte → 8 bits

1 bit cannot be addressed

so we are using 1 byte so that we can address to next byte.

Here memory wastage takes place

float f = 1.2; kam space leta hai. 4 byte 32 bits

double d = 1.2; takes 8 bytes 64 bits can show larger numbers

short → 2 bytes 16 bits

max number

$2^{16} - 1$

जो ज्यादा memory Lose रहा है तो ज्यादा बड़ा Number store कर सकता है।

1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

2 bit max = 3

3 bit max = 7

1111

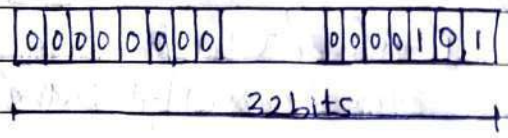
4 bit max = 15

long long → 8 bytes

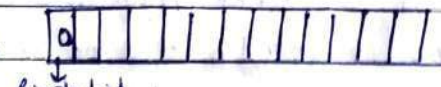
char = $2^8 - 1$ = 255

↳ How data is stored
positive v/s negative

int a=5 4bytes → 32 bits

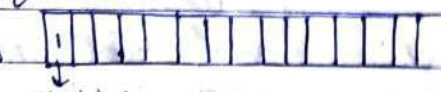


+ve numbers



first bit

-ve numbers



first bit

How -ve nos. are stored in memory
↳ 2's complement

7's complement
111

00101011
1's comp → 11010100
11010101 ← 2's complement = 1's complement + 1

7 00000111
1's comp 11111000
2's comp 11111001

4 How to represent int −5

1. ignor −ve sign 5
2. Binary equivalent 101 0000 0101
3. 2's complement 1111 1011 [−5]

0 → 2's complement → 0000 0101 (2's complement)
[−5]

4 Interesting problem

22	31	43	7
1 byte	1 byte	1 byte	1 byte

int — 1 int
char — 4 char

int a = 5
↓
4 byte, so take 4 bytes consecutively

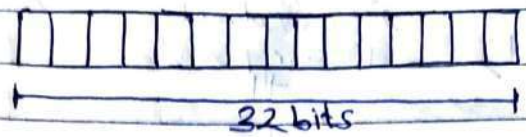
char b = 4
↓
1 byte
so consider only 1 byte

A Datatype usually provides 2 things
i. Which type of data we are storing
ii. How much space we are taking in the memory

int a = 5 [signed] ~~+ve~~ −ve
unsigned +ve nos.

By default numbers are stored in signed variables

int → 4 bytes → 32 bits



Total combinations = $\boxed{2^{32}}$

unsigned 0 to $2^{32}-1$
signed 2^{31} to $2^{31}-1$

↳ short 2 bytes 16 bit

Total combinations $\boxed{2^{16}}$

unsigned : 0 to $2^{16}-1$
signed : -2^{15} to $2^{15}-1$

↳ char 1 byte 8 bits
signed : -2^{7} to $2^{7}-1$
unsigned : 0 to $2^{8}-1$

↳ n bits
signed : $[-2^{n-1} \rightarrow 2^{n-1}-1]$
unsigned : $[0$ to $2^{n}-1]$

↳ xyz 6 byte → 48 bits

unsigned 0 to $2^{48}-1$
signed -2^{47} to $2^{47}-1$

↳ Type Casting or Type conversion
Conversion of one datatype to another in a program.
It can be done in two ways : automatic by compiler
manually by programmer or user.

char int

char ch = 97;
cout << ch; a

int a = 'b';
cout << a; 98

int got converted to char

Automatic handle implicit type conversion

float a = (float) 2; [Explicit type conversion]

double d = 5.7;
int x = (int) d + 2; 5 + 2 = 7
Explicit type conversion

Operators
- Arithmatic
- Relational
- Assignment
- Logical
- Bitwise
  - &
  - |
  - ^ XOR same 0 diff 1
  - ~ not

char ch = 234234; [ch max 255]

[- - - - - - - -]
1 byte 1 byte
[00101000]

ch = 00101000
decimal equivalent: mapped to that character

eg. ch → 1024; → 10 bits
1 byte → 8 bits
0000000 0000100 00000000
ch · 00000000
character

int / int / int

int + int = int
int - int = int
int * int = int

cout << (5.0)/3; 1.66
cout << 5/3.0; 1.66
cout << 5.0/3.0; 1.66
cout << 5/3; 1

cout << ((double) 5/(3); // 1.66
cout << ((float) 5) /(3); // 1.66

(cond1 && cond2 && cond3)
↓ false → ✗

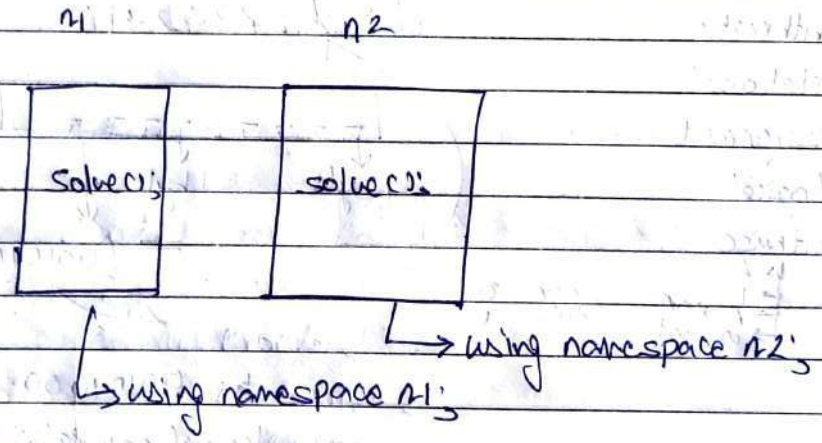
||y

(cond1 || cond2 || cond3)
↓ true → ✗

↳ Namespace

using namespace std;

~~solv~~



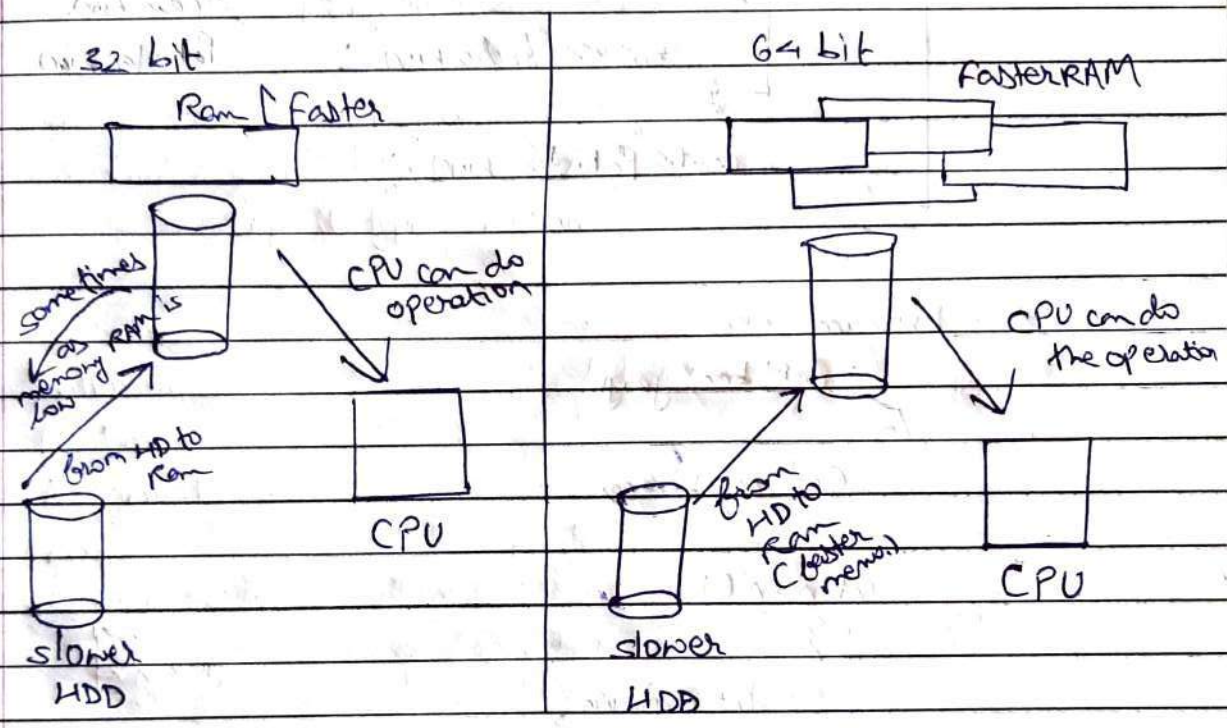
↳

Compiler	Interpreter
• A compiler takes the entire program in one go.	An interpreter takes a single line of code at a time.
• The compiler generates an ~~integ~~ intermediate machine code.	The interpreter never produces an intermediate machine code.
• Compiler is best suited for the production environment. eg. C, C++, C#, java	An interpreter is best suited for a software development environment eg. Python, PHP, Perl etc

↳ 32 bit v/s 64 bit.

32 bit 2^{32} = 4 Gigabytes

64 bit 2^{64} bytes of memory 16 Exabytes



So 64 bit system is faster than 32 bit system

Lecture 3 Lets solve some patterns

if condition

```
if (condition)
{
    You are elegible to vote
}
```

else block dalo to bhi sahi
na dalo to bhi sahi.

int i = (1,2,3); [Last element]
cout<<i; → 3

ex

```
int main()
{
    int score;
    cout << "Enter the score" << endl;
    cin >> score;
    if (score < 300)
    {
        cout << "India wins";
    }
    cout << "Pakistan wins";
}
```

200

India wins
Pakistan wins

ex bronum == 0
↳ Baat ban jaigi.
Ya
Baat nahi banegi

```
int main()
{
    int Bronum;
    Cin >> Bronum;
    if (Bronum == 0)
    cout << "Baat Ban jaigi";
    else {
        cout << "Baat nahi banegi";
    }
}
```

↳ Loops

for loop

```
for (int i = 0; i < 5; i = i + 1)
{
}
```

initialise condition updation

The default value of unsigned variable is undefined
Void occupy 0 bytes
Total types of errors: 3 run time, compile time, system
3/0 is an exception

```
for ( ____ ; ____ ; ____ )
  {
    init      cond^n      updation
    }
```

init → not mandatory
cond^n → not mandatory
updation → not mandatory

not ex

```
int n;
if( cin>>n)
  {
    cout<< "Babbar"} + n;
  }
```

```
int n;
if ( cout << "Babbar")
  {
    cout<<"Love";
  }
```

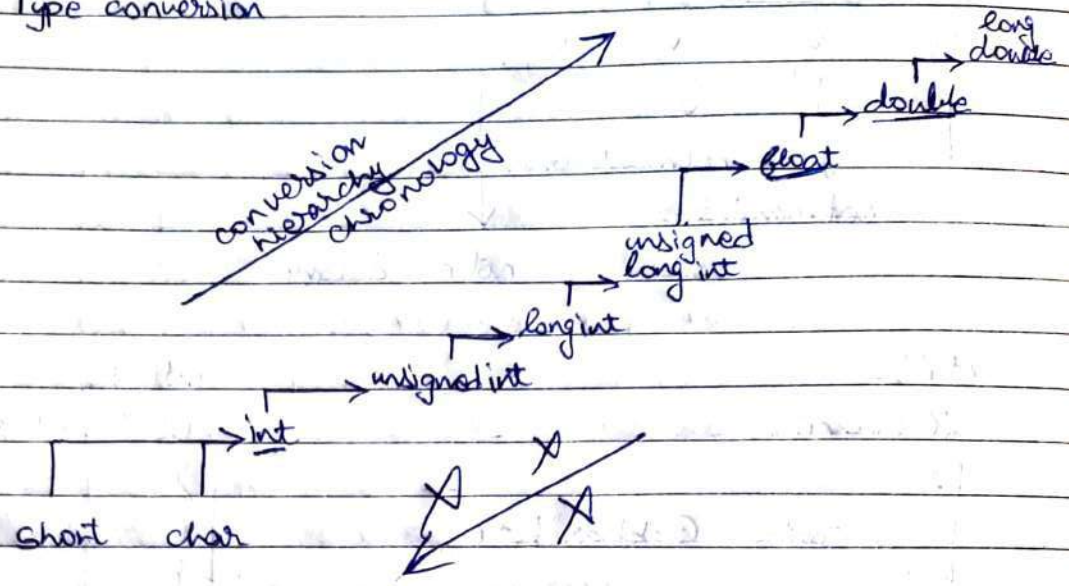
Both will work

↳ Patterns
why
↳ to build logic building
↳ to have a grip on loops

Code
↳ nested loop
- 2 loops
  - Outter loop [Rows]
  - Inner loop [Columns]
- Multiple loops

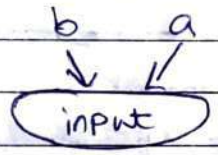
Float and double removes unnecessary 0 from the number.

Type conversion

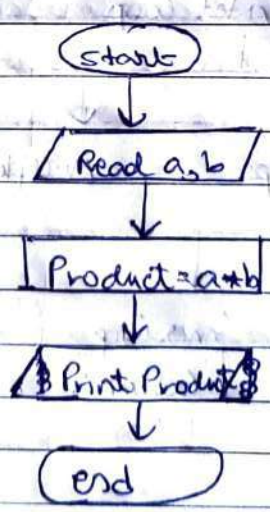


Weekl Assignmat

1.



i. Take input of two numbers from the user (a, b).
ii. Multiply the numbers and store it in a variable named (c).
iii. Display the result of c to the user.



2.

Lecture 4.

```
_ _ _ _ _ *
_ _ _ _ * * _
_ _ _ * _ * _ *
_ _ * _ * _ * _ *
_ * _ * _ * _ * _
* _ * _ * _ * _ * _ *
```

n = total number of rows

```
for(int i=0; i<n; i=i++)
{
    // spaces
    for(int j=0; j<n-i-1; j++)
    {
        cout<<" ";
    }

    //
    for(int k=0; k<i+1; k++)
    {
        cout<<"*";
    }
    cout<<"\n";
}
```

↳ inverted pyramid

* _ * _ * _ * _ *	0 spaces	5 stars
_ * _ * _ * _ *	1 space	4 stars
_ _ * _ * _ *	2 space	3 stars
_ _ _ * _ *	3 space	2 stars
_ _ _ _ *	4 space	1 star

↳ solid pyramid

```
    *
  * _ *
* _ * _ *
* _ * _ * _ *
  * _ * _ *
    * _ * _
      * _
```

b) hollow diamond

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

```
for (int row=0; row<n; row=row+1)
{
    //spaces
    for (int col=0; col<n-row-1; col=col+1)
    {
        cout<<" ";
    }

    //stars
    for (int col=0; col<2*row+1; col=col+1)
    {
        if (col==0)
        {
            cout<<"*"; // first star
        }
        else if (col==2*row)
        {
            cout<<"*"; // last star
        }
        else
        {
            cout<<" ";
        }
    }
    cout<<endl;
}
```

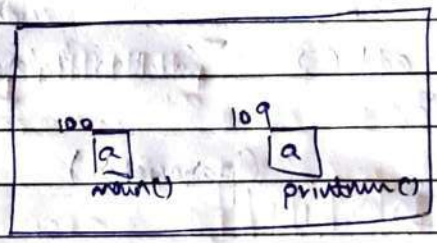
Lecture functions and some problem statements

```
main()
{
    int a=5
    print(a);
}
```

```
void printnum(int a)
{

}
```

100 [5] a

109 [5] a



copy create होती है।

Pass by value [copy create करता]

```
int main()
{ int num=12;
  num++;
  ++num;
  print(num);
  --num;
  cout << num; (13)
}
```

```
void print(int num)
{
  ++num;
  num++;
  cout << num; (16)
  --num;
}
```

num 12 [12] 13 14

num 13 14 [14] 15 16

Copy create होगी actual value Pass नहीं होगी।

add function int main kae upar likha ही होता है।

~~int main {~~

```
int add (int a, int b);

int main()
{
}
```

(argument)

```
int add ()
{
}
```

function declaration.

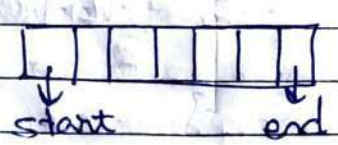
(parameter)

↳ Why do we need size along with array to function.

int arr[30] = {1, 2, 3, 5}

It tells you to create memory for 30 integer type values, but the actual total no. of values in an array are 5. The size of an array contains 5. So explicitely we need to pass the size of any array i.e. how many values it contains)

ex Extreme printing in an array




```
if (start == end)
{
    cout << arr[start];
}
else {
    cout << arr[start];
    cout << arr[end];
}
```

ex Reversing an array

```
start = 0
end = n-1
swap(arr[start], arr[end]);
start++;
end--;
```

↳ Vector to find unique elements

1 2 2 3 4 5 4 5 1

```
int unique=0
for (int i=0; i< arr.size(); i++)
{
    unique = unique ^ arr[i];
}
```

↳ union of two arrays

```
for (int i=0; i<arr.size(); i++)
{
    unionvalue.pushback(arr[i]);
}

for (int i=0; i<brr.size(); i++)
{
    unionvalue.pushback(brr[i]);
}
```

↳ intersection of two arrays

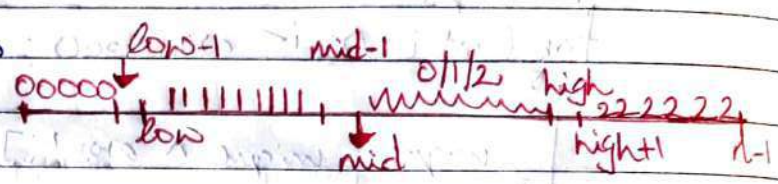
```
for (int i=0; i<arr.size(); i++){
    for (int j=0; j< brr.size(); j++){
        if (arr[i] == brr[j] { intersect.pushback(arr[i]);
    }
}
```

```
findpairsum (vector<int> arr, int sum){
    for (int i=0; i<arr.size(); i++){
        for (int j=i+1; j<arr.size(); j++){
            if(arr[i] + arr[j] == sum)
            cout<<"pair found" << arr[i] << and << arr[j]
        }
    }
}
```

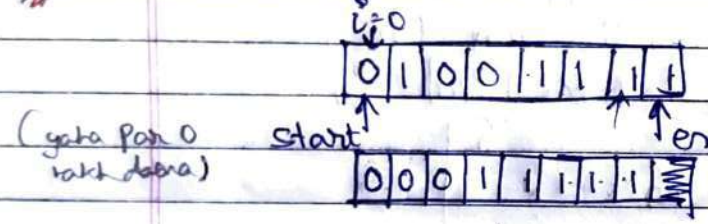
Dutch National flag algo.

IMP

for triplet also



sort 0's and 1's




```
start=0;
end = arr.size()-1
while (i != end) or start <= end int i=0;
{
    if (arr[i]==0) {
        swap (arr[i], arr[start]);
        start++; i++;
    }
    if (arr[i]==1) {
        swap (arr[end], arr[i]); end--;
    }
}
```

end ki ek naya number aaya tha usko pro con karna hoga uski sahi jagaha pahuchane kae liyae.

```
for (auto i : arr)
{
}
```

In function array we need always to give the bound of the 2nd arr[][2]

Arrays class 3 12 Feb 2023

	0	1	2
0	5	7	9
1	12	4	2
2	1	6	7

5	7	9	12	4	2	1	6	7
0	1	2	3	4	5	6	7	8

$= 3 \times 2 + 1$
$= (7)$

index = $C * i + j$

$3 * 1 + 2 = 5$

from 1D to 2D
row $n/3$
column $n\%3$

↳ row sum

```
arr = - - -
~~sum = 0~~
for (int i=0; i<row; i++)
{
    sum=0;
    for (int j=0; j< column; j++)
    {
        sum = sum + arr[i][j]
    }
    cout << sum << endl;
}
```

1	2	3	6
4	5	6	15
7	8	9	24

↳ searching a element in 2D array [linear search]

5	6	8
7	2	4
1	6	9

target = 12

```
for (int i=0; i<row; i++)
{
    for (int j=0; j<col; j++)
    {
        if (target == arr[i][j])
        { cout << i << j; }
    }
}
```

↳ Max number in an array;

```
int max = INT_MIN;

for( int i=0; i< row; i++)
{
    for (int j=0; j<column; j++)
    {
        if(arr[i][j] > max )
            max = arr[i][j]
    }
}
```

↳ minimum number found

```
int min = INT_MAX;
for (int i=0; i<row; i++)
{
    for (int j=0; j<col; j++)
    {
        if( min > arr[i][j])
            min = arr[i][j];
    }
}
```

↳ transpose of 2D array

```
for (int i=0; i<row; i++)
{
    for (int j=0; j< column; j++)
    {
        temp = arr[i][j]
        arr[i][j] = arr[j][i]
        arr[j][i] = temp;
    }
}
```

Why we need to pass the arr[ ][ ] second parameter during the defination of function.

```
int function(int arr[][30])
{
}
```

↳ We need to pass the 2nd parameter of an 2D array during defining the function bcz as we initialize the 2D array but actually on 2D array is stored as 1D contingous memory location. To find at what memory on 2D array is stored we need colums i.e. c*i+j

↓

formula of finding at what memory it is stored where c = no. of colums i = i^{th} row & j = j^{th} rows

↳ Vectors

```
for (int i=0; i< arr.size(); i++)
{
    for (int j=0; j<arr[i].size(); j++){
        cout << arr[i][j];
    }
}
```

Lect 10 Searching

Linear search: TC is O(n) for 1000 elements need 1000 comparisions which is avoided to use.

Sort function TC-(n log n) — Heapsort, Quick Sort, Insertion sort — Mix

Best choice

Binary Search:
100 elements of an array can be searched in 10 comparisons.

Condition is that elements must be in monotonic order, i.e. sorted order either (↑) or (↓).

Linear Search code

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

Binary search

```
int s=0;
int e=5;
int mid=(s+e)/2 or s+(e-s)/2
while (s<=e)
{
    if (arr[mid]==target)
    {
        return mid; -> index;
    }
    elseif (arr[mid]>target)
    {
        e= mid-1;
    }
```

1	2	3	4	5	6
s		mid			e

Time complexity of any sorting algo $n\log(n)$


```
elseif ( arr[mid] < target)
{
    s = mid+1;
}
mid = s+(e-s)/2;
}
```

Time complexity of Binary search = $O(\log n)$

$1024 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

```
#include <algorithm>
binarysearch (arr, arr+n, target);
```

base address

index

for vectors

```
binarysearch ( v.begin(), v.end(), target());
```

↳ finding the first occurance of an element.

```
int start = 0;
end = n-1 = 11-1 = 10
```

	0	1	2	3	4
s ←	1	3	4	4	4	→ e.

	0	1	2	3	4	5	6	7	8	9	10	
start ←	1	3	4	4	4	4	4	6	7	8	9	→ end

target = 4

mid = 5

```
while ( start <= end)
{ if( target == arr[mid]){
    ans = mid; end = mid-1;}
  elseif ( target < arr[mid]){
    end = mid-1;
  }
  else { start = mid+1;
  }
  mid = s+(e-s)/2;
}
return ans;
```

12
35

m : n
3 : 5
4 7

↳ Union

```
int doUnion(int a[], int n, int b[], int m)
{
    set<int> s;
    for(int i=0; i<n; i++)
    {
        s.insert(a[i]);
    }
    for(int i=0; i<m; i++)
    {
        s.insert(b[i]);
    }
    return s.size();
}
```

property of set

sets

set<int> s;

- → associates containers, stores unique elements
- → Duplicates not allowed
- → elements identifies itself
- → addition / deletion ✓ modification ✗
- → sorted order 1 2 3
- → TC O(log n)

set<int> s;	Reverse order set<int, greater<int>> s;
insert() find()	O(1) O(n)
erase() clear()	
begin() size()	
end() max_size()	

set
↳ self balancing BST

unorder set
↳ Hash table

Map
↳ stores key-value pairs
↳ here, key indentifies values
↳ Implementation → SBBST (RBT)
↳ TC - O(log n)

unique
<1, a>
<2, a>

Red black tree

↳ Pascals triangle

	0	1	2	3	4
0					1
1				1	2	1
2			1	3	3	1
3		1	4	6	4	1
4	1	5	10	10	5	1

1
3
5
7

j

i	0	1	2	3	4	5	6
0					1		
1				1	2	1	
2			1	2	3	2	1
3	1	2	3	4	3	2	1

5

4

(n-i-1) spaces
[1 → i+1]

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

5+0-1=4

5 - 8

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

```
for(int i=0; i<n; i++)
{
    for(int j=0; j<n+i; j++)
    { if(j<n-i-1) { cout<<" "; }
      else if(j<n)
      {
        cout<<c;
        c++;
      }
      else if(j==n)
      { c=c-2;
        cout<<c;
        c--;
      }
      else { cout<<c; c--; }
    }
}
```

	0	1	2	3	4	
	0	1				0
	1	1	2	1		2
	2	1	2	3	2	4
	3	1	2	3	4	
	4	1				

$5-3=2$ ×
$5-4=1$
$5-3$
$n-4$
$i-2$
2

$5-j$ $\quad$ $5-3$ $\quad$ $5-4$ $\quad$ $n-j$

$5/2$

1	1	
2	2	1
3	3	2 1

$4-3+1$
$n-j+1$

	0	1	2	3	4
$i-2$	1	2	3	2	1

$n-i-1$

$i-1$	0	1	2	3	4
				2	1

$4-3$ $\quad$ $4-4=0$

ex finding the last occurance of an element.

```
while( start <=end)
{
    if( target == arr[mid])
    {
        ans = mid;
        s = mid+1;
    }
    else if( target < arr[mid])
    {
        e = mid-1;
    }
    else if( target > arr[mid])
    {
        s = mid+1;
        mid = s+(e-s)/2;
    }
}
```

ex Total no. of occurances

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

first occurance 1

last occurance 6

total occurances = first occurance - last occurance + 1;

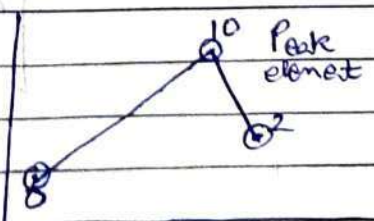
merge both the first and last occurances and print total occurances.

b) Peak element in a mountain - array

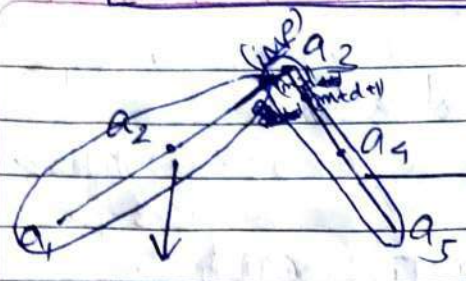
0	1	2	3
0	10	5	2

start <- 0 10 5 2 -> end

mid = $s + \frac{(e-s)}{2}$



$\boxed{arr[i-1] < arr[i] > arr[i+1]}$



$arr[i] > arr[i+1]$

$a_3 = arr[i]$

$arr[i] < arr[i+1]$

$\boxed{arr[i-1] < arr[i] > arr[i+1]}$

```
peakElement (Vector<int> arr){
  int start = 0; end = arr.size() - 1; int mid = start + (e-s)/2;
  while (end > start) {
    if ( arr[mid] < arr[mid+1]) {
      start = mid+1; }
    else if (arr[mid] > arr[mid+1]) {
      end = mid; }

    mid = start + (end - start)/2;
  }

  return start;
}
```

$\boxed{\begin{array}{l} s <= e \\ s = mid+1; \\ e = mid+1; \end{array}}$

ex 16 array is in descending order
same all the

```
(target > arr[mid])
  {
    end = mid - 1;
  }

(target < arr[mid])
  {
    start = mid + 1;
  }
```

$\boxed{\begin{array}{l} s<e \\ s = mid \\ e = mid \end{array}}$

$0 \longrightarrow 26$

number mid = 13

$$\begin{array}{r} 13 \\ \times\ 13 \\ \hline 39 \\ 130 \\ \hline 169 \end{array}$$

ex Square root of any number

```
int squareRoot (int number){
    int s=0; e=num;
    int mid = s+(e-s)/2;
    int ans = -1;

    while (s<=e){
        if(mid * mid == num){
            return mid;
        }
        else if (mid *mid < num){
            ans = mid;
            start = mid+1;
        } else { end=mid-1; }
        mid = s+(e-s)/2;
    }
    return ans;
}
```

```
decimal part
cin>>decimal
double step=0.1;

for (int i=0; i<decimal;
        i++){

    for (double j=finalAns;
        j*j <= nums; j=j+
                step)

        finalAns=i;
    }
    step= step/10;
}
cout << finalans;
```

Binary Search in 2D matrix

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

s → 1, e → 16

n rows
m columns elements = n*m-1;

$\frac{0-15}{2} = \boxed{7}$

(row) 7/4 = 1

arr[ ][ ] (column) 7%4 = 3

ex Search in a nearly sorted array.

$m = s + (e-s)/2;$

10	3	40	20	50	80	70	

target = 40.

```
while ( s<=e )
{
    if ( arr[m] == target)
    {
        return m;
    }
    if ( arr[m+1] == target) && m+1 < ~~size of arr~~ e
    {
        return m+1;
    }
    if ( arr[m-1] == target) && (mid-1) >= ~~0~~ s
    {
        return m-1;
    }
    if ( arr[m] < target)
    {
        s = m+1;
    }
    else
    {
        e = m-1;
    }
    m = s+(e-s)/2;
}
```

inf

$\frac{17}{3}$ 5

ex Divide using Binary search

$\sqrt{25}$ (s) 0 (e) 25

0 digit divident 0 8 17 8

(s) (e) 3

quotient * divisor = divident
+remainder

$\boxed{\text{quotient} * \text{divisor} \le \text{divident}}$

no = 17
divisor = 3

```
int divide (int divisor, int number)
    int s=0; int e=number;
    int m= s+(e-s)/2;
    int ans;

    while (mid <= e)
    {
        if(mid * divisor == number)
        {
            ans = mid;
        }
        else if( mid * divisor > number)
        {
            e= mid-1;
        }
        else{
            ans= mid;
            s=mid+1;
        }
        mid = s+(e-s)/2;
    }
```

ex Find the odd occuring element in an array

- all elements occur even no. of times except 1.
- all repeating occurance of element appear in pairs and pairs are non adjacent (there cannot be more than 2 consecutive occurance of any element)
- find out the element that occur odd number of times

- Find pair with diff. k in array
- Find k closest element to a given value in array
- exponential search
- unbounded binary search
- painter partition
- aggressive cows
- equal sponch
- minimum sponch
- book allocation

are similar

1	1	2	2	3	3	4	4	3	600	600	4	4

imp. XOR of all the elements

$1 \wedge 1 \wedge 2 \wedge 2 - - - \wedge 4$

$\rightarrow TC = O(n)$

- classic problems discussed previously.
- search space में find karna (predicate function)
- to keep an eye on index

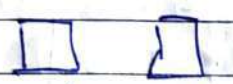
0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	2	3	3	4	4	3	600	600	4	4
e	o	e	o	e	o	e	o		o	e		

imp. ans will always be in even index

$s = 0, e = n-1$

$mid = s + (e-s)/2$

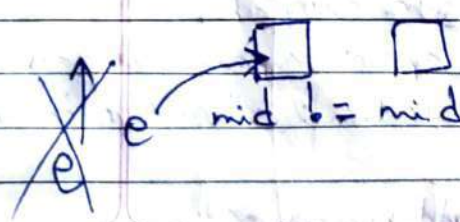
Case I mid → even



left mae ans right में होगा

mid == mid + 1 kahada hai

s = mid + 2;



mid != mid + 1 left search e = mid

mid can be or

mid may be answer

Case II

mid → odd

[ ] [ ] Left side

mid-1 == mid s = mid+1;

[even] [odd]

!= → ans ✗

→ right ki answer hai

left

e = mid-1;

```
int oddoccurance (int arr[], int n)
{
    int s=0; int e=n-1; int mid=s+(e-s)/2; int ans;
    while (s<e)
    {
        if (mid%2==0) //even
        {
            if (arr[mid]==arr[mid+1])
            {
                s=mid+2;
            }
            else
            {
                e=mid;
            }
        }
        else if (mid%2==1) // odd
        {
            if (arr[mid]==arr[mid-1])
            {
                s=mid+1;
            }
            else{
                e=mid-1;
            }
        }
        mid=s+(e-s)/2;
    }return arr[mid];
```

ex Pivot element in a rotated sorted array.

Sorted array

1 2 3 4 7 14 19 99

rotated sorted array

19 [99 1] 2 3 4 7 14

```
int rotatedsortedarrayprotelenat (int nums[], int n)
{
    int s=0; int e = n-1; int m = s+(e-s)/2;
    while (s<=e)
    {
        if (s==e)
        {
            return nums[s];
        }
        if (m+1<e && nums[m] >nums[m+1])
        {
            return nums[m]
        }
        if (m-1 ≥ s && nums[m-1] > nums[m])
        {
            return nums[m-1];
        }
        if (nums[s] > nums[m])
        {
            e = m-1;
        }
        else {
            s = m+1;
        }
        m = s+(e-s)/2;
    }
```

The SC doesn't depend upon the inputs we take in main function. It depends upon the extra space we created in user defined function.

Lect 13. Sorting

To sort small

Unstable (~~[illegible]~~ left)

1. Selection sort.

	0	1	2	3	4	5
5 comparision	10	1	4	8	5	7
5 comp.	1	10	4	8	5	7
4 comp.	1	4	10	8	5	7
3 comp.	1	4	5	10	8	7
2 comp.	1	4	5	7	10	8
	1	4	5	7	8	10

min

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

    for (int j=i+1, j<n ; j++)
    {
        if (arr[j] < arr[mid])
        {
            midindex = j;
        }
    }
    swap (arr[i], arr[mid]);
}
```

Time complexity $O(n^2)$

Space complexity $O(1)$

To bring i^{th} largest element to its correct place.

Stable

Bubble sort. (right)

हर i^{th} round में i^{th} largest element को सही right position पर डालते हैं।

$\boxed{a > b}$ swap

	0	1	2	3	4	5
	10	1	7	6	14	9
	1	10	7	6	14	9
	1	7	10	6	14	9
	1	7	6	10	14	9
	1	7	6	10	9	14

sorted

Round 1

1st Largest element is in its correct place.

1	7	6	10	9	14

Round 2

1	6	7	10	9	14

1	6	7	9	10	14

Sorted

1	6	7	9	10	14

Round 3

1	6	7	9	10	14

Round 4

1	6	7	9	10	14

Round 5

Total no. of comparisions
$= 4 + 3 + 2 + 1$
$\frac{n(n-1)}{2}$
$O(n^2)$

```
for (int round=1 ; round<=n-1 ; round++)
{
    bool swapped = false;
    for( int j=0; j<=n-round-1 ; j++)
    {
        if (arr[j] > arr[j+1])
        {
            swapped = true;
            swap ( arr[j], arr[j+1]);
        }
    } if (swapped == false) { break; }
}
```

Best case

1	2	3	4	5

4 comparisions
$O(n)$ ~~else~~ $O(n^2)$ in other cases

Worst comparision [in reverse sort]

5	4	3	2	1

Round 1

4	3	2	1	5

Round 2

3	2	1	4	5

Round 3

2	1	3	4	5

Round 4

1	2	3	4	5

vector<int> arr = { 10, 1, 7, 6, 14, 9 }
n = size();

stable Insertion sort

10	1	7	6	14	9

1	10	7	6	14	9

1	7	10	6	14	9

1	6	7	10	14	9

1	6	7	10	14	9

1	6	7	9	10	14

```
for (int round = 1; round < n; round++)
  { // step A fetch
      int val = arr[round];
      // step B compare
      for (int j = round-1; j >= 0; j--)
        {
          if (arr[j] > val)
            { // Shift
              arr[j+1] = arr[j];
            }
          else {
                break
            }
        } // copy arr[j+1] = val;
  }
```

fetch
compare
shift
copy

ex kth diff pair

```
int findpair (vector<int> & nums, int k){
    sort (nums.begin(), nums.end());
    set <pair <int, int>> ans;
    int i=0, j=1;
    while (j< nums.size()) {
        int diff = nums[j] - nums[i];
        if (diff == k){ ans.insert({nums[i], nums[j]});
            ++i, ++j;
        }
        elseif (diff > k) { i++; }
        else { j++; }
        if (i==j) { j++; }
    }
    return ans.size();
}
```

Method 2

binary search code.

```
int findpairs (vector<int>&
nums, int k){
 sort(nums.begin(), nums.end());
 set<pair<int, int>> ans;

  for (int i=0; i < nums.size()
      ; i++){
    if (bs(nums, i+1, nums[i]+k) != -1){
    ans.insert({ nums[i] , nums[i]+k});
  } return ans.size();
}
```

k^{th} closest element.

k=4 | x=35

	12	16	22	30	35	39	42	45	48	50	53	55	56
diff	23	19	13	5	0	4	7	10	13	15	18	20	21
	↓	↓	↓					↓	↓	↓	↓	↓	↓
	~~k~~	~~k~~	~~k~~					~~k~~	~~k~~	~~k~~	~~k~~	~~k~~	~~k~~

```
vector <int> findclosetElement (vector <int> &arr, int k, int x)
{
    int l = 0;
    vector <int ans;
    int h = arr.size()-1;
    while (h-l >= k)
    {
        if (x - arr[l] > arr[h] - x) {
            l++;
        }
        else {
            h--;
        }
    }
    while (l <= h) { ans.pushback (arr[l++]); }
    return ans;
}
```

Time complexity ~~as~~ O(n-k) as we have done n-k comparisions.

OR

```
vec <int> C
return ans.begin + l, arr.begin()+h+1);
```

```
bool Comparator (int a, int b)
{
    return (a>b);
}
```

sort(arr.begin, arr.end, Comparator);

1234 a<b 4321

	12	16	22	30	35	39	42	45	48	50	53	55	56
diff	23	19	13	5	0	4	7	10	13	15	18	20	21
			L	L	H	H	H	H					

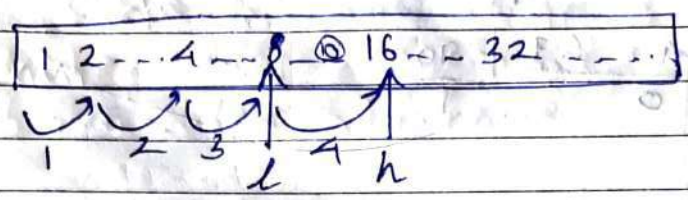
```
vector <int> bs_method (vector <int> arr, int k, int x){
    int h = lowerBound (arr, x);
    int l = h-1;
    while (k--){
        if ( arr(l) - x > arr(h) - x)
        {
            h++;
        }
        else { l--;
        }
    }
    return vector<int> (arr.begin()+l+1, arr.begin+h)
}

int lowerbound ( int arr[], int x )
{   int s=0, end= arr.size()-1;
    int ans = end;
    while (s<=e)
    {   m = s + (e-s)/2;
        if ( arr[m] >= x){ ans = mid;
            e = m-1;
        }
        else if ( x > arr[mid]){
            start = m+1;
        }
        else { end = mid-1;
        }
    } return ans;
}
```

Exponential search (applied on sorted array)

	3	4	5	6	11	$\boxed{13}$	14	15	56	70
index	0	1	2	3	4	5	6	7	8	9

if

TC



$\log(2^4)$ $\log_2 2^4 = 4$ steps

$O(\log(m))$ $\log 2^{\log m - 1}$

or

$\log\left(\frac{2^{\log m}}{2}\right)$

Applications:

search in ∞ arrays

search in array is target element is at the begining

```
search in vector (int vec[], int x, int n){
    if (vec[0]==0){
        return 0;
    }
    else { int i;  for(int i=1; i<=n-1; i=i*2){ if(vec[i]<=x);
                cout<< "we are at index: "<<i<<endl;
    }
    return binarysearch (vec, x, i/2, min(i, n-1));
}
```

ex Book allocation problem

12	34	67	90
0	1	2	3

① 12 | 34 67 90 — m = 2 — max = 191
0 | 1

② 12 34 | 67 90 — max = 157
0 | 1

③ 12 34 67 | 90 — max = 113
0 | 1

min = 113

if no. of students > no. of books
return -1

0 [ ] 203
101 max

int mid = $\frac{0 + 203}{2}$ = 101

Try to allocate each student atmost 101 pages

I → 12 + 34 + ~~67~~ = ~~113~~

II → 67 + ~~90~~ = ~~157~~ — 101 is not a possible solution.

III → ~~90~~ → ✗ — if (not possible)

102 152 203 — 3 — s = mid+1;

mid = $\frac{102 + 203}{2}$ = 152

I → 12 + 34 + 67 + ~~90~~ = ~~203~~ possible

II → 90

if (possible sol) store
{ // minimise the pages
end = mid - 1;
}

102 126 151

I 12 + 34 + 37 + 90 = 203
II 90 Possible

112
102 126

$\frac{102+126}{2} = 113$

I 12 + 34 + 67 + 90 = 203 Possible
II 90

107
102 112

I 12 + 34 + 67 = 113
II = 67 + 90 = 157 not possible solution.
III = 90 ✗

110
108 112

I ✗ 12 + 34 =
II ✗ 67 not possible
III ✗ 90

111 112

I 12 + 34 + 67 = 113 not possible
II 67 + 90 = 157
III 90 start = mid + 1;

112
112 112 } not possible

```
findPages (int A[], int N, int M)
{
    if (M>N) return -1;
    int low = *max_element (A, A+N);
    int high = accumulate (A, A+N, 0);
    int mid, no. of students;
    while (low<high)
    {   mid = low + (high-low)/2;
        no. of students = binarysearch (A, mid, N);
        (no of students > M) ? low = mid+1 : high = mid;
    }
    return low;
}

int binarysearch (int A[], int mid, int N){
    int studentcount = 1;
    int pages = 0;
    for (int i=0; i<N; i++){
        if (pagescount + A[i] <= mid){
            pages count += A[i];
        }
        else {
            studentcount++;
            pages = A[i];
        }
    }
    return studentcount;
}
```

ex Painter partision problem

~~5 10 30 20 15~~

10 20 30 40 k=2 n=4

10 | 20 | 30 | 40 time 90
10 20 | 30 40 time 70
10 20 30 | 40 time 60

```
long long minTime(int arr[], int n, int k){
    long long s=0;
    long long e=0;
    long long mid;
    for(int i=0; i<n; i++){
        e=e+arr[i];
    }

    longlong ans=-1;
    while(s<=e){
        mid= s+(e-s)/2;
        if(isposible(arr, n, k, mid)){
            ans=mid;
            e=mid-1;
        }
        else{
            s=mid+1;
        }
    }
    return ans;
}
```

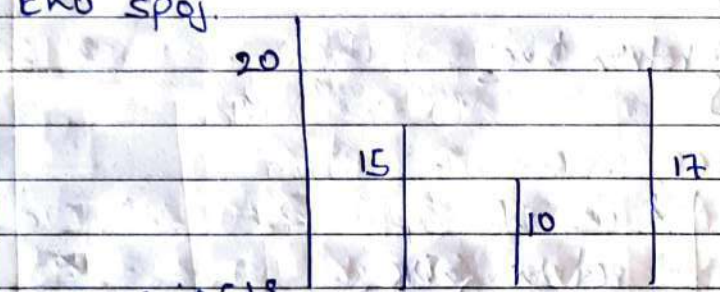
```
bool isPossible(int arr[], int n, int k, long long mid){
    int c=1;
    long long timetaken=0;

    for(int i=0; i<n; i++){
        if (arr[i] >mid)
            {
                return false;
            }
        if ( arr[i] + timetaken > mid)
            {   c++;
                timetaken = arr[i];
                if (c>k) { return false; }
            }

        else {
                timetaken = timetaken + arr[i];
            }
    }
    return true;
}
```

Ex Agressive cows

```
int solve (int n, int k, vector<int> & stalls){
    sort (stalls.begin(), stalls.end());
    int e = stalls[n-1] - stalls[0]; int s=1; int mid=
    while (s<=e){  mid = s+(e-s)/2;
        if (isPossibleSolution(stalls, mid, k)){
            ans=mid;  s= mid+1;
        }
        else {  e = mid-1;
        }
    }
    return;
}
```

```
bool ispossiblesolution (vector <int> stalls, int mid, int k){
  int cow=1;
  int lastcordinate = stalls[0];
  for (int i=1; i< stalls.size() -1; i++){
      if (stalls[i] - lastcoordinate >=mid){
          cow++;
          last coordinate = stalls [i];
      }
      if (cow >=k)
      {
          return true;
      }
  }
  return false;
}
```

ex EKO spoj.




```
int main(){
  vector <int> tree = {20, 15, 10, 17};
  int start = 0; int required = 7; int end = *max_element (
  trees.begin(), trees.end());
  int ans=-1;
  int mid;

  while (start <= end)
  {
      mid = start+(end - start)/2;
      if (ispossible solution (trees, mid, required))
      {  ans =mid;
         start=mid+1;
      }
```

```
    else {
        end = mid - 1;
    }
}
cout << ans;
return 0;
}

bool isposible solution( vector <int> trees, int mid,
    int required ){ int treescollected = 0;
    for (int i=0; i< trees.size (); i++){
        if ( trees[i] > mid ){
            treescollected += trees[i] - mid;
        }
    }
    if ( treescollected >= required )
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

ex Paratha spoj.

int nP = 10, nC = 4

vector <int> cooksRanks = {1, 2, 3, 4}

min = 0 max =

[1 + 2 + 3 + 4 + . . . + 10]

= 55 x 4 = 220

0 — mid = 110 — 220

Cook1

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

= 55 mid

Yes

ans = mid

end = mid-1

0 54 109

Cook1

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

Cook2 in 45 min.

2

0 26 53

Cook1	1+2+3+4+5+6+7 = 21
Cook2	2+4+6 = 12

12

0 25

Cook1	1+2+3+4+	4	10
Cook2	2+4+6	3	
Cook3	3+6+	2	
Cook4	4	1	

5

0 11

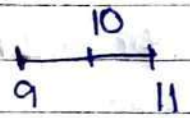
cook1	1+2	2
cook2	2	1
cook3	3	1
cook4	4	1
		total 5

7

6 11

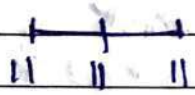
Cook1	1+2+3
Cook2	2+4
Cook3	3
Cook4	4

7 parathas

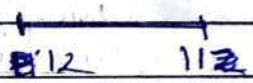


1+ 2+3 +4 — 4
2+4 — 2
3+6 — 2
4 — 1

9 parathas



1+2+3+4
2+4
3+6
4 — 9 parathas



not possible so ans is 12

```
int main()
{   int nP = 10, nC = 4, s = 0, e = 220, mid, ans;
    vector<int> cookesrank = {1, 2, 3, 4};
    while (s<=e)
    {   mid = s + (e - s)/2;
        if (is possiblesolution (cookesrankes, nP, nC, mid))
        {   ans = mid;
            e = mid-1;
        }
        else {
            s = mid+1;
        }
    }
    cout << ans;
    return 0;
}
```

```
bool ispossiblesolution(vector<int> cookerranks, int nP,
    int nC, int mid){
    int parathascooked=0;
    for (int i=0; i<nC; i++){
        int timetaken=0, j=1, currentcookedrank = cookerrank[i]
        while(true) {
            if (timetaken + j* currentcookerrank ≤ mid ){
                timetaken += j * currentcookedrank;
                j++;
                parathacooked++;
                if ( parathascooked >= nP)
                {
                    return true;
                }
            } else{
                break;
            }
        }
    }
    return false;
}
```

ex Length of a string

```
int i=0;
int length=0;
while (arr[i] != '\0') {
    length++;
}
cout << length;
```

ex Reverse of a string

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

ex Uppercase

```
for(i=0; i<strlen(
word); i++){
word[i] = word[i]-32
}
```

lowercase

```
word[i]= word
[i] +32
```

Array is a datastructure and not data type.

end to remove

properties.

S.length()

S.empty()

S.push_back()

S.popback();

S.substring(10, 4) → from index 10 print 4 characters

S.replace(0, 4, word);
start

S.erase(10, 8) → it removes characters remove v... above
index to start

S.compare( );

Hello World S.find("word"); // 6 // if no word is found then garbage value

↳ Pointers

↳ stores address

it is not a datatype it is just a variable that stores an address

is a pointer to

int data int * ptr = & a;

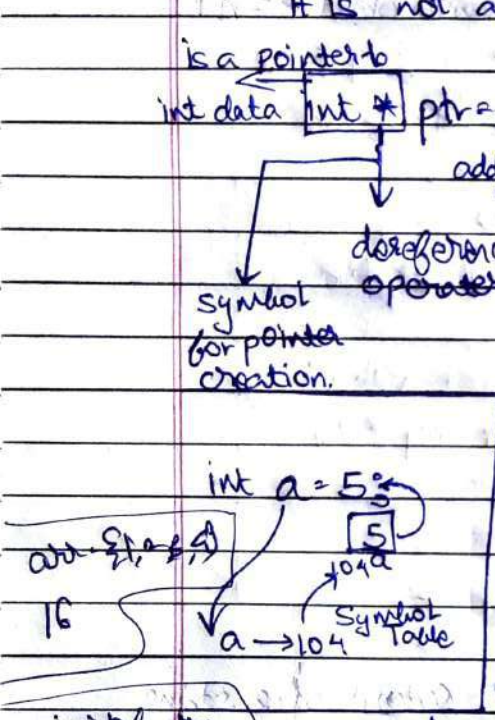
address of operator

dereference operator

symbol for pointer creation.

int a = 5;

int * ptr = & a;



code a/5

cout << *a // error

cout << *ptr // 5

cout << ptr // A1

cout << & a // A1

cout << & ptr // A2

int * p = 0;
null pointer

Use case of pointer

↳ dynamic memory allocation

↳ memory management

↳ pointer arithmatic (go from one location to another)

↳ pass by reference in an array

↳ to create pointer to function (passing a function inside another function as an argument)

int * ptr;

cout << ptr; → garbage value

Help Range
Custom comparator
• Largest number

791 custom sort string
953 verify Alien Dictionary
524 Longest word in Dictionary Through deleting.

Date : 63

imp ex

Palindromic substrings

noon

n
o
oo
noon

int count=0;

int *p[3], marks
int a[] = {20, 4, 60, 75, 34};
int b[] = {81, 22, 11};
int c[] = {10, 30, 50, 72, 49, 37};
int size[3] = {5, 3, 6};
p[0]=a;
p[1]=b;
p[2]=c;
marks = highest marks (p, size, 3);

int max=0, i, j;
for (int i=0; i<n; i++){
for (int j=0; j<size[i]; j++)
if (max< ptr[i][j])
max= ptr[i][j];
} return max;

map:
at(), size(), empty(),
insert(), clear(),
erase(), count(),
find(),

=0; i<s.len; i++)

oddnos = checkaroundindex(s, i, i);
unt = count + oddnos;
t evennos = checkaroundindex(s, i, i+1);
count = count + evennos;
return count;

checkaroundindex(string s, int i, int j){
int count=0;
while (i>=0 && j< s.length() && s[i]== s[j])
{ i--;
j++;
count++;
} return count;
}

bool cmp(char first, char second){
return first > second
}
(→ descending order)

return first < second
(→ ascending order)

sort(s.begin(), s.end(), cmp);

Subtracting an integer from a pointer decrements the pointer. Similarly, p+1 will move p back by 4 bytes to the previous integer. P-5 to an integer, p-1

int a = 5 int*p = &a; → size
char b = 'b' int*q = &b; → size
double c = 1.03 int* r = &c; → size

Imp
8
8
8

P is a pointer to integer datatype
q is a pointer to char datatype
r is a datatype to double datatype

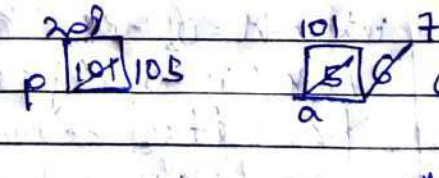
Pointer always stores address of datatype and it depends on architecture but it is always equal.

int *ptr;
cout << *ptr; segmentation fault BAD PRACTICE

int *ptr = 0; // null pointer
cout << *ptr; again segmentation fault

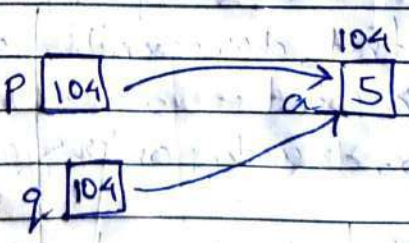
Modern
int * ptr = ~~Null pointer~~ nullptr;

int a = 5;
int * ptr = &a;



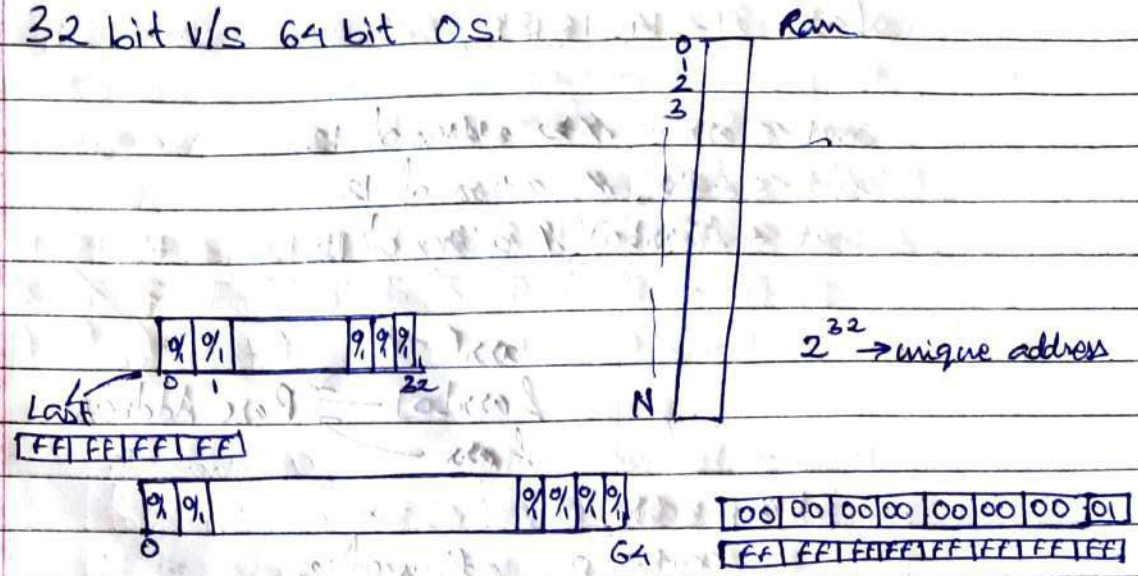
a = a+1;
p = p+1;
*p = *p+1;

int a = 5;
int *ptr = &a;
int *ptr2 = ptr; // copy pointer.



void* pointer is a generic pointers type that can hold the address of any data type. It is often used in the scenerios where you need to work with pointers to data of unknown or varying types.

32 bit v/s 64 bit OS.



32 bit performs operation in 2 cycles 2^{22} addresses-1

64 bit performs operation in 1 cycle 2^{64}-1 addresses

2) 64 bit 4 GB ≤ Ram installation is possible
Resource utilisation

3) Performance 64 bit takes less cycle
it can operate on 1 instruction cycle ⇒ 64 bits
32 bit take one instruction cycle ⇒ 32 bits

4) Compatability 64 bit CPU can run both 32 bit and 64 bit OS while visa versa not possible.

5) Better Graphic performance: 8 types of graphics calculations ~~takes~~ ~~more~~ makes graphics intensive apps

64 bits
8 bits in size pointers

To enable the large space x86-64 bit processor require 8 byte storage. This means any pointer variable in C++ that is used to store memory address on 64 bit platform must be 8 byte in size.

arr[4] = {12, 14, 16, 18};

cout << arr; // address of 12
cout << &arr; // adress of 12
cout << &arr[0]; // address of 12

arr
&arr[0] → Base Address
&arr

int * p = arr;
cout << p address of 12
cout << &p address of p.

int arr[0];

SI

arr → 104

5	9	14	20	24

int * p = arr;

p → 208
arr → 104

*arr+1 = 6
*(arr+1) = 9

cout << arr[i]; ✓
cout << i[arr]; ✓

arr = arr + 2; ✗
but pt = pt + 2; ✓

Diff

① sizeof(arr) 40 | sizeof(ptr) 8
② arr=arr+2; | pt=pt+2;

cout << arr+1	cout << *ptr = arr+1	cout << arr = arr+1;
✓	✓	✗

Because arr is a constant pointer.

