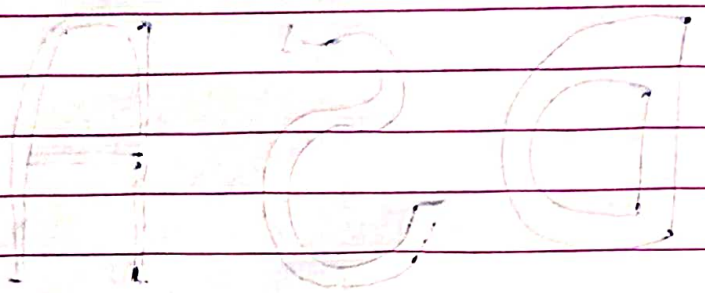
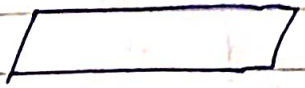
SR. NO.	DATE	TITLE	PAGE NO.	TEACHER'S SIGN



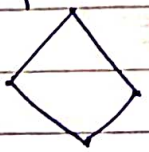
* Flow chart *

→ = arrow

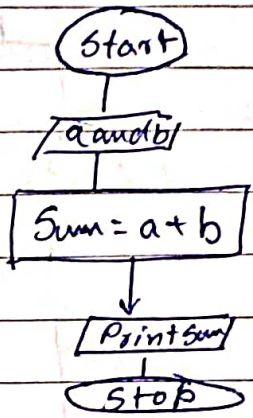
 → Process

 → input / output

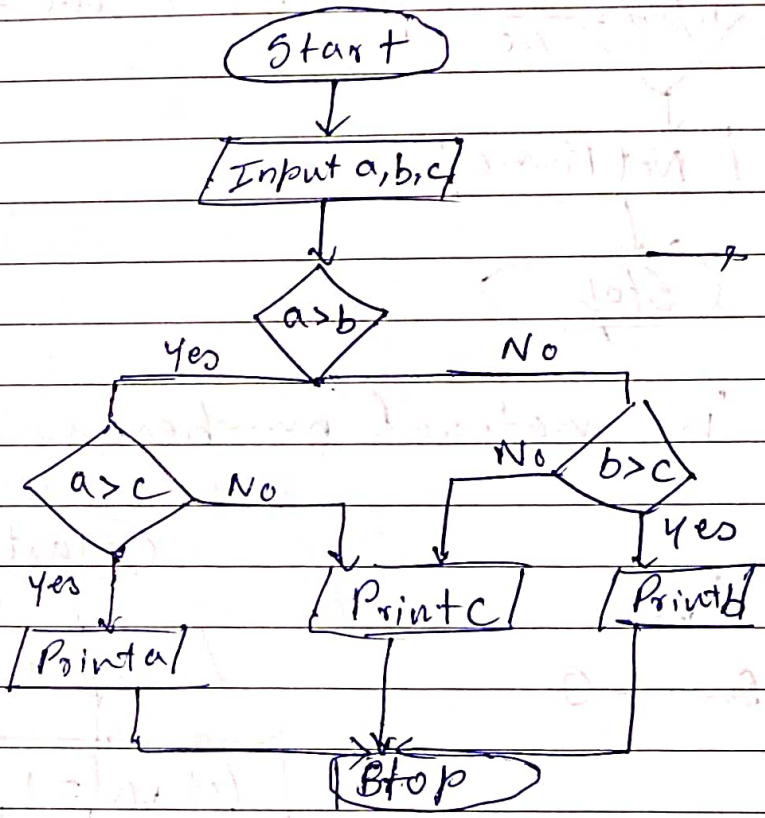
 → Start / Stop

 → Decision

Q Sum of a and b



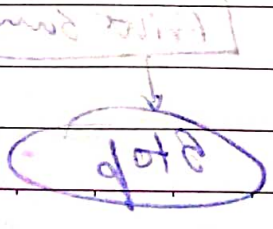
Q find Max of 3 nos

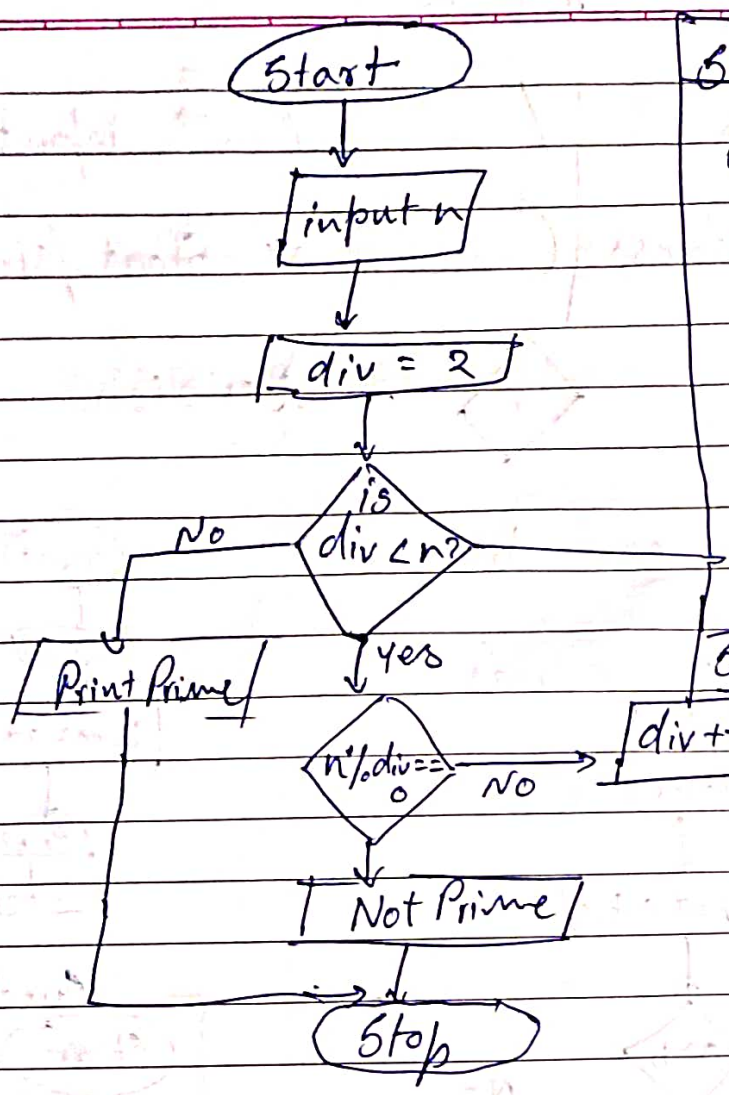


→ Pseudo Code

```
Start
input a, b, c
if a > b do
    if a > c do
        Print a
    else
        Print c
if b > c do
    Print b
else
    print c
Stop
```

Q find if no is prime.



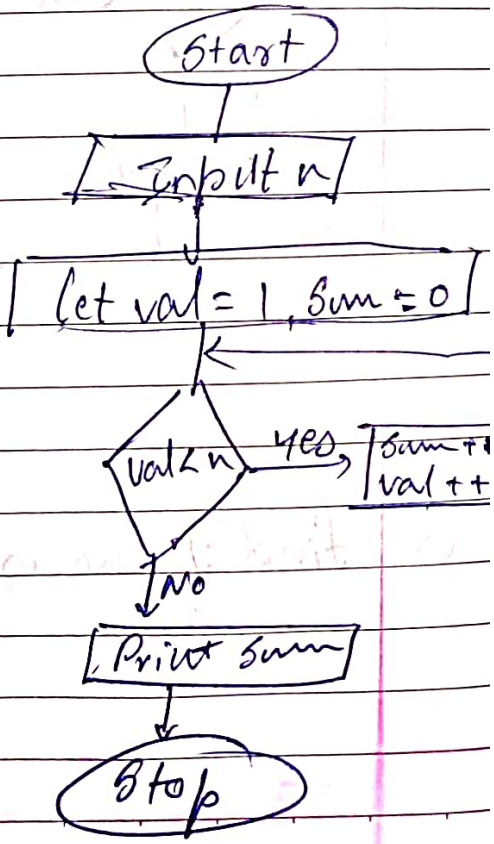


Pseudo code

```
Start
Input n
div = 2
While div < n do
if n % div == 0
  Print "Not Prime"
else exit
  div = div + 1
Print Prime
Stop
```

Q Sum of first 'n' natural number

```
Start
  Input 'n'
  val = 1, Sum = 0
While
  val <= n do
  Sum = Sum + val
  val = val + 1
Print Sum
Stop
```



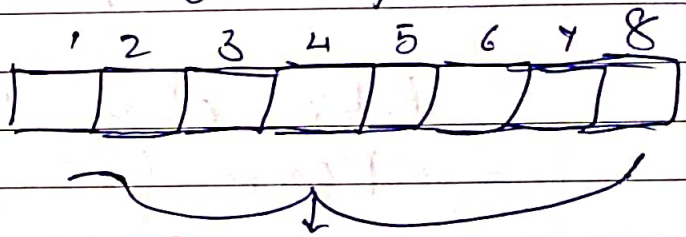
// Java //

* Class name of 1st is same as file name
* Program is first run in "PSVM"

* Public static void main (String args[]) {
  // code is run here System.out.Print "Hello Wor
  }

Premitive → Donot Share value with other Primitive value (Int, float, double, byt
Non Premitive → Created by programmer and d o defined by java (String, class, Interface, object

(Bige)
(-128 to 127) 1 byte → 8 bits
Int → 4 byte
boolean → 1 byte
double → 8 byte
float → 4 byte
long → 8 byte
Char → 2 byte



Q Sum a and b

```
Public class Sumadd {
    Ps vm (String args[]) {
        int a = 10;
        int b = 20;
        Sum = a+b;
    }       S.out.Print (Sum);
}
```

M T W T F S S
Page No.: YOUVA
Date:

{ for Input }

```
import java.util.*;
Scanner {name} = new Scanner (System
String input= name.next()
Sout.(input);
```

next() → String
nextline →
nextInt → Integer

For Error → StackOverflow

Q Product a × b

```
Public class Product {
  PSVM {
    Scanner Productof2no = new Scanner(System
    int a = Productof2no.nextInt();
    int b = "  "  .nextInt();
    int Sum = a + b;
    Sout(Sum);
  }
}
```

Q Area of Circle (πr^2) → π → constant

```
Public class AreaofC {
  PSVM {
    Scanner radius = new Scanner(System.in);
    float rad = radius.nextInt();
    float Pie = 3.14;
    float Area = Pie * rad * rad
    Sout(Area);
  }
}
```

* Java forcing *

M T W T F S S
Page No.:
Date:
YOUVA

* Type Conversion * (Source > destination)

Byte → Short → Int → float → long → double

* Type Casting * Narrowing * Explicit *

eg:- int marks = (int) (99.99f)

(float)

* Type Promotion

1) If Byte, Short char → Convert to int
2) If long, float, double → Convert to long

- JDK → Java Developement Kit.
- JRE → Java Rimtime Enviroment.

iii) file.java → Compiler → Bytecode → JVM
(0,1)
Native Code

* Operator :- {A = A+10 ≅ A+=10}

1] Arithematic Operator (Binary → +, -, *, /, %
Unary → ++, --, ...)

2] logical operator (|| , && , !)
3] Relational operator (==, !=, >, <, >=, <=)
4] Bitwise operator (&, |, ^, ~, <<, >>)
and, or, xor, com, leftshift, rightshift
5] Assignment (=, +=, -=, *=, /=)
A=A+

Pre Increment (Change then use) (++ a) a+1 = a	Post Increment (use then change) (a++) a = a+1
Pre decrement (change then use) (--a) a-1 = a	Post Decrement use then change a-- a = a-1

* And (&&) :-

1st	2nd	Ans
T	T	T
T	F	F
F	T	F
F	F	F

* || (OR) :-

		Ans
T	T	T
T	F	T
F	T	T
F	F	F

* XOR (Ans) *

T	T	F
T	F	T
F	T	T
F	F	F

* ! (Not) :- True → false
false → True

* Condition :- (if - else) (else if)

Q Print if a nos is odd or even

```
import java.util.*;
Public class oddeven {
    PS VM (string args[]) {
        Scanner sc = new Scanner(System.in);
        int number = Sc.nextInt();
        if (number % 2 == 0) {
            Sout (Print) ("Even");
        }
        else {
            Print ("odd");
        }
    }
}
```

* Else-if Example

```
if (    ) {
}
else if (    ) {
}
```

// last is ~~else~~ else

```
else {
```

Q Income tax calculate
i) Income < 5L
0% tax
2) Income 5-10L
20% tax
3] Income > 10L
30% tax

```
Public class taxcalc {
 PS vm{
  int income = input
  int tax(calculate)
 if (income < 5L) {
  Print(tax = 0%);
else if (income ≥ 5L && in < 10
  Print (tax = 20%);
else {
} Print(tax = 30%); }
}
```

Q Print largest of 3 nos?

A = 1, B = 3, C = 6

Start

```
Public class largestno {
  PSVM {
if (A > B) && (A > C) { Print ("A"
else if (B > C) { Print ("B"); }
else { Print ("C"); }
}}
```

* Ternary Operator :-

(True) (False)

Data type Variable = Condition ? Statement : Statement

eg:- i) Boolean large = (5>3) ? 5 : 3;

Dtype var

Print "5";

ii) String type = (5%2==0) ? "even" : "odd"

Print odd;

Q Check if Student will pass or fail

Marks ≥ 40 : Pass

Marks ≤ 39 : fail

Int Marks = 50;

→ ~~Int Marks~~

String reportCard = Marks >= 40 ? "Pass" : "fail"

* Switch Statement :-

```
Switch (variable) {
    case 1 :
        break;
    case 2 :
        break;
    case 3 :
        break;
    default :
}
```

```
int number = 2
Switch (number) {
case 1 : Print ("1");
    break;
case 2 : Print ("2");
    break
case 3 : Print ("3");
    break
default : Print ("0")
}
```

Q Calculator :- (Switch) :

```
import java.util.*;
Public class Calculator {
    PSVM {
        Scanner sc = new Scanner (System.in);
        int a = sc.nextInt(); int b = sc.nextInt();
    Char operator = sc.next().charAt(0);
Switch (operator) {(i) case '+' : Print (a+b);
                        break;
(ii) case (-) : Print (a-b);  break;
    case (*) : Print (a*b);  break;
    case (/) : Print (a/b); break;
    case (%) : Print (a%b) break;
    default : Print ("Not Simple calculation");
    }
} }
```

* Loops :- 1] While → Never use if condition is true
2] for
3] do While

1] While (condition) {
Print ();
}

// Infinite loop if we dont know the infinite point upto condition

eg:- int counter = 0;
while (counter < 100) {
Print ("Hello");
counter ++;
}

// Increasing counter

Q Print nos from 1 to n

Public class numbers {
Public static void main (string args[]) { Scanner sc = new Scanner (System.in);
int counter = 1; int range = sc.nextInt();
while (counter <= range) {
System.out.Print (counter + " ");
counter ++;
}
Sout ();
}
}

2] for loop :- for (initialize ; condition ; updation)
{ " }

Q Number Print from 1 to 10 ? using for loop

```
import java.util.*;
Public class Numbers {
  psvm {
    for (int i=1; i<=10; i=i+1)
    { Print("Hello"); }
```

Q Print Square Pattern using (for)

```
// * * * *   // line 1
* * * *      // line 2
* * * *      // line 3
* * * *      // line 4
for (int line=1; line<4; line++) {
    Print(" * * * * ") }
```

Q Print same ↑ using while

```
int line = 1
while (line<=4) { Print(" * * * ");
} line++;
```

(P.T.O →)

{

* To find last digit of any No divide it by 10
eg 101 = 101 % 10 = Ans = 10 Remainder

Q Sum of 'n' natural numbers using While

```
Public class SumofN {
PSVM {   Scanner sc = new Scanner (System.in);
         int n = sc.nextInt();
         int sum = 0
         int i = 1
      while ( i <= n) {
            sum += i;
            i++; }
      Print (" Sum is : " + sum);
   }
}
```

3] Do While :- do { // Something }
While (cond") ;

Q Print Reverse of number? number = 10899

```
int a = 10899;
while (n > 0) {
   int lastdigit = n % 10;
   Print (" lastdigit + " ");
   n = n / 10; }
```

{ Object name = new object ( ) } (compile time) = (runtime)

* Continue Statement : To skip the current statement.

Q Display all nos entered by user except multiple of 10 ?

```
Public class Basic{
  PSVM{
      Scanner Sc = new Scanner (System.in);
do{
    Print ("Enter your nos : ");
    int n = Sc.nextInt();
   if (n % 10 == 0){ continue;}
     Print (n);
   } while (true);
    }
}
```

Q Star Pattern : $O(n^2)$ acending ?

```
for (int line = 1; line <= 4; line++) {{
   for (int star = 1; star <= line; star++ {
      Print (" * ");
      }
      Print ();  }
}
```

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

Q Star decending

```
for (int i=1 ; i<=4 ; i++){
    for (int star = 1; 5<=(n-i+i) ; 6++) {
        Print ("* ");
    }
}
```

Q Half Pyramid of Nos

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

```
int n = 4;
for(int line=1 ; line<=n ; line++) {
    for (int number = 1 ; number <= line; nn++)}
        Print (number);
}
Print ();
```

Q Character Pattern:

```
int n = 4
Char ch = 'A';
for (int line=1 ; line <= n ; line++) {
    for (int chars = 1 ; chars <= line ; chars++){
        Print (ch);
        ch++ ;
}
Print ();
```

* Function * (logic Reusing and call)

function Syntax : returnType name () {
// body
return Statement ;
}

void → returnType is empty

Public Static void main (String args[]) {
Public → type of class
Static → May have object or may not have object
void → return type
main → Naam dictionary
String args[] → Input
}

* Calling a function :-

Public class Java Basic {
Public Static void PrintSomething () { (function)
System.out.println("Hello"); }
Public Static void main (Strings args[]) {
PrintSomething ();
}

* Method :- The functions written in certain class is the method of that class

Multipe function can be written in Single class

* Function With Parameter :-

return Type name (type: value1, type: va
(void, int, String) // body

return statement ; }
(eg void, int)

Q Calculate Sum with function calling and Parameter

* Without Parameter :-

```
Public Static void CalSum () {
    int a = 10 ;
    int b = 20;
    int sum = a + b ;
    Print (sum) ;
Public Static Void Main (String args[]) {
        calSum ( ) ; }
```

* With Parameter

```
Public Static void CalSum (int a, int b) {
    int sum = a + b ;   Print (sum) ; }
PSVM { int a = 10 ; int b = 20 ;
    calSum ( ) ; }
```

* Java calls func" By Call By Value

* Stack →  } one after another
eg stack of book (bundle of book)

e2) {

Q factorial of number n = 4

function:- Public Static int factorial (int n) {

```
int f = 1
for (int i=1 ; i<=n ; i++) {
    f = f * i ; }
return f }
```

P.S V M { Print (factorial (4)) ;

Q Binomial Coefficient function

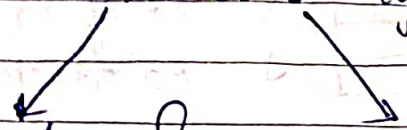
$$ {}^{n}C_{r} = \frac{n!}{r!\,(n-r)!} $$

eg:- ~~int~~ Public Static int Binomial (int n , int r) {

```
int fac_n = factorial (n) ;
int fact_r = factorial (r) ;
int fact_n-r = factorial (n-r) ;

int Binomial = fact_n / (fact_r * fact_n-r
return Binomial.
```

PSVM { Print (Binomial (5 , 2)) ; {

Methods Type



- Math i.e
  - math.Pow
  - math.Min
  - math.max
  - math.sqrt

* Function Overloading :- (Just for using)
* Multiple function with same name but different parameter.
* Function overloading only depends on parameters and not on return Type.

