discrete
Date 8-6-24
Page 1

* Imp Information for strong resume :

1) CGPA → 9.5 +
2) Unique experience : eg. best rank in hackathon, good company Intership, best rank in competitive programming.
3) consistency : atleast 1 hr code each & every day.
4) Strong project : atleast : 2 project of web dev.
   : 1 project base on ML.

- In these course → 4 month DSA
  4.5 month web. development
  full stack web d → MERN
  M - mongo DB
  E - express
  R - React
  N - Node.js

- ppo - pre placement offer - after 3rd year
  ppI - pre placement Intership. - after 2nd year.

- website - Stack overflow :
  - all answer related to coding errors, exceptions etc present in these website.

- The harder you work the lukier you get

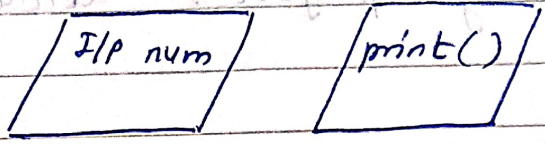
hardwork & lukier

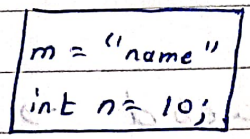
* flowChart & pseudo code :

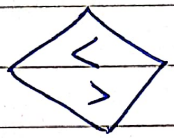
- diagram to represent soln of problems
  1) divide in small part
  2) logically arrange

- Components :

1) Start / end → 

2) I/p ; o/p → (parallelogram) 

3) Process : → (Rectangle) 

4) Decision → 

5) ↓ ⇄ → arrow.

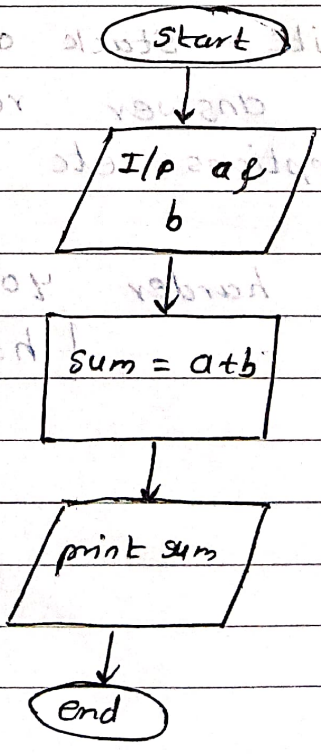
ex: Sum of two number :

I/p : number a & b

o/p : sum of a & b.

- pseudo code :

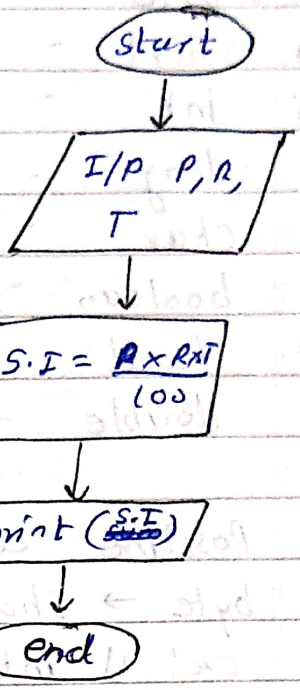
1) Start
2) I/p number a & b
3) Calculate sum = a+b ;
4) print sum
5) end



2) Calculate simple Interest :