The name of a array is actually pointer to the first element of the array. This means you cannot modify the value of the array pointer itself.

arr = arr+1; // compile time error because you are trying to modify constant pointer

imp

```
char ch[10] = "babbar";
char *c = ch;
cout << c << endl;        // Babbar        *c → B
```

c[0]
*c → B

c → Babbar c+2 → bbar

2 से लेकर end तक

ex

```
char ch = 'k';
char * cptr = &ch;        k ^ ^ ^
cout << cptr << endl;     garbage value
```

ex

```
char name[10] = "babbar";
cout << name << endl;     babbar
```

① temp storage → "babbar"
② mem change → copy to name array ki storage

```
char *c = "babbar";
cout << c << endl;        babbar
```

① temp storage → "babbar"

② [ ] c Bad practice

naya pointer bana hai pointer

```
int arr[10] = {1, 2, 3, 4}

int main(){
    sizeof(arr); //40
    solve(arr);
}
```

104
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |

```
void solve(int arr[]);
{
    sizeof(arr); //8
}
```

(pointer)

216 | 104 | arr 8 byte

check for prime

2 _____ sq(n)

Q

Sieve of Eratosthenes

N<21

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
T	T	~~T~~	T	~~T~~	T	~~T~~	~~T~~	~~T~~	T	~~T~~	T	~~T~~	~~T~~	~~T~~	T	~~T~~	T	~~T~~
		F		F		F	F	F		F		F	F	F		F		F

Called as Sieve

```
int countprimes(int n){
    vector<bool> prime(n, true);
    if(n==0) return 0;
    int ans = 0;
    for (int i=2; i<n; i++){
        if (prime[i]){
            ans++;
            int j = i*2;
            while (j<n){
                prime[j] = false;
                j = j+i;
            }
        }
    }
    return ans;
}
```

$TC = n \cdot \left[\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \frac{n}{7} \ldots\right]$

$n^2\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} \ldots\right)$

Tailer series

$O[n \log(\log n)]$

gcd of two nos

$\gcd(a, b) = \gcd(a-b, b) \quad a>b$

$\gcd(b-a, a) \quad a<b$

apply this till one of the parameter becomes 0.

Euclids algorithm

gcd (72, 24)
gcd (48, 24)
gcd (24, 24)
gcd (0, 24)
↳ ans

```
int gcd (int A, int B){
    if (A==0) return B;
    if (B==0) return A;
    while (A>0 && B>0){
        if (A > B) { A = A-B;
        }
        else { B = B-A;
        }
    }
    return A==0 ? B : A;
}
```

ex Fast exponential

```
int fastexpo( int a, int b){
    int ans=1;
    while (b>0)
    { if (b&1) { ans = ans*a;
                }
      a = a*a;
      b = b>>1;
    }
    return ans;
}
```

Time complexity $O(\log b)$

ans = 1 a = 5 b = 4	a = 2 b = 5 ans = 1
a = a*a = 25 b = 2	ans = ans*a 2
a = a*a = 625 b = 1	a = a*a = 4 b = 2
ans = ans*a ans = 625;	a = a*a = 16 b = 1
	ans = ans*a = 16*2 = 32
	b = 0

Wild pointers in C++.

i. Uninitialised pointer

```
int *ptr;
cout<< *ptr <<endl;   //undefined behaviour
```

ii. Deleting pointers.

```
int *ptr = new int;
delete ptr;
cout<< *ptr <<endl; // program crash
```

iii. Pointers to non-existential variables

```
int *ptr = &x   but x is not declared or initialised
cout<< *ptr <<endl;
```

Void pointers = Generic pointers

```
void * ptr;   we cannot dereference it directly
```

```
void * ptr;
int x = 10;
ptr = &x;
```

It is a pointer which has no associated datatype with it. It can point to any data of any datatype and can be typecasted to any

```
int *intPtr = static_cast<int*>(ptr);
cout<< *intptr <<endl; //prints 10
```

```
int main()
{
    int n = 10;
    void * ptr = &n;
    printf("%d", * (int*) ptr);
    return 0;
}
```

typecasted

we cannot dereference a void pointer

```
printf("%d", * ptr);
```

Malloc and Calloc function returns a void pointer. Due to this reason, They can allocate a memory for any type of data.

Syntax: void * malloc (size_t size);

↳ Dangling pointers

A dangling pointer is a pointer which points to some non-existing memory locations

```
int main()
{
    int * ptr = (int *) malloc (sizeof(int));
    ...
    free(ptr);
    return 0;
}
```

ptr is pointing to the 1st Byte of the allocated memory

memory is now released.

But the pointer is still pointing to the deallocated memory.

After freeing the pointer the pointer becomes dangling

Solve

so

```
free(ptr);
ptr = NULL;
```

Now pointer is no more dangling

```
int * fun()
{
    int num = 10;
    return &num;
}
```

int is only limited to this function (LOCAL VARIABLE)

```
int main() {
    int * ptr = NULL;
    ptr = fun();
    printf("%d", *ptr);
    return 0;
}
```

//segmentation fault

arr[10] &arr +1 moves forward 40 bytes
arr +1 moves forward 4 bytes

↳ Null pointer

A null pointer is a pointer that does not point to any memory location. It represents a invalid memory location

When a null pointer is assigned to a pointer, then the pointer is considered as NULL pointer

```
int * ptr = NULL;
```

Use of null pointer

① It is used to initialize a pointer when that pointer isn't assigned any valid memory address yet.

② It is usefull for handling errors when using malloc function

```
int * ptr;
ptr = (int *) malloc ( 2 * sizeof (int));
    if ( ptr == NULL)
        printf ("Memory could not be allocated");
    else
        printf ("Memory allocated Sucessfully");
```

Facts about NULL pointer

① The value of NULL is 0. We can either use NULL or 0 but this is written in context of pointers and is not equivalent to the integer 0.

```
int main() {
    int * ptr = NULL;
    printf ("%d", ptr); //0
    return 0;
}
```

② Size of the null pointer depends upon the platform and is similar to the size of the normal pointers

eg.
```
int main() {
    printf ("%d", sizeof(NULL));
    return 0;
}
```
//8

Some best practices

1. It is good practice to initialize a pointer as NULL.
2. It is a good practice to perform a NULL check before dereferencing any pointers to avoid surprises

```
int main(){
    int a=5;
    int *p = &a;
    util(p);
        cout<<a //5
    return 0;
}
```

```
void util(int *p)
{
    p=p+1;
}
```

*p = *p+1
// a=6

Generally used when passed by reference comes into picture.

Reference variable

Value → [5] a

a++ // 6
Value++// 6

Same memory location ko different naam से बुलाना

```
int a=5;
int & b=a;
```

104 [5] a↑↑b

S.T.
a → 104

b is a reference variable and pointing to the same memory location of a.

```
int a=5;
int & b=a;
cout<< a ;//5
cout<< b ;//5
```

Same memory locations different names.

Reference variable cannot be set to null but a pointer ~~var~~ can. Safety is more in case of RV. Also pointers are difficult to understand.

```
int main(){
    int a=12;
    solve(&a);
}
```

```
solve (int *value){
    *val = *val +1;
}
```

13

```
int main (){
    int a=5;
    int * p=&a;
    cout<<"before "<<p;
    solve(p);
    cout<<"after "<<p;
}
```

datatype

```
solve (int *&p){
    p = p+1;
}
```

// Pass by Reference
// same memory diff. names

imp

```
int * solve()
{
    int a=5;
    int * ans=&a;
    return ans;
}
```

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

```
int arr[10];
int *ptr= arr; ✓
```

```
int arr[10];
int *ptr=&arr;
int *ptr =&arr[0]; ✓
int (*ptr)[10] =&arr;
```

```
cout<<(*ptr)[0]; // 1
cout<<(*ptr)[1]; // 2
```

It is a pointer to an array of 10 integers. This means that ptr is a pointer that points to the first element of an array that contains 10 integers.

Optimising Sieve of eratosthenes

```
int countprimes(int n) {
    vector <bool> prime(n, true);
    if(n==0) return 0;
    int ans = 0;
    for (int i=2, i<n; i++){
        if (prime[i]){
            ans++;
            int j = i*2;
            while (j<n){ prime[j] = false;
                j = j+i;
            }
        }
    }
    return ans;
}
```

OR → $\boxed{i*i<=n}$ opt 2 (imp)

$\boxed{int j = i*i}$ opt 1 (imp)

2 → 4 6 8 10 12 14 16 18 20

3 → 6 9 12 15 18 21 24

5 → 10 15 20 25 30

7 → 14 21 28 ?

i*i से पहले हमने mark कर दिया है।

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
F	T	T	~~T~~ F	T	~~T~~ F	T	~~T~~ F	~~T~~ F	~~T~~ F	T	~~T~~ F	T	~~T~~ F	~~T~~ F	~~T~~ F	T	~~T~~ F	T	F

$j = i*i$ } start

$j = 7\times7 = \underline{49}$

$i = 7$ $j = 49$

$j = 9$ $j = 81$

$i = 2$ $\boxed{i <= \sqrt{N}}$ opt 2

or

$i*i <= N$

7 | 14 21 ~~28~~ ~~35~~

→ start 49 > Range

Problem to be practiced in gfg.

int a[100]
↑
Max size 10^6
(int, double, char) ~~array~~
bool 10^7
Global array int double char 10^7
bool 10^8
Although it depends on the architecture

$1 <= (L, R) <= 10^9$
↳ $(R-L) \leq 10^6$

Segmented sieve
L = 110 R = 130

110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

1 Generate all primes responsible to mark segment sieve
2 find i^{st} index to find marking

```
vector <bool> segSieve (int L, int R){
    vector <bool> sieve = Sieve (sqrt(R));
```

```
vector <bool> Sieve (int n){
    vector <bool> sieve(n+1, true);
    sieve[0] = sieve[1] = false;

    for (int i = 2; i*i <= n; i++){
        if (sieve[i] == true){
            int j = i*i;
            while (j <= n){
                sieve[j] = false;
                j += i;
            }
        }
    }
```

```
int main(){
    vector <bool> sieve =
        Sieve(25);
    for (int i=0; i<=25; i++)
    {
        if (sieve[i]){
            cout << i;
        }
    }
    return 0;
}
```

```
vector <bool> segSieve (int L, int R){
    vector<bool> sieve = Sieve (sqrt(R));
    vector <int> baseprimes;

    for( int i=0; i<sieve.size(); i++) {
        if (sieve[i]){
            base primes.push-back(i);
    }
```

$\sqrt{R}$. 11

2 3 5 7 11

② Now start marking

Index 0 resemble 110
index 20 resemble 130

$\frac{110}{2} \times 2 = 110$

Prime = 2 start marking with 110.

$\frac{110}{3} \times 3$ (36)
108

Normal sieve j = i * i
seg sieve j = max(first multiple, prime * prime)

```
vector<bool> segSieve (R-L+1, true);
    if( L==1 || L==0){
        segSieve [L] = false;
    }

    for (auto prime: baseprimes){
        int first multiple = (L/prime) * prime;
        if ( first-multiple < L){
            first-mult += prime;
        }
        int j = max ( first-mult, i*i);
        while (j <= R){
            segsieve[j-L] = false;
            j += prime;
    }
return segsieve;
```

```
void sieveofEratosthenes (int n)
{
    ~~bool primes[n+1];~~
    vector<bool> sieve (n+1, true);
    sieve[0] = sieve[1] = false;

    for (int i=2; i*i <= n; i++){
        if (sieve[i]) {
            for (int j = i*i ; j <= n; j += i)
            {
                sieve[j] = false;
            }
        }
    }
```

printing

```
for (int p=2; p<=n; p++)
{   if (sieve[p]) {
        cout << p << " ";
    }
}
```

Segmented sieve $O(n \log\log n)$

L = 110 R = 130 Limit = 21 = (R-L+1) [21 से ज्यादा array नहीं बना सकता]

responsible

① Generate all primes to mark segmented sieve from 0 to floor($\sqrt{n}$)

l = 6 n = 10

floor($\sqrt{n}$) = 3

primes[] = {2, 3}

Time complexity $\sqrt{n} \log\log\sqrt{n}$

$O(\sqrt{n})$ Aux space

② Create an array of size (h - l + 1) i.e. (10 - 6 + 1) = 5

isPrime | T | T | T | T | T | (initially all true)

$O(h - l + 1)$ auxilary space

③ Mark multiples of primes[] = {2, 3} in isPrime[]

isPrime | F | T | F | F | F |

$(h - l + 1) \log\log n$

a = {2, 3, 4};
b = {2, 3, 4};

```
vector <long long> prime;
void segSieve (int l, int h)
{
    long long sq = sqrt(h);
    sieve (sq);
vector <bool> isPrime (h-l+1, true);
for (long long p : prime)
{
    long long sm = (l/p) * p;
    if ( sm < l)
        sm += p;
    for (long long m = sm;
        m <= h ; m += p)
        {
            isPrime[m-l] = false
        }
}
```

```
void sieve (int n)
{
    vector <bool> isPrime (n+1, true);
    for (long long i = 2; i <= n; i++)
    {
        if (isPrime[i])
        {
            prime.push_back(i);
            for (long long j = i*i; j <= n; j++)
                isPrime[j] = false;
        }
    }
}
```

```
// Printing
for (long long i = l; i <= h; i++)
{
    if (isPrime[i-l] == true)
    {
        cout << i << " ";
    }
}
```

1	2	3	4

a ↑ x

```
int * ptr = 0;
int a = 10;
* ptr = a;
cout << * ptr;
// error
runtime error
* (0x0000)
```

```
int a[] = {1, 2, 3, 4};
int * p = a++;
cout << * p << endl;
```

a = a+1 ✗ ~~dynami~~ constant pointer in stack

not a dynamic memory in heap.

200 212

ex
```
float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5}
float *ptr1 = &arr[0];
float *ptr2 = ptr1+3;
cout << *ptr2 << " ";
cout << ptr2-ptr1;
```
$\boxed{3}$

$212 - 200 = \frac{12}{4} = 3$

ex
```
void fn (int a[]) {
    cout << a[0] << " ";
}

int main() {
    int a[] = {1, 2, 3, 4}
    fn (a+1);
    cout << a[0];   // 2 1

return 0;
```

ex
```
void Q(int z)
{
    z += 3;
    cout << z;
}

void P(int* y)
{
    int x = *y + 2;
    Q(x);
    *y = x-1;
    cout << x;
}

int main() {
    int x = 5;
    P(&x);
    cout << x;

return 0;
}
```

ex
```
int f( int x, int* py, int** ppz){
    int y, z;
    **ppz += 1;
    z = **ppz;
    *py += 2;
    y = *py;
    x += z;
    return (x+y+z);
}
```
$\boxed{14\ 7\ 6}$

```
int main() { int c, *b, **a;
    c = 4; b = &c; a = &b;
    cout << f(c, b, a);
    return 0;
}
```

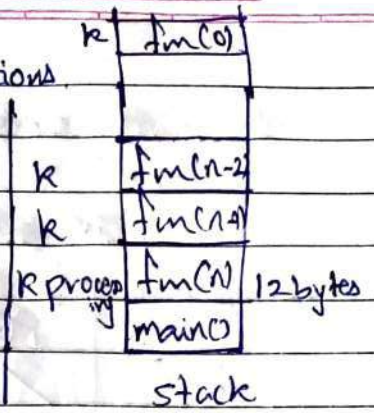
x = 7
y = 7
z = 5

$\boxed{19}$

Lecture Time and space complexity of Recursive solutions

```
main()
{
    fm(n);
}
```

```
fm()
{ if(n==0)
      return ...
  int a,b,c;
  fm(n-1);
}
```



↳ print array (linear traversal of an array)

```
void printArray(int a[], int n){
    if(n==0) return;
    cout << *a << " ";
    printArray(a+1, n-1);
}
```

a | 1 | 2 | 3 | 4 |

Output:
1 2 3 4

TC & SC of Recursive function

$T(n) = k + T(n-1)$

$T(n-1) = k + T(n-2)$

$T(n-2) = k + T(n-3)$

$\vdots$

$T(0) = k_1$

$T(n) = nk + k_1$

$= O(nk + k_1)$

$= O(n)$

Recursive tree method

PA(n) → k

PA(n-1) → k

PA(n-2) → k

PA(n-3) → k

PA(n-4) → k

P(1) → k

P(0) → k

$T(n) = nk + k_1$

$\underline{T(n) = O(n)}$

1	PA(0)	m
1	PA(1)	m
2		
n-2	PA(n-2)	m
n-1	PA(n-1)	m
n	PA(n)	m
	main()	

n space

$\underline{O(n+1)}$ $\underline{O(n)}$ space complexi

ex factorial

```
int fact (int n)
{
    if (n==1)
        return 1;
    return n* fact (n-1);
}
```

$fn = n * f(n-1)$

$TC = k + T(n-1)$ Time complexity

$T(n) = k + T(n-1)$

$T(n-1) = k + T(n-2)$

$T(n-2) = k + T(n-3)$

$\vdots$

$T(1) = k$

$T(n) = nk$

$O(T_n) = O(nk) \Rightarrow \underline{O(n)}$

$f(n) \rightarrow k$

$f(n-1) \rightarrow k$

$f(n-2) \rightarrow k$

$f(n-3) \rightarrow k$

$f(2) \rightarrow k$

$f(1) \rightarrow$

$T(n) = nk$

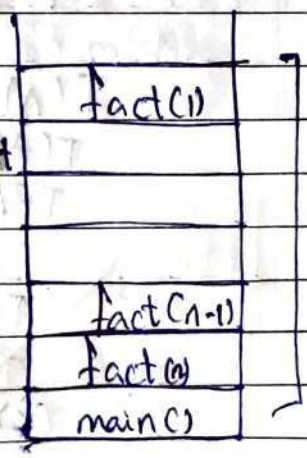
nk

Space complexity of factorial

Space complexity $= O(nm)$

$\hookrightarrow$ const

$= \underline{O(n)}$



ex Binary Search Recursively.

```
int BS(int a[], int k, int start, int end)
{
    if (start > end)
        return -1;

    int mid = start + (end - start) / 2;
    if(a[mid] == k)
        return mid;
    elseif (k > a[mid])
    {
        return bsr(a, k, mid+1, end);
    }
    else
    {
        return bsr(a, k, start, mid-1);
    }
}
```

$f(n) = \underbrace{k}_{\text{processing at every function}} + f(n/2)$

$T(n) = k + T(n/2)$

$T(n/2) = k + T(n/4)$

$T(n/4) = k + T(n/8)$

$\vdots$

$T(2) = k + T(1)$

$T(1) = k$

$T(n) = a * k$

$n \to n/2 \to n/4 \to n/8 \to \vdots \to 1$

2^a steps

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

$a = \log n$

$T(n) = O(\log n)$

Drawback of recursive is that it adds up the space in the stack which is not feasible.

Space complexity

BS(1)	$n/2a = 1 \rightarrow k$
BS(n/4)	$n/4 \rightarrow k$
BS(n/2)	$n/2 \rightarrow k$
BS(n)	$n \rightarrow k$
main()	

$a = \log n$

$O(T_n) = O(k \log n)$

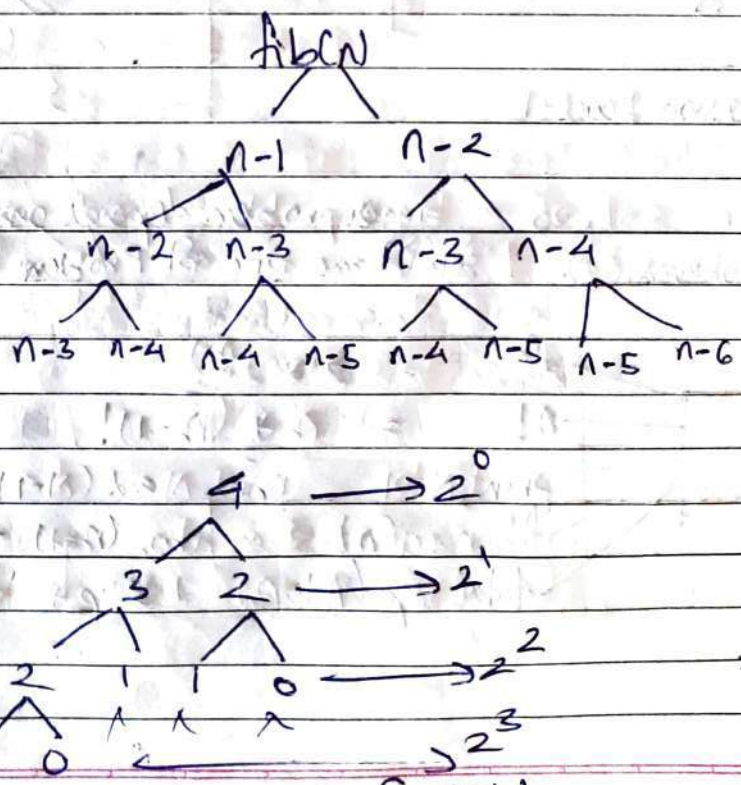
$\boxed{O(n) = \log n}$ → space complexity

Iterative BS > Recursive BS
(less space complexity) (more space complexity)

ex Fibonacii series RC

0 1 1 2 3 5 8
↓ ↓ ↓ ↓ ↓ ↓ ↓
0 1 2 3 4 5 6

```
int fib( int n) {
    if (n==0 || n==1) return n;
    return fib (n-1) + fib(n-2);
}
```



$2^0 * C$

$2^1 * C$

$2^2 * C$

$2^3 * C$

$2^4 * C$

⋮

$2^n * C$

$T(n) \le 2^0 + 2^1 + 2^2 + 2^3 \; (2^{n-1})$

$TC \le 2^0 * C + 2^1 * C + 2^2 * C + \ldots 2^3 \times C \ldots 2^n * C$

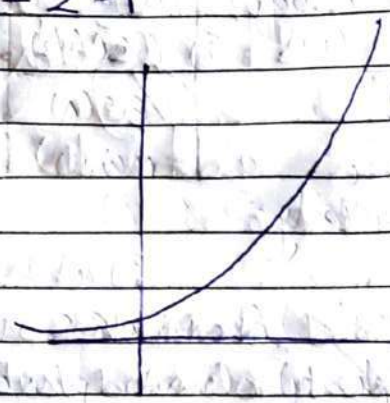
$TC \le C[2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^{n-1}]$

$TC \le C[2^n - 1]$

$TC \le 2^n - 1$

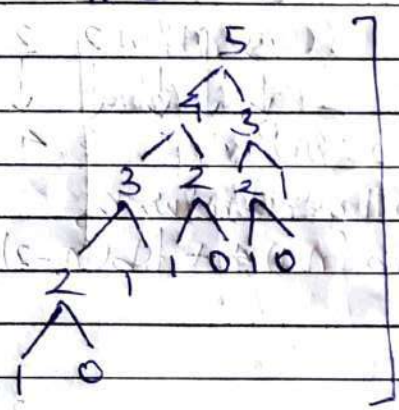
upperbound

$\boxed{T(n) = O(2^n)}$



vv big T(n)
very bad approach
Good approach is
Through DP. which is $O(n)$

$n = 5$



5

fib(1)
fib(2)
fib(3)
fib(4)
fib(5)
main()

$SC = O(n)$

Recursion Level-1

When sol. of a bigger problem depends on similar smaller problems. (s). [छोटी same type की Problems]

Big Problems

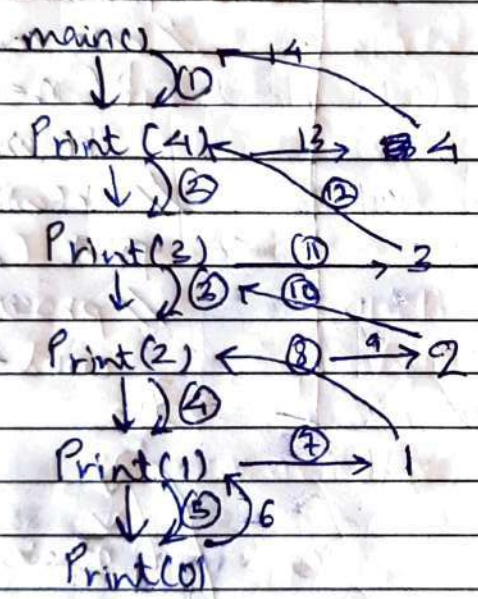
- $2^n = 2 * 2^{n-1}$
- $n! = n * (n-1)!$
- print N to 1 — print N and (N-1) to 1.
- fibonacci(n) — fibo. (n-1) + fibo(n-2)
- total steps(A to B) — 1 step + total steps (A, to B)

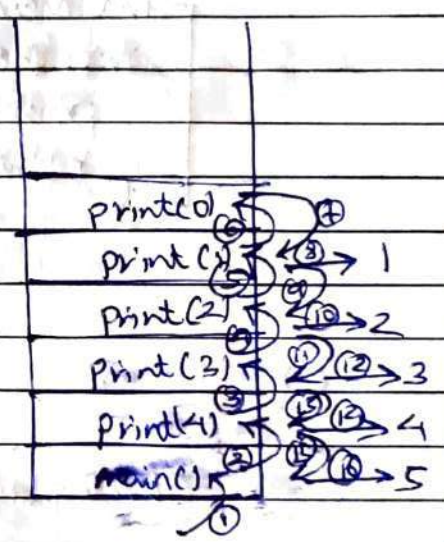
Components of recursion
- ↳ Base case / Stopping [condition रुकना कब है]
- ↳ Recurrence Relations
- ↳ Processing (optional)

Head Recursion
Recursive call before processing

Tail Recursion
Recursive call after processing

Recursive Tree





Fibonacci

```
int main() {
  int n=4;
  int ans= fib(4);
  cout << ans;
  return 0;
}
```

```
int fib(int n) {
  if (n==0 || n==1)
      return n;
  return fib(n-1)+fib(n-2);
}
```

n	0	1	2	3	4
fib(n)	0	1	1	2	3

alloca } used to create space in stack #include <stdlib.h>

[*arr++] X

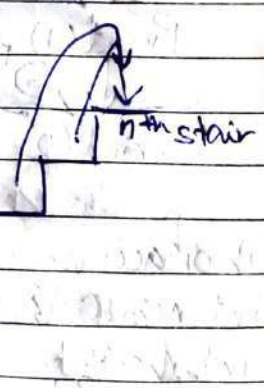
const ptr.
means cannot change the value of pointer

Magical Line:
Ek case solve krlo, baki recursion sambhal lega.
↳ Just believe on it, don't doubt on recursion.

```
int main(){
    int n=4;
    print(n);
    return 0;
}
```

```
void print(int n){
    // base condition
    if(n==0)
        return;
    // recurrence relation
    print(n-1);
    // processing
    cout << n;
}
```

ex: Climb stairs



steps allowed
↳ 1st stair at a time
↳ 2 stairs at a time

$\boxed{f(n) = f(n-1) + f(n-2)}$ total no. of ways

```
int climbstairs (int n){
    if (n==0||n==1)
    {
        return1;
    }
    int ans = climbStairs (n-1)+ climbstairs (n-2);
    return ans;
}
```

stairs
n = 0 return 1
n = 1 return 1

↳ printing elements of array

```
void print (int arr[], int n, int i){
    //base case
    if (i>=n){
        return;
    }
    // 1 case solve krdia
        cout << arr[i]<<" ";
    // baki recursion sambhal lega
        print (arr, n, i+1)
    }
```

max number

```
#include<limits.h>
INT_MIN
```

```
void findMax( int arr[], int n, int i, int maxi)
    if i>=N{
        return; // array khatam ho gaya
    }
    // 1 case solve karna hai
    if (arr[i]> maxi){
        maxi = arr[i];
    }
        findmax (arr, n, i+1, maxi);
```

int *p = &a
address of a

int *p = a
reference variable

When an integer is starts with 0, it is interpreted as an octal (base 8) value.
In the case of int a = 0647 = 423

(64x6)+(8x4)+7
384+32+7
= 423

```
ckkey (string str, int n, char key){
base case
if (i >= n){
    return false;
}
1 case solve kardo
if (str[i] == key)
    return true; i
// baki recursion sambhalegga
bool ans = checkkey (str, i+1, n, key);
    return ans;
}
```

647

ex

```
void printdigits (int n){
    //base case
    if (n==0){
        return;
    }
    int digit = n%10;
    cout << digit << " ";
    int newvalue of N = n/10;
    printDigits (newvalue of N);
}
```

Output
746

```
int newvalue of N = n/10;
printDigits (new value of N);
int digit = n % 10;
cout << digit << " ";
```

output
// 647

PD (647)
print 7
PD(64)
print 4
PD(6)
print 6
PD(0)

Recursion call kar aata cout.

when n = 0647 ans = 423

~~when a integer~~

Lecture Recursion Level 3

ex i/p: array
o/p: is sorted or not sorted
True false

10	20	30	32	50

```
bool isSortedarray (int arr[], int i, int n)
{
    if (i == n-1)        // terminating condition
    {
        return true;
    }

    // solve for 1
    if( arr[i] > arr[i+1] && i+1 < n)
    {
        false
    return ~~isSorted array (arr[], i+1, n)~~;
    }  isSorted array (arr, i+1, n);
    ~~else {~~
        ~~return false;~~
    }
}
```

ex Binary Search.

```
int binarysearch (vector<int> v, int s, int e, int key){
    // base case
    // case1 key not found
    if (s>e)
        return -1;

    int mid = (s+e)/2;
    // case 2 key found
    if( arr[mid] == key)
        return mid;
    // right map search
```

expression ko by reference में नहीं डाल सकते।


```
if (arr[mid] < key) {
    int ~~bool~~ ans = binarysearch (v, mid+1, e, key);
    return ans;
}

else {
    ~~bool~~ int ans = binary search (v, s, mid-1, key);
    return ans;
}
```

ex Subsequences of a string

I/p : "abc"
O/p : print all subsequences

a	b	c	
✓	✓	✓	abc
✓	✓	✗	ab
✓	✗	✗	a
✗	✗	✗	—
✗	✓	✗	b
✗	✗	✓	c
✗	✓	✓	bc
✓	✗	✓	ac

8

abc
ab
a
b
c
bc
ac

7 subsequences

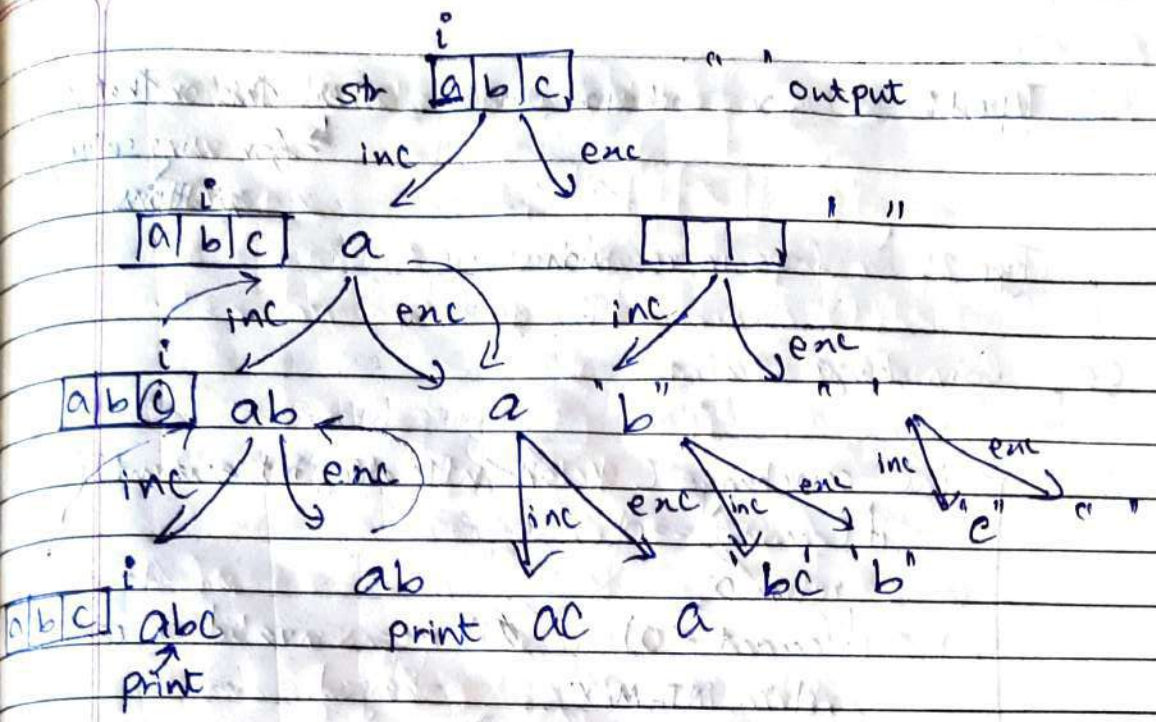
include exclude pattern
"abc"

Arithmatic operation on strings

```
S1 = "Ujjawal";
S2 = "123";
cout << "Ujjwal" + "123";
```
↳ ERROR

```
S1 = S1 + S2;
cout << S1;
```
Ujjawal123

-, *, /, % → not VALID STRINGS




```
void printsubsequences (string str, string output, int i){
    //base case
    if (i>=str.length()){
        cout<< output <<endl;
        return;
    }

    //exclude
    printsubsequences (str, output, i+1);

    // include
        output.push_back (str[i]); //concatination
        printSubsequences (str, output, i+1);
}
```

Also Try with bit masking

```
vector <int> a;
a[0]=1;
error
```

while accessing element use size at the time of declaration

```
vector <int> a(10);
```

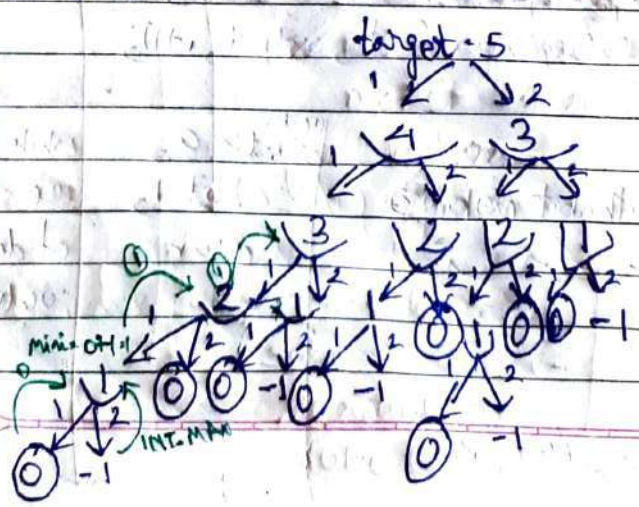
Wild card pattern matching
Regular exp. maching
smallest range in list
median in a string

Type 1: inclusion exclusion pattern, either this or that.
↳ for using subsequence questions

Type 2: for loops in recursion

ex Coin change

```
int coinchange (vector<int> arr, int amount){
    if (amount == 0)
        return 0;
    if (amount < 0)        // it means invalid or can't be
        return INT_MAX;        changed.
    int mini = INT_MAX;

    for (int i = 0; i < arr.size(); i++){
        mini = min( mini, coinchange (arr, target - arr[i]));
    }

    // now min coin required has found

    if (mini == INT_MAX)
        return INT_MAX; // means invalid or can't be
                           changed
    else
        return mini + 1;
}
```



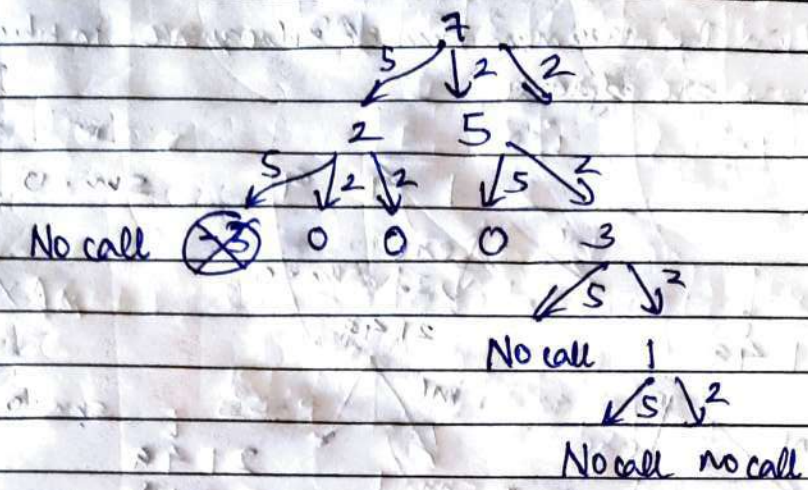
1, 2

{1, 1, 1, 1, 1}
{2, 1, 1, 1}
{1, 2, 1, 1}
{1, 1, 2, 1}
{2, 2, 1}
{1, 1, 1, 2}
{2, 1, 2}
{1, 2, 2}

minimum no. of elements to reach target

ex Cut into segments N x y z segment

Find the max length of segments




```
int solve (int n, int y, int z, int x){
    //base case
    if (n == 0){
        return 0;
    }

    if (n < 0){
        return INT_MIN;
    }

    if (n-x) >= 0
        int ans = solve (n-x, x, y, z) + 1;
    if (n-y) >= 0
        int ans2 = solve (n-y, x, y, z) + 1;
    if (n-z >= 0
        int ans3 = solve (n-z, x, y, z) + 1;

    int ans = max(ans1, max (ans2, ans3));
    return ans;
}
```

n = 7
x = 5
y = 2
z = 2

```
int main()
int ans = solve(n, x, y, z)
    if (ans < 0)
        ans = 0;
}
```

Later there are ∞ coins or notes given inclusion exclusion will not be used.

House Robbery-I

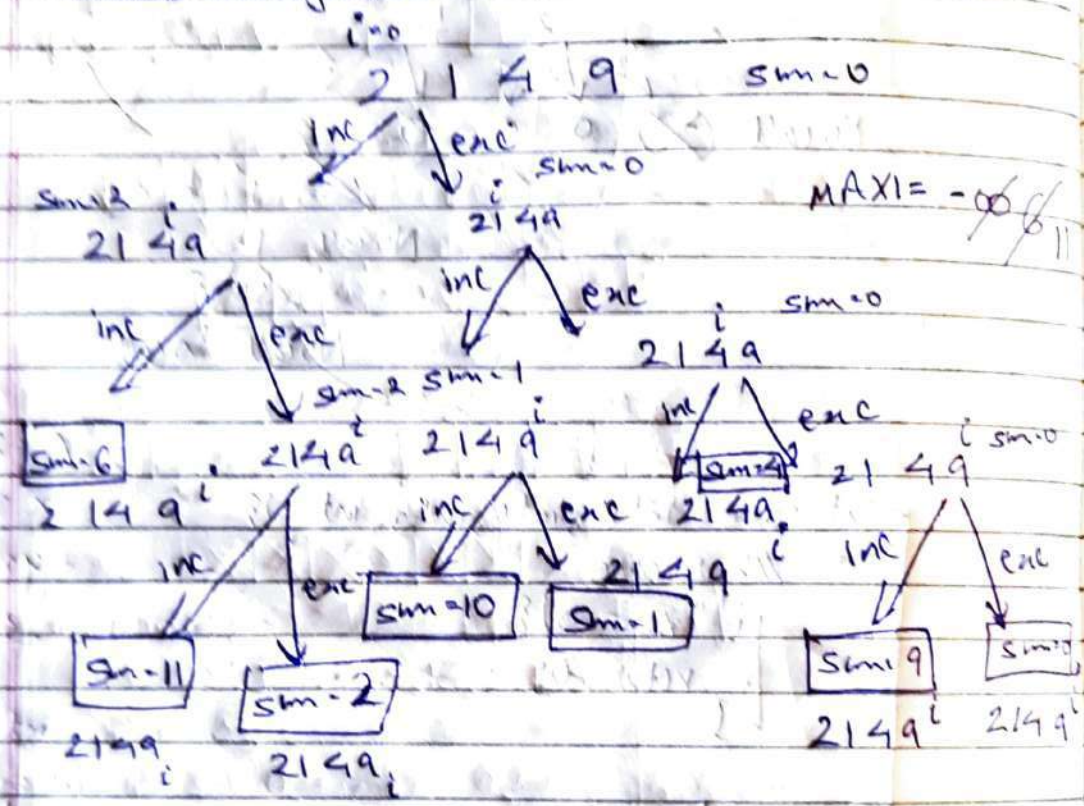
Page No. 96

House Robbery

Q: Max sum of non-adjacent elements

i/p [2 | 1 | 4 | 9] 2+9=11

Return the maximum sum of subsequence in which no two elements are adjacent




```
void solve (vector<int> & arr, int sum, int & maxi, int i)
    // base case
    if (i >= arr.size()){
        // maxi update
        maxi = max (sum, maxi);
        return;
    }

    // include
    solve (arr, i+2, sum+arr[i], maxi);
    // exclude
    solve (arr, i+1, sum, maxi);
}
```

Merge Sort.

A) mid find

B) break 2 half

C) Recursion 2 half → sort.

D) 2 Half → merge

7	3	2	16	24	4	11	9

sorted sorted

sorted array

ex merge 2 sorted arrays

i/p Array1 2 4 6

Array2 3 5 7 9 11

2 pointer approach

2	3	4	5	6	9	11	

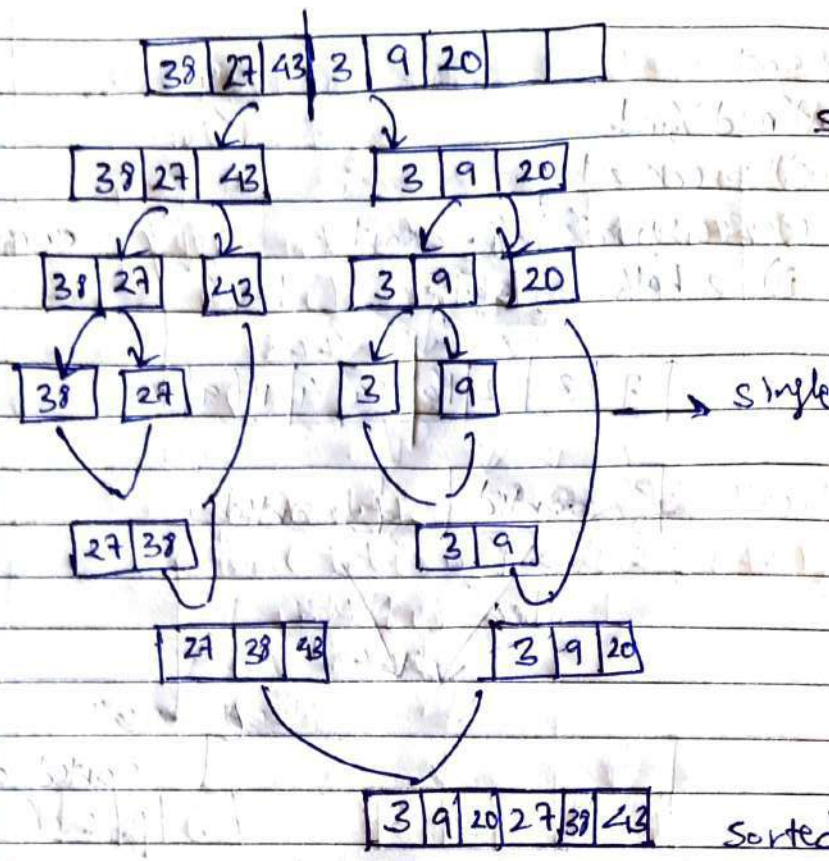
while (i j)

{

}

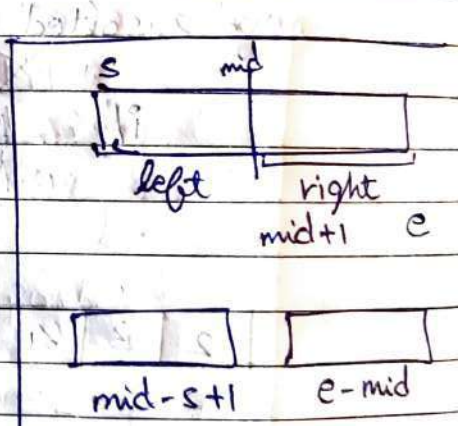
arr1 → remaining element copy

arr2 → remaining element copy

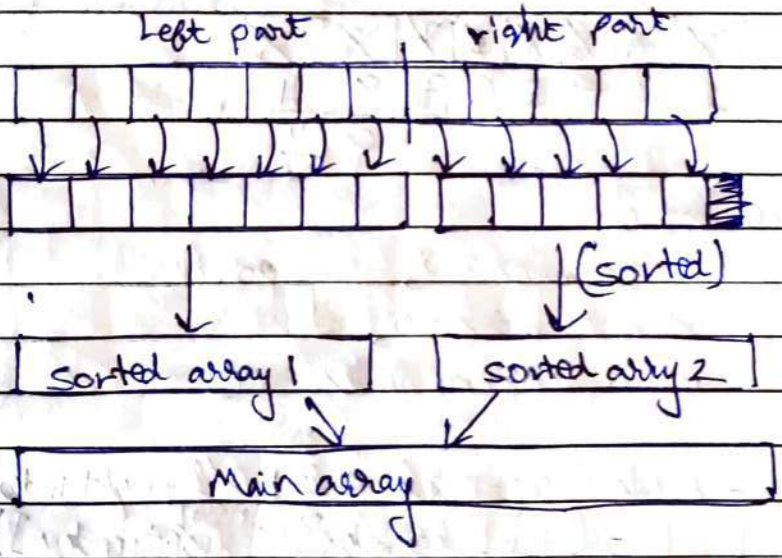


26:43

```
int main(){
    int arr[] = {4, 5, 13, 2, 12}
    int n = 5;
    int s = 0;
    int e = n-1;
    mergesort (arr, s, e);

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

    return 0;
}
```



```
void mergeSort (int* arr, int s, int e){

    // base case
    if (s>=e)
        return;

    int mid = (s+e)/2;
    // left part sorted array
        mergeSort (arr, s, mid);
    // right part sorted array
        mergeSort (arr, mid+1, e);

    // now merge two sorted arrays.
        merge (arr, s, e);
}
```




```
void merge (int* arr, int s, int e){
    int mid = (s+e)/2;
    int len1 = mid - s + 1;
    int len2 = e - mid;

    // assume to create for len1, len2 length
    int* left = new int [len1];
    int* right = new int [len2];
```

left [len1] right [len2]

```
// copy value
int k = s;

for (int i = 0; i < len1; i++){
    left[i] = arr[k];
    k++;
}

k = mid+1;
for (int i = 0; i < len2; i++){
    right[i] = arr[k];
    k++;
}
```

2	4	6	3	5	7	9	11

~~2~~	~~4~~	~~6~~	~~3~~	~~5~~	~~7~~	~~9~~	~~11~~
2	3	4	5	6	7	9	11

copy

```
// merge logic
int leftindex = 0;
int rightindex = 0;
int mainArray index = s;

while (leftindex < len1 && rightindex < leg2){
    if (left[leftindex] < right[rightindex]){
        arr[mainArrayIndex++] = left[leftIndex++];
    }
    else{
        arr[mainArrayIndex++] = right[rightIndex++];
    }
}
```

pointers

Q: Inversion cont


```
// copy logic for left array
while (leftindex < len1){
    arr[mainArrayIndex++] = left[leftindex++];
}

// copy logic for right array
while (rightindex < len2){
    arr[mainArrayIndex++] = right[rightindex++];
}
// also delete the allocated space
```

T.C. avg. $\boxed{n \log n}$

Merge sort without extra space
inplace merge sort

Merge sort {
- BC
- left → RC
- right → RC
- Merge Call

}

merge call {

copy arr1	$O(len1)$	linear time
copy arr2	$O(len2)$	
sort	$O(n)$	
left copy	$O(len1)$	
right copy	$O(len2)$	

}

$$T(n) = \underbrace{k_1}_{BC} + \underbrace{T\left(\frac{n}{2}\right)}_{Right} + \underbrace{T\left(\frac{n}{2}\right)}_{Left} + \underbrace{T(\text{merge})}_{kn}$$

ignore ← $k_1 + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + nk$

$$= 2T\left(\frac{n}{2}\right) + nk$$

$T(n) = 2T(\frac{n}{2}) + n*k$

$\left[T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{2}*k\right] 2$

$\left[T(\frac{n}{4}) = 2T(\frac{n}{8}) + \frac{n}{4}*k\right] 4$

$T(1) = k_1$

$T(n) = k_1 + (a-1)(n*k$

$n \to n/2 \to n/4 \to \dots \to 1$

$\Rightarrow a = \log_2 n$

$T(n) = (k_1) + n(k)(\log n - 1)$

ignored ignored

$\underbrace{k_1}_{\text{ignored}} + nk\log n - nk$

$n(\log n - 1)$

$= n\log n - (n) \to$ ignored

$= \boxed{n \log n}$

- Last occurance
- Reverse string
- Add 2 strings (int stored as strings)
- Palindrome check
- Subarray Printing
- Wild card pattern matching
- House Robbery Problem
- Buy and sell stocks
- Recursively remove all occurance of a character in a string.
- integer to english words
- No. of digit One
- Perfect Square Problem
- Min cost for tickets
- No. of dice roll with target sum

DCN Level 2. Quick Sort.

I/P array

9	3	4	1	20	50	30
0	1	2	3	4	5	6

Steps

Pivot element ↳ 1st no. को सही जगह Pahuchadae. [Partison logic]

बाकी recursion संभाल लेगा।

Randomise QuickSort.

QS
↳ Partision logic
Recursive logic

1	3	4	8	20	50	30

less than 8 | more than 8

8	1	3	4	20	50	30

n=7

A) pivot 8
B) pivot → right place, count small $\boxed{count = 3}$
swap (arr (pivot index), arr [count])

C)

4	1	3	8	20	50	30

i, j

D)

Recursive call
→ left

4	1	3
0	1	2
s		e

↳ right

20	50	30
4	5	6
s		e

quickSort
BC
pivot ele
quickSort(s, p-1);
quickSort(p+1, e);

partision
pivot index
swap (ps, right index)
C pivot B segregation
swap()

```
~~int main~~ void quickSort( int arr[], int s, e){
  //base case
    if( s>=e)
      return;

  // partision logic return pivot element
    int p = partision (arr, s, e);

    quickSort ( arr, s, p-1);
    quickSort (arr, p+1, e);
}
```

```
int partision( int arr[], int s, int e){
    int pivotindex = s;
    int pivotElement = arr[s];

        int count = 0;
        for( int i = s+1; i<=e; i++){
            if (arr[i] <= pivotelement.){
                count++;
            }
        }
        // step2 right position for pivot element
        int rightindex = s+ count;
            swap (arr[pivotindex], arr[rightindex]);
            pivotindex = rightindex;

        step3 // chotae pivot badae
        int i = s;
        int j = e;

        while (i< pivotindex && j > pivot index){
            while (arr[i] <=pivotelement){
                i++
            }

            while (arr[j] > pivotelement){
                j--;
            }

    // 2 case ho sakhtae hai.
    // A) no elements found to swap
    // B) no need to swap
        if (i< pivotindex && j >pivotindex){
            swap (arr[i], arr[j]);
        }
    }
    return pivotindex;
}
```

TC : best
avg: $n \log n$
worst: n^2

partisioning logic: $O(2n)$
$O(n)$ linear

and we need to partision it $\log n$ times so complexity is $n \log n$

worst case

7	6	5	4	3	2	1

1	6	5	4	3	2

6	5	4	3	2

5	4	3	2

4	3	2

3	2

n times

2

$T(c) = n^2$

Special case of recursion

Backtracking (specific form of recursion)

The solution that we have discarded, we will not go again to look or check it again.