Q Optimized function for prime number

```
Public Static boolean isPrime(int n){
    if(n==2){ return true;}
    for(int i=2; i<=Math.sqrt(n); i++){
        if(n%i==0{ return false;}
}
PSVM{ Print(isPrime(n));}
```

Q Convert from Binary to Decimal or D to B

4 2 1 = (5)$_{10}$

Q n = 101 or 1000 = (8)$_{10}$
8 4 2 1

→ Now n = 10100011

```
Public Static void BintoDic (int Binum) {
  int Pow = 0;
  int dec = 0;
  int mynum = binNum
  while (binNum > 0) {
     int lastdigit = binNum % 10;
  decNum = decNum + (lastdigit * Math.Pow
                                  (2, Pow)
     Pow ++
     binNum = binNum / 10; }
     Print ("decimal of:" + binNum + "=" +
                  decNum);
PSVM { bintoDec (mynum);
```

Q Decimal to Binary?

```
Public Static void dectoBin (int n) {
  int Pow = 0; int mynum = n;
  int BinNum = 0;
  while (n > 0) { int rem = n % 2;
     binNum = binNum + (rem
     * (int) Math.Pow (10, Pow));
     Pow ++; }
     n = n / 2; } Print ("Bino:" + mynum + "=" + Bi
                                          Num
PSVM { Print (dectoBin (12)) };
```

// 11:00 //

* Method Scope :- Variable defined in fun.n and method can be used in same fun.n or method

* Block Scope :- only use variable in block
{ } → This is block

---

* Array * {index from 0}

datatype arrayname[] = new datatype[s

eg i) int marks[] = new int[50];
ii) int num[] = {1, 2, 3, 4}
iii) int no[] = {4, 5, 6}
iv) String fruits[] = {"mango", "apple"}

* Linear Search * [O(n)]

Q Largest number;

Public Static class Arraycc {

Public Static int getlargest no (int num[]) {

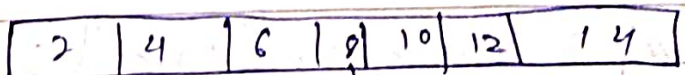
int largest = Integer.minvalue;
for (int i=0; i< num.length; i++) {
if (largest < number[i]) {
largest = number[i]; } }
return largest }

P.S V M {int num[] = {1, 2, 3, 4, 5, 6}

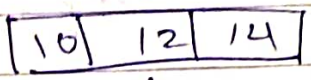
Sout (largest)

$-\infty$ → Integer.Min Value.
$+\infty$ → Integer.Max Value

[O(logn)] * Binary Search * (only for Sorted array)



Start = 0 mid end = n-1 for key = 10



Start mid end = n-1 {min < and mid >}

ge] *

Psuedo code :-
start = 0 end = n-1
While (start <= end)
find mid // (start + end)/2 // formula
compare mid and key

mid == key // found
mid < key // start = mid+1
mid > key // end = mid-1

Q Binary Search

```
Public class Arraycc {
 Public Static int BS (int numbers[], int key) {
int start = 0; end = numbers.length-1;
While (start <= end) { int mid = (start+end)/2;
if (num[mid] == key) { return mid}
if (num[mid < key] { start = mid+1; } else
  end = mid-1;
return -1;
PSVM { int numbers[] = {2, 4, 6, 8, 10, 12}
int key = 10;
  Print (index for key of Binary Search)
```

Q Reverse an array? [2, 4, 6, 8, 10]

* Swap :- temp = number[end];
number[end] = number[start];
number[start] = temp;

Q ~~Pairs in Array~~ ↓

```
PSVM reverse (int numbers[]) {
    int first = 0 last = numbers.length-1;
    while (first < last) {
        int temp = numbers[last];
        numbe[last] = numbers[first];
        num [first] = temp;
        first++;
        last--;
PSVM (String args[]) {
    int numbers[] = {2,4,6,8,10};
    reverse (numbers);
    for (int i=0; i<numbers.length; i++){
        S.out (numbers[i]
```

Q Pairs in Array :- 2, 4, 6, 8, 10

(2,4,) (2,6) (2,8) (2,10)
(4,6) (4,8) (4,10)
(6,8) (6,10)

(nested loop)

```
Publis Static void PrintPairs(int num[]) {
    for (int i=0; i<num.len; i++) {
        int curr = num[i]
        for(int j=i+1; j<num.length; j++){
            S.out ( current + number[j]
```

Q Print Sub Array 2, 4, 6, 8, 10

2 , 24 , 246 , 2468 , 246810

4 , 46 , 468 , 46810 ,
6 , 68 , 6810
8 , 810
10

```
P S V ArraySub {int numb[]) {
    for (int i=0; i<numb.len; i++) {
        int Start = number[i]
        for (int j=i+1; j<numb.len; j++) {
            int end=j;
        for (int K = Start ; K <= end; K++)
```

* Max Sub Array Sum

M T W T F S S
Page No.:
Date:
YOUVA

[Prefix Array] $(O(n^2))$

{ Prefix Array [end] - Prefix Array [start-1] }

```
P S V max Subarray Sum (int num[]) {
    int curr = 0;
    int MaxSum = Integer.MinValue;
    int Prefix[] = new int [numbers.length];

    Prefix[0] = number[0];

    for(int i = 1 ; i < Prefix.length; i++){
        Prefix[i] = Prefix[i-1] + numbe[i];
    }

    for (int i=0; i<num.leng; i++) {
        int Start = i;
        for (int j=1 j<number.length; j++) {
            int end=j;

        CurrSum = Start==0? Prefix[end]:
            Prefix[end]- Prefix[start-1];
        if (maxSum < currSum) {
            maxSum = currSum;
                }
            }
        }
```

* Kadance Max Subarray Sum

* O(n) *

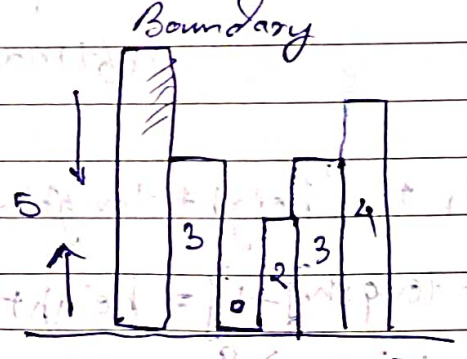
Current Sum : Maximum Sum

-2, -3, 4, -1, -2, 1, 5, -3

○	CS	0	0	4	3	1	2	7
→ ●	MS	0	0	4	4	4	4	7

```
PSV Kadanes (int numb[]){
    int ms = Integer.MinValue;
    int cs = 0;

    for(int i=0; i<num.length; i++){
        cs = cs + num[i];
        if (cs<0){
            cs=0;}
        ms = Math.max(cs, ms); }
    S.out.Print(+ms)
```

2 * Trapping Rain Water (Acid) Jumping * [O(n)]



Will only trap if neighbour have boundary

} height (Water level - bar level height)

* Width = Trap Water

* Auxiliary Array (Helper Array) *

Water level formula = Select

WL = (Max left Boundary, Max Right Boundary) ⇒ Min value

Trapped water = (maxL, maxR) - height

1] Case :- Single bar { no water trap }

2] Case :- two Bar { no water trap }

3] Case :- Ascending and Descending bar doesnot store water

-: Code :- 4, 2, 0, 6, 3, 2, 5

*

```
Public static int trappedRainWater(int heigh
// calculate left and Right max Boundary
    int leftMax[] = new int[height.length
    leftmax[0] = height[0];
    for(int i=1; i<height.length; i++) {
        leftMax[i] = Math.Max(heigth[i],
                              leftmax[i-
// Right Max
    int rightMax = new int[height.length]
    rightMax[heigth.length-1] = height[heigh
    for(int i=n-2; i>=0; i--) {
        RightMax[i] = Math.max(height[i], rightMax
```

```
int trappedWater = 0;
for(int i=0 i<n; i++) {
int waterLevel = Math.min(leftMax[i], rightMax[i]
trappedWater += waterLevel - height[i];
}
return trappedWater;

PSVM { int height[] = {4,2,0,6,3,2,5};
S.out.print(trappedRainWater(height));
} }
```

Q Buy and Sell Stocks to Profit :-

```
P S int BySell(int Prices[]) {
int ByPrice = Integer.Max_value
int MaxProfit = 0;
for (int i=0 i < Price.length; i++) {
if (ByPrice < Price[i]) {
// Todays Profit // → int Profit = Price[i] - ByPrice;
maxProfit = Math.max(maxProfit, Profi
} else { ByPrice = Price[i]
} return MaxProfit;
}
```

```
P S V M {
    int Prices[] = {7, 1, 5, 3, 6, 4}
    S.out { buy.sellStocks(Prices)) ?;
    }
```

* OOPs *

* Classes and Objects *

↓ Group of real world entity

↓ Real World entity (Instance of class)

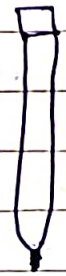
eg:- A classroom with same students having same properties (dress, grade books)

Class → classroom
Students → Objects.

Tejas * Class is Blueprint of object *

Car → Class
Kia, Mar 800 → Object.

In java Class → Start Name with Capital letter
functions Start with Small letter



Pen
- color : String (Blue)
- tip : 5 mm (int)
- Setcolor ();
- Set tip ();

* How to write class :

The filename of code written should include a public class in the code

eg :- OOPs . java

```
Public class OOPs { P S V M (String. args[]){
  Pen P1 = new Pen() // created a pen object (P1
P1.setcolor("Blue") or (Constructor) P1.color = "Yellow"
                Object Pen
* // 1st class example //  }}
  Class Pen {  // Properties + funct"
    String color;
    int tip;

    Void setcolor (String newcolor) {
      color = newcolor;
    }
    Void settip (int newtip) {
      tip = newtip;
```

* Objects are Stored in Heap Memory.

* 2nd Example of class

Class Bankaccount {

Password

* Access Modifiers * specifiers *

Access modifiers	within class	Within Package	Outside Pack by subclass	Outsid Pack
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

* ~~Constructor~~ ⇒ ~~fn~~

Private and Protected are not allowed in front of class

```
class Bankaccount {
Public String username;
Private String pwd;
Public void setPwd(String pwd) {
    password = pwd; }
}
PSVM {
    Bankaccount MyAcc = new Bankaccount();
    Myacc.username = "Sowbhagya";
    MyAcc.setpassword = "abcd"; }
```

---

* Getters and Setters *

Get → to retur value
Set → to modify value

this → it is used to refer to current working object

* eg:- for Pen class

```
String getColor() {
    return this.color; }
```

1] Encapsulation
2] ~~Inheritance~~
3] Polymorphism
4] Abstraction

* Pillars of OOPs :-

1] Encapsulation :- Defined as wrapping up data and method under one single un
It also implements data hiding

eg :- Capsule → Data / method in one capsule

* Constructor :- It is special method invoked automatically at time of object creation

* Properties of it :- Pen p1 = new Pen();
class object constructor

- It have same name as class
- It doesnot have return type
- It can be called once at time object creation
- Memory allocation happens when calling it.

* Types of Constructor : eg Student s1 = new Studen

1] Non Parameterized → student() { S.out(" ")

2] Parameterized → Student(String name) { this.name

3] Copy Constructor → C++ have by default "CC"

1st → Copy → 2nd

Transfer to 2nd Property

s1.name = "Hamza";
s1.roll = "55";

eg:- Student S2 = new Student(S1);
// properties of S1 will transfer to S2

S2.password = "x4x"

* Copy Constructor when one is changed the same change will reference to other object also

S1.name = "ram";
S2.name = "ram";

* Shallow and Deep Copy:-

↓ Changes reflect

↓ Changes Dontreflect

* Deep CC :- Example (lazy Copy)

Student S2 = new Student(S1);
S2.pwd = "4x";
S1.marks[2] = 100;
for (int i=0; i<3; i++){
S.out(S2.marks[i]); }

Student (Student S1) {
marks = new int[3]
this.name = S1.name
this.roll = S1.roll
for(int i=0; i<marks.length; i++){this.mark[i] = S1.marks[i]

* Destructor :- Java have in build destructor it cannot be used in Java
↓
Java Garbage Collector

2] * Inheritance :- Properties method of Parent (Base) class passed to derived class (child)

// Single level //

eg:- // Base class (Parent)

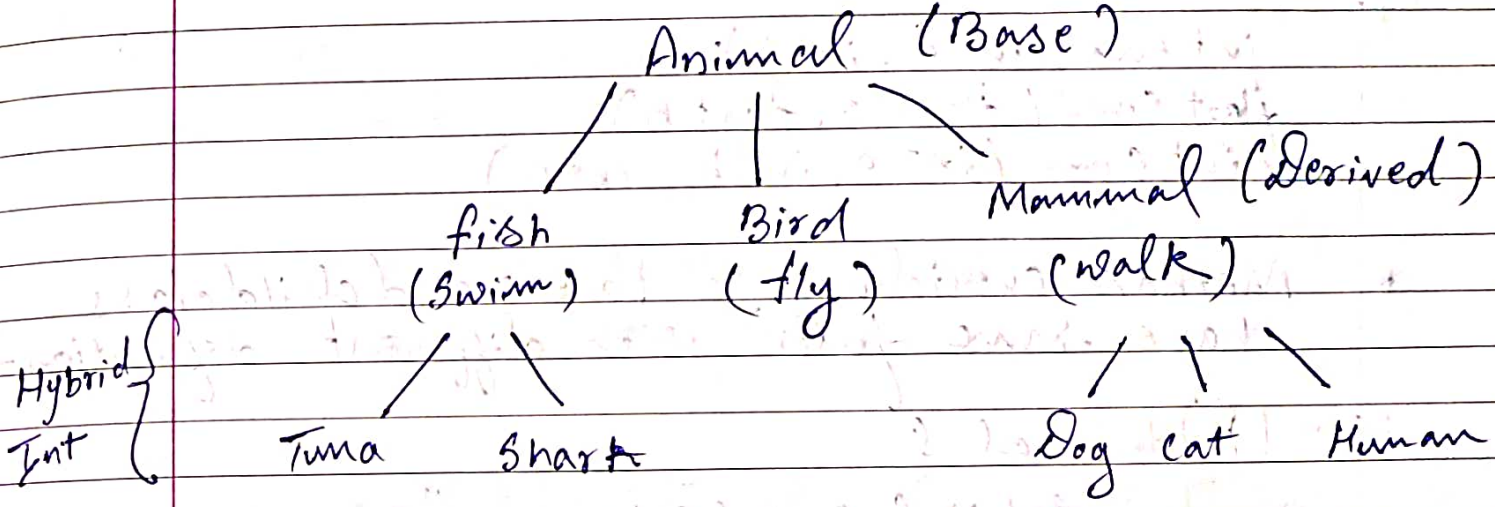
```
Class Animal {
    String Color ;      // Property of class
    void eat() {
        Sout("eats"); }
    void breate() {
        Sout("breathes"); }

// Derived Class (Child)
Class Fish extends Animal {
    int fins
    void swims() { Sout("Swims"); }

PSVM {
    Fish shark = new fish();
    Shark.eat();
```

for(int i=0; i<n; i++) cout

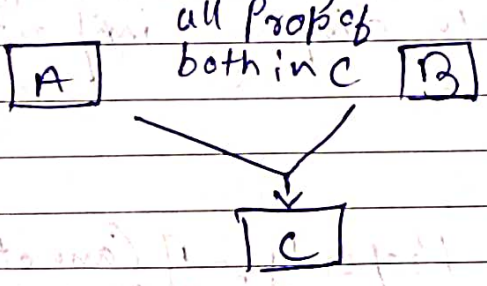
* Types of Inheritance :- Single level inheritance
Multi level, Hierarchical

	Base class	Animal (color) (eat, Breate)
(Multi level Int)	↓	↓ (legs)
	Derived class	Mammal
	↓	↓
	Derived class	Dog, cat, Tiger (breed)

* Hierarchical Inheritance



* C++ have Multiple Inheritance
But Java have Interfaces



3] * Poly Morphism (Many forms):

(Static) Compile Time poly :- Method overloading.

(Dynamic) Run Time poly " Overriding.

* Method overiloading :- Multiple functions with diff" parameters

eg:-

```
int sum (int a, int b)
float sum (float a, float b)
int sum (int a, int b, int c)
```

* Method overiding :- Parent and child class have same funcn with different defination.

eg:- (P) Animal {
eat () { sout ("eat anything") } }

(C) Deer { eat() { sout ("eat grass") } }

4] Abstraction * Hides unecessary details and only display important parts

/ \

1] Abstract classes 2] Interfaces

1] Abstract classes :-

Property :- 1] Cannot create instance of abstract class
2] Can have abstract / Non abstract method
3] Can have Constructor

Class abstract	function abstract
abstract class A { }	abstract void func() { }

eg:- abstract class Animal {

void eat() { Sout("animal eats") }}

abstract void walk() { } // It will give idea to different animals different for each.

class Horse extends Animal {

void walk() { Sout("walks 4 legs")}}

class Chicken extends Animal {

void walk() { Sout("walks 2 legs") }}

2] Interfaces :- It is blueprint of class *

eg:- Car [wheels, speed, engine] (interface)

Maruti 800 [class)

/ | \

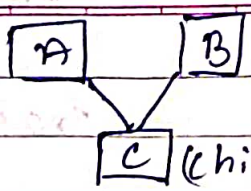
C_1 C_2 C_3 → {objects}

* Multiple Inheritance :-

In C++ it can be implemented by class but in java we use interface.

Interface | Class

Implement | extends



* Why? Interfaces ?

- for using multiple inheritance
- Total abstraction (100%)

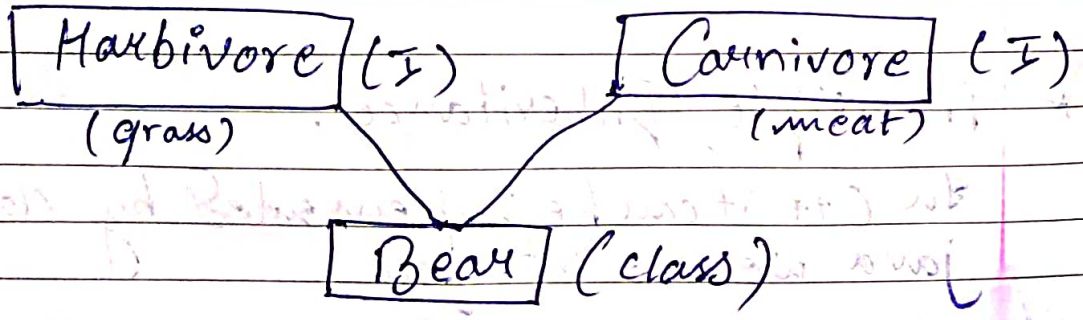
* Interface property :-

1] All method are public abstract and without implementation
2] Used to achieve total abstraction.
3] Variable in interface are final, public and static

eg:-

```
Interface chessplayer{
    void moves();
}
Class Queen implements Chessplayer{
    public void moves(){
        Sout("Left, Right, up, down diagnol");}}
```

eg:-



* Static Keyword:- * It will share same variable and method of given class

Q What can be Static?
→ Properties, function, Blocks, Nested class.