I/p : ~~a & b~~ P, R, T || Principle, rate & time

O/p : ~~Sum~~ $\frac{P.R.T}{100}$



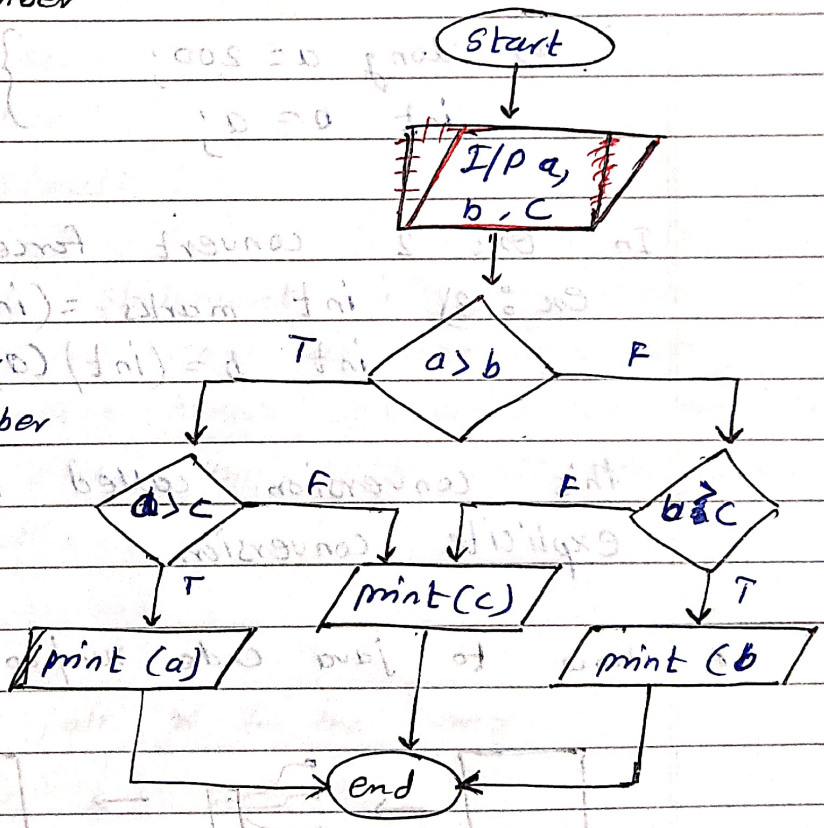
- Pseudo code.

1) Start
2) I/p P, R, T.
3) Calculate simple interest.
4) print the S.I.
5) end

3) find max 3 number

I/p : a, b & c

O/p : max of 3.

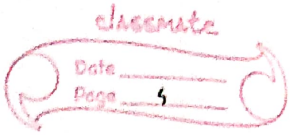


- pseudo code.

1) Start
2) I/p a, b & c number
3) compare three number
4) end

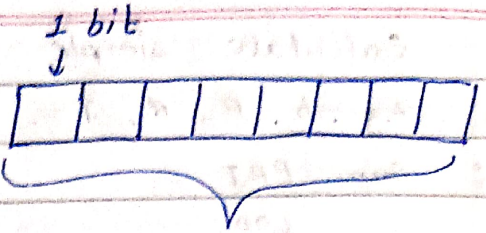
Literals - 1, 2, 3, 4, 5

variable - a, b, ... any (declare to store the data).



* Data type & size :

1) 1 byte - 8 bit
2) short - 2 byte
3) int - 4 byte
4) long - 8 byte
5) char - 2 byte
6) boolean - 1 byte
7) float - 4 byte
8) double - 8 byte



32 bit operating system

[Note: for 16 bit OS size of all data type is half of these. ex- int - 2 byte]

- Possible conversion :

byte → short → int → float → long → double

ex: 1) int a = 100;
long b = a; } possible

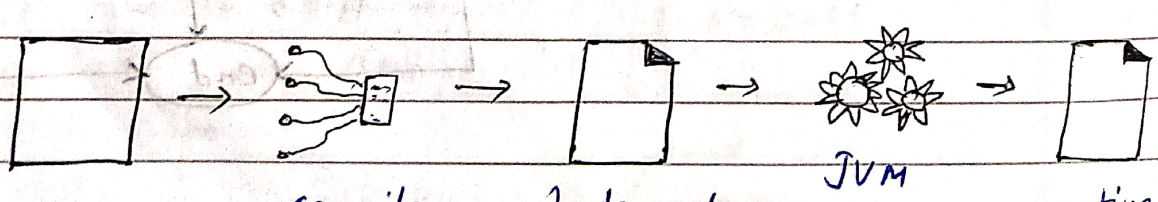
2) long a = 200;
int b = a; } throw error.

In ex: 2 convert forcefully (coercion)

ex: 3) int marks = (int) (99.99f)
int b = (int) (a) — ex: (2)

this conversion called type casting / narrowing / explicite conversion.