But हाँ TC के solutions हम backtracking से करते हैं।

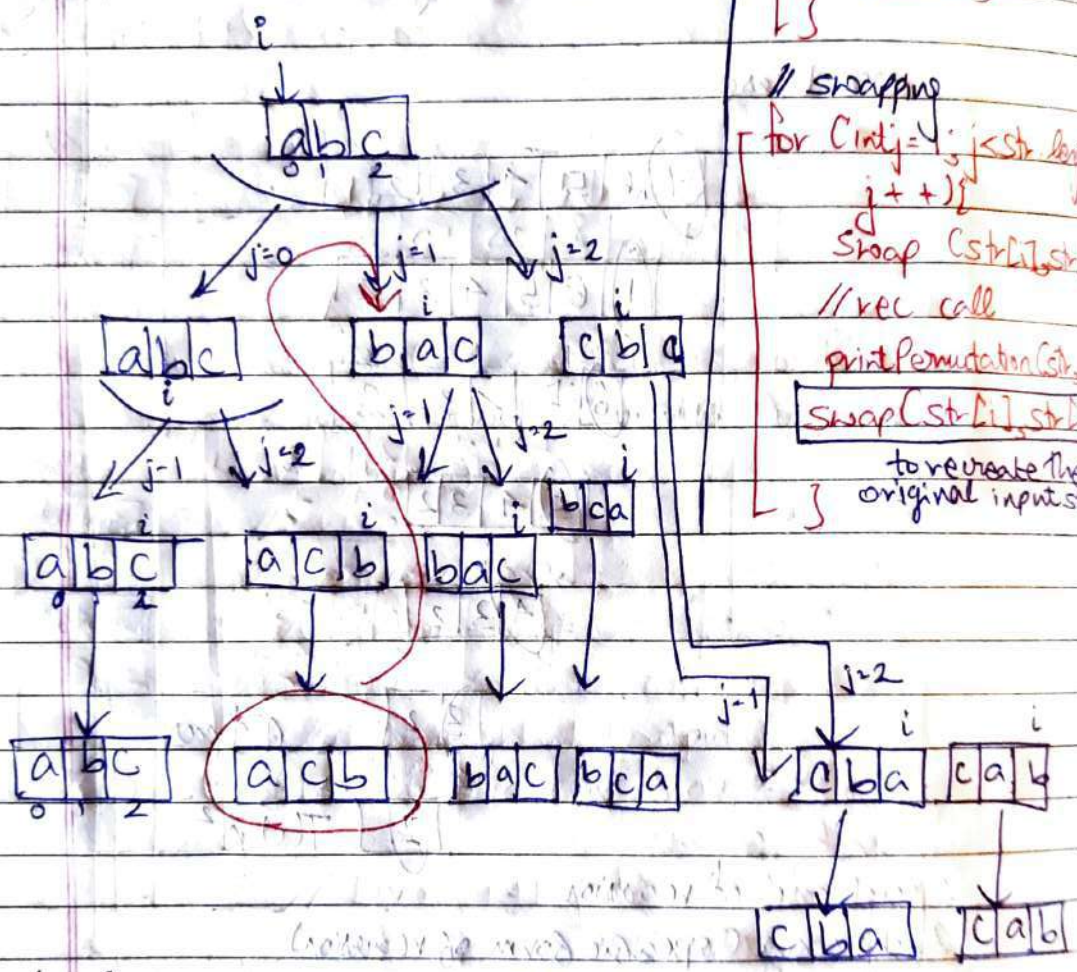
b) Permutation in a string

abc

abc acb
bac
bca
cab
cba

```
int main (){
    string str = "abc";
    int i=0;
    printpermutation (str, i);
    return 0;
}
```

```
void printpermutation(
    string &str, int i){

    // base case
    if(i>=str.length()){
        cout<<str<<" "
        return;
    }

    // swapping
    for (int j=i; j<str.length();
        j++){
        swap (str[i], str[j])
        //rec call
        printPermutation(str, i+1)
        swap(str[i], str[j])
        to recreate the
        original input string
    }
```



↳ Rat in a Maze

D L R U

1	0	0	0	0
1	1	0	0	0
0	1	0	0	0
0	1	1	1	0
0	0	1	1	1

Safe move for rat

1. [1]
2. Indexing side array
3. Not visiting cell again

D L R U
D = (i+1, j) R = (i, j+1)
L = (i, j-1) U = (i-1, j)

```
#include <iostream>
#include <vector>

using namespace std;

int main(){
    int maze[4][4] = { {1, 0, 0, 0},
                       {1, 1, 0, 1},
                       {1, 1, 0, 0},
                       {0, 1, 1, 1}};
    if (maze[0][0] = 0){
        cout << "No path exists" << endl;
        return 0;
    }

    int row = 4;
    int col = 4;

    vector < vector <bool> >visited (row, vector <bool>
      (col, false));
    visited[0][0] = true;
    vector <string> path;
      string output = " ";

    solveMaze (maze, row, col, 0, 0, visited, path, output);
    cout << "printing the results" << endl;
    for (int auto i : path){
        cout << i << " ";
    }

    cout << endl;
    if (path.size() == 0)
      cout << "No path exists" << endl;
    return 0;
}
```

```
bool isSafe (int i, int j, int row, int col, int arr[][4],
        vector <vector <bool>> &visited) {

    if ((( i>=0 && i<row) && (j>=0 && j<col)) &&
        (arr[i][j] ==1) &&
        (visited[i][j] = false)) {
        return true;
    }
    else {
        return false;
    }
}

void solveMaze (int arr[][4], int row, int col, int i, int j,
vector < vector <bool>> &visited, vector <string> &path,
    string output) {

    // base case
    if (i==row-1 && j= col-1) {
        // answer found
        path. pushback (output);
        return;
    }

    // Down i+1, j

    if (isSafe (i+1, j, row, col, arr, visited)) {
        visited [i+1][j]= true;
        solveMaze (arr, row, col, i+1, j, visited, path,
            output + 'D');
        // backtrack
        visited [i+1][j]= false;
    }
```

```
// left -> i  j-1

if( isSafe (i, j-1, row, col, arr, visited)) {
    visited[i][j-1] = true;
    solveMaze (arr, row, col, i, j-1, visited, path, output+'L');
    // backtrack
    visited [i][j-1] = false;
}

// right   i -> j+1

if( isSafe (i, j+1, row, col, arr, visited)) {
    visited[i][j+1] = true;
    solveMaze (arr, row, col, i, j+1, visited, path, output+'R');
    // backtrack
    visited [i][j+1] = false;
}

// up   i-1  j

if( isSafe (i-1, j, row, col, arr, visited)) {
    visited [i-1][j] = true;
    solveMaze (arr, row, col, i-1, j, visited, path, output+'U');
    // backtrack
    visited [i-1][j] = false;
}

}
```

Letter tile combination
No. of square full arrays
world break - 1
Sum of all subset XOR

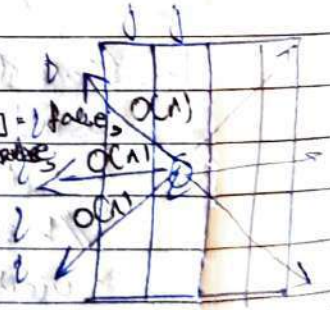
storing

imp

n queen problem.

```
void solve (vector<vector<int>> board,
            int col, int n) {

    // base case
    if (col >= n) {
        printSolution (board, n);
        return;
    }

    // 1 case solve करना है बाकी recursion
    sambhal लेगा

    for (int row=0; row<n; row++) {
        if (isSafe (row, col, board, n)) {
            // rakh do
            board[row][col] = 1;
            // recursion sol^n laega
            solve (board, col+1, n);
            // backtracking
            board[row][col] = 0;
            rowcheck[row] = false;
            upper left diagonal [n+col+row] = false;
            lower left diagonal [row+col] = false;
        }
    }
}
```

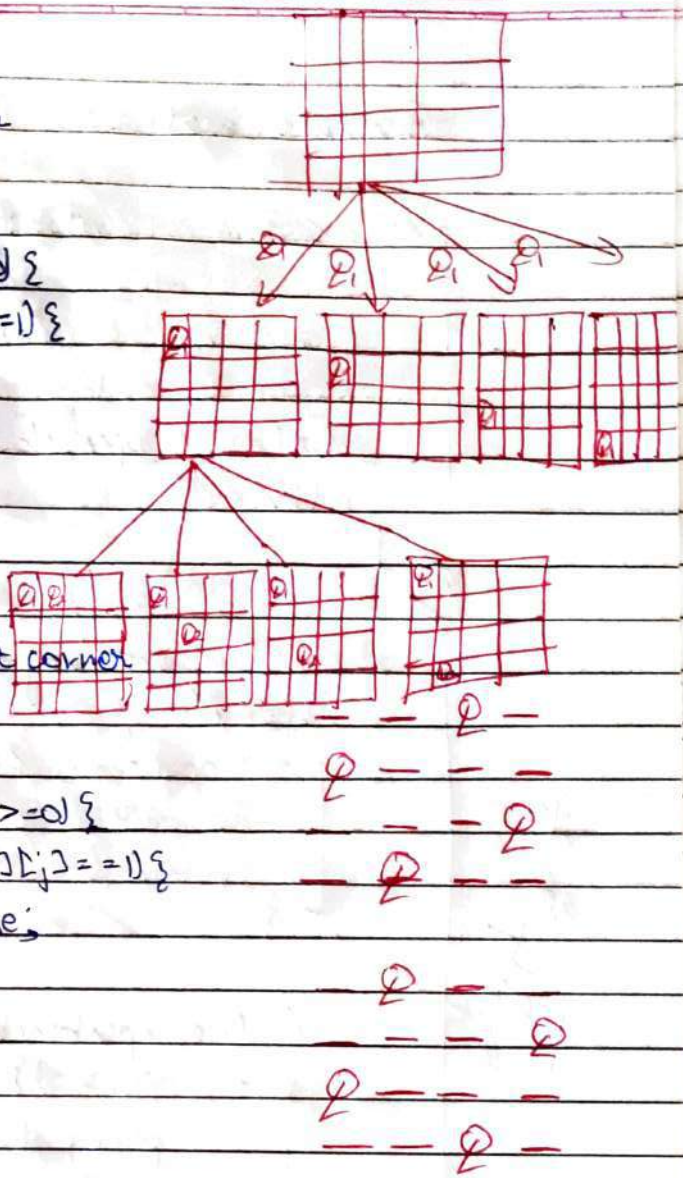
```
#include <iostream>
#include <vector>
using namespace std;
int main(){
    int n=4;
    vector<vector<int>> board;
    int col = 0;
    solve (board, col, n);
unordered map<int, bool> rowcheck;
unordered map<int, bool> upper left;
unordered map<int, bool> bottom left;

void printSolution (vector
vector<int>> board, int
{
    for (int i=0; i<n; i++){
        for (int j=0; j<n; j++){
            cout << board[i][j]
        }
    }
    rowcheck[row] = true
    upper left diagonal [n+col-row] = true
    lower left diagonal [row+col] = true;
    cout << endl
```




```
bool isSafe (int row, int col, vector<vector<int>> board, int n){
    // क्या हम current cell पर queen rakh सकता है या नहीं।
    int i = row;
    int j = col;
    // check row
    while (j>=0) {
        if (board[i][j] == 1) {
            return false;
        }
        j--;
    }
```

O(1)

```
    if (rowcheck[row]
    { return false;
    if (upper left[n+col-row]
    { return false; }
    if (bottom [row+col]
    { return false; }
    return true;
```



```
// check upper left corner
    i = row;
    j = col;
    while (i>=0 && j>=0) {
        if (board[i][j]==1) {
            return false;
        }
        i--;
        j--;
    }
// check for lower left corner
    i = row;
    j = col;
    while (i<n && j>=0) {
        if (board[i][j]==1) {
            return false;
        }
        i++;
        j--;
    }

// Kahi par bhi queen नहीं मिली
// iska मतलब यह Position safe है
// hence return true;
    return true;
}
```

↳ Maps key - value pairs.

unordered
map <string, int> m;
↳ insert m["love"] = 98
m["Rahul"] = 36

cout << m["love"];
⇓
98

solve(ans, 3, 3, 3) → solve(ans, 3, 2, 3, "(") → solve(ans, 3, 1, 3, "(") → solve(ans, 3, 0, 3, "(")

112

-Generate Parothesis (()()) (()()) ((()) (()()) (()()

```
vector <string> generateParothesis(int n){
  vector <string ans;
  int open=n;
  int close=n;
  solve(ans, n, open, close);
  return ans;
}
```

```
void solve(vector<string> &ans, int n, int open, int close){
    //base case
    if (open==0 && close==0){
        ans.push back(output);
        return;
    }

    //include open bracket
    if (open>0){
        output.push-back('(');
        solve(ans, n, open-1, close, output)
        //backtrack
        output.pop-back();
    }

    //include close bracket
    if (close > open){
        output.push back(')');
        solve (ans, n, open, close-1, output)
        //backtrack
        output.pop-back();
    }
}
```

go at last page

output

[(] open-1 close-0

open-1 close-2 [((]

[()] open-1 close-1

open-0 close-3 [(((]

[(()] open-0 close-2

[()()] open-0 close-1

[(()(] open-0 close-1

[(()()] open-0 close-0

open-0 close-1 [((())]

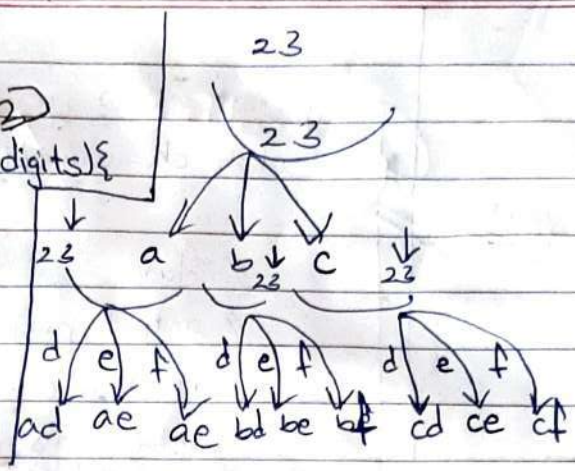
open-1 close-0 [(((()))]

solve (ans, 3, 0, 2, "(((")
↓
solve (ans, 3, 0, 1, "((())")
↓
solve (ans, 3, 0, 0, "(((()))")

output: "(((()))"

Phone Keypad

[23]




```
vector<string> letterCombination(String digits){
    vector<string> ans;

    if(digits.length==0)
        return ans;

    int index=0;
    string output="";
    vector<string> mapping(10);
    mapping[2] = "abc";
    mapping[3] = "def";
    mapping[4] = "ghi";
    mapping[5] = "jkl";
    mapping[6] = "mno";
    mapping[7] = "pqrs";
    mapping[8] = "tuv";
    mapping[9] = "wxyz";
    solve(ans, index, output, digits, mapping);
    return ans;
}

void solve(vector<string>& ans, int index, string output, string digits,
        vector<string>& mapping){
    //base case
    if (index ≥ digits.length()){
        ans.push_back(output);
        return;
    }

    //1 case solve, baki recursion
    char digitCharacter = digits[index];
    int digitIndex = digitCharacter - '0';
    string value = mapping[digitIndex];
```

```
for (int i= 0; i<value.length(); i++) {
    char ch = value[i];

    //include
    // output.push_back(ch);
    // recursive call
    solve (ans, index+1, output+ ch, digits, mapping)
    // backtrack
    // output.pop_back();
}
}
```

There should be atleast 17 clues for making a sudoku a UNIQUE SOLUTION.

ex Sudoku Solver

```
void solveSudoku (vector<vector<char>>& board) {
    solve (board);
}

bool solve (vector<vector<char>>& board) {
    int n= board.size();

    for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
            // check for empty cell
            if (board[i][j]== '.') {
            // try to fill the values ranging from 1 to 9.
                for (char val='1'; val <='9'; val++) {
                    // check for safety
                    if (isSafe [i, j, board, val]) {
                        //insert
                        board[i][j] = val;
                        //recursion sambhal Lagga
                        bool remainingBoardSolution= solve (board
```

```
            if(remainingBoardSolution == true) {
                return true;
            }

            else {
                //backtrack
                board[i][j] = '.';
            }
        }
    }

    // if 1 से 9 तक koi भी value से solution
    // नहीं nikla, current cell par, That means
    // piche kahin pr galti h,
        //go back by returning false
    return false;
    }
  }
}
// all cells filled
return true;
}

bool isSafe (int row, int col, vector<vector<char>>&board, char value){
int n = board.size();
for (int i=0; i<n; i++) {
    //row check
    if (board[row][i] == value)
        return false

    //col check
    if (board[i][col] == value)
        return false;
    // 3 X 3 box check
    if (board[3*(row/3) + (i/3)][3*(col/3) + (i%3)]
                                    == value)
        return false;
}
return true;
```

"Is a" relationship

size of empty class in C++ is 1 byte

```
struct Example {
    char a;   // 4
    int b;    // 4
    char c;   // 4
};            // 12
```

Padding

1) Padding is done because alignment datatypes can be faster than misaligned datatypes

116

↑ static
Animal ramesh;
cout << ramesh.y;

↳ OOps Lect 3

POINTER | OBJECT

upcasting

```
Parent *a = new Parent();
Parent *a = new Child();   (late binding)
Child *a = new Child();
Child *a = (Child*) new Child();
```

with virtual keyword

if not initialised it will lead to garbage

default normal case.
without virtual keyword

✓ Animal *a = new Animal();
animal constructor

static variables jaha tak scope hota hoga vaha tak constructor called hoga.

Animal *a = new Dog(); Constructor dependence
animal constructor
Dog constructor

main mae accessible nahi ho aur inherit ho jai to PROTECTED

✓ Dog *a = new Dog();
animal constructor
Dog constructor

Dog *a = (Dog *) new Animal();
animal constructor

↳ Abstraction [Impleting hiding] (A group of many thing)

Abstraction

Encapsulation: Data hiding
Abstraction: Implementation hiding

Abstraction → essential info show
Ex → key → car start

Encapsulation → wrap (DM / MF)
↳ access modifiers
↳ private

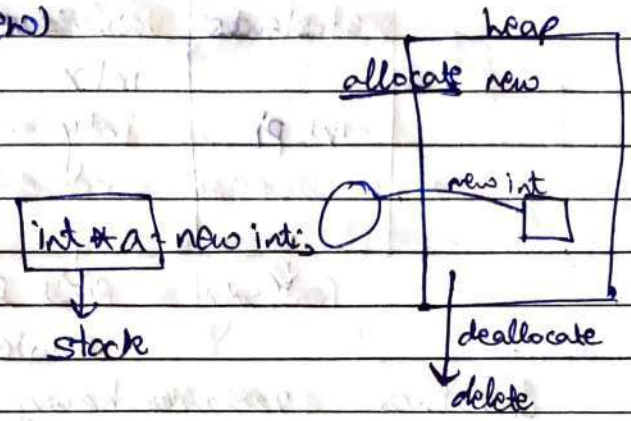
when to use inheritance.
- Use inheritance whenever is-A relationship is identified between objects
- Inheritance is applied where objects exhibit a hierarchical relationship, signifying a specific specialized - generalised connection.
- Do not reinvent the wheel.

Page No.
Date : 117

Dynamic Memory Allocation

→ stack memory (local variables)
(function parameters)
CHOOTI (default) but we can change
int a = 5;

→ heap BADI (default)
(new)



int arr[n] bad practice
↳ if n = 10 lakhs then code will not work
stack

int * arr = new int[n]; → delete []arr

int** arr = new int* [n];
↳ row

n blocks
int*
int*
⋮
int*

for(i=0; i<n; i++)
{
arr[i] = new int [m];
}
↳ column

arr[i] → pointer
new int [m] → integer array of size 'm'

Using vectors

vector < vector <int>> arr(n, vector <int> (m, 0))

n → row
m → column
0 → init

Const key word [like a promise]
variable, function or object is immutable [its value cannot be changed after initialization]

imp Compiler may be able to store constant variables in read only memory, which can result in faster access time

Constants	Variables
int pi	int x int y
↓ Read only	↓ Read & Write

l values variables having memory location int x;
r values variables doesn't have memory location
5 ; int & a = b;
alias

↳ Const with pointers

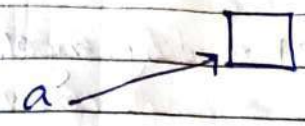
```
Const int x = 5;
int *p = &x;
*p = 10;
cout << x << endl;
```

This used to work in older compilers, but not in modern compilers.

```
int * a = new int;
*a = 2;
cout << *a << endl; // 2
delete a;
int b = 5;
a = &b;
cout << *a << endl;
```

without this there would be memory leak

const int *a = new int;



data constant
content of a pointer constant

pointer itself can be reassigned.

int const *a = new int(2);
OR
const int *a = new int(2); // const data non const pointer

cout << *a; //2
*a = 20; //error
cout << *a; //2

int b = 20;
a = &b;
cout << *a << endl;

↳ constant pointer non constant data

int * const a = new int(2);
*a = 20 //no error
cout << a; 20

cout << *a //2

int b = 50;
a = &b; // error

↳ const pointer constant data

const int * const a = new (10);
cout << *a << endl;
*a = 50; //error;
int b = 100;
a = &b; //error

Default argument are always written from right side

Method const.

This method can't ever change or modify any of the member variable

```
int get_x() const {
    x = 50;      // error.
    return x;
}

void printABC(const abc & a)
{
    // only access const methods
}
```

getters can be made as constants

whereas setters cannot be made as constants

imp

if we want to change x then we can initialise it as mutable int x;

Then this will not generate error

Mutable is only and generally used for DEBUGING.

Old way of making constructor

```
abc (int x, int y, int z=0) {
    x = x;
    y = new int(y);
    z = z;      // if z was const then it would lead to error
}
```

but this does not generate error

Modern way using initialisation list.

```
abc(int x, int y, int z=0) : x(x), y(new int(y)), z(z){}
```

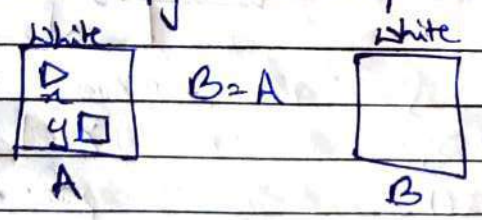
MACROS

In c++ macros are preprocessor derivatives that allow you to define constants, function or code snippits that can be used throughout your code. They are typically defined using the #define derivative and are evaluated by the preprocessor before the code is compiled.

↳ used to define constants
→ creating shortcuts for commonly used expressions.

```
#define PI 3.14159465;

#define MAXX (x, y) (x>y ? x:y)

void fun()
{
    int x=6;
    int y=17;
    int c=  MAXX (x, y);
}
```

↳ Shallow copy v/s deep copy



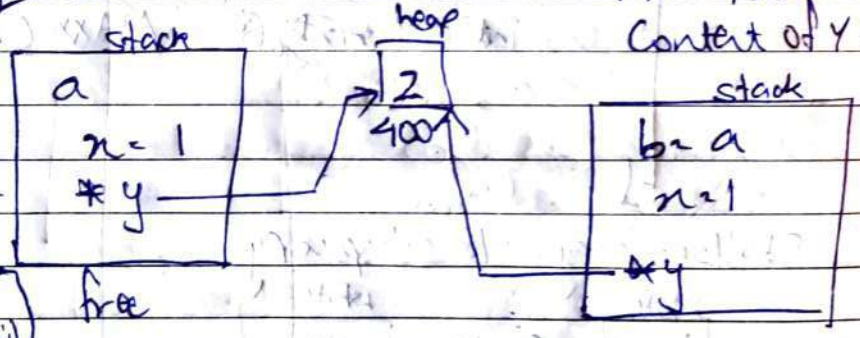
Shallow copy		Deep copy
① White color	White color	
② x y	x y	

```
class abc
{ public
  int x; int *y;
  abc(int _x, int _y): x(_x), y(new int(_y)){}

  void print() const
  {
    printf("x:%d \n ptr:%p \n Content of y:%d", x, y, *y);
  }
};
int main(){
  abc a(1,2);
  a.print();
  abc b=a;
  b.print();
  return 0;
}
```

(b ka copy constructor) By default call hota hai copy constructor

X : 1
ptr = 400
Content of Y : 2

X = 1
Ptr = 400
Content of Y : 2



default dmp copy constructor

abc (const acc&obj)
{
x = obj.x;
y = obj.y;
}

free

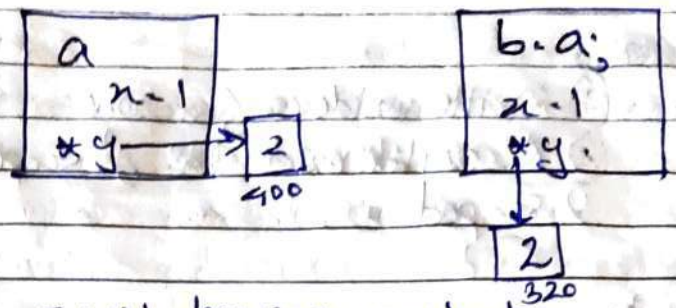
*by = 20;
b.print();

X = 1
ptr = 400
content of y = 20

a.print();

X = 1
ptr = 400
content of y = 20

shallow copy



Deep copy

smart deep copy constructor

Deep copy

```
abc ( const abc & obj)
{
    x = obj.x;
    y = new int (*obj.y);
}
```

smart deepcopy

~~y = obj.y;~~ copy const. ~~shallow copy~~

new code

Destructor

```
~abc(){
    delete y;
}
```

```
abc * a = new abc(1,2);
abc b = *a;
delete a;
b.print();
```

automatically deletes.

stack

abc
*a

Heap

x = 1
*y

302
2

abc
b = *a
x = 1
b.y

there is no memory

error: Double free of object at 302.
So deep copy is favourable rather than shallow copy

Q. Local v/s Global variables

Global: written outside a function
accessible to all functions (same copy)
same address

Priority

(1) Local variables:

(1) Written inside a function
(2) Accessible inside that f^n scope only
(3) Scoped

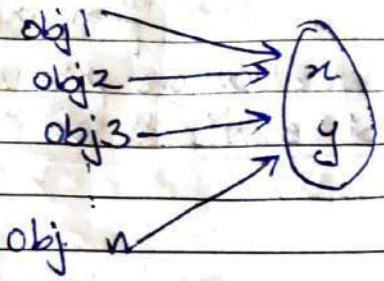
Priority

Local variable > Global Variable

```
cout << ::x // accessing global variable

int x=2;
int main()
{
    ::x=4;
    int x=20;
    cout << x //20
    cout << ::x //4

    { int x=50;
        cout<x;      // 50
        cout << ::x; // 4
    }

}
```

↳ Static Keyword

whenever obj.s are created this pointer is passed to constructer.




```
class abc{
  public:
    static int x,y;
    static void print() const
      {
        cout << x << y;
      }
  };
int abc::x;
int abc::y;
```

static members

They are initialised outside - de int main function

```
int main(){
  abc obj1;
  obj1.x=1;
  obj1.y=2;
  obj1.print(); // 1 2
  abc obj2;
  obj2.x=10;   or abc::x=10;
  obj2.y=20;
  obj1.print(); // 10 20
  obj2.print(); // 10 20
  abc::print();
```

no this pointer access

if added static

a static function can only access static data members.

Static member function: There is no instance of that class being passed into that method.

Static data members: That ~~members~~ variables is going to share memory with all of the class instances

↳ Abstraction

Delivering only essential information to the outerworld while masking the background details

It seperates the inter faces from implementation.

Abstraction in header files
Abstraction in in c++ getters and setters:

access specifiers
public
private
protected → interface

Abstraction divides the code into 2 categories interface & implementation. Somehow creating your component, keep the interface seperate from the implementation so that if the underlying implementation changes, the interface stays the same.

Abstract Class:
These classes that contain atleast one pure virtual function and these classes cannot be instantiated.

```
class Bird
{
    public:
        virtual void eat()=0;
        virtual void fly()=0;
        //classes that inherit this class
            has to implement pure virtual function
};

class sparrow : public Bird
{
    private: void eat() {
                cout << "Sparrow is eating";
             }

             void fly() {
                cout << "Sparrow is flying";
             }
};

class eagle : public Bird
{
    private: void eat() {
                cout << "Eagle is eating";
             }

             void fly() {
                cout << "Eagle is flying";
             }
}
```

In this instance any prog. that uses interface would remain unaffected and would require recompilation with the most recent implement-ation

define at Runtime

इसको Bird के Pointer से ही Call करना पड़ेगा। बाकी नहीं चलेगा

```
int main(){
    Bird *bird = new Pegion()
        bird.dosomething(bird);

    Bird *bird2 = new Bird(); ✗ //error
```

Interface का object नहीं बन सकता

```
    bird.dosomething(Bird *& bird){
        bird -> eat();      // pegion is eating
        bird -> fly();      // pegion is flying
        bird -> eat();      // pegion is eating
    }
```

At compile time

modern compilers are optimised if in inline function exceed certain threshold it removes the inline function

inline function

keyword

A inline function is a regular function that is defined by the inline

The code for an inline function is inserted directly into the code of the calling function by compiler while compiling which can result in faster execution and less overhead compared to regular function calls.

Similar to Preprocessor but this happens at compile time and the preprocessor happens before compilation

Instead of calling functions the statements of functions are pasted in calling functions

Used with small sized functions so that executables are small (handled automatically by compiler optimisation levels)

```
inline void numbershow(int num)
{
    cout << num;
}
```

```
int main(){
    numbershow(10);
    numbershow(10);
    numbershow(10);
    numbershow(10);
    :
}
```

n times

does not prefer in industries

Friend keyword in C++

Friend keyword in C++ that's used to share the information of a class that was previously hidden.

i.e. The private member function are hidden from every other class and cannot be modified except through getters and setters. Similarly the protected members are hidden from all classes other than its children classes.

```
class A {
    private: int x;
    public:
        A (int-value) : x (-val) {};
        int getX() const {return x;}
        void setX (int -val) {x = val;}
    };
    friend class B;
    also
    friend void print(const A&);
```

only declaration

```
class B {
    public:
    void print (const A&a)
    {
        cout << a.getX() << endl;
    }
};
```

① error to resolve

इस method में हम A के object को change नहीं कर सकते। C++ के Compiler को लगता है की getX में change होगा

```
cout << a.x;
```

② error

```
int main()
{
    A a(5);
    B b;
    b.print(a);
    return 0;
}
```

Although information hiding is encouraged because it prevents unwanted bugs in program, there are classes where two classes need to frequently interact with each other.

In such scenerios, one of the classes declares another to be its friend.

The class declaring the friendship can now make all of its data available to its friend.

ctor → constructor
dtor → destructor

Singleton class we can create only 1 instance of a class

```
class A
{
  static A obj = new A();
  Private A()
  {
  }
  public static A get()
  {
    return obj;
  }
}
```

↳ Can constructor be made private?

Yes

par kaise initialise nahi paraega

```
class Box {
    int wi dth
    //ctor
    Box (int-width): width(-width){};
    public:
        int getWidth() const
        {
            return width;
        }

        void setWidth(int-val){
            width=val};
        }
        friend class BoxFactory;
};
```

Private

Learning Resource
Microsoft
Constructors

```
class BoxFactory {
    int count;
    public:
        Box getABox(int-w)
        {
            ++count;
            return Box(-w);
        }
};
```

Box class bolaega ki Mujhe na BoxFactory vali class hi initialise कर सकती है। इसके बुलाता कोई नहीं।

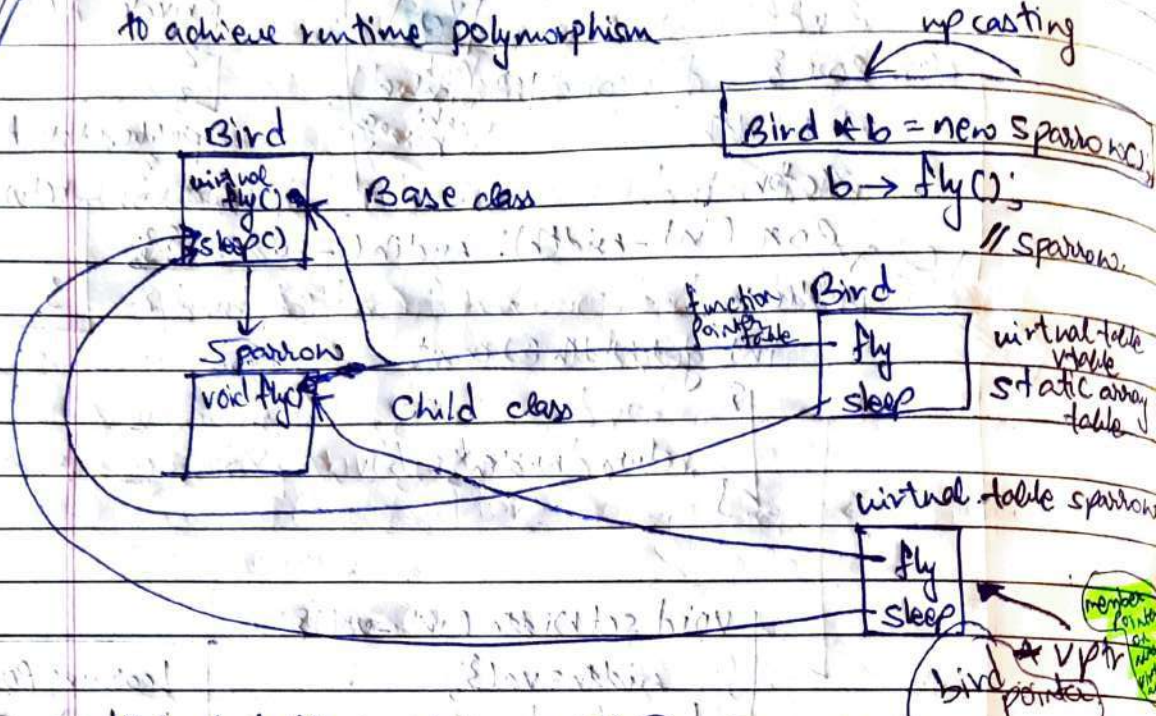
```
int main() { //now object can be made from BoxFactory class only
    BoxFactory bfact;
    Box b = bfact.getABox(5);
    cout << b.getWidth();
}
```

Bird *b = new Sparrow();
b → vptr = sparrow's vTable

Late binding

again

Virtual in c++.
to achieve runtime polymorphism



Virtual table बनती है जिसके अंदर atleast one virtual function हो.

This vptr is made at compile time

vtables

i. Formed for every class having atleast one virtual function and for its derived classes.

ii. It is static arrays, hence one instance for a class

iii. VPtr (a hidden member pointer) is added by compiler to classes with virtual and its derived class.

iv. Depending upon the object type VPtr is bonded to a vTable.

2. VPtr are created at compile time.

3. When object of a particular type is created at runtime, there will be a VPtr which will be initialised to a point to a static VTable at the time of construction.

This is known as LATE BINDING

3

Base *b = new Derived
delete b;
Const. base
const. derived
Base destructor

after
virtual Base()
{ "base dtor"; }
const. base
const derived
derived destructor
~~derived~~ base destructor

Can we make a virtual constructor

- No
- Constructors cannot be virtual, because when constructors of class is executed there is no virtual table in the memory, means no ~~vm~~ virtual pointer defined yet. So the constructor should always be non-virtual. [constructor execute होने के बाद VPtr बनता है]

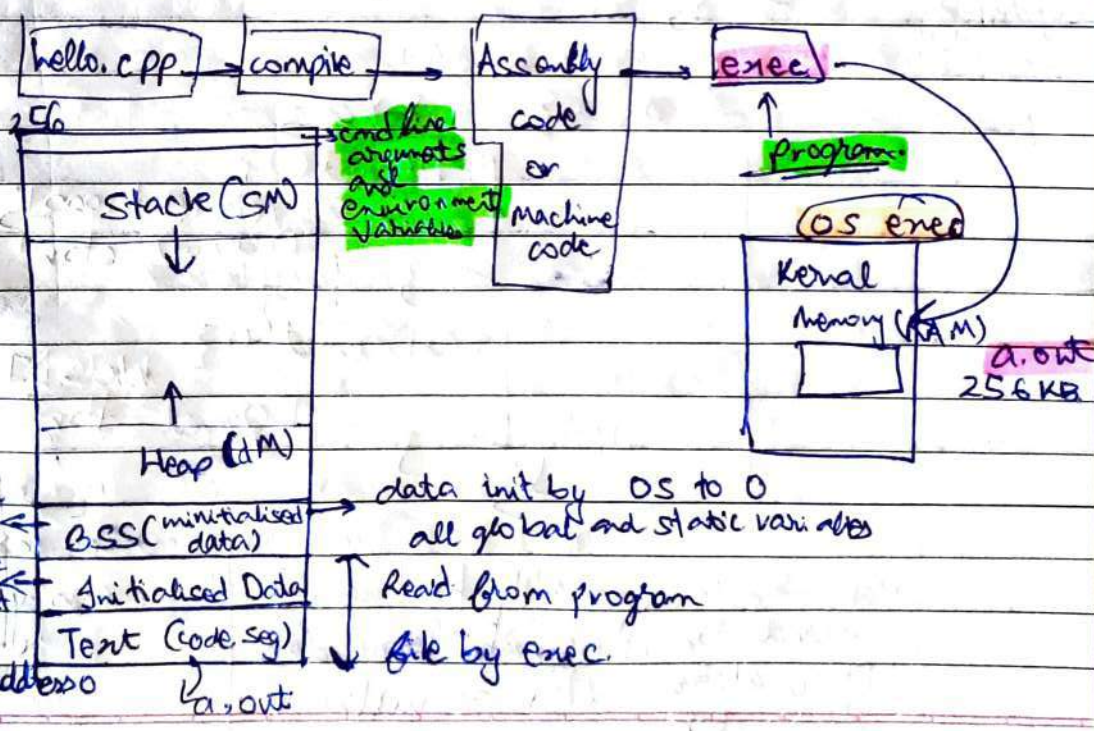
A virtual call is a mechanism to get work done given partial information. In particular, "virtually" allows us to call a function knowing only any interfaces and not the exact type of the object.

To create an object you need 'complete information. In particular, you need to know the exact type of what you want to create. Consequently, a 'call to a constructor' cannot be virtual

Imp

Base के destructor को virtual बना देना।

2> Memory Layout of a Program



hello.cpp

```
int a; // uninitialised data
int b = 1; // initialised data
```

Can we make virtual dtor?

yes

It is important to handle proper destruction of derived class

```
base ctor
Derived ctor
Base dtor   only base dtor is called
```

```
Base {
    Base()
    {
        "Base ctor"
    }
virtual ~Base()
    {
        "Base dtor"
    }
};
```

```
Derived {   int *a;
    Derived() {
        a = new int[1000];
        "Derived ctor"
    }
    ~Derived() {
        delete[] a
        "Derived dtor";
    }
};
```

can't execute

Base ctor
Derived ctor
Derivedctor
Base dtor

can only be executed if we make base class dtor as virtual

hence case of memory leak.

solve (ans, 3, 3, 3)

solve (ans, 3, 2, 3, "(")

solve (ans, 3, 1, 3, "((")

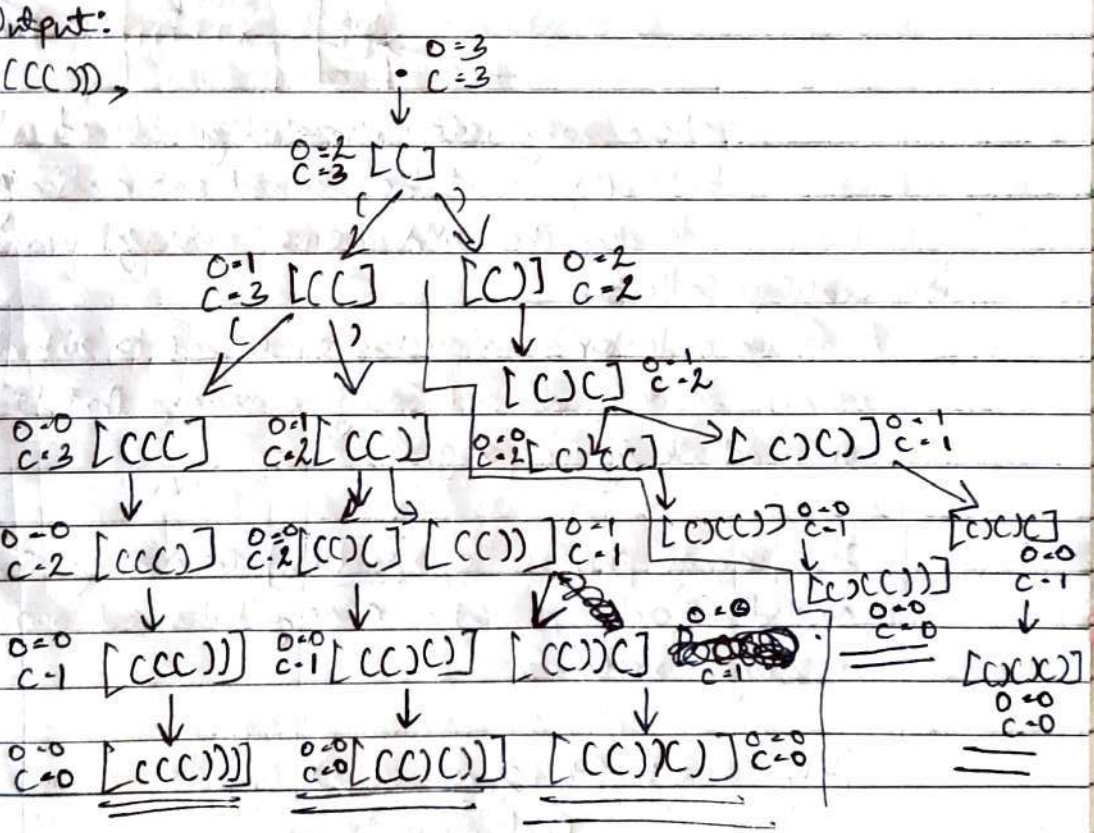
solve (ans 3, 0, 3, "(((")

solve (ans, 3, 0, 2, "(((")")

solve (ans, 3, 0, 1, "((())")

solve (ans 3, 0, 0, "(((solve (ans, 3, 0, 0, "(((()))")

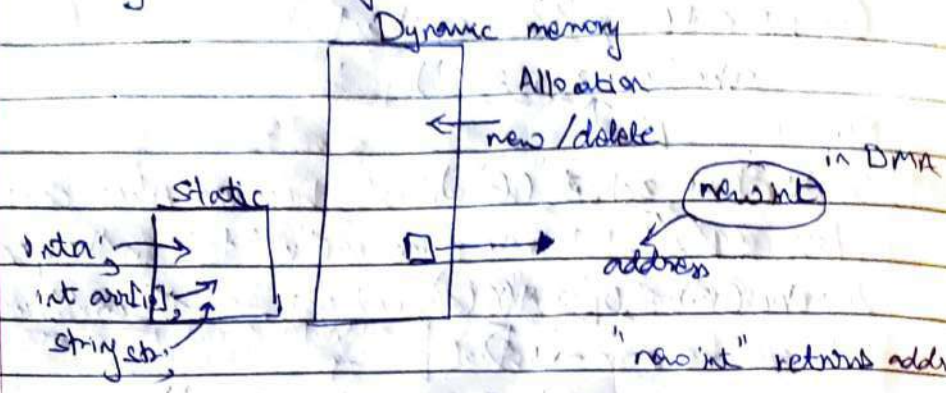
Output:
(((()))



student *s = new Student(); explicitly calls default ctr.

student *s = new student; implicitly calls default ctr.

Dynamic Memory.

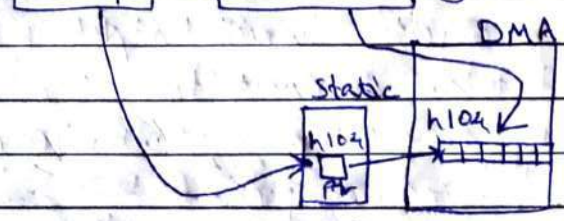


"new int" returns address

int * a = new int;
pointer address

int * left = new int[len];

int * ptr = new int [5];



Static memory gets automatically cleared
but DMA cannot be cleared automatically and we need to clean the DMA space ourself.

↳ Garbage Collector

A garbage collector is a mechanism defined to automatically manage memory by identifying and freeing up memory that is no longer use or reachable by the program.

It's primary purpose is to reclaim the memory that are no longer needed, thereby preventing memory leaks and overall improving memory management.

Their task: Identifying unused Objects
Reclaiming Memory
Memory Defragmentation.

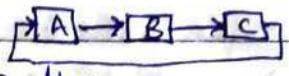
Memory leak: It occurs when a program allocates memory but fails to release it even after it is no longer needed as a result the system is unable to reclaim its memory, and it becomes unavailable for other processes or parts of the program eventually leading to loss of available memory.

Factors

i. Failure in deallocating memory (forgot to use delete).

ii. Lost pointers: Losing all the references to DMA memory without freeing it, making it impossible to ~~make~~ deallocate

iii. Circular references (linklist, graphs) eg circular LL (deleting one node will create an indefinite execution of the destructor



Dynamic memory Allocation

```
Animal* swash = new Animal;
    // dot operator                    // arrow operator
(*swash).age = 15;      OR      swash->age = 17;
(*swash).type = "billi";        swash->type = "kutta";
```

This pointer

```
this->weight = weight;
```

Constructor, default, parameterised, copy constructor

```
Animal b = a;
Animal c (b);
```
} object copy

animal a b

```
Animal (Animal &obj)
{
    PBR
    ———— copy
    ————
    ———— loop
}
```

copy constructor is always used with reference.

List list. [Collection of Node]

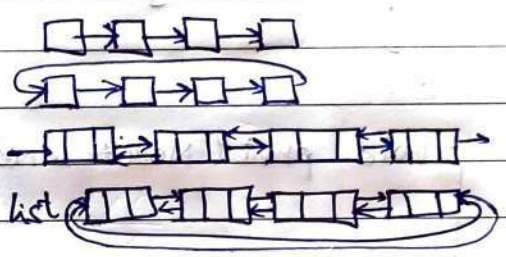
pointer to integer → int *
pointer to node → node *

L-1. Dynamic data structure
↳ non contiguous memory location
↳ no wastage of memory
↳ dynamically grow / shrink

Node
data	adr. of next node
↓ int	↓ Node *

```
class Node
{
    int data;
    Node *next;
}
```

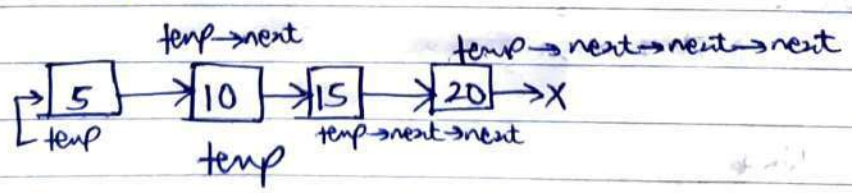
Types: Singly Link list.
Circular Singly Linklist.
Doubly Link list.
Circular Doubly Link list




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

class Node {
    public: int data;
            Node * next;

    Node ()
    {
        This -> data = 0;
        This -> next = NULL;
    }

    Node (int data)
    {
        This -> data = data;
        This -> next = NULL;
    }
};
```

```
int main() {
    Node* first = new Node(10);
    Node* second = new Node(20);        // 10 20 30 40 50
    Node* third = new Node(30);
    Node* fourth = new Node(40);
    Node* fifth = new Node(50);

        first->next = second;
        second->next = third;
        third->next = fourth;
        fourth->next = fifth;
                printing (first);
    return 0;
}
```

```
void print(Node* &head) {

    Node* temp = head;
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
}
```



We cannot do temp++ as Linklink is non contiguous

temp->next->data // 10

temp->next->next->data //15

```
if (head == NULL) {
    Node* newNode = new Node (data);
    head = newNode;
    tail = newNode;
    return;
}
```

Page No :
Date: 28

Inserting at head (head, 20)

↳ create a node

↳ newnode->next = head;

↳ head = new Node;

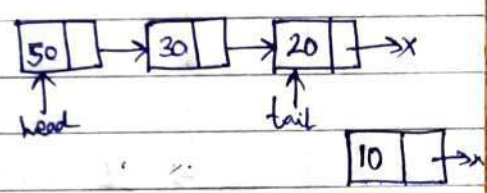
```
void insertAtHead (Node* &head, int data) {
    // step 1
    Node* newNode = new Node (data);
    // step 2
    newNode -> next = head;
    // step 3
    head = newNode;
}
```

By reference because we want to change it in original string and not want to create a copy of it.

```
if (head == NULL) {
    tail = newNode;
}
```

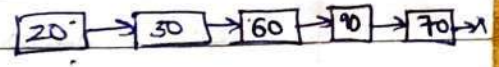
```
int main() {
    Node* tail = NULL;
    Node* head = NULL;
    insertAtHead (head, 20);
    insertAtHead (head, 50);
    insertAtHead (head, 60);
    insertAtHead (head, 90);
}
```

```
void insertAttail (Node* &head, int data) {
    // step1
    Node* newNode = new Node (data);
    // step 2 *
    tail -> next = newNode;
    // step3
    tail = newNode;
}
```



1) create a node
2) tail->next = newNode
3) tail = newNode

insertAttail(70)




```
* if (tail == NULL)
  {
      tail = newNode;
      head = newNode;
  }
  else {
      tail -> next = newNode;
  }
```

insert at head

- LL empty head = tail = NULL

  1) Create a Node
  2) head → [ ]
  3) tail → [ ]
  4) return

- Non empty

  1) Create a new Node
  2) NewNode → next = head
  3) head = new Node

insert at tail

- LL is empty head = tail = NULL

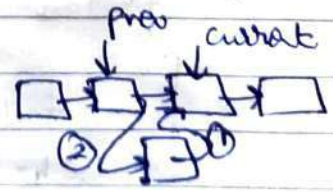
  1) Create a node
  2) head → [ ]
  3) tail → [ ]
  4) return

- LL is nonempty

  1) Create a new Node
  2) tail → next = new Node
  3) tail = new Node

insert at position

- LL is empty

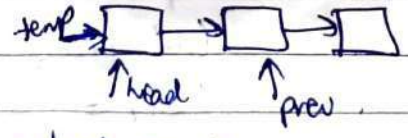


- LL is nonempty

  1) find that position
  2) new Node
  3) newnode.next = curr
  4) prev → next = new Node

```
int data,
void insertAtPosition (int position, Node * &head, Node * tail) {

    if (head == NULL) {
        Node* newNode = new Node (data);
        head = newNode;
        tail = newNode;
        return
    }

    if (position == 0) {
        insertAtHead (head, tail, data);
        return;
    }

    int len = findLength(head);
    if (position == len) {
        insertAttail (head, tail, data);
        return;
    }

// step1  find the position & curr.
    int i = 1;   Node * prev = head;
    while (i < position-1) {
        prev = prev -> next;
        i++;
    }
    Node *curr = prev -> next;

// step2
    Node * newNode = new Node (data);

// step3
    newNode -> next = curr;

// step4
    prev -> next = ~~curr~~ newNode;

int findLength(Node *&head)
{
    int len = 0;
    Node * temp = head;
    while (temp != NULL) {
        temp = temp -> next;
        len ++;
    }
    return len;
}
```

Deleting a Node



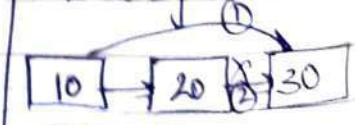
1) head = head -> next;
2) temp -> next = NULL;
3) delete temp

1) prev find
2) prev -> next = NULL
3) tail = prev
4) delete (temp);

```
~Node(){
    delete next;
}

void deleteNode (int position, Node* &head, Node* &tail) {

    if (head == NULL) {
        cout << cannot delete empty linklist";
        return;
    }

    if(position == 1) {
        Node *temp = head;
        head = head -> next;   temp -> next = null;
        delete temp;   return
    }

    int len = findlength (head);
    if(position == len ?) {
    // find prev.
        int i = 1;   Node * prev = head;
        while (i < position - 1) {
            prev = prev -> next;
            i++;
        }
        // step2
        prev -> next = NULL;
        //step3
        Node * temp = tail;
        //step 4
        tail = prev;
        // step 5
        delete temp;
        return;
    }
}
```

deleting 1st node

Deleting middle node



1) find prev & current
2) prev->next = current->next
3) current->next = NULL;
4) delete current;

```
// deleting middle node
    int i=1;
    Node * prev=head;
    while (i<position-1) {
        prev = prev->next;
        i++;
    }
    Node * curr = prev->next;

    // step 2
    prev->next = curr->next;
    //step 3
    curr -> next = NULL;
    // step 4
    delete curr;
}
```

L-2. Dowlly link list.

```
class Node
{ public:
    int data;
    Node * prev;
    Node * next;

    Node () {
        this->data = 0;
        this->prev = NULL;
        this->next = NULL;
    }

    Node (int data) {
        this->data = data;
        this->prev = NULL;
        this->next = NULL;
    }

    ~Node () {
        cout << "Node with value" << this->data;
        << "deleted" << endl;
    }
};
```

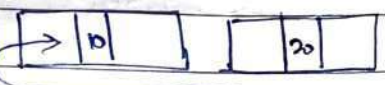
```
void print (Node* &head) {
    Node * temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
}
```

```
int getLength (Node* head) {
    int len = 0;
    Node * temp = head;
    while (temp != NULL) {
        temp = temp->next;
        len++;
    }
    return len;
}
```

```
void insertAthead (Node* &head, Node*
        &tail, int data) {
    if (head == NULL) {
        // LL is empty
        Node * newNode = new Node(data);
        head = newNode
        tail = newNode
        ~~return;~~
    }
    else {
        Node * newNode = new Node(data);
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
```

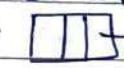
```
int main () {
    Node* first = new Node(10);
    Node* second = new Node(20);
    Node* third = new Node(30);
    Node* head = first;
    Node* tail = third;
    first->next = second;
    second->prev = first;
    second->next = third;
    third->prev = second;
    print (first);
    return 0
}
```

10 20 30



insert at head

1) create node

2) 

3) head->prev = new Node

4) head = new Node

insertAthead (head, tail, 101);

print (head);

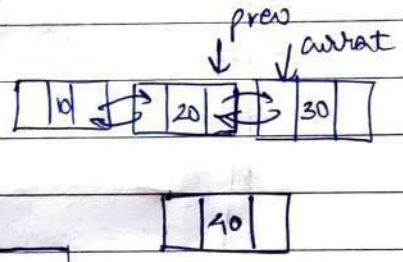
101 10 20 30

```
insertAttail(Node *&head, Node *&tail, int data){
    if (head==NULL){
        Node *newNode = new Node (data);
        head = newNode;
        tail = newNode;
    }
    else {
        Node * newNode = newNode(data);
        tail -> next = newNode;
        newNode -> prev = tail;
        tail = newNode;
    }
}
```

1) create node
2) tail -> next = new Node
3) node -> prev = tail
4) tail = new Node

```
insertAttail (501);
cout << endl;
print (head);
```

// 101 10 20 30 501




```
void insertAtposition(Node *head, Node *tail, int data, int position){
    if(head==NULL){
        Node * newNode = new Node (data);
        head=newNode;
        tail=newNode;
    }
    else {
        if (position==1){
            insertAthead(head, tail, data);
            return;
        }
        int len=getlength(head);
        if position > len){
            insertAttail(head, tail, data);
            return;
        }
```

1. First prev & current
2. create a node
3. prev -> next = new Node
4. ~~next~~ curr -> prev = new Node
5. nextNode -> prev = prevNode
6. newNode -> next = curr;

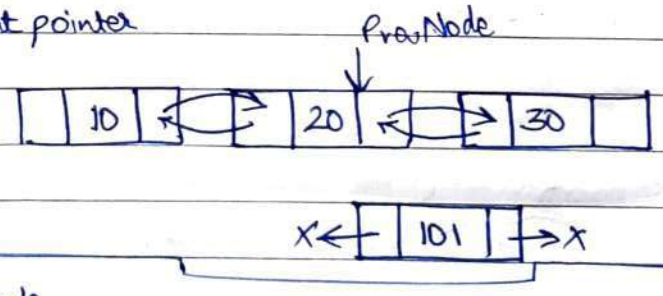
```
// insertAtMiddle
    // find prev and current.

        int i=1;      Node * prevNode = head;
        while (i<position -1) {
                prevNode = prevNode->next;
                i++;
        }

        Node* curr = prevNode->next;

        //step2  create a node
        Node* newNode = newNode (data);

        //step3
        prevNode->next = newNode;
        newNode->prev = prevNode;
        curr->prev = newNode;
        newNode->next = curr;
}
```

insertatPosition(head, tail, 401, 1);

401 101 10 20 30

501

↳ without current pointer



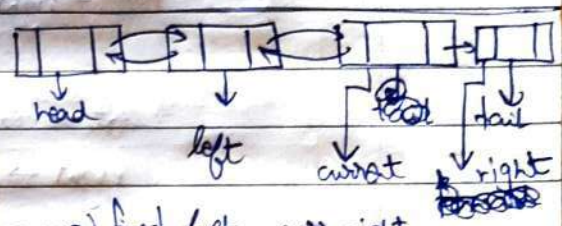
Order matter

1) Find prev node
2) create new node
3) prevNode->next->prev = newNode
4) newNode->next = prevNode->next
5) prevNode->next = newNode
6) NewNode->prev = prevNode

↳ Deleting a Node

a) temp = head;
b) head = head->next;
c) head->prev = NULL;
d) temp->next = NULL;
e) delete temp;

```
void deletefromposition (Node *head, Node *tail, int position) {
    if (head == NULL) {
        cout << "LL is empty";
        return;
    }

    if (head -> next == NULL) {
        // single node
        Node * temp = head;
        head = NULL;
        tail = NULL;
        delete temp;
        return;
    }

    if (position == 1) {
        // want to delete the first node
        Node *temp = head;
        head = head -> next;
        head -> prev = NULL;
        temp -> next = NULL;
        delete temp;
        return;
    }

    int len = getLength(head);
    if position == len () {
        // deleting last node
        Node * temp = tail;
        tail = tail -> prev;
        temp -> prev = NULL;
        tail -> next = NULL;
        delete temp;
        return;
    }

    if (position > len) {
        cout << "please enter correct position";
    }
```



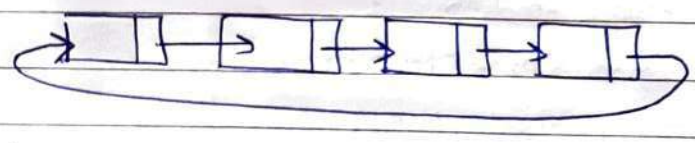
a) find left, curr, right
b) left -> next = right;
c) right -> prev = left;
d) curr -> prev = NULL;
e) curr -> next = NULL;

deleting tail node

```
temp = tail
tail = temp -> prev
temp -> prev = NULL;
temp -> next = NULL;
delete temp;
```

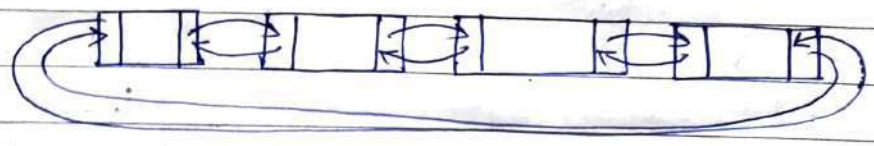
```
// deleting the middle node of linklist.
    int i=1;
    while (i < position-1) {
        left = left -> next;
        i++;
    }
    Node * curr = left -> next;
    Node * right = curr -> next;

    left -> next = right;
    right -> prev = left;
    current -> next = NULL;
    current -> prev = NULL;
    delete ~~NULL~~ current;
}
```

Circular Linklist

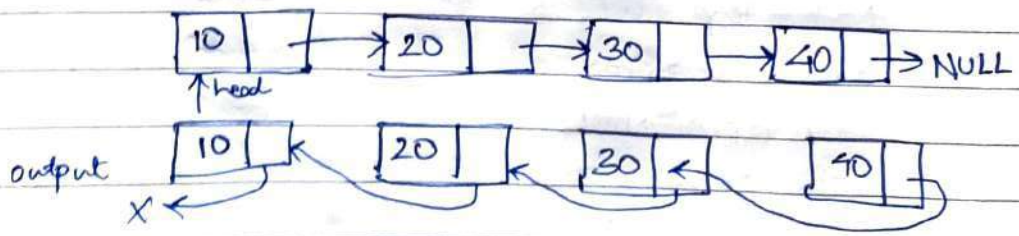


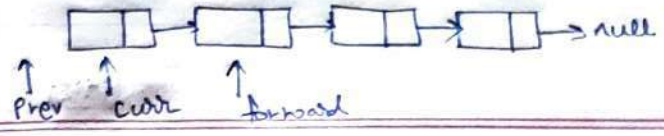
there is no concept of head or tail in Circular link list.

Circular doubly linklist



ex Reverse a Linklist.