* Super Keyword:- It fast refer to Immediate Parent class object
Used to access parent property, function, constructor.

* Questions *

* Time and Space Complexity *

eg:- $an^2 + bn + c$ (Big O) → upper bound (worst case)
$n^2 + n + c$ → constant (Big-Ω) → lower bound (Best case)

Time Comp = $O(n^2)$

* Property of Time Complexity *

1] In any problem always look for worst case
2] Think for big data.
3] Ignore constants
4] Take Most Tc and ignore Smaller Tc. time

* Space Complexity:- Memory / Space
  - Stack (functions)
  - heap (objects)

input space + auxilary space (temp.)

* Loops → loops run in O(n)
  2 loop nested run in $O(n^2)$

* optimized loop:-

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

* Sorting * * Merge Sort p

1) Bubble Sort :- largest element swap to end array and so on.

eg:- n = 5

5	4	1	3	2

4 5 1 3 2
4 1 5 3 2
4 1 3 5 2
4 1 3 2 5

} for swapping 5 to the end.

Now for sorting '4' to the end of array

eg:-
```
Public Static void bubbleSort(int arr[]){
    for(int i=0; i<arr.length-1; i++){
    for(j=0; j<arr.length-1-i); j++){
        if(arr[j] > arr[j+1]){
// Swap //      int temp = arr[j]
                arr[j] = arr[j+1]
                arr[j+1] = temp
                }}}}
PSVM { int arr[] = {5,4,1,3,2};
       bubblesort(arr);
     }
```

* * Time Complexity :- $O(n^2)$

2] Selection Sort :- Pick smallest from unsorte put at begining

n = 5

5	4	1	3	2

1, 5, 4, 3, 2

1, 2, 5, 4, 3

1, 2, 3, 5, 4

1	2	3	4	5

Time Complexity = $O(n^2)$

```
P S Selection Sort(int arr[]) {
 for(int i=0, i<arr.length-1; i++
    int minpos = i;
    for(int j = j+1; j<arr.length
    if(arr[minpos] > arr[j]) j++
        minpos[j]; } }
// Swap int temp = arr[minPos
    arr[minPos] = arr[i]
    arr[i] = temp;
```

3] * Insertion Sort * Pick from unsorted and place in path.

eg:- P S- insertionsort (int arr[]) {

```
for (int i=0, i<arr.length-1, i++) {
    int curr = arr[i]
    int prev = i-1
// find best position correctly
    while (prev >= 0 and arr[prev] > curr)
        arr[prev+1] = arr[prev];
        prev--; }
// Now inserting
    arr[prev+1] = curr;
```

* Time Complexity:- $O(n^2)$

4] * Counting Sort:- Counting frequency of elements in array

eg:- Pub Stat void CountingSort(int arr[]) {

```
int largest = Integer.Min_value;
for(int i=0; i<arr.length; i++) {
    largest = Math.max(largest, arr[i]) }
int count[] = new int[largest+1];
for(int i=0 .i<arr.length; i++) {
    count[arr[i]]++
```

}

[i]

// Sorting:

```
int j=0
for (int i=0 i<count.length; i++) {
 while (count[i] > 0){
    arr[j] = i
    j++
    count[i]--; } }
```

* 2D Array *

Q Spiral Matrix (Border covering)

Row \ Col	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

Top, Right, Down, Left

	1st iteration	2nd	updation
StartRow	0	1	++
endrow	3	2	--
StartCol	0	1	++
endcol	3	2	--

Eg:- PS

```
print spiral (int matrix[][]) {
    int StartRow, ~~Start Col~~ = 0;
    int Start Col = 0;
    int endRow = matrix.length - 1;
    int End col = matrix[0].length - 1;

while (Startrow <= endrow and Start col <= E
{ //top
  for (int j = Start col; j <= end col; j++) {
      S.out (matrix[Startrow][j] + " ")
```

```
// righ part
for (int i = Startrow+1; i<=endrow; i++)
    S.out (matrix[i][endcol] + " ")
// bottom
for (int j = endcol-1; j>=Startcol; j--){
    S.out (matrix[endRow][j] +" ")}
// left
for (int i= endrow-1; i>=Startrow+1;
    S.out (matrix[i][Startcol] +" ")}
Start row ++;
Start col ++;
Endcol--;
end Row--;
}
```

This will move to next step of Spiral matrix

Q Diagnol Sum :- Q Row = Column

```
PS int DiagnolSum (int matrix[][]){
  int Sum=0;
for (int i=0; i<matrix.length; i++){
  for (int j=0; j<matrix[0].length; j++)
    if (i==j){
      Sum += matrix[i][j]; }
```

```
else if (i+j == matrix.length-1){
        Sum += matrix[i][j]; } }
return Sum;
```

Q ~~or for~~ Search in Sorted Matrix :-

1) Q Brute force method. $O(n^2)$

2) Row wise Approach :- { i = 0 to n-1, j = m-1 to 0 }

3) Staircase Search.

m-1, 0	0, m-1
Key < cell value Top	Key < cell value Left
Key > cell value Right	Key > cell value Bottom.

eg:

```
PS boolean StaircaseSearch(int matrix[][] int key) {
int row = 0; col = Matrix[0].length-1;
while (row < matrix.length and col >= 0) {
    if (matrix[row][col] == key) {
        S.out ("row + col");
        return true;
}
```

```
elf. (key < matrix[row][col]) {
        col--
    else { row++ }
```

---

* *** Strings ***

Strings are immutable

```
String str = "abcd";
String str = new String ("xyx");
```

Q Check if Palindrome:-

RACECAR

```
PS  boolean isPalindrome (String str) {
    for (i = 0 ; i < str.length()/2 ; i++) {
        int n = str.length();
        if (str.charAT(i) == str.charat(n-1-i)
            return false; } { return true
```

* **String Builders:-**

```
StringBuilder sb = new StringBuilder("");
    for (char ch='a' ; ch<='z' ; ch++) {
```

sb.append(ch)

* Change 1st letter to upper case.

```
String Builder sb = new "" ("");
 Char ch = Character.toUppercase(str.charAt(0))
   sb.append(ch);
  for (int i=1; i<str.length(); i++) {
    if (str.charAt(i) == ' ' and i<str.length()-1
     sb.append(str.charAT(i));
      i++
      sb.append(Character.toUppercase(str.charAt(i
  else { sb.append(str.charAT(i));
 return sb.toString();
```

* String Compression aaabbb a3b3

---

* Bit Manipulation *

1) &, |, ^, ~, << (left shift), >> (Right shift)

PTO

* Recursion *

* Used in

Trees, Graph, DP

When one function is created and the another function is created and then both are combined to give another function.

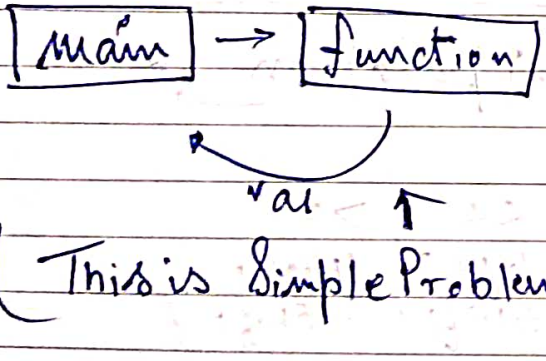
* Smaller problem gives main answer (Recursive)
* Solve Smaller problem to get big answers. of Same instance.

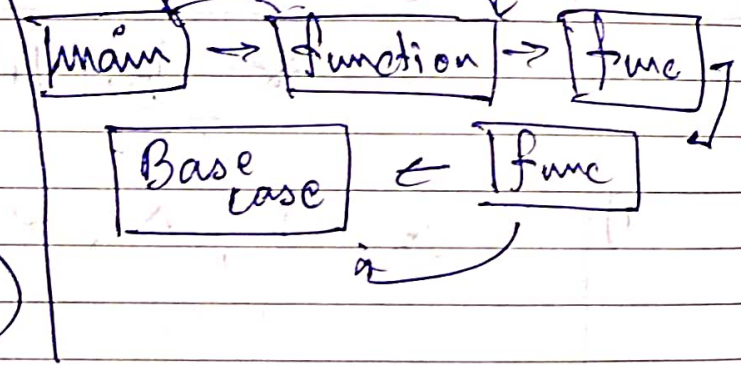
* In Recursion we should always know the base Case

eg:- In 5! = ? but 0! = 1 → Base case

Step1* Top to down going to base case

Step2* Base case to main Case (answer) down to t





* How to use Recursion.

DSA (2)

1] Define Base Case
2] To find the Sol^n of current function (Kaam)
3] Call to inner function.

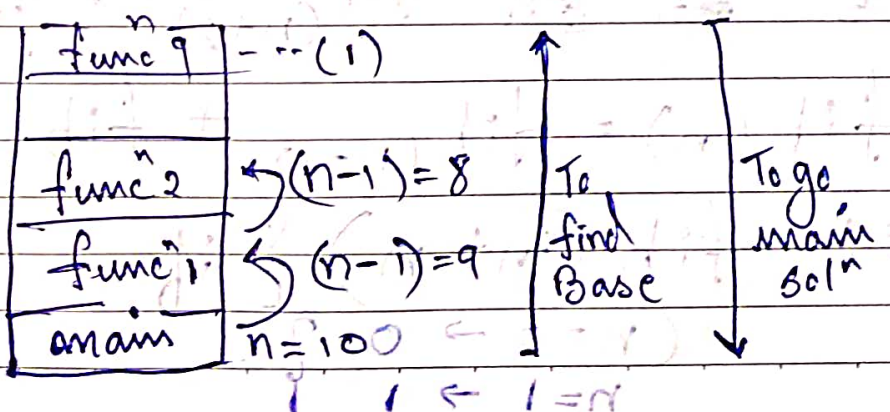
* Example *

Q Print number from 'n' to 1 (Decreasing order) using Recursion.
n = 10
Sol^n = 10, 9, 8, 7, 6, 5, 4, 3, 2, (1) → Base case

→ Given :-

```
P S void PrintDecreasing(int n) {
      S.out(n + " ")
    PrintDecreasing(n-1);
      if (n==1) {
        S.out(n) }
```

* Call Stack :-



* Stack overflow :- It happens if no base case is defined

Q find factorial of number n using Recursion?

→ $n! = n \times (n-1)! \rightarrow f(n) = n \times f(n-1)$

```
PS int fact (int n) {
   if (n == 0) {
        return 1; }
   int fnm1 = fact (n-1);
   int fn = n * fact (n-1);
   return fn;
```

Q Sum of N natural nos

$f(n) = n + f(n-1)$

Q Print nth fibonacci Number | n = 5

1, 2, 3, 5, 8, 13, 21

code :- $fib_{(n+2)} = fib_{(n)} + fib_{(n+1)}$

OR $fib(n) = fib_{(n-1)} + fib_{(n-2)}$

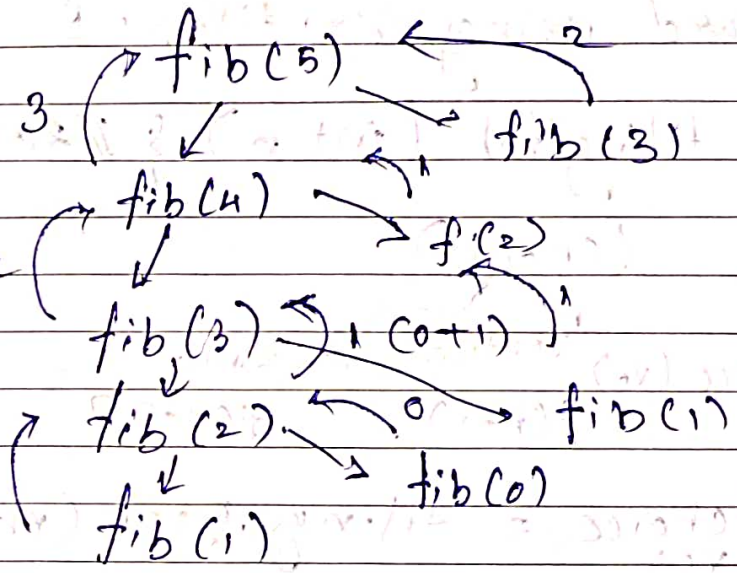
Base Case :- fib (int n) { if (n == 0 → 0; n == 1 → 1 }

eg:-

```
PS   int fib (int n) {
       int fnm1 = fib (n-1);
       int fnm2 = fib (n-2);
       int fn = fnm1 + fnm2;
       return fn;
       if (n==0 || n==1) { return n; }
```

$O(2^n)$

* Tree *



3+2 = 5

fib(5)

fib=3
fib=4
func=5
main=5

Q Check if array is Sorted using recursion

1) $Sorted_N = a[0] < a[i] + Sorted_{N-1}$

```
PS  is Sorted (int arr[], int i) {
      if (arr[i] > arr[i+1]) {
          return false;
```

// base case

if (i == arr.length - 1) {
return true;

Q First/last occurance of element in Array.

Q Tiling Problem :- Given :- i) 2 x n board
(l) (b)
ii) tile Size '2x1' Count no of ways tile can given board using 2x1 tiles.

Eg:- P S. int tilingProb (int n) { // 2xn

// Kaam (choice)
Vertical Horizontal
// vertical choice (Vc) f(n-1) f(n-2)

int verticalchoice = tiling prob (n-1);
(Vc)
// Horizontal choice (Hc)

int Hc = tiling prob (n-2);

int total ways = Vc + Hc;
return totalways

// base case
if (n == 0 || n == 1) {
return 1; }

Q Friend Pairing Problem:-

Given 'n' friend each one can remain single or can be paired up with some other friend. Each can be paired only once. Find out total number of ways in which friend can remain in pair or can be friends single.

→ Base :- n=1 ways = 1
n=2 ways = 2

$$f(n) = f(n-1) + (n-1) * f(n-2)$$

Choice

Single	Pair
$f(n-1)$	$(n-1) * f(n-2)$

* Code:-

```
// base
if (n == 1 || n == 2) {
    return n; }
// Choice Single
int fnm1 = friendPair(n-1);
// Pair Choice
int fnm2 = friendPair(n-2);
int PairWays = (n-1) * fnm2
int totalWays = fnm1 + PairWays
return totalWays;
```

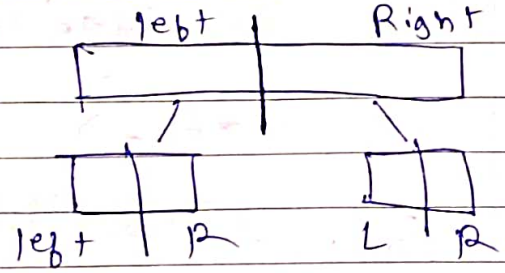
2 Binary String with no Consecutive one.

→

```
PS void BinStr(int n, int lastPlace, StringBuilder str){
  // Kaam
  if(lastplace==0){ str.append("0"); BinStr(n-1,0,
    BinStr(n-1,1,str.append("1");
```

* Divide and Conquer *

* Merge Sort :-



* Approach :-

1) Divide (mid)
2) merge Sort (left) (right)
3) Merge

* 1) Base case :- (si) Start index > ending index (ei)
si = ei (single)

2) Kaam :- divide, merge (lef, Rig)

merge

* Code:-

```
P S   void Print Array (int Arr [] ) {
for (int i = 0 , i < arr.length ; i++) {
        S.out ( arr[i] + " " ) ; }
        S.out ln () ; }

P S  void merge Sort (int arr [], int si, int ei)
{ // base    if (si >= ei) { return ; }
  // kaam
    int mid = si + (ei - si) / 2 // or (si+ei)/2
    merge Sort ( arr , si , mid) ; // left Part
    merge Sort ( arr , mid+1 , ei) ; // Right"
    merge (arr , si, mid , ei ); }

P S  void Merge (int arr[], si, ei, mid) {
  int temp [] = new int [ei - si + 1 ];
  int i = si  // left Part iterator
  int j = mid+1 // Right " "
  int k = 0  // " temp arr

  while (i <= mid and j <= ei ) {
     if (arr [i] < arr [j] {
       temp [k] = arr[i]
       i++, k++
```

```
else { temp[k] = arr[j];
       j++, k++; }
}
// left Part
while (i <= mid) {
    temp[k++] = arr[i++] }
// Right"
while (j <= ei) {
    temp[k++] = arr[j++];
// Copy temp to original array
for (k=0  k < temp.length; k++ i=si,i
    arr[i] = temp[k] }
}
```

* Time Complexity :- O(n log n)
  Space :- O(n)

* Quick Sort :- Pivot and Partition (any Point)
  6, 3, 9, 8, 2, (5) (Pivot)

* Code :-

```
P S   QuickSort(int arr[] si, ei) {
// last element (Pivot)
```

```
int Pind = Partition ( arr , si , ei );

PS   quickSort (arr , si , pind - 1); // left
     quickSort (arr , pind+1 , ei ); // Right Part.

     if (si >= ei) {
          return; }

PS int Partition (int arr [] , si , ei ) {

     int pivot = arr [ ei ];
     int i = si - 1 ; // make space for right elem

     for (int j = si ; j < ei ; j++) {
          if (arr [j] <= pivot)
               i++ ;

// Swap      int temp = arr [j];
               arr [j] = arr [i];
               arr [i] = temp ;  } } i++

     int temp = pivot;
     arr [ei] = arr [i];
     arr [i] = temp;
     return i;
```

* Worst Case TC :- $O(n^2)$.

* Modified Binary Search Rotated Sorted Array *

* Back Tracking *

* Types :- Decision (Yes / No)
Optimization (Shortest Path)
Enumeration (Many Soln)

* BT on Array :-

P S
```
void ChangeArr ( int Arr [], int i , int val
    // recursion
    arr [i] = val;
    ChangeArr (arr, i+1, val+1)
    arr [i] = arr [i] - 2;

    // base case
    if (i == arr.length) {
        printArr (arr);
        return; }
```

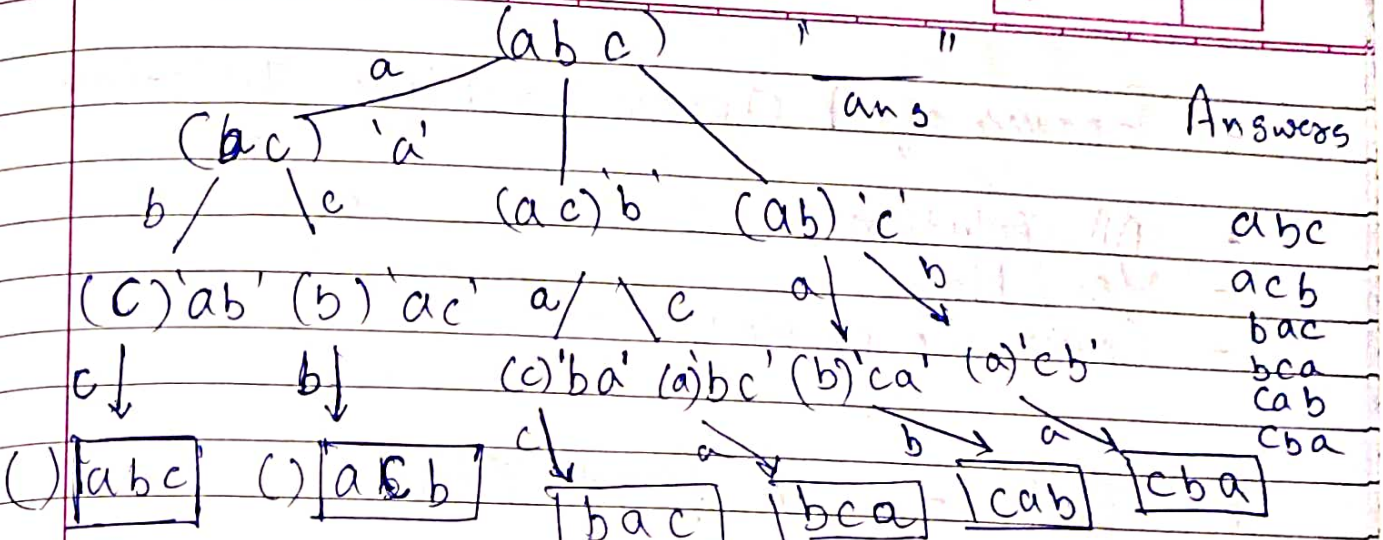
* Find Permutation (Different Arrangement
* (n) elements have (n!) permutation.

