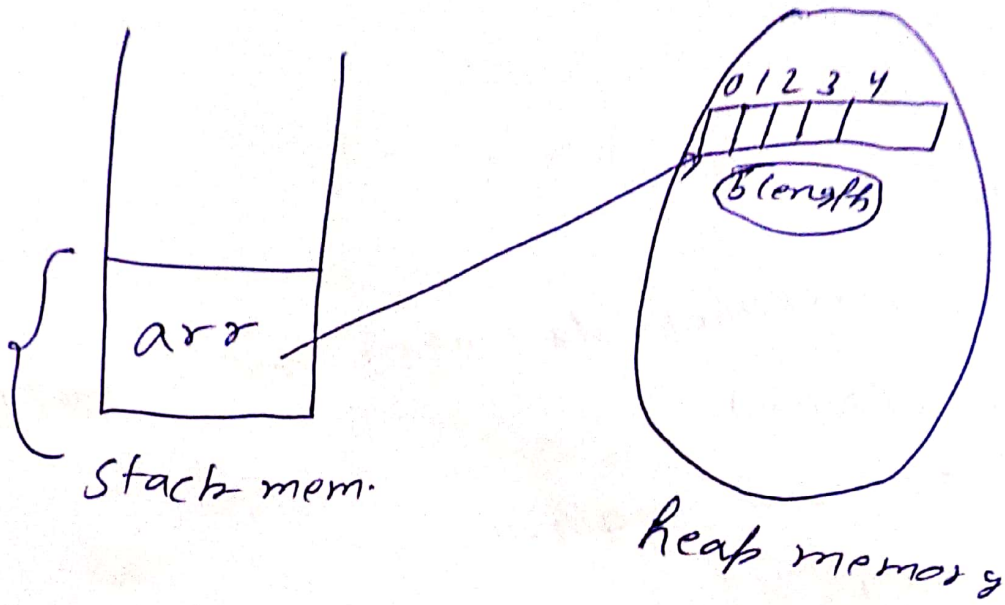
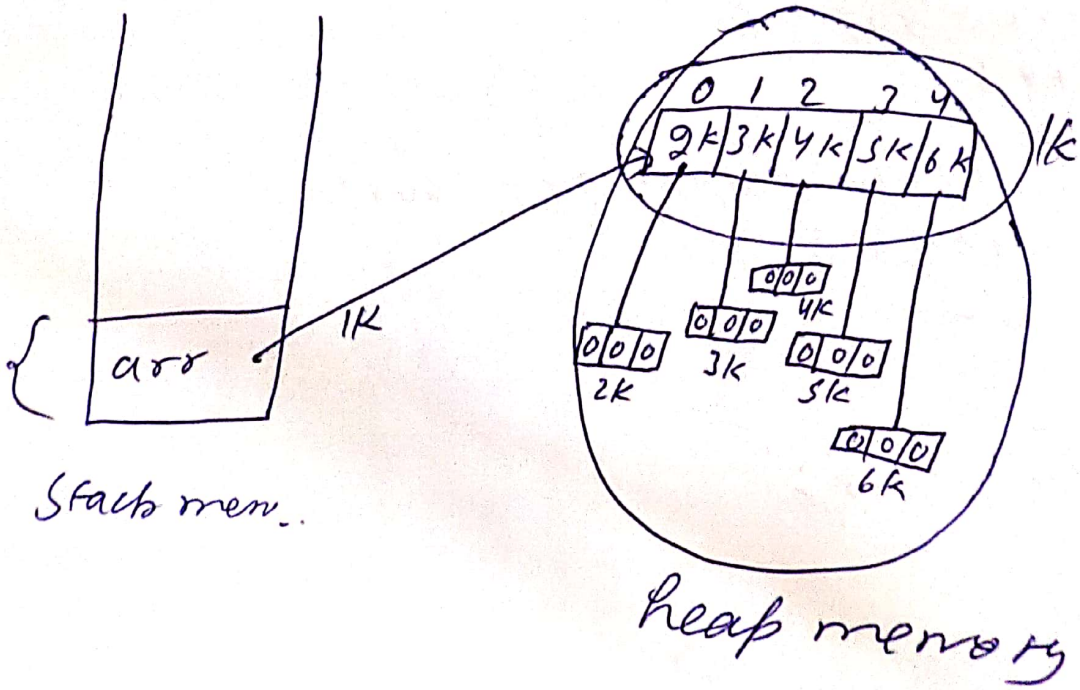
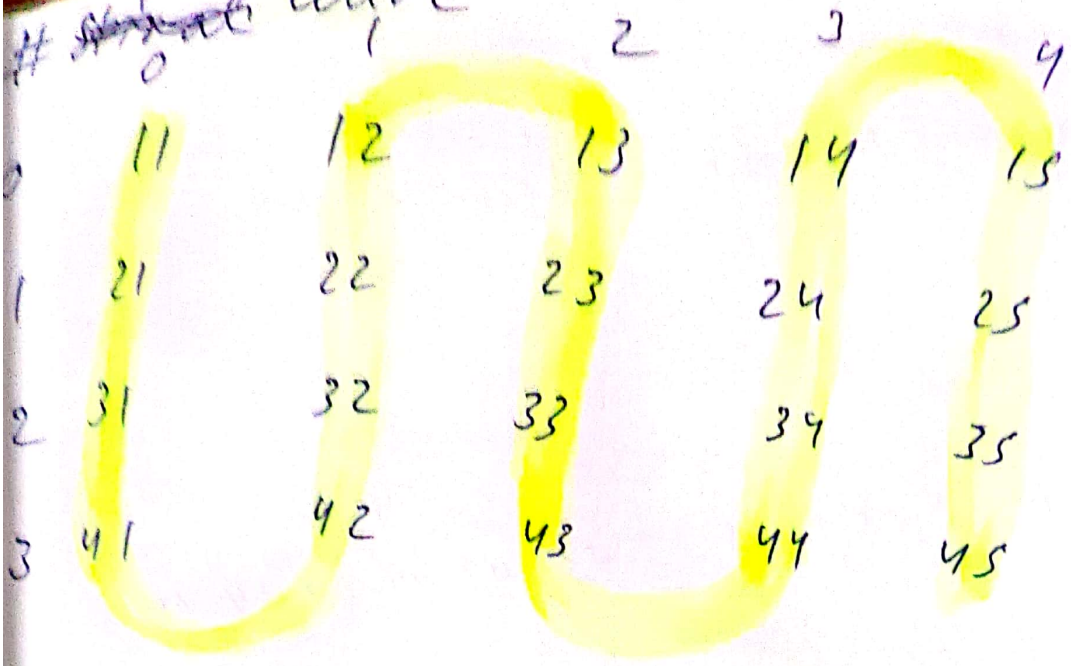
Memory in 2D Array





~~Matrix~~ wave



11. 21 31 41 42 32 22 12 13 23 33 43 44 ——

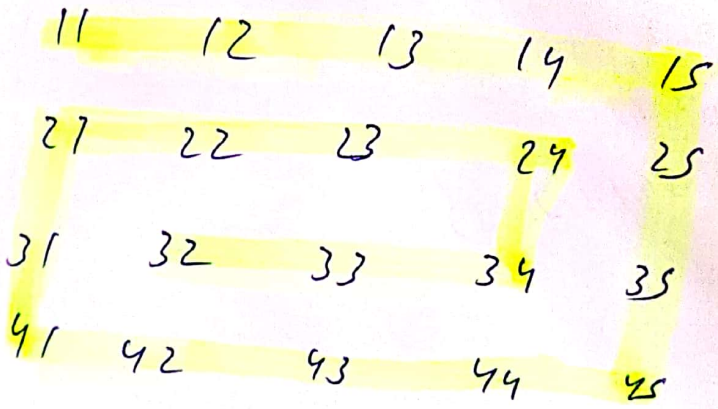
```
{
  int r = sc.nextInt();
  int c = sc.nextInt();
  int[][] arr = new int[r][c];
    input(arr);
    display(arr);
}
static void input(int[][] arr){
    for(  ( (
          ( (
  }
static void display(int[][] arr){
    for(int c=0;c<arr[0].length;c++){
    if(c%2==0){
      for(int r=0;r<arr.length;r++){
        System.out.print(arr[r][c]+" ");
      }
    }else{
```

```
for(int r=arr.length-1;r>=0;r--){
    System.out.print(arr[r][c]+" ");
  }
 }
}
}
```

O/P

11 21 31 41 42 32 22 12 13 23 33 43 44 34 24 14 15 25 35 45

Spiral



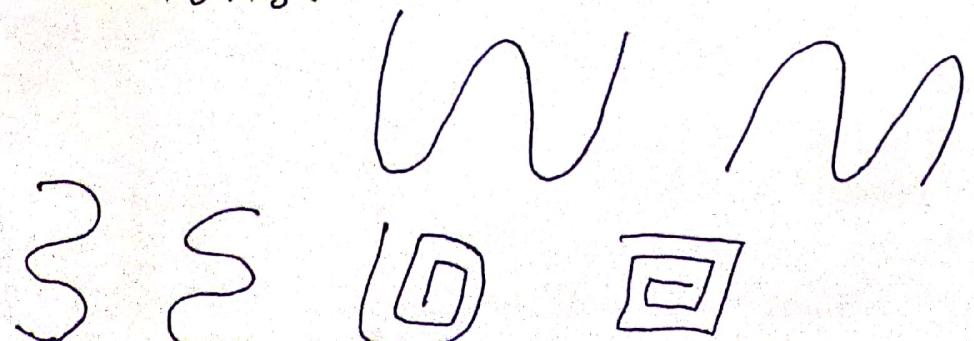

```
static void display(int[][] arr){
  int rmin=0;
  int rmax=arr.length-1;
  int cmin=0;
  int cmax=arr[0].length-1;
  int toe=arr.length*arr[0].length;

  while(toe!=0){
    // top wall

    for(int i=cmin;i<=cmax;i++){
```

```
System.out.print(arr[rmin][i]+" ");
  toe--;
}
rmin++;
// right wall
for(int i=rmin; i<=rmax && toe!=0; i++){
 System.out.print(arr[i][cmax]+" ");
   toe--;
}
cmax--;
// bottom wall
for(int i=cmax; i>=cmin && toe!=0; i--){
  System.out.print(arr[rmax][i]+" ");
   toe--;
}
rmax--;
// left wall
for(int i=rmax; i>=rmin && toe!=0; i--){
  System.out.print(arr[i][cmin]+" ");
   toe--;
}
cmin++;
}
}
```

O/P

11 12 13 14 15 25 35 45 44 43 42 41 31 21 ...

Variations:-



Array multiplication :-

```
Public Class mulArr {
  Static Scanner sc=new Scanner(System.in);
  Public Static void main(String[] args){
      int r1=sc.nextInt();
      int c1=sc.nextInt();

      int[][] arr= new int[r1][c1];
      input(arr);

      int r2=sc.nextInt();
      int c2=sc.nextInt();
      int[][] brr=new int[r2][c2];
      input(brr);

      if(c1!=r2){
        System.out.println("multip.. not Possible");
        return;
      }
    int[][] ans=new int[r1][r2];

    for(int i=0;i<r1;i++){
      for(int j=0;j<c2;j++){
        int sum=0;
        for(int k=0;k<c1;k++){
            sum+=arr[i][k]*brr[k][j];
        }
        ans[i][j]=sum;
      }
    }
    display(ans);
  }
```

```
static void input(int[][] arr){
        //
        //
}
static void display(int[][] arr){
        //
        //
}
}
```

o/p

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

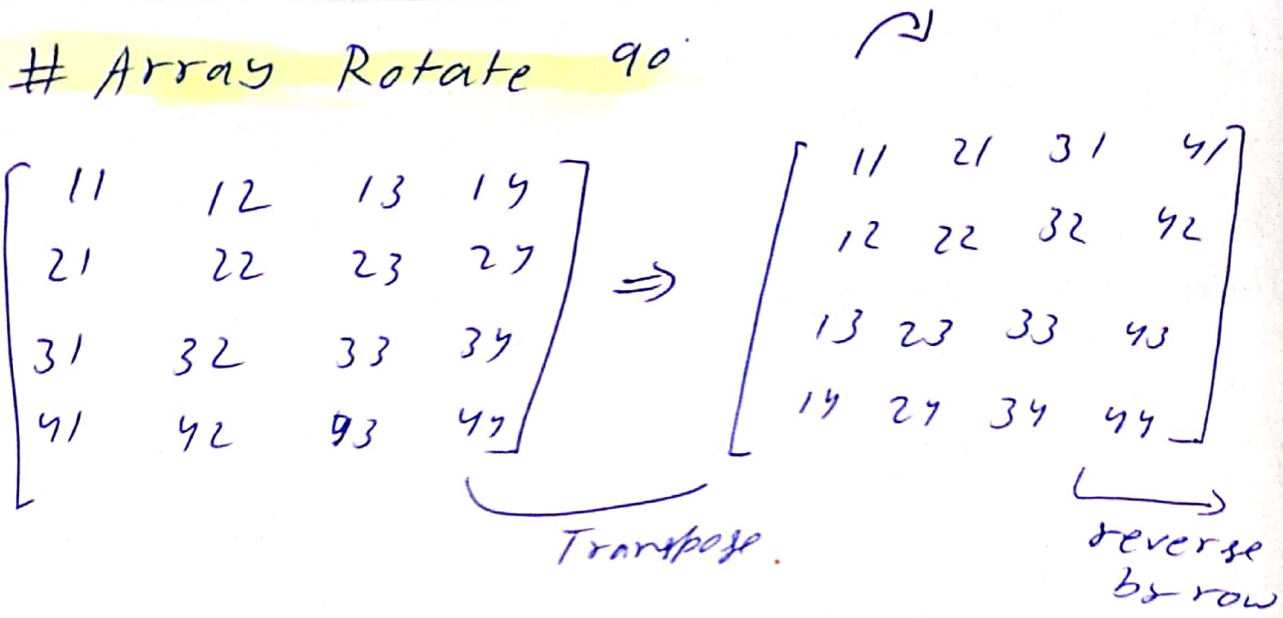
o/p

```
3 3 6
3 3 6
3 3 6
```

// Array Addition

```
if(r1!=r2) || c1!=c2){
    Sout("Addition not possible");
    return;
}
int[][] ans = new int[r1][c1];
for(int i=0; i<r1; i++){
    for(int j=0; j<c1; j++){
        ans[i][j] = arr[i][j] + brr[i][j];
    }
}
display(ans);
}
```

Array Rotate 90°

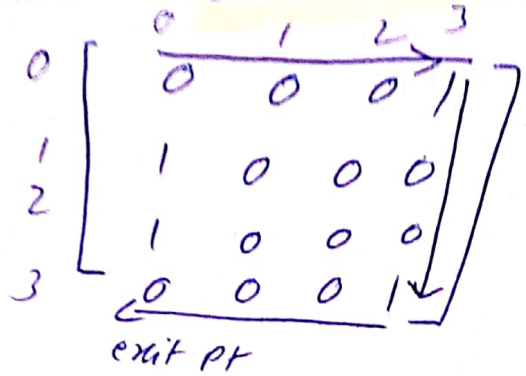


41 31 21 11
42 32 22 12
43 33 23 13
44 34 24 14

```
Public static void main (String[] args){
    int r1 = sc.nextInt();
    int c1 = sc.nextInt();
    int[][] arr = new int[r1][c1];
    input(arr);
    rotate(arr);
}
static void rotate(int[][] arr){
    int r = arr.length;
    int c = arr[0].length;
    // transpose
    for(int i=0; i<r; i++){
        for(int j=0; j<c; j++){
            int temp = arr[i][j];
            arr[i][j] = arr[j][i];
            arr[j][i] = temp;
        }
    }
```

```
        // reverse
        for(int i=0; i<arr.length; i++){
            reverse(arr, i);
        }
        display(arr);
    }
    static void reverse(int[][] arr, int r){
        int i=0;
        int j= arr[0].length-1;
        while(i<j){
            int temp = arr[r][i];
            arr[r][i] =arr[r][j];
            arr[r][j] = temp;
            i++;
            j--;
        }
    }
    static void input(int[][] arr){
            //
    }
    static void display(int[][] arr){
            //
    }
}
```

Exit Point.




```
Public static void main (String[] args) {
    int r1 = sc.nextInt();
    int c1 = sc.nextInt();
    int[][] arr = new int[r1][c1];
    input (arr);
    exitPoint (arr);
}
static void exitPoint (int[][] arr) {
    int d = 0;
    int i = 0;
    int j = 0;

    while (true) {
     d += arr[i][j];
     d %= 4;
     if (d == 0) j++;
    else if (d == 1) i++;
    else if (d == 2) j--;
    else i--;

    if (i < 0) {
      i++;
      System.out.println(i + " " + j);
      return;
```

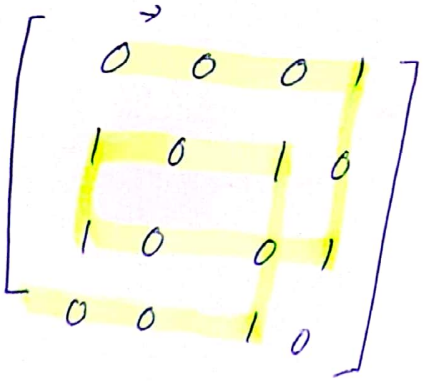
```
}else if( i>=arr.length){
    i--;
    System.out.println(i+"_"+j);
    return;
}else if(j<0){
    j++;
    System.out.println(i+"_"+j);
    return;
}else if(j>=arr[0].length){
    j--;
    System.out.println(i+"_"+j);
    return;
}
}
}
```

O/P ⇒ 3-0

input ?

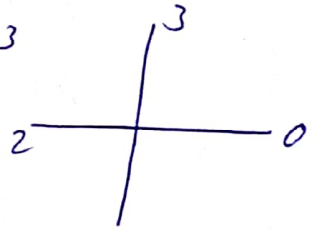
~~display ?~~

```
}
```



i = x x 3

j = 0 1 2



d = 0 j++

d = 1 i++

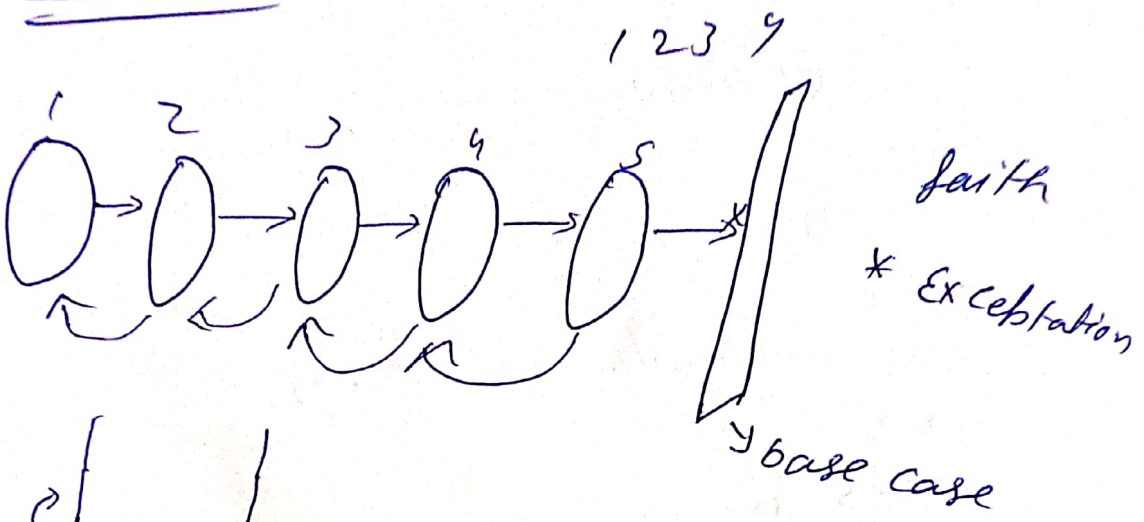
d = 2 j--

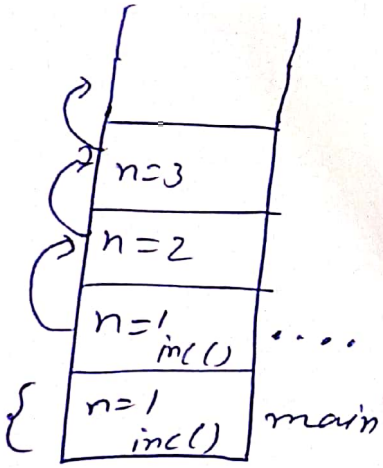
d = 3 i--

0 ⟶ 4

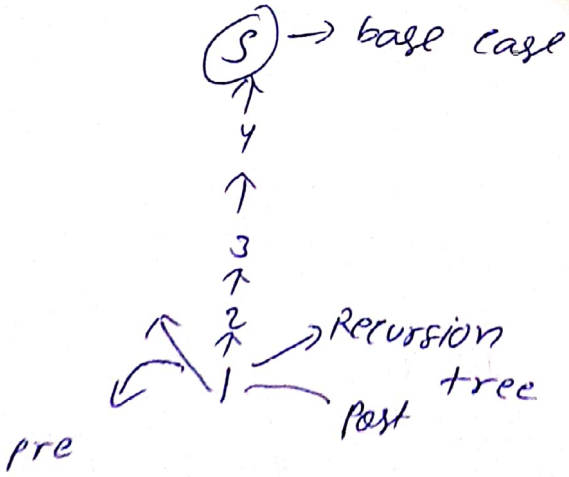
1 ⟶ 5

Recursion. → function call itself





1
2
3
4
5



log n → half of half

Basic Recursion.

```
Public class basicRec {
    Public static void main (String[] args) {
        int n=1;
        inc(n);                                  1
    }                                            2
    static void inc(int n){                      3
        if (n==5){                               4
            System.out.println(n);               5
            return;                              4
        }                                        3
        System.out.println(n);  // Pre work      2
        inc(n+1);                                1
        System.out.println(n); // Post work
    }
}
```

Factorial

```
Public class factorial {
    public static void main(String[] args){
        int n=5;
        System.out.println(fact(n));
    }
    static int fact(int n){
        if(n==0) return 1;
        return n*fact(n-1);          ⇒ 120
    }
```

Q

```
        1
      1 - 2
    1 - - - 3
  1 - - - - - 4
  1 2 3 4 5
```

Q Power of any number

```
Public Class anyPower {
    Public static void main (String[] args) {
        int P=5;
        int n = 4;
        System.out.println (Pow(P, n));
    }
    Static int Pow(int P , int n) {
        If (P==0) return 1;
        int ans = Pow (P/2, n);
        If (P%2==0) {
            return ans x ans;
        }
        return n*ans *ans;
    }
}
```

⇒ 1024

e.g

3x3x3x3x3x3

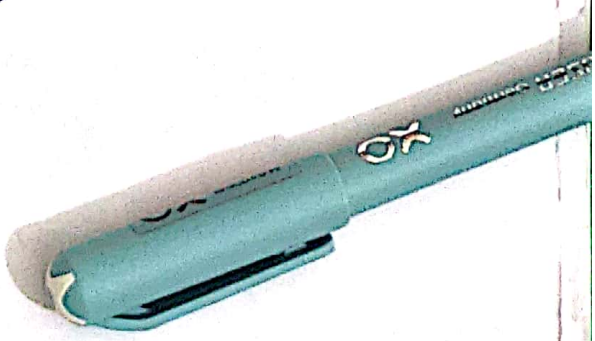
$3 \times 3^2 \times 3 \times 3^2$

n(log n)

Q 1 2 3 4 5

5 4 3 2 1

```
Public class fun {
    Public static void main(String[] args) {
        int n=1;
        Prt(n);
    }
    Public static void Prt(int n){
        if(n>5){
            System.out.println();
            return;
        }
        System.out.print(n+" ");
        Prt(n+1);
        System.out.print(n+" ");
    }
}
```



Buffer Input stream Reader

```
InputStreamReader is = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(is);
int n = Integer.parseInt(br.readLine());
```

Q index of no in array from starting using recursion.

```
public class indx {
    public static void main(String [] args) {
    int arr = {1,2,3,4,5,6,7,5,5,2}
    int idx = 0;
    int temp = 5;
    system.out.print(ind(arr, idx, temp));
}
    public static int ind(int[] arr, int idx, int temp) {
        if(idx == arr.length) return -1;
        if (arr[idx] == temp) return idx;
        return ind(arr, idx+1, temp);
    }
}
```

o/p 4

Q index from back side using recursion.

```
public class indx1 {
    public static void main (String[] args) {
        int arr[] = {1,2,3,4,5,6,4,5,5,7,5}
        int idx = 0;
        int temp = 5;
        int i = -1;
        system.out.print(ind(arr, idx, temp));
    }
    Public static int ind(int[] arr, int idx, int temp){
        int i = 0;
        if (idx == arr.length) return -1;
```

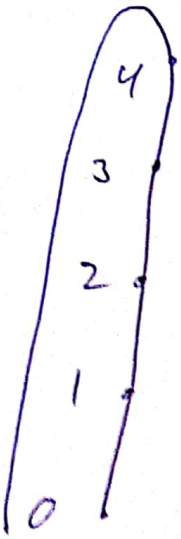
```
i = ind(arr, idx+1, temp);
if(arr[idx] == temp && i<idx){
    i = idx;
}
    return i;
}
}
```

o/p 10

Max - no in array using post order

```
public class MaxArr{
    public static void main(String[] args){
        int arr[] = {10, 2, 3, 9, 5};
        int idx = 0;
        System.out.print(max(arr, idx));
    }
    public static int max(int[] arr, int idx){
        if(idx == arr.length-1){
            return arr[idx];
        }
        int reans = max(arr, idx+1);
        if(arr[idx] > reans){
            return arr[idx];
        }
        return reans;
    }
}
```

<for min



o/p 10

reans → 9 10

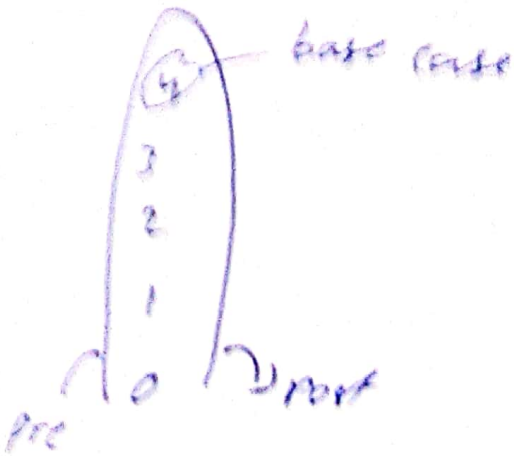
arr.leng → 5

Q. ~~No~~ No of occurrence of no ~~is present at index.~~

```
public class nooce{
    public static void main(String[] args){
    int arr = {1, 2, 3, 5, 6, 4, 5, 8, 7, 5};
     int idx = 0;
    ~~indx = 0;~~
    int temp = 5;
    System.out.print(max_occ(arr, idx, temp));
    }
    public static int max_occ(int[] arr, int idx, int temp){
        if(idx == arr.length) return 0;
          int i = 0;
          i = max_occ(arr, idx+1, temp);
          if(arr[idx] = temp){
             i++;
          }
          return i;
    }
}
```

print Array elements

```
{
  prt(arr, 0);
}
public static void prt(int[] arr, int n){
    if(n == arr.length) return;
    System.out.print(arr[n]);
    prt(arr, n+1);
}
```



Q PMI → Principle of mathematical Induction

Sum of $n = \frac{n(n+1)}{2}$

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

$0 = \frac{0(0+1)}{2} \Rightarrow 0$

$1 = \frac{1(1+1)}{2} \Rightarrow 1$

LHS = RHS

① Proof P(0), P(1) P(2) If true

P(k-1), P(k) P(k+1)

② Induction Hypothesis (assume)

$P(k) = \Sigma k = \frac{k(k+1)}{2}$

③ Prove P(k+1)

$\Rightarrow \Sigma(k+1) = \frac{(k+1)(k+2)}{2}$

$\Sigma(k) + \Sigma(k+1) = \frac{(k+1)(k+2)}{2}$

$\frac{k(k+1)}{2} + (k+1)$

$\frac{k(k+1) + 2(k+1)}{2}$

$\frac{(k+1)(k+2)}{2} \Rightarrow$ RHS

No of ...