* How to java code run/compile :



* Type of operation:

1) Arithmetic → +, -, /, *, % etc.
2) Unary → ++, --
3) Relation operator → <, >, ==, != etc.
4) logical operator → &&, ||, !, etc.
5) bitwise operator → ^, |, &, etc.
6) terary operator → variable = cond" ? state1 : state2;

cond" check

cond" is true then execute state1 otherwise execute statement 2;

ex: ~~boolean~~ int larger = (5>3) ? 5 : 3;
println(larger) → 5

* Switch statement:

Syntax: switch (variable){
case :
case :
default :
}

Note: In switch statement if any case are true then this case is execute but does not write break keyword in ~~this~~ end this case then after this cases all ~~are~~ cases are execute;

* Array:

Q1) print the possible pair ~~of~~ in the array

ex: arr = {1, 2, 3}

o/p: (1, 2); (1, 3)
(1, 3);
total pair = 3

//code//
```
public static void possiblePair(int arr[]){
    int l = arr.length;
    int count = 0;
    for(int i=0; i<l-1; i++){
        for(int j=i+1; j<l; j++){
            System.out.print("("+arr[i]+","+arr[j]+")");
            count++;
        }
        System.out.println();
    }
    System.out.println("total pair :" + count);
}
```

Q2) print all subarray of array

ex: arr = {1, 2, 3};

o/p: 1; 1 2; 1,2 3;
2; 2 3;
3;
total ~~pair~~ subarray = 6;

//code//
```
public static void subArr(int arr[]){
    int l = arr.length;
    int count = 0;
    for(int i=0; i<l; i++){
        for(int j=i; j<l; j++){
            for(int k=i; k<=j; k++){
                System.out.print(arr[k]+" ");
            }
            count++;
            System.out.print("; ");
        }
        System.out.println();
    }
    System.out.println("total subarray :" + count);
}
```

Q3) maximum sum of subArray : (prefix sum):

```
public static int maxSumOfSubArr(int arr[]){
    int l = arr.length;
    int maxSum = Integer.MIN_VALUE;
    int currSum = 0;
    int arr2[] = new int[l];
    arr2[0] = arr[0];
    for(int i=1; i<l; i++){
        arr2[i] = arr2[i-1] + arr[i];
    }
    for(int j=0; j<l; j++){
        for(int k=j; k<l; k++){
            currSum = j==0 ? arr2[k] : arr2[k] - arr2[j-1];
            if(maxSum < currSum){
                maxSum = currSum;
            }
        }
    }
    return maxSum;
}
```

ex: arr = {2, 3, 4, -1, 2, 3};

sumArr = | 2 | 5 | 9 | 8 | 10 | 13 |

subArrays;

i=0: 2 ; 2 3 ; 2 3 4 ; 2 3 4 -1 ; 2 3 4 -1 2 ; 2 3 4 -1 2 3

i=1: 3 ; 3 4 ; 3 4 -1 ; 3 4 -1 2 ; 3 4 -1 2 3 ;

j=0 j=1 j=2 ---- j=5

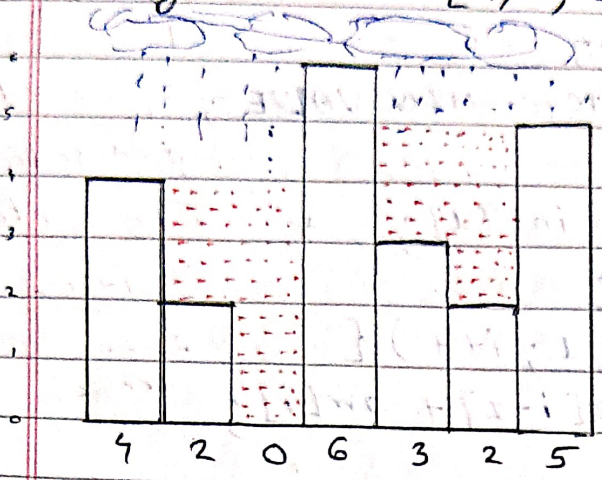
for i=0 sum of subArrays ⇒ sumArr[j]

for i=1 sum of subArrays ⇒ sumArr[j] - sumArr[i-1]

[for second row arr[0] not included then subtract arr[0] - for from arr2[] - for each elements.] then gives sum of subArray for second row.

Q 4) Trapping rain water.

ex: height [] = {4, 2, 0, 6, 3, 2, 5};

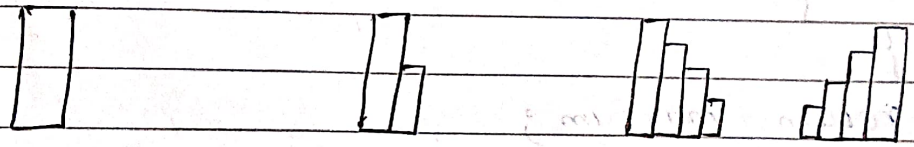


these are building and height is given and weidth is 1 of all building.

process:

1) calculate max height of left & right building by using loop.
2) Calculate water level
   waterlevel = Math.min(leftbound, right bound);
3) trapped water = waterlevel − height[i];

Case 1)



one building | two building | accending or decending order

No water trapped. (~~i==1~~ if i==2)

Step:

1) Create two helper arr & store max height

1) | 4 | 4 | 4 | 6 | 6 | 6 | 6 | → left to right

2) | 6 | 6 | 6 | 6 | 5 | 5 | 5 | → right to left.