Q:- "a b c" n=3 = 3×2×1 = 6 permut

output :- [abc acb bac bca cab cba]

DSA (2)

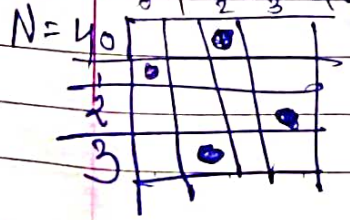
* Backtracking *




```
* PS  void findPerm(String Str String ans)
{   //recursion
    for(int i=0 i<Str.length(); i++){
        Char curr = Str.charAt(i);
String NewStr = Str.substring(0, i) + Str.sub(i+1)
    findPerm(NStr, ans+curr);

    //base case
    if(Str.length()==0){ S.out("ans")
    return;}
```

* N Queen *

Q Place N queen on an N×N chessboard such that no 2 queen can attack each other.



→ PTO

* Forms of N queen problem question
- All Solution
- Yes / No
- Count No of ways to Soln.

* Code :- // Part (1)

```
P S void nQueen(char board[][], int row){
  // column loop
for (int j = 0; j < board.length; j++) {
      board[row][j] = "Q ";
    nQueens(board, row+1); // function call
      board[row][j] = '.'; // backtracking

  // base case
  if (row == board.length) {
      printBoard(board)
        return;

P S  Print Board (char board[][]) {
  for (int i = 0 i< board.length; i++) {
      for (int j = 0 j< board.length; j++) {
          S.out(board[i][j] + " ")
  S.out ln();
```

```
PSVM {
    int n=2;
    Char board[][] = new char [n][n]
    //intialigation
    for (int i=0 i<n i++){
        for (int j=0 j<n j++){
            board[i][j] = ".";
        }
    }
    n Queen (board, 0);
    }
}
```

* Now is it safe to place coin

Q n=4?

```
if (issafe (board, row, j)){
    // column loop
```

```
PS boolean issafe (char board[][], int row, col){
    //vertical direction up
    for (int i=row-1; i>=0; i--){
        if (board[i][col] == 'Q'){
            return false;
        }
    }
    // Diagonal left upward
for (int i=row-1, j=col-1, i>=0, j>=0,
        i--, j--){
    if (board[i][j] == 'Q')
        return false;
```

// Diagonal right upward

```
for (int i = row-1, j = col+1; i >= 0 and j < board.length; i--, j++){
    if (board[i][j] == 'Q'){
        return false;
    }
}
return true;
```

// Same as part (1)

* Time Complexity :- O(n!)

T(n) = 1 Queen * T(n-1) + isSafe()
[O(n)] * T(n-1) + isSafe()

* Counting Ways for n Queen :-
Soln Set a count variable that track

* N Queen Print Yes or No (1 Soln)

* Grid Ways *

Find no of ways to reach from (0,0) to (n-1, m-1) in a N×M Grid

Allowed moves Right or Down.

row, col
function $(x, y) = f(x+1, y) + f(x, y+1)$
Total ways = " (Down) (Right)

Base case :- Atleast 1 way to reach target.

* Code :
```
PSint gridway(int i, j, row(n), col(m)) {
//base case
// condi for last cell
if (i == n-1 and j == m-1) { return 1; } else if
(i == n || j == n) { return 0; }
int w1 = gridway(i+1, j, n, m)
int w2 = gridway(i, j+1, m, m)
return w1 + w2;
PSVM { int n = 3, m = 3 Sout("0, 0, n, m")
```

Or use formula $\left[\frac{(n-1+m-1)!}{(n-1)!\,(m-1)!} \right]$

* Soduku *

Q Write func" to complete a soduku

* Code :-
```
PS boolean SudokoSolver(
int Soduku[][] int row int col) {
// base case
if (row == 9 and col == 9) {
return true }
else if(row == 9) { return false; }
// recursion) int nextrow = row, nextcol = col+1
if(col+1 == 9){
nextrow = row+1 for (int digit = 1 digit <= 9 digit++) {
nextcol = 0;
```

if (Soduko[row][col] != 0) {
return SudokoSolver(Soduko, nextrow nextcol)

if (isSafe(Soduku, row, col, digit)) {
Soduku[row][col] = digit
SudokoSolver(soduko, nextrow, nextcol)
return true; }
Sudoko[row][col] = 0
return false

PS isSafe(
// column condition
for (int i=0 i<=8 i++) {if(Soduko[i][col]
digit {return false}

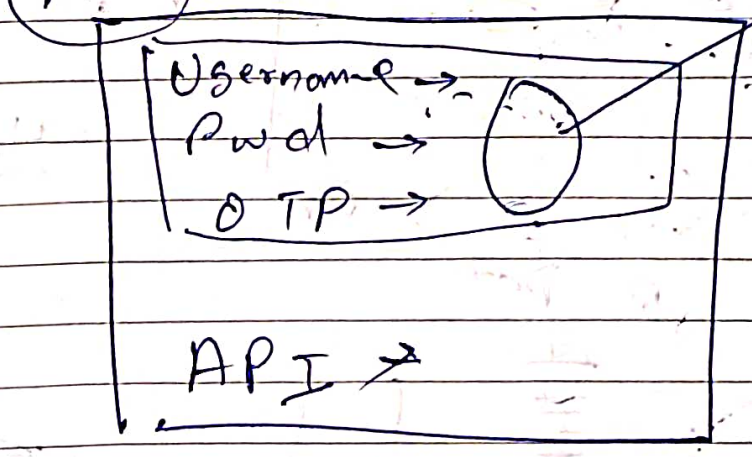
// row
for (int j = 0 j<=8 j++) {if
(Soduko[row][j] == digit
return false }

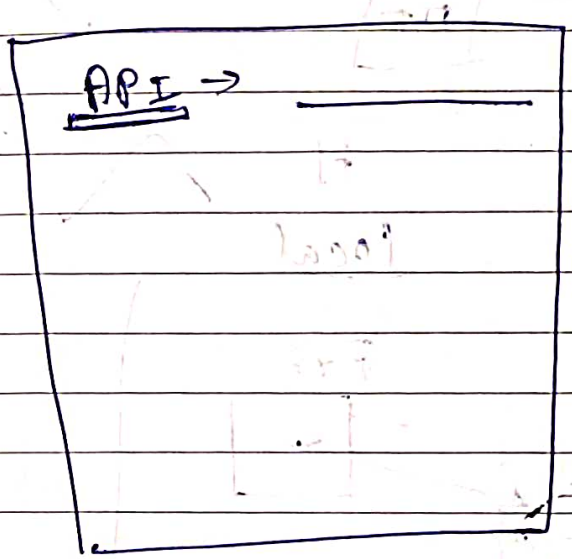
// grid
int startRow(sr) (row/3) * 3;
int startCol(sc) (col/3) * 3;
for (int i=sr i<sr+3 i++) {
for (int j = sc j<=sc+3 j++) {
if(Sudoko[i][j] == digit)
return false } } }

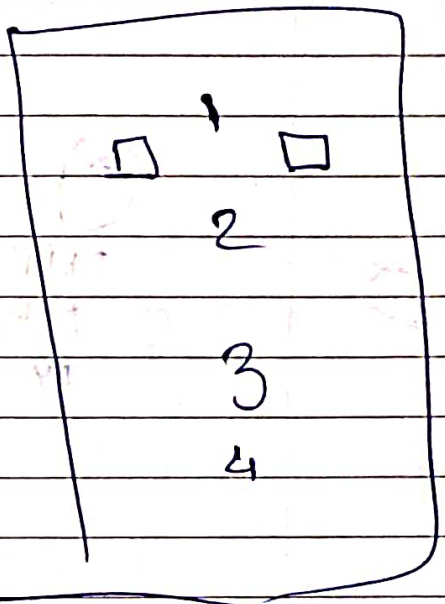
return true

i++
}(++

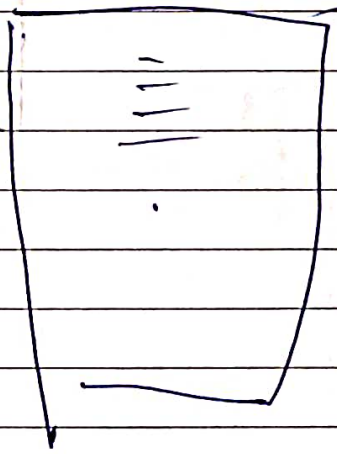
H → E
Q → S
AI →

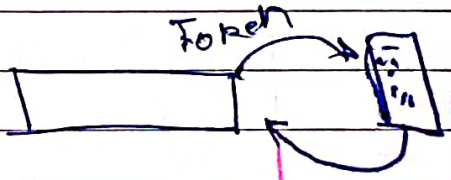


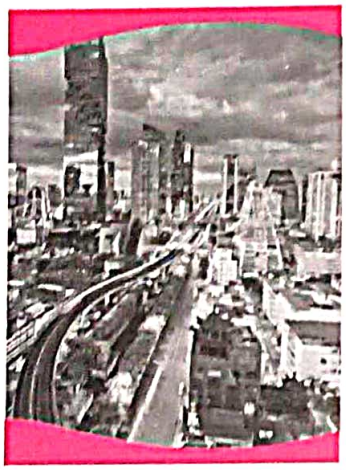




Fin -







MANMADE STRUCTURES

We, as human beings, have always appreciated beauty in all forms.

Inspired by the creations of nature, man has made aesthetic structures

which we all admire. In this series of designs,

we showcase magnificent human creations which will inspire the creator in you.

NAME: Sayed Hamza Sajid

STD: BE (2024) DIV: (IT) ROLL NO: YOUVA

SUBJECT: DSA + other Interview prep

INDEX

SR. NO.	DATE	TITLE	PAGE NO.	TEACHER'S SIGN

DSA 2

- Array List
- LinkedList
- Stack
- Queue

} linear Data Structure

Greedy Algo

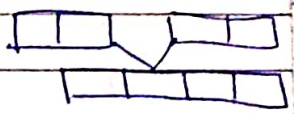
- Binary Tree
- BST
- Hashing
- Tries
- Heaps
- Graphs (½)

} Hierarchical DS

SR. NO.	DATE	TITLE	PAGE NO.	TEACHER SIGN

* DSA (2) *

Array	Array list
- fixed Size	- Dynamic Size
- primitive datatype can be Stored	- primitive Data types cant be Stored



* Array list == Vectors in C++

Arraylist <Integer> list = new Arraylist<>();
" <String> " " "
" <Boolean> " " "

* Operation :- • Add, Set, get, remove, contains
• Size()

* Sorting in Array list.
Collections. Sort()

* Multidimensional Arraylist

Arraylist < Arraylist < Integer >> mainlist = new Arraylist <> ();

---

* 2Pointer Approach * * Container with Most water *

Q for given 'n' lines on x axis use 2 lines to form a container such that it holds maximum water.

hieght = [1, 8, 6, 2, 5, 4, 8, 3, 7]
width = 7 $i_2 - i_1 = 8 - 1 = 7$

```
p s  (Arraylist <Integer> height) {
    int max water = 0;
    int lp = 0
    int rp = height.size() - 1

    while (lp < rp) {
// water Area
        int ht = min (heigh.get(lp), height.get(rp)
        int width = rp - lp;
        int Current water =    ht x width;
    Max water = Math.max(max water, curr water)

// update ptr.
    if (height.get(lp) < height.get(rp) {
            lp ++ } else { r -- }
    return max water

p s v m { Arraylist <Integer> height = new Arraylist<>()

    height.add();
    S.out (Storewater(height);
```

Q Pair Sum (Find if any pair in Sorted arraylist has target Sum

list [1, 2, 3, 4, 5, 6] target = 5

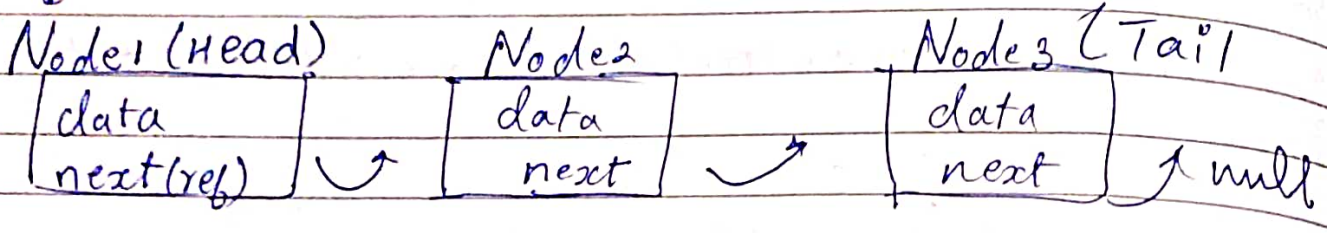
```
P S  boolean (ArrList <Integer> list, int target){
     int lp = 0
     int rp = list.size()-1;
     while (lp != rp){
       if (list.get(lp) + list.get(rp) == target){
           return true;
       if (      "  < target) { lp++ }
       if (      "  > ")  { rp-- }
     return false
```

* Rotated Pair Sum

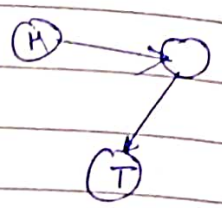
```
P S  PairSum2(  ){
     int bp = -1;        {int n = list.size();
     for (int i=0  i< list.size()  i++){
       if (list.get(i) > list.get(i+1) {
           bp = i
           break; }}
     int lp = bp+1;
     int rp = bp
     while (lp != rp) { if (lp+rp = target){ ret true
                                            }
       if (  " < tar){ lp = (lp+1)%n} else{
       rp = (n+rp-1)%n }
                                  } return false;
```

* Linked list * (LL)

What is ?



* Refrence variable point to an object (node)
* How to create a Node




```
Public static class Node {
    int data;
    Node next;

    Public node (int data) {
        this.data = 'data';
        this.next = 'null'; }

    Public static Node head }
    Public static Node tail }
```

* Add in Linked list :- (Add First)

① Create new node
② New node next = head
③ Head = new node.

```
Public void addfirst (int data) {
if (head == null) { head = tail = newnode; return; }
    // (1)  Node newnode = new Node (data);
    // (2)  newnode.next = head;
    // (3)  head = newnode;
}
```

* Add last : tail.next = newnode;
tail = newnode;

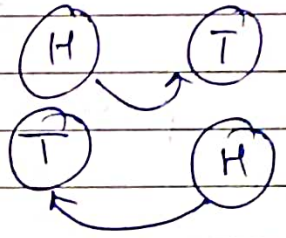
* Printing linked list :- temp → 1 → 2 → 3 → 4 → null
(H) (T)

PS print() { Node temp = head
while (temp != null) {
S.out (temp.data)
temp = temp.next;

* Remove in LL :- int val = head.data
First head = head.next
return val;

* Last Remove :-

* Reverse a linked list



P void reverse() {
Node prev = null;
Node curr = head = tail;
Node next;

while (curr != null) {
next = curr.next;
curr.next = prev
prev = curr;
curr = next; }

head = prev;

Q Find N^{th} node and Remove from end

```
PS deleteNth (int n) {
    int size = 0;                    // calculate size
    Node temp = head;
    while (temp != null) {
        temp = temp.next;
        size ++; }
    if (n == size) {
        head = head.next             // remove 1st
        return; }
    int i = 1
    int itofind = size - n ; Node prev = head
    while (i < itofind) {
        prev = prev.next
        i++ ; }
    prev.next = prev.next.next;
        return;
```

Q Check if link is Palindrome → Slow fast concept
↳ reverse half

① Find midnode
② 2nd half reverse
③ Check if 1st half = 2nd half.

* (Turtle - Hair) (Slow fast)
1 - 2 1 - 2

Step 1]

1] Public Node findMiddle (Node head) {

```
Node slow = head;
Node fast = head;

while (fast != null and fast.next != null) {
    slow = slow.next          // +1 node
    fast = fast.next.next.    // +2 node
}
return slow // mid found
```

2] Public boolean checkPalindrome () { if(head==null (head.next==null) return true;

```
// find mid (1)
Node midNode = findmiddle (head);
// (2) step reverse 2nd half
Node prev = null;
Node curr = midnode;
Node next;
while (curr != null) {
    next = curr.next;
    curr.next = prev;
    prev = curr;
    curr = next; }

Node righ = prev
Node left = head.
```

(d)

left

// Step 3 left = Right.

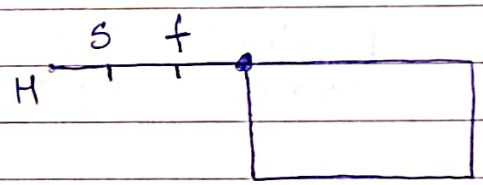
```
while(right != null){
  if(left.data != right.data){
      return false }
  left = left.next.
  right = right.next.
}
return true // is Palindrome.
```

* Detect cycle in LL * Floyds Cycle *

```
PS boolean isCycle(){
  Node slow = head;
  Node fast = head;
  while(fast != null and fast.next != null){
    slow = slow.next
    fast = fast.next.next.
// Cycleexist if(slow == fast){ return true} return false
```

* Remove Cycle in linked list *

* Approach



(1) Detect Cycle.