Q. void fun(int n) {

1 if(n==0) {return;}

2 System.out("A"); // Pre

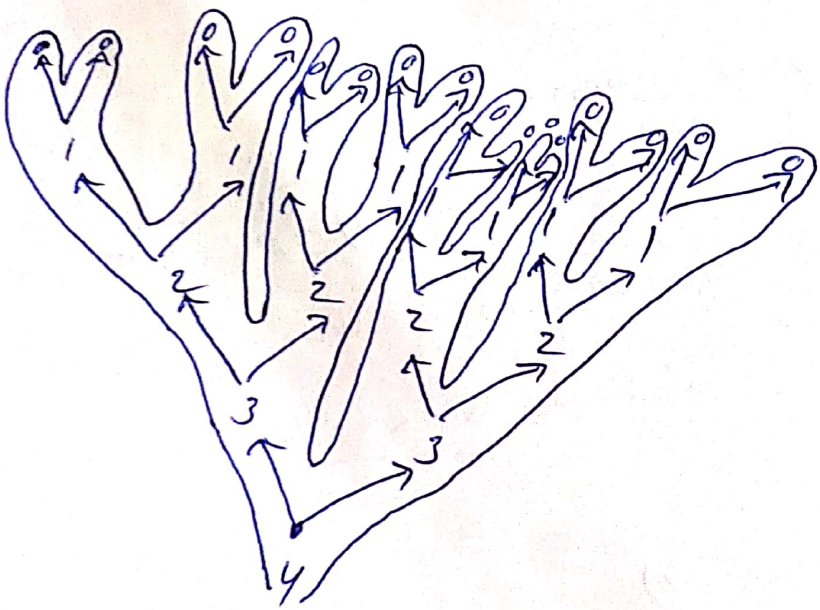
3 fun(n-1);

4 Sout("B");] In

5. fun(n-1);

6 Sysо("C");] Post

}



A A A B C B A B C C B A A B C B A B C C C B A A A ...

Str.length() → String length

String str = new String("...");

Str.charAt(idx);

Str.substring(idx);

[s, e)

↑ include

↳ not included in Java

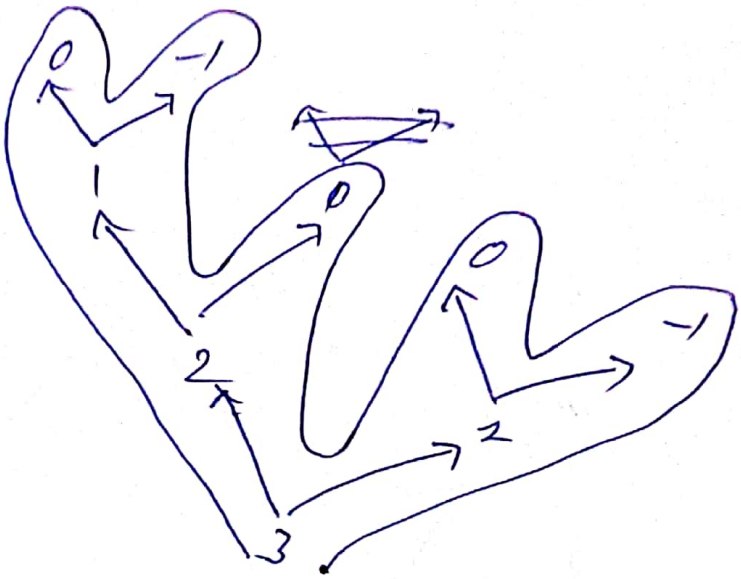
immutable string → content of string obj can't be change

mutable → StringBuffer & StringBuilder are mutable classes

CodingBat → site for practice question on topics.

Q. void fun (int n)

```
if(n<=0) return;
sysO("A_");  ] Pre
fun(n-1);
sysO("B");  ] In
fun(n-2);
sysO("C");  ] POST.
}
```



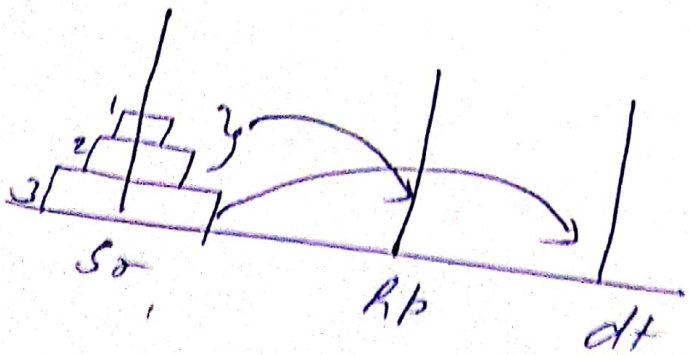
A A A B C B A B C C B A B C C

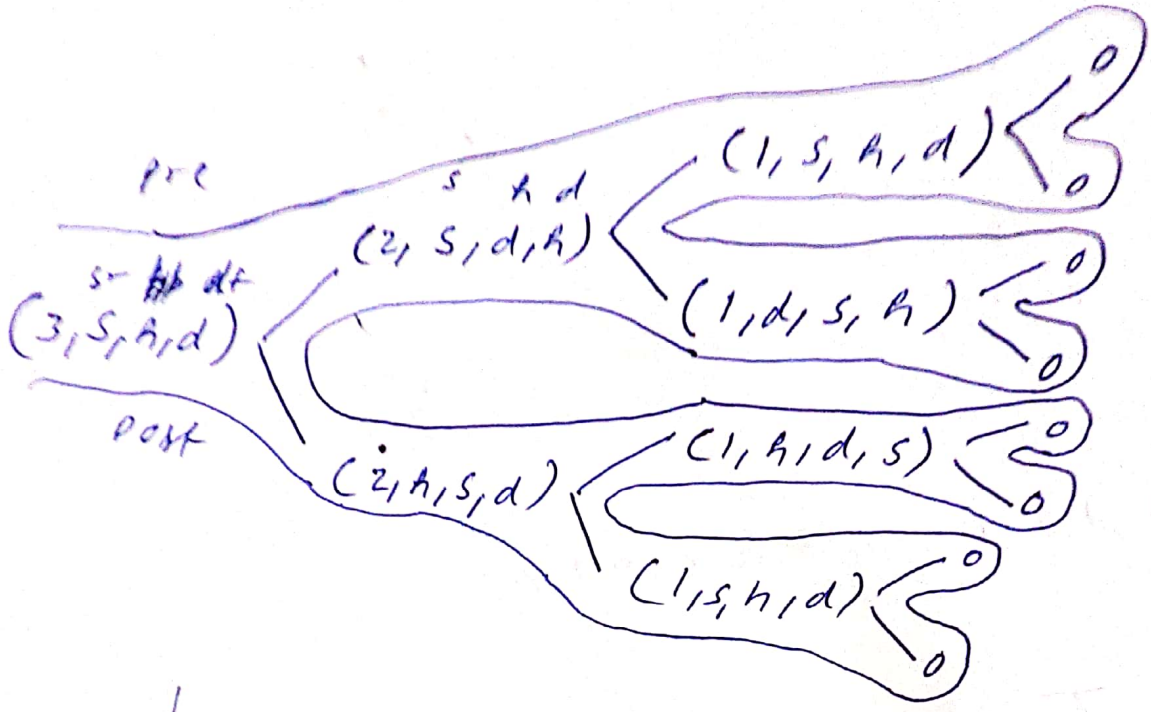
Q. Tower of Hanoi

Condition

1. one disk can pick at one time
2. never put grater disk on small disk
3.

toh (int n, char ch, char s, char d)






```
Public class toh {
    Public static void main (String [] args) {
    int n = 3;
    char sr = 's';
    char hp = 'h';
    char dt = 'd';
    toh_solve (n, sr, hp, dt);
}
public static void toh_solve (int n, char sr, char hp, cha
    dt) {
    if (n == 0) return;
    toh_solve(n-1, sr, dt, hp);
    System.out.println(n + "__" + sr + " => " + dt);
    toh_solve (n-1, hp, sr, dt);
}

}
```

Str.length() → String length.

String Str = new String("...");

Ch = Str.charAt(idx);

Str.substring(idx);

[s, e)

↑ include in java

↳ not included in java

immutable string → content of string obj can't be change

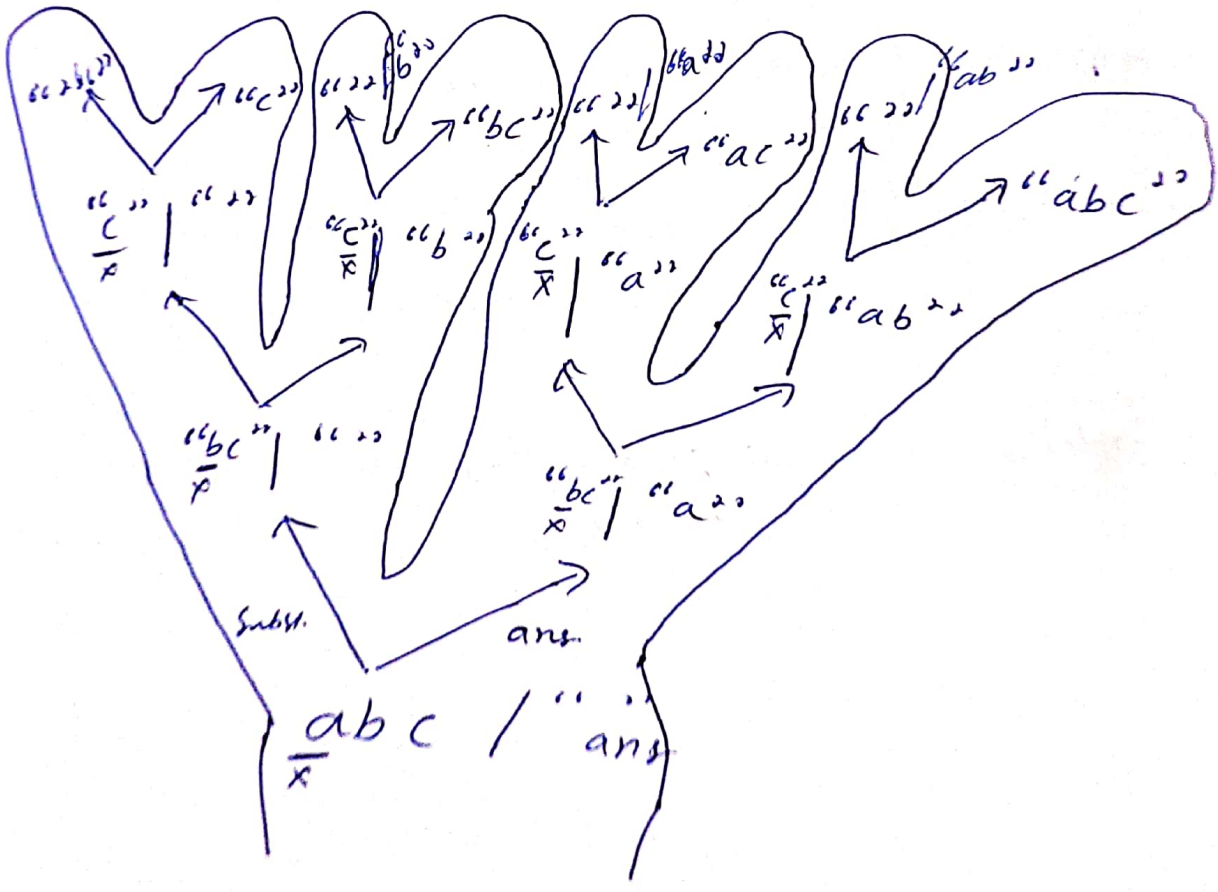
mutable → StringBuffer & StringBuilder are mutable classes

Codingbat → site for practice question on topics.

Q. Print Substring // Pre order approach.

abc = { "", "a", "ab", "ac", "b", "bc", "c", "abc" }

String → non primitive
→ Immutable



```
public class substr {
    public static void main (String[] args) {
        String str = "abc";
        PrintSub(str, "");
    }
    public static void printSub (String str, String ans){
        if(str.length() == 0) {
            if(ans.length() == 0){
                System.out.println("_");
                return;
            }
            System.out.println(ans);
            return;
        }
        char ch = str.charAt(0);
        String ros = str.substring(1);

        PrintSub(ros, ans);
        PrintSub(ros, ans+ch);
    }
}
```

O/P

_
c
b
bc
a
ac
ab
abc

Search in 2D Array

```
import java.util.Scanner;
public class search{
    Static Scanner sc=new Scanner(System.in);
    public static void main(String[] args){
        int r1=sc.nextInt();
        int c1=sc.nextInt();
        int[][] arr=new int[r1][c1];
        input(arr);
        int num=32;
        search(arr, num);
    }
    Static void search(int[][] arr, int num){
        int i=0;
        int j=arr.length-1;
        boolean
        while(true){
        if(arr[i][j]==num){
            System.out.print(i+" "+j);
            return;
        }
        if(arr[i][j]<num) i++;
        if(i==arr.length){
            Syso("elemd not found");
            return;
        }
        if(arr[i][j]>num) j--;
        if(j==-1){
            syso("element not found");
            return;
        }
    }
}
```

11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

no occurrence and its indexes.

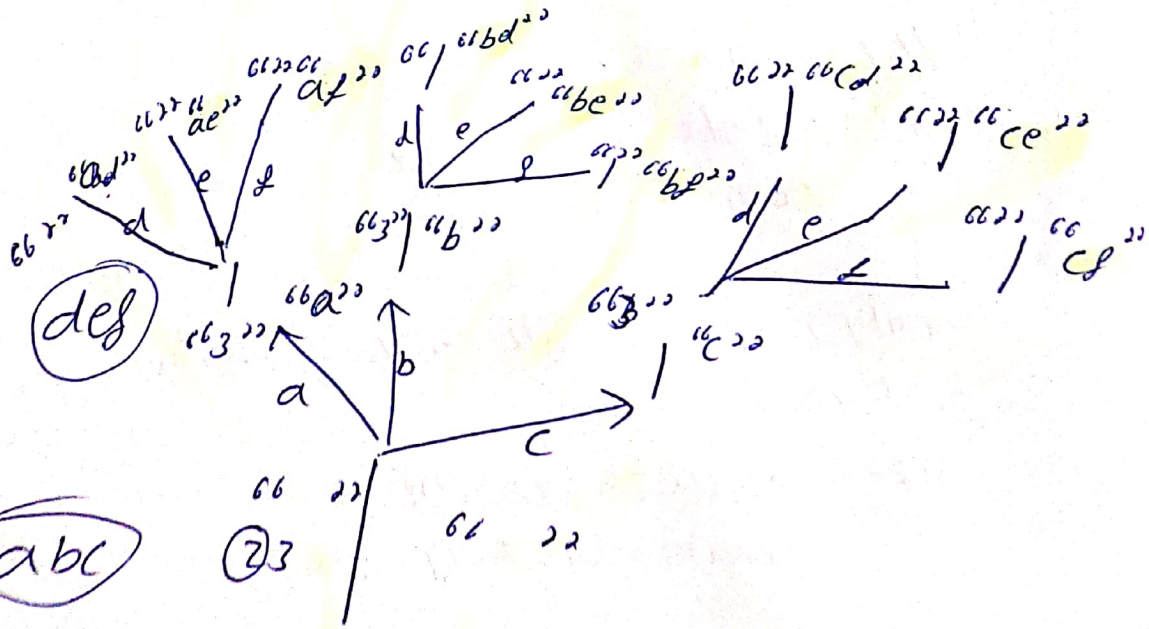
```
Public class allindex{
    Public static void main(String[]args)
        {
            int[] arr= {1, 2,2,3,5,2};
            int num =2;
            int [] ans= alloccindex(arr, 0,0, num);

            for(int i: ans) {  // for each loop
                System. out. print (i +" ");
            }
    }
    public static int[] alloccindex(int[] arr, int idx, int
        count, int num) {
        if(idx == arr.length) {
            int[] base =new int [count];
            return base;
        }
        // pre area
        if (arr[idx ] == num) {
            count++;
        }
        int[] recarr = alloccindex (arr, idx+1, count, num);
        // Post area
        if(arr[idx ] == num){
            recarr[count -1] = idx;
        }
        return recarr;
    }
}
```

Dial pad . Java

1	2	3
•	abc	def
4	5	6
ghi	jkl	mno
7	8	9
pqrs	tuv	wxyz
*	0	#
x	#	x

ASCII

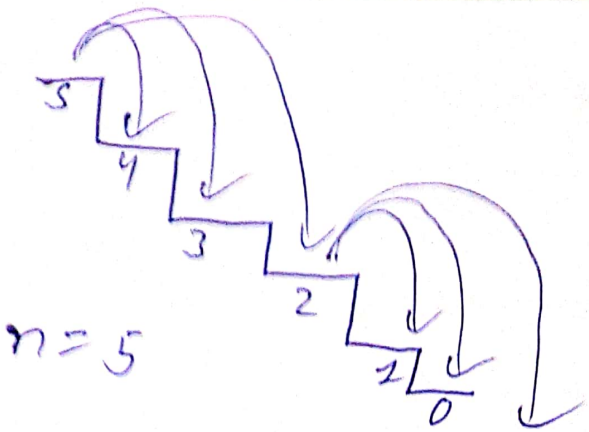
$$\begin{cases} 48 \leftrightarrow 58 \rightarrow 0-9 \\ 97 \leftrightarrow 122 \rightarrow a-z \\ 65 - 90 \rightarrow A-Z \end{cases}$$



["#", ".", "abc", ...]

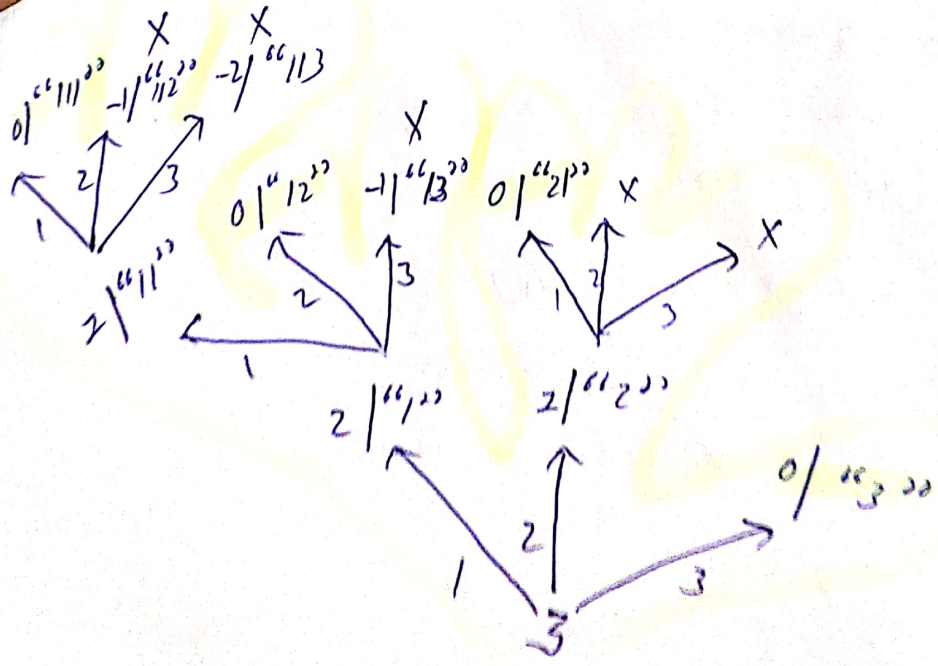
0 1 2

```
public class dialpad{
    Static String[] keys = {"#", ".", "abc", "
    def", "ghi", "jkl", "mno", "pqrs", "Tuv",
    "wxyz"};
    public static void main(String[] args){
        string type = "23";
        kpc(type, "");
    }
    static void kpc(String str, String ans){
        if(str.length() == 0){
            System.out.println(ans);
            return;
        }
        Char ch = str.charAt(0);
        String ros = str.substring(1);
        int idx = ch - 48;
        String press = keys[idx];
        for(int i=0; i< press.length(); i++){
            kpc(ros, ans+press.charAt(i));
        }
    }
}
```



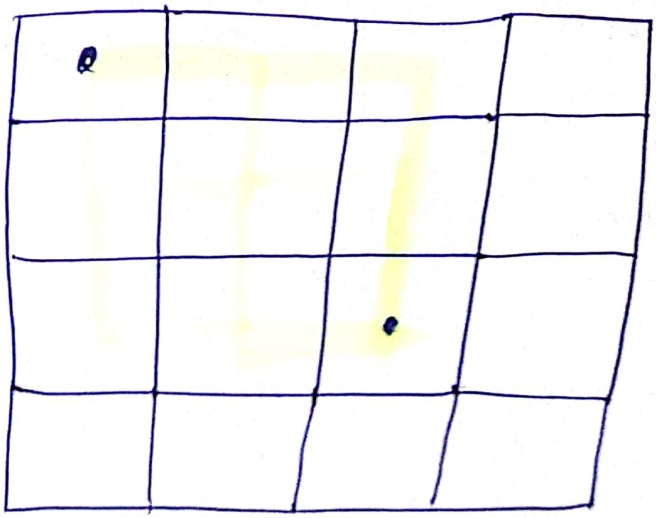
$n = 5$

$n = 3$

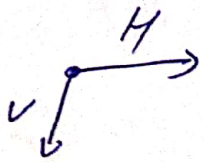


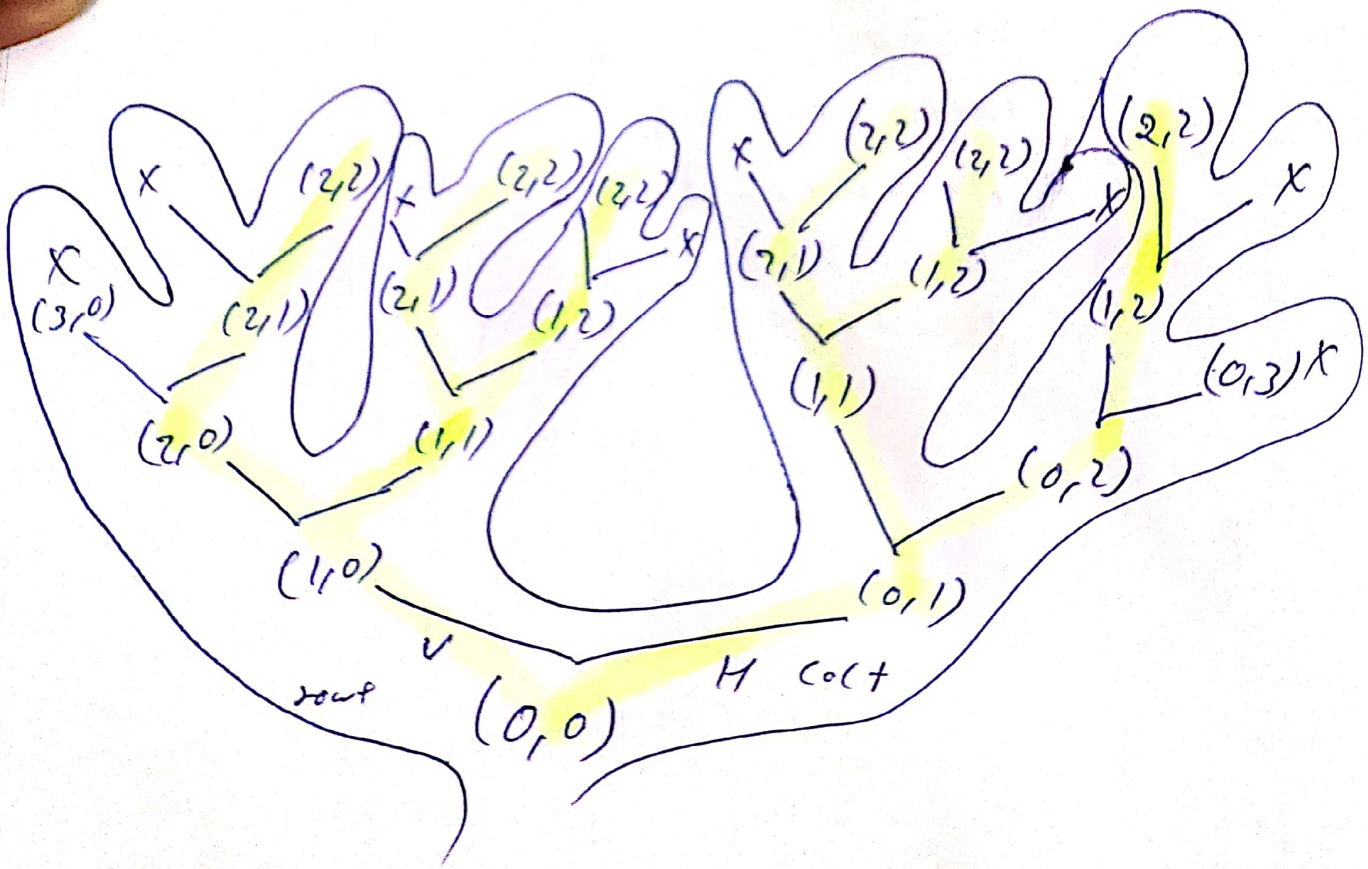
```
public class stairs{
    public static void main(String[] args){
        int n = 3;
        printpath(n, "");
    }
    public static void printpath(int n, String ans){
        if(n==0){                          //pro active approach
            //if(n<0) return;
            System.out.println(ans);
            return;
        }
                                           //re-active approach
        if(n-1)>=0)
        printpath(n-1, ans+1);
        if((n-2)>=0)
        printpath(n-2, ans+2);
        if(n-3>=0)
        printpath(n-2, ans+3);
    }
}
```

Maze Path.



(2,2)





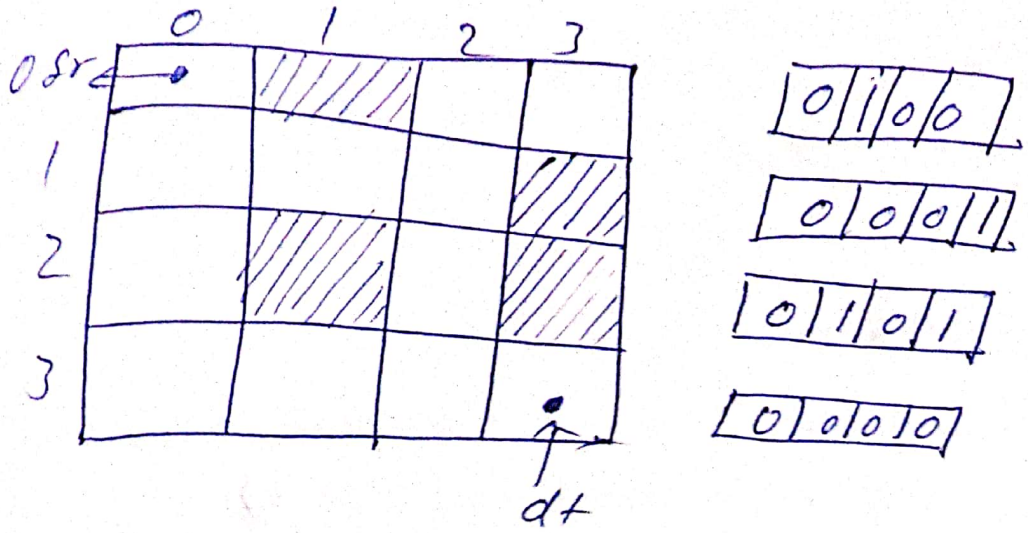
Mazepath.