2) waterlevel = min value from these arrays.
   ⇒ 4, 4, 4, 6, 5, 5, 5

3) trappedwater = water level − height
   ⇒ 4-4; 4-2; 4-0; 6-6; 5-3; 5-2; 5-5
   ⇒ 0 + 2 + 4 + 0 + 2 + 3 + 0 ⇒ 11

```
//code// public static void rainWaterTrapped (int height[]){
        int l = height.length;
        if( l == 1 || l == 2 ) {
            System.out.println ("No water Trapped");
            return;
        }
        int leftMax = height[0];
        int leftMaxBound = new int [l];
        for( int i=0; i< l ; i++ ) {
            if( leftMax < height[i]) {
                leftMax = height[i];
            }
            leftMaxBound[i] = leftMax;
        }
        int rightMax = height[l-1];
        int rightMaxBound[] = new int [l];
        rightMaxBound[l-1] = height[l-1];
        for ( int j=l-2 ; j>=0 ; j-- ) {
            if ( rightMax < height[j]){
                rightMax = height [j];
            }
            rightMaxBound [j] = rightMax;
        }
        int waterlevel = 0;
        int waterTrapped = 0;
        for( int i=0 ; i<l ; i++ ) {
            waterlevel = Math.min(leftMaxBound[i], rightMaxBound[i])
                         - height[i];
            waterTrapped += waterlevel;
        }
        System.out.println("water Trapped :" + waterTrapped );
    }
```

leftMax boundry calculate

rightMax boundry calculate

calculate water level

leftMaxBound[0] = height[0];

remaining line

Q. 5) Buy & sell stock :

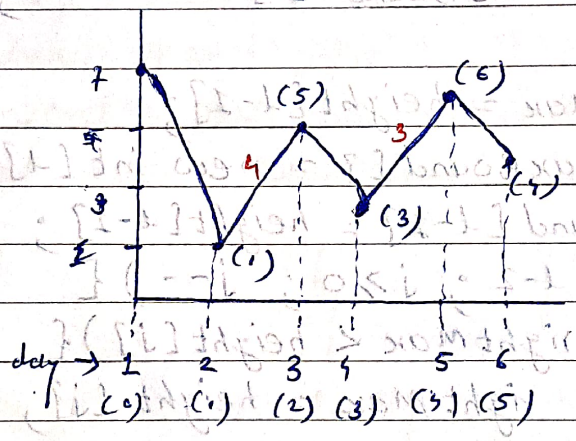
- you are given an array prices where prices [i] is the price of given stock on the ith day. You want to maximum your profit by choosing a single day to buy one stock & choosing a different day in the future to sell that stock. Return the maximum profit you can achive from this transaction. If you cannot achieve any profit return 0.

maximum value ↑ minimum value ↑

→ Profit = selling price − buy price
(max)

ex: prices [] = {7, 1, 5, 3, 6, 4};

graph:



for day 1 → only buying of stocle
buyStocle = 7. ; Profit = 0;

for day 2 → selling price = 1 (loss)
buy stocle = selling prices. ; Profit = 0

for day 3 → selling price = 5
buy Stocle = 1 ; Profit = 4.

for day 4 → selling price = 3
buy stocle = 5 ; Profit = 2 < 4 = 4

for day 5 → selling price = 6 ;
buy Stocle = 3 ; Profit = 3 < 4 = 4

Creating 2 variable to store the minStock price and maximum stocle price in maxProfit.