(a) slow = head

(b) slow → +1, fast → +1 | slow == fast // Cycle detected

```
PS void RemoveCycle() {
  // detect cycle
  Node slow = head
  Node fast = head ; boolean cycle = false;
  while (fast != null and fast.next == null) {
    slow = slow.next
    fast = fast.next.next
    if (fast == slow) {
        cycle = true;
        break; }
    if (cycle == false) {return}
  // find meeting point:
    slow = head;
    Node prev = null;
  while (slow != fast) {
    prev = fast
    slow = slow.next
    fast = fast.next. }

    prev.next = null;

PSVM { head = new node(1)
       head.next = new node(2)
       "  .next = "  "  (3)
       "  "  "  = head.
```

M T W T F S S
Page No.:
Date:
YOUVA

* JCF * * Java Collection Framework *

Study folder in "Photos" App

* LL in JCF :- 1] add first
2] add last
3] Remove first
4] remove last

* Merge Sort LL :-

* Public node mergeSort (Node head) {

// find mid

Node mid = getMid();

// left and right Merge Sort

Node righthead = mid.next

mid.next = null

Node newL = mergeSort (head)

Node NewR = mergeSort (righthead)

// merge

return merge (newLeft, NewRight)

// base case

if (head == null and head.next == null)

return head.

```
* Private Node getmid(Node head) {
    Node slow = head
    Node fast = head.next    // for even LL
  while ( fast != null and fast.next != null) {
    slow = slow.next
    fast = fast.next.next; }
      return slow; }

* Private Node merge(Node head1, Node head2){
    Node MergedLL = new Node(-1);
    Node temp = mergedLL;
    while(head1 != null and head2 != null){
  if (head1.data <= head2.data) {
     temp.next = head1;
      head1 = head1.next
       temp = temp.next; } else {
       temp.next = head2
         head2 = head2.next;
         temp = temp.next;  } }
    while (head1 != null) {
        temp.next=head1;
        head1 = head1.next
        temp = temp.next  }
while (head2 != null){
```

```
temp.next = head2;
head2 = head2.next;
temp = temp.next; }
return mergedLL.next.
```

* Zig Zag linked list :-

```
P V zigzag () {
```

i) // find mid

```
Node slow = head
Node fast = head.next;
while (fast != null and fast.next != null) {
slow = slow.next
fast = fast.next.next; }

Node mid = slow;
```

ii) // Reverse 2nd half

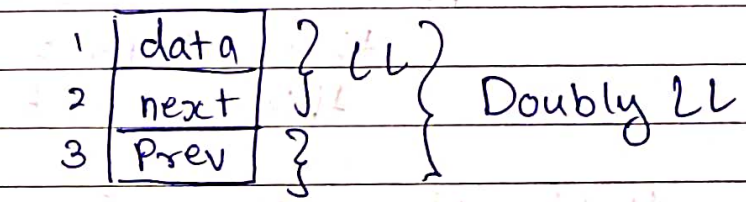
```
Node Curr = mid.next
mid.next = null
Node prev = null
Node next;
while (curr != null) {
next = curr.next
curr.next = prev.
prev = curr
curr = next; }
Node left = head
```

```
Node right = prev;
Node NextL, NextR;
```

3] // Zig Zag merge

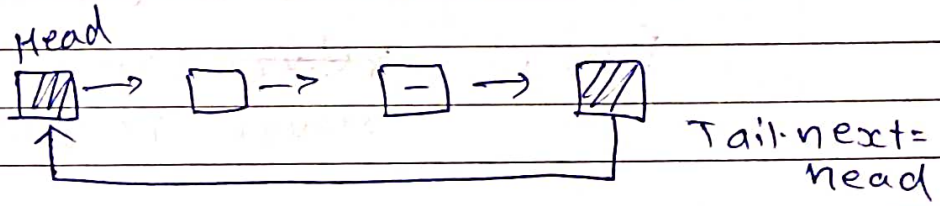
```
while (left != null and right != null) {
    nextL = left.next;
    left.next = right
    nextR = right.next
    right.next = nextL;

    left = nextL;
    right = nextR; }}
```

* Doubly LL :- and Reverse LL

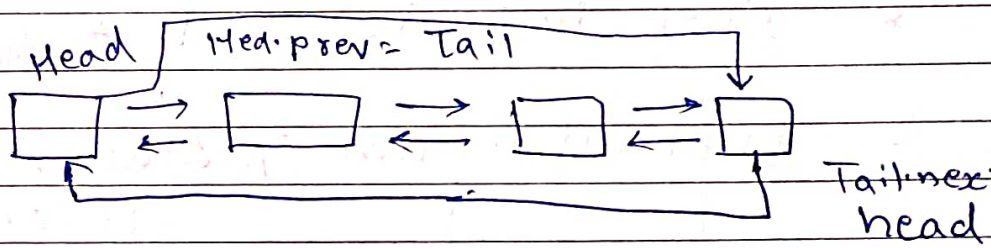


* Circular LL :-

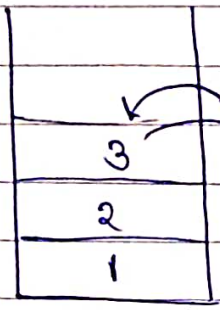
Singly LL :-



Doubly LL :-



* Stack * LIFO



1] Push O(1)
2] Pop O(1)
3] Peek O(1)

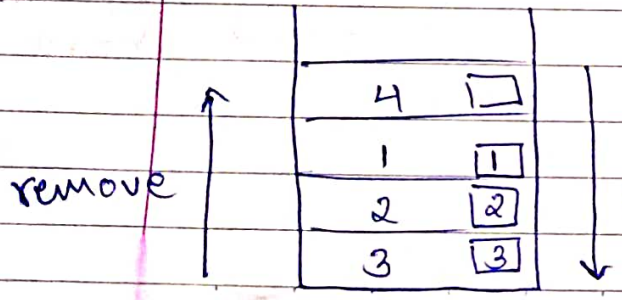
{LIFO}
{last in first out}

* Stack using LL * * Head is top in LL *

```
Public class Stack {
    Static class Node {
        int data
        Node next;
        Node (int data)
            this.data = data;
            this.next = null; } }
    Static class Stack {
        Static Node head = null;
    Public ~~return~~ Static boolean isEmpty() {
        return head = null; }
```

* Stak in LL in JCF * `Stack<Integer> S = new Stack<>();`

* Push at Bottom of Stack:-



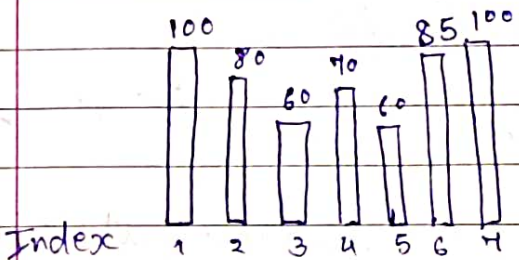

```
if (S.isEmpty()) { S.push(data)
                   return
int top = S.pop();
Push at Bottom (S, data)
S.push(top);
```

* Reverse a Stack ((*

```
PS v   reverse Stack(Stack<Integer>) S {
    int top = S.pop();
   reverse Stack (S);
   Push at Bottom (S, top);
```

* Stock Span problem :-
(Change)

Span → max no. of consecutive days for which Price ≤ todays Price



Span (1, 1, 1, 2, 1, 5, 7)

* Span = index(i) - prevHigh

```
PS void StockSpan(int Stocks[], int Span[]){
Stack<Integer> S = new Stack<>();
  Span[0] = 1;
  S.push(0);

 for(int i=1 ; i< Stock.length i++) {
    int currprice = Stock[i];
  while(!isEmpty() and currprice > Stocks[S.peak()]
{      S.pop(); }
    if(S.isEmpty()) {
          Span[i] = i+1; }
  else{ int prevHigh = S.peek()
}(
```

```
        Span[i] = i - prevHigh; }
        S.push(i); } }
PSVM { int stocks[] = {100, 80, 60, 70, 60, 85, 100};
  int span[] = new int [stock.length];
  Stocks span(stocks, span);
  for (int i=0  i<span.length i++) {
    S.out(spanp[i] +) } }
```

Next Greater Element :- O(n) ***

Q The next greater element of some element x in an array is the first greater element that is to the right of x in same array.

arr = [6, 8, 0, 1, 3]
next Greater = [8, -1, 1, 3, -1]
(6) (8) (0) (1) (3)

* Code

```
PSVM { int arr[] = {6, 8, 0, 1, 3};
Stack<Integer> s = new stack<>();
int nextGreater[] = new int [arr.length];
for (int i = arr.length - 1; i >= 0; i--) {
```

// while
while(!isEmpty() and arr[s.peak()] <= arr[i]) {
s.pop(); }

② // if-else
if (s.isEmpty()) {
nextGreater[i] = -1; }
else { nextGreater[i] = arr[s.peek()]; }

// 3 Push in s
s.push(i);

for (int i=0 i<nextGreater.length; i++) {
s.out(nextGreater[i] + " ")

---

* Valid Parenthesis *

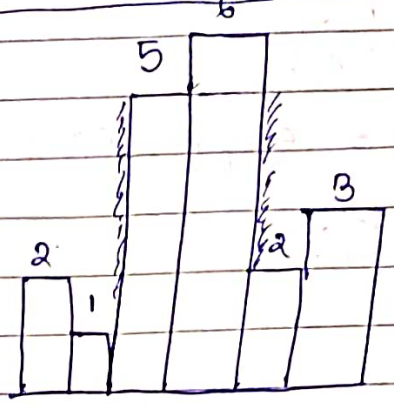
Given a String s containing just the character '(', ')', '{', '}', '[', ']' determine if input String is valid [ ] , ( ) , [ )]
Yes, Yes, No

Input String is valid if :-

1) - Open is closed by same (()) bracket type
2) - Open is closed in correct order
3) - Every close has corresponding open of same type.

```
PS void M {
    String Str = "([{})[]";
    S.out (isValid(Str));
Public Static boolean isValid (String Str){
Stack<character> S = new Stack<>();
for (int i=0, i< Str.length(); i++ ) {
   char ch = Str.CharAt(i);
   //opening
if (ch=='(' || ch=='{' || ch=='[' ){
        S.push(ch); }
   else{ if (S.isEmpty()) {
        return false; }
if (ch==')'and S.peek()=='(') ||
    S.peek()=='{' and ch=='}') ||
     S.peek()=='[' and ch==']') {
        S.pop(); }
     else { return false; } }}
if (S.isEmpty()){
     retur true; }
else {return false}
```

Q Max Area in Histogram:- **



Given an array of integer height represing Hist bar Height where the width of each bar is 1 return the area of largest rectangle in histogram.

height = [2, 1, 5, 6, 2, 3]

Ans = 10

Area = hex width

for 5 boundries of next Smallest left, Right.

width = (j - 1 - 1)

NSLeft (i) NSRight (j)

* Code:-

```
P S V  Max Area(int arr[]) {
  // Next Smaller Right (NSR)
  int maxArea = 0;
  int nsr[] = new int[arr.length];
  int nsl[] = new int[arr.length];
  // Next Smaller Right
  Stack<Integer> s = new stack<stack>();
  for(int i=arr.length-1; i>=0; i--)
    while(!s.isEmpty() and arr[s.peek] >= arr[i]){
```

```
S.pop(); } if(S.isEmpty()){
    nsr[i] = arr.length;
} else {
    nsr[i] = S.peek();
}
S.push(i);
```

// Next Smaller left

```
S = new Stack<>();
,, // Same as NSR
while(!isEmpty() and arr[S.peek()] >= ar
" // Same as NSR
```

// Current Area

```
for(int i=0 i<arr.length i++){
    int height = arr[i]
    int width = nsr[i] - nsl[i] - 1
    int CurrArea = height * width
Max Area = Math.max(Currarea, maxarea
}
S.out(maxarea)
```

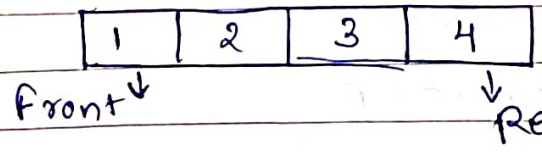
```
PSVM{
    int arr[] = {2,1,5,6,2,3}
max area(arr); }
```

}[i] ans

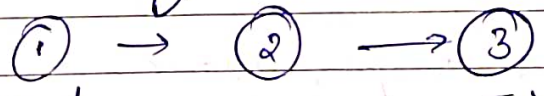
* Queue * * FIFO *

* is a Interface *

* Operations :- Add O(1) - Enque.
Remove O(1) - Dequeue.
Peek O(1) - Front.



Q Queue using linked list :-



Head (Front) Tail (Rear) Tail.next = newnode

```
Static class Node {
    int data;
    int next;

Node (int data)
    this.data = data.
    this.next = null;
Static class queue {
    Static Node head = null;
    Static node tail = null;

P S boolean isEmpty () {
    return head == null and tail == null; }
//add
P S V add (int data) {
    if (isfull Node newnode = new Node(dat
```

```
if (head == null) {
    head = tail = newnode
    return; }
tail.next = newnode
tail = newnode

// remove
PS int remove() {
if (isEmpty()) {
    S.out (Empty queue)
    return -1;

int front = head.data;
// Single element
if (tail == head)
  tail = head = null; }
else { head = head.next; }

return front;
// Peep
PS int Peep() {
if (isEmpty()) {
S.out ("empty queue)
    return -1 }
return head.data;
```

* Queue in LL using JCF *

```
Queue<Integer> q = new LinkedList<>();
```

* Queue using 2 Stack. * Queue
* Make FIFO using 2 LIFO *

Push() Pop()

* Code

```
Static class Queue {
    Static Stack<Integer> s1 = new Stack<>();
    Static "  "  s2 = "  ;

PS boolean isEmpty () {
    return s1.isEmpty ();
// add
  PS add {
  while (!s1.isEmpty()) { s2.push (s1.pop())}
    s1.push (data)
    while (! s2.isEmpty() ) {
      s1.push (s2.pop ())} }
// remove
PS remove () {
    return s1.pop ();
```

* Deque * Double ended queue *

* Mainly Sorting in this Algo * * Greedy Algorithm * Approach *

① ~~Optimization → Min, Max~~: It is a problem solving technique where we make locally optimum choice at each step

Pros	Cons
Simple/Easy Good enough TC	A lot of time global optimum is not achieved

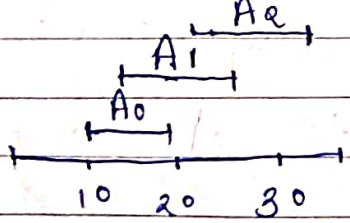
* Activity Selection :-

Q You are given 'n' Activity with their start and end time select max^m number of activities that can be performed by a single person assuming that a person can only work on a single activity at a time

* Activities are Sorted according to end times

```
        A0  A1  A2
Start = [10, 12, 20]
End   = [20, 25, 30]
```

Ans = 2 (A0 and A2)



{ Non overlaping Activity
2 (A0, A2)

* Approach :-

```
        A0 A1 A2 A3 A4 A5
Start = [1, 3, 0, 5, 8, 5]
        [2, 4, 6, 7, 9, 9]
```

* Non overlaping (disjoint) :- Start Time >= last activity end time

* Code :-

```
PS vM {
    int Start [] = { 1, 3, 0, 5, 8, 5 }
    int end [] = { 2, 4, 6, 7, 9, 9 }

    // end time basis sorted
        int maxAct = 0;
Arraylist < Integer > ans = new Arraylist<>();

    // 1st Activity
    Max Act = 1;
    ans.add(0);
int last end = end[0];
    for (int i=1 ; i < end.length ; i++)
        if (Start [i] >= last end) {
        max act ++;
        ans.add (i);
        last end = end [i]; }}
        S.out (max Act)
        for (int i=0 i < ans.size () i++) {
            S.out (ans.get (i)); }
```

* Activities of end time is Unsorted :-

```
// Sorting
int Activities [] [] = new int [ Start.length [3]
for (int i = 0  i < Start.length } i++) {
```

```
activities[i][0] = i
activities[i][1] = start[i];
activities[i][2] = end[i];
// lambda function                    }
Arrays.sort(activities, Comparator.comparing(doubl
                                (o -> o[2]));
```

Pg108] * **Comparator :-** It is Interface used to order objects of user defined classes it is capable to compare two object of same class.

* **Fractional Knapsack :-**

Given weight and value of N times Put these items in a knapsack of capacity W to get the maximum total value in knapsack.

Value = [60, 100, 120]
Weight = [10, 20, 30] Ans = 240
W = 50 max

* Approach :- ratio [6, 5, 4]

PSVM {
```
int val[] = {60, 100, 120}
int weight[] = {10, 20, 30}
int W = 50;
double ratio[][] = new double[val.length][2];
// 0th col → index   1st col → ratio //
```

[5] 0

```
for (int i=0 ; i<val.length ; i++) {
    ratio [i][0] = i;
    ratio [i][1] = val[i] /(double) weight[i];
//Acending order Sorting
Arrays.Sort (ratio, Comparator.ComparingDouble(o->o[1]
    int capacity = W;
    int final val = 0;
for (int = ratio.length-1  i>=0  i--) {
    int indx = (int) ratio [i][0];
    if (capacity >= weight [indx]) {
        final val += val [indx];
        capacity -= weight [indx]; }
    else { final value += (ratio [i][1] *
                                    capacity)
        capacity = 0;
        break; }
    S.out (final value);
```

---

* Job Sequencing Problem :-

Given array of jobs where every job has a deadline and profit if job is finished before deadline it is also given that every job takes single unit of time, so that min^m possible deadline for any job is 1. Maximize the total profit if only one job can be scheduled at a time.

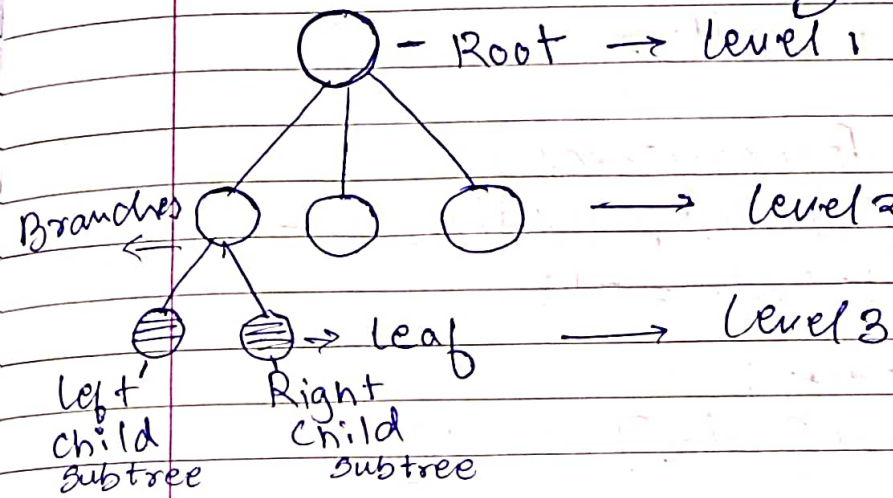
Job A (4, 20) Job B (1, 10) C (1, 40) D (1, 30)

[Ans CA]

```
Static class Job {
    int deadline;
    int profit;
    int id;
}

PSVM {
int JobInfo[][] = {{4,20},{1,10},{1,40},{1,30}};
Job jobs[] = new Job[jobsInfo.length];
ArrayList<Job> jobs = new ArrayList<>();
for(int i=0 i<jobs.length i++) {
    job[i] = new job(i, JobInfo[i][0],
        JobsInfo[i][1]); }
Collections.Sort(jobs, (obj1, obj2) ->
        obj2.profit - obj1.profit);
// decending order of profit
ArrayList<Integer> Seq = new ArrayList<>();
for(int i=0 i<job.Size() i++) {
    job curr = job.get(i);
    if(curr.deadline > time) {
        Seq.add(curr.id);
        time++; } }
S.out(+ Seq.Size());
for(int i=0 i<Seq.Size(); i++)
    S.out(Seq.get(i)); }

public Job(int i, int d, p){
    id = i;
    deadline = d;
    profit = p; }
```