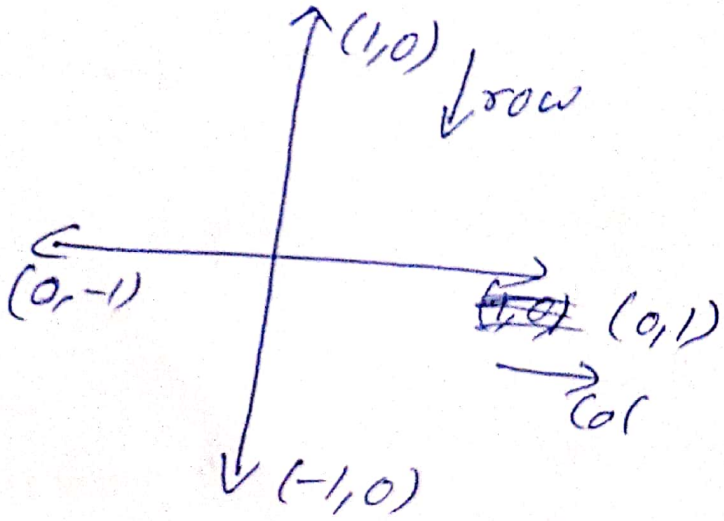
```
public class mazepath {
    public static void main(String[] args) {
        int sr = 0;
        int sc = 0;
        int dr = 2;
        int dc = 2;

        printmaze(sr, sc, dr, dc, "");
    }

    public static void printmaze(int sr, int sc, int dr,
        int dc, String ans) {

        // if(sr>dr || sc>dc) return;

        if(sr == dr && sc == dc) {
            System.out.println(ans);
            return;
        }
        if(sr+1 <= dr)
        printmaze(sr+1, sc, dr, dc, ans + "V");
        if(sc+1 <= dc)
        printmaze(sr, sc+1, dr, dc, ans + "H");
    }
}
```

Steps when max jump = n.

```
Public class Stairs1{
    Public static void main(String[] args) {
        int n=4;
        printpath(n,n,"");
    }
    public static void printpath(int n, int jump, String ans)
        if(n == 0){
         System.out.println(ans);
         return;
        }
        for(int i=1 ; i<= jump; i++){
            if((n-i)>=0)
            printpath(n-i, jump, ans+i);
        }
    }
}
```

Rat Maze / Flood Fill





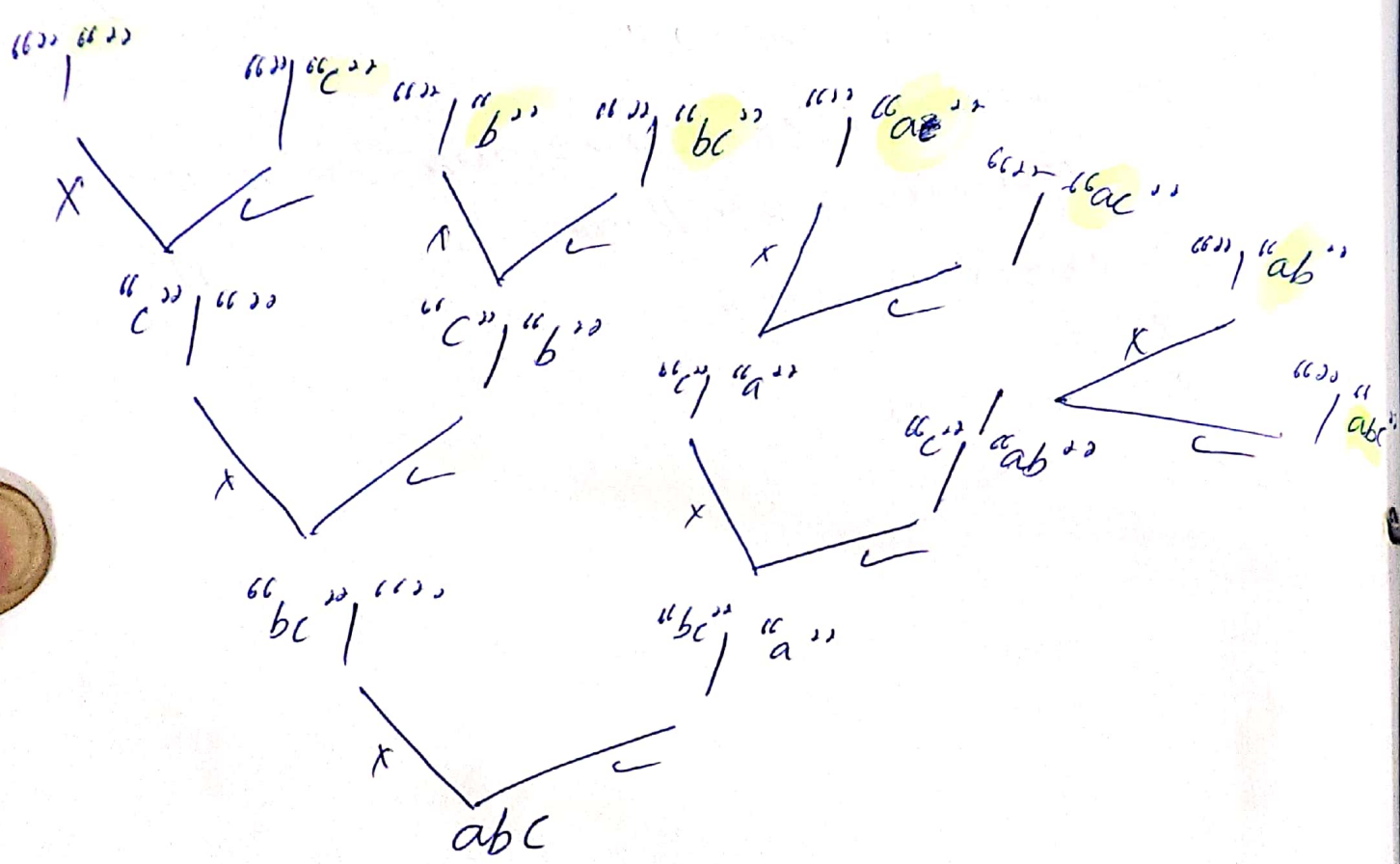
Public class ratmaze {

```
    Public static void main (string[] args){
        int sr = 0;
        int sc = 0;
        int dr = 3;
        int dc = 3;
        int[][] maze = {
                {0,1,0,0},
                {0,0,0,1},
                {0,1,0,1},
                {0,0,0,0}  };
```

```
boolean [][] visited = new boolean [maze.length] [maze
        length];
        Path(sr, sc, dr, dc, "", maze, visited);
}
// Backtracking
Static void Path(int sr, int sc, int dr, int dc, String ans,
        int[][] maze, boolean[][] visited){
    if(sr==dr && sc==dc){
        System.out.println(ans);
        return;
    }
    visited[sr][sc] = true;
    // LDRU
    // left
    if(issafe(sr, sc-1, dr, dc, maze) && !visited[sr][sc-1]){
        Path(sr, sc-1, dr, dc, ans+"L", maze, visited);
    }
    // down
    if(issafe(sr+1, sc, dr, dc, maze) && !visited[sr+1][sc]){
        Path(sr+1, sc, dr, dc, ans+"D", maze, visited);
    }
    // Right
    if(issafe(sr, sc+1, dr, dc, maze) && !visited[sr][sc+1]){
        Path(sr, sc+1, dr, dc, ans+"R", maze, visited);
    }
    // up
    if(issafe(sr-1, sc, dr, dc, maze) && !visited[sr-1][sc]){
        Path(sr-1, sc, dr, dc, ans+"U", maze, visited);
    }
```

```
    visited[sr][sc] = false;
}

static boolean isSafe(int sr, int sc, int dr, int dc,
        int[][] maze){
    if(sr<0 || sc<0 || sr>=maze.length || sc>=
        maze[0].length || maze[sr][sc]==1){
        return false;
    }
    return true;
}
}
```

Sub Seq by Post area

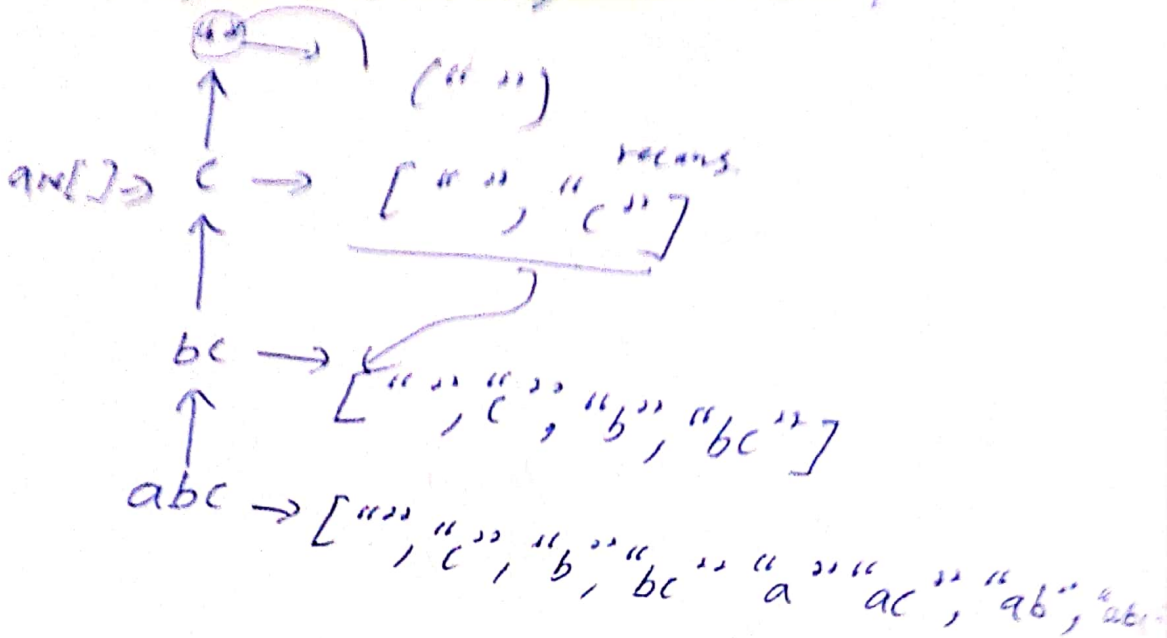


abc → ϕ, a, ab, abc, b, bc, c, ac

```
Import java.util.ArrayList;
Public Class substring {
    Public static void main(String[] args) {
        String str = "abcd";
        ArrayList<String> ans = getSubs2(str, "");
        System.out.println(ans);
        System.out.println(ans.size());
    }
```

```
Static Arraylist<String> getSubSq(String str, String ans){
    if(str.length() == 0){
        Arraylist<String> base = new ArrayList<>();
        base.add(ans);
        return base;
    }

    Char ch = str.charAt(0);
    String ros = str.substring(1);

    ArrayList<String> recans1 = getSubSq(ros, ans);
    ArrayList<String> recans2 = getSubSq(ros, ans+ch);

    Arraylist<String> retans = new ArrayList<>();

    retans.addAll(recans1);
    retans.addAll(recans2);
    return retans;
}
}
```

Substring in String 2 way Postorder

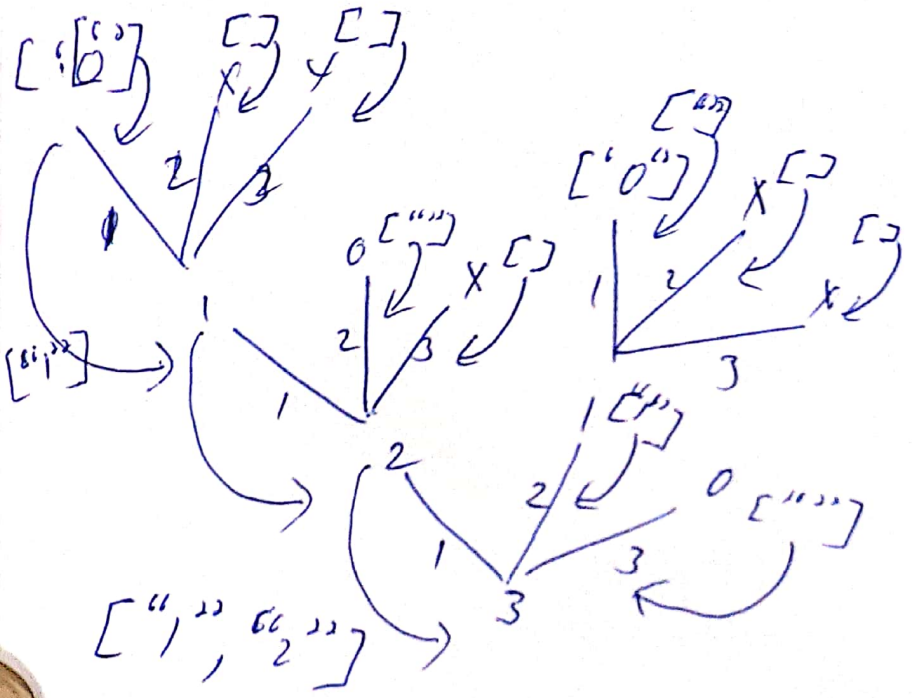



```
# import java.util.ArrayList;
Public class substring2 {
    Public static void main (String[] args) {
        String str = "abc";
        ArrayList<String> ans = getsubsquay2(str);
        System.out.print(ans);
    }
    Static void ArrayList<String> getsubsange(String str
        if (str.length() == 0) {
            ArrayList<String> base = new arraylist<>();
            base.add("");
            return base;
        }
        string ros = str.substring(1);
        ArrayList<String> recans = getsubsquay2(ros);
        // Post Area
```

```
ArrayList<String> ans = new ArrayList<>();
ans.addAll(recans);

char ch = str.charAt(0);

for(String s: recans){
        ans.add(ch+s);
 }
return ans;
}
}
```

~~# Dial pad post order~~

Stair path by post order.



["111", "12", "21", "3"]

```
Import java.util.ArrayList;
Public class StairJump {
    Public static void main(String[] args)
    {
      System.out.println(getPath(3));
    }
    Public static ArrayList<String> getPath(int n){
        if(n<0) return new ArrayList<>();
        if(n==0) {
            ArrayList<String> base = new ArrayList<>();
```

```
base.add("");
return base;
}
ArrayList<String> ans = new ArrayList<>();
    base.add("");
    return base;
ArrayList<String> one = getPath(n-1);
ArrayList<String> two = getPath(n-2);
ArrayList<String> three = getPath(n-3);
for (String s:one){
    ans.add("1"+s);
}
for(String s : two){
    ans.add("2"+s);
}
for(String s : three){
    ans.add("3"+s);
}
return ans;
}
```

DialPad By post order.

```
import java.util.ArrayList;
Public class dialpad {
static String[] keys = {"#", ".", "abc", "def", "ghi",
  "jkl", "mno", "pqrs", "tuv", "wxyz"};
Public static void main(String[] args) {
      System.out.print(kpc("23"));
}
static ArrayList<String> kpc(String str) {
   if(str.length() == 0){
     ArrayList<String> base = new ArrayList<>();
     base.add("");
     return base;
   }
   ArrayList<String> recans = kpc(str.substring(1));
   char ch = str.charAt(0);
   int idx = ch - 48;
   String pressKey = keys[idx];
   ArrayList<String> ans = new ArrayList<>();
   // for(char c: pressKey.toCharArray()){
      for(int i=0; i<pressKey.length(); i++){
         char c = pressKey.~~length(); i++){~~
                          charAt(i);
```

```
for(string s: recans){
    ans.add(c+s);
}
}
return ans;
}
}
```

Stack overflow
Quora
Medium
Codingbat
onlinegdb.com → C-compiler
Codechef

Escape Sequences

\t Insert a tab in the text at this point.
\b Insert a backspace.
\n Insert a newline in the text at this point.
\r Insert a carriage return in the text at this point

\f Insert a formatted in the text at this point
\' Insert a single quote character in the text
\" Insert a double quote character in the text
\\ Insert a backslash character in the text

(arr[0] >= arr[1]) ? 0 : 1; → ternary operator

Maze Path - multible shift Pre order

```
Public class mazepath1 {
    Public static void main (String[] args) {
    int sr = 0;
    int sc = 0;
    int dr = 2;
    int dc = 2;
    PrintMaze (sr, sc, dr, dc, "");
    }
    Public static void PrintMaze (int sr, int sc, int dr, int dc,
      String ans) {
    // if (sr > dr || sc > dc  return;
    if (sr == dr  || sc == dc) {
      System.out.println (ans);
      return;
    }
    for (int i = 1; i <= dr; i++) {
      if (sr + i <= dr && sc <= dc)
        PrintMaze (sr+i, sc, dr, dc, ans + "V" + i + " ");
    }
    for (int i = 1; i <= dc; i++) {
      if (sr + i <= dr && sc + i <= dc)
        PrintMaze (sr+i, sc+i, dr, dc, ans + "D" + i + " ");
    }
```

```
for (int i = 1; i <= dc; i++) {
    if (sc + i <= dc && sr <= dr)
        PrintMaze (sr, sc+i, dr, dc, ans + "H-" + i + " ");
}
}
```

V-1 V-1 H-1 H-1
V-1 V-1 H-2
V-1 D-1 H-1
V-1 H-1 V-1 H-1
V-1 H-1 D-1
V-1 H-1 H-1 V-1
V-1 H-2 V-1
V-2 H-1 H-1
V-2 H-2
D-1 V-1 H-1
D-1 D-1
D-1 H-1 V-1
D-2
H-1 V-1 V-1 H-1
H-1 V-1 D-1
H-1 V-1 H-1 V-1

H-1 V-2 H-1
H-1 D-1 V-1
H-1 H-1 V-1 V-1
H-1 H-1 V-2
H-2 V-1 V-1
H-2 V-2

Google

1- Programming Language-*

- DSA*
- Projects*

Internship along the way

Open source contribution

Competitive programming

website to learn language

learnpython.org

freecodecamp(n)

Resources

- Udemy
- coursera
- Udacity
- Youtube

after language or parallely

Development , DSA , Open source contri

- after learn language make project

Algorithms.

Divide & Conquer, Recursion, Back Tracking DP, Greeds

Development (Startups, Resume, MNC) Boost

- ML, Quantum Computing, 5G, DARQ
- Web development, Internet of Things
- Android development, Cyber security
- Blockchain Development, Augmented Reality etc Automation

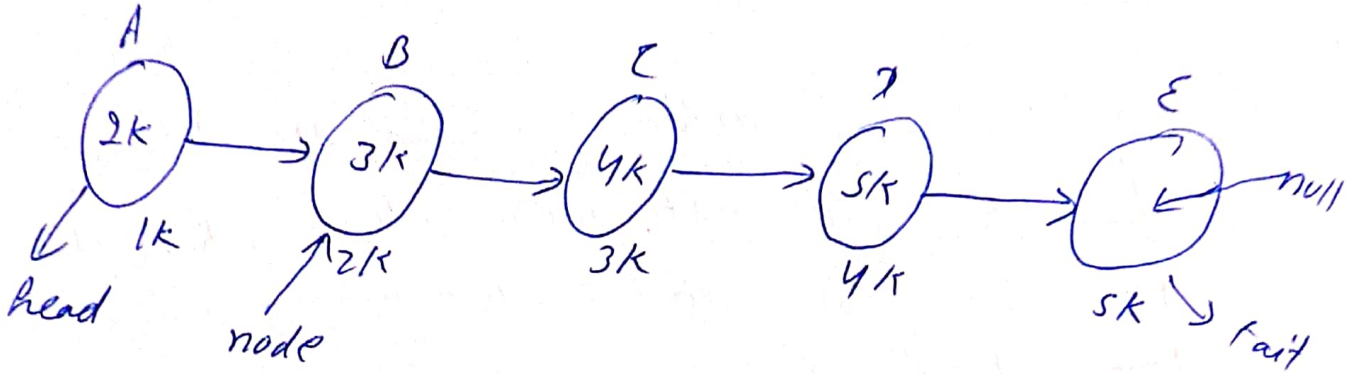
Mindset

It is okay to take time

DSA → It is okay to check Editorials

Coding bat

linked list is a data structure




```
Public class LinkedList {
    Static Node head;
    Static class Node {
        int data;
        Node next;

        Node () {}

        Node(int data) {
            this.data = data;
        }
    }
    Static void display (Node iter) {
        while (iter != null) {
            System.out.println(iter.data);
            iter = iter.next;
        }
    }
    Static void addFirst (int data) {
        if (head == null) {
            head = new Node (data);
        } else {
```

```
        Node node = new Node(data);
            node.next = head;
            head = node;

        }
    }
    static void addAt(int idx, int data){
        if(idx<=0){
            System.out.println("index out of bound");
            return;
        }
        if(head == null){
            if(idx==0){
                head = new Node(data);
            }else{
                Sy. --  ( ou bound );
            }
        }else{
            if(idx==1){
                Node node1 = new Node(data);
                node1.next = head;
                head = node1;
                return;
            }
            Node node = new Node(data);
            Node temp = head;
            while(idx--!=2 && temp!=null) temp = temp.next;
            if(temp==null){
                System.out -- (out of bound);
                return;
            }
```

```
        node.next = temp.next;
        temp.next = node;
    }
}
public static void main(String[] args){
    addFirst(10);
    addFirst(30);
    addFirst(40);
    addFirst(50);
    addAt(4,70);

    display(head);
}
}
```

(int) -1e9
-10^9

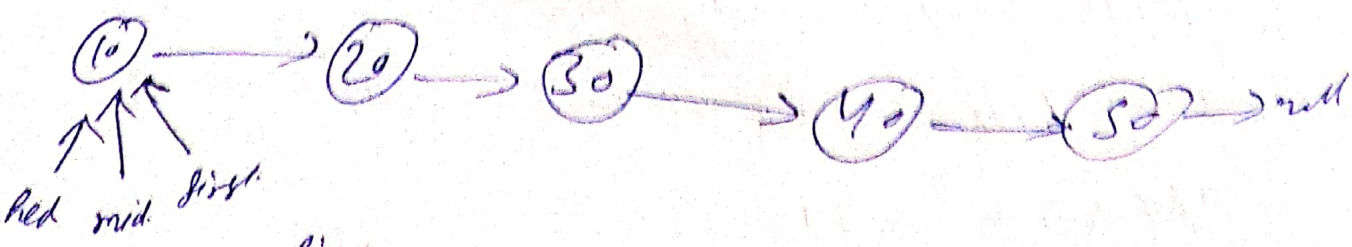
Q find mid without using size in linked list using

Print mid

get size

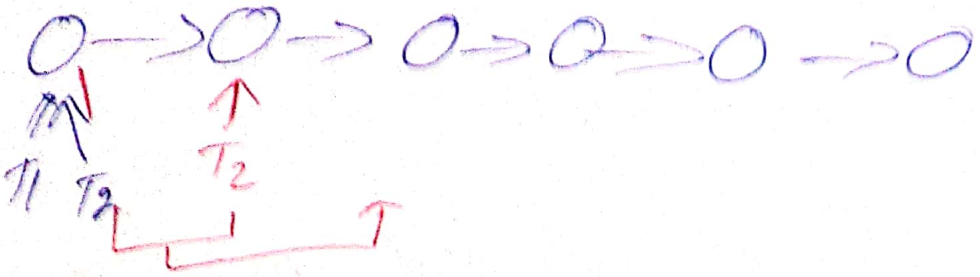
find mid.

Q find the middle of without using size

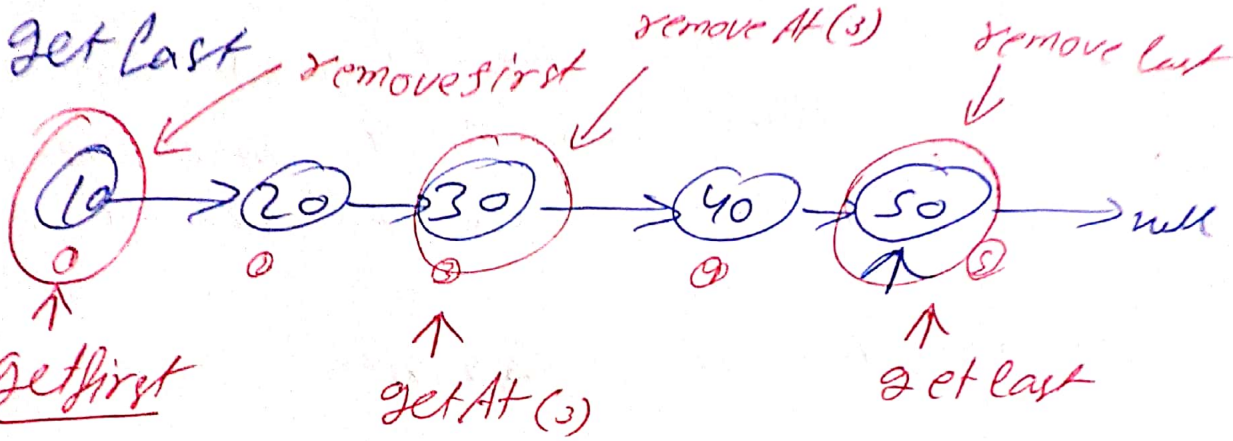


first - 2
mid - 1

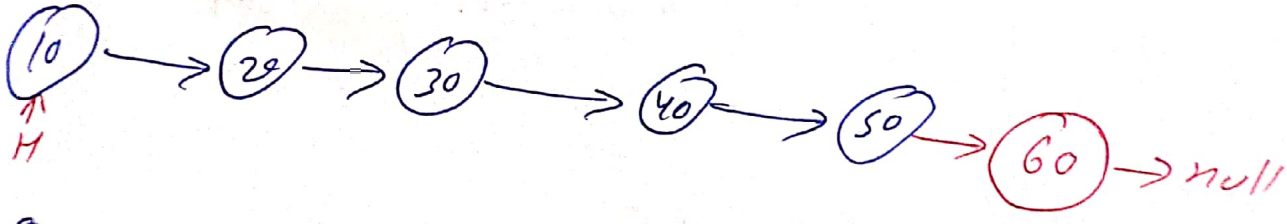
Q Last 2nd element using 2 pointer



Q getLast



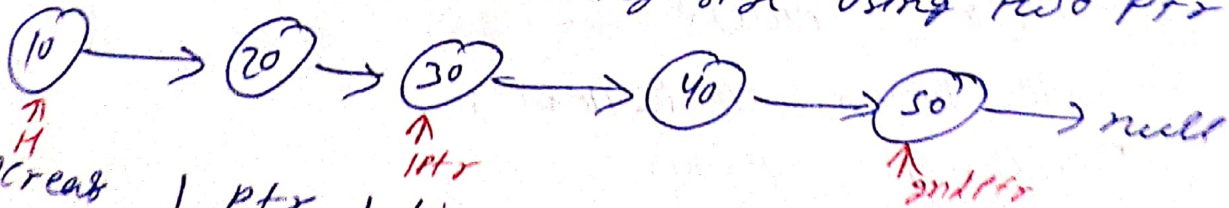
Q



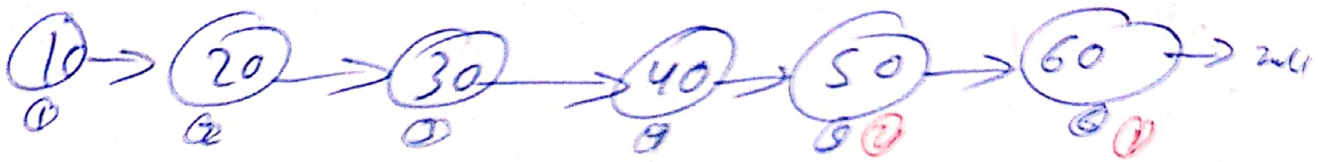
Q size → List size

while(temp != null)

Q find mid without using size using two ptr.



① Index from last k. Using 2 ptr

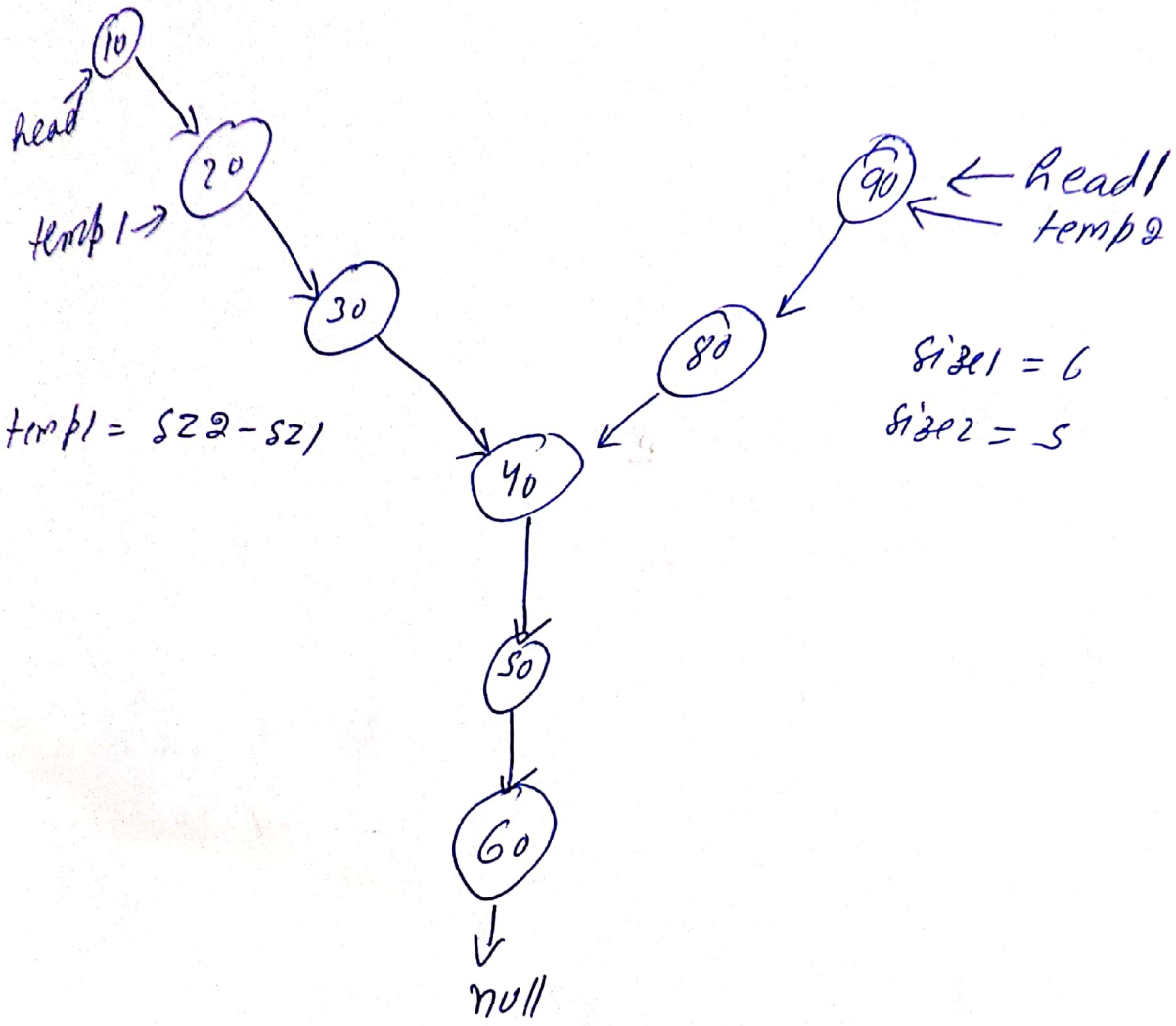


Let we have to find last 3rd node the move any 1 ptr k-1 times (1 time) then move both ptr along from head until ptr2 != null at that pt ptr1 give us the desired result.