```
Node* reverse(Node* &prev, Node* &curr) {
    // base case
    if (curr == NULL) {
        return prev;   // LL reverse ho chuka hai.
    }

    // 1 case solve कर दो बाकी recursion संभाल लेगा

    Node* forward = curr->next;
    current->next = prev;
    reverse(curr, forward);
}
```

```
Node* prev = NULL;
Node* curr = head;
head = reverse(prev, curr);
print(head);
```

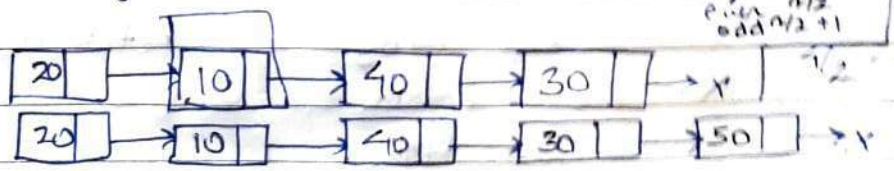
```
//iterative
Node* reverseusingLoop(Node* head) {
    Node* prev = NULL;
    Node* curr = head;

    while (curr != NULL) {
        Node* temp = curr->next;
        curr->next = prev;
        prev = curr;
        curr = temp;
    }

    return prev;
}
```

eg. Find middle of Linklink

length $O(n)$ | Ans

even $n/2$

odd $n/2 + 1$

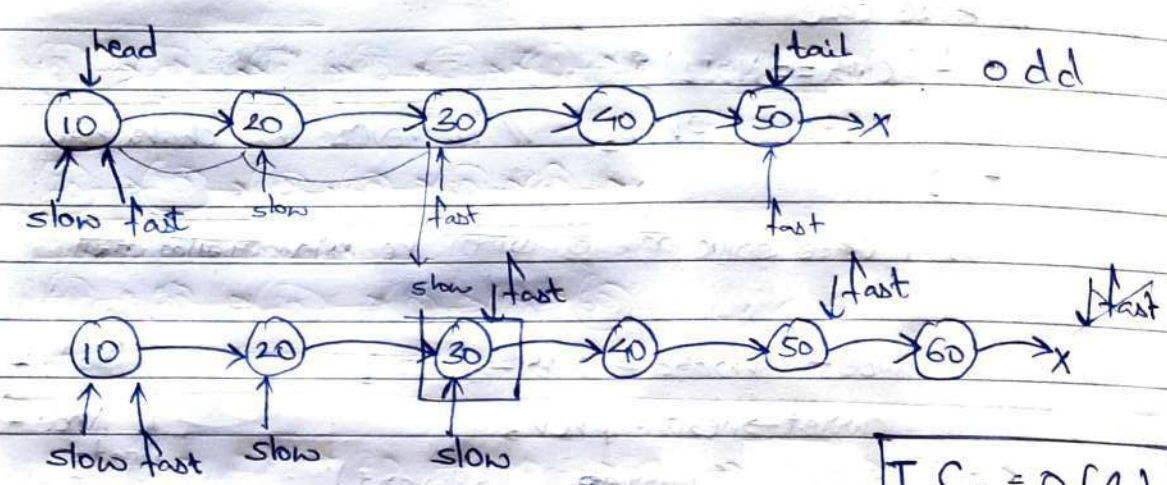


$4/2 = 2$

$5/2 = 2 + 1$

Approach-2.

ex 2-pointer approach, slow & fast pointer approach & Tortoise algorithm



$\boxed{T.C. = O\left(\frac{n}{2}\right)}$

```
class Node {
    public:
        int data;
        Node * next;

        Node (int data) {
            this -> data = data;
            this -> next = null;
        }
}
```

```
void print (Node * head) {
    Node * temp = head;
    while (temp != NULL) {
        cout << temp -> data;
        temp = temp -> next;
    }
    cout << endl;
}
```

```
int main() {
    Node * head = new Node (10);
    Node * second = new Node (20);
    Node * third = new Node (30);
    Node * fourth = new Node (40);
    Node * fifth = new Node (50);
    head -> next = second;
    second -> next = third;
    third -> next = fourth;
    fourth -> next = fifth;
    fifth -> next = NULL;

    print (head);
    ~~return~~
    cout << getMiddle (head) -> data;
        // 30

    return 0;
}
```

```
Node* getMiddleNode (Node*head){
    if (head == NULL) {
        cout"LL is empty";
        return;
    }

    if (head -> next = NULL) {
        // only 1 node in LL.
        return head;
    }

    // LL have > 1 node
    Node* slow = head;
    Node* fast = head;
```

no need as fast would be always ahead of slow pointer.

```
    while (slow != NULL && fast != NULL){
        fast = fast -> next;
        if ( fast ≠ NULL){
            fast = fast -> next;
            slow = slow -> next;
        }
    }

    return slow;
}
```

10 20 30 40 50

// 30

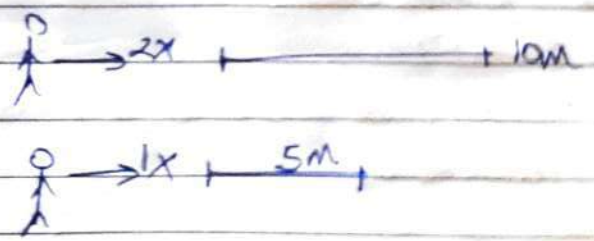
10 20 30 40 50

30

Node* fast = head -> next;

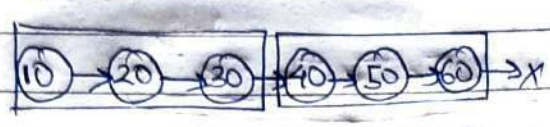
40

Node* fast = head;

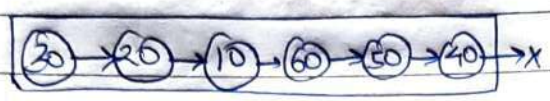


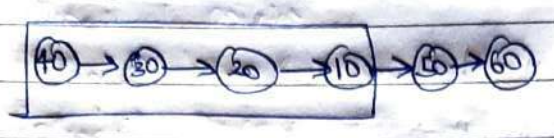
imp

ex K groups reverse LL

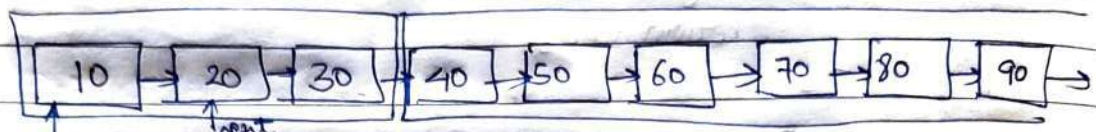


If k=3



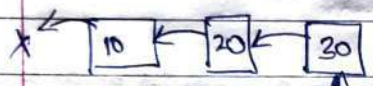


If k=4.

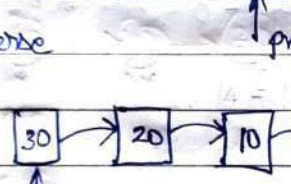


1 case hum solve

baki recursion sambhal लेगा।



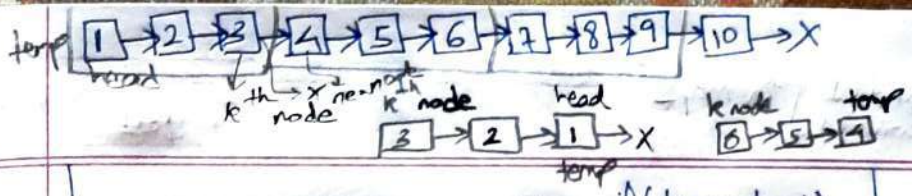
A) k node reverse



(B) - head → next = recursion ka ans

(C) return prev;

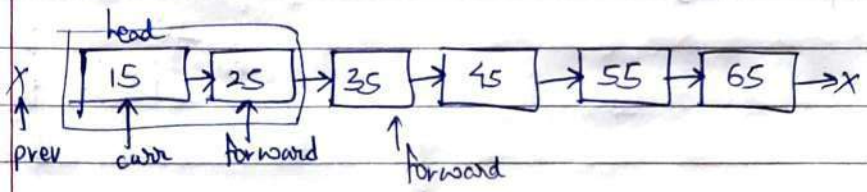
```
Node* reverseKNodes(Node* head, int k){
    if(head==NULL){
        cout<<"LL is empty"<<endl;
        return NULL;
    }

    int len = getLength(head);
    if(k>len){
        cout<<"Enter valid value for k"<<endl;
        return head;
    }

    //it means number of nodes in LL >= k
    //step A reverse first k nodes of LL
```

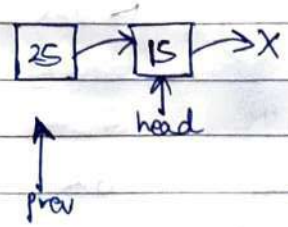



```
Node* prev = NULL;
Node* curr = head;
Node* forward = curr->next;
int count = 0;

while ( count < k) {
        forward = curr->next;
        current->next = prev;
        prev = curr;
        curr = forward;
        count++;
}

if ( forward != NULL) {
    // we still have nodes to reverse
    head->next = reverseKNodes ( forward, k);
}

return prev;
}
```

```
prevNode if (temp == head)
{ // update head
  head = k^th node
}
```

```
temp = head;   NewNode
while (temp != NULL)
{
    k^th Node = findKNode(temp, k);
    if (kNode == NULL)
    {
        if(prevNode) prevNode -> temp;
        break;
    }
    NextNode = kNode->next
    kNode->next = NULL;
    reverse (temp);
    if (temp == head)
    {
        head = k^th Node
    }
    else {
        prevNode->next = k^th node
        prevNode = temp;
        temp = nextNode
```





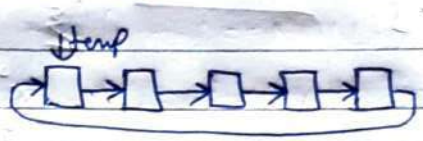
Node* recursionkaAns = reverseNode (forward, k);

c) head->next = recursion;

d) return prev;

find() } in maps
count()

↳ LL is circular or not.

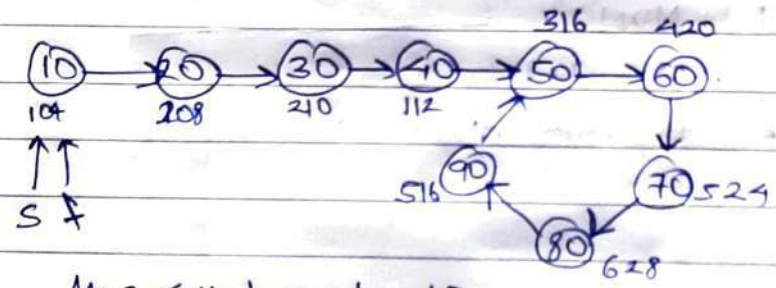
1) 

temp = head;
क्या तुम head node तो नहीं हो?

Method 1) 1 pointer
Method 2) slow fast pointer
Method 3) Maps

↳ Delete & detect a loop

↳ check loop is present in LL or not.
↳ starting point of loop
↳ remove loop



Approach 1)

Map <Node*, bool>
map [temp] = true;

Map

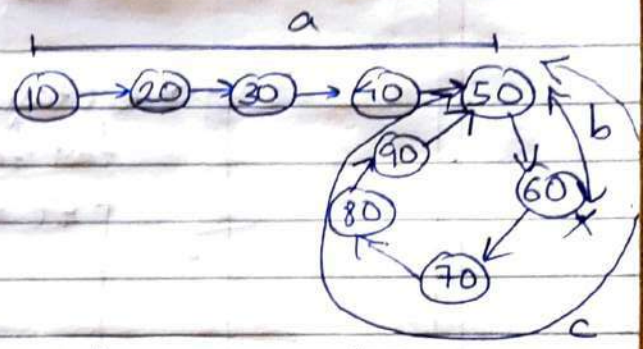
Keys	Values
address	bool
104	true
208	true
210	true
112	true
316	true
420	true
524	true
628	true
516	true
316	

hence loop

Floyd Cycle Detection (FCD) (Slow fast Algorithm)

slow == fast → loop present
fast == NULL → loop absent.

```
int main() {
  Node* head = new Node(10);
  Node* second = new Node(20);
  Node* third = new Node(30);
  Node* fourth = new Node(40);
  Node* fifth = new Node(50);
  Node* sixth = new Node(60);
  Node* seventh = new Node(70);
  Node* eigth = new Node(80);
  Node* ninth = new Node(90);
  head->next = second
  second->next = third;
  third->next = fourth
  fourth->next = fifth;
  fifth->next = sixth;
  sixth->next = seventh;
  seventh->next = eight;
  eight->next = ninth;
  nieth->next = fifth;
```



$2d(\text{slow}) = d(\text{fast})$

$2(a+xc+b) = a+yc+b$

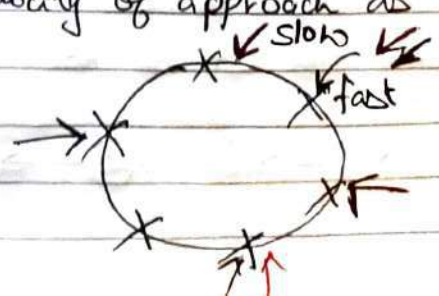
$a+b = (y-2x)c$

$y-2x = n$

$a+b = nc$

$b+a = nc$

In a circle both the pointers gets in and the fast is moving in the same direction as that of slow pointer with double speed as that of slow pointer making velocity of approach as v times, so each time 1 node gets nearer



5) 4) 3) 2) 1) 0)

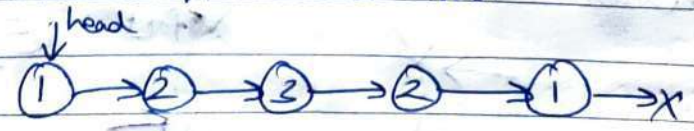
~ starting point of loop
~ remove loop

```
Node*
bool checkforLoop (Node* head) {
    if( head == NULL) {
        cout << "LL is empty" << endl;
        return false;   NULL
    }

    Node* slow = head;
    Node* fast = head;

    while (fast != NULL) {
        fast = fast -> next;
        if( fast != NULL) {
            fast = fast -> next;
            slow = slow -> next;
        }

        if (slow == fast) {
            return true; // loop present   slow = head;
                                           break;
        }
    }

    // fast NULL hoga
    return false;
}
```

```
Node* prev = fast;
while (slow != fast) {
    prev = fast;
    slow = slow -> next;
    fast = fast -> next;
}
prev -> next = NULL;
return slow;
```

LL- Level 4.

Palindrome Linklist or not.



Approach 1

linklist
reverse linklist → compare → equal (palindrome)
→ unequal (non palindrome)

Approach 2

LL data → array (copy)

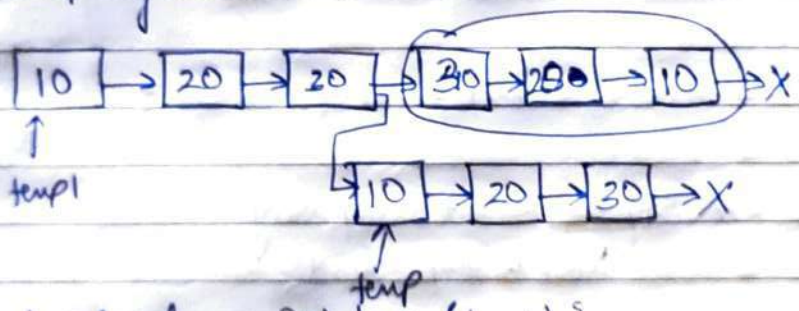
s++; e--;

Approach 3

a) Find middle $O(n)$
b) reverse LL after middle node. $O(n)$
c) start comparing both the halves. $O(n)$

$O(n)$

$SC = O(1)$




```
bool CheckPalindrome (Node* &head) {
    if (head == NULL) {
        cout << "LL is empty";
        return true;
    }

    if (head -> next == NULL)
        return true    // only 1 node
```

>1 node

```
// step A) find middle node
Node * slow = head;
Node * fast = head -> next;

while (fast != NULL) {
        fast = fast -> next;
        if (fast != NULL) {
                slow = slow -> next;
                fast = fast -> next;
        }
}
```

```
Node * reverse (Node * head) {
    Node * prev = NULL;
    Node * curr = head;
    Node * next = curr -> next;
    while (curr != NULL) {
        next = curr -> next;
        curr -> next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```

// slow pointer is pointing to the middle node

```
// Step 2 reverse LL after middle node
Node * reverseKahead = reverse (slow -> next);

slow -> next = reverseLLkahead;

// Step 3 start comparing
Node * temp1 = head;
Node * temp2 = reverseKahead;

while (temp2 != NULL) {
        if (temp1 -> data != temp2 -> data) {
                return false;
        }
        else {
                temp1 = temp1 -> next;
                temp2 = temp2 -> next;
        }
}
return true;
}
```

ex Remove duplicated from a sorted linklist

$TC = O(n)$
$SC = O(1)$

i/p 1 → 2 → 2 → 2 → 1 → 1 → x

o/p 1 → 2 → 3 → 1 → x

curr → data == curr → next

equal — not equal

Current → next = curr → next → next Current = current → next;

```
void removeduplicates (Node *head) {
    if (head == NULL) {
        cout << "Link list is empty";
        return;
    }
    if (head → next = NULL) {
        cout << "Linklist contains 1 element";
        return;
    }

    // >1 node in LL
    Node * curr = head;
    while (curr != NULL) {
        if (curr → next != NULL) && (curr → data == curr → next → data) {
            //equal
            Node * temp = curr → next;
            curr → next = curr → next → next;
            temp → next = NULL;
            delete temp;
        }
        else {
            //not equal
            curr = curr → next;
        }
    }
}
```

H.W Remove duplicated from unsorted LL

i/p [4] → [2] → [1] → [4] → [2] → [7] → X

o/p [4] → [2] → [1] → [7] → X

#1) nested loops
#2) Maps
#3) sort and prev logic

ex Sort 0's, 1's and 2's

i/p [2] → [1] → [1] → [0] → [0] → [2] → [1] → X

o/p [0] → [0] → [1] → [1] → [1] → [2] → [2] → X

zero count = ~~0~~ 2
one count = ~~0~~ 3
two count = ~~0~~ 2

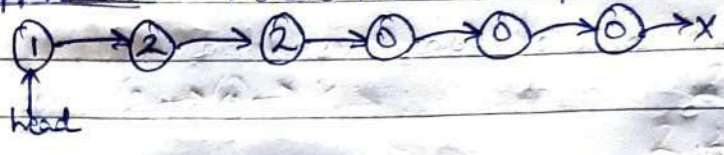
TC - O(n)
SC - O(1)

Data replacement Not allowed.

```
int main() {
Node* head = new Node (1);
Node* second = new Node (2);
Node* Third = new Node (2);
Node* fourth = new Node (0);
Node* fifth = new Node (0);
Node* sixth = new Node (0);
head -> next = second;
second -> next = third;
third -> next = fourth;
fourth -> next = fifth;
fifth -> next = sixth;
cout << "input LL:";
print(head);
sortzeroOnetwo(head);
print(head);
}
```

```
// with data replacement
void sortZeroOneTwo (Node * &head)
    // step1 find count of zeros, ones and twos
    int zero = 0;
    int one = 0;
    int two = 0;

    Node * temp = head;
    while (temp != NULL) {
        if (temp -> data == 0)
            zero++;
        else if (temp -> data == 1)
            one++;
        else if (temp -> data == 2)
            two++;
        temp = temp -> next;
    }

    // step2 fill 0, 1 and 2 in the original LL
    temp = head;
    while (temp != NULL) {    // This loop we can also skip
        while (zero--) {
            temp -> data = 0;
            temp = temp -> next;
        }

        while (one--) {
            temp -> data = 1;
            temp = temp -> next;
        }

        while (two--) {
            temp -> data = 2;
            temp = temp -> next;
        }
    }
}
```

Approach 2 There is no data replacement.

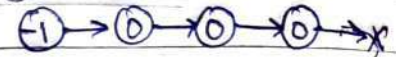


A) temp = head

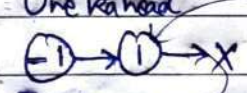
B) head = head → next;

C) temp → next = NULL;

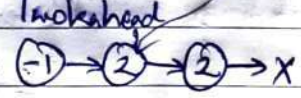
zerokahead



Onekahead



Twokahead




```
Node* sort12and0(Node* &head) {
    Node* zerohead = new Node(-1);
    Node* zerotail = ~~new Node~~ zerohead;

    Node* onehead = new Node(-1);
    Node* onetail = onehead;

    Node* twohead = new Node(-1);
    Node* ~~t~~otail = twohead;

    // traverse The original linklist.
    Node* curr = head;
    while (curr != NULL) {

        if(curr->data == 0) {
            Node* temp = curr;
            curr = curr->next;
            temp->next = NULL;
        }

        zeroTail->next = temp;
        zeroTail = temp;
    }
```

A) Join

B) Remove dummy nodes

C) return head.

Optional.

```
if(head == NULL){
    cout << "LL is empty" << endl;
    return NULL;
}

if(head->next == NULL){
    return head;
}
```

```
else if ( curr -> data == 1) {
    Node * temp = curr;
    curr = curr -> next;
    temp -> next = NULL;

    onetail -> next = temp;
    onetail = temp;
}

else if ( curr -> data == 2) {
    Node * temp = curr;
    curr = curr -> next;
    temp -> next = NULL;
    twotail -> next = temp;
    twotail = temp;
}
}

// Join them // remove dummy nodes
// Modify one zero list.
Node *temp = oneHead;
oneHead = oneHead -> next;
temp -> next = NULL;
delete temp;

// Modify two zero list.
temp = twohead;
twohead = twohead -> next;
temp -> next = NULL;
delete temp;

if (onehead != NULL) {
    zerotail -> next = onehead;
    if (twohead != NULL)
        onetail -> next = twohead;
}

else {
    // one zero list is
    // empty
    if (twoHead != NULL)
        zeroTail -> next = twohead;
}

temp = zerohead;
zerohead = zerohead -> next;
temp -> next = NULL;
delete temp;
return zerohead;
}
```

ex Add 2 numbers represented by linklist.

TC = O(Max(m,n))
SC = O(1)

i/p (2) → (4) → X

(2) → (3) → (4) → X

$$\begin{array}{r} 24 \\ 234 \\ \hline 258 \end{array}$$

on reversing

(4) → (2) → X

(4) → (3) → (2) → X

A) Reverse both LL
B) Add both of them
C) Reverse the ans LL

```
int main() {
    Node * head1 = new Node(2);
    Node * second1 = new Node(4);
    head1 -> next = second1;

    Node * head2 = new Node(2);
    Node * second2 = new Node(3);
    Node * third2 = new Node(4);
    head2 -> next = second2;
    second2 -> next = third2;

    solve(head1, head2);
    print(ans);
}
```

```
Node * reverse(Node * head){
    Node * prev = NULL;
    Node * curr = head;
    Node * next = curr -> next;
    while (curr != NULL){
        next = curr -> next;
        curr -> next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```

```
Node * solve (Node * & head1, Node * & head2) {
    // step A reverse both the link list.
    head1 = reverse(head1);
    head2 = reverse(head2);

    if (head1 == NULL)
        return head2;

    if (head2 == NULL)
        return head1;
```

• HW
adding 1 to LL.
• Merge two sorted LL.


```
//step 2 add both the LL
Node * ansHead = NULL;
Node * ansTail = NULL;
int carry = 0;

while (head1 != NULL && head2 != NULL) {
    int sum = carry + head1->data + head2->data;
    int digit = sum % 10;
    carry = sum / 10;

    Node * newNode (digit);

    if (ansHead == NULL) { // first node insertion
        ansHead = newNode;
        ansTail = newNode;
    }
    else {
        ansTail->next = newNode;
        ansTail = newNode;
    }

    head1 = head1->next;
    head2 = head2->next;
}
```

```
while (head1 != NULL) {
    int sum = carry + head1->data;
    int digit = sum % 10;
    carry = sum / 10;
    Node * newNode = newNode (digit);
    ansTail->next = newNode;
    ansTail = newNode;
    head1 = head1->next;
}
```

```
while (head != NULL)
    int sum = carry + head2->data;
    int digit = sum % 10;
    carry = sum / 10;
    Node * newNode = newNode (digit);
    ansTail->next = newNode;
    ansTail = newNode;
    head2 = head2->next;
}
```

```
// handle carry
while ( carry != 0) {
    int sum = carry;
    int digit = sum%10;
    carry = sum/10;
    Node* newNode = newNode (digit);
    ansTail -> next = newNode;
    ansTail = newNode;
}

// Step3 reverse the final answer
ans head = reverse (ansHead);
return ansHead;
}
```

Lecture1 Stacks

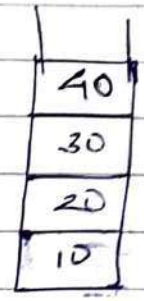
FO
First in Last Out.

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

int main() {
    stack <int> st;    // creation
    st. push(10);
    st.push(20);
    st.push(30);
```

```
S. push(50);
S. push(55);
S. push(420);
S. pop();
S. pop();
S. size();
S. empty();
```

T F

st.push(40);



//remove
s.pop();

// check the element on the top

cout<< st.top(); // 30
cout<< st.size(); // 3
cout<< st.empty(); // F

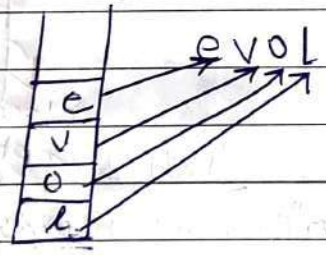
Stack में index based accessing नहीं होती, बस top से access कर सकते हैं।

```
while(! s.empty()){
cout<< s.top();
s.pop();
}
```

Printing Stack

push / pop
तो मुझे reverse order में मिलते वाले हैं।

love



To create stack we can use
① Array
② LinkedList

1 array → 1 Stack

```
class Stack {
    public:
        int *arr;
        int top; // top element index
        int size; (it does not symbolyse no. of elements
                  it only measures the size of array.
```

size

0	1	2	3	4	5	6

```
Stack (int size) {
    arr = new int [size];   top = -1;
    this -> size = size;
    top = -1;
}
```

size = 7
top = -1

// functions

push

space available → insert: top++, arr[top] = data

space not available → overflow

size - top ≤ 1

size - top > 1

```
void push (int data) {
    if ( size - top > 1) {
        // space available
        // insert:
        top++;
        arr[top] = data;
    }

    else {
        // space not available
        cout << "stack overflow";
    }
}
```

pop

stack empty → underflow

stack not empty → top--;

```
void pop() {
    if (top == -1) {
        cout << "Stack is underflow";
    }

    else {
        top--; // stack is not empty
    }
}
```

```
int getTop() {
    if(top==-1) {
        cout<<"there is no element in stack";
    }
    else {
        return arr[top];
    }

//returns no. of valid elements in stack
int getSize() {
    return top+1;
}

bool isEmpty() {
    if(top==-1) {
        return true;
    }
    else {
        return false;
    }
}
};

int main() {
//insertion
    S.push(10);          //40 30 20 10
    S.push(20);
    S.push(30);
    S.push(40);
}
```

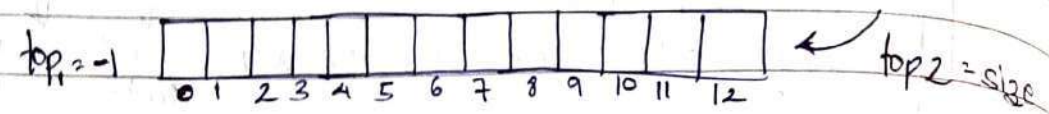
getTop()

stack empty

stack not empty

return arr[top]

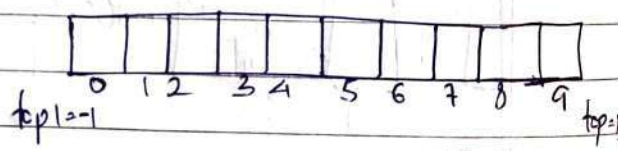
// 2 stacks in an array



#1 half - half
S1 S2 [memory wastage]

#2 either sides
[most optimal]

```
class Stack {
    public:
        int *arr;
        int size;
        int top1;
        int top2;

        Stack (int size) {
            arr = new int[size];
            this->size = size;
            top1 = -1;
            top2 = size;
        }

        // functions
        void push1(int data) {
            if(top2-top1 == 1) {
                cout << "overflow";
            }
            else {
                top1++;
                arr[top] = data;
            }
        }
}
```



push1()
space not available → overflow
space available → top++ arr[top] = data;

top2 - top1 = 1

```
void push2 (int data){
    if( top2-top1==1) {
        cout<< "overflow";
    }
    else {
        top2 --;
        arr [top2] = data;
    }
}
```

pop1

stack empty → underflow

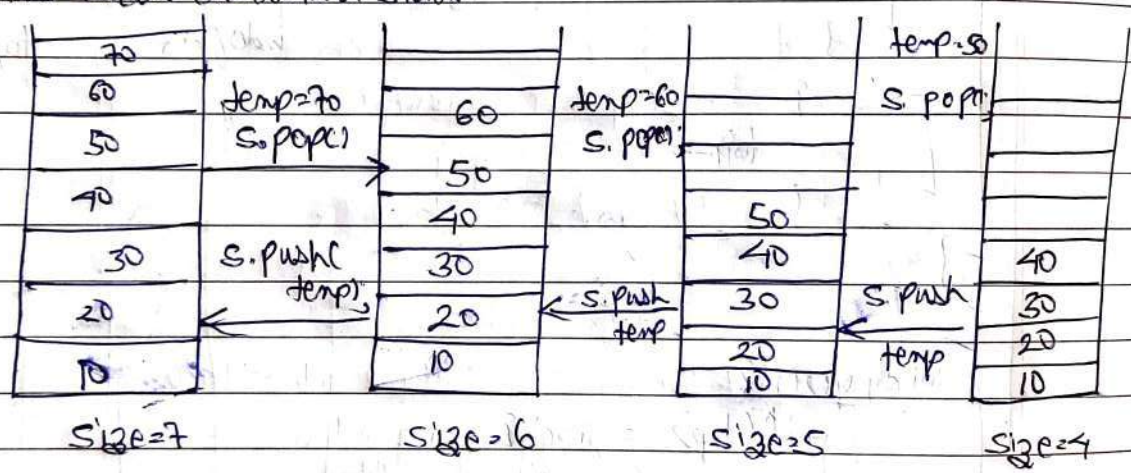
stack not empty → top--

```
void pop1() {
    if (top1==-1) {
        cout << underflow S1";
    }
    else {
        top--;
    }
}

void pop2() {
    if(top2==size) {
        cout << underflow S2";
    }
    else {
        top2++;
    }
}
};
```

Q. Reverse a string using stack

```
Stack <char> S;
String str = "banbbar";

for(int i=0; i< str.length(); i++){
        S.push (str[i]);
}

while (! S.empty()) {
        Cout<<S. top();
        S.pop();
}
```

ex find middle element in a stack

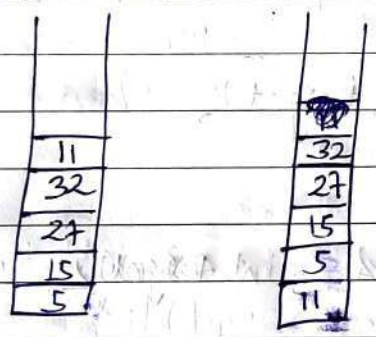


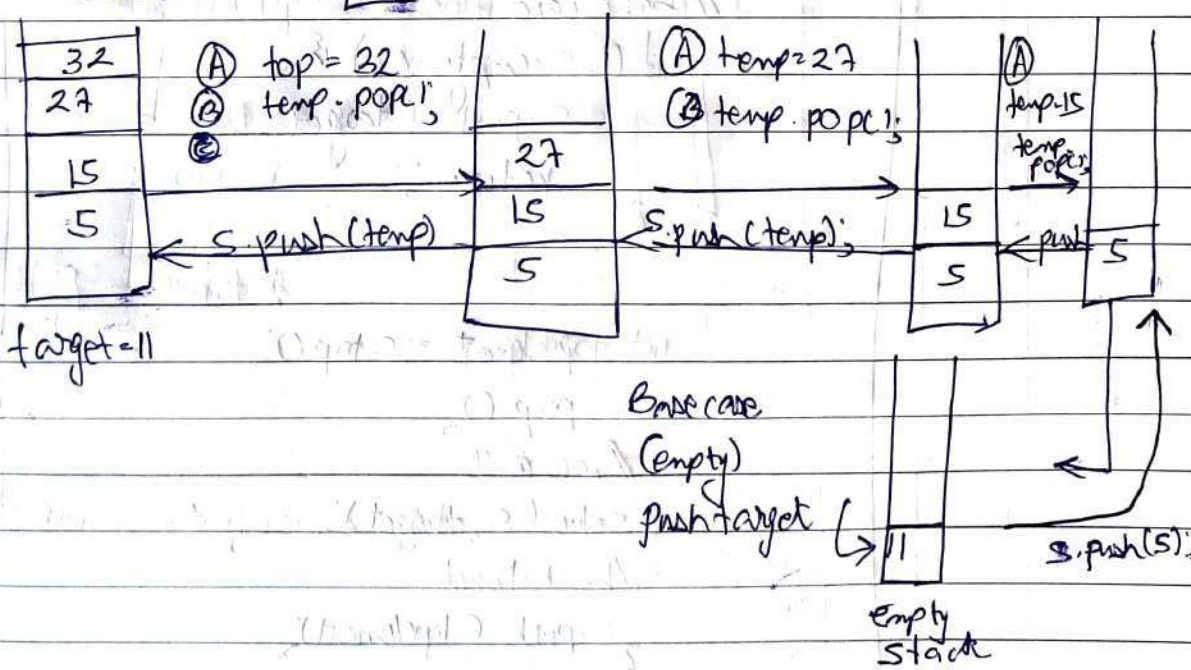

```
Stack(int> S;
    S.push(10);
    S.push(20);
    S.push(30);
    S.push(40);
    S.push(50);
    S.push(60);
    S.push(70);
    print Middle (S, S.size());
```

```
if( total size / 2 + 1 == size)
{
    print s.top
}
s.push(temp);
```

```
void printMiddle(stack<int> &s, int &totalsize) {
    if(s.size()==0) { cout<< "No element in stack"; }
    if(s.size() == totalsize /2 +1) {
        cout<< "Middle element is " << s.top()<<endl;
        return;
    }

        // recursive call
        int temp = s.top();
        s.pop();
        printmiddle (s, totalsize);
        // backtrack
        s.push(temp);
    }
```

ex insert at bottom





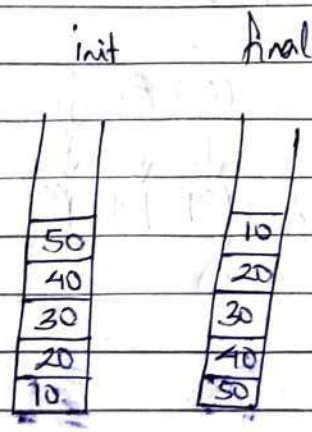
```
stack<int> s;
  s.push(10);
  s.push(20);
  s.push(30);
  s.push(40);
  s.push(50);

    insertatBottom(s);
```

Output

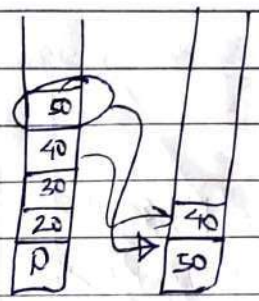
// 40 30 20 10 50

```
void insertatBottom(stack<int>&s) {
        if (s.empty()) {
            cout<< "stack is empty";
            return;
        }
        int target = s.top();
        s.pop();
        solve(s, target);
}

void solve(stack<int>&s, int target) {
        // base case
        if( s.empty()) {
            s.push(target);
            return;
        }

        int topelement = s.top();
        s.pop();
        // rec call
        solve(s, target);
        // back track
        s.push(topelement);
}
```

ex Reverse a stack



ans



Ek ko निचे लाऊंगे बाकी recursion sambhal kaega

target ko alag से रख लो फिर बाकी के stack को alag से रख लो और इसको बोलो reverse होके आए | insert at bottom

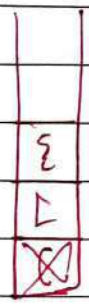
```
stack<int> s;
  s.push(10);
  s.push(20);
  s.push(30);
  s.push(40);
  s.push(50);

  reverse stack (s);

void insertatBottom(stack<int>&s){
  if(s.empty()){
      cout<< "stack is empty";
      return;
    }

  int target = s.top();
    s.pop();
    solve (s, target);
}
```

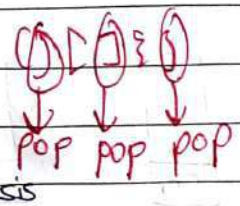
```
void reverse stack(stack<int>&s){
  //base case
  if(s.empty()){
    return;
  }

  int target=s.top();
    s.pop();
  // reverse stack
    reverse stack (s);
  // insert at bottom target ko
  insert at Bottom (s, target);
}
```

imp

Valid parenthesis problem.

([{





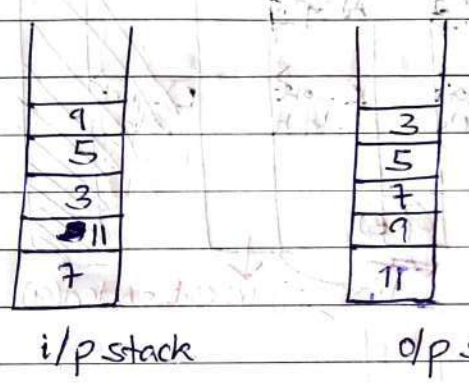
stackempty $\xrightarrow{\text{true}}$ valid parenthesis

$\downarrow$ false
invalid parenthesis

```
bool isvalid(string s){
    stack <char> st;
    for(int i=0; i<s.length(); i++){
        char ch = s[i];
        if(ch=='(' || ch='{' || ch=='[') {
            st.push(ch);
        }
        else { // closing brackets
            if(!st.empty()) {
                char topch = st.top();
                if(ch==')' && topch=='(') {
                    s.pop();
                }
                else if (ch=='}' && topch=='{') {
                    s.pop();
                }
                else if (ch=='] && topch=='[') {
                    s.pop();
                }
                else {
                    return false;
                }
            }
            else { return false;
            }
```

```
    }
  }
  if (st.empty()) {
      return true;
  }
  else {
      return false;
  }
}
```

Ex Sort a Stack.

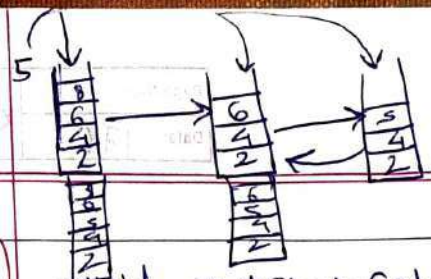


I task
↳ insertSort()

II task
↳ sortStack()

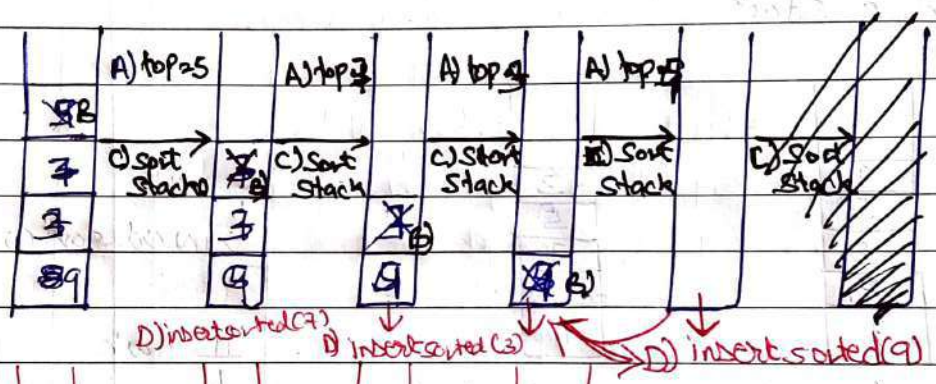
(2)

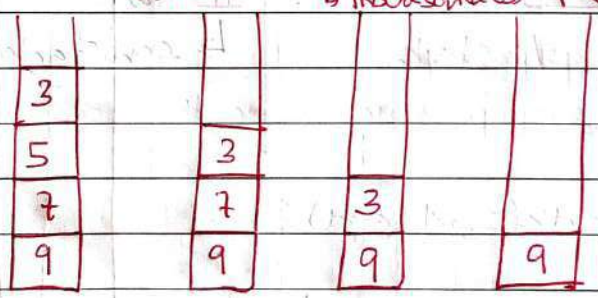
```
void insertSort (stack<int> &s, int target) {
    //base case
    if (s.empty()) {
        s.push(target);
        return;
    }
    if (s.top() >= target) {
        s.push(target);
        return;
    }
    int topElement = s.top();
    s.pop();
    insertSorted (s, target);
    // BT
    s.push(topElement);
}
```

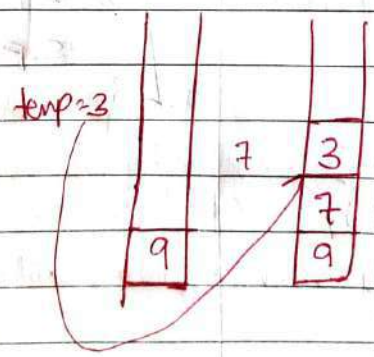


1

```
void sortStack (stack <int &s) {
    //base case
    if( s.empty()) {
        return;
    }

    int topElement = s.top();
    s.pop();
    sortStack (s);
    insertSorted (s, topElement);
}
```







Ex Remove Redundant brackets

(c) (a+b))

(a+b*c)(d))

(a+b)

(a*b + (c/d))

(c)(a/b))

दो Bracket के Beech में operator नहीं होगा valid bracket बनाने के लिए

(> push()
+

(a+b)

operator → true (flag)

↳ ((a+b))

operator true

operator = false

↳ (a+b) + (d)

operator = true

operator false

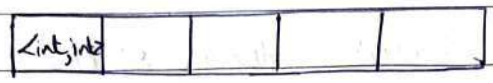
```
void findRedundant(string& str)
{
    bool ans = checkRedundant(str);
    if (ans==true)
        cout << "Yes";
    else
        cout << "No";
}
```

```
bool checkRedundancy(string& str)
{
    stack<char> st;

    for (auto& ch : str) {

        if (ch==')') {
            char top = st.top();
            st.pop();
            bool flag = true; //invalid parenthesis

            while (!st.empty && top != '(') {
                if (top=='+' || top=='-' ||
                    top=='*' || top=='/')
                    flag = false;

                top = st.top();
                st.pop();
            }

            if (flag==true)
                return true;   // invalid parenthesis
        }
        else
            st.push(ch);
    }

    return false; //valid parenthesis
}
```

(find '(' +) (false) preset (true) absent

st.back or st[st.size() - 1];

ex Min Stack



pair <int, int> p;
p.first;
p.second;

< value, vahatak kaun min hai >

2 1 4 3

v.back.second = 1
v.back.first = 3

vector < pair<int, int>> st;

```
void push (int val) {
    if( st.empty ()) {
        pair<int,int> p = make_pair(val, val);
        st.push_back(p);
    }
    else {
        pair <int, int>p;
        p.first = val;
        p.second = min(value, st.back().second);
        st.push_back(p);
    }
}
```

```
void pop() {
    st.pop_back();
}
```

```
int top() {
    return st.back().first;
}
```

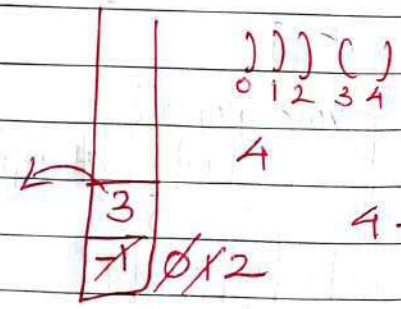
```
int getmin() {
    return st.back().second;
}
```

ex Longest valid parenthesis

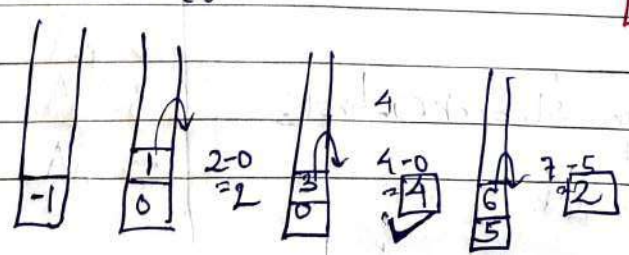
(→ insert index in stack
) → pop → ~~l~~ legth

```
int longestValidParenthesis(string s) {
    stack <int> st;
    st.push(-1);
    int maxLen = 0;
    for (int i=0; i< s.length(); i++) {
        char ch = s[i];
        if (ch == '(') {
            st.push(i);
        }
        else {
            st.pop();
            if (st.empty()) {
                st.push(i);
            }
            else {
                int len = i - st.top();
                maxLen = max(len, maxLen);
            }
        }
    }
    return maxLength;
}
```

imp



0 1 2 3 4 5 6 7
↳) ( ) ( )) ( )



ex Next smaller element.

$\boxed{\text{complexity } (O(n))}$

1	-1	3	-1
2	1	4	3

1 से chota koi और number है या नहीं

ans 1 -1 3 -1

L-R

$O(n^2)$

R→L

Hint: Last वाले element का ans. hamesha -1 aaiga

2	1	4	3

4
3
-1

1
-1

1, -1, 3, -1

A) stack -1

B) array K (R→L) element chota store ← milgya ← pop()

C) array k → push

```
Stack<int> s;
s.push(-1);

vector <int> ans (v.size());
for (int i = v.size()-1; i>=0; i--) {
    int curr = v[i];
    while (s.top() >= curr) {      // जब तक chota element
        s.pop();                   // नहीं मिल जाता Pop करते
    }                              // जा.
    ans[i] = s.top();              // chota element मिल चुका है तो ans store
    s.push(curr);                  // push करो current element को.
}
```

ex Previous smaller element

-1	-1	1	1
2	1	4	3

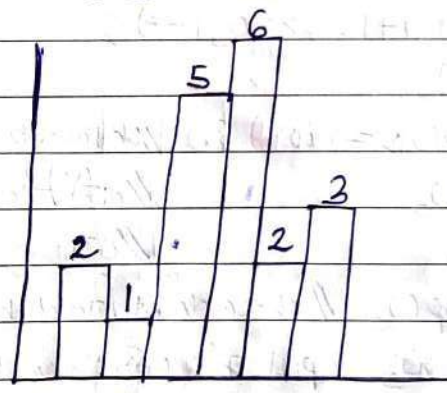
ans: -1, -1, 1, 1

O(n) Time complexity

```
vector <int> prevSmaller( vector<int>& input) {
    stack<int> s;
    s.push(-1);
    vector <int> ans (input.size());

    for( int i=0; i< input.size(); i++) {
        int curr = input[i];
        while (s.top() >= curr) {
            s.pop();
        }
        ans[i] = s.top();
        // push karda curr element ko
        s.push(curr);
    }

    return ans;
```

ex: Largest Rectangular in a histogram

2, 1, 5, 6, 2, 3



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

prev smaller element index

-1	-1	1	2	1	4

next smaller element index

1	-1	4	4	-1	-1

1	6	4	4	6	6

$A_1 = 2$
$A_2 = 6$
$A_3 = 10$
$A_4 = 6$
$A_5 = 8$
$A_6 = 3$

$w_1 = n-p-1$
$2-1$
$= 1 \times 2 \; \boxed{2}$

$w_2 = 6+1-1$
$= 6 \times 1$
$= \boxed{6}$

$w_3 = 4-1-1$
$= 2$
$= \boxed{10}$

$w_4 = 4-2-1$
$= 1 \; \boxed{6}$

$w_5 = 6-1-1$
$= 4 \; \boxed{8}$

$w_6 = 6-4-1$
$= 1 \; \boxed{3}$

```
vector<int> prevsmallerElemet (vector<int>&input){
    stack <int> s;
    s.push (-1);
    vector<int> ans (input.size());
        // left to right smaller
    for (int i=0; i <input.size(); i++){
        int curr = input[i];
        while (s.top() ≠ -1 && input[s.top()] ≥ curr){
            s.pop();
        }
        ans[i] = s.top();
        s.push(i);
    }
    return ans;
}

vector <int> nextsmaller (vector<int>&input>){
    stack <int>s;
    s.push(-1);
    vector <int> ans (input.size());
    for (int i= input.size()-1; i ≥0; i--){
        int curr = input[i];
        while (s.top() ≠ -1 && input [s.top()] ≥ curr){
            s.pop();
        }
        ans[i] = s.top();
        s push (i);
    }
    return ans;
}

int main(){
    vector <int>v;
    v.pushback(2);
    v.pushback(1);
    v.pushback(5);
    v.pushback(6);
    v.pushback(2);
    v.push(back(3);
} cout<<"Ans is" << getRectangularHist
    (v);
```

[Circular Petrol problem]


```
int getRectangleAreaHistogram (vector<int> & height) {
    vector <int> prev = prevSmallelement (height);
    vector <int> next = nextSmallerelement (height);
    int maxArea = INT_MIN;
    int size = height.size();

    for (int i = 0; i < height.size(); i++) {
        int length = height[i];
        if ( next[i] == -1) {
            next[i] = size;
        }
        int width = next[i] - prev[i] - 1;
        int area = length * width;
        maxArea = max (maxArea, area);
    }

    return maxArea;
}
```

~~vector~~ Queues

front FIFO (Ordering)

[traversal
Graphs
Sliding Windows]

insertion → rear side

deletion → front side

```
push(5);
push(6);
push(7);
push(11);
pop();
pop();
```

```
Class Queue {
  public:
  int *arr;
  int size;
  int front;
  int rear;

  Queue (int size) {
    this->size = size;
    arr = new int [size];
    front = 0;
    rear = 0;
  }

  void push (int data) {
    if (rear == size) {
      cout << "Queue is full";
    }
    else {
      arr[rear] = data;
      rear++;
    }
  }

  void pop() {
    if (front == rear) {
      cout << "Queue is empty";
    }
    else {
      arr[front] = -1;
      front++;
      if (front == rear) {
        rear = 0;
        front = 0;
      }
```

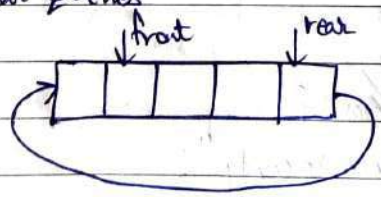
```
int getfront ()
    if (front==rear) {
        cout<< "Queue is empty" << endl;
        return -1;
    }
    else {
        return arr[front];
    }
}
```

```
Queue q(10);
q.push(5);
q.push(15);
q.push(25);
q.push(55);
q.size(); //4
q.pop();
q.size(); //3
q.getfront// 15
```

```
bool isEmpty() {
    if (front==rear) {
        return true;
    }
    else {
        return false;
    }
}
```

```
int getSize() {
    return rear - front;
}
```

↳ Circular Queues

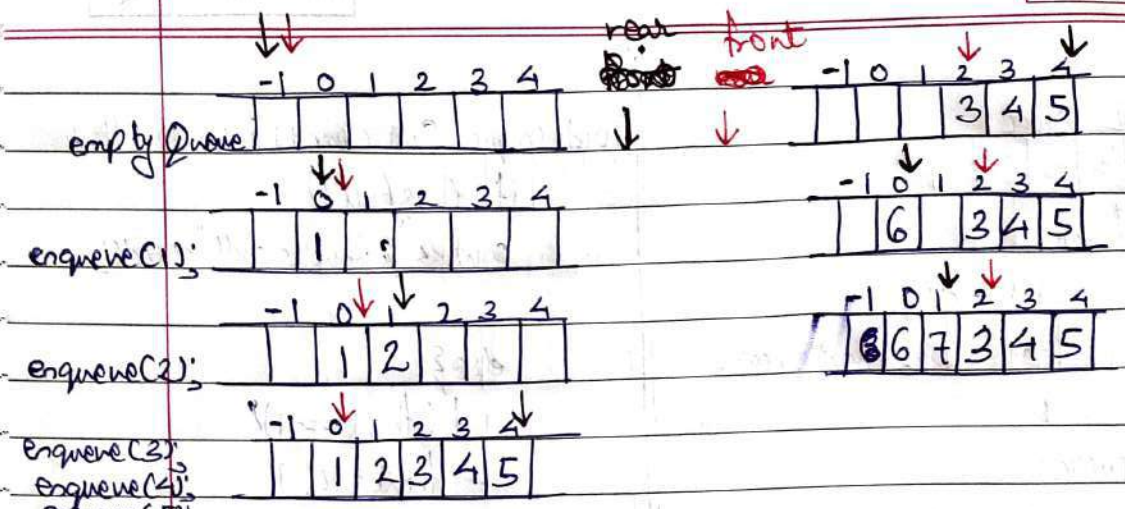


push

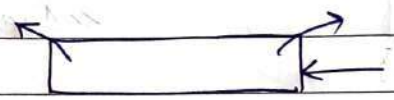
↳ ① if (Q full)
    ↳ cout<< Queue is full
↳ first element f=r=0 insert
↳ circular nature (rear = n+1 && front!=0)
    ↳ rear = 0
↳ default → rear++ , arr[rear] = data

```
class Queue
{
  private:
      int *arr;
      int size, front, rear;

  public:
      Queue (int size)
      {
          this->size = size;
          arr = new [size];
          front = -1;
          rear = -1;
      }

      bool isfull()
      {
          if ( front == 0 && rear = size -1){
              return true;}

          if( rear = front -1){
              return true;}

          return false;
      }

      bool isEmpty(){
          if ( front == -1)
              return true;
          else
              return false;
      }

      void enque (int element) {
          if ( isfull())
          cout << "Queue is full" << endl;

          else {
              if (front == -1){
                front = 0; }
              rear = (rear +1) % size;
              arr [rear] = element;
              cout << endl;
                  << Inserted << element;
      }

      int dequeue {
          int element;
          if ( isEmpty()) {
              cout << "Queue is empty";
              return (-1);
          }
          else {
              element = arr[front];
              if( front == rear) {
                  front = -1;
                  rear = -1; //single element
              }
              else {
                  front = (front +1) % size;
              }
          }
      }
```

rear = (front -1) % (size -1)



i/p restricted Queue

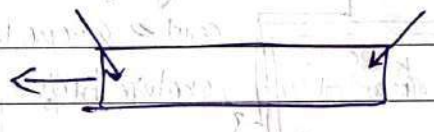


rear → input

pop → front pop-front

→ rear pop-rear

o/p restricted Queue

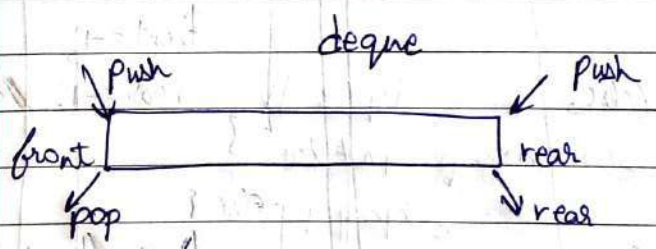


pop → front

push → rear push-rear

→ front push-front.

Doubly ended Queue

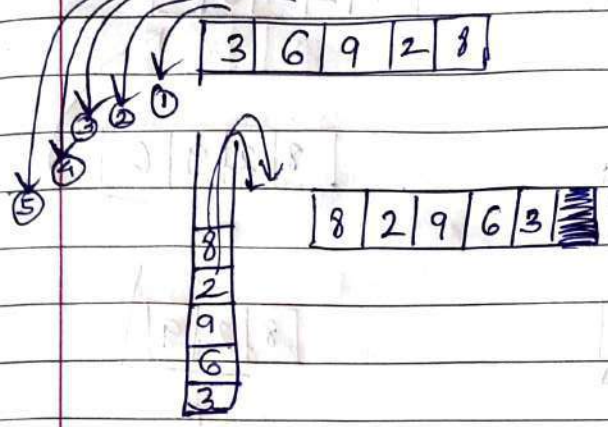


dequeue → pop	Different
enqueue → insert or pop	

push front
pop front
push rear
pop rear

Do it yourself.