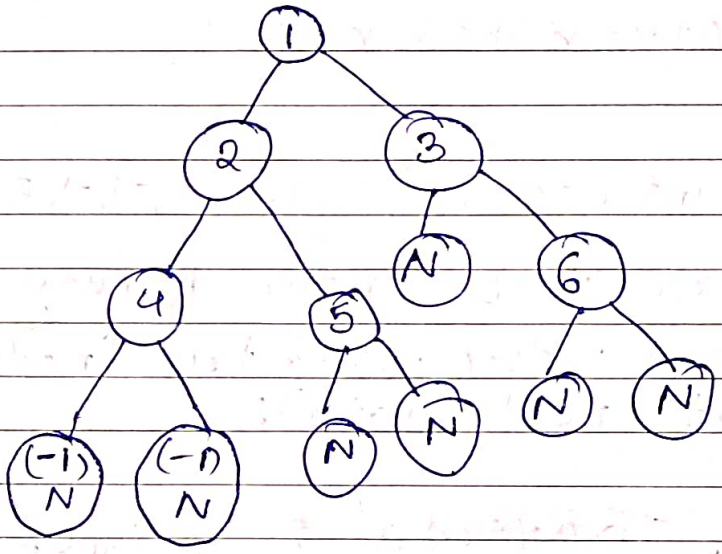
* Chocola Problem :-

* Binary Tree * Hierarchical Data structure



* Build tree preorder :-

1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1



N -> Null

* Code

```
Pub Class BinaryTreesEx{
    Static class Node {
        int data;
        Node left;
        Node right;
```

```
Node( int data) {
    this.data = data
    this.left = null
    this.right = null. }

Static class BinaryTrees {
    Public static Node buildtree (int nodes[]) {
    , Static int indx = -1;
        { indx++;
        if (nodes[idx] == -1) {
            return null; }
        Node newnode = new Node (nodes[idx]);
    {   newnode.left = buildtree (nodes);
        newnode.right = buildtree(nodes);
            return newnode;

PSVM {
        int nodes[] = {1,2,4,-1,-1,5,-1,-1,3,-1,6,-
    BinaryTree tree = new BinaryTree();
    Node root = tree.buildtree (nodes)
      S.out (root.data);
```

1] * Pre order Traversal :- (Types) (O(n))

1] Root — DFS

2] left Sub tree

3] Right "

```
Pub static preorder (Node root) {
    if (root == null)
        return; {
    S.out (root.data + " ")
    preorder (root.left)
    preorder (root.right); }
```

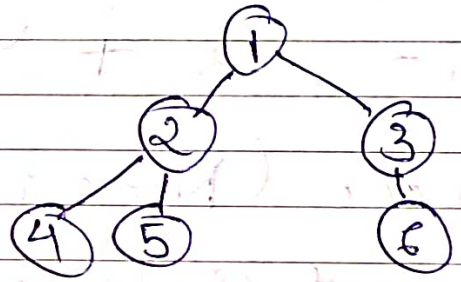
2] * Inorder :- 1] left Subtree DFS
2] Root
3] Right Subtree

```
P S V  inorder (node root) {
    if (root == null) {
        return {
    inorder (root.left)
    S.out (root.data + " ")
    inorder (root.right)
```

1,-1}

3] * Post Order :- 1] left 2] Right 3] root (DFS)

4] * level Order :- using FIFO (BFS)



1 → L1

2 3 → L2

4 5 6 → L3

```
PS levelorder (Node root) {
Queue<Node> q = new LinkedList<>();
    q.add(root);
    q.add(null);

While (!q.isEmpty()){

    Node currnode = q.remove();
    if (currnode == null) {
        S.out();
        if (q.isEmpty()) {
            break }
    }else { q.add(null);}

  else{
        S.out(currnode.data + " ")
        if (currnode.left != null) {
            q.add(currNode.left); }
        if (currnode.right != null){
            q.add (currennode.right }
```

---

* Questions *

* Using Recursion :-

Q) Count of Nodes :- x + y + currnode

```
PS int Countnode (Node root) {
int leftnode = count of Nodes (root.left)
int Rightnode = count of Nodes (root.right)
```

return leftNode + rightnode + 1;

Q2] Sum of Nodes :-

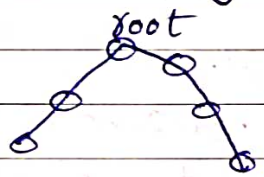
```
PS  sumofNodes (Node root){
if (root == null){
        return 0 ?
int leftSum = SumofNodes (root.left)
int rightSum = SumofNodes (root.right)
return leftSum + rightSum + root.data;
PSVM {
        S.out (Sum of Nodes (root))
```

Q3 Height of Tree :-

```
PS height (Node root) {
// same as above
int myHeight = Math.max (leftHeight, rightHeight)
                + 1
return myHeight;
```

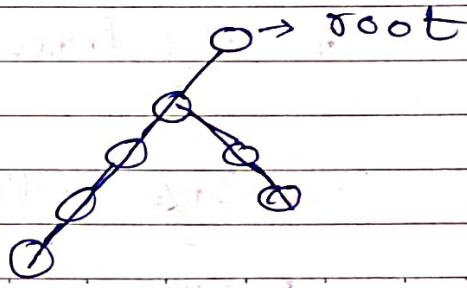
Q4 Diagmeter of Tree : No. of Nodes in a longest path between any 2 Nodes

1] Case 1 :-



Traverse through root

2] Case 2 :- Not through root

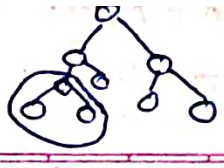


Approach 1 :- $O(n^2)$

```
PS  int diameter (Node root) {
    int diam1 = diameter (root.left)
    int diam2 = diameter (root.right)
    int diam3 = height (root.left) + height(
                (root.right) + 1

    return Math.max (diameter 3, Math.max
                (diam1 , diam2))
```

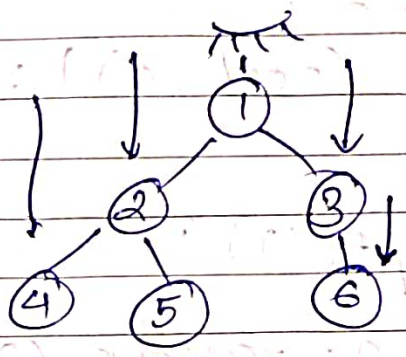
Approach 2 :- $O(n)$ (Height , Diameter)

```
Static Class TreeInfo {
    int ht ;
    int diam ;
    TreeInfo (int ht, int diam) {
        this.ht = ht ;
        this.diam = diam ;

PS  TreeInfo diameter2 (Node root) {
    TreeInfo left = diameter2 (root.left)
    TreeInfo Right = diameter2 (root.right)
    int myHeight = Math.max (left.ht, right.ht) +
    int diam1 = left.diam
    int diam2 = right.diam
    int diam3 = left.ht + right.ht + 1
    int myDiam = Math.max (Math.max (diam1, diam2
                    diam3) ;
    TreeInfo myInfo = new TreeInfo (myHeight, myD
    return myInfo ;
```



Q5] Sub Root Identical problem :-

Q6] Top View of Binary Tree :-



1) (o(n)) all * HashMap * Key : Value
↓ unique ↓ different

① Create :-

```
HashMap <String, Integer> map = new Hashmap <> ();
```

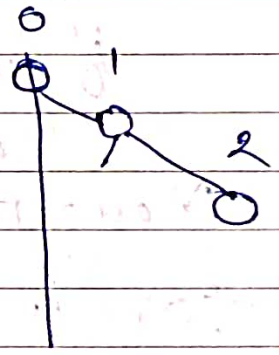
② Add :- (Put)

```
map.put (key, value)
```

③ Get :-

```
map.get (key)
```

2] Horizontal Distance




```
Static class Info {
    Node Node;
    int hd;
```

```
Public Info (Node node, ...){
    this.node = node;
    this.hd = hd;
}

P S void topview(Node root) {
//Level Order
    Queue<Info> q = new LinkedList<>();
HashMap<Integer, Node> map = new HashMap<>();

int min = 0  max = 0;
 q.add (new Info (root, 0));
  q.add (null);

 while (!q.isEmpty()){

      Info curr = q.remove()
      if (curr == null) {
          if (q.isEmpty()) {
              break;
      }else {
                  q.add (null); }
//hd
          if (map.containskey (curr.hd)){
              map.put(curr.hd, curr.node)

      if (curr.node.left != null) {
          q.add (new Info (curr.node.left,
                  curr.hd-1)
      min = Math.min (min, curr.hd-1) }

      if (curr.node.right != null ) {
          q.add (new Info (curr.node.right, curr.hd
Max = Math.max (max, curr.hd+1) }}
for(int i = Min  i<= max ; i++) {
      S.out (map.get (i).data + " ")
```

* Binary Search Tree *

O(n) → Binary tree n → no. of nodes
O(H) → Binary Search Tree H → height

Q What is BST?

* Property:- a) Left Subtree Nodes < Root
b) Right " > Root
c) Left and Right Subtree are also BST with no duplicates

* Search in BST :- O(H)

Root = value value > Key → go left
Value < Key → go Right

```
PS boolean Search (Node root, int Key) {
    if (root == null) {
        return false }
    if (root.data == key) {
        return true }
    if (root.data > key) {
    return Search (root.left, Key) }
    else { return Search (root.right, Key)
    } }
```

* Delete a node :- Inorder Successor → left Most node in Rign Subtree

* Root to leaf :- Backtrack by leaf node and delete leaf node.

* Validate BST :- Inorder traversal (Sorted)

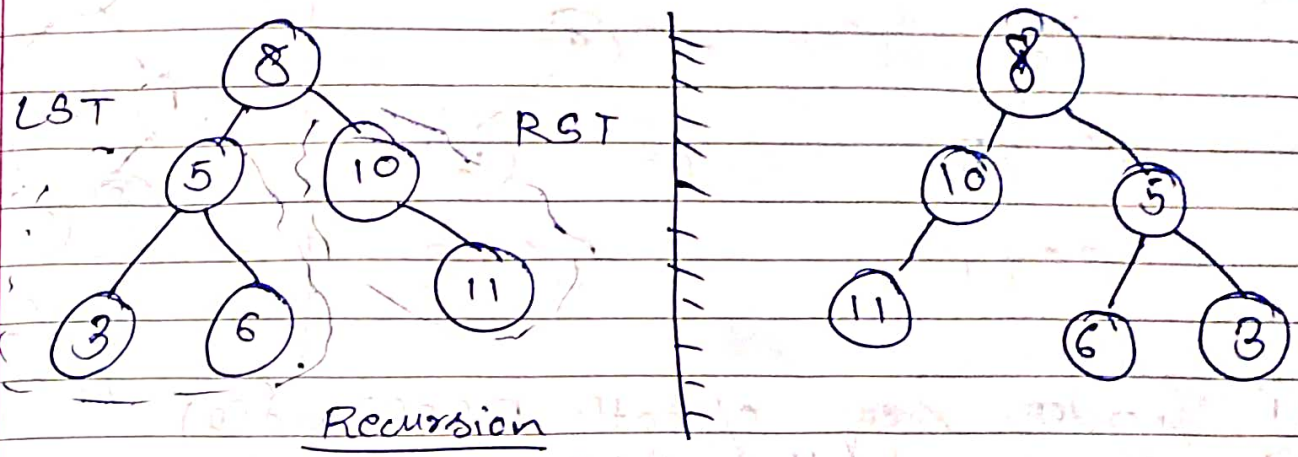
Approach :- Minvalue in Right Subtree > Node
Max value in left Subtree < Node

* Code :- PS boolean isvalidBST (Node root, Node min, Node max)

```
if (root == null) {
    return true; }
if (min != null and root.data <= min.data)
    return false; }
else if (max != null and root.data >= max.d
    return false }
return isvalidBST (root.left, min, root)
    and isvalidBST (root.right, root, max)
```

* PSVM {int values[] = { 8,5,3,6,10,11,14 };
Node root = null;
for (int i=0 i<value.length; i++) {
root = insert (root, values[i]);
inorder (root);
S.out ()
if (is validBST (root, null, ~~max~~ null)).
S.out (valid) else { notvalid)

* Mirror a BST :-



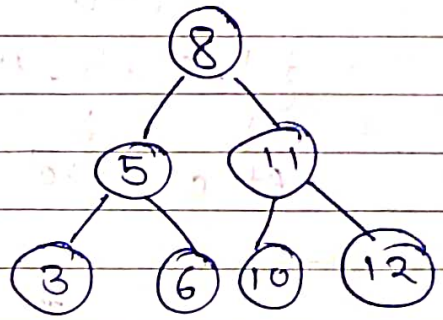
* Sorted Array to Balanced BST *

{3, 5, 6, 8, 10, 11, 12}

st=0 (5, 6) Mid end=6

(Inorder Sequence)

sorted

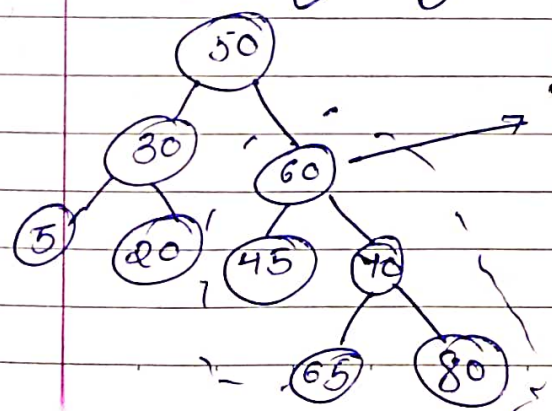


* Convert BST (unbalanced) to Balanced BST

* Approach :- Sorted → Balanced.

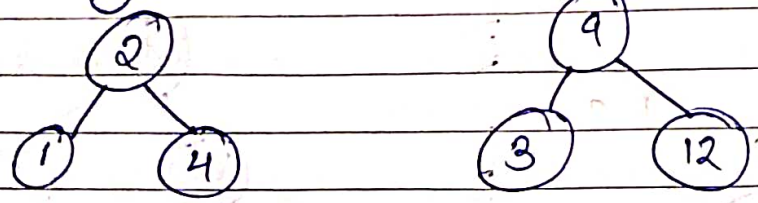
① Inorder Seq func | ② Sorted Arraylist → Balanced BST

* Size of Largest BST in BT :-

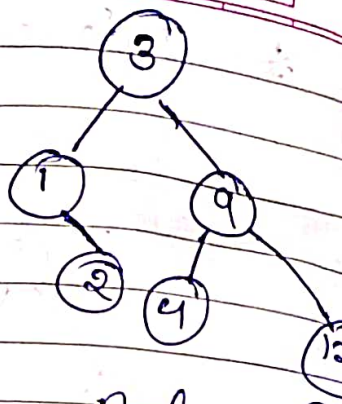


Size = 5

* Merge 2 BST



BST 1 BST 2



Balanced Merged BS

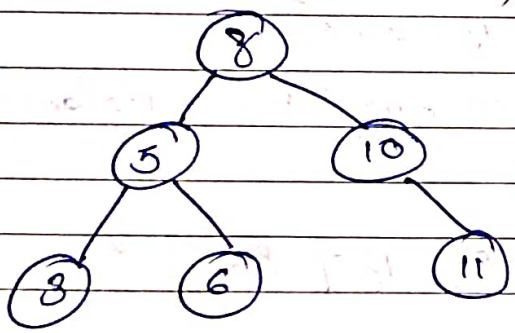
(1) Inorder Seq of both BST(1) and (2)
(2) Merge both BST
(3) Sorted arr → Balanced BST
List

{1, 2, 3, 4, 9, 12}

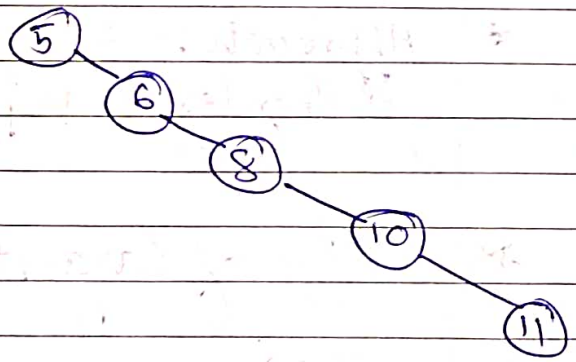
st=0 mid end

* AVL trees :- Self Balanced BS

Balanced (AVL)



unbalanced



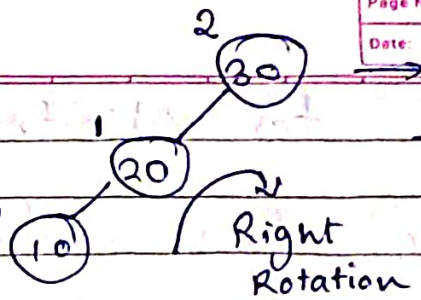
Property Balanced factor

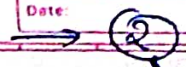
$|HL - HL| < 2$

↓

AVL

* Balance factor :-



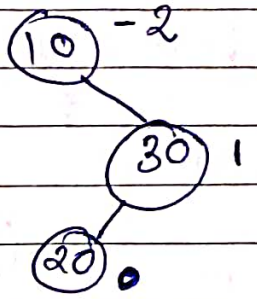


(1) Left Left Case
↓
Right Rotation (RR)

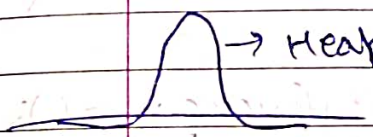
(2) Right Right → LR

(3) LR → ① LR ② RR →

(4) RL → ① RR ② LR →



* Heaps * Priority Queue



* PQ in Java Collection framework (JCF)

func" :- add() , remove() peek()

* PriorityQueue<Integer> pq = new PriorityQueue<>();

using PSVM {

pq.add(3), (4), (5) (6)

while (!pq.isEmpty()) {

S.out(pq.peek())

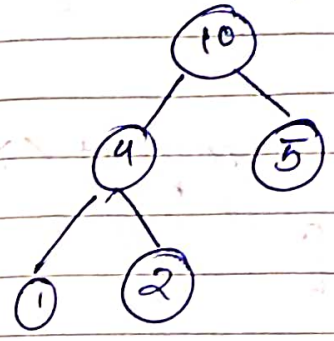
pq.remove();

* PQ for objects :-

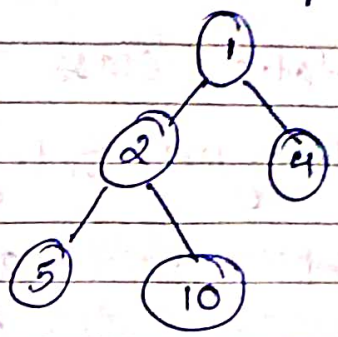
```
Public class StudentClassroom {
                    implements Comparable<Stud
    Static class Student {
        String name
        int Rank

        Public Student (String name int Rank)
            this.name = name
            this.rank = rank }}

        @Override
        Public int Compareto (Student s2){
            return this.rank - s2.rank ; }}
PSVM {
PriorityQueue<Student> Pq = new PriorityQueue:<>();
 pq.add (new Student (name "A"  rank 4) )
Pq.add (      ''          "B"    ''  1 ) )
Pq.add (  ''              "C"    ''    ) ) )

While (! Pq.isEmpty()) {
    S.out (Pq.peek().name + → + Pq.peek().r
    Pq.remove ()
```

* Heap → Binary Tree in Priority

Max Heap



Min Heap



* Property :- 1] It is a Binary Tree with 2 children

2] Complete Binary tree :- Completely filled than last

3] Heap Order Property :-

Children >= Parent (Min Heap) - Known as
Children <= Parent (Max Heap) "

* Implement heap always using array and A list

* How to find Index in Heap ?