② Reverse the node of LL


```
Static Void reverse () {
    Node Prev = null;
    Node Next = head;

    while (Next != null) {
        Node temp = Next.next;
        Next.next = Prev
        Prev = Next;
        Next = temp;
    }
    head = Prev;
}
```

Q Find intersection



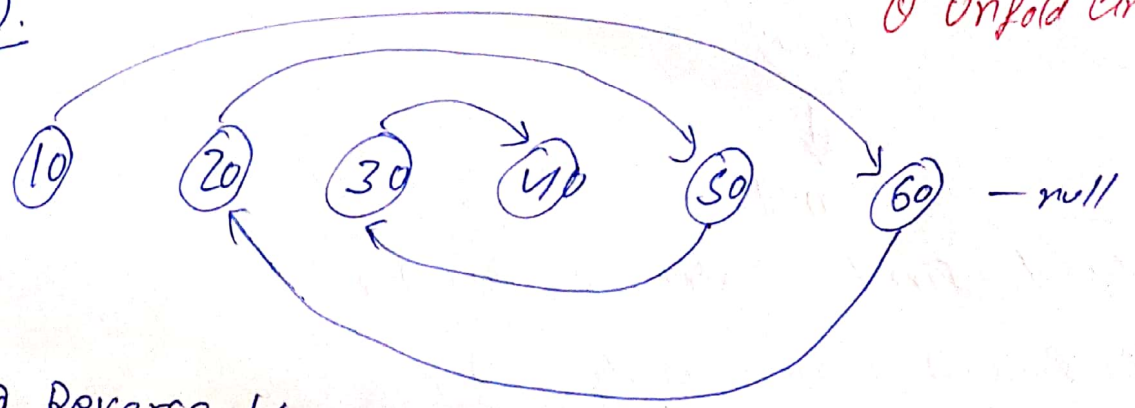

```
static void find_intersection() {
    Node head2 = new node(90);
    Node node1 = new Node(80);

    head2.next = node1;
    node1.next = getNodeAt(4);

    int size1 = getSize(head);
    int size2 = getSize(head2);
    int diff = Math.abs(size1 - size2);
    Node temp1 = head;
    Node temp2 = head2;
```

```
If (size1 > size2) {
  while (diff--!=0){
      temp1= temp1.next;
  }
}else {
    while (diff--!=0){
      temp2 = temp2.next;
    }
}

while ( temp1 != temp2){
      temp1 = temp1.next;
      temp2 = temp2.next;
}

sysO (temp1.data);
```

@ fold a linkedList

@ unfold linkedList(obj)

Q.

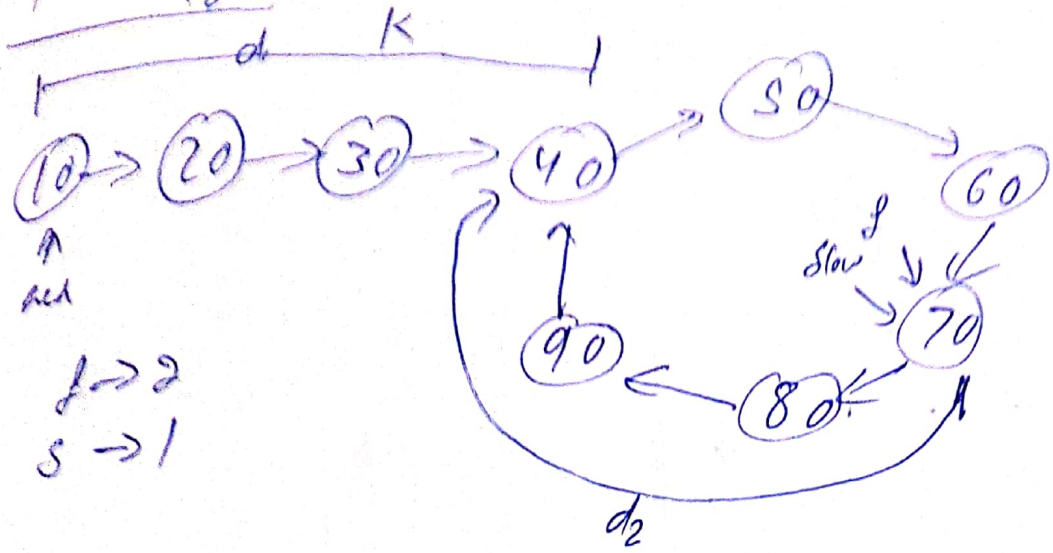


@ Reverse it

k = 3

10	20	30	40	50	60	70	80
30	20	10	60	50	40	70	80

Q Find cycle



$d_1 = d_2$

Stack

Q Balance or not :-

[a+{b+(d+e)}] ✓

[a+{a+(d+e})] ✗

next greater

5 4 8 3 2 7 16 4 12 14 3

8 8 11 7 7 16 -1 12 14 -1 -1

NGE → NGE Right

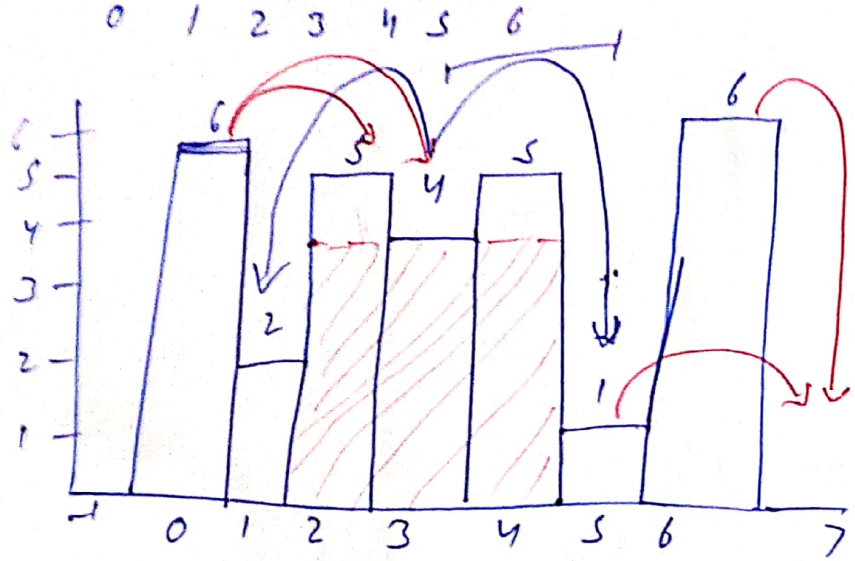
NGE → NGE left

{ index → start → End
  value → End → start }

{ Index → End → start
  value → start → End }

Q Find Area

arr [6, 2, 5, 4, 5, 1, 6]
0 1 2 3 4 5 6



R :- | 1 | 5 | 3 | 5 | 5 | 7 | 7 |
0 1 2 3 4 5 6

L :- | -1 | -1 | 1 | 1 | 3 | -1 | 5 |
0 1 2 3 4 5 6

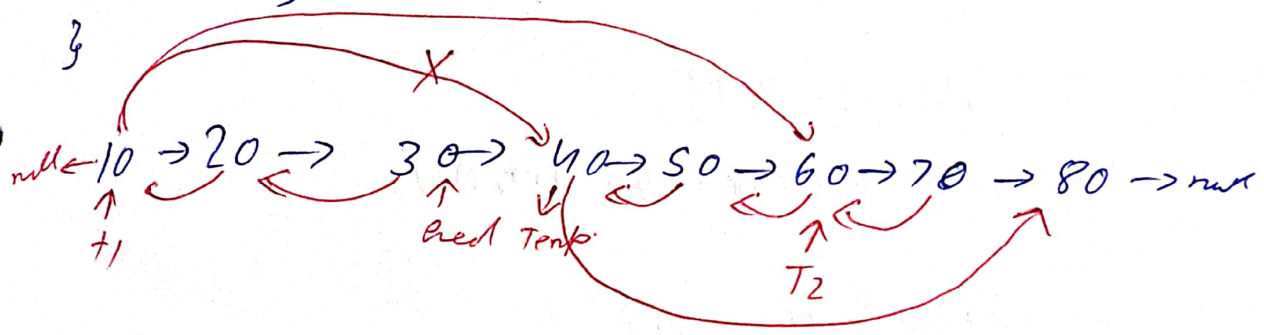
Q Stock Span

0 1 2 3 4 5
[3, 2, 8, 1, 7, 8]
[1 1 3 1 2 6]

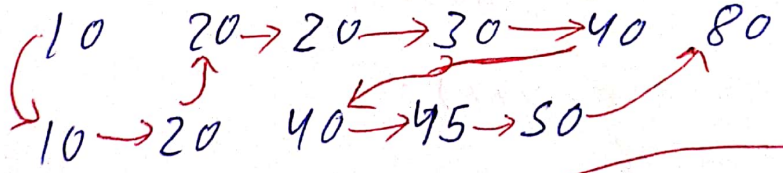
Q reverse k LL

```
Static Node reverse_Helper (Node node, int k){
Node prev = null;
Node curr = node;

for (int i=1; i<=k; i++){
  Node temp = curr.next;
   curr.next = prev;
    prev = curr;
    curr = temp;
}
node.next = curr;
```

```
    return prev;
}
static void reverseKl(Node node, int k) {
    int sz = size(head);
    Node t1 = node;
    head = reverseHelper(head, k);
    for(int i=1; i<sz/k; i++){
        Node t2 = reverseHelper(t1.next, k);
        Node temp = t1.next;
        t1.next = t2;
        t1 = temp;
    }
```

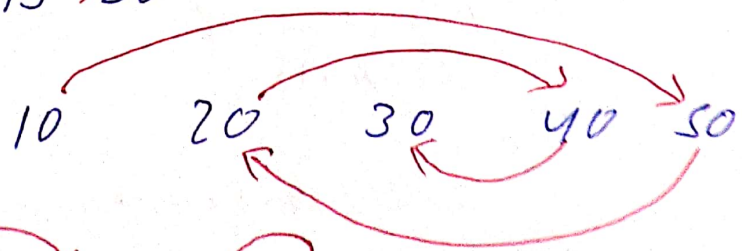


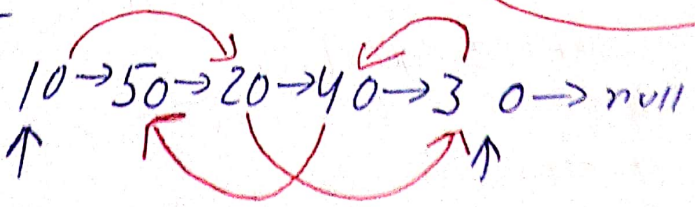
Q Merge two sorted LL.



Q Fold a LL

Q Unfold a LL



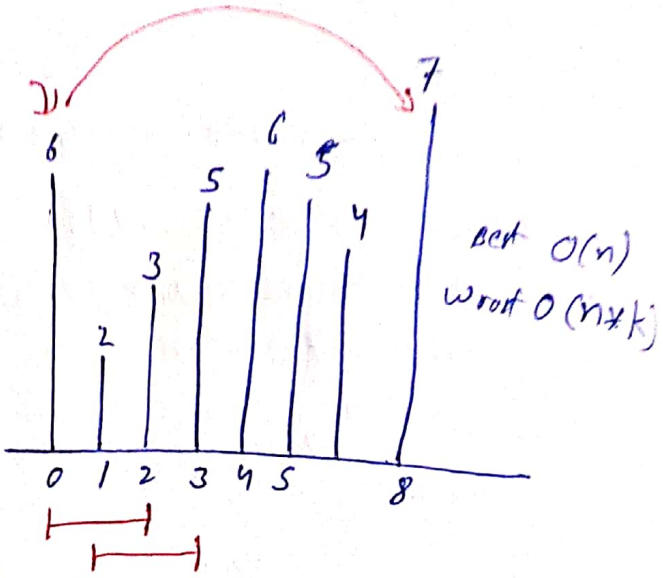


Q. Maximum in k window

[6 2 3 1 5 6 5 4 7]

NGE [7 3 5 5 6 7 7 7 0]

[6 3 5 6 6 6 7]




```
int[]arr = {6,2,3,1,5,6,5,4,7};
Sout (Arrays.toString(maxkwind(arr,3)));

NGE
Static int[] ngeright(int[] arr){
   int[] ans= new int[arr.length];
      Stack<Integer> st. new stack<>();

      for (int i=0; i<arr.length;i++){
         if (st.size()==0){
            st.Push(i);
         }else{
            while(st.size()>0 && arr[st.Peek()]<arr[i]){
               ans[st.POP()] = i;
            }
            st.Push(i);
         }
      }
   while(st.size()>0){
         ans[st.POP()]=arr.length;
   }
   return ans;
}
```

```
static int[] maxkwind(int[] arr, int k){
    int[] nge=ngeright(arr);
    Syso(Arays
    int[] ans = new int[arr.length-k+1];
    for(int i=0; i<=arr.length-k; i++){
        if(i+k-1<nge[i]){
            ans[i]=arr[i];
        }else{
            int j=nge[i];
            while(i+k-1>=nge[j]){
                j=nge[j];
            }
            ans[i]=arr[j];
        }
    }
    return ans;
}
```

Exp (5+(2+((6*2)+3)))

Infix

Prefix +5+2+*623

Post 5623*3+++

Infix :→ Eval. ✓

Convert → Prefix ✓
→ Postfix ✓

Prefix :- Eval ✓
→ Infix ✓
→ Postfix ✓

Postfix :- Eval ✓
→ Infix ✓
→ prefix. ✓

Evaluations :- infix Eval

```
Static int infixEval (String exp){
    Stack<Integer> oper = new Stack<>();
    Stack<Character> opera = new Stack<>();
    for(char ch : exp.toCharArray()){
        if(ch == '('){
            opera.push(ch);
        }else if(ch==')'){
            while(opera.peek() != '('){
                int val2=oper.pop();
                int val1 = oper.pop();
                int res = operation(val1, val2, opera.pop());
                oper.push(res);
            }
            opera.pop();
        }
        else if(ch>='0' && ch<='9'){
            oper.push(ch-'0');
        }else{
            while(opera.size()>0 &&
                precedence(ch)< precedence(opera.peek())){
                int val2 = oper.pop();
                int val1 = oper.pop();
                int res = operation(val2, val1, oper.pop
                oper.push(res);
            }
            opera.push(ch);
        }
    }
```

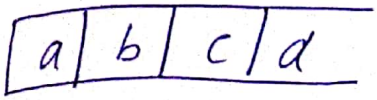
```
Static int Precedence (char op) {
    if (op == '(' || op == ')') return 0;
    if (op == '*' || op == '/') return 2;
    return 1;
}

Static int Operation (int val1, int val2, Char op){
    if (op == '+'){
        return val1+val2;
    } else if ( op == '-'){
        return val1-val2;
    } else if (op == '*'){
        return val1 x val2;
    } else {
        return val1 /val2;
    }
}

Static String infixToPre (String exp){
    Stack <String> oper = new Stack <> ();
    Stack <Character> opera = new Stack <> ();
    for (char ch : exp.toCharArray()){
        if (ch == '('){
            opera.Push (ch);
        } else if (ch == ')'){
            while (opera.Peek() != '('){
                String val2 = oper.pop();
                String val1 = oper.pop();
                String res = opera.pop() + val1 +val2;
                Oper.Push (res);
            }
            opera.pop();
```

```
}
else if(ch>='0' && ch<='9'){
    oper.Push(""+ch);
}else{
    while(opera.size()>0 && Precedence(ch)<Precedence
          opera.peek())){
        String val2=oper.pop();
        string val1=oper.pop();
        String res=opera.pop()+val1+val2;
        oper.push(res);
    }
    opera.push(ch);
}
}
while(opera.size()!=0){
    String val2=oper.pop();
    string val1=oper.pop();
    String res=opera.pop()+val1+val2;
    oper.push(res);
}
return oper.pop();
```

~~Static St~~

Adapters :-

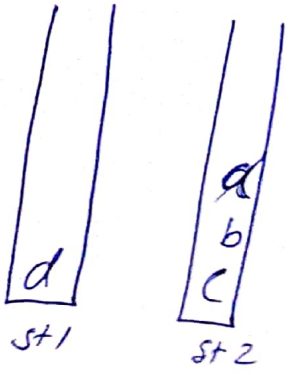
Q1 Implement Queue using Stack



FIFO

Push (a)
Push (b)
c

Pop()
(d)

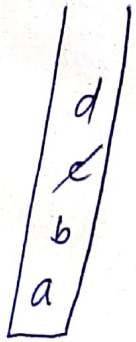


O(1) } ← Push efficient
O(n) } → Pop

O(1) } Pop efficient
O(n) → Push

2 Stack

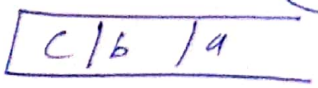
Q2. Implement Stack using queue



LIFO

Push (a)
(b)
(c)
Pop() //c
Push (d)

FIFO

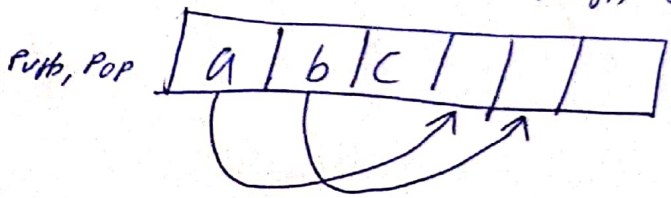


Queue 1

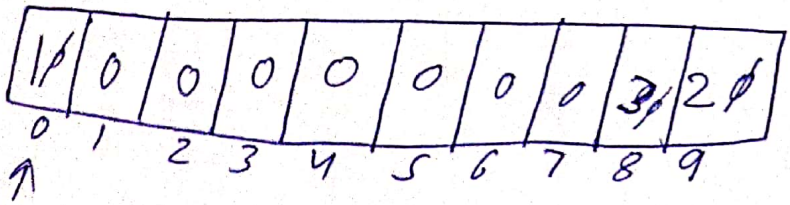
1,2 queue

Pop efficient

Push efficient



Q 2 Stack using Array :-



P1

Push(1) 1
Push (2) 2
Push (3) 2

Pop()

K - Stack
Array (fix Size)
Divide into parts
loss of space

} Approach 1

Changing Method
Better Utilization of space

} Approach 2

~~K~~ Stack
~~Array~~ (fix size)
Divide into ~~parts~~
~~loss of space~~

arr

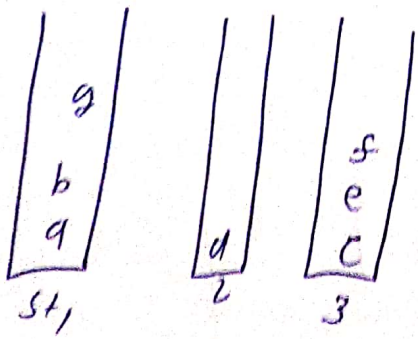
a	b	c	d	f	g				
0	1	2	3	4	5	6	7	8	9

top

6	3	5
1	2	3

next :-

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



free = 7
Plac = 6

(a, 1)
(b, 1)
(c, 3)
(d, 2)
(e, 3)
(f, 3)
(g, 1)

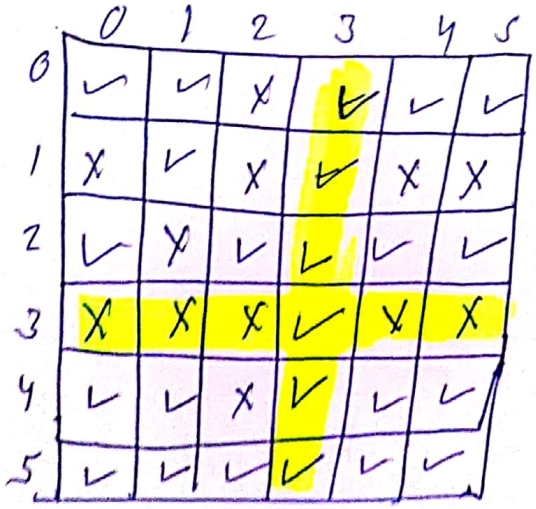
~~arr~~ Push

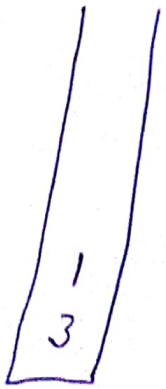
free global variable

```
* arr[free] = val ;
int place = free ;
free = next[place];
next[place] = top[sno]
top[sno] = place
```

```
Pop (int sno)
{
  int ans_idx = top[sno]
  int ans = arr[ans_idx]
  top[sno] = next[ans_idx]
  next[ans_idx] = free;
   free = ans_idx;
   return ans;
}
```

Celebrity





1 2 3 4 5

with 2D Array

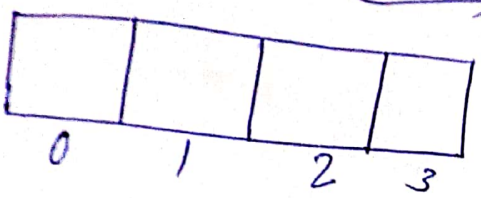
Permutation - queen Problem

$q_0\ q_1\ q_2\ q_3$

→ Permutation

→ Arrangement

$^nP_r = {}^nC_r \times r!$ $\quad \frac{n!}{(n-r)!}$



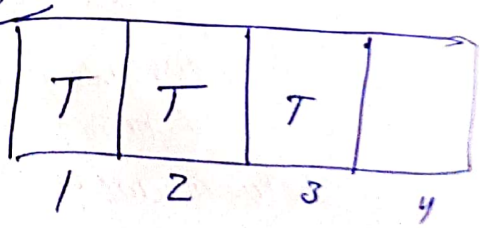
$n = 4$

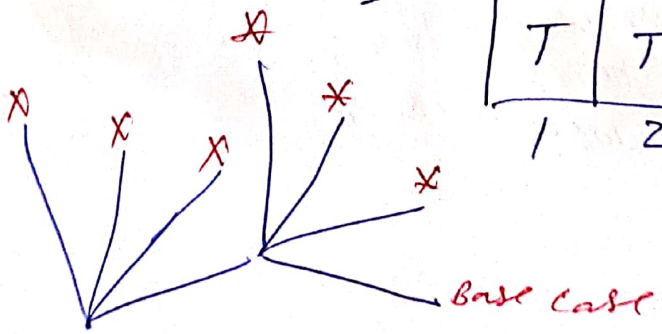
$r = 4$ $\quad \frac{4!}{(4-4)!} \Rightarrow 4!$