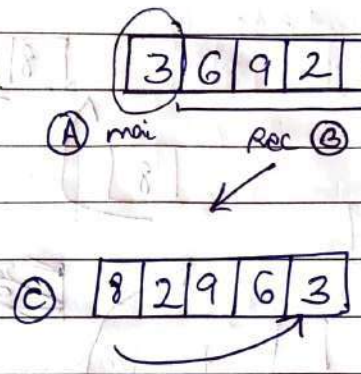
ex Queue Reverse



stacks O(n) TC
O(n) SC

recursion

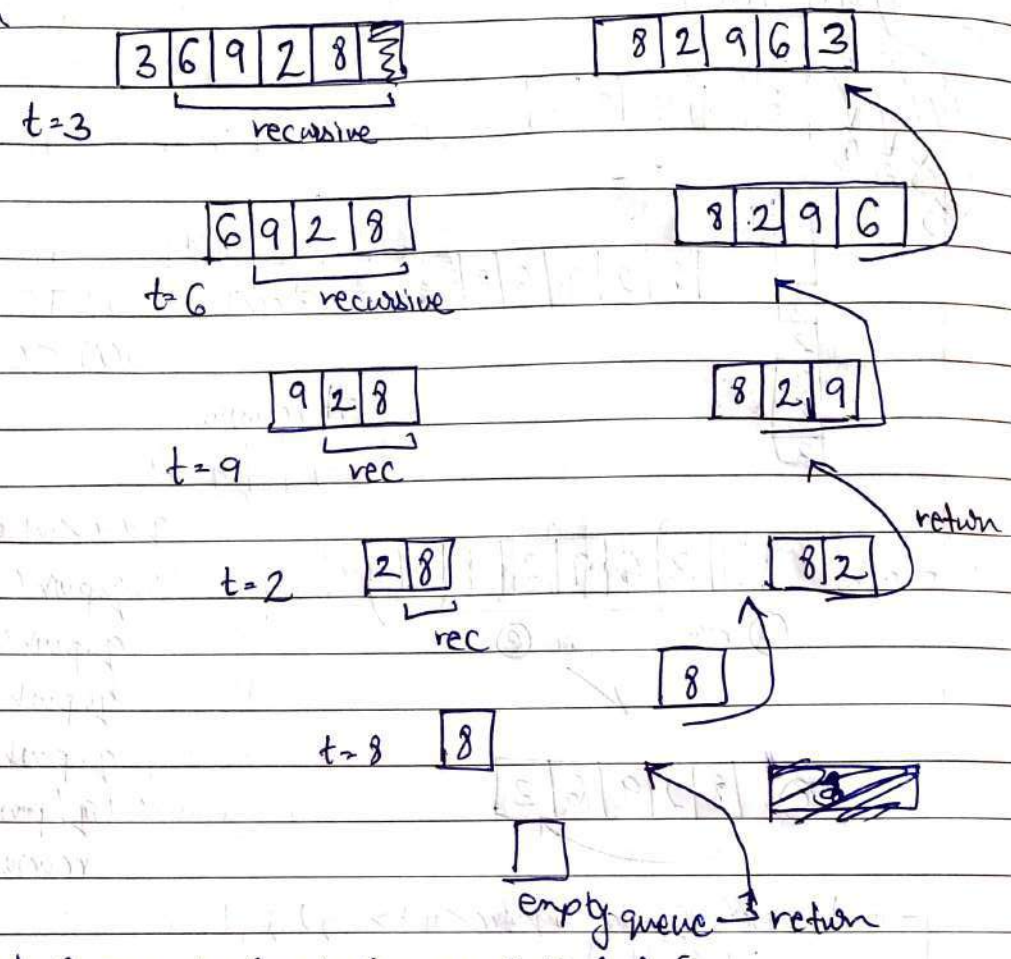
as # stack




```
queue<int q> q;
q.push(3);
q.push(6);
q.push(9);
q.push(2);
q.push(8);
reverseQueue(q);
```

```
void reverseQueue (queue<int> &q) {
    stack<int> s;
    while (!q.empty()) {                    // step1 put all elements of
        int element = q.front();            // q into s
        q.pop();
        s.push(element);
    }

    // step2 put all elements from stack into q.

    while (!s.empty()) {
        int element = s.top();
        s.pop();
        q.push(element);
    }
}
```

ans # recursion




```
void ReverseUsingRecursion(queue<int>&q) {

    // Base case
    if(q.empty())
        return;

    // step A
    int temp = q.front();
    q.pop();

    // step B
    reverseQueueRecursion(q);

    // step C
    q.push(temp);
}
```

Time: O(1)
Space O(1)

ex Reverse first k elements of queue

$k=3$

3	6	9	12	15

n

rev (k) (n-k)

o/p

9	6	3	12	15

9
6
3

(A) k elements stack
(B) stack → queue
(C) (n-k) element pop → push

```
void reverseKC queue<int>&q, int k){   TC = O(n)
    // step A. queue → k elements → stack
    stack <int> s;
    int count = 0;
    int n = q.size();

    if( k==0 || k>n)
        return;

    while (!q.isempty()){
        int temp = q.front();
        q.pop();
        s.push(temp);
        count++;
        if (count==k)
            break;
    }

    // step B. stack → q me qapas
    while(!s.empty){
        int temp = s.top();
        s.pop();
        q.push(temp);
    }

    // step C
    count = 0;
    while (!q.empty()){
        int temp = q.front();
        q.pop();
        q.push(temp);
        count++;

        if count==n-k)
            break;
            return;
    }
}
```

kk n-k!=0

ex Interleave first and second half of Queue.

i/p

10	20	30	40	50	60	70	80

o/p

10	50	20	60	30	70	40	80

10	20	30	40	50	60	70	80

A → take first half of Q into a new Queue.

O(n/2)

10	20	30	40

1 2 5 7

O(n)

50	60	70	80	10	50	20	60	30	70	40	80

2 6 8 1

TC = O(n)

```
void interleaveQueue (queue<int> q) {

    // step A.        if (q.empty)
                          return;
    int n = q.size();
    int k = n/2;
    int count = 0;
    queue<int> q2;

    while (!q.empty) {
        int temp = q.front();
        q.pop();
        q2.push(temp);
        count++;
    }   if (count = k)
            break;

    // step B interleaving
```

```
while(true) {
    if()
        break;
}
```

Bad practice

```
while(! q.empty() && ! q2.empty()) {
    int first = q2.~~firs~~front();
    q2.pop();
    q.push(first);
    int second = q.front();
    q.pop();
    q.push(second);
}
```

n == odd

```
if (n&1) {
    int element = q.front();
    q.pop();
    q.push(element);
}
```

0	1	2	3	4	5	6
10	20	30	40	50	60	70

$\frac{n}{2} = 3$

2nd half: 40 50 60 70

1st half: 10 20 30

q → odd

q.front → back();

IMP

ex First -ve integer in every window of size k

12	-1	-7	8	-15	30	16	28
0	1	2	3	4	5	6	7

k=3

-1, -1, -7, -15, -15, 0

(A) First k size → process

-ve no. → index push

12	-1	-7	8	-15	30	16	28

1	2	

loop (k+1 element → n element)

① q.front() → arr[q.front()] → 0

② remove out of window element.

③ new element insertion

```
#include<deque>
int arr[] = {12, -1, -7, 8, -15, 30, 16, 28}
int size = 8;
solve (arr, size, k);

void solve (int arr[], int n, int k){
    deque <int> q;

    // process first window of size k
    for (int i=0; i < k; i++){
        if (arr[i] <0){
            q.push-back (i);
        }
    }

    // remaining window ko process karo

    for (int i=k; i<n; i++){
        // answer dedo pwai window ka
        if (q.empty()){
            cout << 0<< " ";
        }
        else {
            cout <<arr[ q.front()] <<" ";

        // out of window element ko remove
        while(! q.empty() && (i - q.front() >= k)){
            q.pop-front();
        }
        // check current element for insertion
        if (arr[i] <0){
            q.push-back (i);
```

12	-1	-2	8	-15	30	6	28

1	2

i=4
q.front=1
k=3

i - q-front >= k
3

HW Ex non repeated character in a ~~string~~ stream.
↳ arr
↳ queue

// answer print karo for last window

```
if ( q.empty() ){
        cout << 0 << " ";
}
else{
        cout << arr[q.front()] << " ";
}
```

0	1	2	3	4
-8	2	3	-6	10

k = 2

~~0~~	3

TC: O(n)
SC: O(k)
O(1) also

-8, 0, -6, -6
last answer

first non-repeating character in a stream (L to R).

i/p a a b c
o/p a # b b

i/p a a c
o/p a # c

i/p a a r c a a a r d
~~o/p 2 2 2 2 2 2 #~~
o/p 2 2 2 2 2 2 r c c

to track the elements in a stream so that the frequency of elements in the stream ~~seem~~ remain only once to get accepted or rejected in the case of multiple occurances.

Queue
↓
track
↓
FIFO

i/p [a] a b c

→ freq / cont ++ → array
→ q.push(ch);

q.front → character
repeating → pop()
non repeating → (print)

q.empty() → #

```
string str="aabc";
int freq[26]={0};
queue<char> q;
string ans="";

for (int i=0; i<str.length(); i++) {
    char ch = str[i];

    //increment the frequency
    freq[ch-'a']++;
    q.push(ch);

    while(!q.empty){
        if (freq[q.front()-'a']>1){
            q.pop();
        }
        else {
            ans.push_back(q.front());
            break;
        }
    }
    if (q.empty()) {
        ans.push_back('#');
    }
}

cout<"Final answer"<< ans;
```

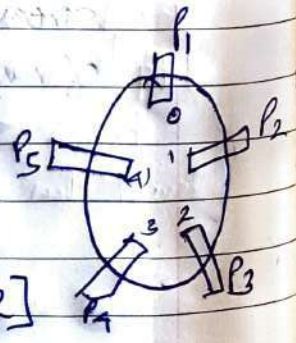
(queue) Inlet and outlet

TC: O(n)

SC: O(n) queue (space)

imp ex

Circular tour / Gas station (petrolpump)

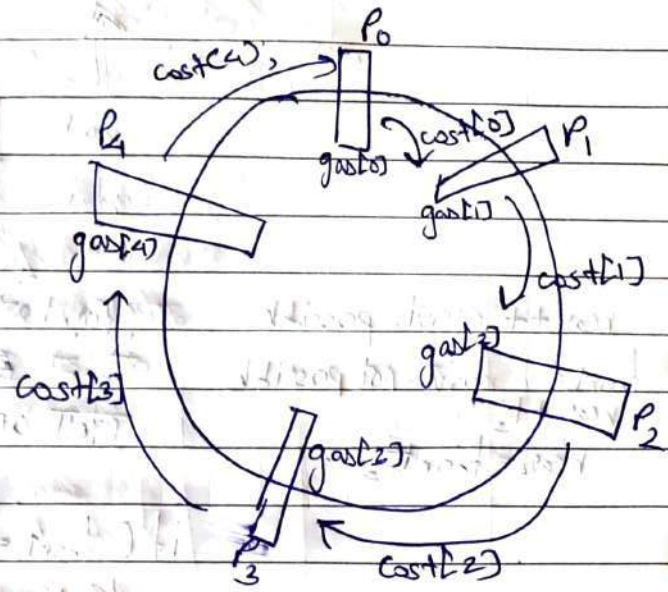


petindex:

gas [1, 2, 3, 4, 5] cost [3, 4, 5, 1, 2]

	0	1	2	3	4
gas →	1	2	3	4	5
cost →	3	4	5	1	2

1 litre → 1 km



balance = 1 ✓ 0th index
dist = 3 ✗

balance = 2 ✓ 1st index
dist = 4 ✗

balance = 3 2nd index
distance = 5 ✗

balance = 4 3rd index
dist = 1 ✓

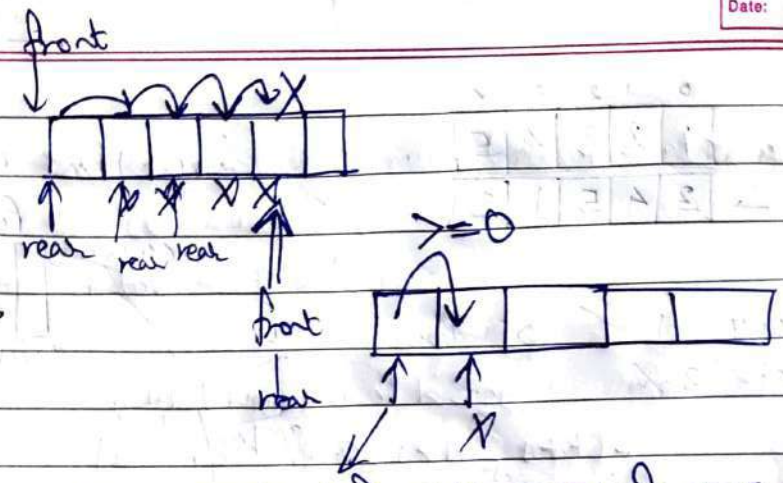
#1 2 loop.
$O(n)^2$

↓

at P_5
balance = 3 + 5 = 8
dist = 2
↓ P_0 at
balance = 6 + 1 = 7
dist = 3
↓
at P_1 balance 7 − 3 + 2 = 6
dist = 4
↓ at P_2
balance = 6 − 4 + 3 = 5
dist = 5
↓ at P_2
balance = 0
dist = 1

3 valid index

#2



rear++ movt. possible
front= movt. not possible
rear+1;
rear=front;

इस point से start करकर भी तुम आगे नहीं जा पा रहे हो इसका Petrol add करकर भी तो तुम इसके बिना जाने की सोच भी कैसे सकते हैं।

if (front==rear)
↳ 1 circle complete

#3

1	2	3	4	5

3	4	5	1	2

deficit = gas (p-d) balance

balance >= deficit
No → Circle not possible
Yes → circle possible

i.e.

gas	4	6(+7)	3(+1-4)	4	8(+3)
cost	3	6	7	1	3

3 litre deficit

balance - 8 litres of petrol

balance > deficit
So they will travel

```
int canCompleteCircuit (vector<int>& gas, vector<int>& cost){

    // kitna petrol kam hai.
      int deficit=0;
    // kitna petrol bacha hua hai.
      int balance=0;
    // circuit kaha se start karni hai.
      int start=0;

      for (int i=0; i<gas.size(); i++){
          balance += gas[i] - cost[i];
          if (balance <0){
              deficit += balance;   // saarae deficit ko add karo
              start = i+1;
              balance=0;
          }
      }

      if (deficit + balance >=0) {
          return start;
      }
      else {
          return(-1);
      }
}
```

for loop
TC = O(1) only 1
SC = O(1)

imp
Hard
ex

Sliding Window Maximum

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

k=3

~~1~~	~~3~~	~~-1~~					

कुछ नहीं पता होगा या उससे ज्यादा element पड़े हुए होंगे

index	1	2				

```
vector <int> maxSlidingWindow (vector <int> & nums, int k) {
    deque <int> dq;
    vector <int> ans;

    //first window of k size
    for (int i=0; i<k; i++) {
        //chote elements ko remove karo
        while (!dq.empty() && nums[i] >= nums[dq.back()]) {
            dq.pop_back();
        }
        //inserting index, so that we can checkout of wind
          -ow element
        dq.push_back(i);
    }

    //store answer for first window
    ans.push_back (nums[dq.front()]);

    // remaining windows ko process
    for (int i=k; i< nums.size(); i++) {
        //out of window element ko remove kardo
        if (!dq.empty() && i-dq.front() >= k) {
            dq.pop_front();
        }

        // ab fresh current element k liyae chote element
        // ko remove kra h.
        while (!dq.empty() && nums[i] >= nums[dq.
              back()]) {
            dq.pop_back();
        }

        //inserting index, so that we can checkout of
          window element
```

Hard ex max no. of task you can assign

```
dq.push_back(i);

// current window ka answer store karna hai.
ans.push_back[nums[dq.front()]];
}
}
}
```

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

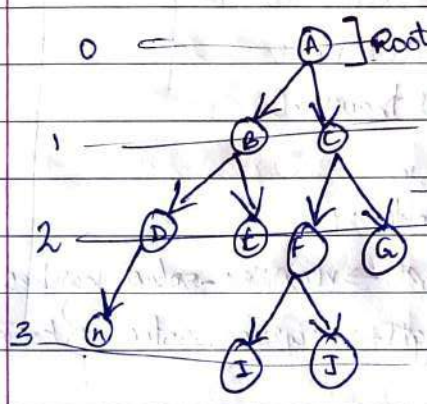
k=3

dq: 0 (1), 1 (3), 2 (-1), 3 (-3), 4 (5), 5 (3), 6 (6), 7 (7)

ans: 3 3 5 5 6 7

Trees.

Class-1. It is a non linear data structure in which data is stored in hierarchical form.

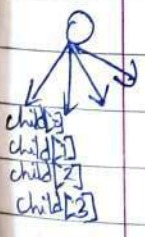


Linked List — Linear

binary tree

```
class Node {
    int data;
    Node* left;
    Node* right;
}
```

Terms
- ↳ nodes
- ↳ root
- ↳ parent
- ↳ child
- ↳ ~~anchor~~ ancestor
- ↳ descendent
- ↳ sibling
- ↳ leaf




```
class Node
{ int data;
  vector <Node*> child;
};
```

C's sibling F, G
F's sibling I, J
B's sibling D, E

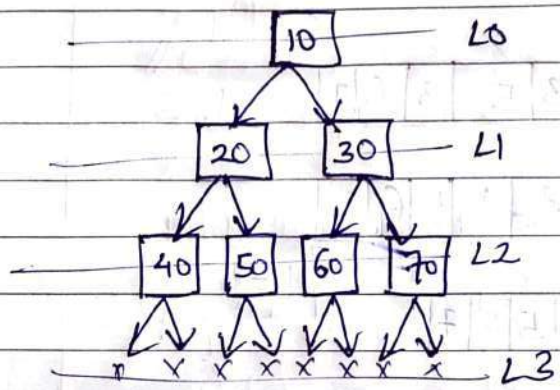
A is a parent of B, C
B is a parent of D, E.
Ancestor of I are F, C, A.
Decedent of C are F, G, I, J

Trees

Interviews

Online test.
- MCQ → (formulas)
- Coding Questions

UBER
↳ Trees, heap/Map



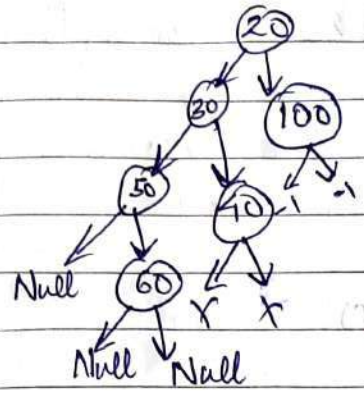
Level Order Traversal
Inorder/Preorder/Postorder
Morris Traversal

Level Order Traversal:
10 20 30 40 50 60 70

Building Tree.
1. Create tree using level order traversal
2. Recursively
   - (A) Root, Create Node
   - (B) Root → left → recursion solve karke लाऊंगा
   - (C) Root → right → recursion solve karke लाऊंगा

i/p assum
!= -1 valid data
== -1 invalid data

20 30 50 -1 60 -1 -1 40 -1 -1 100 -1 -1

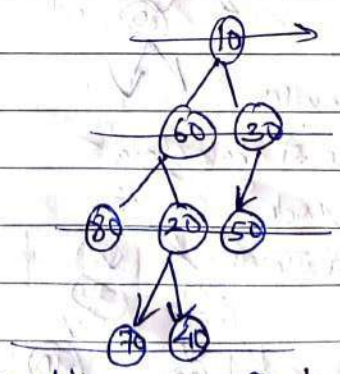



```
class Node {
    public: int data;
            Node* left;
            Node* right;
    Node(int data) {
        this -> data = data;
        Left = NULL;
        right = NULL;
    }
};
```

```
Node* buildTree() {
    int data;
    cout << "Enter data";
    cin>>data;
    if (data==-1) {
        return NULL;
    }

    Node* root=new Node(data);
    root->left = buildTree();
    root->right = buildTree();
    return root;
}
```

```
int main() {
    Node* root = NULL;
    root= buildTree();
}
```

Level Order Traversal.



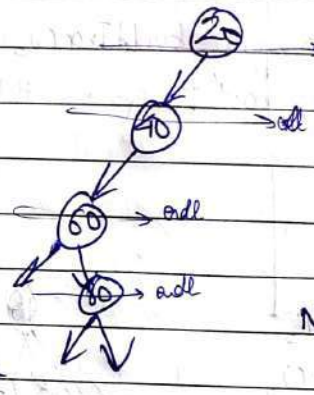
ans: 10, 60, 30, 80, 20, 50, 70, 40

q.push(root);

(A) temp = q.front();
(B) q.pop();
(C) print-ans
(D) child-insert

```
void levelordertraversal(Node* root){
    queue<Node*> q;
    q.push(root);
    while(!q.empty()){
        // A
        Node* temp = q.front();
        // B
        q.pop();
        // C
        cout << temp->data << " ";
        // D
        if(temp->left){
            q.push(temp->left);
        }
        if(temp->right){
            q.push(temp->right);
        }
    }
}
```

i/p: 20, 40, 60, -1, 80, -1, -1, -1, -1

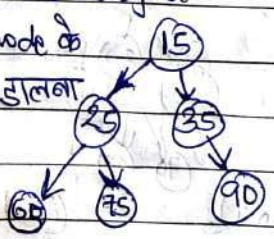


o/p: 20 40 60 80

62 5'00 level के बाद endl chahiye

NULL (1) Pura level complete ho chuka hai.

Null जैसे ही encounter हो, next line में chahe jana chahiye aur us node के सारे child queue में डालना होता है।



Root node is always NULL

(2) Push NULL condition

!q.empty()

15 NULL
25 35 NULL
65 75 90 NULL

15	NULL	25	35	NULL	60	75	90	NULL

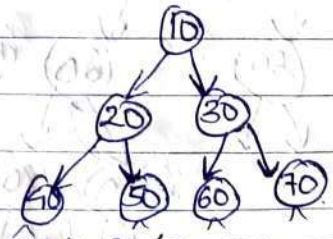
```
void level traversal (Node *root) {
    queue <Node*> q;
    //initially
    q.push(root);
    q.push(null);
    while (!q.empty()) {
        Node* temp = q.front();
        q.pop();
        if (temp == NULL) {
            cout << endl;
            if (!q.empty()) {
                q.push(NULL);
            }
        }
        else {
            cout << temp->data << " ";
            if (temp->left) {
                q.push (temp->left);
            }
            if (temp->right) {
                q.push(temp->right);
            }
        }
    }
}
```

15
25 35
65 75 90

Inorder / Preorder / Post order

LNR — NLR — LRN

left, Right, current node



inorder: 40, 20, 50, 10, 60, 30, 70

preorder: 10, 20, 40, 50, 30, 60, 70

Postorder: 40, 50, 20, 60, 70, 30, 10

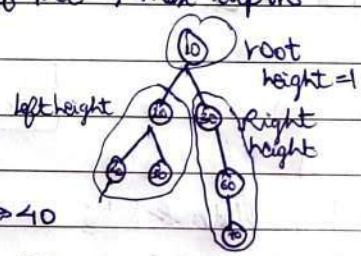
① Inorder, Pre order, Post order] Iterative approach using loops

② Construct tree using level order traversal

```
void inordertraversal (Node * root) {
    //base case
    if (root == NULL) {
        return;
    }

    //LNR
    inordertraversal (root->left);   ①
    cout << root->data << " ";       ②
    inordertraversal (root->right);  ③
}
```

Preorder	Post Order
2	1
1	3
3	2

ex Find height of tree / max depth



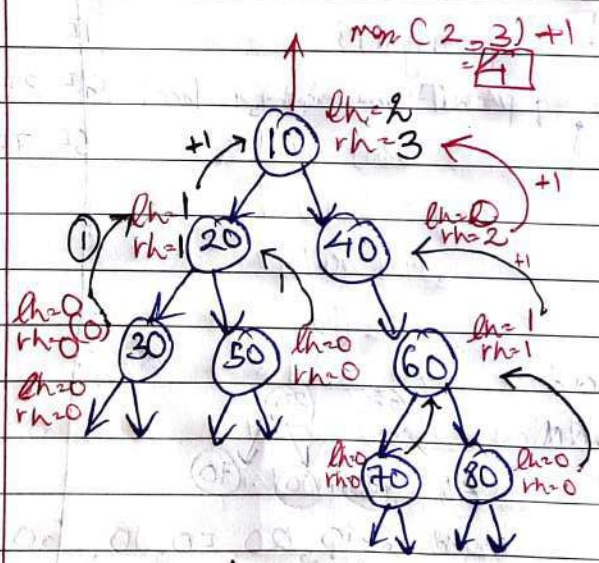
No. of nodes from root node to the farthest leaf node

10 → 20 → 40

10 → 20 → 50

$\boxed{10 \to 30 \to 60 \to 70}$

```
int height (Node * root) {
    if (root == NULL)
        return 0;
    int leftheight = height (root->left);
    int rightheight = height (root->right);
    int ans = max (leftheight, rightheight);
    return ans;
}
```



ex Diameter of tree.

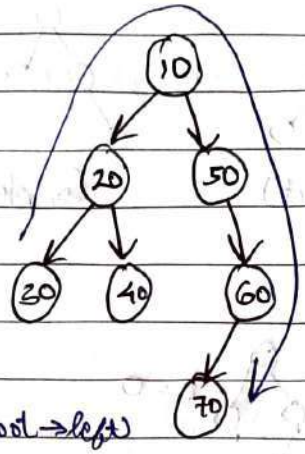
Diameter of a binary tree is the length of the longest path between any two nodes in a tree.

The path may or may not pass through the root.

ex check if two trees are identical or not.

ex check if two trees are mirror of each other or not.

ex Balanced Binary tree (in every node height → abs(difference <=1))



diameter = 5

opt 1) diameter(root→left)

opt 2) diameter (root→right)

opt 3) diameter through root.

height of left + height of right + 1

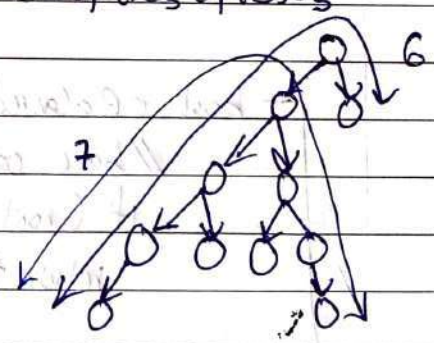
for root node

Max.

imp

for edges no need to add this 1

```
int diameterofBinarytree (TreeNode * Node){
    if (root==NULL)
        return 0;

    int opt1 = diameterofBinaryTree ( root→left);
    int opt2 = diameterofBinaryTree ( root→right);
    int opt3 = maxdepth(root→left) + maxdepth (root→right);
    int ans = max (opt1, max (opt2, opt3));
    return ans;
}
```



↳ Classe Trees.

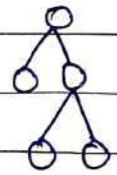
ex check if a binary tree is $\boxed{\text{balanced}}$ or not.

each node

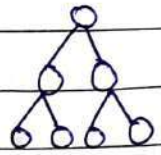
left subtree height — right subtree height

absolute difference

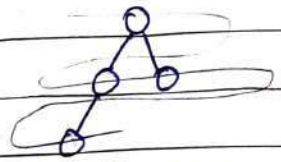
$\boxed{K=1}$



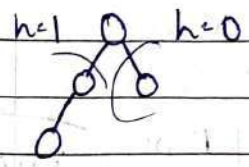
Binary tree



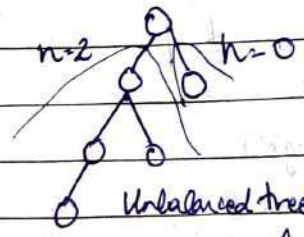
Fully (perfect) Binary tree



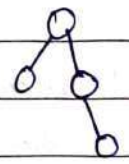
Complete binary tree



Balanced tree
The diff. betⁿ the heights of left subtree and right subtree is 1 (at most) in $\boxed{\text{EVER node}}$



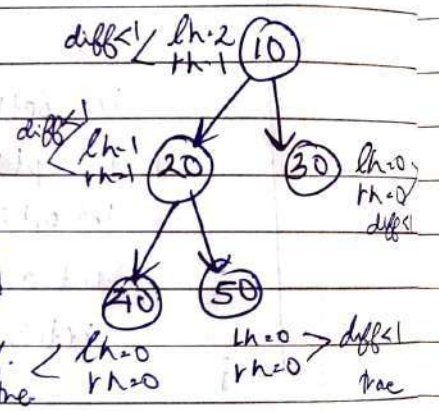
Unbalanced tree
As difference betⁿ height of left subtree and right subtree is 2.



Balanced but not Complete

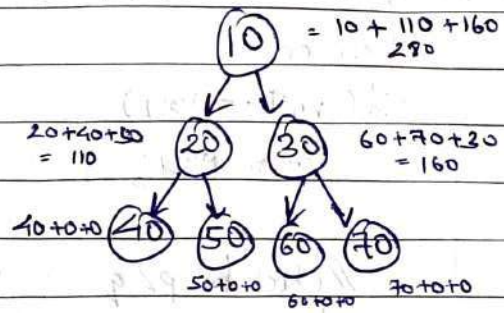
every node

LT height → diff. <=1 ← RT height




```
bool isBalanced(TreeNode *root){
    //base case
    if (root == NULL)
        return true;

    //1 case
    int leftheight = height(root->left);
    int rightheight = height(root->right);
    int diff = abs(leftheight - rightheight);
    bool ans1 = (diff <= 1);
    //recursion
    bool leftans = isBalanced(root->left);
    bool rightans = isBalanced(root->right);
    if(ans1 && leftans && rightans)
        return true;
    else
        return false;
}
```

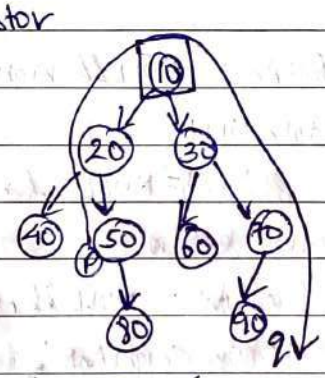
Ex. Convert into sum tree




```
int convertintoSumtree(Node* root)
{ // BC
    if( root == NULL)
        return 0;
    int left = convertintoSumtree (root->left);
    int right = convertintoSumtree ( root->right);
    root->data = left + root->data + right ;
    return root->data;
}
```

Imp Microsoft.

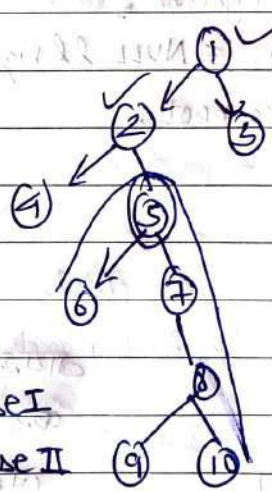
Ex Lowest common ancestor



p = 50

q = 90

common ancestor = 10

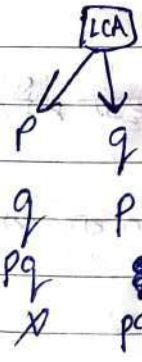


p = 6

q = 10

lowest common ancestor 5, 2, 1

lowest common ancestor



Case I

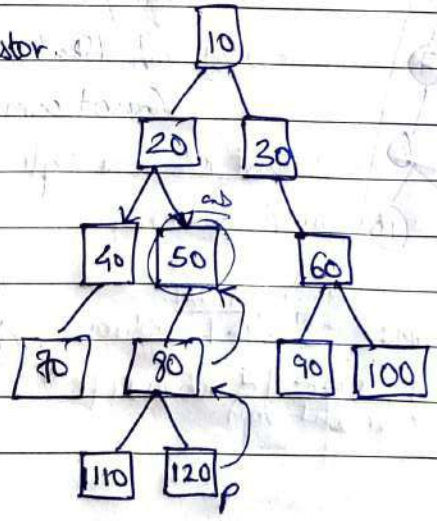
Case II

Case III

Case IV

```
TreeNode * lowestcommonancestor (TreeNode *root, TreeNode *p, TreeNode *q){
    //base case.
    if(root == NULL)
        return NULL;

    //check for p & q
    if (root->val == p->val)
        return p;
    if(root->val == q->val)
        return q;

    TreeNode* leftAns = lowestCommonAncestor(root->left, p, q);
    TreeNode* rightAns = lowestCommonAncestor(root->right, p, q);

        if(leftAns == NULL && rightAns == NULL)
            return NULL;
        else if (leftAns != NULL && rightAns == NULL)
            return leftAns;
        else if (leftAns == NULL && rightAns != NULL)
            return rightAns;
        else (leftAns != NULL && right_Ans != NULL){
            return root;
}
```

ex K^{th} ancestor.

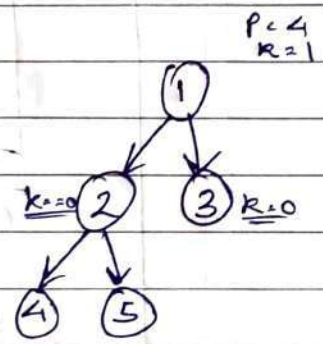


k=2

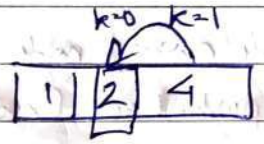
बायस जाते हुए k को minus (-) कर देना

जहा k==0 वह ans है।

```
bool
k^th Ancestor (Node* root, int& k, int p)
{ //BC
    if (root == NULL)
        return false;
    if (root->data == p)
        return true;
bool left = k^th Ancestor (root->left, k, p);
bool right = k^th Ancestor (root->right, k, p);

    if (length == true || right == true)
        k--;
    if (k==0) {
        k=-1;
        cout << root->data;
    }
    return left || right;
}
```

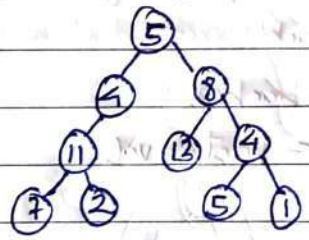


using vector



push_back() pop back() k times

ex Path sum II



All root → leaf

1 5 → 4 → 11 → 7 = 27

2 5 → 4 → 11 → 2 → 22

3 5 → 8 → 13 → 26

4 5 → 8 → 4 → 5 = 22

5 5 → 8 → 4 → 1 = 18

```
vector <vector<int>> pathSum (TreeNode* root, int targetSum) {
    vector <vector<int>> ans;
    int sum = 0;
    vector <int> temp;
    solve (root, targetSum, sum, temp, ans);
    return ans;
}
```

ex path from any node to any other node

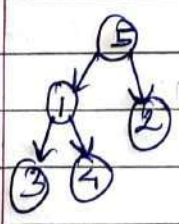
ex sum of longest bloodline of BT. target-sum = 70

```
void solve(TreeNode * root, int targetSum, int currSum, vector<int> path,
        vector<vector<int>>& ans)
{
    if(root == NULL){
        return;
    }

    // leaf node
    if(root->left == NULL && root->right == NULL){
        path.push_back(root->val);
        currsum += root->val;
        if(currsum == targetsum){
            ans.push_back(path);
        }
        path.pop_back();
        currSum -= root->val;
        return;
    }

    // include current node
    path.push_back(root->val);
    currsum += root.val;
    solve(root->left, targetSum, currSum, path, ans);
    solve(root->right, targetSum, currsum, path, ans);
    // backtrack
    path.pop_back();
    currsum -= root->val;
}
```

Class 3 started for 10 seconds and was ended

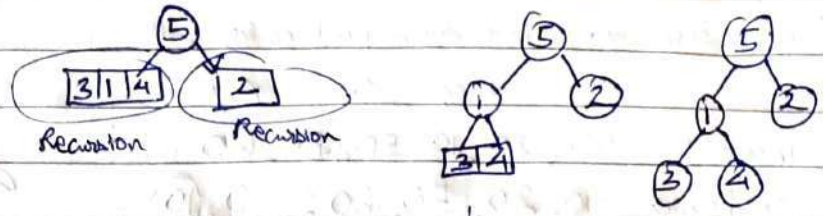
Class 4.

i/p: inorder traversal — construct a tree
preorder traversal — or tell what would be the post order traversal.



Left Root Right
inorder 3 1 4 5 2 LNR

Preorder 5 | 3 4 2 NLR
Root Left Right



inorder: 3 1 4 5 2 (i-1, i, i+1)
pre order: 5 1 3 4 2

```
Node * buildTreefrompreorderInorder (int
inorder[], int preorder[], int size, int
& preindex, int inorderStart, int inorderend) {
```

ek bar preindex agar आ गया तो वो Piche नहीं आ सकता

Preindex में एकबार हम आगे बढ़ गए हैं तो उसकी node already बन चुकी है। तो हमें वापस पीछे नहीं जाना है।

```
// base case
if ( preorderIndex >= size || inorderStart > inorderend){
    return NULL;
}

// step A.
int element = preorder[ preorder++];
Node * root = new Node (element);
int pos = findelement (inorder, size, element);
```

position

```
// step B  root -> left solve
root->left = buildTreefrompreorderInorder (inorder, preorder, size,
    preindex, ~~inordersize~~ inorderstart, pos-1);

// step C  root -> right solve
root -> right = buildTreefrompreorderInorder (inorder, preorder, size,
    preindex, pos+1, inorderEnd);

return root;
}
```

Homework. Replace this with map to make it O(1). O(n)

```
int findposition (int arr[], int n, int
    element) {
    for (int i=0; i<n; i++) {
        if ( arr[i] == element) {
            return i;
        }
    }
    return -1;
}
```

HW
Preorder
Postorder and Then build tree

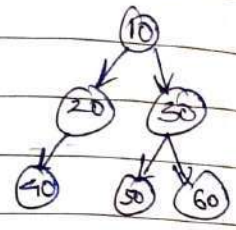
ex Create tree using inorder & post order

Left Root Right

inorder: 40, 20, 10, 50, 30, 60

postorder: 40, 20, 50, 60, 30, 10

Left Right Root. Root



Right to left

```
Node * buildTreefrompostorderInorder(int inorder[], int postorder[], int
postindex, int size, int inorderstart, int inorderend) {
```
→ unordered_map<int, int> & mapping)

```
    //base case
    if(postindex <0 || inorder > inorderend) {
        return NULL;
    }

    int element = postelement[ postindex--];
    Node * root = newNode(element);
    int position = find postion(inorder, size, element);  replace
    int position = mapping[element];   // O(1).
    //root right solve
    root->right = buildTreefrompostorderInorder(inorder, postorder,
      postindex, size, pos+1, inorderend);

    //root left solve;
    root->left = buildTreefrompostorder(inorder, postorder, postindex,
      size, inorderstart, pos-1);

    return root;
}
```

```
int findposition(int arr[], int n, int elem
    for(int i = 0; i <n; i++) {
        if (arr[i] == element){
            return i;
        }
    }
}
```

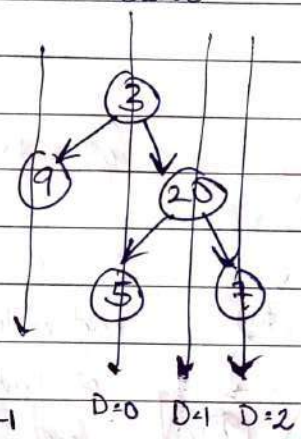
HW/ Boundary
Top/left, Bottom, Right

// in the int main function we will create a map beforehand ~~after~~ before making call for the ~~conversion~~ building of tree from preorder inorder

```
void CreateMap (unordered_map<int, int> & mapping, int inorder[], int n){
    for (int i=0; i<n; i++){
        mapping [inorder [i]] = i;
    }
}
```

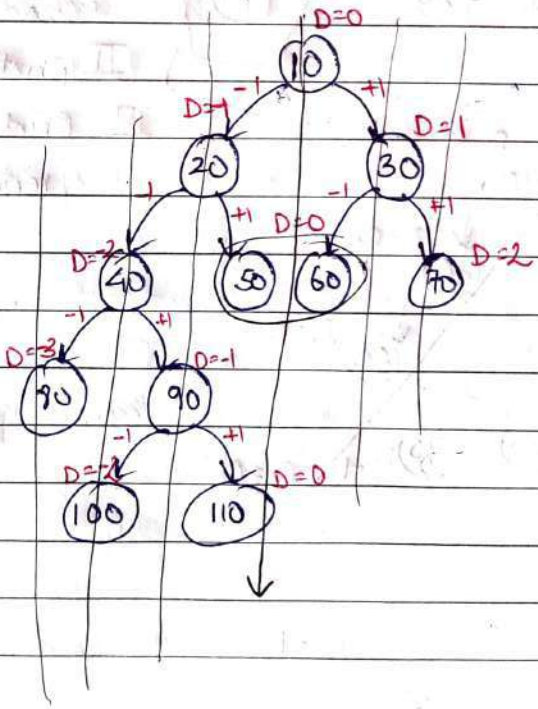
value index

eg Vertical order traversal

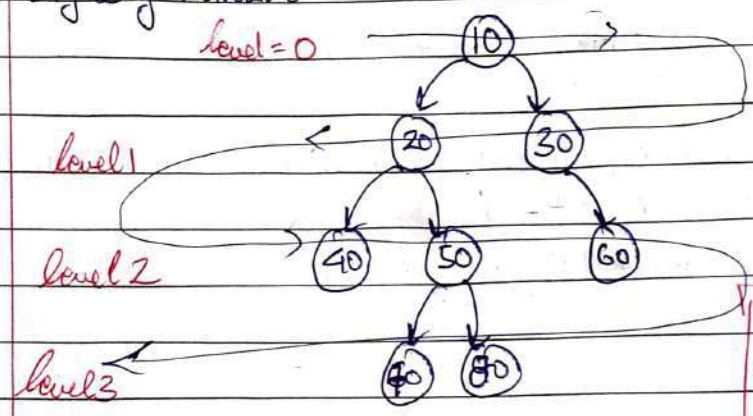


Same vertical line ka Distance same होगा

9, 3, 15, 20, 7



ex Zig Zag traversal

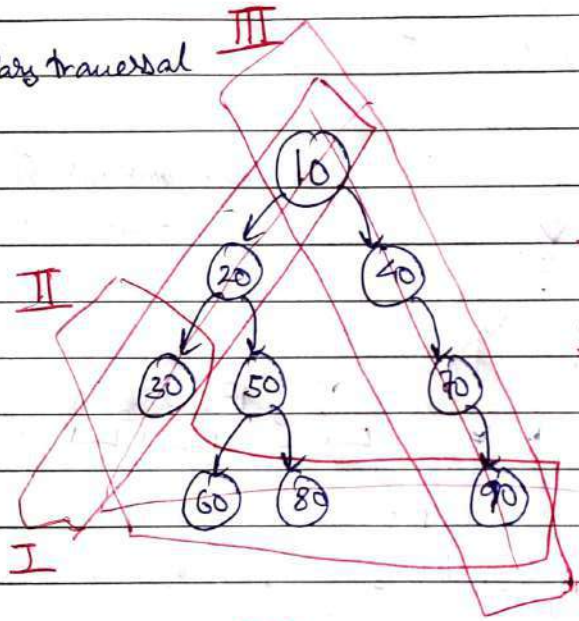


10, 30, 20, 40, 50, 60, 80, 70

even level L to R
odd level R to L

queue
even right then left
odd left then right

ex Boundary traversal



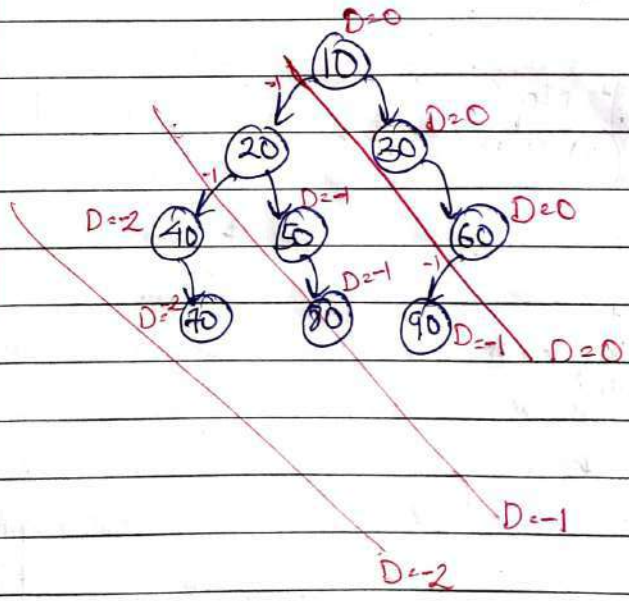
left nodes + left view + right

I
root node से start करो nodes को Print karte चलो जब तक leaf node ना मिले

II printing leaf nodes

III print should follow after recursion.

ex

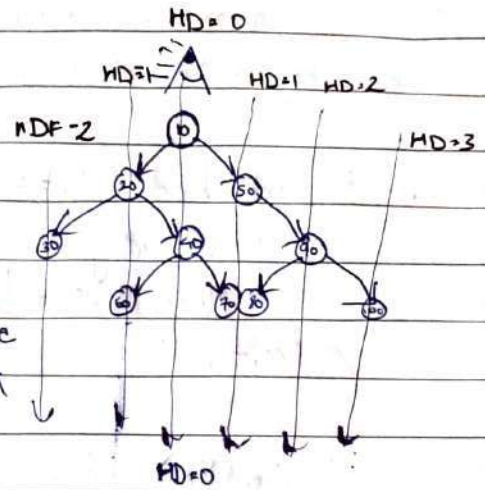


view [top, right, left, bottom] are related to levels or distances

imp

Trees class 5

eg Top view



O/p

30, 20, 10, 50, 90, 100

जब भी कोई नया horizontal distance आएगा तो उसको मैं save कर लूंगा

preorder N L R

0 10

-1 20

-2 30

| 1 | 70 | fault

2 50

3 100

pre order ✗

using level order traversal

0 → 10

-1 → 20

1 → 50

-2 → 30

2 → 90

3 → 100 ✓

```
void printTopView(Node* root){
    if (root==NULL)
        return;

    // create a map for storing topview's data on the basis of HD
    map<int, int> topNode;

    // level order
    // we will store a pair consisting of Node and horizontal distance
    queue <pair <Node*, int>> q;
    q.push(make_pair(root, 0));
```

level order traversal

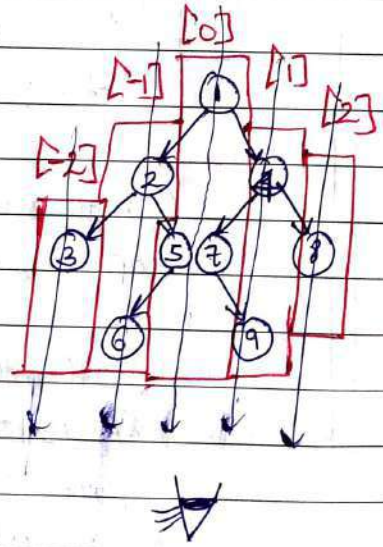
```
    while (!q.isempty()){
        Pair<Node*, int> temp = q.front();
        q.pop();
        Node* frontNode = temp.first;
        int hd = temp.second;
```

```
// jo bhi horizontal distance aaya hai, check if answer for that
   hd (horizontal distance) already exists or not.

if( topNode.find(hd) == topNode.end()) {   // no need for this in case of bottom view
    // create entry
        topNode[hd] = frontNode -> data;
}   // omit.

    if(frontNode -> left)
        q.push(make_pair(frontNode -> left, hd-1));

    if(frontNode -> right)
        q.push(make_pair(frontNode -> right, hd+1));
}
}
```

ex Bottom View.

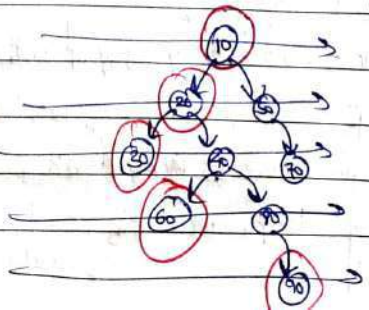
o/p: 3, 6, 7, 9, 8

0 → ~~1~~ 5 7
-1 → ~~2~~ 6
+1 → ~~4~~ 9
-2 → 3
2 → 8



only remove the condition in the top view to get the bottom view

Left view.

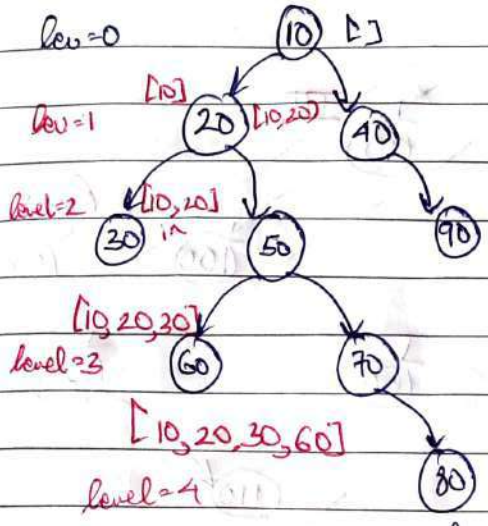


o/p: 10, 20, 30, 60, 90

#1
Through Level order traversal
↳ Homework

#2 Recursion.

* new level → ans store
  if ans exist for a level
  do not up date it.
* Root Node is always left view.



no. of element = 4
so insert [10, 20, 30, 60, 80]

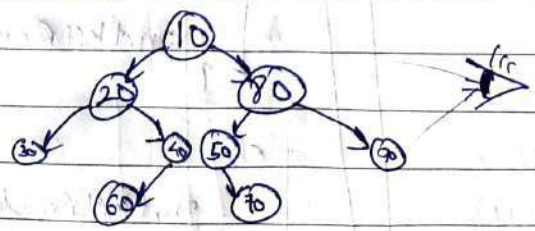
imp where no. of elements matches no. of levels so insert it.

```
void printleftTree (Node* root, vector<int>&ans, int level) {
    //base case
    if (root == NULL)
        return;

    if (level == ans.size()) {
        ans.push_back(root->data);
    }

    //left
    printleftTree (root->left, ans, level+1);
    //right
    printleftTree (root->right, ans, level+1);
}
```

interchange both of them to get the right view

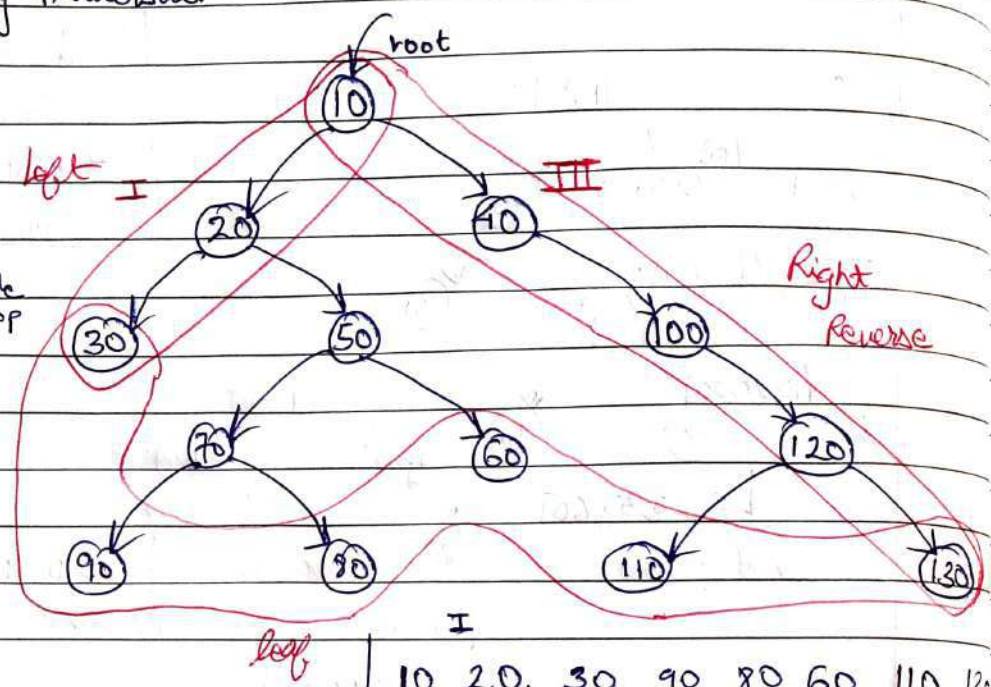
imp Rec

Right view



o/p: 10, 80, 90, 70

ex Boundary traversal.



NLR

leaf node
Par stop

I

10 20, 30 90 80 60 110 130

leaf II

120, 100 40,

right III

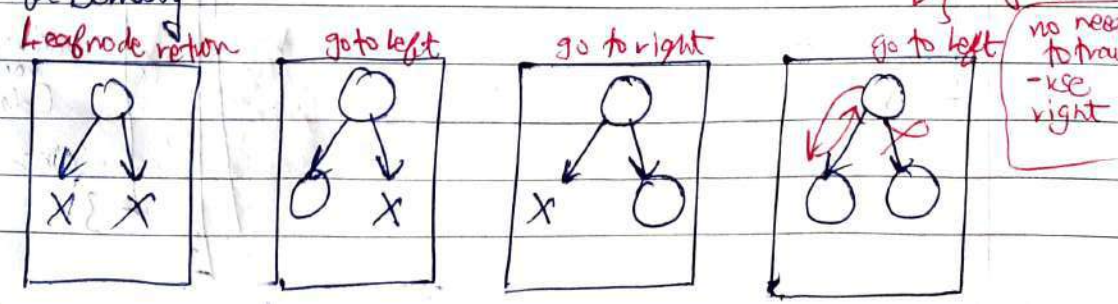
```
void boundaryTraversal (Node * root) {
    if (root == NULL)
        return;
        cout << root -> data << " ";
    // A
    printleftBoundary (root -> left);        O(n)
    // B
    printleafBoundary (root);                O(n)
    // C
    PrintRightBoundary (root -> right);      O(n)
}
```

TC = O(n) + O(n) + O(n) = O(

```
void printleftBoundary (Node * root){
    // base case if root is null, go
    // back or when root is a leaf node
    go back
    if (root == NULL)
        return;
    if (root -> left == NULL &&
        root -> right == NULL)
        return;
    cout << root -> data << " ";
    printleftBoundary (root -> left);
    printleftBoundary (root -> right);
    if (root -> left) {
        printleftBoundary (root -> left)
    }
    else {
        printleftBoundary (root -> right)
    }
}
```

```
void printRightBoundary(Node *root){
    //base case
    if( root==NULL)
        return;
    if( root->left==NULL && root->right==NULL)
        return;

    if (root->right){
        printRight~~Bottom~~Boundary (root->right);
    }

    else{
        printRightBoundary(root->left);
    }
}

void printLeafBoundary (Node *root){
    //base case
    if(root== NULL)
        return;
    if (root->left==NULL && root->right==NULL){
        cout << root->data << " ";
    }

    printLeafBoundary(root->left);
    printLeafBoundary(root->right);
}
```

Left Boundary



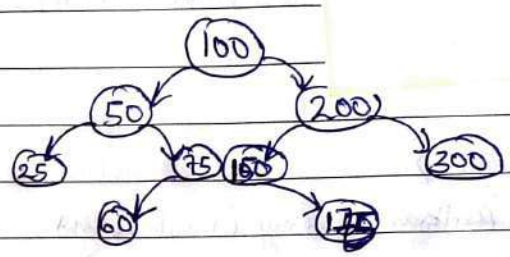
Inorder traversal of Binary Search Tree always gives us sorted order.
imp

(BST) Binary Search Tree.

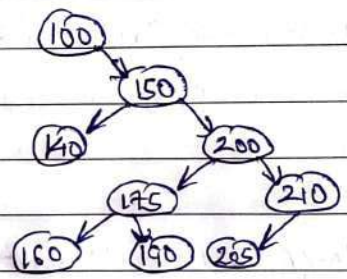
for every node

root → data > left.sw

root → data < right.s

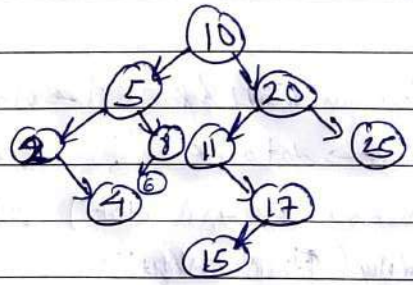


root = NULL i/p: 100, 150, 200, 175 160, 140, 210, 205, 190



i/p: 10 20 5 11 17 2 4 8 6 25 15 -1

-1 कहने तक जाएगी।




```
int main() {
    Node * root = NULL;
    Cout << " Enter the data for Node" << endl;
    takeInput();
    return 0;
}
```

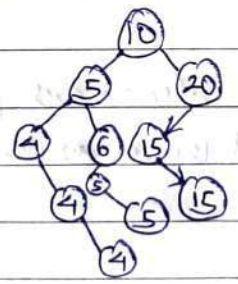
Node * & root

```
void takeInput() {
    int data;
    cin >> data;
    while ( data != -1) {
        root = insertIntoBST
            root,
            (data);
        cin >> data;
    }
}
```

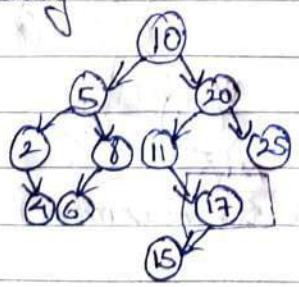
```
Node * insertIntoBST (Node * root, int data){
    if (root == NULL) {
        //This is the first node we have to create
        root = newNode (data);
        return root;
    }

    // this is not a first node
    if (root->data > data){
        //insert in left
        root->left = insertIntoBST (root->left, data);
    }
    else {
        //insert into right
        root->right = insert into BST (root->right, data);
    }
    return root;
}
```

~~Searching~~

i/p: 10 20 15 5 6 4 4 4 55 15



Searching



target: 17

TC: O(h)

h - height

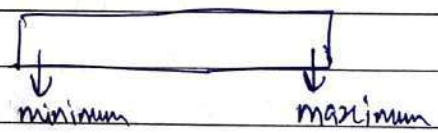
worst case: O(n)

avg case height: O(log n)

```
// assuming that there are only unique values in trees
bool findNodeinBST(Node* root, int target){
    // base case
    if(root==NULL){
        return false;
    }

    if(root->data==target)
        return true;

    if(target > root->data) //right subtree me search karo
        return findNodeinBST(root->right, target);

    else{
        return findNodeinBST(root->left, target);
    }
}
```

ex Minimum value in BST

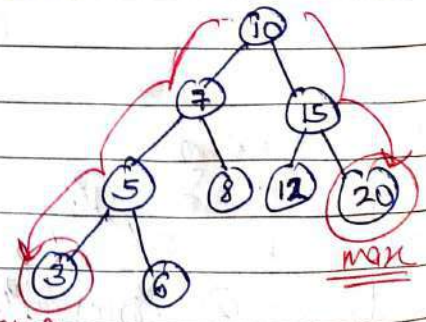
#1 inorder: sort



#2 Recursion

#3 Iterative (take left whenso ever possible and when it is not possible it is your answer).