(node) indx = i

left child = 2i + 1
right child = 2i + 2
Child (x) → Parent $\left(\frac{x-1}{2}\right)$

* In Heap index Starts with 1 not 0

O(log n)

* Insert in Heap :- Step :- ① add in last ind
② fix heap (swap)

eg:- Static class Heap {

Arraylist<Integer> arr = new Arraylist<>();

Public void add (int data) {

arr.add (data)

int x = arr.size() - 1; // x is child index
int Parent = (x-1)/2; // Par index

while (arr.get(x) < arr.get (Parent) {

// Swap

int temp = arr.get(x)
arr.set (x, arr.get (par))
arr.set (par, temp) }}

* Delete in Heap

① 1st and last node Swap
② remove last index (arr.remove (arr.size()-
③ Heapify function (fix heap)

Heapify → root = i
left = 2i + 1
right = 2i + 2

Min (i, 2i+1, 2i+2)
Swap with root

```
public int remove() {
  int data = arr.get (index 0)
// Step 1 → Swap 1st and last

  int temp = arr.get(0)
  arr.set (0, arr.size()-1)
  arr.set ( arr.size()-1 , temp )

// Step 2 → delete last
    arr.remove (arr.size() - 1);
```

* // Step 3 Heapify :- (O (log n))

```
private void Heapify (int i) {
    int left = 2*i+1 ;
    int right = 2*i+2;
    int minindex = i ;

  if (left < arr.size () and arr.get (minIdx)
                arr.get (left)                    >
{       Min Indx = left }
if (right < arr.size() and arr.get(minIndx)>
                arr.get (right) {
              Min Indx = right;  }
  if (minIndx != i) {
      // Swap  int temp = arr.get (i)
arr.set (i, arr.get (minIndx))
arr.set (minIndx , temp) ;
    heapify (minIndx)
```

// Step 3

heapify (i : 0)
return data; }}

---

* Heap Sort :- $O(n \times \log n)$

Ascending Order → Max Heap
descending order → Min Heap

* Steps :- 1) Convert arr → Max Heap
Non-leaf Node → Heapify
of last node
* for (int $i = \frac{n}{2}$; $i >= 0$; $i--$)

2) Sort Ascending order (Selection Sort Swap)
largest element push at last
Swap (0, last)
heapify (0) → (n - 1)

```
Public Static Void HeapSort (int arr [])
    // Step 1 build max Heap
    int n = arr.length;
    for (int i = n/2 ; i >= 0 ; i--)
        heapify (i , n);
                 arr,
```

Step 2 Push largest at end

```
for (int i = n-1 ; i > 0 ; i--) {
  // Swap (1arg with ,5t)
   int temp = arr[0]
     arr[0] = arr[i]
     arr[i] = temp

  heapify (0, i) ;
           arr,
```

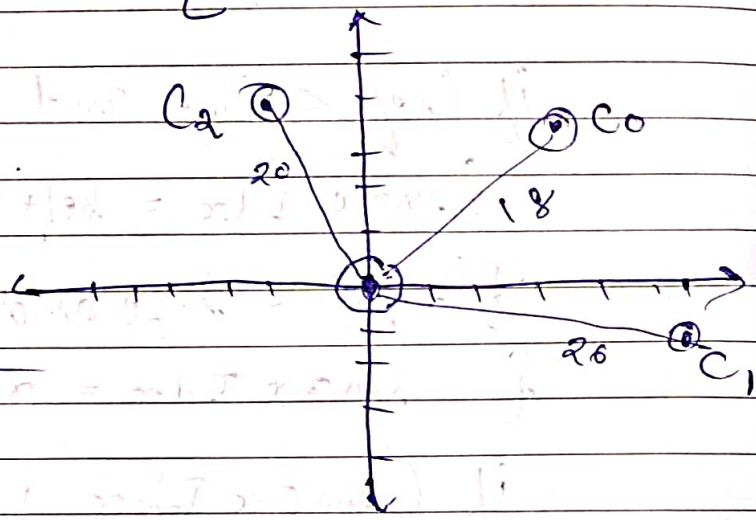
* P S `void heapify (int i, int size) {` `int arr[],`

```
int left = 2*i+1 ;
int right = 2*i+2;
int maxIdx = i;

if (left < size and arr[left] > arr[maxIdx]
 { maxIdx = left; }
if (right < size and arr[right > arr{maxIdx]
{  maxIdx = right;
 if (maxIdx != i) {
     // Swap
int  temp = arr[i]
  arr[i] = arr[maxIdx]
  arr[maxIdx] = temp
  heapify (arr , Max Idx size)
```

```
PSVM {
    int arr[] = { 1, 2, 4, 5, 3 };
    heap Sort (arr)
//print
for (int i=0  i<arr.length  i++)
    S.out (arr[i] + }
```

Q Nearby Cars :- We are given N point in 2D plane which are locations of N cars. If we are at the origin print the Nearest K cars $K = 2$ { distance = $\sqrt{x^2 + y^2}$

C_0 (3, 3) = 18
C_1 (5, -1) = 26
C_2 (-2, 4) = 20



Answer = C_0 and C_2

```
Static class Point implement Comparable<point
    int x : y
    int distance Square
    int indx;
Public Point (int x, y, distanSq int idx)
```

```
this.x = x
this.y = y
this.distanceSq = distanceSq
this.indx = indx
```

(a) override

```
Public int compareto (point p2) {
    return this.distanceSq - p2.distanceSq
```

PSVM {

```
int pts[] = {{3,3},{5,-1},{-2,4}}
int K = 2
PriorityQueue<Point> pq = new PriorityQueue<>()
for (int i = 0 i < pts.length i++) {
int distanceSq = [x^2 + y^2]
```

pq.add(new Point(pts[i][0], pts[i][1], distanceSq, i))

```
// Nearest K cars
for (int i = 0 i<k; i++) {
    S.out ("C" + pq.remove().indx)
```

}

Q **Connect N Ropes:-** Give N ropes connect them into one rope with minimum cost such that cost of connect two rope is equal to sum of their length

Rope = {4, 3, 2, 6}

Connect 2 and 3 [5]
Connect 5 and 4 [9]
" 9 and 6 [15]

* Approach Solve using priority Queue :-

```
PSVM {
    int ropes[] = {2, 3, 3, 4, 6};
PriorityQueue<Integer> pq = new PriorityQue<>()
for (int i=0 i<ropes.length i++) {
    Pq.add (ropes[i]) }

int cost = 0
while (pq.Size() > 1) {
    int min = pq.remove()
    int min2 = pq.remove()
    cost += min + min2;
    pq.add (min + min2) }
    S.out (cost)
```

Q Weakest Soldiers :-

Q Sliding Window Maximum (Hard level)

Maximum of all Subarray of size K

K = 3 Subarray 2 (max = 4)

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

Subarray 1 (max = 3)

Ans = 3, 4, 5, 6, 7, 8, 9, 10

PQ as window as Max Heap

* Step :- ① Add K number to PQ (1st Window)

② ① PQ.peek () → Window[0]

② while (pq.peek().indx <= (i - K))

pq.remove

pq.add (current)

wind[] = pq.peek()

* Approach :- Q = 1, 3, -1, -3, 5, 3, 6, 7

Ans = 3, 3, 5, 5, 6, 7 (K = 3)

Code (O(n log K))

```
Static Class Pair {implement Comparable<Pair>{
    int value
    int indx

Public Pair (int value, int indx) {
    this.value = value
    this.indx = indx
```

(9) Overide

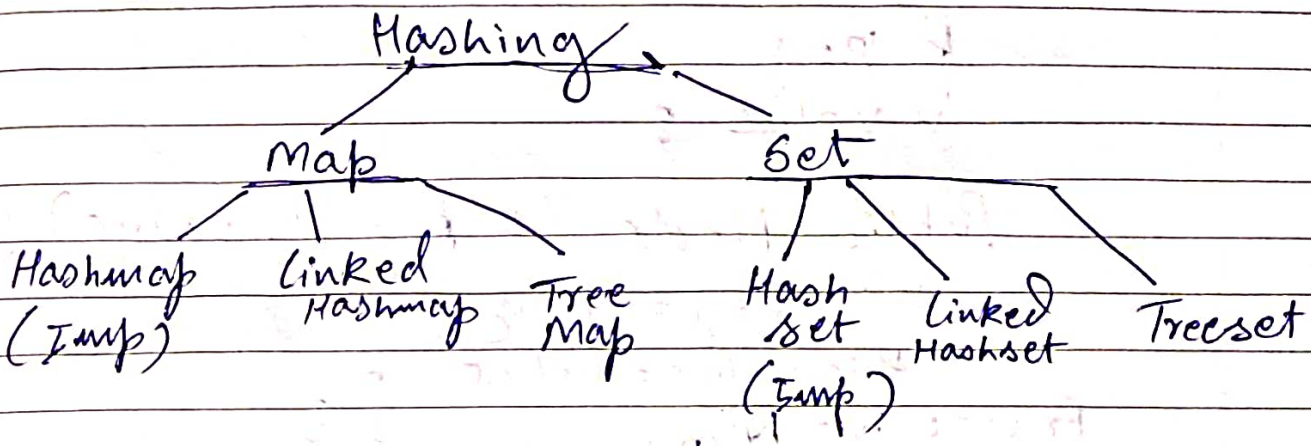
```
Public Int Compare to ( Pair P2) {
// descending      P2.value this.
    return ~~this.value~~ ~~P2~~.value
```

```
PSVM{ int arr[] = { 1,3,-1,-3,5,3,6,7}
  int k = 3; // window size
  int res[] = new int[arr.length - k + 1];
// ie n-k+1

PriorityQueue<Pair> pq = new priorityQueue<>()
// 1st window
  for (int i = 0 i < k i++){
    pq.add (new Pair (arr[i], i)); }
      res[0] = pq.peek().val;
    for (int i = k; i < arr.length i++){
      while (Pq.size() > 0 and pq.peek.idx
      <= (i-k){
                  { pq.remove }
      pq.add ( ~~arr[i]~~ new Pair[ arr[i],i)
      res [i-k+1] = pq.peek().val. }

// Print result
for (int i=0 i< result.length i++) {
    S.out (res[i] + " ").
```

* Hashing *

1234 → " $#XYZ "

Hash function



* Hashmap:- (key, value)
(unique)

HashMap <String, Integer> HM = new HashMap <>();

1. Put (key, value) — Hm.put("India", 100)
2. get (key) — Hm.get( )
3. Contains key (key) — HM.containsKey
4. remove (key) — HM.remove("India")
5. • Clear()

* Iteration on HashMap:-

Set < Integes > keys = hm.keySet();

* Implementation HashMap:-

```
Public class HashMap Code {
  Static class HashMap <K, V> {
    private class Node {
      K key
      V value ;
      Public Node (K key , V value) {
        this.key = key
        this.value = value
      } }
    Private int N;
    Private int ~~cap~~ n;
    Private LinkedList <Node> bucket [];

  ② Supress Warning ("unchecked")
    Public HashMap() {
      ~~Public void put (K key , V value)~~ {
        this. N = 4
        ~~this. n = 4~~
        this. bucket = new LinkedList [4]
        for (int i=0 i<4 i++) {
          this. buckets [i] = new linked list <> ()
        }

  ① Public void put (K key, V value) {
      int bi = hashfunction (key) ;
      (bucketIndex)
      int dataindex = SearchInLL (key, bi)
      if (dataindex != -1) {
        Node node = buckets [bi] . get (di) ;
        node. value = value
```

else { buckets[bi].add(new Node(key, value)); }
n ~~size~~ ++;
double ~~int~~ lambda = n/N }
if (lambda > 2.0) { rehash();

```
* Private int hashfunction (K Key) {
    int hc = key.hashCode();
return Math.abs(hc) % ~~size~~ ~~buckets.length~~ (N)

Private int SearchinLL (K key, int bi) {
  linkedlist<Node> LL = buckets[bi]

  for (int i=0  i<LL.size()  i++) {
      Node node = ll.get(i)
      if (node.key == Key) {
          return dataindex }
                              } dataindex ++ }
  return -1;

* Private void rehash() {

linkedlist<Node> oldbuck[] = buckets;
 buckets = new Linkedlist [N×2]
     N = 2×N
 for (int i=0  i<buckets.length  i++) {
     buckets[i] = new Linkedlist<>() }

// Nodes -> add in bucket
 for (int i=0  i<oldbucket.length ; i++) {
      Linkedlist<Node> ll = oldbuck[i];
  for (int j=0 j<ll.size(); j++) {
       Node node = ll.remove();
  put (node.key, node.value); } }
```

PSVM. {

HashMap<String, Integer> hm = new HashMap<>();

// Put values here

ArrayList<String> keys = hm.keySet();

for (String string : keys { S.out (keys)

}

Public boolean Containskey (K key) {

// Same code as put return true or false

}

Public V get (K key) {

// Same as (*) put

if (") {

return node.value}

else {null}

Public V remove (K key) {

int bi = hashfunction (key)

int di = Search in LL

// Same as put (n-- in if Statement)

* Public ArrayList<K> keySet () {

ArrayList<K> keys = new ArrayList<>()

for (int i = 0 i < buckets.length i++)

LinkedList<Node> ll = buckets[i]

for (Node node : ll) {

Keys.add (node.keys.) } return keys

2] * Linked HashMap :- Inorder Printing

LinkedHashMap < > HM = new LinkedHashMap<>();

3] * TreeMap :- Keys are Sorted (O(logn))

TreeMap < > HM = new TreeMap <>();

↓

Red Black trees → BST (Self Balancing)

[TC :- O(1)]

Q Given Integer array of size 'n' find all element that appear more than [n/3] times

nums[] = {1, 3, 2, 5, 1, 3, 1, 5, 1}

* Majority Elements :-

```
psvm {
 int arr[] = {1, 3, 2, 5, 1, 3, 1, 5, 1}
HashMap < Integer, Integer > map = new HashMap<>();
for(int i=0 i<arr.length i++){
    if (map.containsKey(arr[i]) {
      map.put(arr[i], map.get(arr[i]+1); }
  else{
      map.put (arr[i], 1);
  }
}
Set<Integer> keyset = map.keyset
for(Integer key: keyset) {
```

```
if (map.get(key) > arr.length/3) {
    S.out (key)}
```

* Valid Anagram :- { Same = Same, 1234 = 1234, 4 = 4 }

Hashmap<character, frequency> HM = new ... }

* Hash Set *

Properties → 1] No duplicates
2] Unordered
3] Null is allowed

1] Hashset is implemented using HashMap

Hashset<Integer> Set = new Hashset<>()

①	add (key)	set.add (1) - - -
②	contains (key)	set.contains (1) → Yes
③	remove (key)	set.remove (1) → True
④	Size ()	
⑤	Clear ()	
⑥	isEmpty ()	

* Iteration on HashSet :-

a) Using Iterator :- Iterator it = Set.iterator ()

① it.hasNext ()
② it.next ()

b] Using for loop:-

```
for (String city : cities) {
        S.out (city) }
```

2] * Linked HashSet :- Same as LHM

3] * TreeSet :- ① Sorted in ascending order
② Null values are not allowed

RedBlack Tree → TreeMap → TreeSet
(1) (2) (3)

TC : O(logn)

* Count Distinct Element

nums = 4, 3, 2, 5, 6, 7, 4, 2, 1

ans = 7

```
for (int i = 0 i < num.length; i++) {
    set.add (nums[i]) }
    S.out ( set.size() )
```

* Union and Intersection of two Array :-

* Tries *

* Also known as prefix tree / Retrieval tries

APPLE → APP

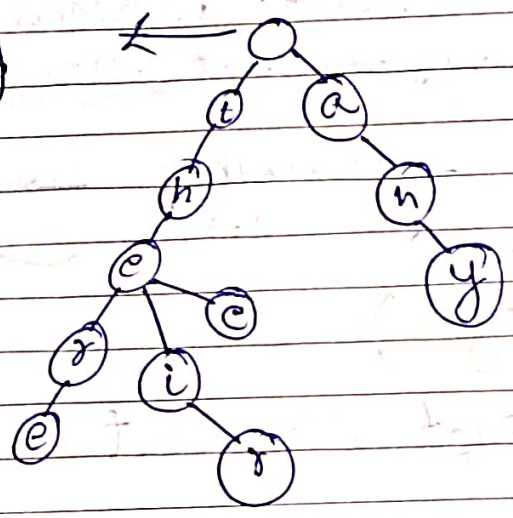
↓

Prefix

Q What is Trie? → K-array tree

Words [] = "the", "a", "there", "their", "any", "thee"

Empty




```
Class Node {
Node[] children ;
boolean endOfWord;
}
```

```
Static class Node {
Node children[]=new Node[26]
    boolean eow = false;

Node () {
    for (int i=0 i<26 i++) {
        children[i] = null; } }
Public Static Node root = new Node ();
```

* Insert in Trie :- (O(L)) → Word length

```
PS void insert (String word) {
    Node current = root;
    for (int level=0 level< word.length() level++)
    { int idx = word.charAT (level) - 'a';
      if (curr.children[idx] == null) {
          curr.children[idx] = new Node()
      }
      curr = curr.Children [idx]
    }
      curr.eow = true;

PSVM {
  String Words[] = { (*) }
  for (int i=0 i< words.length i++) {
        insert (words[i])
```

* Search in Trie :- (level wise Search in tree)

* Word Break Problem :- * (Imp)

Q Given an input String and a dictionary of word find out if input String can be broken into Space Separated Sequence of dictionary words.

* words[] = { i, like, sam, samsung, mobile, ic
key = "i like samsung"

⟹ output = true

* Step:- 1) Create trie for string
2)

// same code as before

PS boolean wordBreak (String key) {
for(int i=1; i<=key.length(); i++){
if(search (key.substring (0, i)) and
wordBreak(key.substring (i)) {
return true
else { return false }
if (key.length() == 0) {
return true
}

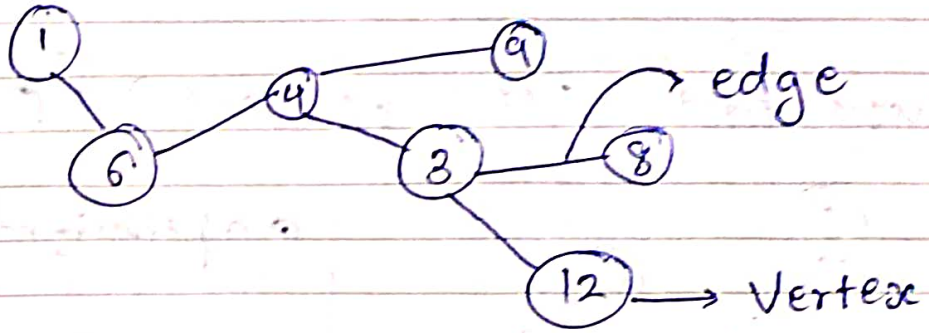
PSVM {
String arr[] = { * }
for(int i=0 i< arr.length i++){
insert (arr[i]);
} String key = "i like samsung"
S.out (wordBreak (key))

* Prefix Problem :- find shortest unique prefix for every word in given list

hint assume no word is prefix of another

* Graphs * (maps Google)

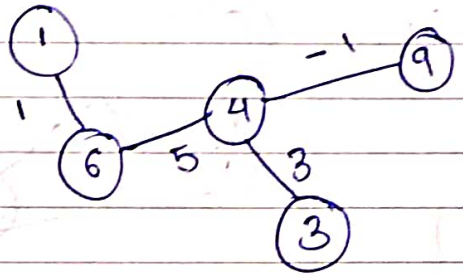
(1) Collection / Network of nodes.