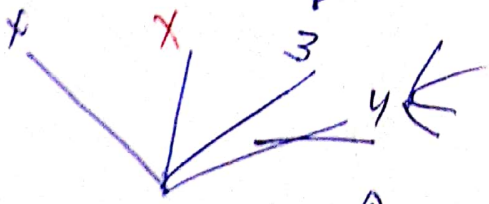
q_2 ___ ___ ___

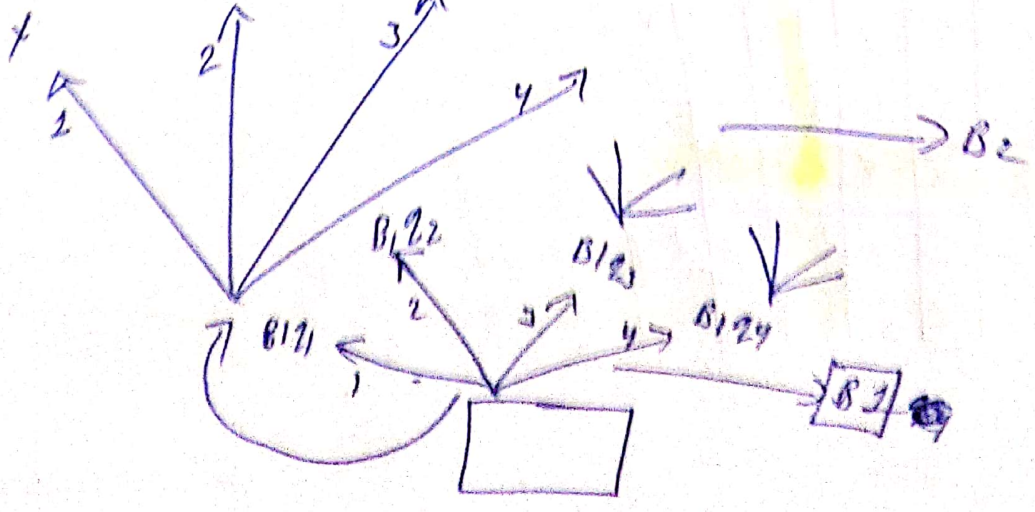
Given Box = 4

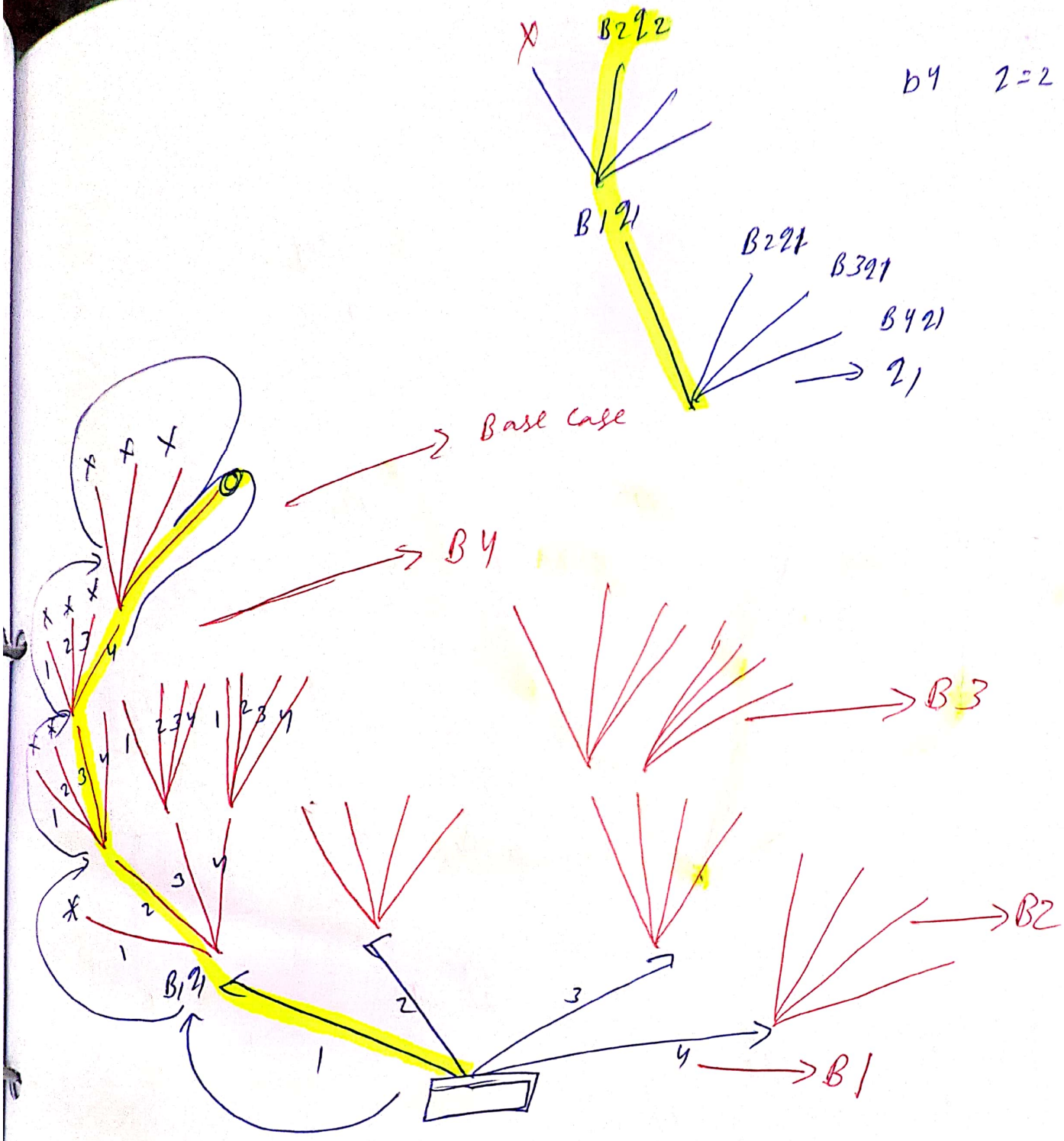
Visited







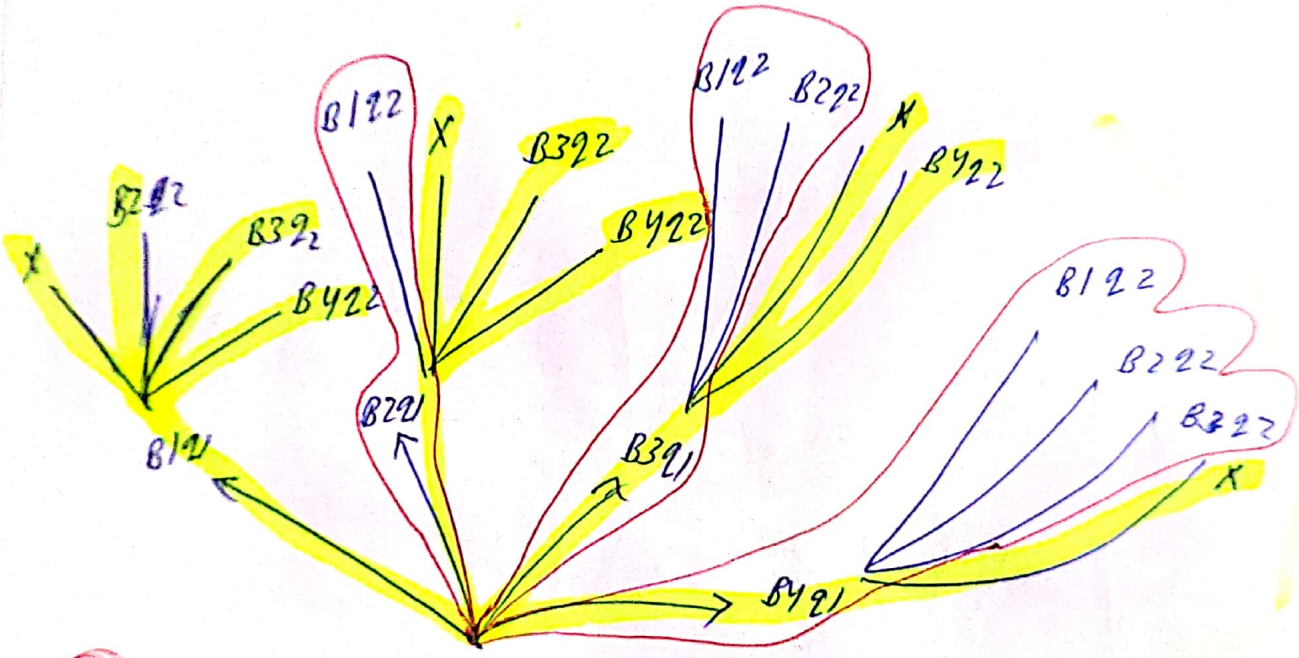



B2 2 2
B1 2 1
B2 2 1
B3 2 1
B4 2 1
2 1
Base case
B 4
B 3
B 2
B 1
B1 4

Combination in n queen

$$\frac{n!}{(n-x)!x!}$$

B4
q2



1

q1 q2 _ _
q1 _ q3 _
q1 _ _ q4
_ _ _ _

2

q2 q1 _ _
_ q1 q2 _
_ q1 _ q2
_ _ _ _

3

q2 _ _ q1
_ q2 q1 _
_ _ q1 q2

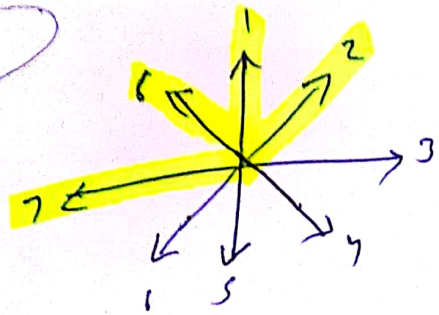
4

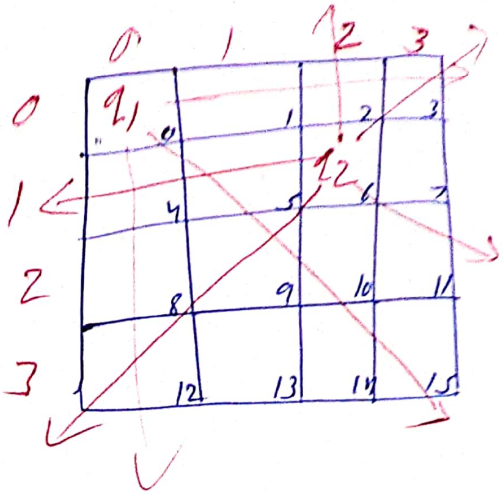
q2 _ _ q1
_ q2 _ q1
_ _ q2 q1

n queen Problem :-

4x4

safe





2D → 1D

4/4 = 1 r = 1

3%4 = 3 c = 2

0 1 2 3 4 5

r*4+c ⇒ 6

r, c, 4

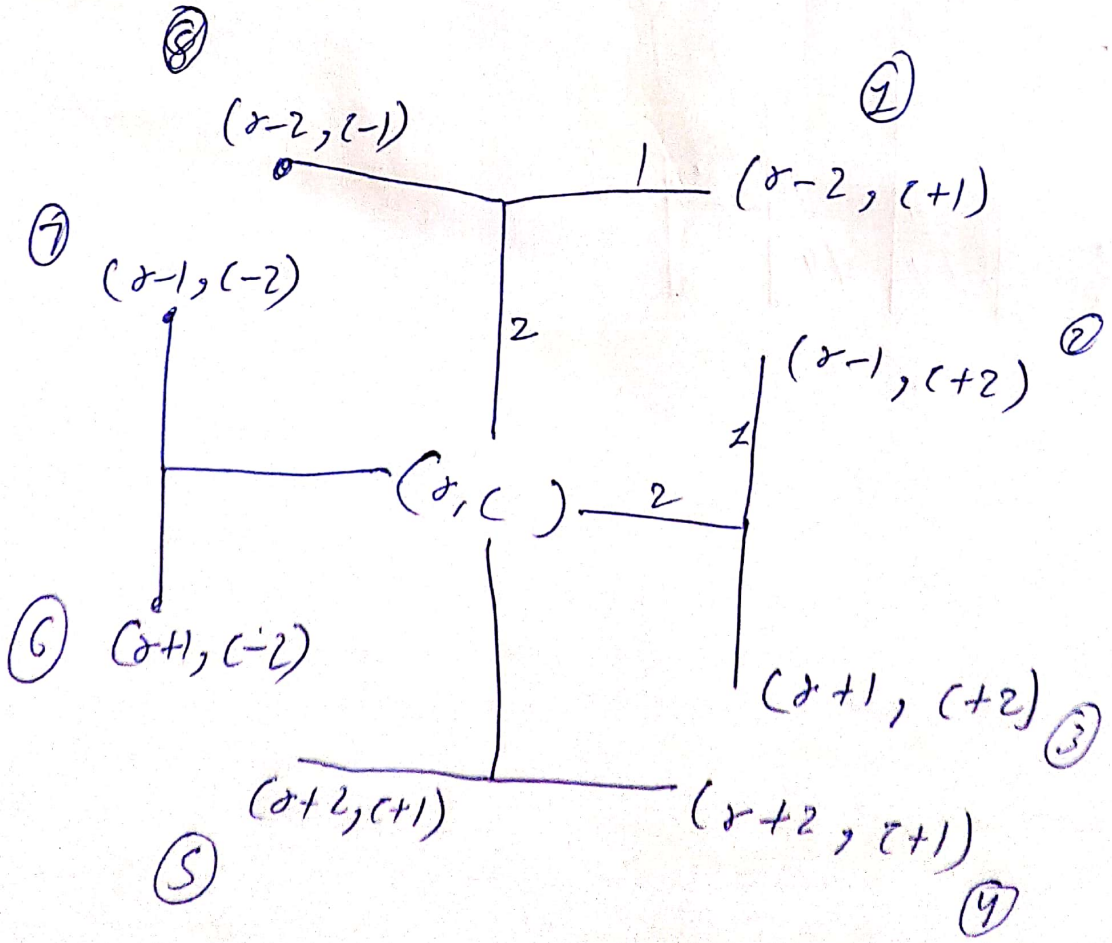
r = 1
c = 2

0	q1 1	q2 2	3
q2 4	5	6	q2 7
q3 8	9	10	q3 11
12	q4 13	q4 14	15

N- knight problem

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

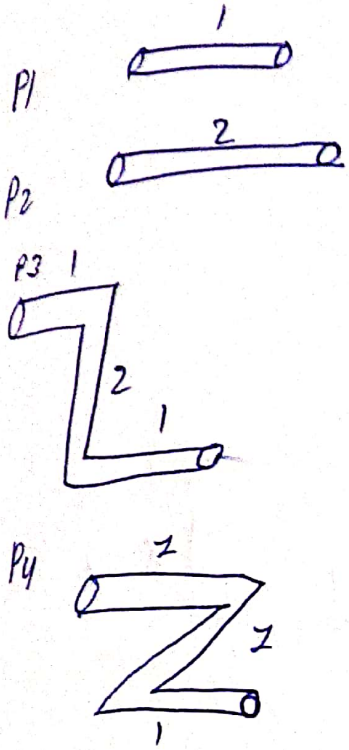
horse

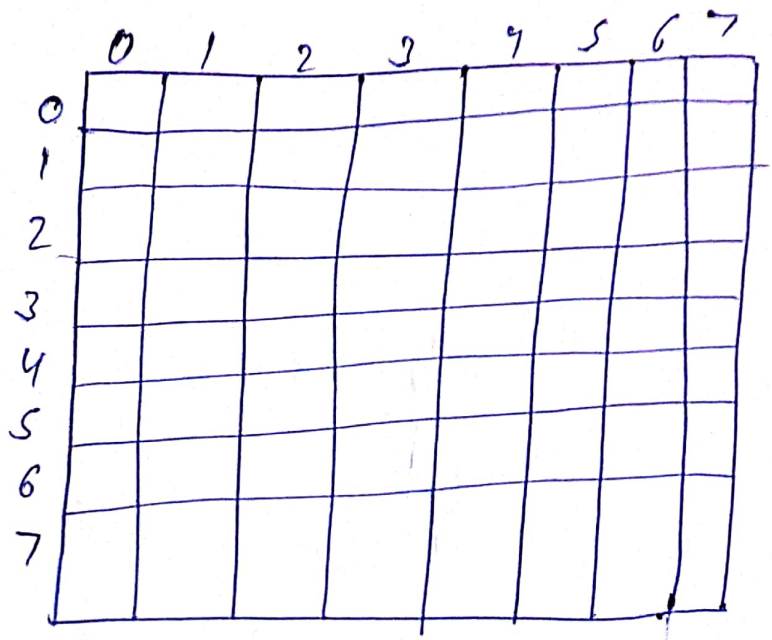


req = [int[][] ans]

Count

Pipe and queue

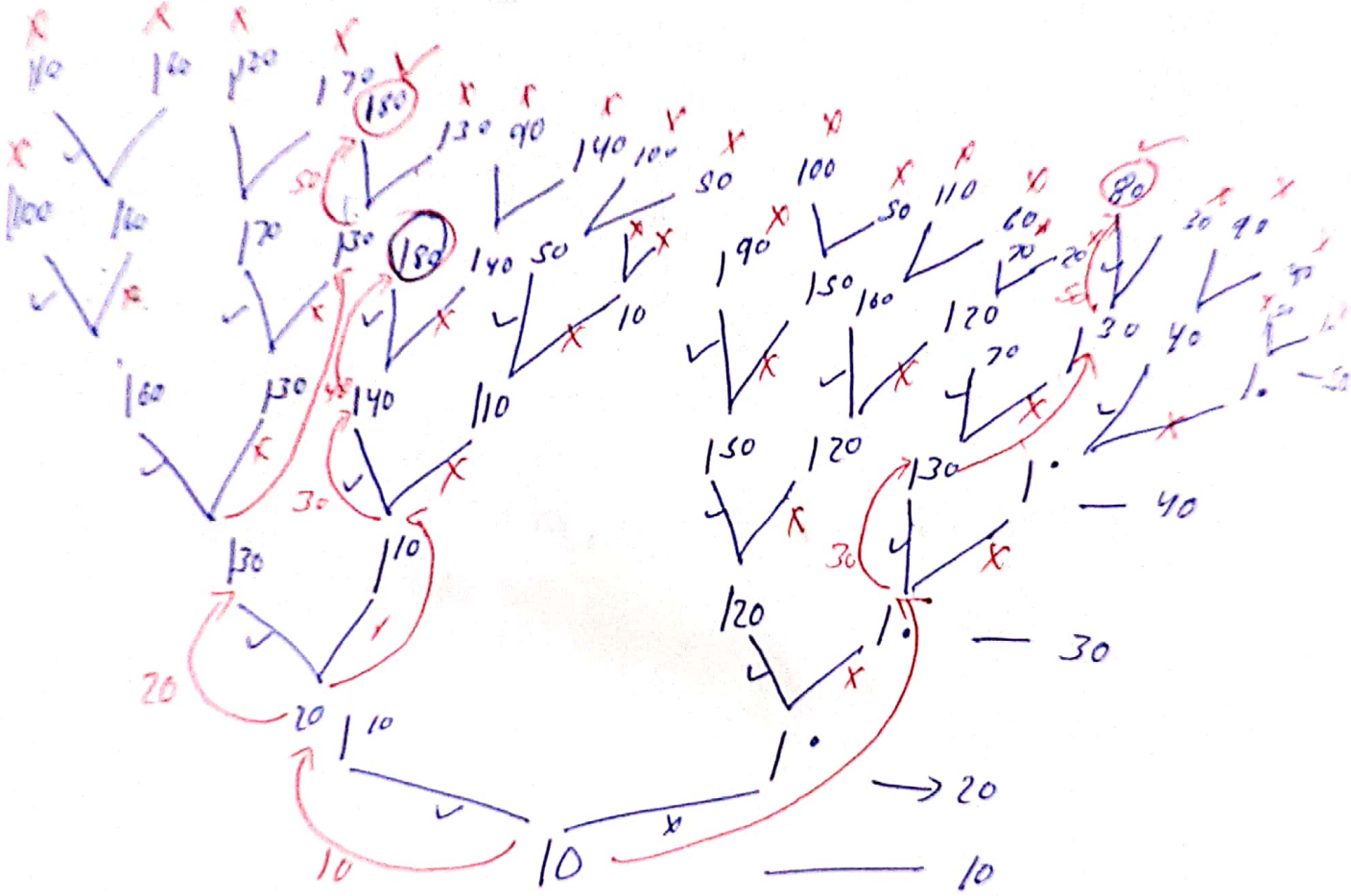





```
four (sr+r, sc+c, (ount+1);
```

Target sum subset :-

{10, 20, 30, 40, 50} target = 80;



Encoding :-

0 2 8

1 5 7 2

→ 1 - 5 - 7 - 2

→ 15 - 7 - 2

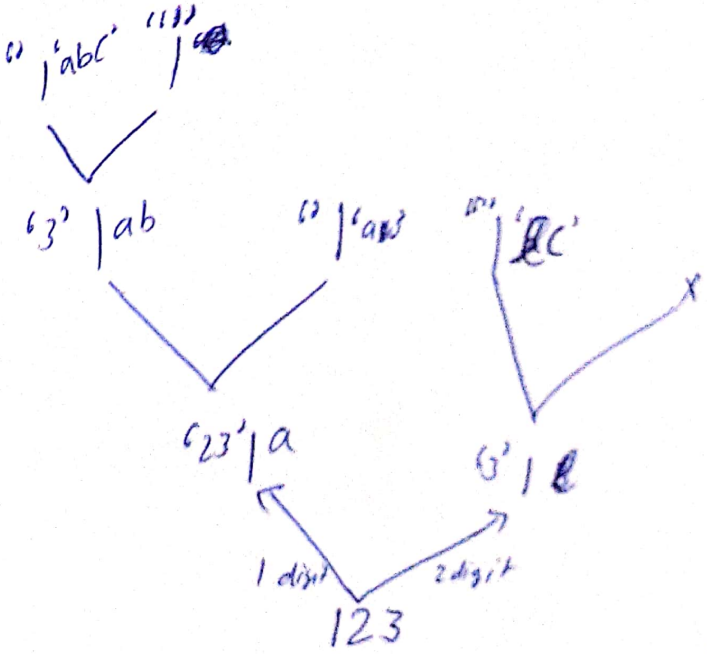
1 0 2 8 →

1 — 0 2 — 8 ✗

1 0 — 2 — 8 ✓

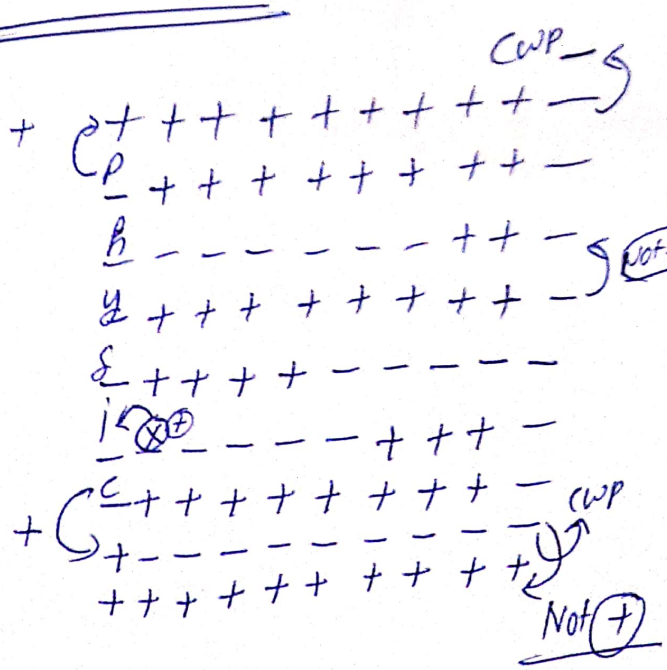
1 0 — 2 8 ✗

num1 = ch1 - '0' 49 - 48 original =
num X 10 + num2



- Encoding
- Sudoku
- Crypto
- Crossword
- Target sum subset

Cross word :



Physics maths
Chemistry history
Civics, geography

Test can we Place (H, V)
Place (H, V)
Unplace (H, V)

If word length > diff value

Tree

- Generic tree
- Binary tree
  - perfect BT
  - Complete BT
  - full Perfect
- Binary Search tree

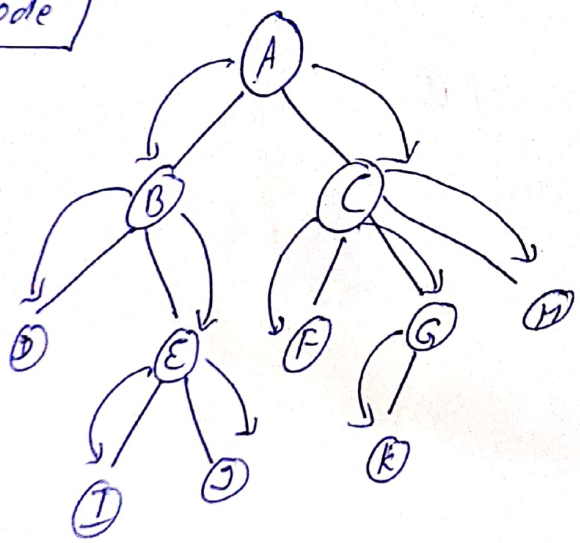
and so on

AVL Tree
B+ Tree
B Tree
Red Black Tree

} Algo

Generic Tree :- Step 1. Construction

leaf node
root node



One Parent can have any no of child
(No structured followed)

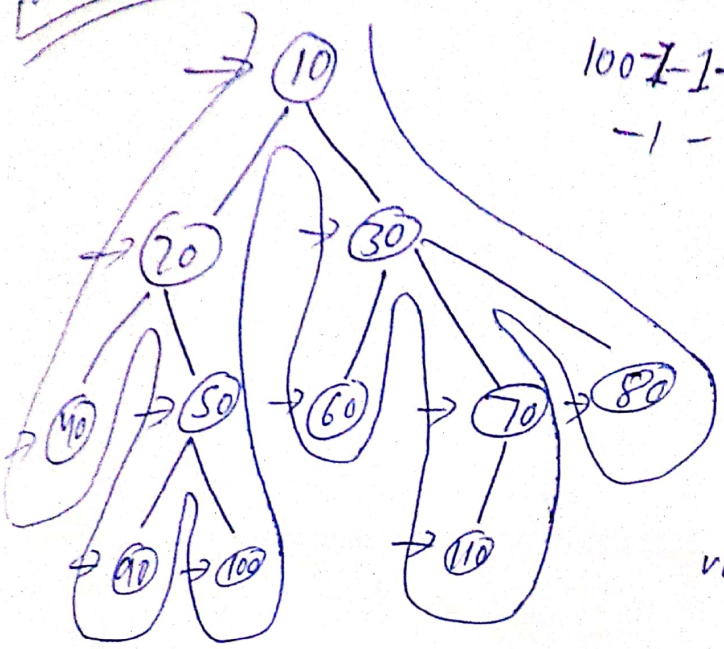
requirement

→ Treenode
  → structure

```
{ Class TreeNode {
  int val;
  ArrayList<TreeNode>
  }
```

Pre order Traversal :-

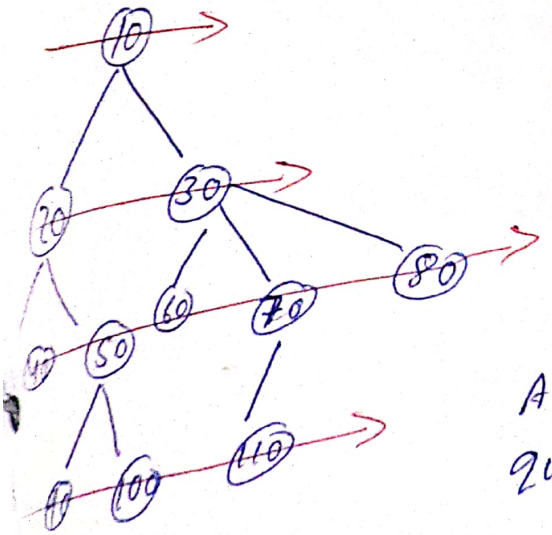
10 20 40 -1 50 90 -1
100 -1 -1 30 60 -1 70 110 -1 80
-1 -1 -1



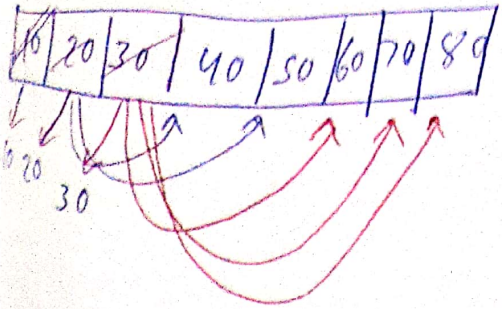
- Array List
- LL
- SNQ
- Recursion

```
void Display {
    Syso root.val
    for(TreeNode Nod : root.children){
        Syso (node.val).
    }
    for( . ( (  ↵  ) {
        display(node);
    }
}
```

Level order traversal :-

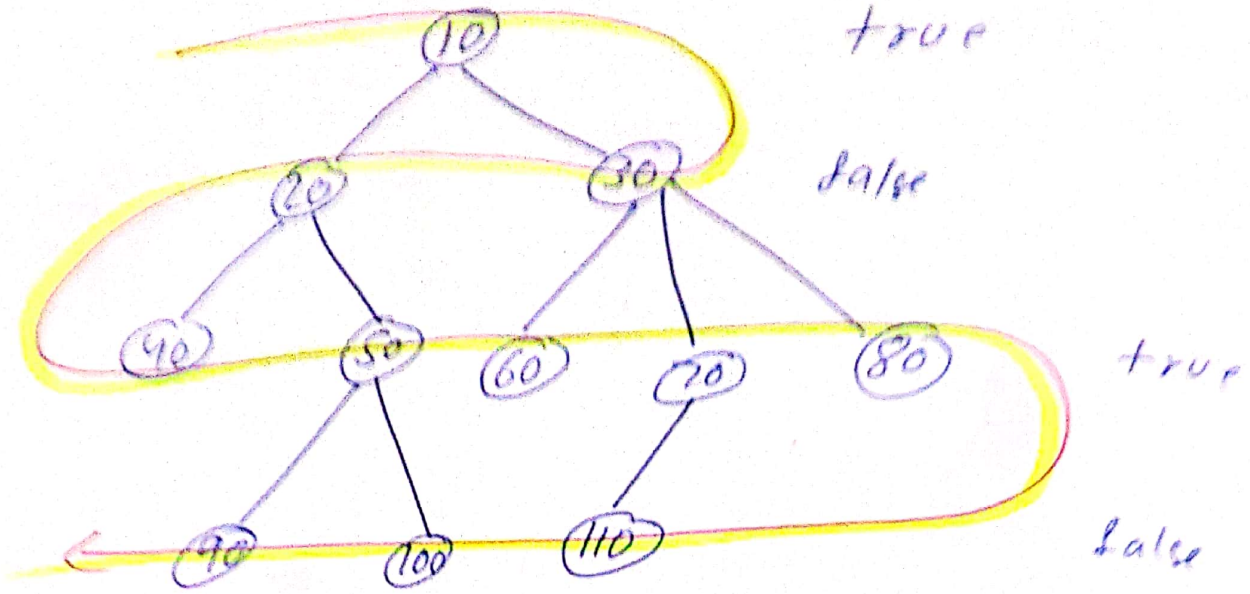


Array list
queue
LL

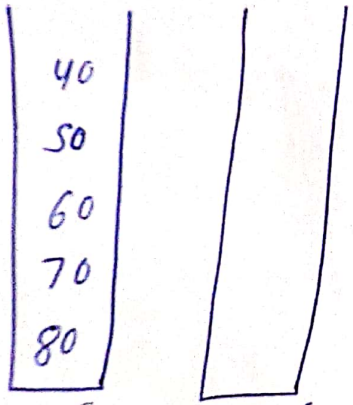



```
Array List que ——————
que add(roo);
while(que.size != 0) {
    int sz = que.size();
    while (sz--!=0) {
        TreeNod rn = que.remo()
        Print ( )
        for (node : children) {
            que.add (node);
        }
    Syso ( );
}
```

Print Zigzak :-



10, 30, 20, 40, 50, 60, 70, 80, 110, 100, 90



~~st1~~
~~st2~~
st1

~~st2~~
~~st1~~
st2

flag ~~true~~
~~false~~
true

Swap stack

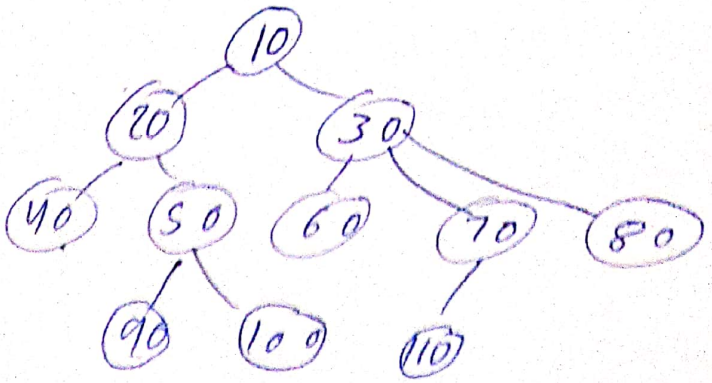
Printzig zag

st1
st2

flag = true
st1. Push (root);
while (st1.size() != 0) {
    while (st1.size() != 0) {
        rn = st1.pop();

```
Syso (rn.val+ " ");

if (flag){
    for(int i=0; i<rn.children.size(); i++){
        st2.Push(rn.children.get(i));
    }
}else{
    for(int i=rn.childre.size()-1; i>=0; i--){
        st2.Push(rn.child.get(i));
    }
}
}
flag= !flag;
Stack<TreeNode> Temp = st2;
    st2= st2;
    st2= temp;
}
}
}
```

Size, height, max, min, floor, ceil, Preceded, succeeded



Size = 11
height ⇒ 4
Edge height = 3
max = 110
min = 10
floor(86) ⇒ 80
ceil(86) ⇒ 90