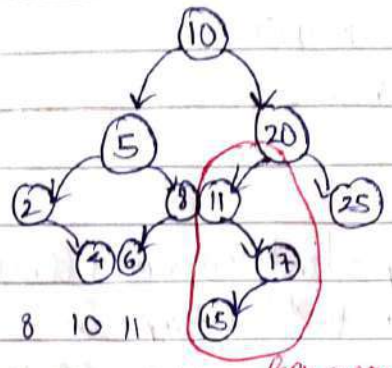
```
int minVal(Node* root){
    Node* temp = root;
    if(temp == NULL){
        return -1;
    }

    while(temp->left != NULL){
        temp = temp->left; //(right for maximum)
    }
    return temp->data;
}
```



Amazon question: deletion of a node in link list in the leaf nodes, such that it should maintain the link list even after deleting should sustain its link list.

Page No.:
Date: 22.6

Ex Inorder predecessor / successor.



#1 inorder -- predecessor
inorder ++ successor

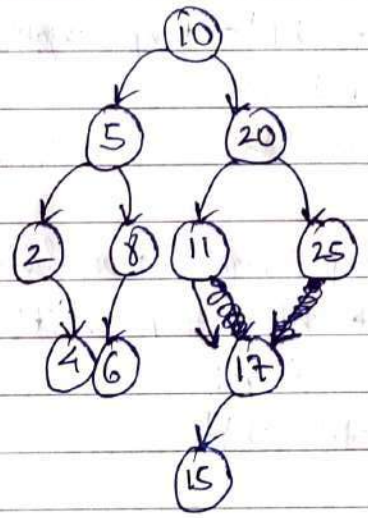
Inorder: 2 4 5 6 8 10 11 15 17 20 25

#2

left subtree ka max value.

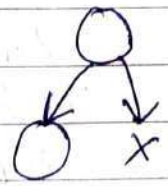
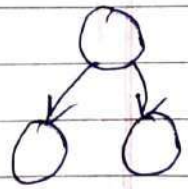
inorder predecessor of 20 : 17
inorder successor of 20 : 25.

Ex Deletion in BST



1. 25 tak pahucho
2. 25 ko delete karo.
3. Use 4 Cases discussed below.

target : 25

Case I	Case II	Case III	Case IV
			
return null	return root→right	return root→left	replace the node with Inorder predecessor or or Inorder Success or and delete that end node

Raw

faulty Code

```
Node * deleteNodeInBST(Node* root, int target){
    // base case
    if(root==NULL)
      return root;

    // Step 1
    Node * temp = findNodeinBST(root, target);
    // I want to delete temp
    if(temp->left==NULL && temp->right==NULL){
      //leafnode
        delete temp;
        return NULL;
    }
    else if(temp->left==NULL && temp->right !=NULL){
        Node* child = temp->right;
          delete temp;
          returnchild;
    }
    else if (temp->right==NULL && temp->left != NULL){
        Node* child = temp->left;
          delete temp;
          return child;
    }

    else {
          // dono child exist kartae hai.
          //inorder predecessor left subtree max
        int inorderPre = maxVal(temp->left);
          temp->data = inorderPre;
    temp->left= deleteNodeinBST(temp->left, inorderPre);
          return root;
    }
}
```

```
Node* deleteNodeInBST (Node* root, int target){
    // base case
    if(root==NULL){
        return NULL;
    }

    if(root->data==target){
        // isko delete karna hai  4 cases
        if(root->left==NULL && root->right==NULL)
            return NULL;

        else if(root->left==NULL && root->right != NULL){
            Node* child = root->right;
            return child;
        }

        else if(root->left!=NULL && root->right==NULL){
            Node* child = root->left;
            return child;
        }

        else { // both child  inorder predecessor
            int inorderPre = maxVal(root->left);
            root->data = inorderPre;
            root->left = deleteNodeInBST(root->left, inorderPre);
            return root;
        }
    }

    else if (target > root->data ){
        root->right = deleteNodeInBST (root->right, target);
    }

    else if (target < root->data){
        root->left = deleteNodeInBST(root->left, target);
    }
    return root;
}
```

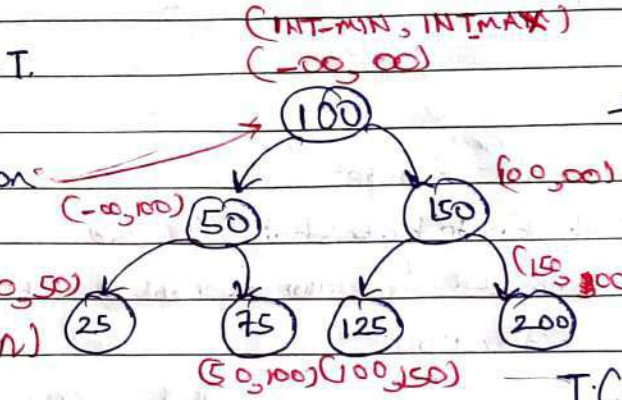
BST class 2.

height & diameter of BST (same as an ordinary tree)

Validate BST.

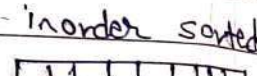
#2 Recursion:

ranges



TC = O(log n)

#1 inorder sorted



if sorted BST

if not sorted not a BST

T.C = O(n) linear

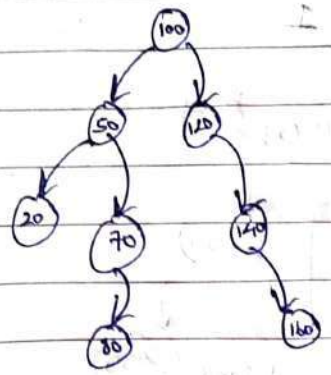
S.C = O(n).

```
bool isValidBST (TreeNode *root){
    int lowerbound = INT-MIN;
    int upperbound = INT-MAX;
    bool ans = solve (root, lowerBound, upperbound);
    return ans;
}
```

```
bool solve (TreeNode * root, int lb, int ub){
    //base case
    if (root == NULL)
        return true;

    if (root->val > lb && root->val < ub) {
        bool leftAns = solve (root->left, lb, root->val);
        bool rightAns = solve (root->right, root->val, ub);
        return leftAns && rightAns;
    }
    else {
        return false;
    }
}
```

BST Bramastra #1 inorder sorted
2 bst property

Lowest Common Ancestor (LCA) of BST.

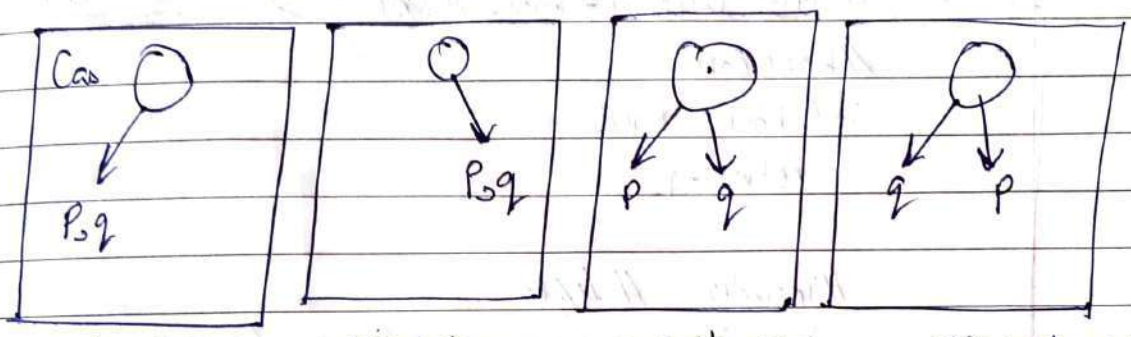


p = 70
q = 160

#1 same as normal tree (but not optimised)

#2.

ans = 100



p < root→data && q < root→data	p > root→data && q > root→data	p < root→data && q > root→data	p > root→data && q < root→data
Recursion	Recursion	ans = root	ans = root
[Left ki call maro]	[Right ki call maro]		

TC = O(log n)

```
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q){
    //base case
    if(root == NULL)
        return NULL;

    if(p->val < root->val && q->val < root->val){
        return lowestCommonAncestor(root->left, p, q);
    }
    if(p->val > root->val && q->val > root->val){
        return lowestCommonAncestor(root->right, p, q);
    }
    return root;
}
```

Case 1

Case 2

Case 3 and 4

imp ① Largest BST in a binary tree
② merge two BST

Ex k^{th} smallest element.

#1 inorder store

50 | 100 | 110 | 150 | 200 | 3

Ans

3rd smallest

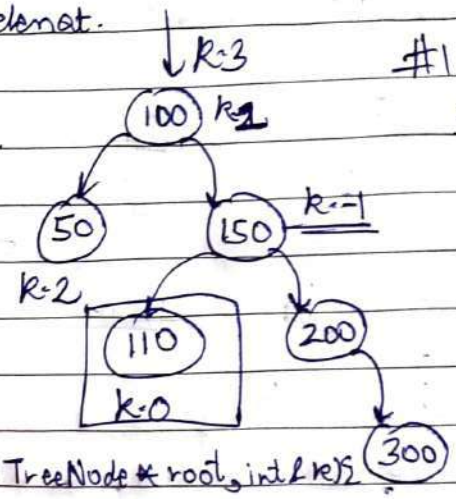
k=3

TC = O(n)

SC = O(n)

#2 Recursive

LNR




```
int kthSmallest(TreeNode* root, int& k){
    //base case
    if(root == NULL)
        return -1;

    //inorder  // left
    int leftAns = kthSmallest(root->left, k);
        if (leftAns != -1)
            return leftAns;

            N -> root
            k--;
            if(k==0){
            return root->val;
            }
}
```

Ex Create ~~mor~~ BST from inorder traversal

10	20	30	40	50	60	70	80
0	1	2	3	4	5	6	7

root

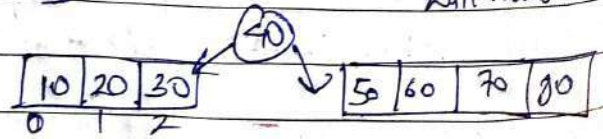
left | right

S = 0
e = 7

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

LNR

Balanced binary tree looks good so we will make it balanced



LMP

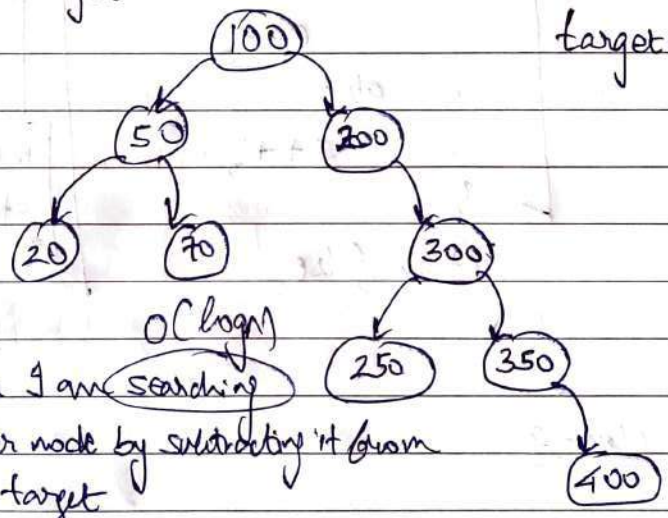
TC = O(n)

HW: Convert a BST into a balanced BST.


```
Node* bstUsing~~Recursion~~Inorder(int inorder[], int s, int e){
    //base case
    if (s > e)
        return NULL;

    int mid = (s+e)/2;
        int element = inorder[mid];
        Node* root = new Node(element);
        root->left = bstUsing~~Recursion~~Inorder(inorder, s, mid-1);
        root->right = bstUsing~~Recursion~~Inorder(inorder, mid+1, e);
    return root;
}
```

Ex 2 Sum target.

target = 320



#1
1st node traversal

O(logn)

for each option I am searching other node by subtracting it from target

TC = O(nlogn). worst case: n^2

#2 inorder sorted

Sorted array

20	50	70	100	200	250	300	350	400
0	1	2	3	4	5	6	7	8

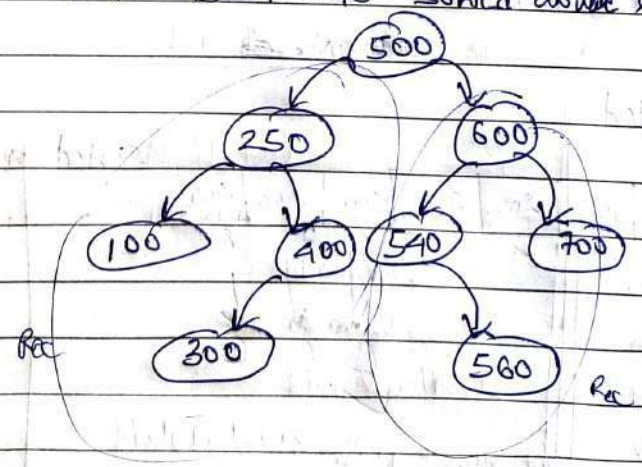
s → 0, e → 6, ~~d~~ 7, ~~e~~ 8

```
bool findTarget (TreeNode *root, int k) {
    vector <int> inorder;
    storeInorder (root, inorder);

    int s=0;
    int e= inorder.size()-1;

    while (s<e) {
        int sum = inorder[s] + inorder[e];

        if (sum==k)
            return true;

        if (sum> k)
            e--;
        else
            s++;
    }
    return false;
}
```

```
void storeInorder (TreeNode* root, vector <int> & inorder) {
    if (root==NULL)
        return;
    storeInorder(root->left, inorder);
    inorder.push_back(root->val);
    storeInorder (root->right, inorder);
}
```

↳ BST Class-3

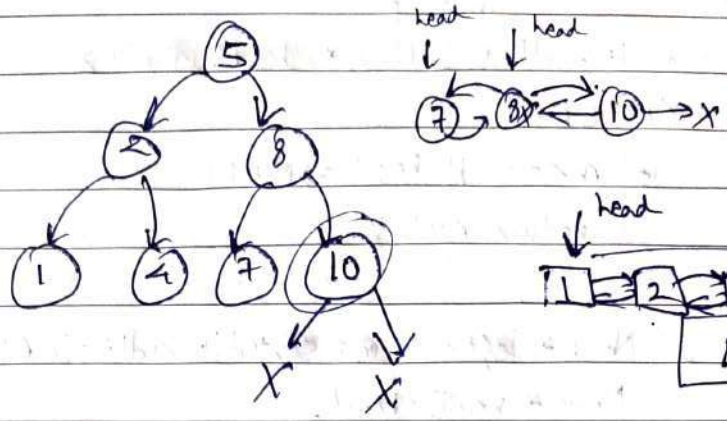
eg Convert BST to sorted double link list.



right → next
left → prev

DLL

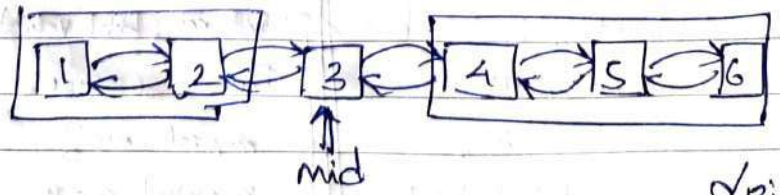
LNR inorder



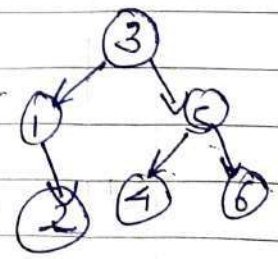
traverse O(n) so left is not recommended

```
void convertIntoSortedDLL (Node* root, Node*& head) {
    //base case
    if (root==NULL) {
        return;
    }

    //right subtree into LL
    convertIntoSortedDLL (root->right, head);

    //attach root node
    root->right=head;
    if (head != NULL)
        head->left =root;

    //update head
    head =root;
    //left subtree Link list.
    convertIntoSortedDLL (root->Left, head);
}
```

Eg Convert a sorted LL to BST



o/p:



SC: O(n) // inplace memory is changed

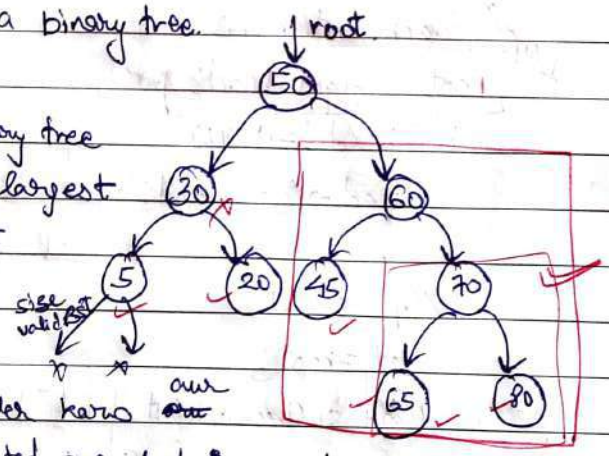
TC: O(n)

```
Node* sortedLinkedlistintoBST (Node *&head, int n) {
    //base case
    if(n<=0 || head==NULL)
        return null;

    Node* leftsubtree = sortedLinkedList in BST (head, n/2)
    Node* root = head;
    root->left = leftsubtree;
    head = head->right;

    //right part solve karna hai.
    root->right = sortedlinkedlistintoBST (head, n-n/2-1);
    return root;
}
```

imp

Largest BST in a binary tree.

This is a binary tree
locate largest
BST



Bottom up Approach

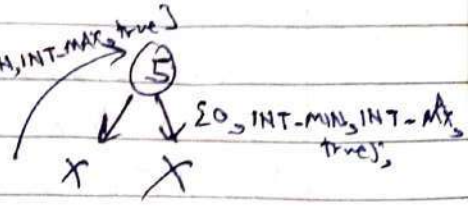
#1 Bt
ka inorder karo aur
wt sorted subset ho no other elements Largest BST ki size

LRN → recursive call

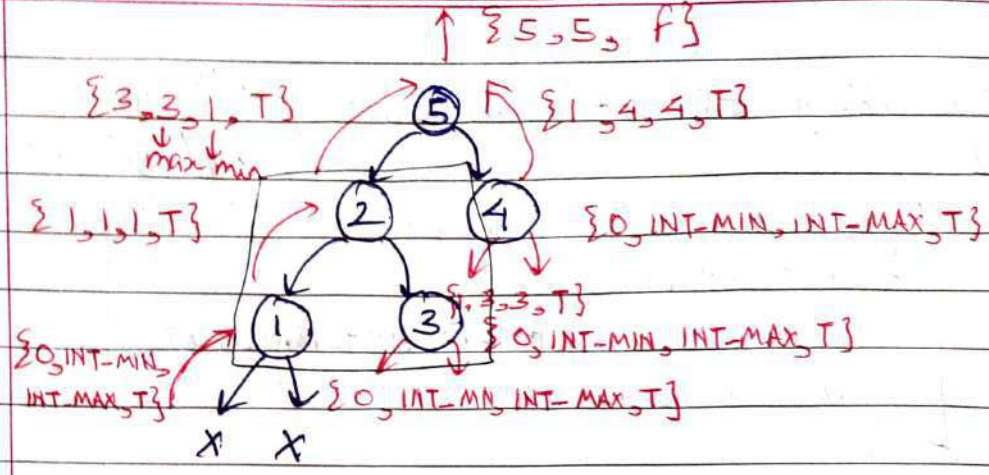
Node
↳ return
↳ valid BST
↳ max value from left compare
↳ min value from right
↳ size.

```
class NodeData {
    public: int size;
            int minVal;
            int maxVal;
            bool validBST;
    NodeData() {} //empty constructor
}
```

```
NodeData(int size, int max,
    int min, bool valid) {
    This->size = size;
    minval = min;
    maxval = max;
    validbst = valid;
}
```




```
NodeData findLargestBST( Node* root, int &ans){ 0, INT_MIN, INT_MAX, true }
{
    // base case
    if( root == NULL) {
        NodeData temp (0, INT_MIN, INT_MAX, true);
        return temp;
    }

    NodeData leftkaAns = findLargestBST (root->left, ans);
    NodeData RightkaAns = findLargest BST( root->right, ans);

    // checking starts here
    NodeData currentNodekaAns;

    currentNodekaAns. size = leftkaAns.size + rightkaAns.size +1;
    currentNodekaAns. maxVal = max (root->data, rightkaAns.maxVal);
    currentNodekaAns. minval = min( root->data , leftkaAns . minVal);

    if(leftkaAns.validBST && rightkaAns. validBST && (root->
    data > leftkaAns. maxVal && root->data < rightkaAns.min
    value )) {
        currentNode.validBST = true;
    }

    else {
        currentNode.validBST = false;
    }

    if (currentNodekaAns. valid BST) {
        ans = max (ans, currentNodekaAns.size);
    }

    return currentNodekaAns;
}
```



50 and 3

↳ Heap (data structure).

use stl

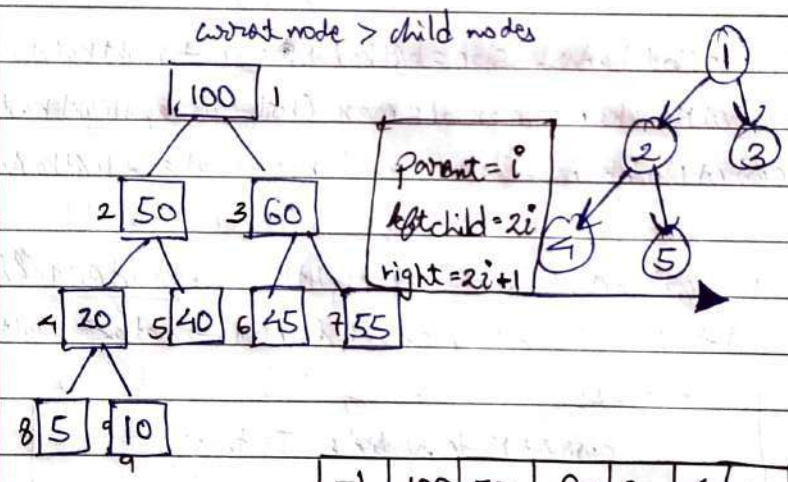
↳ CBT (Complete binary tree) + That holds a heap property

min max

CBT

↳ all levels are completely filled (except last one)

↳ node creation / insertion is always L to R



insertion is done through tree but creation is done through ARRAYS

-1	100	50	60	20	40	45	55	5	10
0	1	2	3	4	5	6	7	8	9

If indexing is done from index 0 then

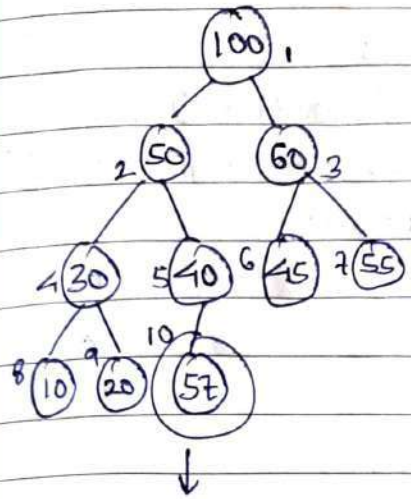
parent = i

leftchild = 2i+1

rightchild = 2i+2

index i
↓
parent i/2

new node: 57.

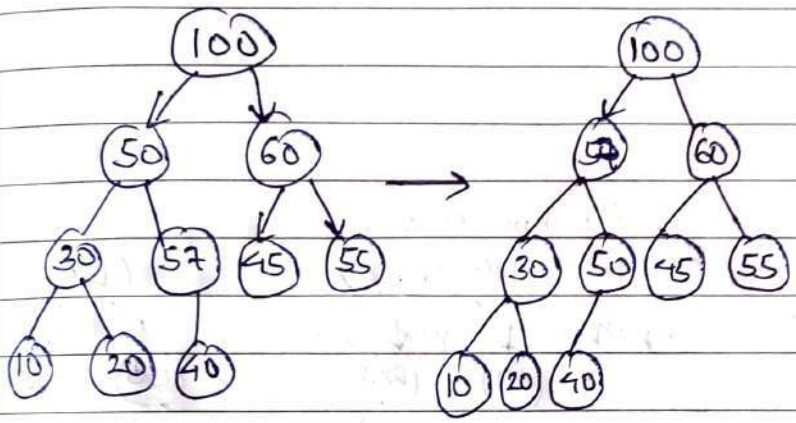


insertion

O(1) A) add node at the end of array

O(logN) B) sahi jagaha par lakar aao.

TC = O(logN)




```
class Heap{
    public:
        int arr[101];
        int size;      can also be
                       implemented dynamically

        Heap() {
            size = 0;
        }

        void insert(int value){
            size = size + 1;
            int index = size;
            arr[index] = value;
            // iss value ko place at right {sahi position}.
```

correct

Why heap:
Max value of arr O(1) top root element.
rather than O(n) or O(log n)


```
while (index > 1){
    int parentIndex = index/2;
    if (arr[index] > arr[parentIndex]){
        swap (arr[index], arr[parentIndex]);
        index = parentIndex;
    }
    else {
        break;
    }
}
}
```

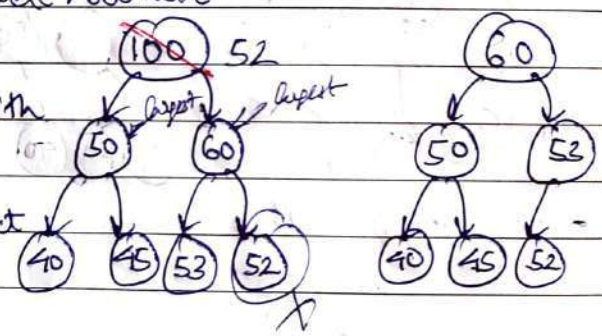
Deletion

we can only delete root node

O(1) a) Replace last value with root node

O(log n) b) Root node → correct part

$\boxed{TC = O(\log n)}$




```
int delete(){ int ans = arr[1];
    // delete root node value with last node data
    arr[1] = arr[size];
    size--;

    // place root node ka ~~answer~~ data on its correct position
    int index = 1;
    while (index < size){
        int left = 2 * index;
        int right = 2 * index + 1;
        int largest = index;
```

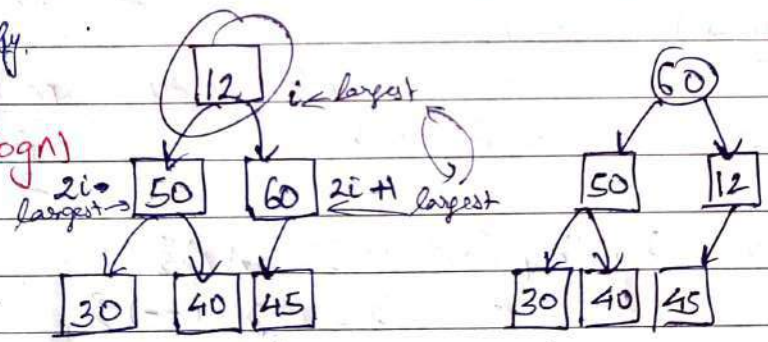
imp heapify: kisi bhi node ko uski correct position par laekar aana

```
if(left ≤ size && arr[largest] < arr[left]){
    largest = left;
}

if( right ≤ size && arr[largest < arr[right]){
    largest = right;
}
    if (largest == index){
        // value is at correct position
        ~~return ans;~~ break;
    }
    else {
        swap (arr[index], arr[largest]);
        index = largest;
    }
} return ans;
}
```

Desk work imp ex

heapify.

TC: O(logn)




```
void heapify( int arr[], int n, int i) {
    int index = i;
    int leftindex = 2*i;
    int rightindex = 2*i+1; int largest = index;
    if (leftindex < n && arr[largest] < arr[leftindex]) {
        largest = leftindex;
    }
```

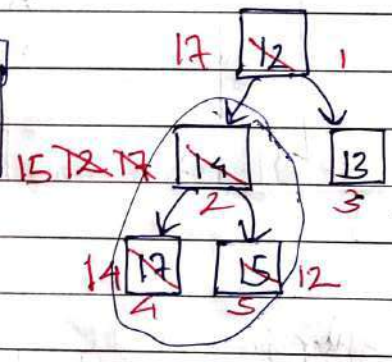
insertion logn
deletion logn
heapify logn
[Build heap O(n)]
heapsort O(logn)


```
if (rightindex < n && arr[largest] < arr[rightindex]){
    largest = rightindex;
}

if (index != largest) {
    //left ya right child me sae koi > hogya current
    swap (arr[index], arr[largest]);
    index = largest;
    heapify (arr, n, index);
}
}
```

Eg Building heap from array

-1	12	14	13	17	15
0	1	2	3	4	5

heap



leaf node heapify
no need

$\left(\frac{n}{2}+1\right) \longrightarrow n$

nodes are leaf nodes (no heapify)

$3 \longrightarrow 5$ leaf node

$\frac{n}{2} \longrightarrow > 0$
Yes heapify

```
void buildheap (int arr[], int n){
    for (int i = n/2; i>0; i--) {
        heapify (arr, n, i);
    }
}
```

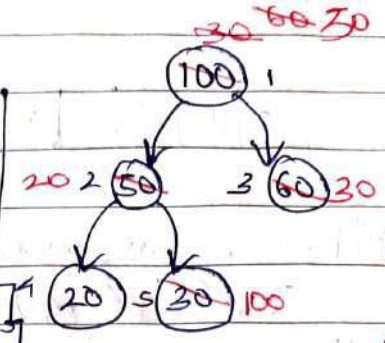
TC - $\frac{n}{2}$

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

wrong: $n \log n$ ✗

Hint: Correct TC - O(n)

TC:
Heapsort $(n \log n)$



1) first element to last element so swap.

$n = 5$

index

100 50 60 20 30

index

60 30 30 20 | 100
heap sorted

50 20 30 | 60 100

30 20 | 50 60 100

20 | 30 50 60 100

20 30 30 60 100
sorted

```
void heapsort (int arr[],
    int n){
  while (n != 1){
    swap (arr[1], arr[n--]);

    //heapify
    heapify (arr, n, 1);
  }
}
```

b) Heap class 2 cppreference.com for more reference
(stl)

Priority_queue (heap)

```
#include <queue>
Priority_queue <int> pq; //max heap
  pq. push (3);
  pq. push (6);
  pq. push (9);
  pq. push (4);
  pq. push (8);
  cout << "top element" << pq.top() << endl; // 9.   3
  pq.top();
  cout << "top element" << pq.top() << endl; //8   4
```

priority_queue<int, vector<int>, greater<int>> pq;
// min heap

k^{th} smallest
k^{th} greatest
merge sorted arrays
merge sorted linklist } heap Think about HEAP first.

Ex k^{th} smallest no.

arr
i/p

3	11	6	9	4	12	2	8

k

#1 → sort (arr, arr+n) → k^{th} smallest arr[k-1]
TC: O(n log n)

#2. min-heap

n size heap → pop (n-k) times
TC: O(n) TC = O(k)

#3 Max-heap.

insert → new element < heap.top()
heap.pop();
heap.push(new element);

O(k)

k small element

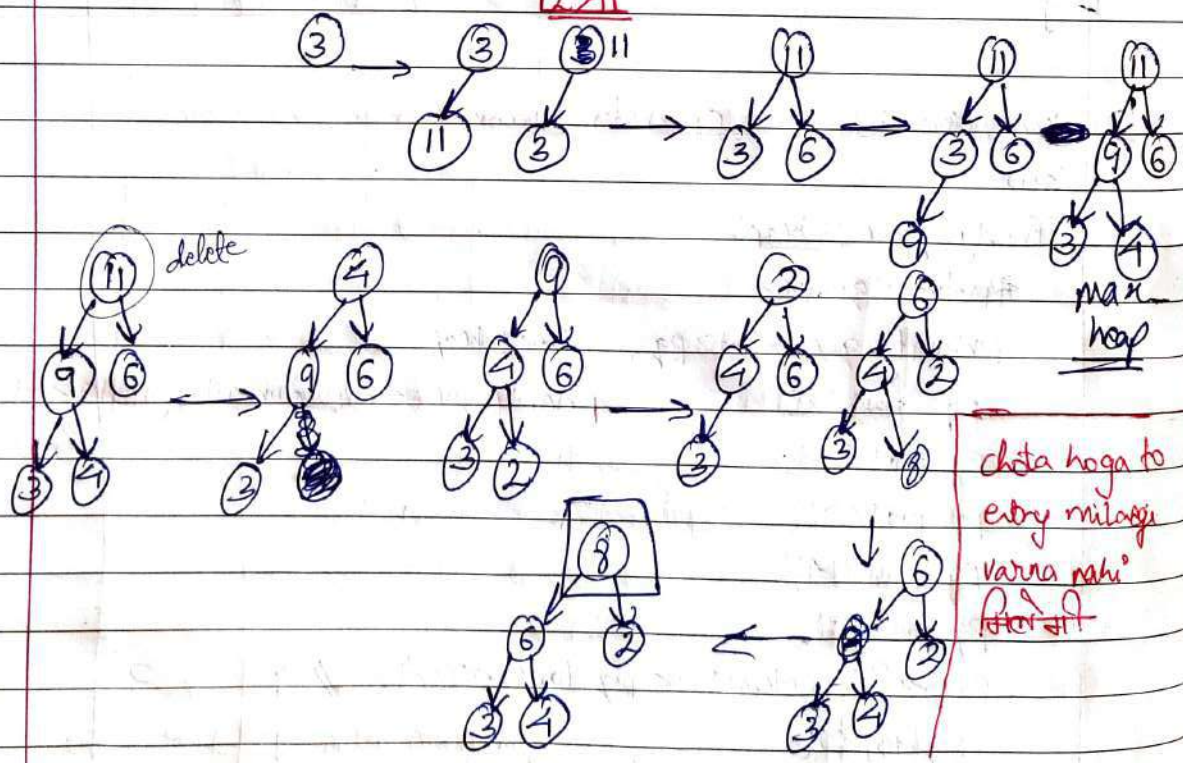
top → k^{th} smallest element.

ignore

3	11	6	9	4	12	2	8	
0	1	2	3	4	5	6	7	

k = 5

12 > 11



chota hoga to entry milegi varna nahi मिलेगी

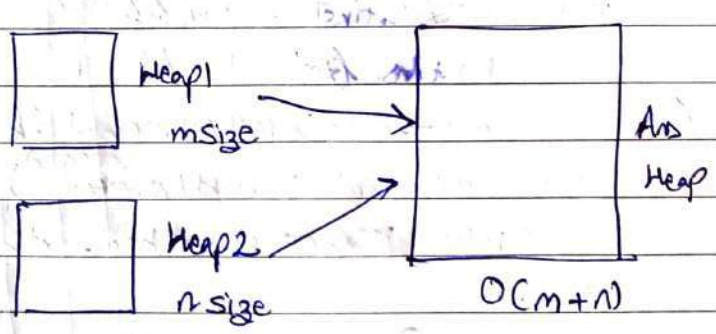
```
int getKthsmallestelemet (int arr[], int n, int k) {
    // create max heap
      priority-queue<int> pq;
    //insert initial k elemets of array
    for (int i=0; i<k; i++) {
        q.push(arr[i]);
    }
```

SC = O(k)
TC = O(n log k)

```
    // for remaining elemets, push only if they are less than top
    for (int i=k; i<n; i++) {
        int elemet = arr[i];
        if (elemet < pq.top()) { for kth greatest elemet
            pq.top();
            pq.push(elemet);
        }
    }
    int ans = pq.top();
    return ans;
}
```

kth elemet ko alag treatment dena
↳ de-queue
↳ sliding window

H.W
Q. Merge two max heap



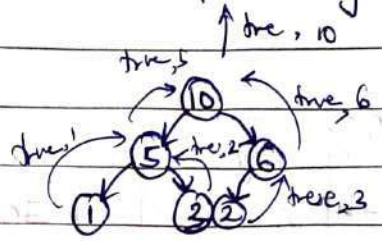
#1 merge two array
O(m+n)
then build heap

#2 task1
heap create

task2
traverse O(n)
then push
n log n

Eg Given a CBT (complete binary tree) tell whether it is a heap or not.

CBT — heap property check

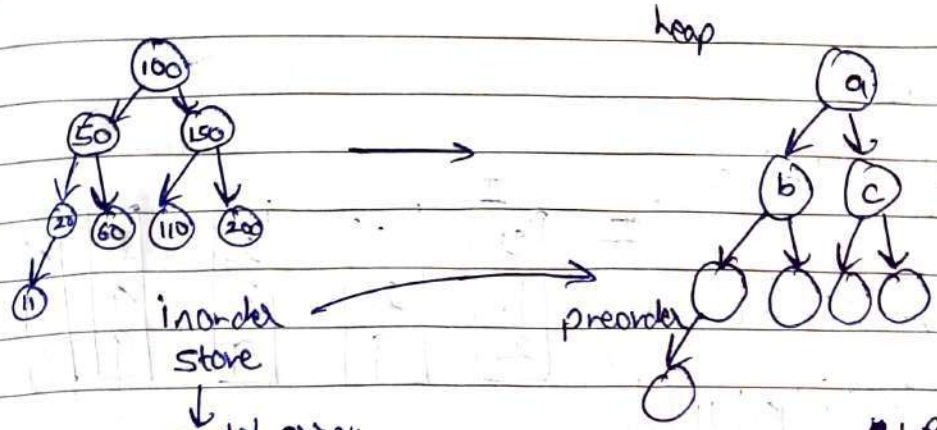



```
pair <bool, int> solve (Node* root){
    // basecase
    if (root==NULL){
        pair<bool, int> p = make_pair(true, INT_MIN);
        return p;
    }

    if(root->left ==NULL && root->right==NULL)
    {
        // left node
        pair <bool, int> p = make_pair(true, root->data);
        return p;
    }

    pair <bool, int> leftAns = solve(root->left);
    pair <bool, int> rightAns = solve(root->right);

    if( leftAns.first ==true &&
        rightAns.first ==true &&
        root->data > leftAns.second &&
        root->data > rightAns.second)
        return pair<bool, int> make_pair(true, root->data);
    else
        return pair<bool, int> make_pair(false, root->data);
}
```

Eg Convert BST into maxheap. #1 heapify.

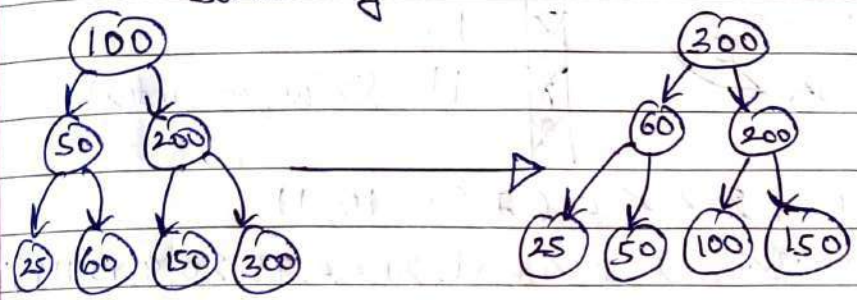
BST
max heap property

converting BST to max heap structure of the tree remains same ∧ only value we have to change.



LRN

postorder store
insert.



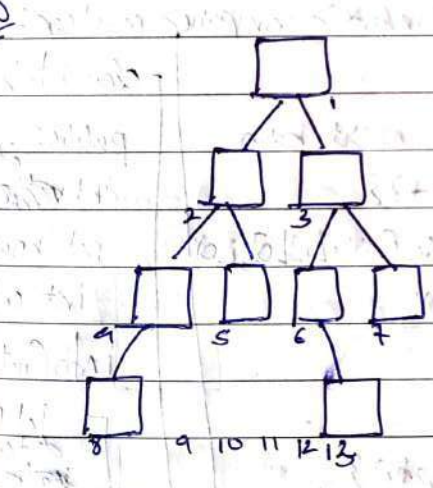
array ko store kia bina bhi kar sakte hai. 1:16:49.

Ex Check whether tree is a CBT

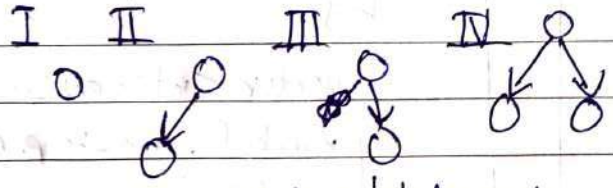
H.W

#1 - level order.

#2 Recursively.



13>9 so it is not a CBT



cont > total-cont.

Ex Merge k sorted arrays

1st array	2	4	6	8	10	12
2nd array	3	9	15	18	12	24
k	5	20	25	30	35	40

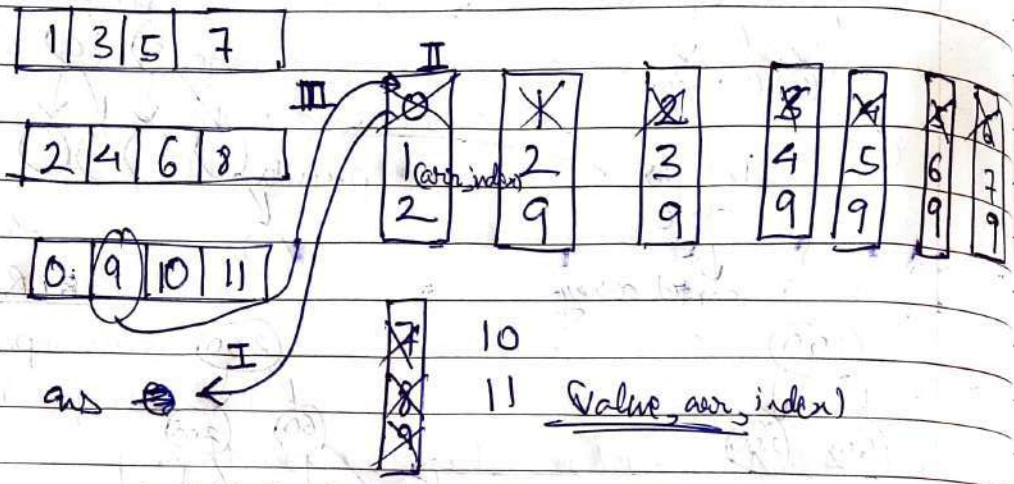
#1 move Ans as to 1 array
sort
$O(nk \log(nk))$

#2 min heap from 1st elements

2	3	5	6	9	10	12	15	18	20	21
24	25	30	35	40						

CPP reference
Custom comparator comparator

↳ Heap Lect 3



0 1 2 3 4 5 6 7 8 9 10 11

```
int arr[][4] = {{2,4,6,8},{1,3,5,7},{0,9,10,11}};
int k=3;
int n=4;
```

```
vector<int> mergekSortedArrays(int arr[][4], int k, int n){
priority_queue<info*, vector<info*>, compare> minheap;

// hr array ka first element insert karo
for(int i=0; i<k; i++){
    info* temp = new info(arr[i][0], i, 0);
    minHeap.push(temp);
}

vector<int> ans;
while(!min_heap.empty){
    info* temp = min_heap.top();
    int top_element = temp->data;
    int top_row = temp->row;
    int topcol = temp->col;
    minHeap.pop(); O(1)
    ans.push_back(top_element);
```

Klog(k)

(n×k − k) times

```
class info{
public:
    int data;
    int row;
    int col;
    info(int val, int r, int c){
        data = val;
        row = r;
        col = c;
    }
};

class compare{
public:
    bool operator()(info* a, info* b){
        return a->data > b->data;
    }
```

```
        if (topCol+1<n){
log(k)      info* newInfo = new info(arr[topRow][topCol+1], topRow
                                , topCol+1);
            minHeap.push(newInfo);
        }
    }
    return ans;
}
```

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

~~(1, 0, 0)~~	~~(2, 1, 0)~~		
(2, 1, 0)	(3, 2, 0)		
(3, 2, 0)	(5, 0, 1)		
(13, 3, 0)	(13, 3, 0)		

k = 4, n = 4

(val, row, col)

ans =

eg Merge k sorted linklist.

2 → 4 → 6 → 8 → X

1 → 3 → 5 → 7 → X

0 → 9 → 10 → 11 → X

head = NULL
tail = NULL

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

ans: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

7	8	9	
8	9	10	11
9			

```
class compare {
    public:
    bool operator() (ListNode * a, ListNode * b){
        return a->val > b->val;
    }
};

class Solution {
    public:
        ListNode * mergeLists (vector < ListNode * > & lists){
            priority_queue < ListNode *, vector Node*>, compare
            minheap;

            int k = lists.size();
            if(k==0)
                return NULL;

            // first element of every linked list ko insert
               karo heap mae.
            for (int i=0, i < k; i++){
                if ( lists[i] != NULL){
                    minheap.push (lists[i]);
                }
            }
            ListNode * head = NULL;
            ListNode * tail = NULL;
            while (! minheap.empty) {
                ListNode * temp = minheap.top();
                minheap.pop();
                // temp may or may not be the first element
                   of answer linked list
```

```
if ( head = = NULL ) {
    //temp → first element of ans LL
    head = temp;
    tail = temp;
    if ( tail → next ! = NULL ) {
        minHeap.push ( tail → next );
    }
}
else {
    //temp is not the first element of LL
    tail → next = temp;
    tail = temp;
    if ( tail → next ! = NULL )
        minHeap.push ( tail → next );
    }
}
return head;
}
};
```

Amazon 3rd round

Ex Smallest range in k list.

(Smallest Range covering elements from k lists

Smallest हो और हर list से एक element तो आ ही जाए

[4, 10, 15, 24, 26] (24)
[0, 9, 12, 20] (20)
[5, 18, 22, 30] (22)

Output: [20, 24]

| 1 | 2 | 3 |

0 min = 1 max = 10

1 min = 1 max = 2 min → max

| 1 | 2 | 3 |

1 min 1 max = 2

मतलब हर 5'वा element आ रहा होगा

0 min 2 max = 2

| 1 | 2 | 3 |

1 min 2 max = 3 min - max

1 min 2 max = 3 min max

0 min 3 max 3 min - max

min - max

ans = 0

smallest difference

maxi - mini + 1

smallest - range

min - max -

→ ←

min - heap

min → increase ✓

max → decrease ✗

| 4 | 10 | 15 | 24 | 26 |

| 0 | 9 | 12 | 20 |

| 5 | 18 | 22 | 30 |

5 0 - 5 5 5 → 10 8 10 - 18 5 15 - 20

5 4 - 9 9 9 → 18 6 12 - 18 6 18 - 24

4 20 - 24

```
class node {
    public:
     int data;
     int row;
     int col;
        node(int d, int r, int c){
            data = d;
            row = r;
            col = c;
        }
};
```

```
class compare {
    public:
    bool operator()(
        node* a, node* b){
        return a->data >
            b->data;
        }
};
```

```
class Solution {
  public:
    vector<int> smallestRange(vector<vector<int>>&nums){
      int mini = INT_MAX;
      int maxi = INT_MIN;
      priority_queue<node*, vector<node*>, compare> minHeap;
      int k = nums.size();

      for (int i=0; i<k; i++){
          int element = nums[i][0];
          maxi = max (maxi, element);
          mini = min (mini, element);
          minHeap.push (new node (element, i, 0));
      }

      Int ansStart = mini;
      int ansEnd = maxi;

      while (! minHeap.empty()){
          node* top = minHeap.top();
          int topelement = top->data;
          int topRow = top->row;
          int topcol = top->col;
          minHeap.pop();

          // mini updated
          mini = topelement;

          // check for answer
          int currRange = maxi - mini
          int ansRange = ansEnd - ansStart;
          if( currRange < ansRange){
              ansStart = mini;
              ansEnd = maxi;
          }
```

```
// Check for new element in the same list.
if (topCol + 1 < nums[topRow].size()){
    maxi = max(maxi, nums[topRow][topCol+1]);
    node* newNode = new node(nums[topRow][topCol+1],
                    topRow, topCol+1);
    minHeap.push(newNode);
}
else {
    // There is no element in the same array or list
    break;
}
}

vector<int> ans;
ans.push_back(ans.start());
ans.push_back(ans.end());
return ans;
}
```

Level-1
online test
level question

Heap class 4. [Greedy]

Medium

Leetcode 1962 Remove Stone to minimise the total

Pile

5	8	11	12
0th	1st	2nd	3rd

k operation
total - floor(total/2)

MaxHeap → k operations

min stones after k operation

5 8 11 ~~12~~
~~6~~ 3

k=3

~~12~~	6	~~11~~	6	~~8~~	4
11		8		6	
8		6		6	
5		5		5	

total = 21

```
int minStoneSum(vector<int>& piles, int k){
    priority_queue<int> maxHeap;

    for (int i = 0; i < piles.size(); i++){
        maxHeap.push(piles[i]);
    }

    while (k--){
        int maxElement = maxHeap.top();
        maxHeap.pop();
        maxElement = maxElement - floor(maxElement/2);
        maxHeap.push(maxElement);
    }
    int sum = 0;
    while (!maxHeap.empty()) {
        int temp = maxHeap.top();
        maxHeap.pop();
        sum += temp;
    }
    return sum;
}
```

TC = n + k log n

int ① heapify function() ② heapify cpp reference] HW

lamda expressions

Similar to
HW [minimum-cost to cut the ropes]
↳ heaps

ex Reorganise strings Leetcode 767. Medium
I/p: aab if not possible then return empty string
o/p: aba

map

aab a→2 max occurance letter should
b→1 be used more

(a,2)
(b,1)

" " + a + b + a

aaa bccdd cc a→3
b→1
c→2
d→2
e→2

a→3 → diff	a=2 →
c→2 → diff	d=2 →
d→2	e=2
e→2	c=1
b→1	b=1

ans = " "

" " + a + c + a + d + e + a + e + d + c + b

e=2 →	c=1 →	e=1 →	e=0
a=1 →	b=1 →	d=1 →	d=0
b=1	e=1	a=0	a=0
d=1	d=1	b=0	b=0
c=1	a=0	e=0	c=0

```
class Node {
    public:
        char data;
        int cont;

        node(char d, int c) {
            data=d;
            cont=c;
        }
};
```

```
class compare {
    public:
        bool operator()(Node* a, Node* b) {
            return a.cont<b.cont
        }
};
```

```
class Solution {
public:
    string reorganizestring (string s) {

        // Create mapping
        int freq[26] = {0};

        for (int i=0; i<s.length(); i++){
            char ch = s[i];
            freq[ch-'a']++;
        }

        priority_queue <node, vector <node>, compare> maxHeap;
        for(int i=0; i<26; i++){
            if( freq[i] !=0) {
                node temp(i+'a', freq[i]);
                maxHeap.push(temp);
            }
        }

        string ans="";
        while (maxHeap.size()>1) {
            node first = maxHeap.top();
            maxHeap.pop();
            node second = maxHeap.top();
            maxHeap.pop();

            ans += first.data;
            ans += second.data;

            first.count--;
            second.count--;
            if (first.count !=0)
                maxHeap.push (first);
            if(second.count !=0)
                maxHeap.push(second);
        }
        if( maxheap.size()==1){
            node temp = maxheap.top();
            maxheap().top();
            maxheap.pop();
            if (temp.count==1)
                ans += temp.data;
            else
                return "";
        }
        return ans;
    }
};
```

signum function ?

do it yourself

DIY Ex

Greedy approach

1405 Longest happy string

return longest possible happy string

a, b, c

aaa, bbb, ccc

a atmost a times

b atmost b times

c atmost c times

i/p a = 1 b = 1 c = 7

o/p c a c c b c c

i/p: a = 7 b = 1 c = 0

o/p: a a b a a 3 longest possible happy string

second count > first count

imp Ex

Median in a stream

2, 3, 5, 1, 4, 9

median median median median median

#1 ~~med~~ vector → push → sort → middle element → median

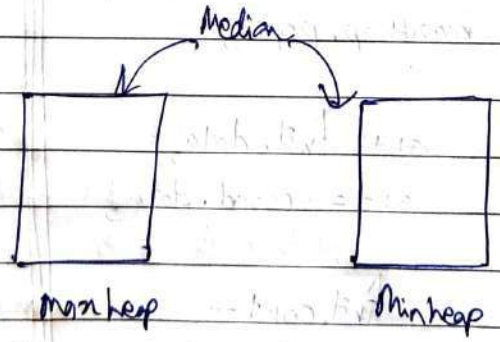
nlogn

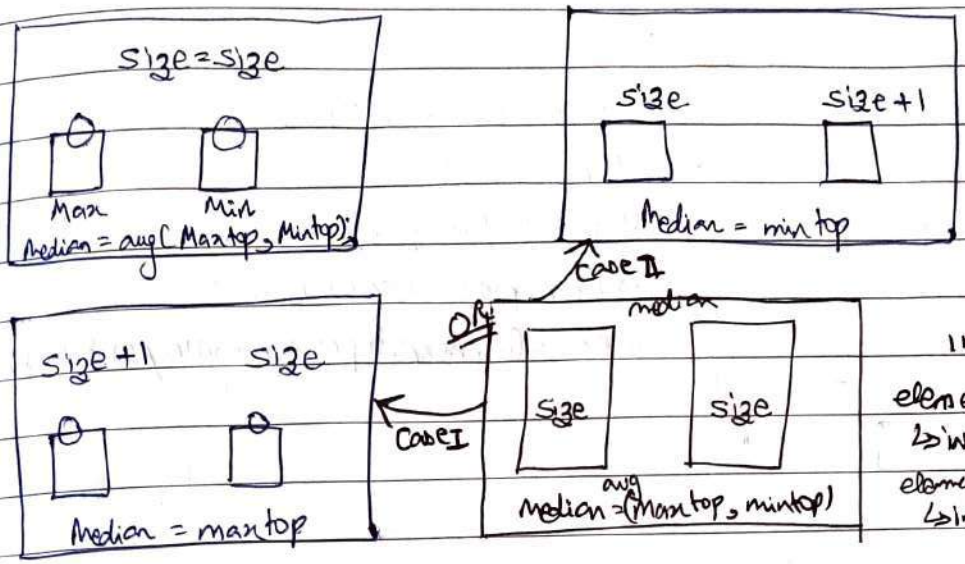
#2 BST no → BST create → inorder sorted

log n insert

n

#3: minheap + maxheap combinations





1 new element
element > median
↳ insert it in minheap
element < median
↳ insert it in max heap

1 new element
element > median
↳ insert min heap

element < median
↳ insert max heap

```
void callMedian (int& median, priority_queue <int> &maxHeap, priority_queue
<int, vector <int>, greater <int>> & minHeap, int element){

    if( minHeap.size() == maxHeap.size()){

        if(element > median ){
            minHeap.push (element);
            median = ~~mid~~ minHeap.top();
        }

        else {
            maxHeap.push(element);
            median = maxHeap.top();
        }
    }

    else if( maxHeap.size() > minHeap.size()){
        if ( element > median){
            minHeap.push(element);
            median=(minHeap.top() + maxHeap.top())/2;
        }
```

median ?