//code//

```
public static void maxProfitOfStock(int prices[]) {
    int l = prices.length;
    int buyPrice = prices[0];
    int minPrice = 0; maxPrice = 0;
    int maxProfit = Integer.MIN_VALUE;
    for(int i=0; i<l; i++){
        if(buyPrice < prices[i]){
            int profit = prices[i] - buyPrice;
            if(profit > maxProfit){
                maxPrice = prices[i];
                maxProfit = profit;
            }
        }else{
            buyPrices = prices[i];
            minPrice = prices[i];
        }
    }
    System.out.println("Min Price of stock : " + minPrice);
    System.out.println("Max Price of stock : " + maxPrice);
    System.out.println("Max Profit of stock : " + maxProfit);
}
```

Q6) Sorting :
1) bubble sort
2) selection sort
3) Insersion sort

} present book-I

4)

~~Q7)~~ // book I - present question related to 1-D array //

* 2-D Array *

1) // basic, I/p - o/p of 2-D array //
2) // addition of two matrices //
Imp 3) // Spiral matrix //
4) // multiplication of two matrix //
5) // rotate matrix // ; etc present in book-I

Q. Sum of diagonals.

ex:

1	2	3
4	5	6
7	8	9

Sum = 15 + 10 = 25

1	2	3	4
5	6	7	8
9	10	11	12
1	2	3	4

Sum = 22 + 22 = 44

// brute force logic → primary diagonal ⇒ i==j
secondary diagonal ⇒ i+j == l-1
// create two nested loop & writedown these cond" //
time complexity = $O(n^2)$:
∴ optimise code

// code //

```
public static void sumOfDiagonal(int arr[][]){
    int l = arr.length;
    int sum = 0;
    for(int i=0; i<l; i++){
        sum += arr[i][i];
        sum += arr[i][l-1-i];
    }
    int num = 0;
    if(l%2 != 0){
        num = l/2;
        sum -= arr[num][num];
        System.out.println(sum);
    }else{
        System.out.println(sum);
    }
}
```

Q7) find key in sorted array (2D).

ex: arr[][] ⇒

10	20	30	40
15	25	35	45
27	28	38	48
29	33	49	50

} both row & column are sorted.

* Step:

~~cond consider~~ ~~first~~ key is start to find from position (0,3) and (3,0). choose any one position check key is < or > , then move position.

//code: public static void findKeyInSortedArr(
int arr[][], int r, int c, int key) {

```
int top=0; int bott=r-1; int left=0; int right=c-1;
while (top<=bott || right<=left){
    if (key == arr[top][right]){
        printf("key is present (%d, %d)", top, right);
        return;
    }
    else if (key < arr[top][right]){
        right--;
    }
    else {
        top++;
    }
}
System.out.println("key does not present in array");
}
```

Q8) transpose of matrice

Q9) pascal triangle using 2D array.

(*) Strings *

- String is Immutable ; they are does not change
- if change the string then create another variable of string & store changed string.

Q 1) Check string is palidrome.

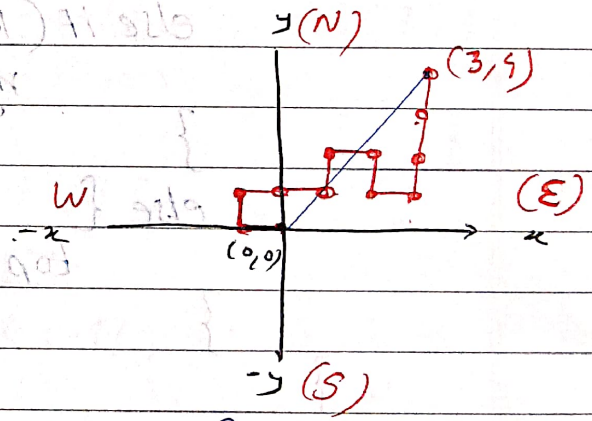
→ ex: "madam", "noon", "racecar" ;

```
// code//  public static boolean checkPalidrone(String st){
            int l = st.length();
            for(int i=0; i<l/2; i++){
                if(st.charAt(i) != st.charAt(l-1-i){
                    return false;
                }
            }
            return true;
        }
```

Q 2) Given a route containing 4 directions (E, W, N, S) find the shortest distance/path to reach destination;

→ Ex: "WNEENESENNN"

∴ sort dist. = $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$




```
// code// public static void shortDistance(String st){
        int x=0; int y=0;
        for(int i=0; i<st.length(); i++){
            if(st.charAt(i)=='W'){
                x--;
            }if(st.charAt(i)=='E'){
                x++;
            }else if(st.charAt(i)=='N'){
```

```
            y++;
        }else {
            y--;
        }
    }
    int dist = (x*x) + (y*y);
    float shortDist = Math.sqrt(dist); (float) Math.sqrt(dist);
    System.out.println(dist)
}
```

Q3) Print largest string lexicographically.