Preceded & succeeded we can only find if the val is present in tree

Size of Generic tree :-

```
{
int count = 1;
boolean flag = true;
for (TreeNode node : root.children){
    count += size(node);
}
return count;
}
```

Height :-

```
int maxVal = 0
for (TreeNode node : root.children){
    int temp = height(node);
    if(temp > maxval){
        maxval = temp;
    }
}
return maxval+1;
```

max/min :-

```
int maxval = root.val;
for (TreeNode node : root.children){
    int temp = max(node);
    if(temp > maxval){
        maxval = temp;
    }
}
return maxval;
```

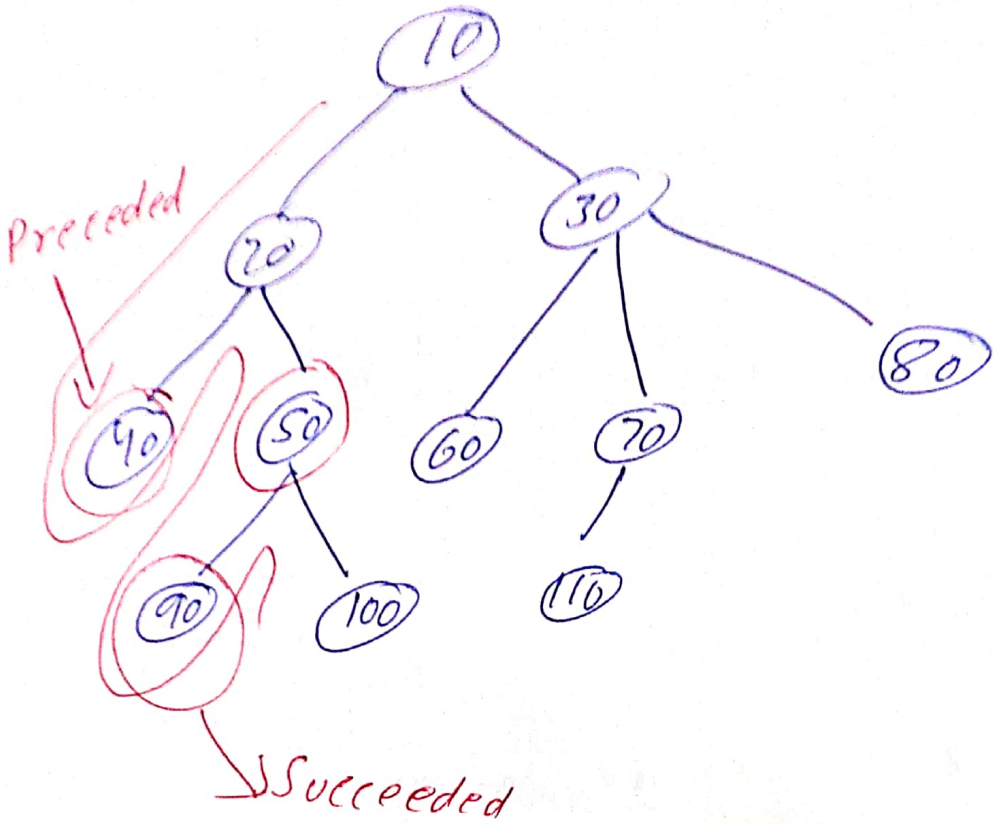
Ceil

```
Static int ~~Ceil (Tree~~ ceilval;
Static void Ceil (TreeNode root, int val){
    if(root.val > val && root.val < (ceilval){
        ceilval = root.val;
    }
    for (TreeNode node : root.children){
        Ceil(node, val);
    }
}
ceilval = (int) -1e9;
Syso (ceilval)
Ceil (root, 86)
```

Floor :-

```
Static void floor(TreeNode root, int val){
    if(root.val < val && root.val > floorval){
        floorval = root.val;
    }
    for(TreeNode node: root.children){
        floor(node, val);
    }
}
Static int floorval;
floorval = (int) 1e9;
floor(root, 86);
Syso(floorval);
```

Preceded & Succeeded



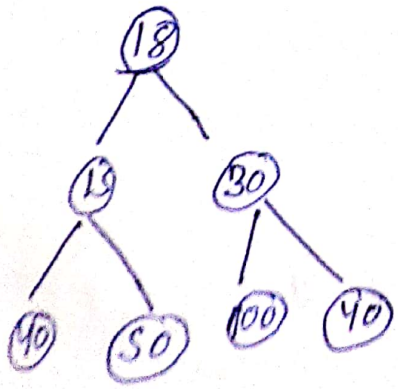

```
boolean flag = find(root, 50); → present or not

if(flag){
  PredandSucc(root, 50);
    Sysso(pred + "\n" + succ);
}

Static int pred = -1
//      //  succ = -1
//      //  flag = 0;

Static void PredandSucc(TreeNode root, int val){
    if(flag == 0 && root.val != val){
        pred = root.val;
  else if(root.val == val){
        flag = 1;
} else if(flag == 1){
```

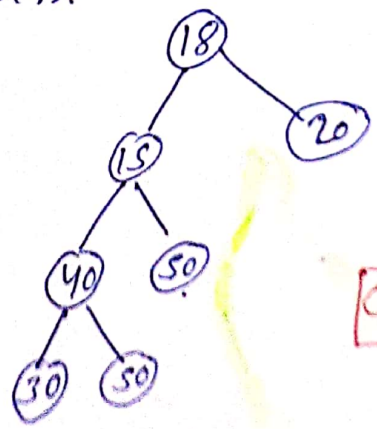
succ = root.val;
flag = 2;
}
for (TreeNode node : root.children) {
if (flag == 2) {
return;
}
predandsucc(node, val);
}
}

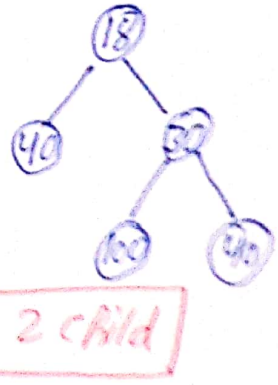
Binary Tree# (3 Types)

→ Full Binary Tree

If every node has 0 or 2 children. We can also say that except the leaf node all the nodes have 2 children.

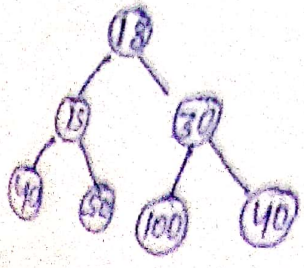


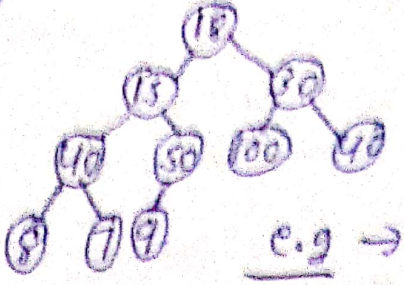




0 or 2 child

Complete Binary Tree: all the levels are completely filled except possibly the last level & the last level has all keys as left as possible.

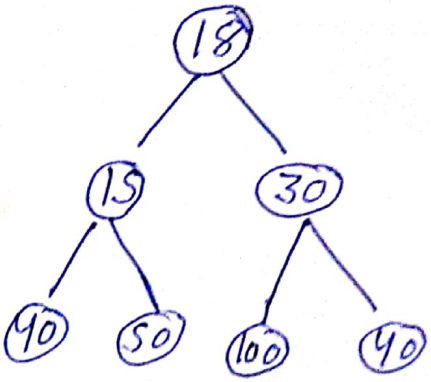


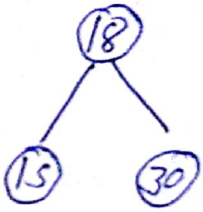


filled level by level
& filled from left

e.g → Binary Heap

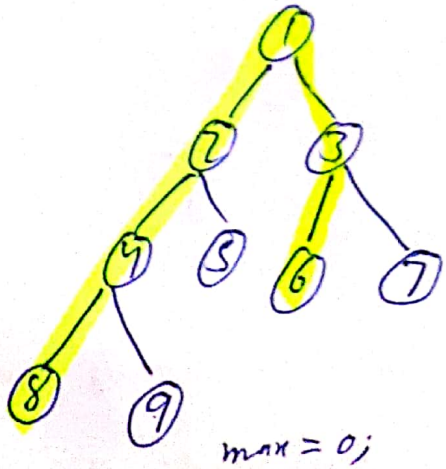
Perfect Binary tree :— A Binary tree is a Perfect Binary tree in which all the internal nodes have two children and all the leaf nodes are at the same level.





Construct the tree

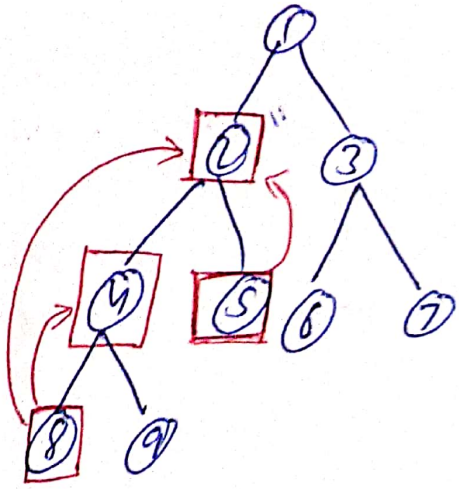
- display → Pre, Post, In, Level, Wave,
- Ceil, Floor, max, min, Size, height, Preceded, Succeded. all qe in one method by class variable.
- Max Diameter




```
max = 0;
{
    a = maxdiam(root.left);
    b = maxdiam(root.right);
    if((a+b+1) > max) {
        max = a+b+1;
    }
    if (a>b) return a+1;
    return b+1;
}
```

```
(PrevL + PrevR + 1) > max
    max = ____
if (PrevL > _R) {
    return ____
}
return R____
```

@ LCA (least Common Anchestor.

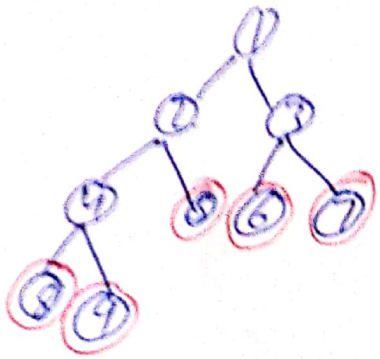


8,5 ⟶ 2
4,8 ⟶ 4
9,7 ⟶ 1

```
static boolean lca(TreeNode root, int a, int b){
    if(root==null) return false;
    boolean flag = False;
    if(root.val=a || root.val==b){
        flag = true;
    }
    boolean L_ans = lca(root.left, a, b);
    boolean R_ans = lca(root.right, a, b);
    if( L_ans && r_ans) || ( L_ans && flag) || (r_ans && flag)){
        System.out.println(root.val);
        return True;
    }
    if((!L_ans && r_ans) || (!r_ans && L_ans) || ((!r_ans && !L_ans)
                                                  && flag)) ret True.

return False
```

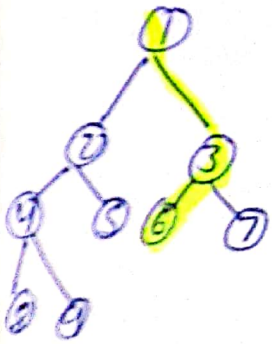
F T && flag return True
T F && flag // //
F F && flag // //

Remove leaf by Return :- (Pre, Post order)



```
if(root == null) || (root.left == null && root.right
                    == null)) return null;

root.left = removerLeaf(root.left);
root.right = removeLeaf(root.right);
return root;
```

Root Integer



8 → 1, 2, 4, 8

6 → 1, 3, 6

```
If(root == null) return
```

2 → Parameter root, p

return ArrayList.

root null → return → null Array

root = val return Array list with root

```
ArrayList L = fn call for left child
 //   R =  //      //   Right "
If L & R list size 0 return L
If L size > 0 add to L & return L
L.add(0, root.val);
return L;
```

Package in Java :-

```
Package bank;
class Account {
    Public String name;
}
Public class Bank {

}
```

```
import bank;
bank.Account account1 = new bank.Account ();
account1.name = "Customer1";
```

Inheritence in Java.

→ Single Level inheritance
→ multilevel //
→ hierarchical inheritance
→ hybrid inheritance

There is no multiple inheritance in Java we use Package in Java for the same.

Access Modifier are 4 types in Java 3 type in CPP

Public → Visible in the class or accessible in the Package through obj

private → Scope is within the class only even obj can't access it here we use //getters & setters

protected → everyone can access in current Package and in other Package Sub classes can access it

Default → Package private visible within Same Package Not on other. Declaration without keyword

```
int a;
```

Encapsulation :- Creating object and classes

```
Class A {
    String name;
}
Class B extends A {
        ---
        ---
    }
```

Data Hidding → is implemented using Access Modifier

Abstraction → Data hiding is the process of protecting members of class from unintended changes whereas, abstraction is hiding the implementation detail and show only important & useful parts to the user.

1 way → Abstract keyword use
Abstract class, member, function or method

2 way → use interfaces

```
Abstract class Animal {
    Abstract void walk();
}
class horse extends Animal{
    Public void walk() {
        Syso("walk on 4 legs");
    }
}
Class Chicken extends Animal{
    Public void walk() {
        Syso("walk on 2 legs");
    }
}
Public class oops {
    Public static void main(String args[]) {
```

```
Horse horse = new Horse();
horse.walk();
```

abstract → Concept

```
}
```

→ `Animal animal = new Animal();` run time error we cannot do so because it is of abstract type

```
animal.walk();
```

when we create an obj of derived class first of all base class constructor is called then derived class. this concept is called Constructor chaining in Java.

Properties of abstract class

- An abstract class must be declared with an Abstract keyword.
- It can have abstract & non abstract methods. (Pure & non-pure abstraction)
- It cannot be instantiated

Interfaces (multiple inheritance is done through it)

- All the fields in interfaces are public, static and final by default
- All methods are public & abstract by default
- A class that implements an interface must implement all the methods declared in the interface.
- Interface supports the functionality of multiple inheritance.

```
Interface Animal {
    int eyes = 2;
    void walk();   // we can also add Public keyword in front but by default it is Public Static & final;
}
class Horse implements Animal {
    Public void walk() {
        syso(_____);
    }
}
```

→ Here it is compulsory Public bcz it is class member

→ Single inheritance

```
Public class oops {
    Public static void main(String args[]) {
        Horse horse = new Horse();
        horse.walk();
    }
}
```

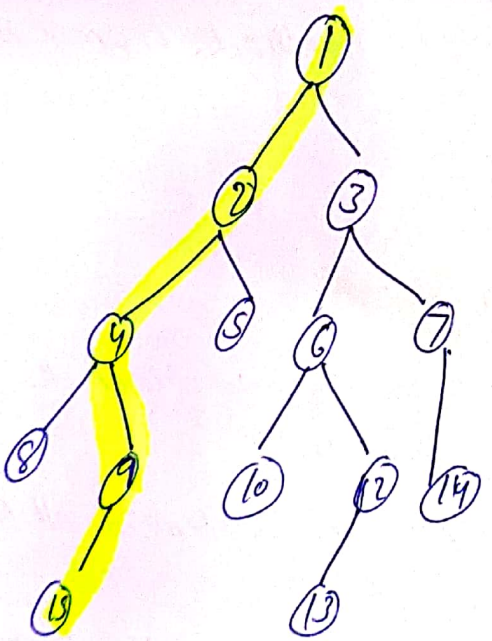
Class Horse implements Animal, Herbivore {

↳ multiple inheritance

}

Static keyword :- The particular member belongs to a type itself we'll create only one instance of that static member that is shared across all instances of the class.

Root To Node Path :-



15

return type ArrayList

If root null return empty ArrayList

If (root.val == val){

new ArrayList

add root to new ArrayList

return new arraylist;

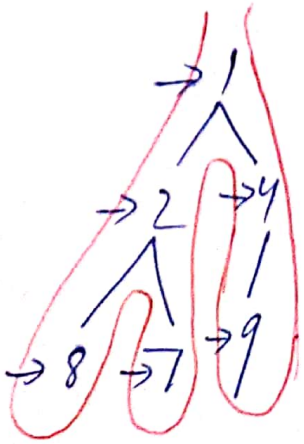
}

arraylist L = f^n call for left child

" R = " " right

```
if(( q r size = 0)   retur  L
if(L size > 0) {
      L. add(0, root)
       return L;
}
 ~~ret~~ r. add(0, root);
      return r;
```

BTree Pre order

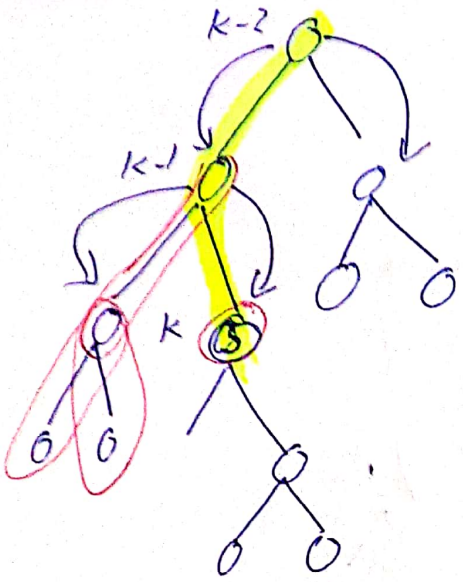



```
Static int i=0;
void preorde(TN root, int[] ans){
   if(root) == null) return;
   ans[i++] = root.val
   if(root.left != null) {
         preorder(root.left, ans);
   }
    // righ   // {
          // right;
   }
```

BTree post order

BTree inorder

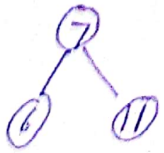
k-far



p=5

```
ArrayList<TreeNode> rtn = rootTONode(root,5);
int k = 2;
for(int i = rtn.size()-1; i>=0; i--){
    TreeNode blocker = i==rtn.size()-1 ? null : rtn.get(i+1);
    PrintKdis(rtn.get(i), k, blocker);
    k--;
}

Static void PrintKdis(TreeNode &root, int k, TreeNode blocker){
    if(root==null) return;
    if(k==0){
        Syso(root.val);
        return;
    }
    if(root.left!=blocker){
        PrintKdis(root.left, k-1, blocker);
    }
    if(root.right!=blocker){
        PrintKdis(root.right, k-1, blocker);
    }
}
```

Binary Search tree :-



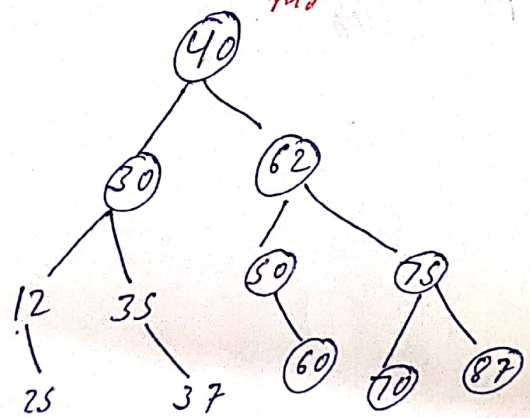
It is a type of binary tree

properties

1) All nodes of the left subtree are lesser
2) All nodes of the right subtree are greater.
3) left & right subtree are also BST
4) There are no duplicate nodes
5) In-order traversal is Ascending sorted Array.

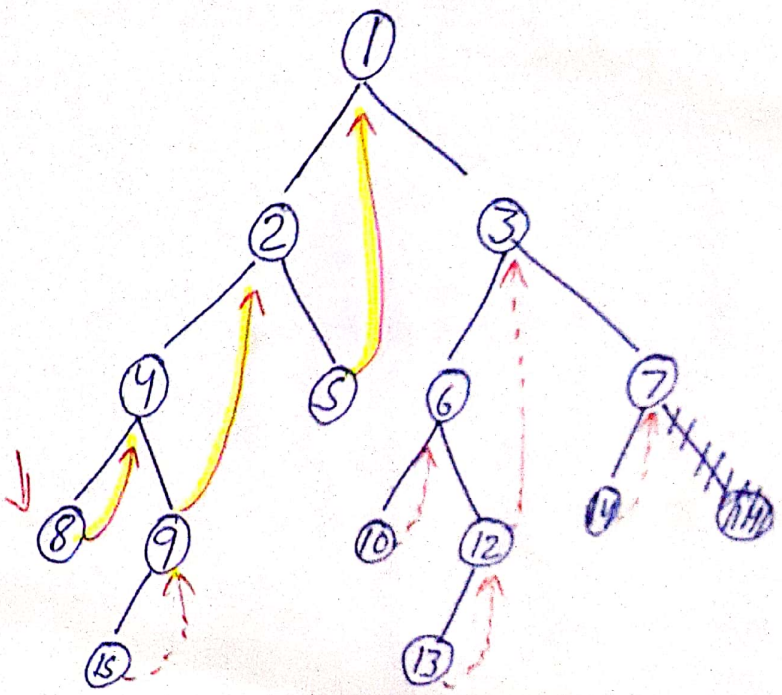
12, 25, 30, 35, 37, 40, 50, 60, 62, 70, 75, 87

mid




```
Static TreeNode construct(int[] arr, int l, int r){
    if(l>r) return null;
    int mid = l+(r-l)/2;        // mid = (l+r)/2;
    TreeNode node = new TreeNode(arr[mid]);
    if(root == null){
        root = node;
    }
    node.left = construct(arr, l, mid-1);
    node.right = construct(arr, mid+1, r);
    return node;
}
```

Morris Traversal :- using Morris Traversal, we can traverse the tree without using stack and recursion. Morris traversal is for inorder / preorder




```
while (curr != null){
    if (curr.left == null){
        print ------------> Pre order
        curr = curr.right
    }else
        temp = curr.left
        while (temp.right != null && temp.right != curr){
            temp = temp.right
        }
        if (temp.right == null){
            temp.right = curr;
            curr = curr.left;
        }else {
            temp.right = null;
            curr = curr.right;  ------------> print // in order
        }
}
```

not linked

- why we don't go for Post order

- Check BST is valid
- check BST is Balanced
- Find a element in Tree
- Add an element
- Remove an element
- Delete Node in a BST → 450 Leetcode

Artical Morris algo

Reason why we don't go for Post order in Morris Traversal

Hashmap Implementation :- (Data structure)

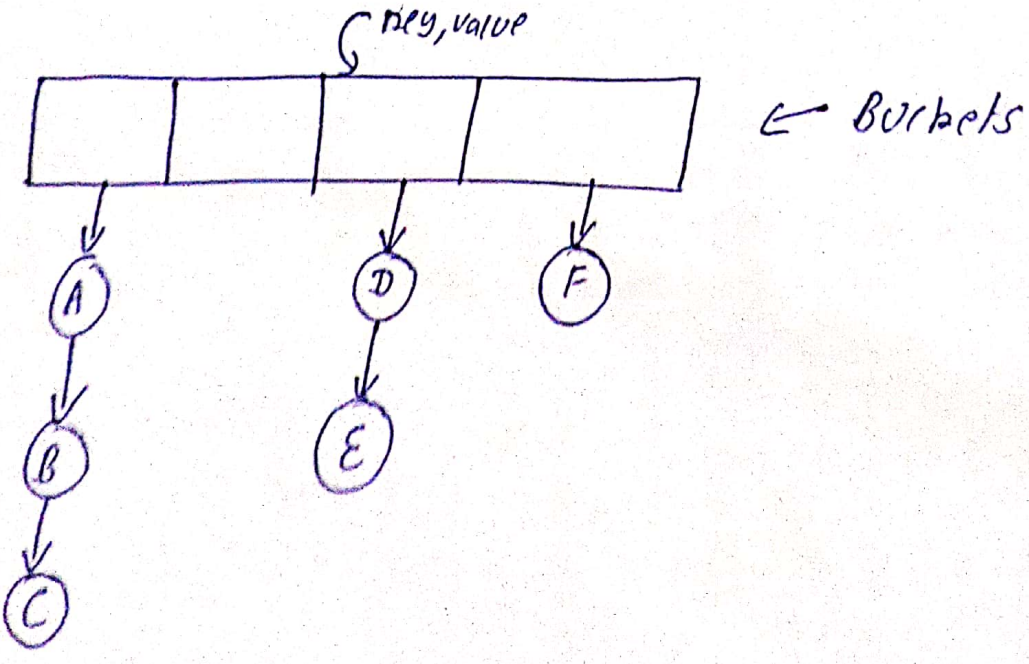
HashMap<K,V> It stores data in (key, value) pairs

If we try to insert duplicate key it will replace the element of corresponding key.

HashMap it known as hashmap because it use a technique called hashing. Hashing is a technique of converting a large string to small String that represent the same string. A short value helps in indexing and fast searches.

key features:-

* get
* put
* containkey (T&F) } O(1)
* remove
* size



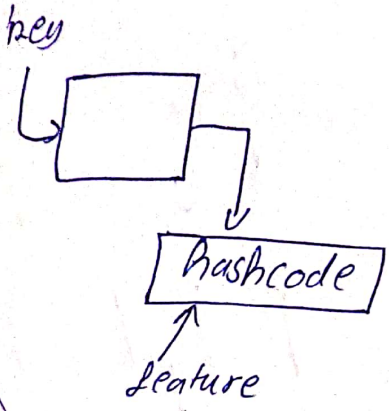
N → 4 Bucket.

n → no of elements

Load factor (λ) = $\frac{n}{N}$

$\frac{7}{4} \Rightarrow 1.8$

$\frac{9}{4} \Rightarrow 2.2$



Templating <T, V>

abstraction / hiding [Private]

[NoSql MongoDB]

Java use hashcode

Requirements :-

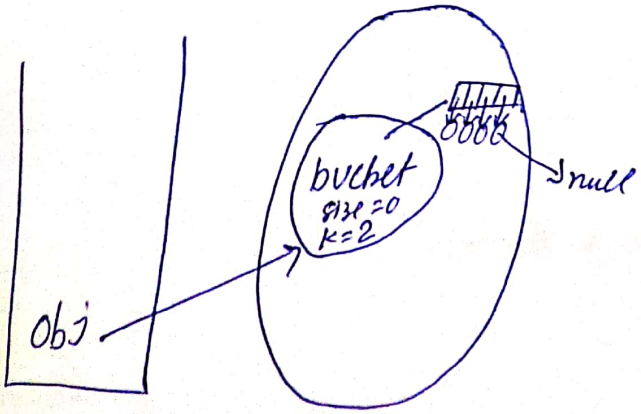
```
Class Node {
    int key;
    char value;
}
linkList<Node>[] bucket
k ← loading factor
```

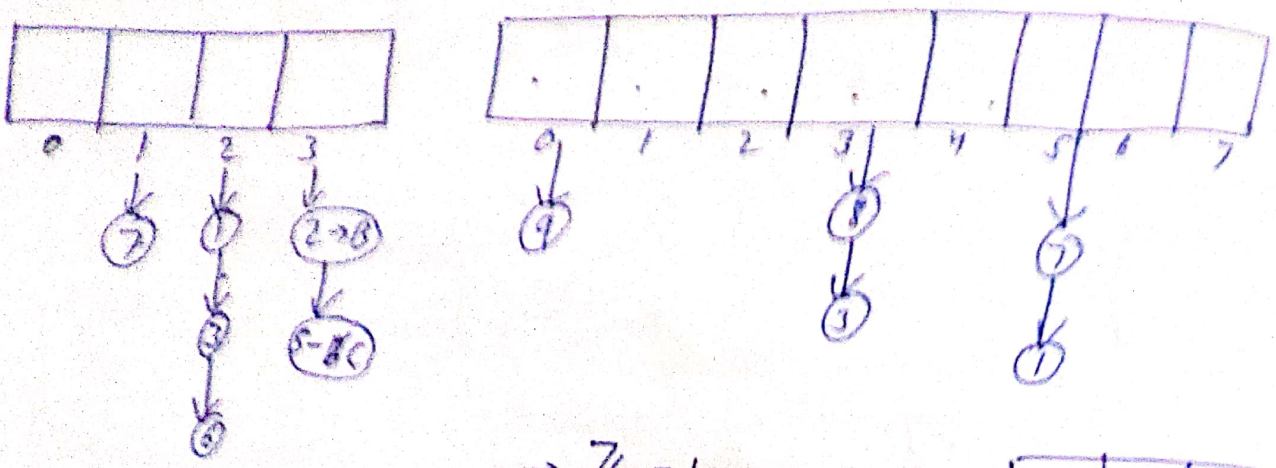
Time complexity :-

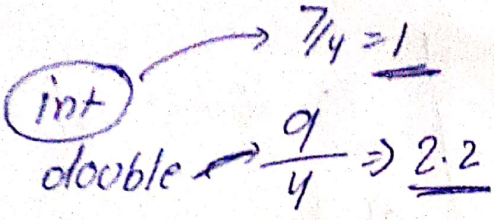
best (O(1))

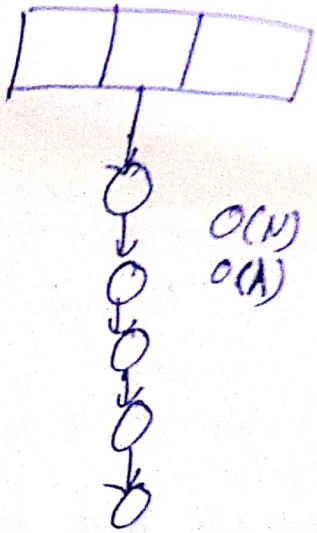
worst O(n)

```
Hashmap() {
    buck = new linkList[4];
    size = 0;
    k = 2;
    init();
}
```









O(N) = O(1)

1 ⇒ constant → O(1)

```
Static class HashMap<E
    Private class Node {
        int key;
        Char val;
      Nod(Int k, V v){
          // Constru
      }
    }
    Private LinkedList<Node>[] bucket;
    Private int size; // total no of keys.
    Private int k; // loading factor

    HashMap<V>() {
        init(4);
    }

    Private void init(int s2) {
        bucket = new LinkedList[s2];
        size = 0;
        k = 2;
```

```
    for(int i=0; i<bucket.length; i++){
        bucket[i] = new LinkedList<>();
    }
}
// generate hashcode
private int hcode(Integer key){
    int hc = key.hashCode();
    return Math.abs(hc)%bucket.length;
}
Public void Put(int key, char val){
    int hc = hcode(key);
    for(Node node: bucket[hc]){
        If(node.key = key){
            nod.val = val;
            return;
        }
    }
    Node newnode = new Node(key, val);
    bucket[hc].addlast(newnode);
    size++;
    double lamda = (size*1.0)/bucket.length;
    If(lamda > k){
        rehashing();
    }
}
Private void rehashing(){
    LinkList<Node>[] copybucket = bucket;
    init(bucket.length * 2);
```

```
for (int i=0; i< copyBucket.length; i++){
    for (Node node: copyBucket[i]){
        Put(nod.key, node.val);
    }
}
}
```

```
main
    HashMap obj = new HashMap();
    obj.put(1, 'A');
```

Map Map<K,V> Interface Map<K,V>

Map keys to values No duplicate keys each key has one
one value.