2 3 5 6 11 12 13

↳ median

2 3 [4 5] 6 7

↳ $\frac{4+5}{2} = 4.5$

```
    else {
        int maxTop = maxHeap.top();
        maxHeap.pop();
        minHeap.push(maxTop);
        maxHeap.push(element);
        median = (minHeap.top() + maxHeap().top())/2;
    }
}

else {
    // maxHeap.size() < minHeap.size()
    if (element > median) {
        int minTop = minHeap.top();
        minHeap.pop();
        maxHeap.push(minTop);
        minHeap.push(element);
        median = (minHeap.top() + maxHeap.top())/2;
    }

    else {
        maxHeap.push(element);
        median = (minHeap.top() + maxHeap.top())/2;
    }
}
}

int main(){
    int arr[6] = {5, 7, 2, 9, 3, 8}
    //          5 6 5 6 5 6
    int median = 0;
    priority-queue <int> maxHeap;
    priority-queue <int, vector<int>, greater <int>> minHeap;
```

```
    for (int i=0; i<n; i++){
        int element = arr[i];
        Callmedian (median, maxHeap, minHeap, element);
        cout << "->" << median << endl;
    }  cout << endl;
    return 0;
}
```

Longest happy string

```
class node {
    public:
    char data;
    int count;

    node (char d, int c) {
        data = d;
        count = c;
    }
};
```

```
class compare {
    public:
    bool operator()(node a, node
        b) {
        return a.count < b.count;
    }
};
```

```
class Solution {
    public:
    string longestDiverseString (int a, int b, int c) {
        priority_queue<node, vector<node>, compare> maxHeap;

        if (a>0) {
            node temp ('a', a);
            maxHeap.push (temp);
        }

        if (b>0) {
            node temp ('b', b);
            maxHeap.push (temp);
        }

        if (c>0) {
            node temp ('c', c);
            maxHeap.push (temp);
        }
```

```
String ans="";
while(maxHeap.size()>1){
    node first=maxHeap.top();
    maxHeap.pop();
    node second=maxHeap.top();
    maxHeap.pop();

    if(first.count>2){
        ans+=first.data;
        ans+=first.data;
        first.count-=2;
    }
    else {
        ans+=first.data;
        first.count--;
    }

    if( second.count>2 && second.count> first.count){
        ans+=second.data;
        ans+=second.data;
        second.count-=2;
    }
    else {
        ans+=second.data
        second.count--;
    }
    if( first.count >0) {
        maxheap.push(first);
    }
    if( second.count > 0) {
        maxheap.push(second);
    }
}
```

```
if (maxHeap.size() == 1) {
    node temp = maxHeap.top();
    maxHeap.pop();

    if (temp.count ≥ 2) {
        ans += temp.data;
        ans += temp.data;
        temp.count -= 2;
    }

    else {
        ans += temp.data;
        temp.count--;
    }
}
return ans;
}
};
```

Lect-1

↳ Hashmap. Datastructure [very beneficial in online tests]
That stores data in key value pair

(string) key	value (int)
Scorpio	9
balino	9
fortuner	10

<key, value>

HashMap
↙ ↘
Ordered — $O(\log n)$
mordered — $O(1)$

imp
HW
Ex

Create a data structure that insertion, deletion, searching, getrandom $O(1)$ time complexity

```
#include <unordered_map>

//creation
unordered_map<string,int> m;

//insertion
pair <string, int> p = make_pair("Scorpio", 9);
m.insert(p);

pair <string, int> p2("alto", 2);
m.insert(p2);

m["fortuner"] = 10;

//access
cout << m.at("alto") << endl // 2
cout << m["fortuner"] << endl // 10

//Search
cout << m.count("Scorpio") << endl; //1 yes it is present
```

```
if( m.find("Fortuner") != m.end()){
    cout<< "fortuner found" <<endl;
}

else {
    cout<< "fortuner not found" <<endl;
}

cout<< m.size() <<endl;   //3

cout<< m.size()<<endl;   //3
cout<< m["hummer"] <<endl;
cout<<  m.size()<<endl;
                //4
```

Accessing this way may lead to creation of an entry in the hashtable of hummer and corresponding 0 to it.

```
// for printing the values
for(auto i:m){
    cout<< i.first <<"->"<< i.second <<endl;
}
```

Ordered map O(logn) based on BST
unordered map O(1)

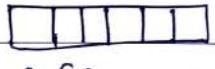
implementation

↳ linklist

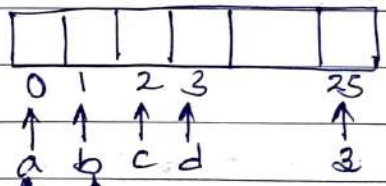
TC= O(n)

↳ BST O(logn) Balanced BST

O(logn)

↳ Arrays  bucketarray or hashtable

O(1)



arr[ch-'a']++

but what if dhairya pandya

Mapping "dhairy" with index 0 requires the ~~requires~~ requirement of hash function

"love" → [HF] → 0th index

Hash function
↳ hash code
↳ + Compression function

hash code

key [HC] → numeric value

i.e. 56

This numeric value is bought to range by compression function

$56 \% 26 = \underline{\underline{4}}$

0 – 25 range

babbar → [HC: b+a+b+b+a+r, 2+1+2+2+1+18] → $\underline{\underline{26}}$ → [CF: 26%26] → 0th index

baarbb → ?

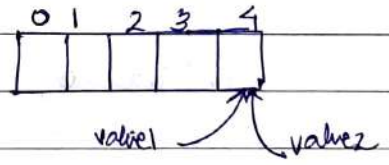
Collision?

Hash function

babbar → [ ] → 26
baarbb → [ ] → 26

Collision handling

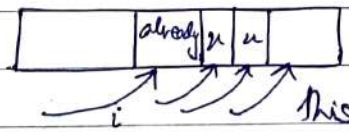
#1 ↳ open hashing

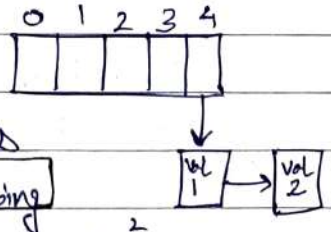


#2 closed addressing

$hf(i) + \text{free space}(i)$

we will store both the values through Linkelist.





This is known as

linear probbing

Quadratic Probbing 1 step ahead then $(2)^2$ step ahead 3^2 step ahead.

a	b		c			d

no. of elemets - n
no. of free box = b

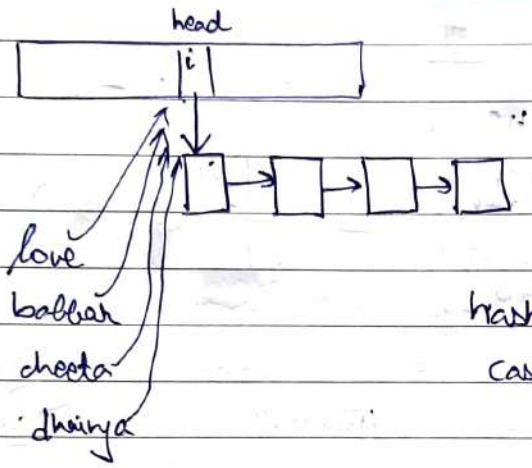
$\frac{n}{b} < 0.7$ for good hash function.

load factor

↓↓↓↓↓↓
babbar
↓
HF

TC = O(K)

But here we are considering
$n >>>>> k$



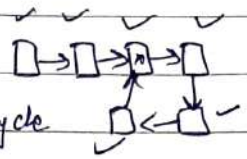
O(n)
worst case
hash function is so strong that this case doesn't exist.

ex

```
string str="thiruvananthapuram";
unordered-map<char, int> freq;

for( int i=0; i<str.length(); i++){
        char ch = str[i];
        freq[ch]++;
}

for( auto i: freq){
    cout << i.first << " " << i.second << endl;
}
```

unorderedmap		ordered map	
m	1	a	4
p	1	h	2
n	2	i	2
a	4	m	1
r	2	n	2
v	1	p	1
i	1	r	2
u	2	t	2
h	2	u	2
t	2	v	1

Check for circular link list cycle



```
bool checkCircular(Node *head){
    unordered_map<node*, bool> vis;
    Node* temp = head;

    while (temp != null) {
        if( vis.find(temp) != vis.end()) {
            visited[temp] = true;
        }
        else {
            return true;
        }
        temp = temp->next;
    }
    return false;
}
```

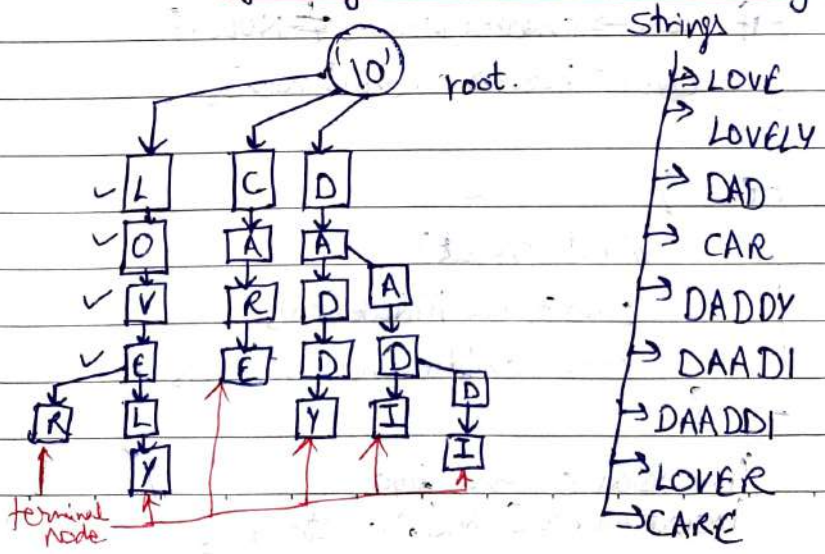
Tries 2-2.

node present hai तो चले जाओ और agar node नहीं है तो node create करो और चले जाओ।

Data Structure

Multi way tree structure

generally used for pattern searching



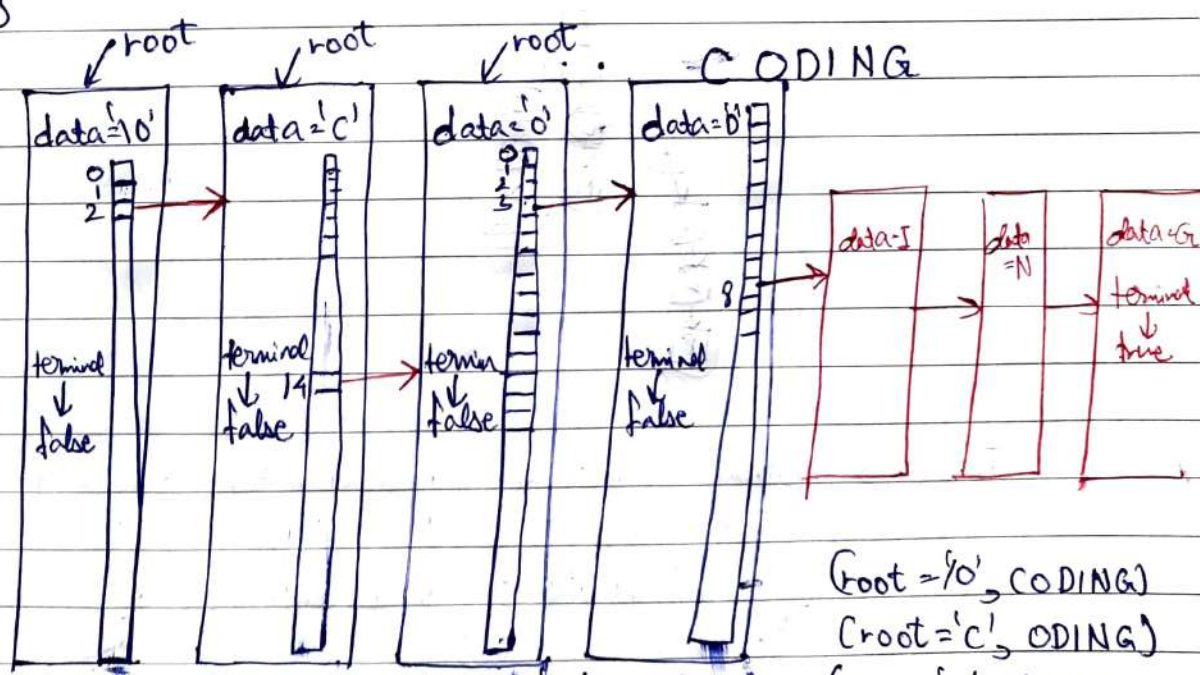
TC ∝ pattern ki length

Project: use tries to make a auto suggestion

26 entries

```
class TriNode {
    public :
     char data;
     Trinode * children[26];
     bool isTerminal;

     TriNode (char d) {
        this -> data = d;
        for (int i = 0; i<26; i++) {
            children[i] = NULL;
        }

        this -> isTerminal = false;
     }
};

void insertWord (TriNode* root, string word) {
    if (word.length() == 0) {
        root -> isTerminal = true;
        return;
    }

    char ch = word[0];
    int index = ch - 'a';
    TriNode * child;
    // present
    if (root -> children [index] != NULL) {
        child = root -> children[index];
    }
    else {
        // not present)
        child = new TriNode (ch);
        root -> children[index] = child;
    }

    // recursion sambhal lega
    insertWord (child, substr(1));
}
```

```
bool searchWord (TriNode* root, String word){
    // base case
    if(word.length()==0){
        return root->isTerminal;
    }

    char ch = word[0];
    int index = ch-'a';
    TriNode* child;

    // preset
    if(root->children[index]≠NULL){
        child = root->children[index];
    }
    else {
        return false;
    }

    //recall
    return searchWord(child, word.substr(1));
}
```



(root='N', G);
(root='G', '');

(root='O', CODING)
(root='C', ODING)
(root='O', DING)
(root='D', ING);
(root='I', NG);

L-3

Trie O(n) Pattern matching
↳ n is the length of string

ex Longest Common Prefix easy 14 Leetcode Bruteforce Best but can also be solved through tries.
By the way this sum is already done previously.

#1 [C][O][d]e

Bruteforce

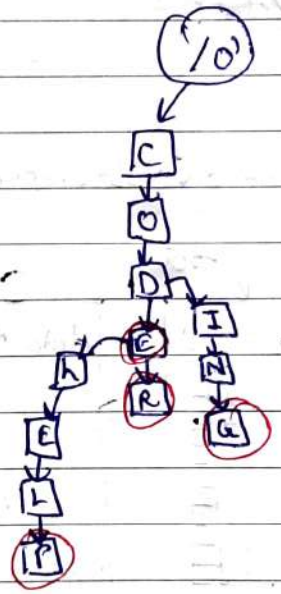
ans: cod

CODE
CODER
CODING
CODEHELP
n-1

for( )
{
for (n-1) strings
{
}
}

TC = O(n×m)

#2



Sirf 1 hi child होना चाहिए / koi terminal node होना चाहिए

```
String longestCommonPrefix(vector<string> & strs){
    string ans="";
    for (int i=0; i<str[0].length(); i++){
        char ch= strs[0][i];
        bool match=true;
```

```
//compare this character with all the remaining strings at the
same index
for(int j=1; j < strs.size(); j++) {
    //compare
    if( ch != strs[j][i] ) { || && strs[j].size() < i ||
        match = false;
        break;
    }
}
if (match == false)
    break;
else
    ans.push_back(ch);
}
return ans;
};
```

TC = $\underline{O(M \times N)}$
constant

Method2 Through tries [just a alter method but not recommended]

//copy the trie code and adding a variable as childcount.

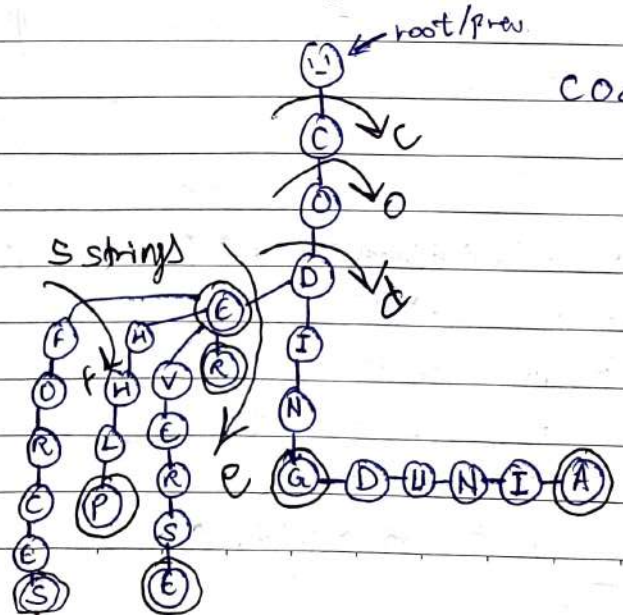
```
findLCP (string first, string& ans, TrieNode* root) {
    if( root->isTerminal ){
        return;
    }
    for ( int i=0; i<first.length(); i++) {
        char ch = first[i];
        if( root->childCount == 1) {
            ans.push_back(ch);
            int index = ch-'a';
            root = root->children[index];
        }
        else
            break;
        if( root->isTerminal)
            break;
    }
}
```

```
string longestCommonPrefix(vector<string>& strs){
    TrieNode* root = new TrieNode('-');
    // insert strings
    for (int i=0; i< strs.size(); i++) {
        insertWord (root, str[i]);
    }

    string ans="";
    string first = strs[0];
    find LCP (first, ans, root);
    return ans;
}
```

ex Search engine reccommendation

vec[] = {coding, coder, codehelp, code, codeverse, codingDuniya, codeforces}

c [7 strings]
co [7 str]
cod [7 str]
code [5 string]
codef [codeforces]



$T(m \times n \times n) = T(m n^2)$
length of string

Code print out.

↳ DP-1. L-1

Those who ~~condemn~~ forget the past are condem to repeat it.

जो भी अपने experiences हैं उन्हें सिखलो वरना फिर वही गलतीयो का सामना दोबारा करना पड़ेगा

jo ek baar calculate कर चुके हो उसे दोबारा calculate करने की need नहीं है।

DP
- → Top down [Recursion + memoisation]
- → Bottom-up [tabulation method] iterative
- → Space - Optimisation → $O(1)$

When to apply DP?

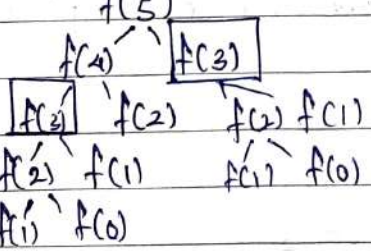
→ overlapping subproblems → Same problem को बार बार solve कर रहे हो

→ optimal substructure → Optimal solution of bigger problem can be calculated using optimal solution of smaller problem

Fibonacci 0 1 1 2 3 5 8 13 21

$f(n) = f(n-1) + f(n-2)$

Condition	Base case
$f(1) = 1$	
$f(0) = 0$	



Recursively TC = (exponential)

① Create dp array
vector<int> dp (n+1, -1);
② ans store dp
③ BC have just based
check if answer already exists

```
int fib(int n)
{
    // BC
    if( n==1 || n==0)
        return n;
    int ans = fib (n-1) + fib(n-2);
    return ans;
}
```

if (dp[n] != -1)
return dp[n];

dp[n]

ex Fibonacci number Leetcode 509

→ Recursion + Memoisation

```
int topDownSolve(int n, vector <int> &dp) {
    //base case
    if( n==1 || n==0) {
        return n;
    }

    //step3 check if answer already exists
    if( dp[n] != -1) {
        return dp[n];
    }

    //step2 replace answer with dp[n]
    dp[n] = topDownSolve (n-1, dp) + topDownSolve(n-2,
    return dp[n];
}

int fib(int n) {
    vector <int> dp (n-1, -1);
    int ans = topDownSolve (n, dp);
    return ans;
}
```

```
int bottomUpApproach(int n){
    //step-1  create dp array
    vector<int> dp(n+1,-1);
    //step2  observe base case in above solution
     dp[0] = 0;
      if(n==0)
         return dp[0];
     dp[1] = 1;
     if(n==1)
        return dp[1];

     //step3
     for (int i=2; i<=n; i++){
         dp[i] = dp[i-1] + dp[i-2];
     }
     return dp[n];
}
```

initial

-1	-1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6

step-1

0	1	✗	-1	-1	-1	-1
		1	2	3	5	8

ip: $n = 0$
dp size = 1 size
step2: $dp[0] = 0$ &
$dp[1] = 1$
is ki wajah dhaage
Phat rahaha.

BottomUpApproach: TC $O(n)$
SC. $O(n)$
} better as compared to topdown in terms of SC

Recursion: TC $O(\exp(n))$
SC $O(n)$

can sometimes gives us Top Down TLE

TC = $O(n)$
SC = $O(n) + O(n)$

can be avoided if we convert it from top Down to BottomUpApproach.

Space optimisation.

dp[i] only and only depends on dp[i-1]
dp[i-2]

(climbing ladder) DP

1) recursion 2) recursion + memoisation (top down) 3) bottomup (tabulation) 4) Space optimisation

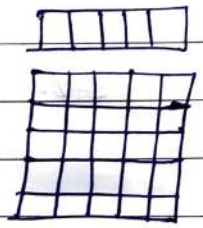
```
int spaceOptimisation(int n){
    int prev2=0;
    int prev1 = 1;

    if(n==0)
    return prev2;
    if(n==1)
    return prev1;
    int curr;

    for(int i=2; i<=n; i++){
        curr = prev1 + prev2;
        //shifting
        prev2 = prev1;
        prev1 = curr;
    }
    return curr;
}
```

Space optimisation

i → i-1, i-2 } pattern → according

1 parameter change ho रहा है तो वो 1 D DP

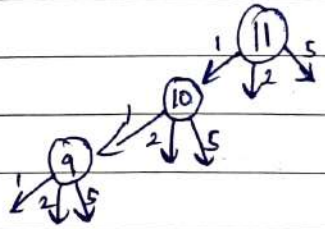
2 parameter change ho रहा है तो वो 2 D DP



↳ DP lecture 2

Adobe, Online coding

ex coin change Leetcode 322 Medium



no. of calls

for loop for all the coins

```
int coinChange(vector <int>& coins, int amount) {
    int ans = solveUsingRecursion( coins, amount);
    if( ans == INT_MAX)
        return -1;
    else
        return ans;
}

int solveUsingRecursion(vector<int>& coins, int amount) {
    //base case
    if( amount==0){
        return 0;
    }
    if( amount<0){
        return INT_MAX;
    }
    int mini = INT_MAX;
    for (int i=0; i< coins.size(); i++) {
        int ans = solveUsingRecursion( coins, amount - coins[i]);
        if( ans != INT_MAX) {
            mini = min( mini, ans+1);
        }
    } return mini;
}
```

↳ This will lead to TLE error.

```
int solveMem(vector <int>& coins, int amount, vector<int> &dp){
    //base case
    if(amount==0){
        return 0;
    }
    if(amount<0){
        return INT_MAX;
```

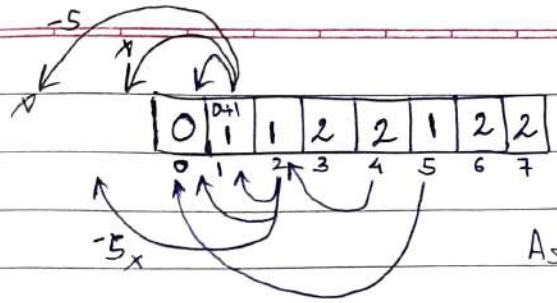
```
int coinChange (vector<int>& coins, int amount) {
    vector<int> dp(amount+1, -1);
    int ans = solveMem(coins, amount, dp);
    if (ans == INT_MAX)
        return -1;
    else return ans;
```

```
    // answer already exists
    if(dp[amount] != -1)
    {
        return dp[amount];
    }

    int mini = INT_MAX;
    for (int i=0; i < coins.size(); i++) {
        int ans = solveMem(coins, amount - coins[i], dp);
        if( ans != INT_MAX) {
            mini = min (mini, 1+ans);
        }
    }

    dp[amount] = mini;
    return mini;
}
```

Through tabulation

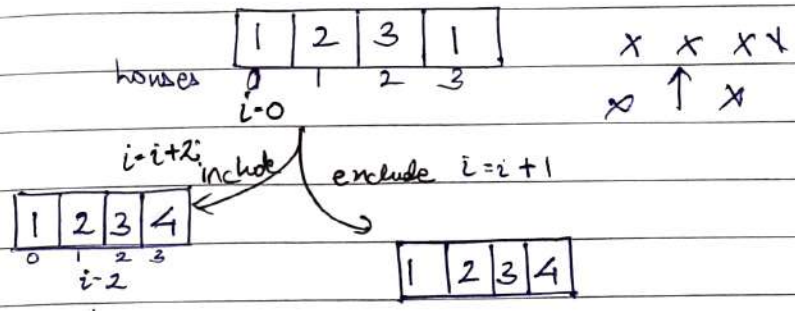
```
int solveTab (vector<int>& coins, int amount) {
    vector<int> dp (amount+1, INT_MAX);
    dp[0]=0;

    for (int i=1; i<=amount; i++) {
        for (int j=0; j< coins.size(); j++) {
            if (i- coins[j] >=0 && dp[i- coins[j]] != INT_MAX
            {
                int ans = dp[i- coins[j]];
                dp[i] = min (dp[i], 1+ans);
            }
        }
    }

    return dp[amount];
}
```

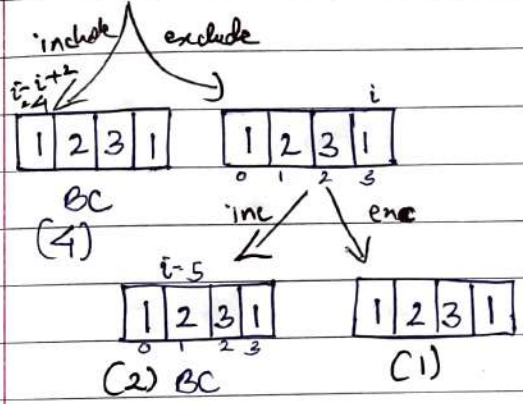
HR. House Robery - 2



As there are no pattern seen so there is no space optimisation that we can apply

Ex 198 House Robber. Medium





TLE

```
int solveUsingRecursion(vector<int> nums, int n){
    if(n<0)
        return 0;
    if(n==0)          // agar ek hi ghar hai to
        return nums[0];

    //include
    int include = solveUsingRecursion(nums, n-2) + nums[n];
    int exclude = solveUsingRecursion(nums, n-2) + 0;

    return max(include, exclude);
}
```

Optimised code in the printout.

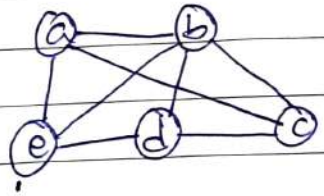
C++ library management system.

Resume
↳ Bus system
C++ graphics

shortest path using Dijkstra's Algo.

Graphs: DS → node (to store data)
+
→ edges (to connect nodes)

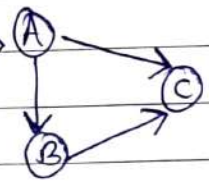
vertex
nodes: a b c d e
edges: ab, b-c, c-d,
ae, ac, be, bd,
de,



Applications: Google Maps / facebook

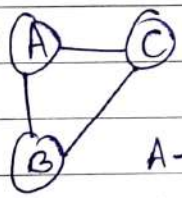
Types:

Directed graphs



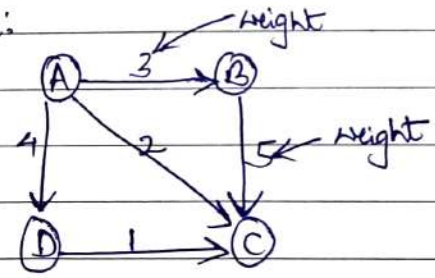
A–C ✓
C–A ✗

Undirected graphs

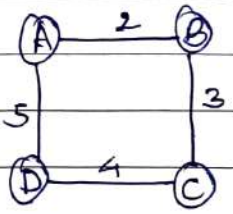


A–C ✓
C–A ✓

Directed Graph:

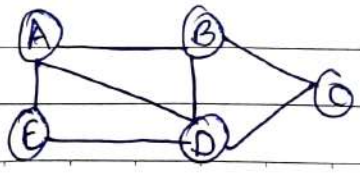


weighted graph + Directed graph
= weighted Directed graph



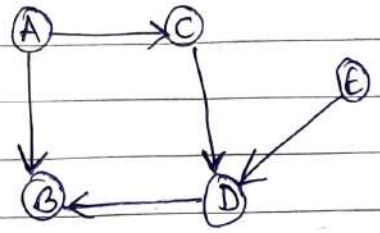
weighted undirected graph

Degree



Degree(D) = 4
Degree(A) = 3
Degree(C) = 2
Degree(E) = 2
Degree(B) = 3

In degree / Out degree

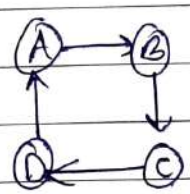


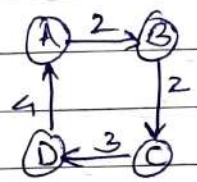
Indegree(A) = 0.
Indegree(B) = 2
Outdegree(D) = 1

path: A → C → D → B

Path kaa andar same node kaa repeat नहीं करना होता।

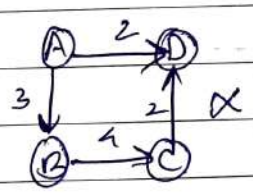
Cyclic Graph



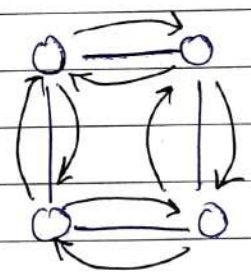


weighted + cyclic-directed graph

Acyclic Graph



weighted Acyclic directed graph.

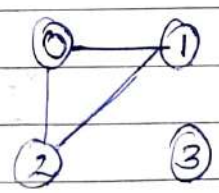


Undirected graphs are always cyclic.

Graph Implementation: Adjacency Matrix (2D array)
Adjacency List (map/list/vector/set)

	0	1	2	3
0	0	1	1	0
1	1	0	1	1
2	1	1	0	0
3	0	1	0	0

← graph.



```
int main(){
    int n;
    cout<< "Enter the no. of nodes" <<endl;
    cin>>n;

    vector<vector<int>> adj(n, vector<int>(n,0));
    int e;
    cout<< "Enter the no. of edges: " <<endl;
    cin>>e;

    for( int i=0; i<e; i++){
        int u,v;
        cin>> u >>v;
        adj[u][v]=1;
    }

    // printing

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

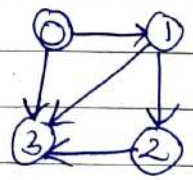
no. of nodes
3

no. of edges
6.
0 1
1 0
1 2
2 1
0 2
2 0

	0	1	2
0	0	1	1
1	1	0	1
2	1	1	0

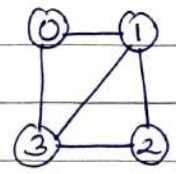
Adjacency list.

0 → {1, 3}
1 → {3, 2}
2 → {3}
3 → {}



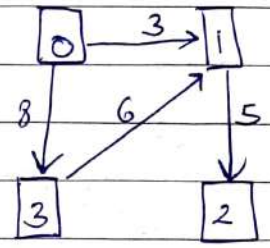
'0' से आम करा जा सकते हैं

0 → {1, 3}
1 → {0, 2, 3}
2 → {1, 3}
3 → {1, 2, 0}



3. Weighted graphs

0 → {(1, 3), (3, 8)}
1 → {(2, 5)}
2 → {}
3 → {(1, 6)}



edges:
$0 \xrightarrow{3} 1$
$3 \xrightarrow{6} 1$
$1 \xrightarrow{5} 2$
$0 \xrightarrow{8} 3$

```
class Graph {
  public:
    unordered_map<int, list<pair<int, int>>> adjlist;
                                  int

    void addEdge (int u, int v, int weight, bool direction) {
        // direction=0   undirected graph
        // direction=1   directed graph
        // create an edge from u to v.
        adjlist[u].push_back({v, weight});
        if(direction == 0){
            // undirected edge
            // create an edge from v to u
            adjlist[v].push_back({u, weight});
        }
    }

    void printAdjacentList(){
        for(auto node: adjlist){
            cout << node.first << "->";
            for(auto neighbour: node.second){
                cout << "(" << neighbour.first << "," << neighbour.second
                    << ")";
            }
```

template T } to be read in cpp reference.

```
        cout << endl;
    }
}
};
```

```
int main() {                    // source node, destination node, weight, direction
    Graph g;
    // directed input
    g.addEdge(0,1,5,1);
    g.addEdge(1,2,8,1);
    g.addEdge(0,2,6,1);
    cout << endl;
    g.printAdjacencylist();

    // undirected input
    g.addEdge(0,1,2);
    g.addEdge(1,2,0);
    g.addEdge(0,2,0);
    cout << endl;
    g.printAdjacencylist();
    return 0;
}
```

Adjacency Matrix:

SC = $O(n^2)$ nodes

SC = $O(V^2)$ or vertex

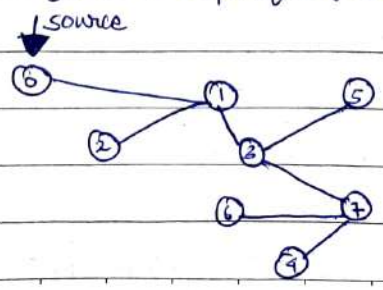
Adjacency list SC = $O(V+E)$ Average case

SC = $O(V^2)$ Worst case

Traversal:

BFS (Breth first search) level order traversal

DFS (Depth first search)



BFS

0

1

2 3

5 7

6 4

Every tree a graph ✓
but not every graph is a tree. ✗

Queue

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

Adj. list.

$0 \to \{1\}$
$1 \to \{2, 3, 0\}$
$2 \to \{1\}$
$3 \to \{1, 5, 7\}$
$4 \to \{7\}$
$5 \to \{3\}$
$6 \to \{7\}$
$7 \to \{3, 4, 6\}$

edges.

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

visited or not

Map

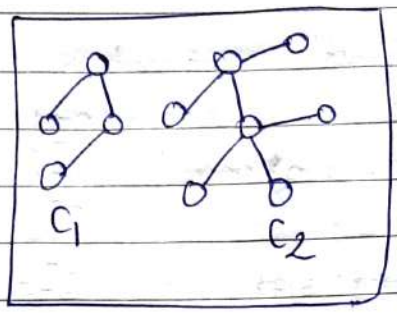
1 → ~~F~~ T	6 → ~~F~~ T
0 → ~~F~~ T	7 → ~~F~~ T
2 → ~~F~~ T	
3 → ~~F~~ T	
4 → ~~F~~ T	
5 → ~~F~~ T	

Source = 0.

```
void bfs (int src){
    queue <int> q;
    unordered_map <int, bool> visited;
    q.push (src);
    visited [src] = true;   // sirf source वाली node को visited true
                               mark किया है
    while (!q.isEmpty()){
        int frontNode = q.front();
        q.pop();
        cout << frontNode << " ";

        //insert neighbours.
        for (auto neighbour : adjlist [frontNode]){
            if (!visited [neighbour]){
                q.push (neighbour);
                visited [neighbour] = true;
            }
        }
    }
}
```

$\boxed{TC = O(V+E)}$

Linear Time Complexity



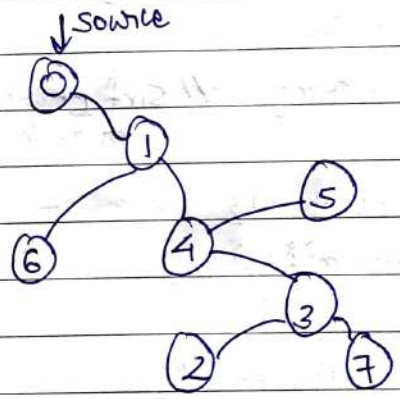
Disconnected Graph
G_1

How to handle disconnected graph

हम हर node को source मानकर call मारेंगे।

```
for (int i=0; i<=7; i++){
    if(!visited[i]){
        g.dfs(i, visited);
    }
}
```

ex DFS



0, 1, 4, 3, 7, 2, 5, 6. a simple location

or

0 1 6 4 3 7 2 5

or

any other solution

```
for (int i=0; i<=7; i++){
    if(!visited[i]){
        g.dfs(i, visited);
    }
}
```

```
void dfs(int src, unordered_map<int, bool>& visited){
    cout<<src<<", ";
    visited[src] = true;

    for(auto neighbour : adjList[src]){
        if (!visited[neighbour]){
            dfs(neighbour, visited);
        }
    }
}
```

Graph Class-2.

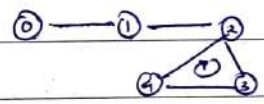
Imp

Amazon Ex online test

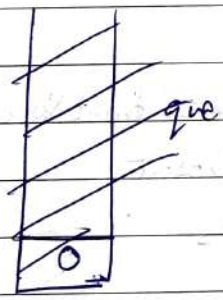
Cycle detection

- directed → BFS / DFS
- undirected → BFS / DFS

undirected graphs.



Cycle → 2-3

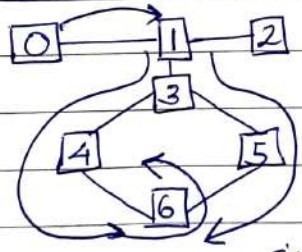


Que

0	1	2	3	4

it is a CYCLE

a node which is already visited and is not a parent.



0	1	2	3	4	5	6

visited

0	F
1	F
2	F
3	F
4	F
5	F
6	T

→ Already Visited.

sirf ekhi baar update hoga

parent

0	-1
1	0
2	1
3	1
4	3
5	3
6	4

0	1
1	0, 2, 3
2	1
3	1, 4, 5
4	3, 6
5	3, 6
6	4, 5

Parent nahi hai
+
Already visited

Scoring

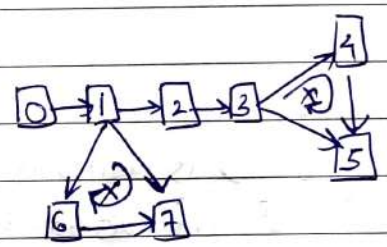
```
bool checkCyclicUsingBfs(int src, unordered_map<int,bool>& visited){
    queue<int> q;
    unordered_map<int,int> parent;
      q.push(src);
      visited[src] = true;
      parent[src] = -1;

      while (!q.isEmpty()) {
          int frontNode = q.front();
            q.pop();

          for (auto nbr : adjList[frontNode]) {
              if(!visited[nbr]) {
                  q.push(nbr);
                  visited[nbr] = true;
                  parent[nbr] = frontNode;
              }
              else {
                  // already visited
                  if(nbr != parent[frontNode]) {
                      // cycle is preset.
                      return true;
                  }
              }
          }
      }
    return false;
};
```

scoring

```
bool checkCyclicUsingDfs (int src, unordered_map <int, bool> & visited, int parent)
{
    visited[src] = true;
    for ( auto nbr : adjList[src] ){
        if ( !visited[nbr] ) {
            bool checkCyclicUsingDfs (nbr, visited, src);
            if ( checkAgekaAns == true)
                return true;
        }

        if ( visited[nbr] && nbr != parent) {
            // cyclic present
            return true;
        }
    }

    return false;
}
```

$\boxed{TC = O(\text{linear})}$

Directed Graph cycle detection → BFS [Later not now]
→ DFS



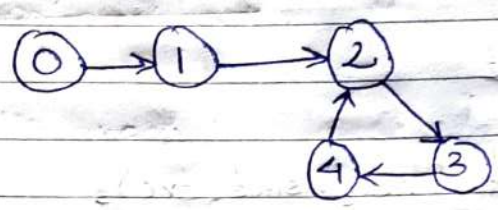
can't we apply same logic?

dfs (0, -1)
dfs (1, 0)
dfs (2, 1)
dfs (3, 2)
dfs (4, 3)
dfs (5, 4)

dfs(5, already visited
5 != parent[3]
5 != 2

So cycle is present but logically it is not present

Here we will have to take a track of how many calls are left in the recursion stack



Adj. List.

0→1	2→3
1→2	3→4
	4→2

Dfs Visited

0→~~F~~T	2→~~F~~T
1→~~F~~T	3→~~F~~T
	4→~~F~~T

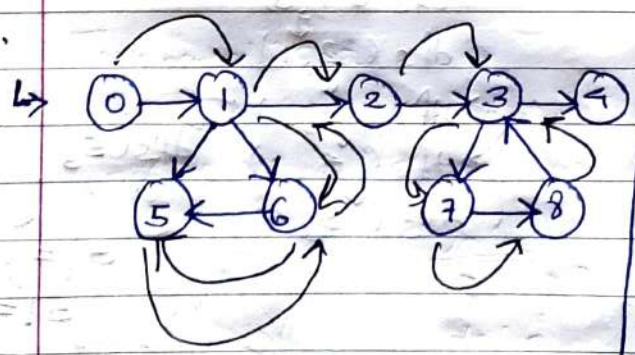
Visited

0→~~F~~T	2→~~F~~T
1→~~F~~T	3→~~F~~T
	4→~~F~~T

idhar ek baar agar koi visited ho gaya to ho gaya, अर्थात सत्य

dfs(0, -1)
↓
dfs(1, 0)
↓
dfs(2, 1)
↓
dfs(3, 2)
↓
dfs(4, 3)
↓
dfs(2, 4)] cycle

2 is visited in Dfs visited.
2's dfs visited = true
2's visited = true



Adj. List.	Visited	Dfs visited
0→1	0→~~F~~T	0→~~F~~T
1→5,6,2	1→~~F~~T	1→~~F~~T
2→3	2→~~F~~T	2→~~F~~T
3→7,4	3→~~F~~T	3→~~F~~T
4→{}	4→F	4→F
5→{}	5→~~F~~T	5→~~F~~T~~F~~F
6→5	6→~~F~~T	6→~~F~~T~~F~~F
7→8	7→~~F~~T	7→~~F~~T
8→3	8→~~F~~T	8→~~F~~T

dfs(0,-1) → dfs(1,0) → dfs(6,1) → dfs(5,6)

dfsvisited = 1 dfsvisited = 1

↓

dfs(2,1)

↓

dfs(3,2)

↓

dfs(7,3)

↓

dfs(8,7)

↓

dfs(3,_)] cycle found dfs-visited = true

visited = true.

```
bool checkCyclicDirectedGraphUsingDfs( int src, unordered_map<int, bool>
& visited, unordered_map<int, bool> dfsVisited){
    visited[src] = true;
    dfsVisited[src] = true;

    for(auto nbr: adjList[src]){
        if( ! visited[nbr]){
            bool aagekaAnswer = checkCyclicDirectedGraphUsingDfs(
            nbr, visited, dfsvisited);
            if(aagekaAnswer == true)
                return true;
        }
        if(visited[nbr] == true && dfsVisited[nbr] == true){
            return true;
        }
    }
    dfsVisited[src] = false;  → backtracking
    return false;
}
```

In graph story line matters.

red - black tree] Pay TM

Graph 2-3.

~~Topological sort.~~

imp ex Rotten oranges

1	0	0	0	0
1	1	0	0	0
0	1	1	1	1
0	0	0	0	1
0	0	0	0	1
0	0	0	0	1

Min time to rot all oranges

1 sec / adj. orange rotten

concept: bfs. or dfs

ex Given a graph find no. of component

↓

bfs or dfs

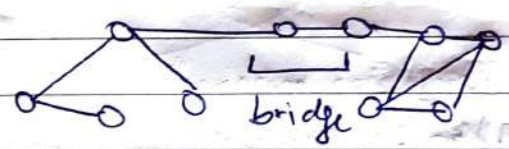
int cont ++;

imp ex no. of islands

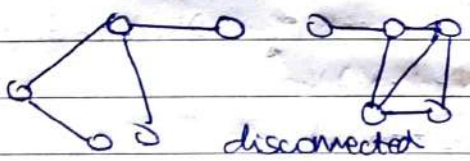
1	0	0	0	0	0
1	0	0	0	1	1
1	1	0	1	1	1
0	0	0	0	0	0

no. of islands

ex Find bridge in a graph



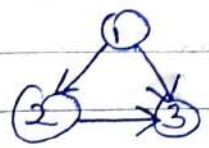
brute force: हर 5वे edge को हटाके no. of nodes. को cont kar lo.



ex Snack and ladder problem

Topological sort.

Linear ordering of vertices such that for every edge $u \to v$ u comes before v. in that ordering



Valid

$1 \to 2 \to 3$

Valid

$1 \to 3$

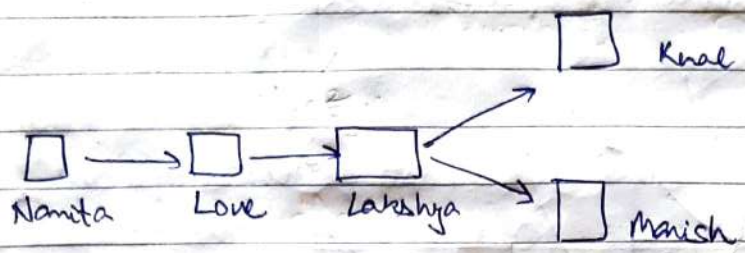
$1 \to 2$

$2 \to 3$

$2 \to 1 \to 3$

invalid

तो node जिसकी ordering में dependency खत्म हो चुकी है। तो पहले आएगी।

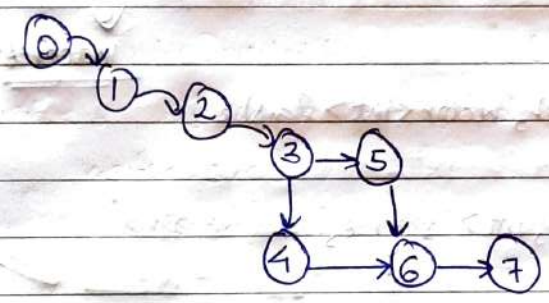


Namita, love, lakshya, Kunal, Manish] Topological sort

DFS BFS

(Way of structuring)

Topological sort can only be applied in DAG (Directed Acyclic Graph).



dfs hi hai bhai

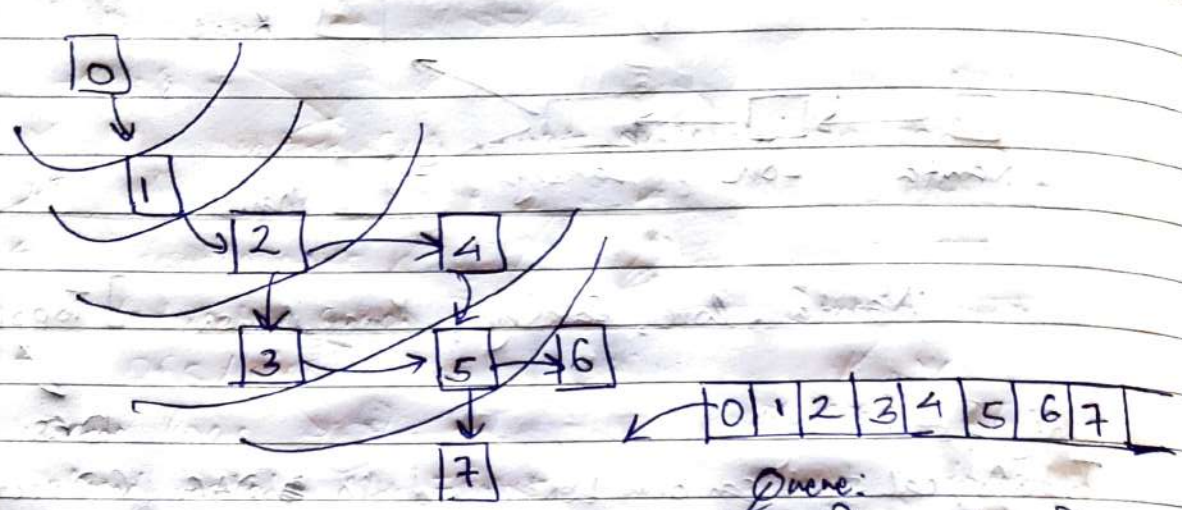
जिसका 5' के line का फर्क है।

Stack वाली line.

```
void topoSortDfs (int src, unordered_map <int, bool> & visited, stack
  <int> & ans) {
      visited = true;
      for ( auto neighbour : adjList [src] ) {
          if ( ! visited [neighbour] ) {
              topoSortDfs (neighbour, visited, ans);
          }
      }

      // while returning store the value in stack
      ans.push (src);
  }
```

↳ we have changed only this line in Dfs

Topological Sort. BFS [Kahn's algo]



Topological sort: 0 1 2 3 4 5 6 7.

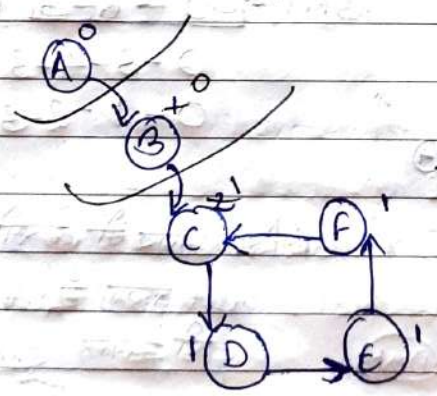
Queue:
सिर्फ वही आएगा जो आज़ाद होगा।
indegree=0

```
void toposortBfs(int n, vector<int> &ans){
    queue<int> q;
    unordered_map<int, int> indegree;

    //indegree calculate
    for(auto i: adjList){
        int src = i.first;
        for (auto nbr: i.second){
            ~~inorde~~ indegree[nbr]++;
        }
    }

    // put all the nodes inside queue, which has indegree=0.
    for(int i=0; i<n; i++){
        if(indegree[i]==0){
            q.push(i);
        }
    }
```

```
//bfs logic
while (!q.empty()){
    int fNode = q.front();
    q.pop();
    ans.push_back(fNode);

    for(auto nbr : adjList[fNode]){
        indegree[nbr]--;
        //check for 0 again
        if(indegree[nbr]==0){
            q.push(nbr);
        }
    }
}
```

↳ Cyclic Graph topological sort. Is is possible.

<u>NO</u>



A	B	

No. of ~~elements~~ nodes = 6.

no. of elements in queue = 2

So in cyclic graph there would be no. TS

TS length $\neq n$

no. of nodes in a graph

Cycle detection

directed + <u>BFS</u>

Graph → valid TS → DAG

Graph → invalid TS → cyclic graph

condition: length $\neq n$

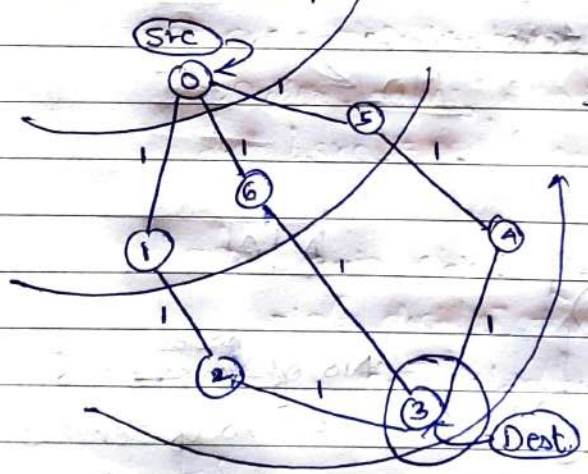
in topo Bfs

```
if (ans.size() == N) {
    cout << "It is a valid topo sort";
}

else {
    cout << "Cycle present or invalid topo sort";
}
```

Undirected Graph	Directed Graph
DFS / BFS	BFS / DFS
Same logic	BFS: invalid TS
already visited & nbr != parent	DFS: ↳ visited ↳ dfsvisited

↳ Graph 2-4

Shortest node path



shortest path

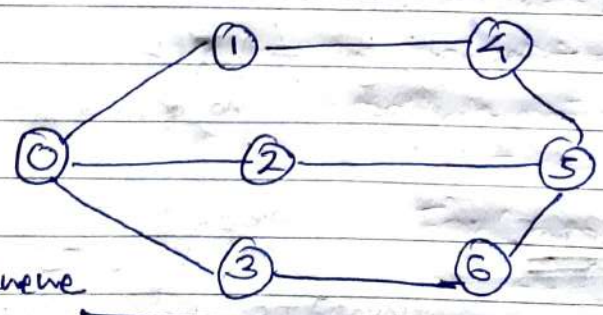
shortest $\boxed{0-6-3}$ ✓

0-5-4-3

0-1-2-3

Bfs: अगर मैं किसी भी node को visit करता हूँ for the first time वही उसका shortest path होता है।

ex



queue

0	1	2	3	4	5	6

parent	visited	
0 → -1	0 → ~~F~~	T
1 → 0	1 → ~~F~~	T
2 → 0	2 → ~~F~~	T
3 → 0	3 → ~~F~~	T
4 → 1	4 → ~~F~~	T
5 → 2	5 → ~~F~~	T
6 → 3	6 → ~~F~~	T

parent

-1	0	0	0	1	2	3
0	1	2	3	4	5	6

5 → 2 → 0] reverse

dest. source

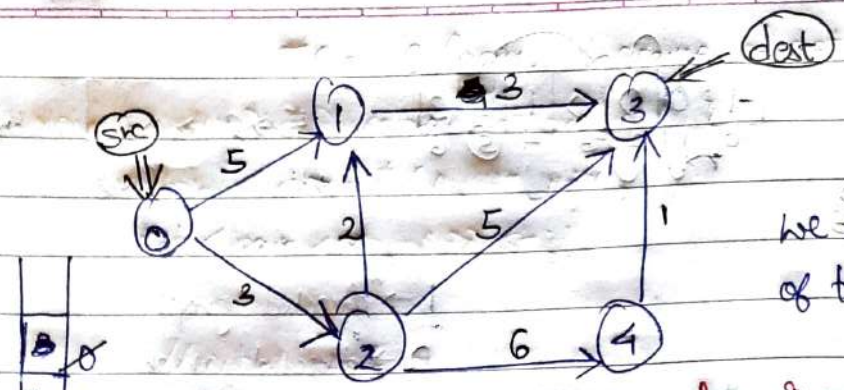
src = 0
dest = 5

1 → directed graph
0 → directed graph

```
void shortestPathBfs(int src, int dest){
    queue<int> q;
    unordered_map<int, bool> visited;
    unordered_map<int, int> parent;

    // initial steps for src
    q.push(src);
    visited[src] = 1;
    parent[src] = -1;

    while(!q.empty()){
        int fNode = q.front();
        q.pop();

        for(auto nbr: adjlist[fNode]){
            if(!visited[nbr.first]){
                q.push(nbr.first);
                visited[nbr.first] = 1;
                parent[nbr.first] = fNode;
            }
        }
    }
    // Store path in ans, after traversing in the parent array.
    vector<int> ans;
    int node = dest;
    while(node != -1){
        ans.push_back(node);
        node = parent[node];
    }
    reverse(ans.begin(), ans.end());
}
```

TC = O(n)

// 054

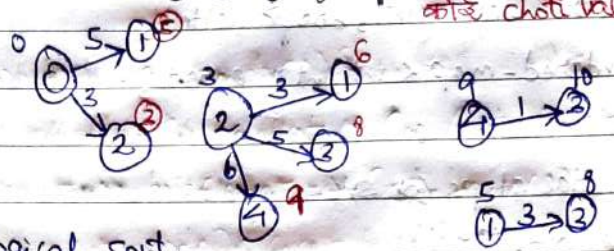


we will use here concept of topological sort.

distance array

0	1	2	3	4
∞	∞	∞	∞	∞
0	5	6	8	9

Array में update तभी करेंगे जब array में उसके corresponding कोई छोटी value मिल जाए।



i) Find topological sort.

ii) For each node in stack update the distance array.