→ ex: "apple", "mango", "banana";

∴ a < b < m ; hence largest string = "mango";

use the method st.compareTo(string);

//code//
```
public static void main(String args[]){
    String fruits[] = {"apple", "mango", "banana"};
    String large = fruits[0];
    for(int i=1; i<fruits.length; i++){
        if(large.compareTo(fruits[i]) < 0){
            large = fruits[i];
        }
    }
    System.out.println(large);
}
```

In internally st1.compareTo(st2) > 0 → st1 > st2

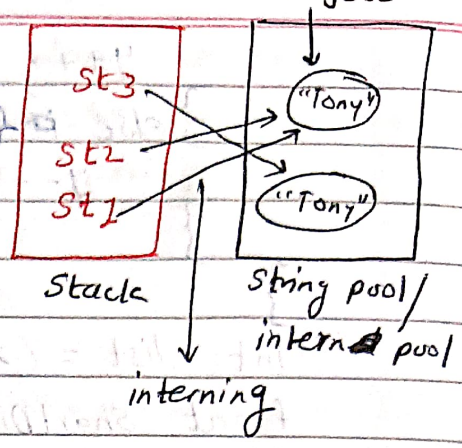
st1.compareTo(st2) < 0 → st1 < st2

st1.compareTo(st2) = 0 → equal.

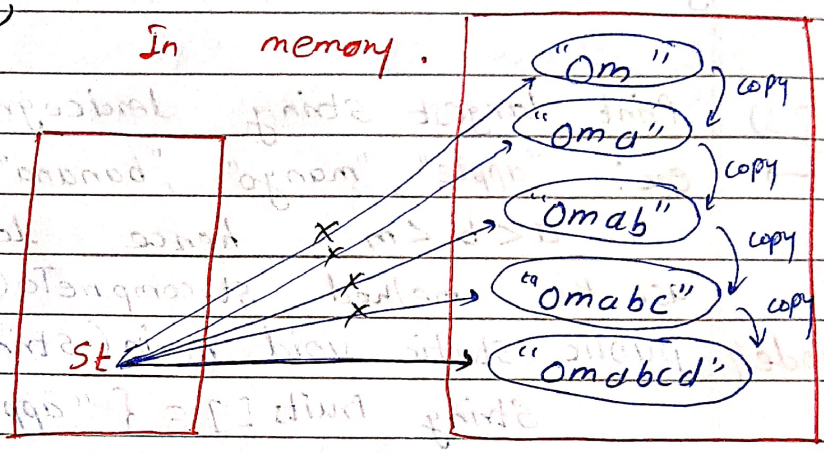
time complexity = O(n*l);

* String In memory

1) ex: St1 = "Tony";
St2 = "Tony";
St3 = new String("Tony");



2) String st = "Om"; char = 'a';
for(int i=0; i<n; i++){
st += char;
char++;
}



Note: when adding character in string then pointing the string in memory to second string & remove pointing arrow from 1st string.

- In case n is 1000 - 100000... then string is copying n times & loop is n times.
time complexity ≅ $O(n^2)$.

- In these case, string is many time change then ~~you~~ use new data structure StringBuilder.

* StringBuilder:
Syntax: StringBuilder sb = new StringBuilder("");

ex: for(Char ch='a'; ch≤'z'; ch++){
~~sb += ch;~~ sb.append(ch);
}

time complexity = O(n);

calculate length → sb.length();

Q4) for given string convert each first letter of each word to upperCase;

→ ex: String st = "onkar appasaheb shinde";
String ans = "Onkar Appasaheb Shinde

```
//code//
public static String toUpperCase(String st);
    StringBuilder sb = new StringBuilder("");
    Char ch = st.Character.toUpperCase(st.charAt(0));
    sb.append(ch);
    for(int i=0; i<st.length()-1; i++){
        Char curChar = st.charAt(i+1);
        if (st.charAt(i) == ' '){
            curChar = Character.toUpperCase(st.curChar);
        }
        sb.append(curChar);
    }
    return sb.toString();
}
```

Q5) Compress the String :

→ ex: "aabbbccddd" —compress→ "a2b3c2d3"

```
//code//
public Static String compressString(String st){
    StringBuilder sb = new StringBuilder("");
    Integer count = new Integer(1);
    for(int i=0; i<st.length(); i++){
        Char curChar = st.charAt(i);
        sb.append(curChar);
        while(i<st.length()-1 && st.charAt(i)==st.charAt(i+1)){
            count++;
            i++;
        }
```

```
        if (count > 1) {
            String newSt = count.toString();
            sb.append(newSt);
        }
        count = 1;
    }
    return sb.toString();
}
```

// you can also solve by string concatenation //

time complexity = O(n)

* Bit Manipulation:

Note: Operator & bitwise operators & binary number system in book-I → page 42 to 45.