Map is interface

Class HashMap<K,V>:

→ Hashmap a class which implements the map interface.

→ Permits null values and null keys. & ~~Permits~~ unsynchronized
↳ main diff b/w Map & HashMap.

→ Constant time performance

Q Find Frequency of occurence of element

[8, 2, 1, 0, 36, 1]

8 → 1
2 → 1
1 → 2
0 → 1
36 → 1

java.util.HashMap

```
Static void freq(){
   create HM
  arr ————————
for( int i: arr){
      if(hm.containskey(i)){
          hm.put(i, hm.get(i)+1);
      }else{
          hm.put(i,1);
      }
}
ArrayList<Integer> keys = new ArrayList<>(hm.keyset());
for(int key:keys){
      syso(key+"->"+hm.get(key));
}
}
```

Q arr = [1, 2, 3, 4, 1, 2, 3]

brr = [3, 2, 8, 9, 4, 1, 3]

[1, 2, 3, 4]

1 → 2	3 → 2
2 → 2	2 → 1
3 → 2	8 → 1
4 → 1	9 → 1
	4 → 1
	1 → 1

(key, mathmin(HM.get(key), HM.get(key)))

key set

```
Array list arr
for i< bucket.length{
    for Node: bucket[i] {
        arr.add(node.key);
    }
}
return arr
```

contain key

```
int hc = hcode(key);
for ( Node: bucket[hc]) {
    if (node.key == key) {
        return true
    }
}return False;
```

value set

Same as key set

node.val

char get

```
Int hc = hcode(key);
for(node: bucket[hc]) {
    if (node.key == key) {
        return node.val;
    }
} return "";
```

remove

```
int hc = hcode(key);
for node : bucket[hc]
    if node.key = key
        bucket[hc].remove(node);
        size--;
        return;
    }
}
return;
```

Intersection :-

i/p ⇒ {1,2,0,4}
{1,2,3,4,8,9}

o/p ⇒ {1,2,4}

```
int arr = { }
int brr = { }
create hm1 & hm2 & ans  ← HashMap
if( arr.len < brr.len){
    for ( i< arr.len){
        if (hm1.containsKey(arr[i]) && hm2.containsKey(arr[i]))
            ans.put(arr[i], arr[i]);
        }
    }
}else{
    same for brr
}
```

RM

```
ArrayList<Integer> keys = new ArrayList<>(ans.keySet());
for ( i : keys)
    Syso (i)
```

Q # Find largest consicutive number

arr = {15, 3, 14, 2, 13, 5, 8, 7, 1, 12, 16, 9} ← i/p

findLarCon(arr); o/p → {12, 13, 14, 15, 16}

1) hm create of Array

2)
```
for( int v: arr){
  If (hm. containsKey (v+1)){
        hm.put (v+1, false);
  }
}
```

3
```
int gsn = -1      // global seq
int gmlen = 1     // global max len.
```

4
```
ArrayList<Integer> keys = new ArrayList<>(hm.keySet());
// O(N)

for ( int key: keys){
    If(hm. containsKey(key)){
        int csp = key;
        int cmlen = 1;
        while (hm. containsKey(csp+ cmlen)){
            cmlen++;
        }

        If( cmlen > gmlen){
            gmlen = cmlen;
            gsn = csp;
        }
    }
}
Sysо ("Max size of consicutive no is "+ gmlen);
```

```
for (int i=0 i<gmeen ; i++) {
    syso (gsn + i + "==>");
}
```

arr = [15, 3, 14, 12, 13, 5, 8, 7, 1, 12, 16, 9]

15 +	8 +	[1 2 3]	LSt
3 +	7 −		LSp=5
14 +	1 −		cmL=1
2 +	12 −	[7, 8, 9]	while()
13 +	16 +		
5 −	9 +	[12, 13, 14, 15, 16]	

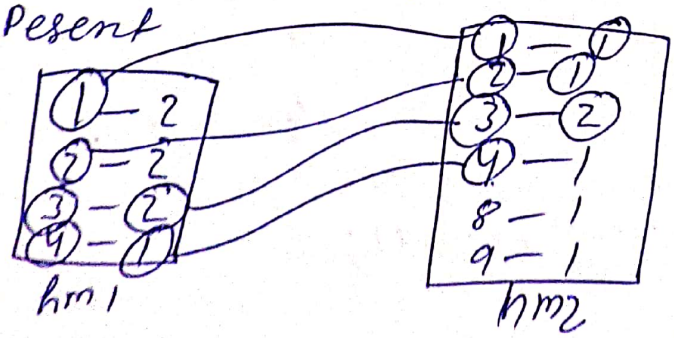
No of occurrence in Intersection

```
arr = {1, 2, 3, 4, 1, 2, 3}
brr = {3, 2, 8, 9, 4, 1, 3}
```

o/p
1 — 1
2 — 1
3 — 2
4 — 1

Ans →

Create hm by counting the key present



Choose min occ

HM ans.

```
If (arr.len < brr.len) {
    for (i : arr) {
        If (hm2.containsKey(i)) {
            ans.put (i, math.min(hm1.get(i), hm2.get(i)));
        }
    }
} else one of brr.
```

Q Total Max no of K-Sum Pairs :-

tar = a + b

a = tar - b

<> hm <>

arr = {1, 5, 7, -1, 5}

```
int count = 0
for (int b : nums) {
    int a = k - b;
    if (hm.containsKey(a)) {
        syso("[" + a + "," + b + "]");
        count++;
        if (hm.get(a) > 1) {
            hm.put(a, hm.get(a) - 1);
        } else {
            hm.remove(a);
        }
    } else {
        if (hm.containsKey(b)) {
            hm.put(b, hm.get(b) + 1);
        } else {
            hm.put(b, 1);
        }
    }
}
syso("Total no of pair" + count);
```

Consider unique pair

In case of non unique pair method is same bu we don't need to remove from hashmap and not need to count freq of occ

If Present a pair Possible print & put b ;

else put b

No of Subarrays having sum exactly equal to k

```
PSum = arr[i]
if (hm.containsKey(PSum - tar))
    count += hm.get(PSum - tar)
        + (Psum);
```

Priority queue :- A Priority queue is an abstract datatype, every element has some priority. The Priority elements are removed from the Priority queue. therefore all the elements are either arranged in an ascending or descending order.

Operation :-
1) Insertion
2) Deletion
3) Peep

Type :-
1) Ascending
2) Descending

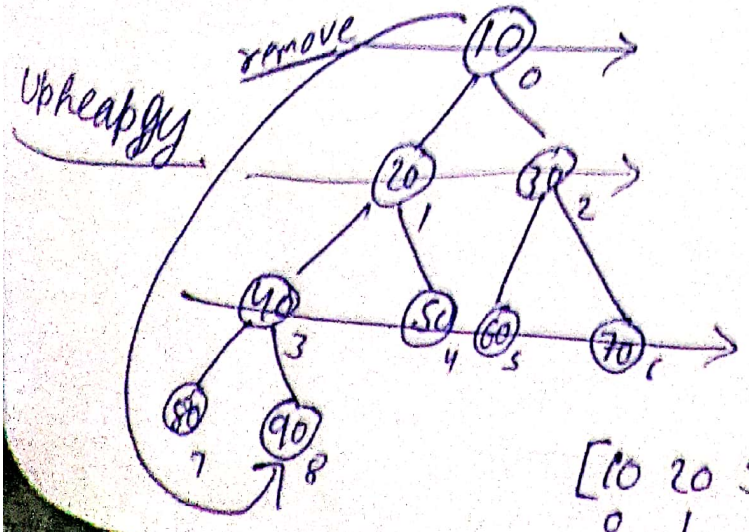
Javap util.java.util.PriorityQueue

Can be implemented through :-

Arrays
LinkedList
Heap data structure
Binary search tree

Applications :-

→ CPU scheduling
→ Graph algorithms like Dijkstra's short path. ~~&~~ Prim's minimum spanning tree, etc.
→ Stack Implementation
→ All queue applications where priority is involved
→ Data compression in Huffman code.



* HOP heap order property
* CBT cmp binary tree

min / max

[10 20 30 40 50 60 70 80 90]
0 1 2 3 4 5 6 7 8

$li = 2 * pi + 1$
$ri = 2 * pi + 2$
} left / Right child

$pi = \frac{(ci - 1)}{2}$

upheapify → pi

Operation	Complexity
add	O(logn)
remove	O(logn)
get	O(1)
size	

downheapify → li, ri

Priority que Implementation :-

```
Static class PriorityQueue1{
    Private ArrayList<Integer> tree;
    Priority-Queue1(){
        tree = new Arraylist<>();
    }

    get →    tree.get(0);
    size →   tree.size();

    add →  add(int val){
     tree.add(val);
     int ci= tree.size()-1;
     Upheapify(ci);
    }
    UpHeapify(int ci){
     if(ci==0){
        return;
     }
     int pi = (ci-1)/2;
```

```
if (tree.get(pi) > tree.get(ci)){
        swap(pi, ci);
        UpHeapify(pi);
}
}
```

tree.set(Pi, Val ci)

```
remove() {
    swap(0, last);
    tree.remove(last);
    downheapify(0);

private void downheapify(int pi){
        int li = 2*pi+1;
        int ri = 2*pi+2;
        int mini = pi;
    if(li< tree.size() && tree.get(li) < tree.get(mini)) {
            mini = li;
    }
    if(ri< tree.size() && tree.get(ri) < tree.get(mini)){
            mini = ri;
    }
    if(mini != pi) {
        swap(mini, pi);
        downheapify(mini);
    }
}
    display tree(){
        sysso(tree);
    }
    public void clear(){
        tree = new ArrayList<>();
    }
}

    main()
PriorityQueue1 pq = new PriorityQueue1();
```

Lamda fn change from high order PQ to low PQ & vice-versa

1) Collections.reverseOrder()
2) Lamda fn → ((a,b) → {return b-a;})
3) multibly with -1

→ Poll() → ~~root element~~ remove the top element & return

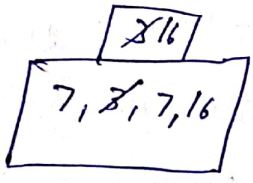
Q K largest Element in PQ O(n log k)

k=3 arr [7,3,7,-8,16,15,8,9]

for (i:arr)

if (pq.size < k)
pq.add(arr[i]);
} else if (pq.Poll() < arr[i]) {
pq.add(arr[i]);
}

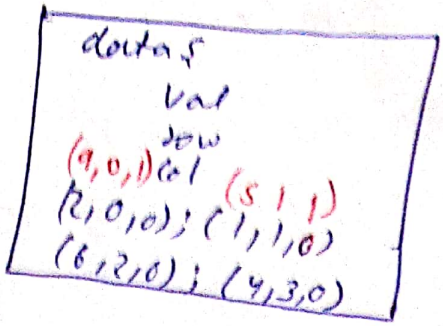
while (pq.size > 0) {
Syso (pq.remove() + " ");
}



Q K - Sorted list :-

[[2 9 19 25 35]
[1 5 7 12]
[6 14 49 54 60 77]
[4 12 20]]

(n x m) log k



```
class data implements Comparable {
    int va ; int row ; int cd;
    -> Const-
    -> Public int compareTo (data o)
            return this.val - o.val;
    }
}
```

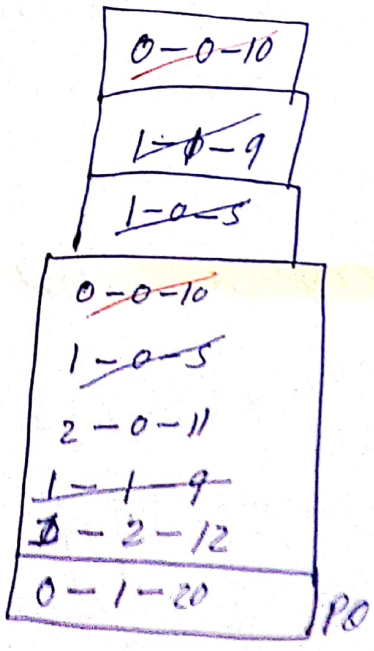
[1, 2, 4, 5, 6, 7, 9, 12, 12, 14, 19, 20, 25, 35, 49, 55

Comparable – vs – Comparator in. java – Pfg

@ Merge k sorted lists

0	10 →	20	30	40	
1	5	9	12	18	32
2	11	15	17		

[5, 9, 10]



Comparable Interface :-

1) Interface.

2) Contract of function (→ provide body to f^n)

3) Interface only has name & signature of f^n.

4) Class implement interface (call will provide body for all f^n's named in interface.

5) Comparable → Interface already in java which contains
$1\ f^n$ compareTo ( )

6) PQ use it because it depends on comparable. because it want
7) to compare obj. ~~and~~

PQ depends Comparable

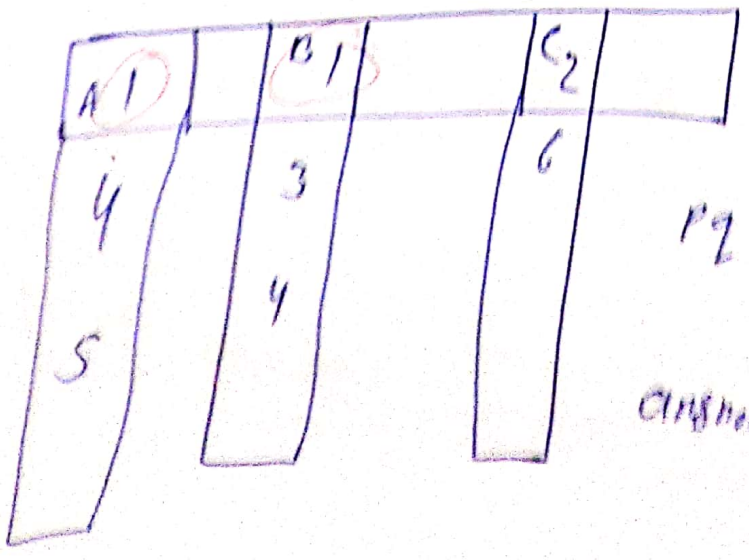
```
Cob = (C)p1;
Cob = (C)p2;
If (o1.CTo(o2) <0)
```

C → Comparable

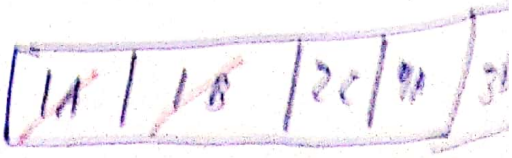
-ve val obj1 < obj2
+val obj1 > obj2
0 → equal
this.obj - 0

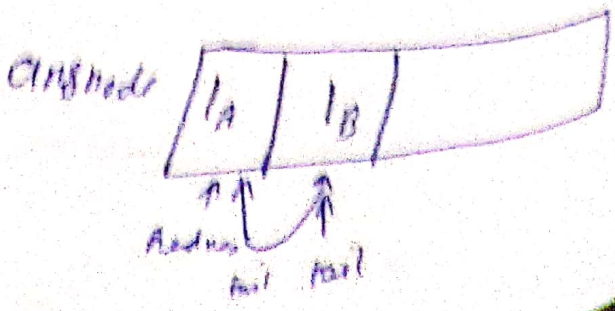
⑩ Merge k Sorted list (n log k)

[[1,4,5], [1,3,4], [2,6]]



node → obj
data class
head = null ans
tail = null ans



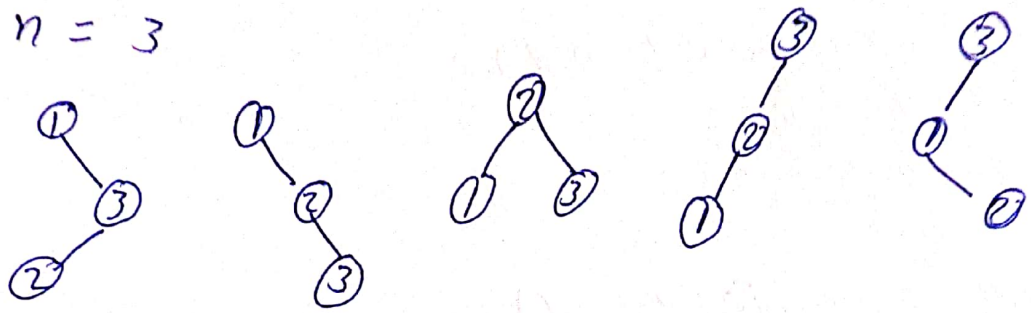


```
Static class data implements Comparable<data> {
    int val;
    ListNode node;
    data (int val, ListNode node) {
        this.val = val;
        this.node = node;
    }
    Public int compareTO (data d) {
        return this.val - d.val;
    }
    Public ListNode mergeklists (ListNode[] Lists) {
        If (lists.length == 1) return lists[0];
        PriorityQueue<data> pq = new PriorityQueue<>();
        for (ListNode node : lists) {
            If (node == null) continue;
            data noded = new data (node.val, node)
            Pq.add (noded);
        }
// using for ans creation
    ListNode head = null;
    ListNode tail = null;
    While (Pq.size() > 0) {
        data nodertm = Pq.remove();
        ListNode ansnode = new ListNode (nodertm.val);
        If (head == null) {
            head = ansnode;
            tail = ansnode;
        } else {
            tail.next = ansnode;
            tail = tail.next;
        }
    If (nodertm.node.next != null) {
        data noded = new data (nodertm.node.next.val, nodertm.node.next);
        pq.add (noded)
    }
}
return head;
```

best sorting algo nlogn quick, heap, merge,

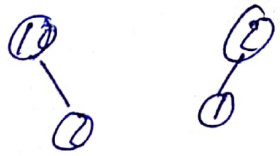
Q Unique Binary search trees.

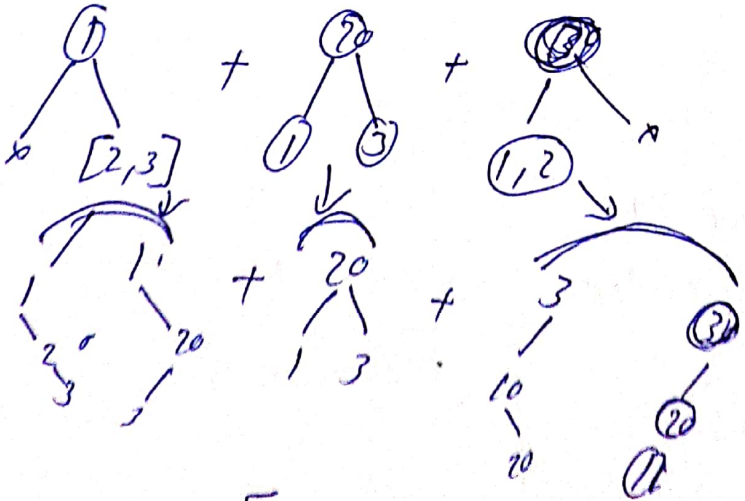
n = 3



O/P → 5

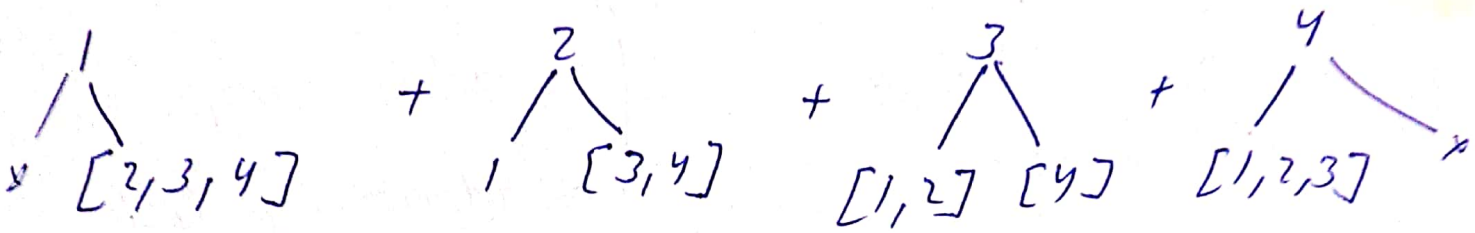
n	BSTs
0	1
1 [1]	1
2 [1,2]	2
3 [10,20,30]	5





5

4 [1, 2, 3, 4]



1.5 + 1.2 + 2.1 + 5.1

⇒ 14

```
int dp = new int [n+1];
    dp[0] = 1;
    dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        int l = 0
    int r = i-1;
     while ( l <= i-1) {
         dp[i] += dp[l] * dp[r];
             l++;
             r--;
      }
  }
return dp[n];
```

Graph → Tree
↖ ↙ Loop✗
→ Loop → ✓

Type
→ adjacency Matrix
→ adjacency list

Terminology :-
Directed → Unweighted / Weighted
Undirected → Unweighted / Weighted
→ bidirectional
unidirectional

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

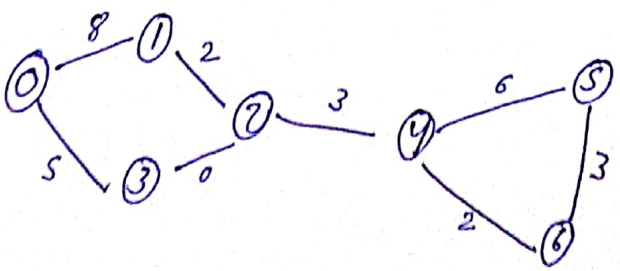
Self loop
Self direction

edges → Out degree
→ in-degree

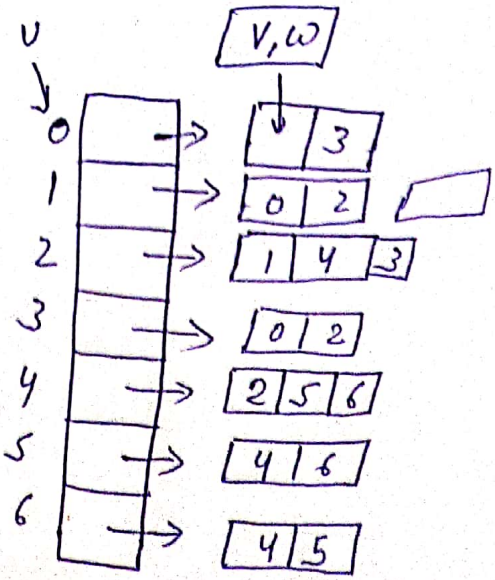
u —w→ v

Vertices
Degree of vertices → Indegree → no of edges incoming
→ Outdegree → no of edges outgoing

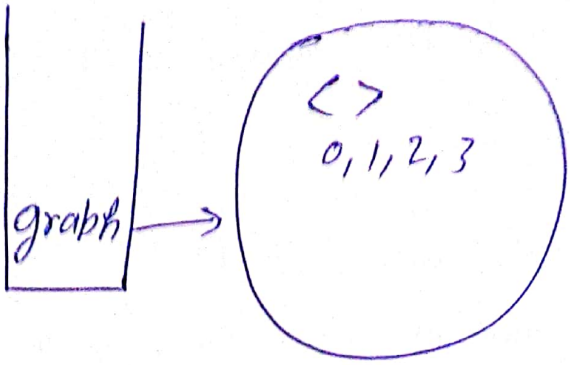
total no of max edges → n×(n-1)



edges = {
{0,1,6}
{0,3,5}
{1,2,2}
{3,2,0}
{2,4,3}
{4,5,6}
{4,6,2}
{5,6,3}
}



AL<AL<edge>> graph



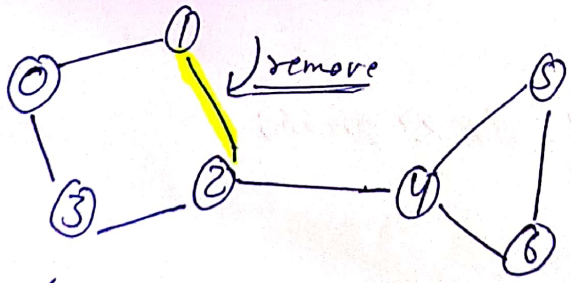
Construct By

AL<AL> 3X3 ⇒9
HM<HM> AL<LL>
LL<LL> AL<HM>

```
Public class construction{
    static class edge{
        int v;
        int w;

        edge(int v, int w){
            this.v = v;
            this.w = w;
        }
        edge(){}
    }
    Static ArrayList<ArrayList<edge>> graph;

    Pub main(){
        edges {_ _ _ _};
        vertices = 7;
        make(edges, vertices);
        displayGraph();
    }

    make(int[][] edges, int vertices){
        graph = new ArrayList<>();
        for(int i= 0 ; i< vertices; i++){
            graph.add( new ArrayList<edge>());
        }
```

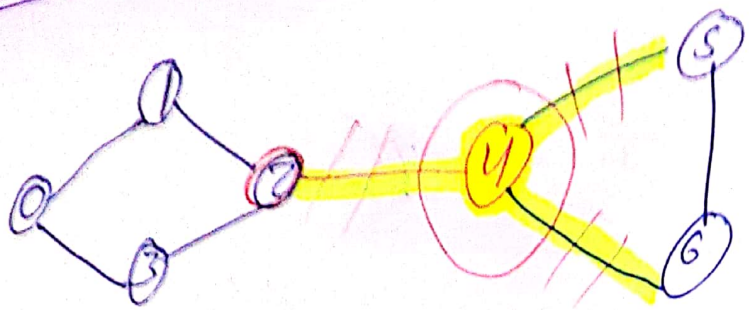
```
        for(int[] arr:edges){
            addEdge(arr[0], arr[1], arr[2]);
        }
    }

    addEdge(int u, int v, int w){
        graph.get(u).add(new edge(v,w));
        graph.get(v).add(new edge(u,w));
    }

    displayGraph(){
        for(int i=0; i<graph.size(); i++){
            Syso(i+"==>");
            for(edge ed: graph.get(i)){
                Syso("["+ed.v+", w-"+ed.w+"];");
            }
            Syso();
        }
    }
```

Remove edge




```
for(int i=0; i<graph.get(u).size(); i++){        (u, v)
    if(graph.get(u).get(i).v==v){
        graph.get(u).remove(i);
        break;
    }
}
for(int i=0; i<graph.get(v).size(); i++){
    if(graph.get(v).get(i).v==u){
        graph.get(v).remove(i);
        break;
    }
}
```