```
void shortestpathDfs (int dest, stack<int> topOrder, int n){
    vector<int> dist(n, INT_MAX);
    int src = topOrder.top();
    topOrder.pop();
    dist[src] = 0;

    for(auto nbr : adjList[0]){
        if(dist[0] + nbr.second < dist[nbr.first]){
            dist[nbr.first] = dist[0] + nbr.second;
        }
    }

    while(!topoOrder.empty()){
        int topElement = topOrder.top();
        topOrder.pop();
        if(dist[topElement] != INT_MAX){
            for(auto nbr : adjList[topElement]){
                if(dist[topElement] + nbr.second < dist
                    dist[nbr.first] = dist[topElement] + nbr.s
            }
        }
    }
```

(node) nbr. first nbr second (dist).

0 → [1, 5] [2, 3]

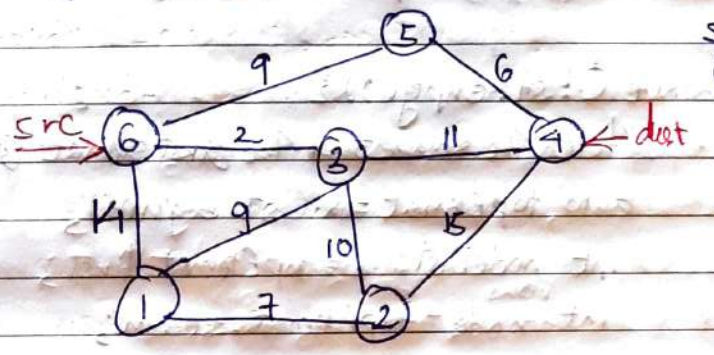
src nbr

Page No.:
Date: 29/9

```
cout << "Printing Adj: " << endl;
for( int i=0; i<n; i++){
    cout << i << "->" << dist[i] << endl;
}
}
```

imp

eg Dijkstra's Algorithm



src dst.
6 → 4

① minheap.

② set

(→ it is a data structure who stores values (unique) in sorted order.

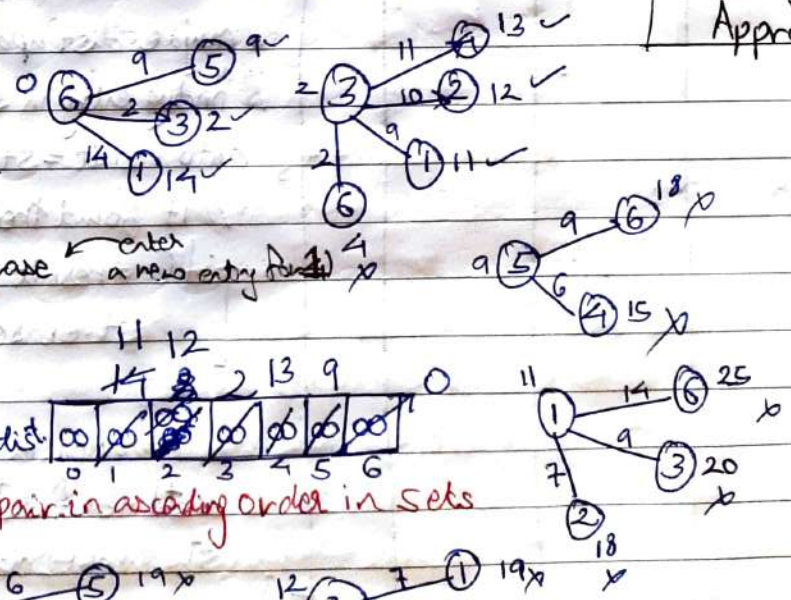
pair< int, int >
↓ dist ↓ node

node dist → ~~0~~ ~~2~~ 9
node → ~~6~~ ~~3~~ 5

Approach: Greedy Approach

set

(13, 4)
~~(12, 2)~~
~~(14, 1)~~ ~~(11, 1)~~ erase ← enter a new entry for 1
~~(2, 3)~~
~~(9, 5)~~
~~(0, 6)~~ (src)



it will sort the pair in ascending order in sets

{00, 11, 12, 2, 13, 9, 0}

[nbr, first]{
cond:

set $\leq \log(V)$
insertion $\log(n)$

Page No.: Date: 200

TC= $e \log v$. because of sets

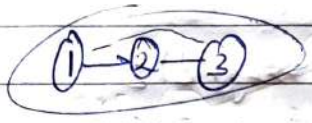
```
void shortestDistDijkstra (int src, int n) {
    vector <int> dist (n, INT_MAX);
    set <pair <int, int>> st;

    //initial steps
    dist [src] = 0;
    st. insert (make_pair (0, src));

    while (! st. empty()) {
        // fetch the smallest or the first element from set.
        auto topElement = * (st. begin());
        int nodeDistance = topElement. first;
        int node = topElement. second;

        // pop from set
        st. erase (st. begin());
        // neighbour traverse
        for (auto nbr: adjList[node]) {
            if (nodeDistance + nbr. second < dist[nbr. first])
            // mujhe dist update karna hai.
            // finding entry in set.
            auto result = st. find (make_pair (dist[nbr. first], nbr. first));
            // if found then remove
            if (result != st. end()) {
                st. erase (result);
            }

            // updation in dist array and set.
            dist[nbr. first] = nodeDist + nbr. second;
            st. insert (make_pair (dist[nbr. first], nbr. first));
        }
    }
}
```

```
cout<< "printing ans:" <<endl;
for (int i=0; i<n; i++){
    cout<< dist[i]<<" ";
}
cout<<endl;
}
```

Leetcode 547 medium

Ex Number of provinces.

TC = $O(n^2)$



no. of component

```
int findCircleNum(vector<vector<int>>& isConnected){
    mordered_map<int, bool> visited;
    int cont = 0;
    int n = isConnected.size();
    for (int i=0; i<n; i++){
        if(!isVisited[i]){
            dfs(visited, i, isConnected);
            cont++;
        }
    }
    return cont++;
}
```

1	1	0
1	1	0
0	0	1

= o/p = 2

```
void dfs(mordered_map<int, bool>& visited, int src, vector<vector<int>>
& isconnected){
    vis[src] = true;
    int size = isConnected[src].size();
    for (int col = 0; col < size; col++){
        if(src-1 != col && isConnected[src][col] ==1){
            if(!vis[col]){
                dfs(vis, col+1, isConnected);
            }
        }
    }
}
```

Leetcode 200

No. of islands.

no. of islands = 1

1	1	1	1	0
1	1	0	1	0
1	1	0	0	0
0	0	0	0	0

```
int numIslands (vector <vector <char >> &grid) {
    map <pair <int, int >, bool > visited;
    int count = 0;

    for ( int row = 0; row < grid.size(); row++) {
        int n = grid [row]. size();
        for (int col = 0; col < n; col++) {
            if( !visited [{ row, column}] && grid [row][col] == '1')
                bfs (visited, row, col, grid);
                count++;
            }
        }
    }

    return count++;
}
```

```
void bfs (map <pair <int, int>, bool> &visited, int row, int col,
    vector <vector <char>> & grid) {
        queue <pair <int, int>> q;
        // Initial steps
            q. push ({ row, col});
            visited [{ row, col }] = true;
```

```
while (!q.empty()) {
    pair <int,int> fNode = q.front();
    q.pop();
    int x = fNode.first;
    int y = fNode.second;

    // we can move in four directions.
    int dx[] = {-1, 0, 1, 0};
    int dy[] = {0, 1, 0, -1};
    for (int i=0; i<4; i++) {
        int newX = x + dx[i];
        int newY = y + dy[i];
        if( newX >=0 &&
            newX < grid.size() &&
            newY >=0 &&
            newY < grid[0].size() &&
            ! visited[{newX, newY}] &&
            grid[newX][newY] == '1') {
            q.push({newX, newY});
            visited[{newX, newY}] = true;
        }
    }
}
}
```

imp Leetcode 733 Flood Fill. Easy. [Sumology company] Hard [do not underestimate]

1	1	1
1	1	0
1	0	1

color = 2

$\Longrightarrow$

2	2	2
2	2	0
2	0	1

dfs or bfs as order does not matter

```
vector <vector <int>> floodfill (vector <vector <int>> &image, int sr, int sc,
int color){
    int oldcolor = image[sr][sc];
    map<pair <int, int> ,bool > vis;
    vector <vector <int>> ans = image;
    dfs (sr, sc, oldcolor, color, vis, ans);
    return ans;
}
```

```
void dfs(int x, int y, int old color, int newcolor, map <pair <int int>,bool>
&vis, vector <vector <int>> &arr){
    vis [{x, y}] = true; ] not needed
    arr [x][y] = newcolor;
    int dx[] = {-1, 0, 1, 0};
    int dy [] = {0, 1, 0, -1};

    for (int i=0; i<4; i++){
        int newx = x + dx[i];
        int newy = y + dx[i];

        if ( newx >0 &&
            newx < arr.size() &&
            newy >0 &&
            newy < arr[0].size() &&
            !vis [{newx, newy}] && ] old color wali cond
                                     add करना
            arr[newx][newy] == oldcolor){
            dfs (newx, newy, old color, newcolor, vis, arr)
        }
    }
}
```

similar to burning tree

3rd round or 2nd round

Amazon. Leetcode 994 Rotting Oranges [Medium] [bfs traversal]

0 → empty 1 → fresh orange
2 → rotten orange

algo:
→ queue mae सारे rotten oranges insert कर दुंगा
→ bfs 4 direction

```
int orangesRotting(vector<vector<int>>& grid){
    queue<pair<pair<int,int>,int>> ~~queue~~ q;
                                   ↳ time
    vector<vector<int>> arr = grid;
    int cont = 0;
    int ansTime = 0;
    //insert all rotten oranges in queue.

    for(int row=0; row< grid.size(); row++){
        for( int col=0; col < grid[row].size(); col++){
            if( grid[row][col] == 2){
                // rotten oranges found
                pair<int,int> coordinate = make_pair(row,col);
                q.push({coordinate, 0});
            }
            // total fresh ~~con~~ orange cont ++
        }
    }

    while(!q.empty()){
        auto fNode = q.front();
        q.pop();
        int x = fNode.first.first;
        int y = fNode.first.second;
        int time = fNode.second;
        int dx[] = {-1, 0, 1, 0};
        int dy[] = {0, 1, 0, -1};
```

1 As course

queue < pair < pair <int, int>, int >> q

coordinates (x,y) → time

10 quest. each

Page No.: Date: 306

```
for(int i=0; i<4; i++){
    int newX = x + dx[i];
    int newY = y + dy[i];

    if( newX >=0 &&
        newX < arr.size() &&
        newY >=0 &&
        newY < arr[0].size() &&
        arr[newX][newY] == 1){
        pair<int,int> newCoordinates = makepair(newX, newY);
        ansTime = max(ansTime, time+1);
        q.push({newCoordinates, time+1});
        // mark rotten
        arr[newX][newY] = 2; // fresh orange cont--;
    }
}
}

// check for fresh orange
for(int i=0; i<arr.size(); i++){
    for( int j=0; j<arr[0].size(); j++){
        if( arr[i][j] == 1){
            return -1;
        }
    }
}
```

This could also be solved without use of two for loop
cont of fresh orange left

```
return ansTime;
}
```

↳ Graph Class-6

ex Course Schedule 207 leetcode

[ ] → [a b] b → a → c → d
[c a] a is depended on b
[d, c] c is dependent on a
d is dipedent on c

topological sort
valid → DAG b → a → c → d ✓
↳ Cyclic (invalid) graph b → a → c
cycle ✗ finish not possible

valid ( ) } condition check
top length = n } for cycle detection

ex Course Schedule II
by using vectors

ex Word Ladder
hit
26 opt 26 opt } verify it with word list.
26 opt
bfs
↳ shortest path.

hit → hot → dot → dog → cog
ait bit cit — sit ✗
hat hbt hct (hit) hzt ✓
hia, hib, hic — hiz ✗
aot, bot, cot — zot ✗
hat hbt hzt ✗ 2 ways
hoa hob dot hoz ✓ — mark hit as visited
✓ we will use this — delete hit.

<cog, 5> → finally 5
<dog, 4>
<dot, 3>
<hot, 2>
<hit, 1>

we will use here unordered set
→ can also be solved using maps

```
int ladderlength (string beginword, string endword, vector <string> & wordlist){
    queue <pair < string, int>> q;
    q.push ({beginword, 1});
    unordered_set<string> st(wordlist.begin(), wordlist.end());
    //jo bhi word queue mae insert karuga, to usko set mae se remove
      karuga
    st.erase (beginword);
    while (!q.empty()){
        pair <string, int> fNode = q.front();
        q.pop();
        string currString = fNode.first;
        int currCount = fNode.second;

            // check kaie destination toe nahi [illegible]
            if(currString == endword)
                return currCount;

            for (int index=0; index < currstring.length(); index++){
                //hai index par jo value hai usko main 'a' to 'z' se replace
                  karuga
                char originalCharacter = currString[index];
                for (char ch='a'; ch <='z'; ch++){
                    currString[index] = ch;
                    // check in the wordlist.
                    if(st.find (currString) != st.end()){
                        q.push({currString, currCount+1});
                        st.erase (currString);
                    }
                }
                // bring back the curr string to its original state
                currString[index] = originalCharacter;
            }
        }
    return 0;
}
```

The question is little bit tricky

pair <int, pair<int, int>>

Priority queue — coordinates

Ex Path with minimum effort - Leetcode 1631. Medium

	0	1	2
0	1	2	2
1	3	8	2
2	5	3	5

{6, (2, 0)}
{4, (2, 0)}
{4, (0, 2)}
{3, (1, 2)}
{3, (1, 1)}
{1, (0, 2)}
{2, (1, 0)}
{1, (0, 1)}
{0, (0, 0)}

[Shortest distance]

// We will use here dikstra algo using priority queue = Greedy approach

	0	1	2
0	0	∞1	∞1
1	∞2	∞7	∞1
2	∞4	∞6	∞4

```
int minimumEffortPath(vector<vector<int>> heights) {
    priority_queue <pair <int, pair <int,int>>, vector<pair <int, pair <int,
    int>>>, greater <pair <int, pair<int,int>>>> pq;
    vector <vector <int>> dist (heights.size(), vector <int> (heights[0].size(
    INT_MAX));
    pq.push({0, {0,0}});
    dist[0][0] = 0;
    while (!pq.empty()) {
        auto fNode = pq.top();
        pq.pop();
        int frontNodeDifference = fNode.first;
        int x = fNode.second.first;
        int y = fNode.second.second;
        //check ab tak toh hai pahuch gaya
        if(x == heights.size()-1 &&
            y == heights[0].size()-1) {
            return dist[x][y];
        }
        int dx[] = {-1, 0, 1, 0};
        int dy[] = {0, 1, 0, -1};
        for (int i=0; i<4; i++) {
            int newX = x + dx[i];
            int newY = y + dy[i];
```

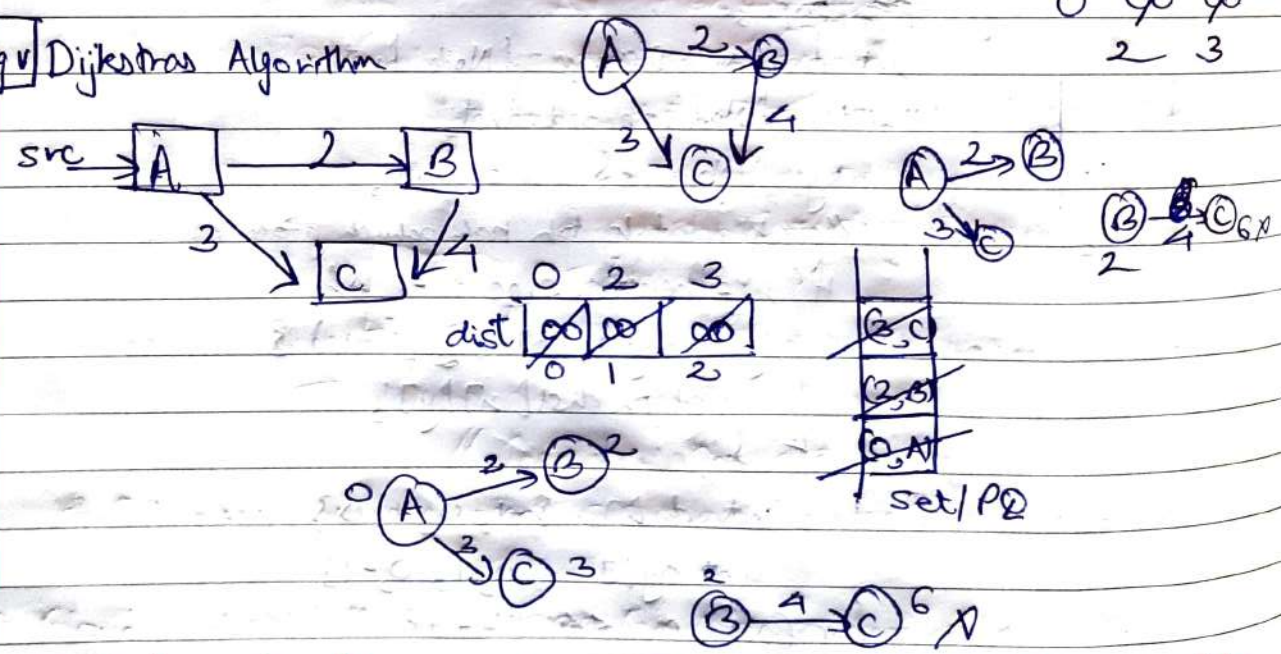
koi toh test case fataega.? Dijkstra -ve cycle

```
if( newx >=0 && newy >=0 &&
    newx < heights.size() &&
    newy < heights[0].size()) {
  int currDifference = abs(heights[x][y] - heights[newx][newy]);
  int newMax = max(frontNodeDifference, currDifference);
    if( newMax < dist[newx][newy]) {
      dist[newx][newy] = newMax;
      pq.push({newMax, {newx, newy}});
    }
  }
}
return 0;
}
```

Graphs. Lecture 7.

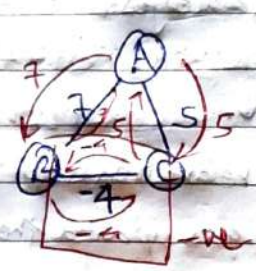
dist A B C
0 ∞ ∞
2 3

TC - elogv Dijkstras Algorithm



Dijkstra algorithm can be applied in a circle, but it cannot be applied in a negative weighted graph.

In non directed -ve edge



-ve cycle so ~~dijs~~ Dijstra cannot be applied.

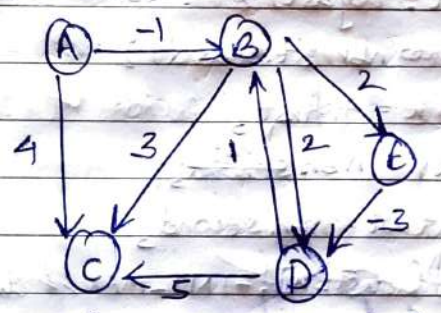
rare ex

Bellman-Ford Algo

↳ relaxation n-1 times (n nodes)

Then after we get all the shortest path

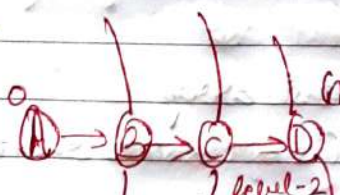
↳ dist[u] + wt < dist[v] } this is known as relaxation



A→B -1
B→E 2
B→C 3
B→D 2
D→B 1
A→C 4
E→D -3
D→C 5

	A 0	1 B	2 C	3 D	4 E
dist	0	-1 ~~∞~~	2 ~~∞~~	-2 ~~∞~~	1 ~~∞~~

1st pass
2nd pass



before or at (n-1) pass, we will get our right ans.
↳ 1 more time to indicate that it is a -ve cycle

level-1 after pass 1 we are able to assign optimal paths only at level -1.
pass 1

-ve cycle हैं तो infinite passes में update होता रहेगा।

तुम कहा जा रही हो?

BF Algo
↳ ① (n-1) times relaxation on all edges.
↳ ② to find/ detect -ve graph.
↓

elogv <<< TC = O(~~elog~~ n×V) run relaxation step 1 more time and if dist. gets updated then, there's a -ve cycle.

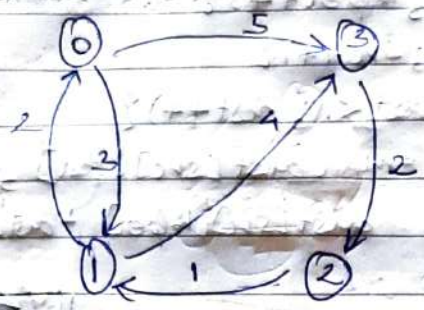
~~dijkstra~~ dijkstra algo

```
Void bellmanFordAlgo(int n, int src) {
    // assuming directed weighted graph
    vector<int> dist(n, INT_MAX);
    dist[src]=0;
    // n-1 relaxation step
    for(int i=0; i<n-1; i++) {
        // for all edges
        for(auto t : adjlist) {
            for(auto nbr : t.second) {
                int u = t.first;
                int v = nbr.~~second~~ first;
                int wt = nbr.second;
                if(dist[u]!=INT_MAX && dist[u]+wt<dist[v])
                {
                    dist[v] = dist[u] + wt;
                }
            }
        }
    }

    // then print the dist. array.
```

Tillnow we have seen SSSP (single source shortest path)

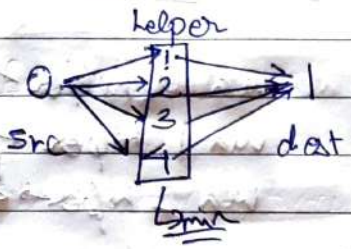
Floyd Warshall MSSP (multiple source shortest path)

1 e 9] what is this



dest.

src	0	1	2	3
0	0	∞	∞	∞
1	∞	0	∞	∞
2	∞	∞	0	∞
3	∞	∞	∞	0



floyd

↳ using helper nodes (permutations & combinations) I will explore all the nodes ad the shortest
↳ min

```
void floydwarshal (int n) {
    vector<vector<int>> dist (n, vector<int>(n, 100000));
    // diagonal pr zero mark kardo.
    for (int i=0; i<n; i++) {
        dist[i][i]=0;
    }

    // graph kae according dist. insert kar dia hai.
    for (auto t: adjlist) {
        for (auto nbr: t.second) {
            int u = t.first;
            int v = nbr.first;
            int wt = nbr.second;
            dist[u][v] = wt;
        }
    }
```

Hot topics: Quick Sort Imp
Heap Sort
merge sort in array merge sort in LL
Dijstra Algo.

Page No.
Date: 3/4

Q not asked [dp > graph]

```
for (int helper = 0; helper < n; helper++) {
    for (int src = 0; src < n; src++) {
        for (int dest = 0; dest < n; dest++) {
            dist[src][dest] = min(dist[src][dest],
                                  dist[src][helper] + dist[helper][dest])
        }
    }
}
```

kind of dp as we are using historical data for present computational

16 (4x4) comp

via 0 helper → 16

via 1 helper → 16 comp

via 2 helper → 16 comp

(n-1) helper

↳ Graph class 8

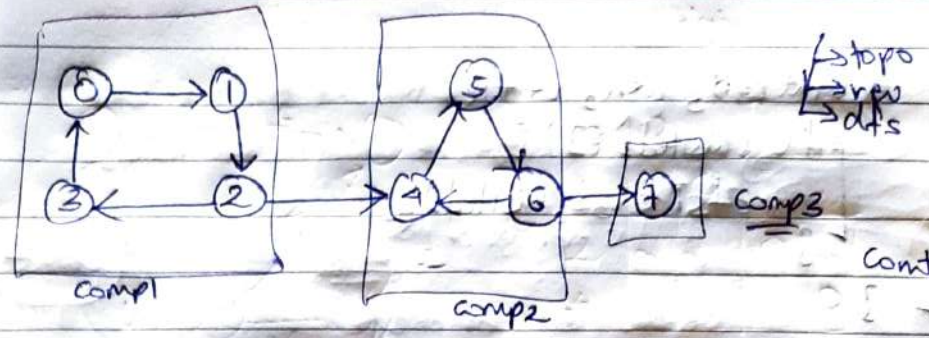
Dijkstra
↳ implementation
↳ PQ/set - re-evaluation ✗
↳ PQ/set - re-evaluation ✓ (allowed) } Both are correct.

PQ/set
directed -ve weight ✓
re-evaluation (allowed)
provided -ve cycle does not exist.

ex Strongly connected components (SCC). Kosaraju algorithm.
prerequisite: topological sort
: dfs
: adjacency list reverse

from any and every vertex we can travel to any other vertex.
↳ cont
or print.

H.W
Ex graph coloring



topo
rev
dfs

Count of SCC = 3

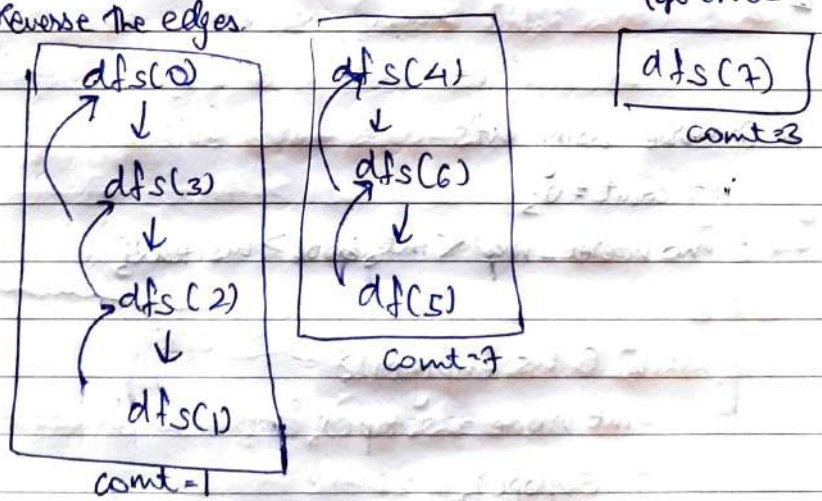
1) dfs(0)
↓
dfs(1)
↓
dfs(2)
↓ ↓
dfs(3) dfs(4)
↓
dfs(5)
↓
dfs(6)
↓
dfs(7)

- find ordering in which we should traverse the graph
- reverse the edges] all
- count the components

0
1
2
4
5
6
7
3

topo order

2) Reverse the edges.




```
int countSCC (int N) {
    stack <int> S;
    mordered_map <int, bool> visited;
    // find topological ordering
```

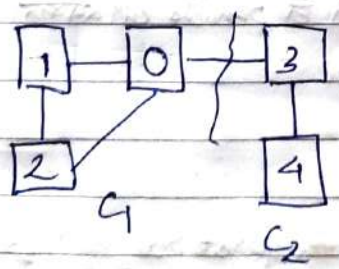
```
for(int i=0; i<n; i++){
    if(!visited[i]){
        dfs1(i, s, visited);
    }
}

// reverse all the edges
unordered_map<int, list<int>> adjNew;

for (auto t : adjlist){
    for(auto nbr : t.second){
        int u = t.first;
        int v = nbr;
        //v->u insert
        adjNew[v].push_back(u);
    }
}

// traverse using dfs
int count=0;
unordered_map<int, bool> visited2;

while(!s.empty()){
    int node = s.top();
    s.pop();
    if(!visited2[node]){
        cout << "Printing" << count+1 << "th SCC:";
        dfs2(node, visited2, adjNew);
        cout << endl;
        count++;
    }
} return count;
}
```

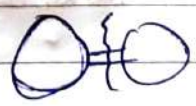
$TC = O(V+E)$

```
void dfs2 (int src, unordered_map <int, bool> & visited, unordered_map <int,
list <int> & adjNew) {
        visited[src] = true;
        cout << src << " ";

        for( auto nbr : adjNew[src]){
            if( !visited[nbr]){
                dfs2 (nbr, visited, adjNew);
            }
        }
}
```

```
void dfs1 (int src, stack <int> & s, unordered_map <int, bool> & vis) {
    vis[src] = true;
    for (auto nbr : adjList[src]) {
        if( !vis[nbr]) {
            dfs1 (nbr, s, vis);
        }
    }
    s.push (src);
}
```

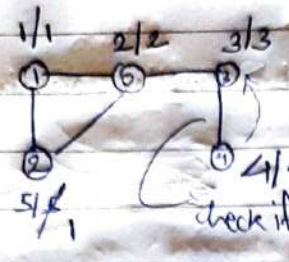
ex Bridge in a graph [tarjan's algorithm]





#1 try removing each each and count the components. The no. of components should increase.

$O[E \times (V+E)]$



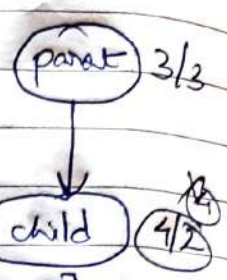
time of insertion

low time → lowest insertion time

check if it is a bridge or not.

$4 > 3$

so this is a bridge



कभी भी low update करो तो
parent को देखकर update मत करना

low[child] > low[parent]
↳ parent exists

path possible but
visited

low[child] <= low[parent]
that means child तक पहुंचने का कोई और रास्ता भी था।

Observations:-

a/b c/d

1) P → C

d > b ✗

↳ ! (low[child] > low[parent])

↳ c tak कहीं और से भी पहुंचा जा सकता है।

c – d is not a bridge

a/b c/d

2) P — C

d > b

low[child] > low[parent]

↳ c tak sirf p से ही pahuncha जा सकता है।

↳ c – p is a bridge

3) for (nbr – adj[])

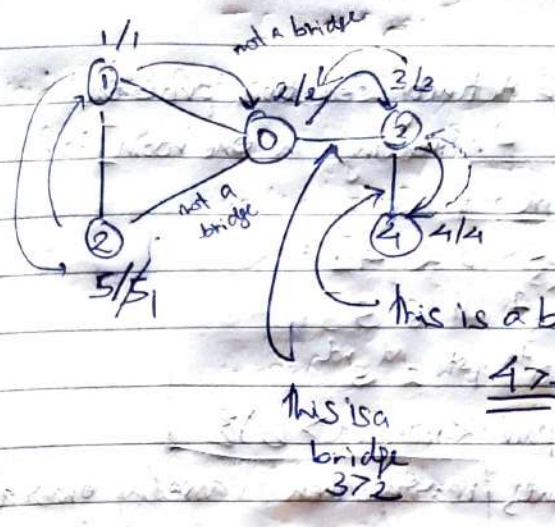
↳ no comparison with parent.

→ nbr → !visited call dfs()

↳ nbr → visited && nbr not a parent → low → update

low update

(check for bridge)



wapas jaaro
↳ low update
↳ check for bridge

Agar back tracking करते time कोई chota rasta exist नहीं करता तो वो brige है।

```
void findBridges (int src, int parent, int & timer, vector <int> & tin,
vector <int> & low, unordered_map<int, bool> vis){

    vis [src] = true;
    tin [src] = timer;
    low [src] = timer;
    timer++;

    for (auto nbr : adjlist[src]) {
        if( nbr == parent) {
            continue;
        }

        if( ! vis[nbr] ) {
            // dfs call
            findBridges (nbr, src, timer, tin, low, vis);
            // low update
            low[src] = min (low[src], low [nbr]);
            // check for bridge
            if( low [nbr] > tin [src] ) {
                cout << nbr << " - " << src << " is a bridge";
            }
        }

        else {
            low[src] = min(low[src], low[nbr]);
        }
    }
}
```

Google

Ex) 1192 Critical connections in a Network
Hard

↳ Bonus lecture Greedy [Activity selection problem]
[n meetings in One room]

arrival time [5 8 2 4]
departure time [7 11 6 5]
0th 1st 2nd 3rd

```
bool comp(pair <int,int> a, pair <int,int> b){
    return a.second < b.second;
}
```

rearrange them in ascending order as per departure time (4,5), (2,6), (5,7), (8,11)

```
int solve (int n, int arr[], int dept[]){
    vector <pair <int, int>> data;
    for( int i=0; i<n; i++){
        data.push_back ({arr[i], dept[i]});
    }

    sort (data.begin(), data.end(), comp);
    int cnt=1;    // we have already taken train 0 into consideration
    int lastDept = data[0].second;
    for( int i=1; i<n; i++){
        if( data[i].first >= ~~data[i-1].second~~ lastDept){
            // current train can be included
            cnt++;
            lastDept = data[i].second;
        }
    }
    return cnt;
}
```

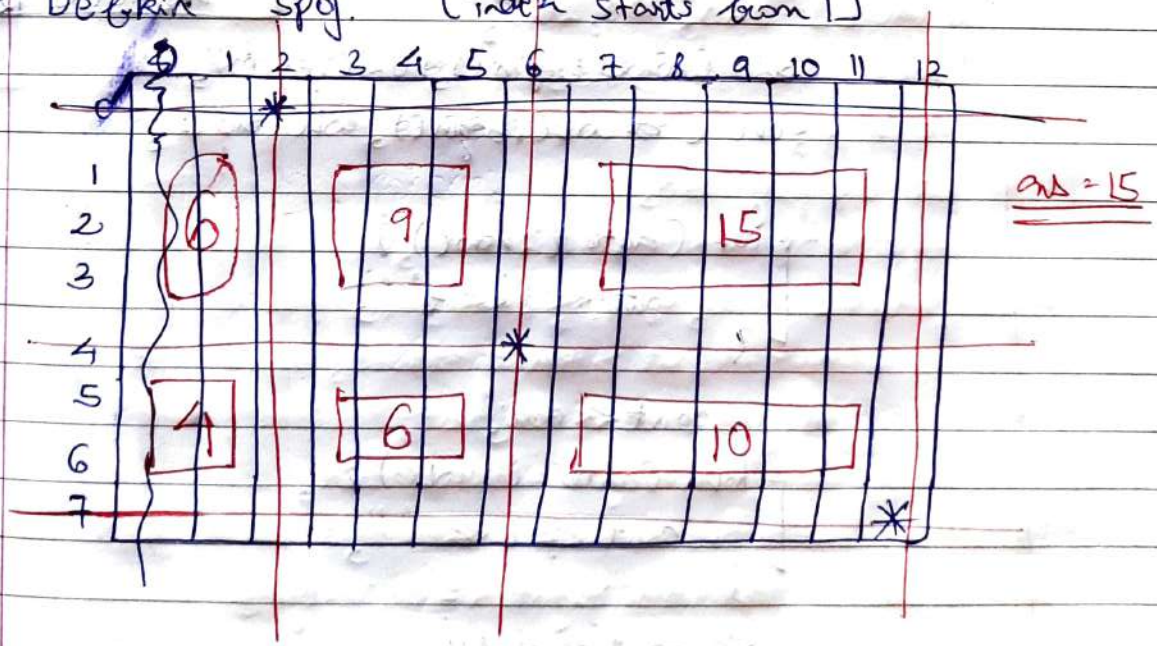
ans = 3

Ex prices = {2, 9, 8, 4, 6, 7} lollipop [Shop candies] GFG
2+4=6 1 llp
Min. पैसे खर्च करके सारी की सारी lolipop खरीदनी है। ↳ 2 llp free

{2, 4, 6, 7, 8, 9}
buy free
(buy > free) to break the loop.

```
int main(){
int prices[] = {2,8,6,9,4,7};
  int n=6;

  sort (prices, prices+n);
    int amt=0;
    int buy=0;
    int free = n-1;

    while (buy <= free){
      amount += prices[buy];
      buy++;
      free -=2;
    }

    cout << amount << endl; //3
    return0;
}
```

ex: Defkin spoj. (index starts from 1)




```
int main (){
  int x,y,n;
    cin>> x>>y>>n;
    int t = n;
```

```
vector <int> rowC;
vector <int> colC;

rowC.push_back(0);
colC.push_back(0);

while (t--){
    int a, b;
    cin>>a>>b;
    rowC.push_back(a);
    colC.push_back(b);
}

// size of 2D grid
rowC.push_back(x+1);
rowC.push_back(y+1);

sort (rowC.begin(), rowC.end());
sort (colC.begin(), colC.end());

for (auto i:rowC){
    cout<<i<<" ";
}

cout<<endl;
for (auto i:colC){
    cout<<i<<" ";
}

<<cout<<endl;
int maxlen = INT_MIN;
for (int i=1; i<rowC.size(); i++){
    int a= rowC[i-1];
    int b= rowC[i];
    maxlen= max(maxlen, b-a-1);
}
```

15	8	3
3	8	
11	2	
8	6	

0	3	8	11	16
0	2	6	8	9

answer is 12.

```
int maxWidth = INT_MIN;
for( int i=1; i<col.size(); i++){
    int a = col[i-1];
    int b = col[i];
    maxWidth = max(maxWidth, b-a-1);
}

cout << "ans is" << maxWidth;
}
```

Q2 min. cost of ropes

2 3

2+3 (a+b) cost.

arr[] = {4, 3, 2, 6} 5 rs

{4, 5, 6} 9 rs

{9, 6} 15 rs

{15} 29 Rs

we will use here min heap O(1) TC:

```
int arr[] = {4,3,2,9};
int n = 4;
priority_queue<int, vector<int>, greater<int>> pq;
for( int i=0; i<n; i++){
    pq.push(arr[i]);
}

int cost = 0;
while(pq.size()!=1){
    int first = pq.top();
    pq.pop();
    int second = pq.top();
    pq.pop();
    cost += first + second;
    int newLength = first + second;
    pq.push_back(newLength);
}
```

```
cout << cost;
//cost is 29
```

HW
Ex. Huffman encoding

ex Fractional knapsack 0/1

Value	60	100	120	Capacity : 50kg
Weight	10	20	30	max. Value
Value/weight	6	5	4	fractional

$60 + 100 + \frac{+20}{30} \times 20 = 80$

$= 240$

Value/weight → sort (desc order)

ans 240

```
bool cmp (pair<int, int> a, pair<int, int> b){
    double ratio1 = ((1.0) * a.first) / a.second;
    double ratio2 = ((1.0) * b.first) / b.second;
    return ratio1 > ratio2;
}

int main(){
    int val[] = {60, 100, 20};
    int wt[] = {10, 20, 30};
    int n = 3;
    int capacity = 50;
    vector<pair<int, int>> data;
    for (int i=0; i<n; i++){
        pair<int, int> item = data[i];
        int itemValue = item.first;
        int itemWeight = item.second;
        // entire inclusion wala case
        if (itemWeight <= capacity){
            // add karo value ko
            totalValue += itemValue;
            // update karo capacity ko
            capacity = capacity - itemWeight;
        }
```

```
else {
    // fraction include karo
    // update the value
    int ratio = ((1.0) * itemValue) / itemweight;
    int valueToAdd = ratio * capacity;
    totalValue += valueToAdd;

    //update capacity
    capacity = 0;
  }
}

cout << "Answer is" << totalValue << endl;
return 0;
}
```

ex Hackerrank [2 times in interview]

Chocolate distribution problem

(packets) N = 8 M = 5 (students)

arr = {3, 4, 1, 9, 5, 6, 7, 9, 12}

↳ no of chocolates

ans = 6

Distribute chocolate among m students

↳ st. exactly 1 packet.

↳ diff. b/w max no. of ~~chto~~ chocolates given to a student & min no. of chocolates given to a student is minimum

sorted {1, 3, 4, 7, 9, 9, 12, 5 [}

differences 8, 6, 8, 49 min diff : 6

min

```
int main() {
    int n = 8;
    int m = 5;
    int arr[] = {2, 4, 1, 9, 56, 7, 9, 12};
    //sort
    sort (arr, arr+n);

    //check for differences in m size window
    int i=0;
    int j= m-1;
    int diff = INT_MAX;
    while (j<n) {
        int newDiff = arr[j] - arr[i];
        diff = min (diff, newDiff);
        j++;
        i++;
    }
    cout << "Answer is" << diff << endl; // 6
    return 0;
}
```

↳ Bonus lecture 2. Sliding window Techniques.

(2 pointer approach)
(doubly ended queue) } we have already practiced

The sliding window technique is a common ~~subset~~ approach used to solve problems that involve finding or analyzing a subset of elements in a larger data structure, such as an array or string. Here are the key patterns often used in sliding window statements:

↳ Fixed size window
↳ Variable size window
↳ Two pointer approach
↳ Optimised sliding window

~~Fixed size~~ Sliding window ~~leetcode~~ 239 Hard

Fixed Size window:

- Maintain a window of a fixed size (usually a contiguous subset of elements).
- Slide the window one element at a time to process the data.
- Useful for finding the maximum, minimum or some other calculation within a fixed-sized window.

Problem statements

- Given an array of integers and a window size, find the maximum sum of any subarray of the given window size.
- Given a string, find the length of the longest substring containing only unique characters within a fixed size window.
- Given an array of temperatures, find out the minimum number of days it takes for a contiguous window of size k to have increasing temperatures.
- Given a binary array (containing only 0's and 1's), find the maximum number of consecutive 1s with at most k 0s allowed in the window.
- Given a string of characters, find the smallest window length that contain the all the characters of a given pattern.

Variable size window

- Start with a small window and gradually expand ~~out~~ or shrink it based on certain conditions.
- ~~Usually~~ Useful when the size of the window is not fixed and depends on the problem requirements.
- Often used to find a ~~subset~~ subarray or substring satisfying specific conditions (eg. sum, ~~problem~~ product, or a certain property).

Problem statements

- Given a array of positive integers and a target sum, find the minimum subarray length whose sum is greater than or equal to the target.

- Given a string, find out the longest sub-string with at most K distinct characters

- Given an array of non-negative integers, find the subarray of length k with the maximum average value.

- Given a string, find the length of the longest substring that can be formed by the repeating exactly k times.

- Given an array of integers, find the maximum product of a subarray with size at most k.

2 pointer approach

- Use two pointer to define the boundaries of the windows
- One pointer usually represents the start of the window, and the other represents the end.
- Move the pointers within the data structure based on certain conditions.
- Commonly used when the problem ~~requirements~~ requires finding a target sum, a pair of elements, or a ~~substring~~ subarray with specific characteristics

- Problem Statements
- Given an array of integers and a target sum, find two numbers that add up to the target.
- Given a sorted array of integers, find two numbers whose absolute difference is closest to a given target value.
- Given a string, find the longest palindrome string.
- Given a string, find the length of the longest substring without repeating characters.

- Given an array of integers, find out the subarray with the given sum.

Optimised window.

- Apply additional optimised techniques to improve the sliding window solution.

- Examples include precomputing sums or frequencies, utilising data structures like heaps or hash-maps or skipping unnecessary calculations.

- Helps to reduce the time complexity and optimised the overall solution

- Given an array of integers, find out the subarray with the maximum sum using the kadane's algorithm.

- Given a string, find the longest substring with atmost k distinct characters using a hash map for character frequency tracking.

- Given an array of integers, find the median of every sliding window of size k using a min heap and max heap.

- Given a binary array (containing only 0's and 1's), find the minimum size of a subarray that needs to be flipped to make all elements 1.

- Given an array of integers and a target sum, find the subarray with the smallest length that sums up to the target using a two-pointer approach and dynamic window resizing.

- Given a string, find the longest substring that ~~cont~~ contains ~~the~~ at most k distinct characters and has at least k/2 repeating characters.

- Given an array of integers, find out the maximum product of subarray with size at most k and a minimum product of a subarray with size at most k.

- Given two strings, find the minimum window in the first string that contains all the characters of the second string, containing multiple possible windows.

- Given a string, find the longest palindromic substring with at-most k deletions allowed.

- Given an array of integers, find the maximum sum of k non-overlapping sub-arrays.

Fixed sized

Leetcode 239 Sliding Window (Hard)

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

1) process 1st window
2) process remaining windows

```
vector <int> maxSlidingWindow (vector <int> & nums, int k) {
    deque <int> dq;
      //we will store indexes in dequeue
    vector <int> ans;
        // process the first k size window
    for (int i=0; i<k; i++) {
        while (!dq.empty() && nums[dq.back()] <nums[i])
            dq.pop_back();
            // insert elements
    dq.push_back(i);
    }

    //ans store results for 1st window
    ans.push_back (nums[dq.front()]);
```

can also be solved using - doubly ended queue, priority que, arrays

```
// remaining windows
for (int i=k; i < nums.size(); i++){
    //removal
    if(! dq.empty() && i-k >= dq.front())
        dq.pop_front();

    // additional
    while (! dq.empty() && nums[dq.back()] < nums[i])
        dq.pop_back();
    // insert elements
    dq.push_back(i);
    //ans store
    ans.push_back(nums[dq.front()]);
}

return ans;
}
```

ex Twosum (easy) Leetcode 1.

2	7	11	15
0	1	2	3

#1 2 for loops brutefore

#2 sort the array.

i ↓ ↓ j

2	7	11	15

$2+15 = 17 > 9$

index store करना पड़ेगा

```
static bool cmp (pair<int,int> a, pair<int,int> b){
    return a.first < b.first;
}

vector <int> twoSum (vector<int> &nums, int target) {
    vector <pair <int,int>> data;
    vector <int> ans;
```

5 ways
→ sliding window

: Longest palindromic substring [expand around the center]

```
for ( int i=0; i< nums.size ();i++) {
    int val = nums[i];
    int index = i;
    data.push_back({val, index});
}

sort (data.begin, data.end, cmp);
int start = 0;
int end = data.size ()-1;
while (start < end) {
    if( data[start].first + data[end].first == target){
        ans.push_back (data[start].second);
        ans.push_back (data[end].second);
        break;
    }
    else if ( data[start].first + data[end].first > target){
        end--;
    }
    else {
        start++;
    }
}
return ans
}
```

TC- O(n log n)
can also be solved using binary search

Variable size window

ex Leetcode 209
Minimum size of sub-array sum

start, end

nums	2	3	1	2	4	3
	0	1	2	3	4	5

```
int minSubArrayLen (int target, vector <int> nums) {
    int i = 0;
    int j = 0;
    int sum = 0;
    int la = INT_MAX;
```

```
while (j < nums.size()) {
    //include current number
    sum = sum + nums[j];
    //naval pass, ek window ka sum ready hai.
    while (sum >= target) {
    //minimise → sum me se decrease karo,
    len bhi update karna i / start ko aagae badha paiga
    length = min (len, j - i + 1);
    sum = sum - nums[i];
    i++;
    }
    j++;
}
if (len == INT_MAX)
    return 0;
else
    return len;
}
```

algo 1) start with 1 size window from the extreme left.
2) increment end till you achieve the target specified in Problem Statement (P/S)
3) Once you achieve the target, try minimising the window

Variable window size

Leetcode: Smallest window in a string containing all the characters of another string [Minimum window substring]

S: time to practice

p = toc

Output: to prac

```
string smallestwindow( string s, string p)
{
    int len1 = s.length();
    int len2 = p.length();
    int start = 0;
    int ansIndex = -1;
    int ansLen = INT_MAX;
```

Reading: summary of sliding window Patterns for subarray / substring


```
if(len1 < len2){
    return "";
}

unordered_map <char, int> strmap;
unordered_map <char, int> patMap;
// to keep track of all characters of p string.
for( int i=0; i< p.length(); i++){
    char ch = p[i];
    patMap[ch]++;
}

int cont = 0;
for( int i=0; i< s.length(); i++){
    char ch = s[i];
    strMap[ch]++;
    //valid character -> jo character tumhare pattern mein hai
    if(strMap[ch] <= patMap[ch]){
        cont++;
    }
    if( cont == len2){
        // window is ready
        // minimise the window -> freq decrement, ans update
        // start ko aagae badha
        while (strMap[s[start]] > patMap[s[start]]
                || patMap[s[start]] == 0){
                if( strMap[s[start]] > patMap[s[start]]
                    strMap[s[start]]--;

                start++;
        }
        // answer update
        int length of window = i - start + 1;
        if( length of window < answer) {
            answer = length of window;
            ansIndex = start;
```

```
    }
    }
    }
    if (ansIndex == -1)
        return "-1";
    else
        return s.substr(ansIndex, ansLen);
}
```

↳ Bonus Bit Manipulation

& | ^ ~ << >> /2

×2

imp

a	b	&	|	^	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

$\ll i$ $\times 2^i$ $\gg i$ $/2^i$

int a = 0

$\underline{\underline{\sim a = 1}}$? $\underline{\underline{(-1)}}$

int a = 0

a = 0000 0000 0000 0000 ... 0000

~a = 1111 1111 1111 1111 ... 1111 (32 times)

-ve

↳ 2's complement

000 0000 0000 0000 ... 0000

\+ 1

0000 0000 0000 0000 0001

− 1

ex i/p : n=5 tell it is even or odd.

3 11
5 101
7 111

& 1 → 0 even
& 1 → 1 odd

% 2 == 0 even computationally
% 2 == 1 - odd heavy operation

↳ Mask - Bit masking (Bit manipulation)

get i^{th} bit = n=10
$i = 3^{rd}$

0000 0000 0000 1010
3 2 1 0 n bit

& 0000 0000 1000

0000 0000 1000

1

0000 0000 ... 0001
<<3

0000 0000 ... 1000

we could also do 3 time right shift to get the 3rd bit.

```
int getithbit(int n, int i) {
    int mask = 1<<i;
    int ans = n & mask;
    if( ans==0)
        return 0;
    else
        return 1;
}
```

ex set i^{th} bit to 1 n = 1010 i = 2

3 2 1 0
1 0 1 0

OR ans
mask 0000 0100
0000 0000 1110

Pata lega set clear
& 1 or 1


```
void setithbit (int &n, int i) {
    int mask = 1 << i;
    int ans = n | mask;
    cout << "After setting" << ans << endl;
}
```

1010
setbit(10, 2)
1110
// 14

n.10
0000 0000 0000 1010
mask & 1111 1111 1111 1101 ← ~ (0000 0000 ... 0010)
0000 0000 0000 1000

Ex
```
void clear ithbit ( int &n, int i) {
    int mask = ~ (1 << i);
    int ans = n & mask;
    cout << " after clearing " << ans << endl;
```

// clearbit (10, 1)
// 8

Ex update ith bit. update (n, i, 0/1) target

imp
// stairs question
// recursion subsequence of a string

→ clear ith bit.
→ mask → target << i
n | mask

```
void updating ithbit (int &n, int i, int target) {
    clearithbit (n, i);
    int mask = target << i;
    n = n | mask;
    cout << after updating << n;
```

(10, 0, 1)
11

imp
Ex Single no. Leetcode easy. [GFG Missing element] xor

i/p: 2 2 1 4 1 2 1 2 4 ^ ~~2~~ ^ ~~2~~ ^ ~~1~~ ^ ~~1~~
o/p 1 4 4

```
int singleNumber(vector<int>& nums) {
    int ans = 0;
    for (int i=0; i<nums.size(); i++) {
        ans = ans ^ nums[i];
    }
    return ans;
}
```

Ex Last 4 bit clear

```
        0000    1111    1111
(mask) & 1111   1111    0000
        0000    1111    0000 ans
```

```
void clearLastiBits(int n, int i) {
    int mask = (-1 << i);
    n = n & mask;
    cout << "after clearing" n;
}
```

1111 1111 1111 << 4
(-1 << i)

clearLastiBit(7, 2); //4
0000 0111
0000 0100

Imp Ex check power of 2

i/p: n = 16
2^4 can it be represented in the form of 2^i Y/N.

2	10	#1 Brutforce	n/2
4	100		n/2
8	1000		n/2 (-1)
16	10000		

```
  16  10000
& 15  01111
      0000
```

```
  8 - 1000
  7 - 0111
      0000 ✓

  4   100
& 3   011
      000
```

n & (n-1) == 0
it is a power of 2.

```
bool checkpowerof2 (int n){
    if((n & (n-1)) == 0)
        return true;
    else
        return false;
}
```

imp

Count setbits

8 0000 0000 ... 0000 1000

no. of set bits = 1

7 0000 0000 ... 0000 0111

no. of set bits = 3

10. 0000 0000 ... 1010

no. of set bits = 2

```
int countsetbits (int n){
    int count = 0;
    while (n != 0){
        int lastBit = n & 1;
        if(lastBit)
            count++;
        //right shift
        n = n >> 1;
    }
    return count;
}
```

#1. right shift one by one and count++.

TC = log(n) } binary representation manipulation.

#2 faster

0 000 1 010 1100

imp

```
int countsetBitsfast (int n){
    int count = 0;
    while (n != 0){ //remove last set bit
        n = (n & (n-1));
        count++;
    }
    return count;
}
```

1010 1110
and 1001 1111
1000 count=1
0111
0 count=2

H.W.

Q. Fast exponential (done during recursion) [Using bit manipulation]

n - even / odd

↳ even $a^{n/2}$

↳ odd $a^{n/2}$

ans - a

i j ← clear →

Ex i.p. 111 0 1 0 1 1 1 0 111

o/p 111 0000000 0 111 clear bits in a range

111 1 0000 0000 111

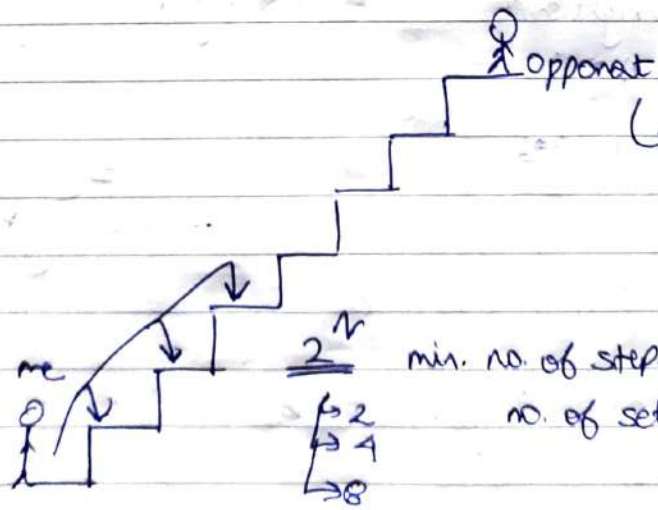
$-1 << i+1$ a) 1111 0000 0000 000

0000 001000-1 b) 0000 0000 0000 111 OR

mask: 1111 0000 0000 111

```
void clearBitsinRange(int n, int i, int j){
    int a = (-1 << (i+1));
    int b = (1<<j) << -1;
    int mask = a|b;
    cout << "After clearing in range" << n << endl;
}
```

Ex



↳ no. of set bit hogi vahi answer hoga

min. no. of steps to reach opponent.

no. of set bit.

Rather than using recursion we can use bit manipulation.

```
while (n!=0){
    n = (n-1);   last set bit ko remove karti hai
    count++;
}
```

2^n 1 set bit.

1 jump → 1 set bit.

no. of jumps = no. of set bits in no.

Ex Subsequence using bit masking

str = "abc"

	Powerset or subsequence
a b c	a
a b c	b
a b c	c
a b c	ab
a b c	ac
a b c	bc
a b c	abc
abc	""

"" a b c ab bc ca abc

2^n

#1: Recursion inclusion/exclusion pattern

abc → length = n Power set = 2^n.

$n = 2$

$2^n = 2^3 = 8$

$0 \to 2^n - 1 = 2^3 - 1 = 7$

$0 \to 7$

0	000	abc	""
1	001	abc	c
2	010	abc	b
3	011	abc	bc
4	100	abc	a
5	101	abc	ac
6	110	abc	ab
7	111	abc	abc

$1 << n = 2^n$.


```
void getSubsequences (string str) {
    vector <string> ans;
    int n = str.length();

    for (int num = 0; num < (1<<n) ; num++){
        string temp = "";
        // we will create subsequence string in this temp string
        for (int i=0; i<n; i++){
            char ch = str[i];
            if ( num & (1 << i) {
                // non zero value kae liye, include current character
                temp.push_back(ch);
            }
        }
        if (temp.length()>=0){
            // store in ans
            ans.push_back(temp);
        }
    }
```

$\boxed{TC = O(n^2)}$ not that good but OK.

Homework questions

1. Find duplicate number

#1
```
sort( arr.begin(), arr.end);

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

#2
```
int ans =-1
for( int i=0; i< arr.size(); i++){
    int index= abs( nums[i]);

    if( nums[index] <0){
        ans = index;
        break;
    }

    //visited mark
    nums[index]*=-1;
}
return ans;
```

#3
```
Positioning method
while (nums[0] != nums[nums[0]]){
    swap (nums[0], nums[nums[0]]);
}
return nums[0];
```

Q1 Missing element from an array with duplicates

```
void findMissing (int *a, int n){
    // visited hash
    for(int i=0; i<n; i++){
        int index = abs (a[i]);
        if( a[index] -1] > 0){
            a[index-1] *= -1;
        }
    }

        // all positive indexes are missing

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

method 2 sorting + swapping 1 | 3 | 5 | 3 | 4

```
int i=0;
while (i<n) {
    int index = a[i]-1;
    if( a[i] != a[index]) {
        swap (a[i], a[index]);
    }
    else {
        ++i;
    }
}
for(int i=0; i<n; i++){
    if( a[i] != i+1) {
        cout << i+1 << " ";
    }
}
```

ex Find first repeating element.

1	5	3	4	3	5	6
index 1	2	3	4	5	6	7

o/p: 2 (index) 5.

#1.

int firstelement (

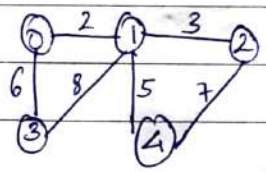
Prims Algo.

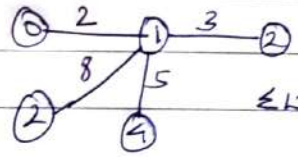
↳ To find mini. spanning tree.

Spanning Tree: ① Convert graph to a tree. (no cycle)

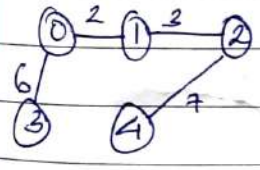
② In tree, every node should be reachable by every other node.

③ Tree would have 'n' nodes & 'n-1' edges

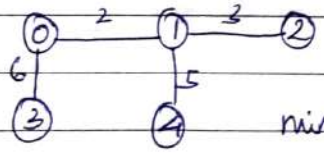
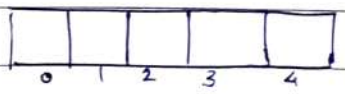
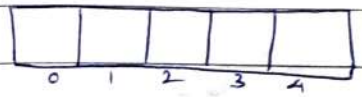
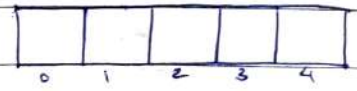




$\Sigma w = 18$



$\Sigma w = 18$



minimum spanning tree