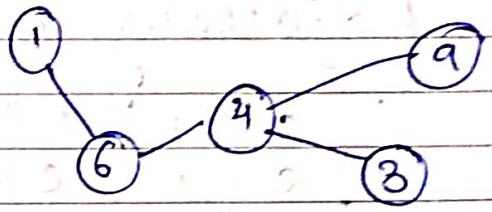
* Edges :- (1) Uni directional → one direction
(2) Un directional → Both direction
(3) Bi directional → go and come

* Types :-

Weighted



Un-Weighted



M T W T F S S

Page No.:

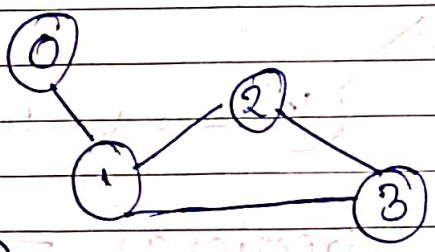
Date:

YOUVA

* Storing a Graph :- (types)

1. Adjacency List
2. Adjacency Matrix
3. Edge List
4. 2D Matric (Implicit Graph)

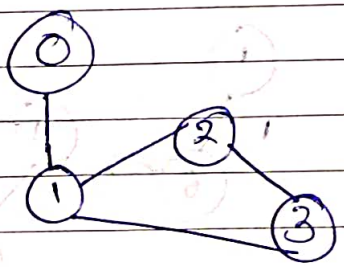
* i] Adjacency list (List of list) [O(1)]



dis[0] 0 → (1)
" [1] 1 → (0, 2, 3)
" [2] 2 → (1, 3)
" [3] 3 → (1, 2)

optimized

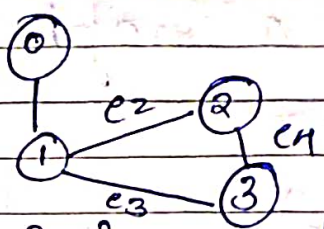
2] Adjacency Matrix :-



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

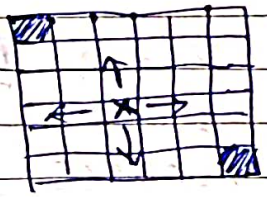
O (Vertex)

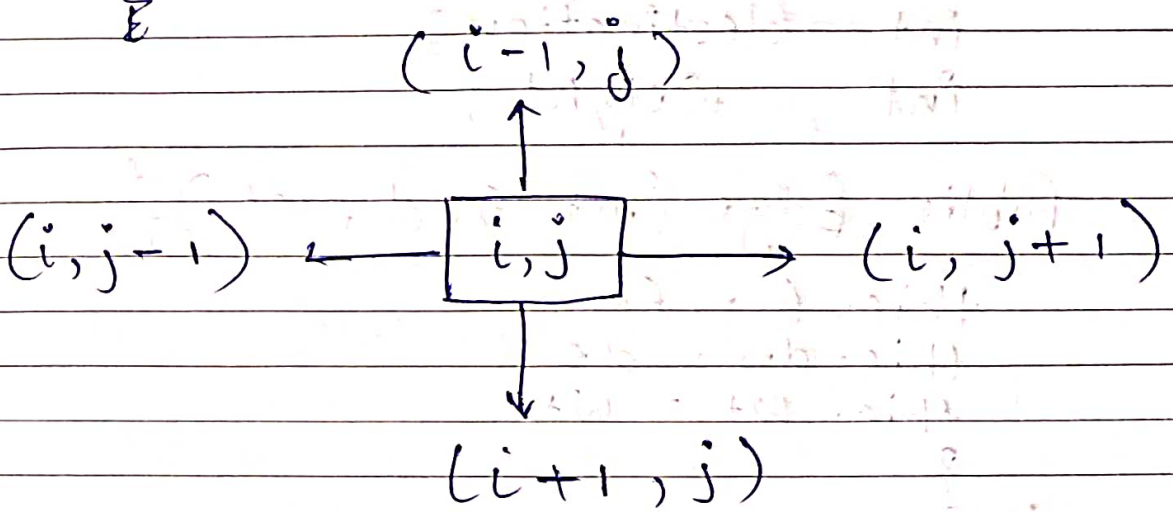
3] Edge list:-



Edges = { {0,1}, {1,2}, {1,3}, {2,3} }
{ e1 , e2 , e3 , e4 }

4] Implicit Graph

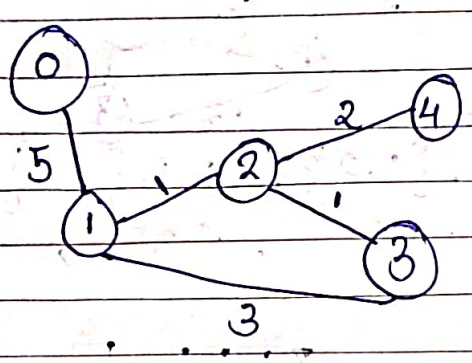




* Application *

a) map (Shortest path)
b) Social Network
c) DeliveryNetwork (cyclic Root)
d) Routing Algorithm.
e) Machine learning (computational Graph)
f) Computer Vision (Image Segmentation)

* Create a Graph :- (Array of Arraylist)




```
Static class Edge {
    int Source ;
    int destination;
    int Weight ;

Public Edge (int S, d, wt){
    this. S = Sou
    this. des = des
    this. wt = wt
    }
}

PSVM {
int Verteces = 5
```

* Arraylist <Edge>[] graph = new Arraylist[vert

```
for(int i=0 i<Verteces ; i++){
    graph[i] = new Arraylist<>(); }
```

[Vertex 0] → graph[0].add(new Edge(0, 1, 5));

```
//vertex 1
graph[1].add(new Edge(1,0,5);
              (1,2,1)
              (1,3,3)

// vertex 2
              (2,1,1)
              (2,4,2)
              (2,3,1)

// vertex 3
              (3,1,3)
              (3,2,1)

// vertex 4
              (4,2,2)

// for Vertex (2) neighbour
for(int i=0 i<graph[2].size() i++){
    Edge e = graph[2].get(i)
}
    S.out(e.dest)
```

* Graph Traversals *

1] BFS
↓
Go to immediate neighbour first (level wise)

2] DFS
↓

$O(V+E) \cong O(n)$

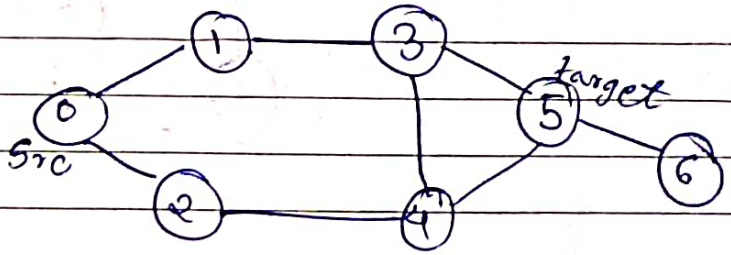
```
PS void BFS (ArrayList<Edge> [] graph) {
Queue < Integer > q = new Linked list <> ();
boolean visitedArr [] = new boolean [graph.length]
q.add (0)
while ( ! q.isEmpty()) {
    int curr = q.remove ();
    if (! vis [curr]) {
        S.out (curr)
        vis [curr] = true
    for (int i = 0 i < graph [curr].size (): i++
    Edge e = graph [curr]. get (i);
        q.add (e.dest);
    }
}
PSVM {
Arraylist < Edge > graph [] = new Arraylist [V]
create Graph (graph);
bfs (graph)
```

* DFS :- (O(V+E))

```
P S void Dfs (Arraylist<Edge> graph[], int curr,
                boolean vis[])

    S.out (curr);
    vis [curr] = true;
for (int i=0 i<graph [curr].size(); i++) {
   Edge e = graph [curr].get (i)
     dfs (graph, e.dest, vis)
     if ( vis [e.dest] == false )

-PSVM {
     int V=7

Arraylist<Edge> graph[] = new Arraylist[V]
  creategraph (graph)

  boolean vis[] = new boolean [v];
  dfs (graph, 0, vis)
    S.out Println ();
}}
```

* All Path from Src to Path Target

Src = 0 trgt = 5



// DFS code

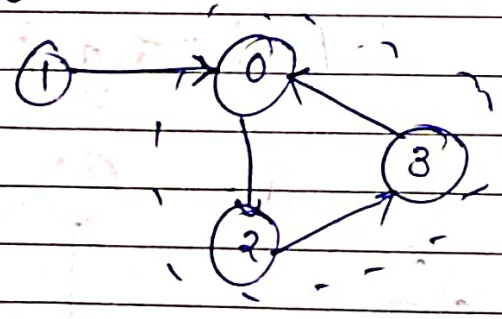
P 5 printallPath (Arraylist<Edge> graph, boolean vis[]
int curr, String path, int target)

```
if (curr = targ) {
        S.out (path);
        return
    }
for(int i=0 i<graph[curr].size(); i++){
    Edge e = graph [curr].get(i);
    if (!vis[e.dest]) {
        vis [curr] = true;
        print allpath (graph, vis, e.dest, path+e.dest, tar
        vis [curr] = false;
```

PSVM { int V = 7

```
Arraylist<Edge> graph[] = new Arraylist [V]
  Create graph (graph)
  int Src = 0   targ = 5
print All path (graph, new boolean[V], Src, 0, target)
```

* Cycle Detection :- in Graph



```
P S  boolean isCycledetected (ArrayList<Edge> graph[],
                boolean vis[], int curr, boolean rec[]) {

  vis[curr] = true
// recursion Stack (rec[curr] = true;

  for (int i=0 i<graph[curr].size() i++) {
      Edge e = graph[curr].get(i)
   if (rec[e.dest]) {
       return true; }
   else if (!vis[e.dest]) {
    if (isCycle detected (graph, vis, e.dest, rec) {
          return true; }
   }} rec[curr] = false;
      return false;

 PSVM { int V = 4
    S.out (isCycledetected (graph, new boolean [V], 0,
                new boolean [V]
```

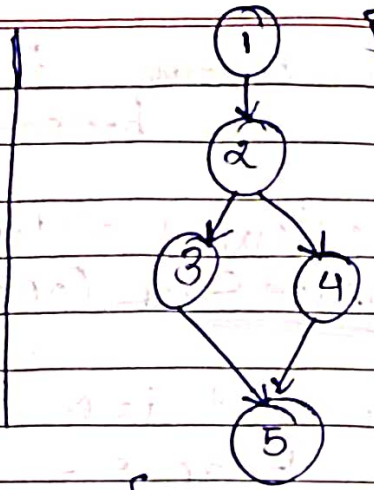
---

* Topological Sorting :- $O(V+E)$

* Properties :-) Directed Asyclic graph (DAG)

2) It is linear order of vertices such that every directed edge $u \rightarrow v$ the vertex 'u' comes before 'v' in order.

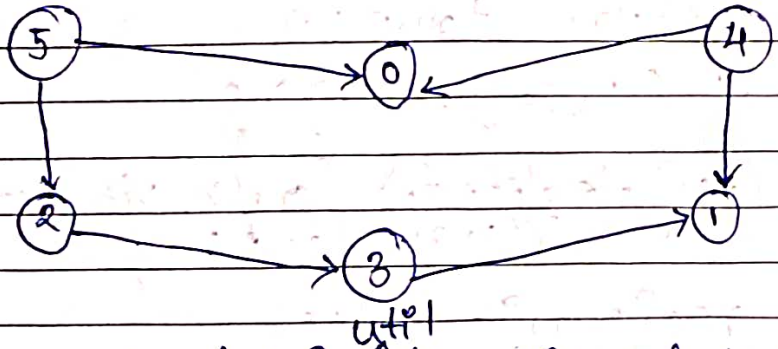
1) Buy laptop
2) Install OS
3) Intall codeeditor
4) Install Java
5) Write Code



DAG
Dependency

TO = [1, 2, 3, 4, 5]

Q



util

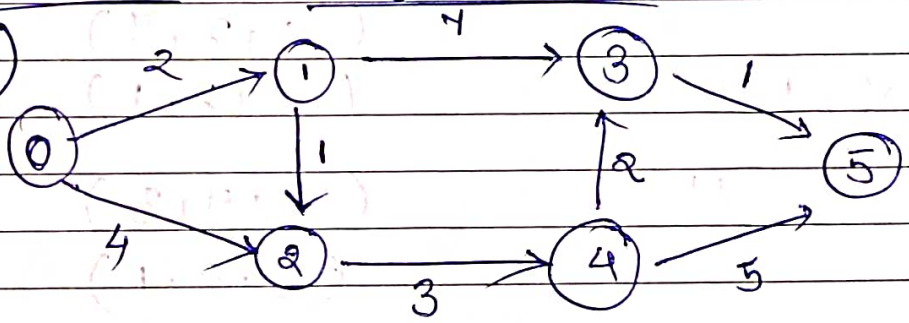
```
PS void topSort(ArrayList<Edge> graph[], int curr,
                boolean vis[], Stack<Integer stack){
for(int i=0 i<graph[curr].size() i++) {
    Edge e = graph[curr].get(i);
    if(!vis[e.dest]){
        topSortUtil(graph, e.dest, vis, Stack)
    }}
    Stack.push(curr)
}
PS void topSort(ArrayList<Edge> graph[]){
    boolean vis[] = new boolean[V]
Stack<Integer> stack = new Stack<>();
```

```
for (int i=0  i<V  i++) {
  if(!vis(i)) {
    topSortUtil(graph, i, vis, Stack)
  }
}
while(!Stack.isEmpty()) {
    S.out(Stack.pop());
psvm{ int V=6
AL<E> graph = new ...
  create graph(graph)
  topSort(graph, V)
}
```

---

* Cycle Detection (Undirected graph)

* Dijkstra Algorithm :- Shortest distance from source to all vertex (Greedy) Algo



(BFS)

Shortest dist

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

* Relaxation :- updation current weight to new low wt.

```
dis[u] + wt  <  dist[v]
dist[v] = dis[u] + wt
```

* Static class Edge {
  int src;
  int dis;
  int wt;

  Public Edge (int s, d, w) {
    this.src = s
    this.dest = d
    this.wt = w }

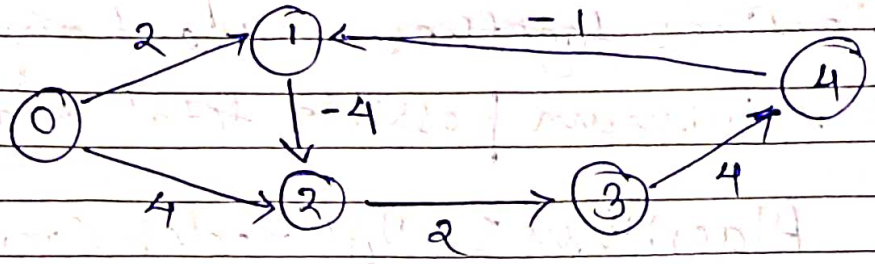
Public Static void CreateGraph (Arraylist<Edge>graph
  for(int i=0 i< graph.length i++) {
    graph[i] = new Arraylist<Edge> ();
  }
  graph[0].add(new Edge (0, 1, 2))
  " [0] " (0,2,4)
  " [1] " (1,3,7)
  " [1] " (1,2,1)
  " [2] " (2,4,3)
  " [3] " (3,5,1)
  " [4] " (4,3,2)
  " [4] " (4,5,5)

```
Public static class Pair implements Comparable<Pair>{
    int node;
    int dist;

  Public Pair (int n , d) {
     this.node = n
      this.dist = d
  }

  @ overide
    Public int Compareto (Pair p2){
          return this.dist - p2.dist // ascending
Not write { // descending p2.dist - this.dist } // dont write this line.

P S  void dijkstra (AL<E> grap [], int src, int V
]){  PriorityQueue<Pair> pq = new PriQue<>()
    int dis[] = new int [V]
    boolean vis[] = new boolean [V]
    // write this line Here
     for (int i=0  i<V  i++){
     if (i != src) {
       dist[i] = Integer.Max_value
     }
    pq.add (new Pair (0, 0))
```

// BFS

```
while (!pq.isEmpty()) {
Pair curr = pq.remove()
 if (!vis[curr.node]) {
    vis[curr.node] = true
 for (int  i=0   i < graph[curr.node].size()
    Edge e = graph[curr.node].get(i);
    int u = e.src
    int v = e.dist
 if (dist[u] + e.wt < dist[v]) {
    dist[v] = dist[u] + e.wt
    pq.add (new pair [v, dist[v]])
}}
}} for(int i=0  i<v  i++) {
    S.out (dist[i])
}
PSVM { int V=6
Arraylist<Edge> graph[] = new Arraylist[V]
 creategraph (graph)
 dijkstra (graph, 0, V)
```

* Bellman Ford Algorithm :- Shortest distance with negative wt.
[It does not work in -ve wt cycle]




```
PS void BellmanFord (AL<E> graph[], int Src, int V)
int dist[] = new int[V]
for(int i=0  i<V  i++) {
  if (i != Src) {
    dist[i] = Integer.Max_value
}
{ for(int k=0  k<V-1  k++) {
    for (int i=0  i<V  i++) {
      for (int j=0  j<graph[i].size()  i++) {
        Edge e = graph[i].get[j]
          int u = e.Src
          int v = e.dest
if (dist[u] != Integer.Max_value and
      dist[u]+e.wt < dist[v]) {
}}
  }} for (i=0  i<dis.length  i++) {
        S.out (dist[i])
PSVM { bellman ford (graph, 0, V)
```

* Minimum Spanning Tree :- (MST)

A (minimum ST) is subset of edge of connected edge weighted undirected graph that connects all the vertices together without any cycle and with minimum possible total edge weight

* Prims Algorithm :- 1] All nodes connected
2] No cycle
3] Cost Min (Wt min)

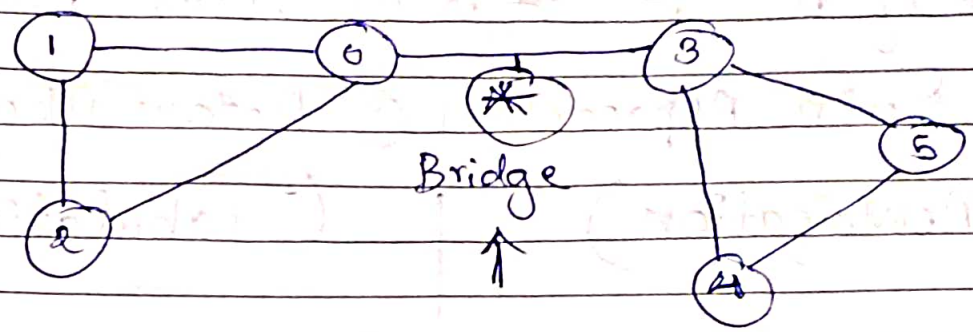
* Kosaraju's Algorithm :- Reverse DFS

* Strongly Connected Component : Only directed graph

* Steps :- 1) Get node in stack (topological order) sort
2) Transpose graph
3) Do DFS according to stack nodes on the transpose graph.

* Bridge in Graph :- Bridge is an edge whose deletion increases the graph's number of connected components

* Tarjans Algorithm 1] Discovery time [u] <[v]
2] lowest discovery time



* Articulation Point :- When edges are removed Sub vertices increases.

network vulnerability: To find

* Dynamic Programming *

1] It is used to find optimal Sol" for problem.
2] Break down complex problem into simple subproblem
3] find optimal Sol" to these Sub problem.
4] Store result of Sub problem. [In some Memory to reuse]
5] Reuse the Sub problem.
6] Calculate result of complex problem.

1 + 1 = 2

1 + 1 + 1
2 + 1 = 3

[Memoization]
[Overlapping SubPro
[Optimal Substruc

[v] Q Imp

1] Coin Change Problem :-

2]

* Ways of DP :- It is all about Pattern

① Top Down Approach (Memoization)	② Bottom up approach (Tabulation)

* Important Questions [In Google photos Study Album]