Remove vertex :-



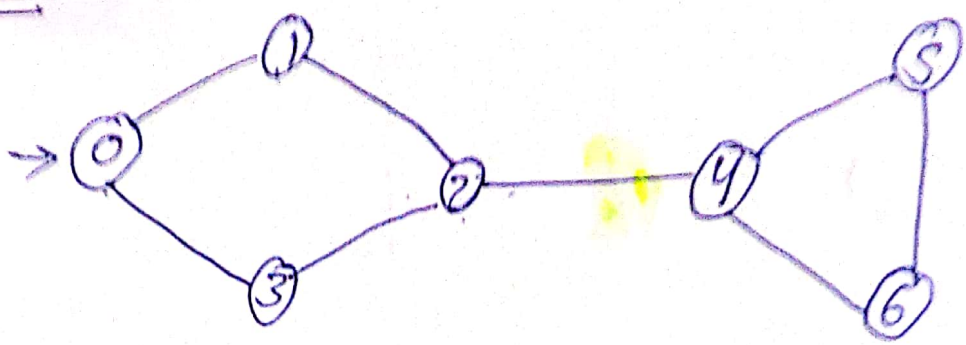

```
public void removeVertex(int ver){
    ArrayList<edge> edges = graph.
            get(ver);
    for(edge ed: edges){
        ArrayList<edge> rmed =
            graph.get(ed.v);
        for(int i=0; i<rmed.size(); i++){
            if(ver==rmed.get(i).v){
                rmed.remove(i);
                break;
            }
        }
    }
    graph.set(ver, new ArrayList<
            edge>());
}
```

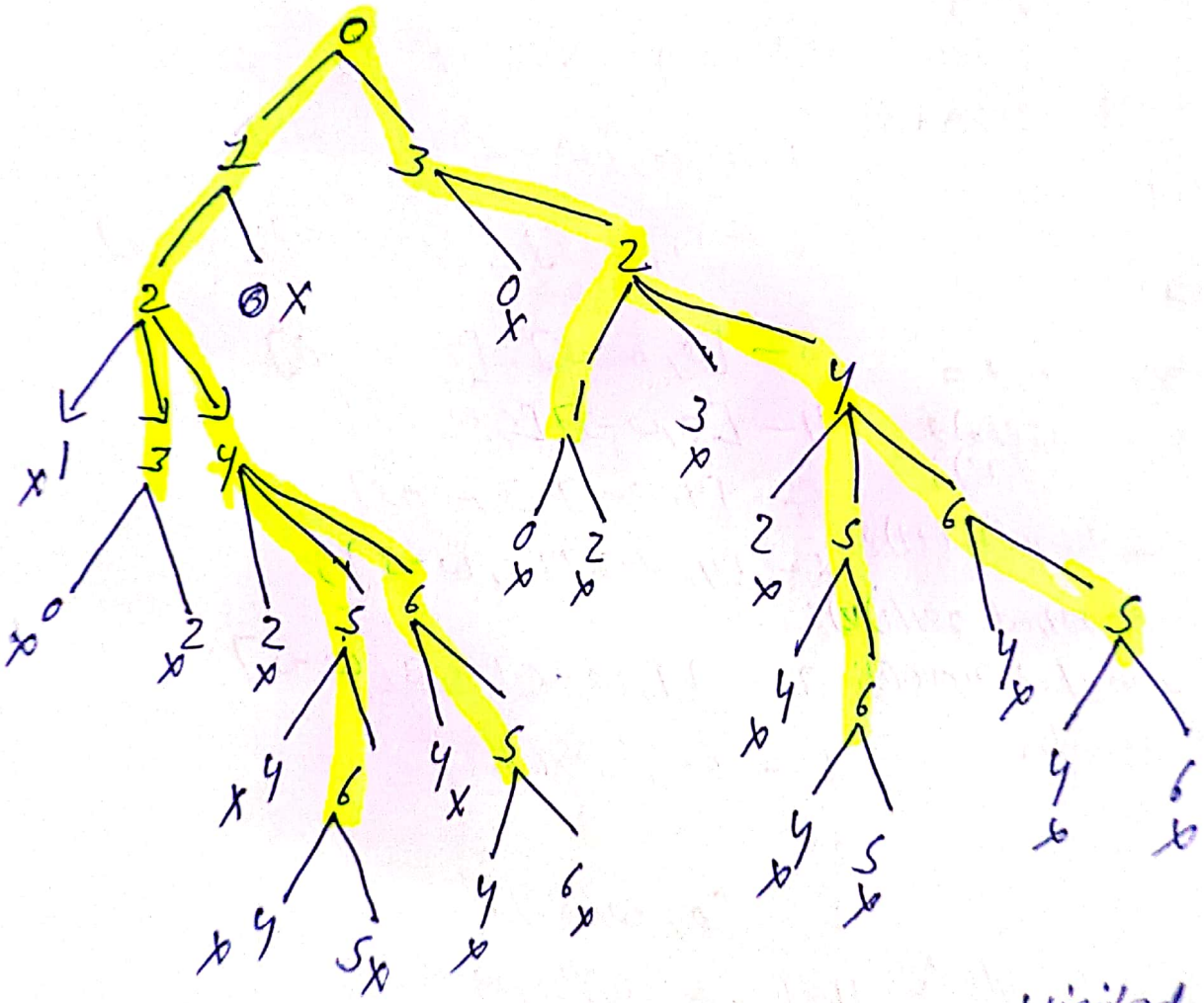
0 → 1, [w-6] [3, w-5]
1 — 0 [w-6]; [2, w-2];
2 — [1, w-2] ; [3, w-0]; [4, w-3]
3 — [0, w-5]; [2, w-0]
4 — [2, w-3] [5, w-6] [6, w-2]
5 — [4, w-6] [6, w-3]
6 — [4, w-2] [5, w-3];

---

0 — [1, w-6] [3, w-5];
1 — [0, w-6];
2 —
3 — [0, w-5];
4 — [5, w-6]; [6, w-2];
5 — [4, w-6]; [6, w-3];
6 — [4, w-2]; [5, w-3];

DFS





Visited

0, 1, 2, 3

1 0 → 1 → 2 → 3

2 0 → 1 → 2 → 4 → 5 → 6

3 0 → 1 → 2 → 4 → 6 → 5

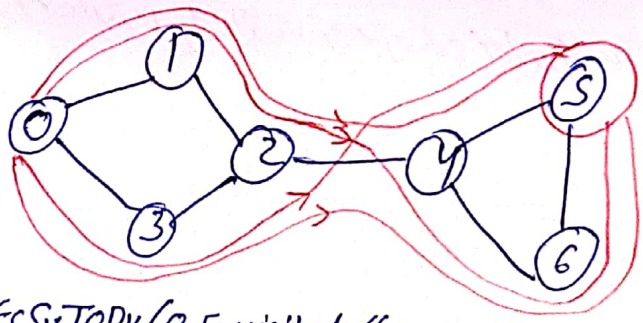
4 0 → 3 → 2 → 1

5 0 → 3 → 2 → 4 → 5 → 6

6 0 → 3 → 2 → 4 → 6 → 5

```
static void dfs(int cv, boolean[] visited, String psf){
    visited[cv] = true;
    boolean flag = false;

    for(edge ed: graph.get(cv)){
        if(!visited[ed.v]){
            flag = true;
            dfs(ed.v, visited, psf+ed.v+"==>");
        }
    }

    if(!flag){
        System.out.println(psf);
    }
    visited[cv] = false;
}
```

DFS from Starting to Destination vertex

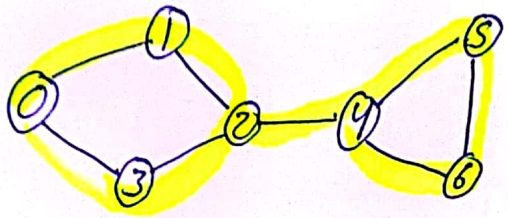


(5)

0 → 1 → 2 → 4 → 5
0 → 3 → 2 → 4 → 6 → 5
0 → 1 → 2 → 4 → 6 → 5
0 → 3 → 2 → 4 → 5

```
dfsSvToDv(0,5,visited,"0==>");
static void dfsSvToDv(int cv, int dv, boolean[] visited, String psf){
    visited[cv] = true;

    for(edge ed: graph.get(cv)){
        if(ed.v == dv){
            syso(psf+dv);
        }else if(!visited[ed.v]){
            dfsSvToDv(ed.v, dv, visited, psf+ed.v+"==>");
        }
    }
    visited[cv] = false;
}
```

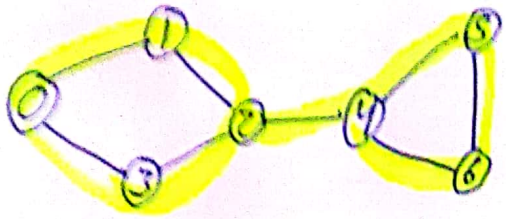
Find cycle



findCycle(0, -1, visit, visited)); current vertex, previous vertex

```
static boolean findCycle(int cv, int pv, boolean[] visit, visited){
    visited[cv] = true;
    visit[cv] = true;

    boolean flag = false;

    for(edge ed: graph.get(cv)){
        if(visited[ed.v] && ed.v != pv){
            return true;
        }
        if(!visited[ed.v]){
            flag = flag || findCycle(ed.v, cv, visited, visited);
        }
    }
    visited[cv] = false;  (X)
    return flag;  (X)
}
```

Find Cycle



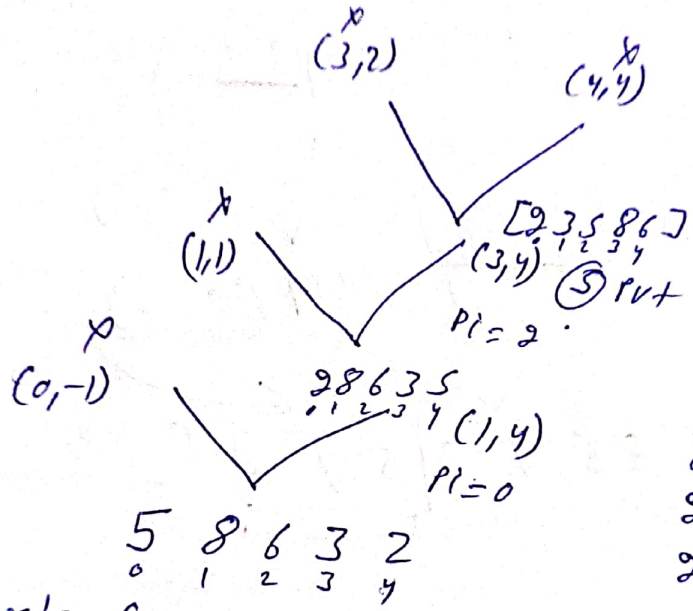
findCycle(0, -1, visit, visited)); current vertex, Previous vertex

```
static boolean findCycle(int cv, int pv, boolean[] visit, visited
    visited[cv] = true;
    visit[cv] = true;
    boolean flag = false;
    for(edge ed: graph.get(cv)){
        if(visited[ed.v] && ed.v != pv){
            return true;
        }
        if(!visited[ed.v]){
            flag = flag || findCycle(ed.v, cv, visited, visited);
        }
    }
    visited[cv] = false;
    return flag;
}
```

Quick Sort

Worst case → $O(n^2)$

Best case ⇒ $2T(n/2) + O(n)$



Pi = 3

Pi = 2

(5)

(1,4)

(2) 0

Pivot

(0,4)

Partition

5 8 6 3 2
0 1 2 3 4

2 8 6 3 5

2 3 6 8 5

2 3 5 8 1

2 3 5 6 8

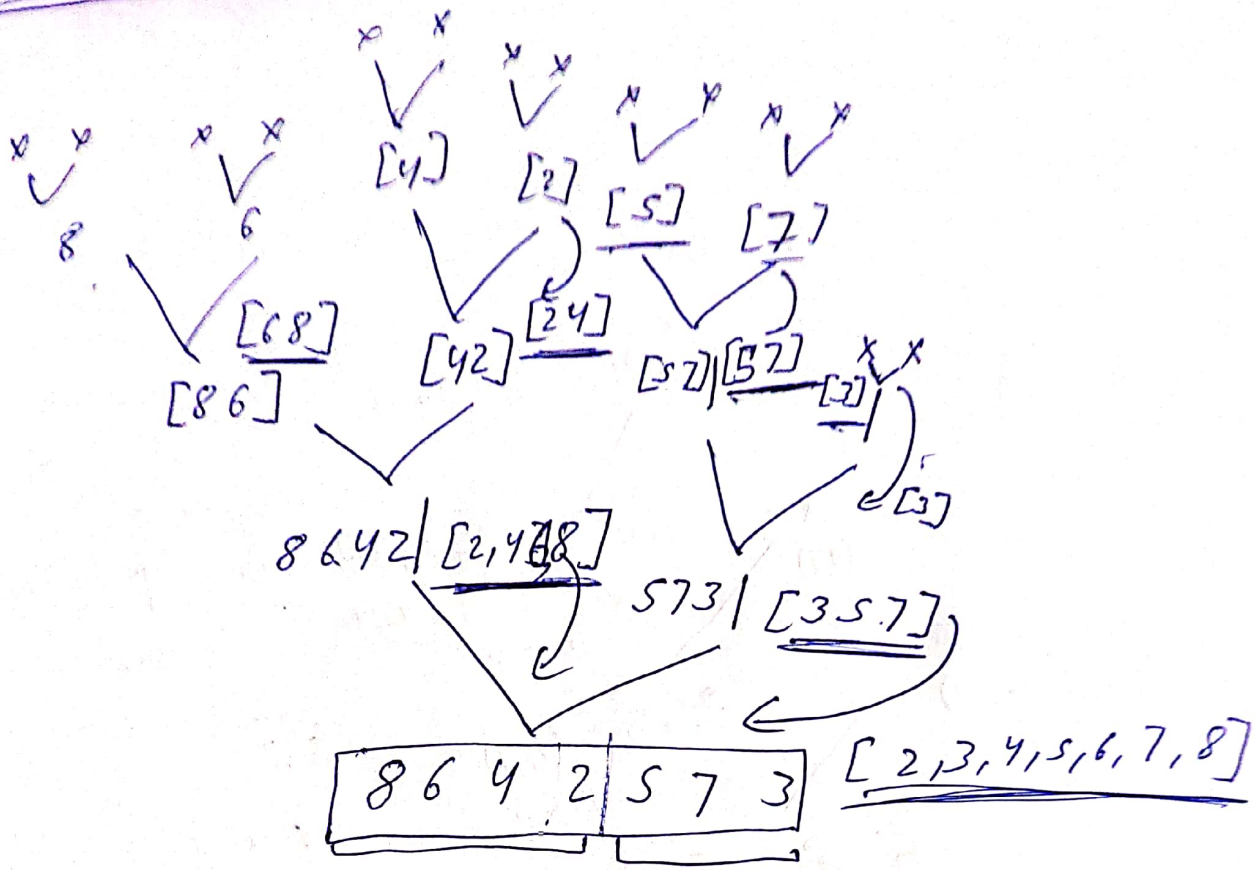
high

```
quickSort (arr, low, high){
if (low >= high) return
int pivot = arr[arr.length-1];
int Pi = Partition (arr, pivot, low, high);
 quickSort (arr, low, Pi-1);
 quickSort (arr, Pi+1, high);
}
Partition ( arr , pivot , low, high){
   int i= low; int pivot = arr[high];
   int j= low;
   while ( i<= high){
      if (arr[i] > pivot){
         i++;
      } else {
           swap(arr, i, j);
            j++; i++;
      }
   return j-1;
```

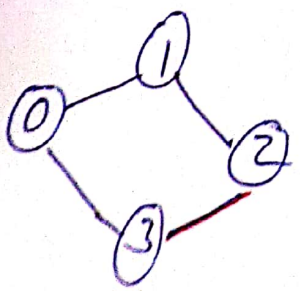
pivot 2 i=0 j=0

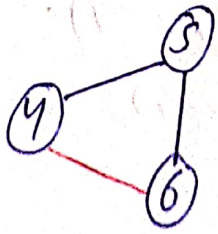
Replace with 4

Merge Sort :-



Cycle in disconnected Graph





True

False

```
boolean flag = false;
for(int i=0; i<graph.size(); i++){
    if(!visited[i] && !flag){
        flag = findCycle(i, -1, visited);
                    [src, prv vertex]
    }
}
sysO(flag);
```

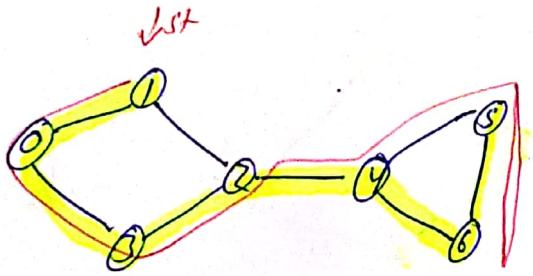
```
for (edge i : ed){
    if(iv == st){
        syso(ans + "*");   // print cycle & path
        return;
    }
}
syso(ans);                 // print path
return;
}

visited[sc] = true;
ArrayList<edge> ed = graph.get(sc);
for(edge i : ed){
    if(!visited[i.v]){
        hamiltonian cycle(st, i.v, count+1, visited, ans + "==>" i.v);
    }
}
visited[sc] = false;
}
```

findCycle (cv, pv, visited)

```
{ visited[cv] = true;
  boolean flag = false;
  for(edge ed : graph.get(cv)){
      if(visited[ed.v] && ed.v != pv){
          return true;
      }
      if(!visited[ed.v]){
          flag = flag || findCycle(ed.v, cv, visited);
      }
  }
  return flag;
}
```

HamiltoniumPath

1 → 0 → 3 → 2 → 4 → 6 → 5
1 → 0 → 3 → 2 → 4 → 5 → 6



Approach

→ we are going to visit all the vertices one by one using dfs when all the nodes are visited then we have to stop and print and which we are generating along fn calls to check all the nodes are visited we compare vertices size with count variable.

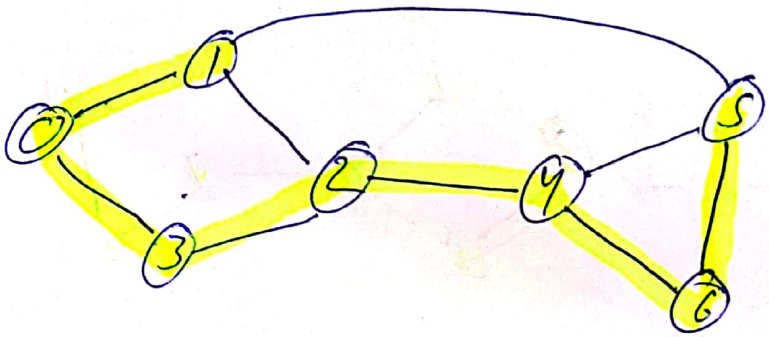
```
if( count == graph.size()){
    sysout(ans);
}   return;

visited[sc] = true
ArrayList ed = graph.get(sc);
for( edge i : ed ){
    if(!visited[i.v]){
    make a call
```

```
        hamiltonianPath(iV, count+1, visited, ans+"=>"+iV);
    }
}
visited[sc] = false;
```

Hamiltonian Path & Cycle :-

is an undirected graph is a path that visits each vertex exactly once.

A Hamiltonian cycle is a Hamiltonian Path such that there is an edge from the last vertex to the first vertex of the Hamiltonian Path.



Cycle

Hamiltonian cycle is always Hamiltonian Path but not vice versa.

Approach :-

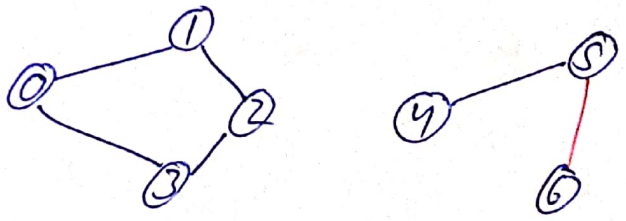
Visit vertices from starting vertex until all the vertices are visit and at last check is there a path to the starting vertex.

```
Static void HamiltoniumCycle(int st, sc, count, visited, ans){
    If( count == graph.size {
        ArrayList cl = graph.get(sc);
```

```
for (edge i : ed) {
    if (i.v == st) {
        syso(ans + "*");    // print cycle & path
        return;
    }
}
syso(ans);                  // print path
return;
}

visited[sc] = true;
ArrayList<edge> ed = graph.get(sc);
for (edge i : ed) {
    if (!visited[i.v]) {
        hamiltonian cycle (st, i.v, countEl, visited, ans + "==>" i.v);
    }
}
visited[sc] = false;
}
```

Get Component Count (gcc)

→ Given an undirected graph g, the task is to print the number of connected components in the graph.



Ans 3

Ans — 2

```
Static void masterGcc(){
    boolean[] visited = new boolean[graph.size()];
    ArrayList<ArrayList<Integer>> result = new ArrayList<>();
    int count = 0;
    for (int i=0; i<graph.size(); i++){
        if (!visited[i]){
            count++;
            ArrayList<Integer> ans = gcc(i, visited);
            result.add(ans);
            syso(count);
        }
    }
    syso(result);
}

Static ArrayList<Integer> gcc(int sc, boolean[] visited){
    visited[sc] = true;
    ArrayList<Integer> ans = new ArrayList<>();
    ans.add(sc);

    for (edge nbr : graph.get(sc)){
        if(!visited[nbr.v]){
            ArrayList<Integer> res = gcc(nbr.v, visited);
            ans.addAll(res);
        }
    }
```

return res;

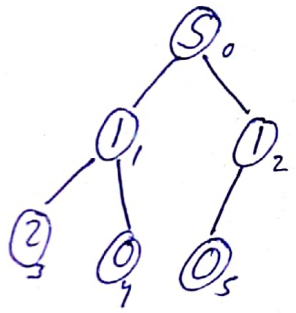
}

Approach :-

We are making call for each not visited call and that call will return the graph node. and we will count the no of calls or the no of arr arraylist returned.

Heap sort :-

[5, 1, 1, 2, 0, 0]



Approach :- Call heapSort

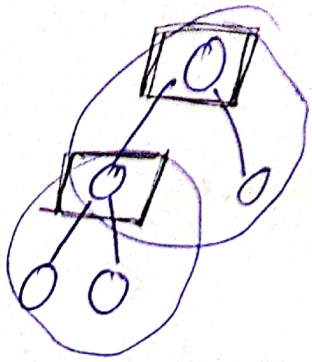
```
heapsort (arr)
    build heap (arr, arr.len)
for( i = arr.len-1; i>0) {
    swap(arr, 0, i);
    heapify(arr, i, 0);
}
```

```
heapify (arr, n, ci){
    large = ci;
    cl = (ci*2)+1;
    cr = (ci*2)+2;

if( cl<n && arr[cl] > arr[large]){
        large = cl;
}
if(cr<n && arr[cr] > arr[large]){
        large = cr;
}

if ( large != ci) {
    swap ( arr, ci, large);
    heapify (arr, n, large)
}
```

Swaps the largest elem to the end and ignore the last element;

① heapSort {
    build heap (arr)
    for (i = size ;)
    swap (arr[0], arr[i])
② build heap (arr, len)
③ heapify (arr, n, ci)

```
buildheap(arr, len){
    int n = arr.length;
    Start = (n/2)-1;
    for( i=start; i>=0; i--){
        heapify(arr, n, i);
    }
}
```



Heap sort Algorithm

1) Build a max heap from the input data.
2) At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally heapify the root of the tree.
3. Repeat the step 2 while the size of the heap is greater than 1.

Time Complexity :- Heapify → O(log n)

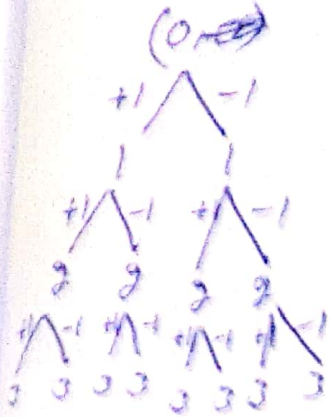
Create & Buildheap → O(n)

Overall time complexity → O(n log n)

Grows logarithmically

Target Sum :-

nums = [1, 1, 1, 1, 1] target = 3



-1+1+1+1+1 ⇒ 3
+1-1+1+1+1 ⇒ 3
+1+1-1+1+1 = 3
+1+1+1-1+1 = 3
+1+1+1+1-1 = 3

```
backtracking ( int nums , target , curr) {
    if (curr == arr.len  && tar == 0) {
        count ++;  return;
    }
    if (curr >= arr.len ) return.
    pos = arr[curr] x 1;
    neg = arr[curr] x (-1);

    backtracking ( arr, target - pos , curr+1);
    backtracking ( arr , target - neg , curr+1);

  }
}
```

What are NP, P, NP-Complete and NP-Hard Problems.

⇒ P is a set of problem that can be solved by Deterministic Turing machine in Polynomial time.

⇒ NP is set of decision problems that can be solved by non-deterministic Turing machine in Polynomial time.

P is subset of NP (any problem that can be solved by deterministic machine in Polynomial time can also be solved by a non-deterministic machine in Polynomial time.

NP Complete → hardest problem set in NP

A decision Problem ~~NP~~ L ~~is~~ is NP-complete if:

1) L is in NP (Any given soln for NP-complete prob can be verified quickely, but there is no efficient know soln.

2) Every problem in NP is reducible to L in Polynomial time

A Problem is NP Hard if it follows Property 2 doesn't need to follow Property 1. Therefor NP-complete is also a subset of NP-Hard set.

Maximum Subarray Sum using Divide & Conquer

```
main ( ){
return   maxSubSum (arr, 0, n-1)
}
maxSubSum ( arr, low, high) {
   low == high   retur arr[low];
   mid = (low+high)/2

   return max( max(maxsum (arr, low, mid), maxSum(mid+1,
                ), maxCrossingSum(arr, low, mid, high));
}

maxCrossingSum( arr, low, mid, high){
   Sum = 0
   left-sm =  Integer.MIN_VALUE;
   for( i= mid ; i ≥ low ; i-- ){
      Sumt = arr[i] ;
```

```
    If ( sumleft > sum) sumlef = sum;
}

sum = 0
sumright = Integer. MIN_VALUE;
for( i= mid+1; i≤ high; i++){
    sum+ = arr[i];
    If (sum > sumrigh) sumrigh= sum;
}
return max( max(leftsum + rightsum), max(leftSum,
                                         rightsum));
```

Find the kth smallest element in the array.

Approach 1 → 1. Sort the array using heap sort $O(n \log n)$

2. return the kth element of the sorted array $O(1)$

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

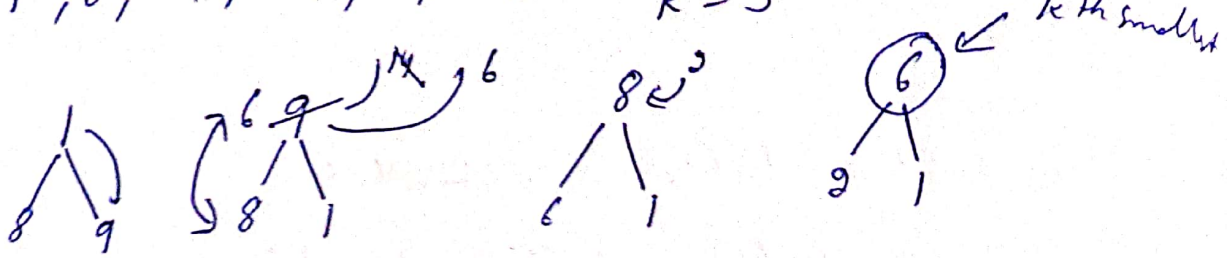
Approach 2. By using the Max heap.

Step 1. Create a heap of size k (max heap)

Step 2. traverse the array and compare the heap peek element with arr[i] elem
If arr[i] > peek then ignore else
Poll from heap & Push.

Step 3. return the peek elemet of the heap.

{ 1, 8, 9, 14, 6, 2 } k = 3



Approach 3. Using the min heap

Step 1 Create a min heap of the array Using the min heap

Step 2. Poll the ~~k~~ k-1 element from the min heap

Step 3. return the minheap peek element.

BFS In Graph :-

```
make (edges[], vertices) {
    graph = new
    ( (    ( (
}
addEdge (u, v, w){

static void bfs (int srv){
    boolean[] visited = new boolean[graph.size()];
    ArrayList<Integer> q = new ArrayList<>();
    q.add(srV);
    int lev = 0;
    while(q.size() != 0){
        int sz = q.size();
        sysо (eev + "==>");
```

```
while(sz--!=0){
    int rm = q.remove();
    if(visited[rm]){
        // sys.o("cycle");
        continue;
    }
    visited[rm] = true;
    Syso.out.print(rm + " ");
    for(edge nbr : graph.get(rm)){
        if(!visited[nbr.v]){
            q.add(nbr.v);
        }
    }
}
syso()
cev++;
}
```

Asymptotic Analysis :-

mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value

orders of growth :-

is a set of functions whose asymptotic growth behavior is considered equivalent. (Big-oh notation) The growth of a fn is determined by the highest order term.

Asymptotic Notations

- Big O: Exact or Upper
- Theta: Exact
- omega: Exact or