* one's complement (~) : actual working:

ex: ~5.

∴ 5 = 00000101 ← MSB (most significant bit) ← (LSB) List significant bit

∴ ~5 = 11111010

∴ ~(~5+1) = 00000101 + 1 1 (carry)

00000110

addition of binary numbers.

0 to 0 ; 1+1 = 0 → carry 1

1 to 0 = 1.

MSB = 0 → +ve value (6)$_{10}$

MSB = 1 → -ve value

Step of 2's complement

ex: 1) 1's complement ~(~n)

2) and 1 binary

3) check step 1 MSB & put sign in step 2 answer.

$\therefore$ $\sim 5 = \sim(00000101) = 11111010$ — MSB = 1 (-ve)

step 1: $\sim(\sim 5) = 00000101$

step 2: add 1 $\Rightarrow$ $00000101 + 1 = 00000110 \Rightarrow (6)_{10}$

$\therefore$ $\sim 5 = (-6)_{10}$

ex 2) $(\sim 0) = (-1)_{10}$;

ex 3) $(\sim 1) = (-2)_{10}$.

Q. check the given number is even or odd by using bitwise operator.

→ ex: num = 4

if((num & 0) == 0) → even

if((num & 1) == 1) → odd

1	001
2	010
3	011
4	100

```
//code//
public static void evenOddNum(int num){
    int bitmask = num & 1;   ---- // num & 000001 //
    if(bitmask == 0){
        System.out.println("Given number is even");
    }else{
        System.out.println("Given number is odd");
    }
}
```

Q. get bit of ith position.

→ ex: num = 00101 , posi = 2 (3rd bit).

step: consider bit = 1 (0001).

move 1 in left shift by using (<<) by posi times.

$\therefore$ ~~num~~ bitmask = num & (bit << pos);

= 00101 & (00100).

= 00100.

if(bitmask == 0) $\Rightarrow$ 0 present

otherwise $\Rightarrow$ 1 present

// code //

```
public static void get_ith_Term(int num, int pos){
    int bitmask = num & (1<<pos){
    if( bitmask == 0){
        System.out.println("0");
    }else{
        System.out.println("1");
    }
}
```

Q. Set bit in ith term.

→ ex: 10101 set 1 position 1.

Step 1) bit = 1 << pos.

2) num | bit

// code //

```
public static void set_ith_Term(int num, int pos){
    int bitmask = num | (1<<pos);
    System.out.println(bitmask);
}
```

Q clear bit in ith position.

→ ex: num = 10101 pos = 2nd.

step: 1) ope = ~(1<<pos)

2) bitmask = num & ope;

// code //

```
public static void clear_ith_Term(int num, int pos){
    int ope = ~(1<<pos);
    int bitmask = num & ope;
    System.out.println(bitmask);
}
```

Q. clear last n bits.

→ ex: num = 1111 (15) n = 2

o/p: 1100

Imp: Note: n(0) = 11111111

(-1) = 11111111

for (-1)<<2 = 11111100 then and with num.

∴ num = 1111 n=2

∴ bitmask = num & (-1<<2); ⇒ 1100

```
//code
public static void clearLast_N_Bits(int num, int n){
    int bitmask = num & (-1<<n);
    System.out.println(bitmask);
}
```

Q. clear bits from i to j terms.

Ex: num = 10110101 → clear bits i=2 to j=7.

position → 8 7 6 5 4 3 2 1 0

O/P: 100000011

logic:

00000 1 = <<1 = 2^1-1

000011 = $(3)_{10}$ = 2^2-1

000111 = $(7)_{10}$ = 2^3-1

001111 = $(15)_{10}$ = 2^4-1

011111 = $(31)_{10}$ = 2^5-1

∴ 1<< b = 2^b

∴ 1<< i = 2^i

Step 1: a = (-1) << (j+1);

b = (1<<i) - 1;

bitmask = num & (a | b);

```
//code
public static void clear_i_to_j_bits(int num, int i, int j){
    int a = (~0) << (j+1);
    int b = (1<<i) - 1;
    int bitmask = num & (a | b);
    System.out.println(bitmask);
}
```

Imp:

2^1 = 10 — 1 = 1

2^2 = 100 — 3 = 011

2^3 = 1000 — 7 = 0111

2^4 = 10000 — 15 = 01111

n & (n-5) = 0 ↑ n is number of power 2

Imp amazon, google

Q. count set bits in number (count 1 in number).

→ ex: num = 1010

step: In while loop → check num & 1 == 1 (0001)
then num = num >> 1;

num = num >> 1 = 0101
num = num >> 1 = 0010
num = num >> 1 = 0001
num = num >> 1 = 0000 stop if num == 0;

//code//:
```
public static void countSetBits(int num){
    int count = 0;
    while(num != 0){
        if((num & 1) == 1){
            count++;
        }
        num = num >> 1;
    }
    System.out.println(count);
}
```

∴ time complexity = O(log(n))

n = log n + 1
16 = 10000 (5 digit)
5 = $\log_2(16) + 1$

Q. Fast exponential Go function. (a^n).

→ ex: ans = 5^3 → 125

step: assume convert power in binary

∴ ans = 5^{011}

∴ if(n != 0) → ans = ans * a
a = a * a;

ex 2: a = 3, n = 5

∴ ans = 3^5 ⇒ $(3)^{(0101)}$ → 0 1 0 1
a^0 a^1 a^2 a^4

//code//

```
public static void powerOfNumber(int a, int n){
    int ans = 1;
    while(n!=0){
        if((n&1)==1){
            ans = ans * a;
        }
        a = a*a;
        n = n>>1;
    }
    System.out.println(ans);
}
```

* Oops : object oriented programming

group of entities → ← entities in real word.

* Class & object :
- object store in Heap type memory.
- alway write first letter of class is capti capital and first of method/function is small.
- object is use to define real word object in code.
- and the set of object store the class

ex: all property of pen is a object & pen is class.

Q. create class of pen & set ~~etc~~ color & tip.

//code//
```
class pen {
    String color;
    int tip;
    // function //
    void setColor(String newColor){
        color = newColor;
    }
    void setTip (int newTip){
        tip = newTip;
    }
}
```

- In ~~main~~ public class & main method create object of these class & access.
- In java runtime first find public class & main method.

In main method ⇒

```
Pen p1 = new Pen();
p1.setColor ("red");
p1.setTip (5);
System.out.println (p1.color);
```

Imp *

access Modifier :

- It is used to which type of data can be access or not access by user in outside of the class, method or package.
- there are 4 types :

1) Private :

- private variable/object does not access within package, other class & outside packages.
- it is access only within the class.

Syntax : ex: private password

private password

- Syntax : private type obj = u ;

ex : private String password = "onkar77*77";

2) Default :

- It is access only within class & within package, does not access by out package of class.

Syntax : type obj = u ;

ex : String userName = "ShindeOnkar";

3) Protected :

- It is access within class, within package & outside package of subclass,
- does not access otside package.

Syntax : protected type obj = u ;

4) Public :

- It can be access anywere.

Syntax : public type obj = u ;

* four pillars of oops:

1) Encapsulation
2) Inheritance
3) Polymorphism
4) Data abstraction

(*) 1) Encapsulation:

- Encapsulation is the wrapping up of data & methods under a single unit. It is also implement data hiding.

- It is a single entity to wrapp data & functions in class
- by using access modifier implementing data hiding.

• Constructor:

- constructor is special method which is called automatically at the time to create object of class
- Name of the constructor is same as class
- it does not any return type
- memory allocation happens when constructor is called.

Ex: class onkar {

```
    onkar() {        ---- constructor.
    }
}
```

- There are three type of constructor

1) No argument:
2) With argument:
3) Copy constructor:

3) Copy Constructor:

- In java copy constructor create by users, but in C++ copy constructor is present by the fault.
- copy constructor is the copy properties of one object ot another object.

ex: obj1 & obj2 ⇒ obj2 = copy (obj1).

```
//code//
class Student {
    String name;
    int rollNo;
    int marks[];
    Student() {
        this.marks = new int[3];
    }
    Student(Student obj1){
        this.name = obj1.name;
        this.rollNo = obj1.rollNo;
        this.marks = obj1.marks;
    }
    void getInfo() {
        System.out.println(name);
        System.out.println(rollNo);
        for(int i=0; i<marks.length; i++){
            System.out.print(marks[i] + " ");
        }
        System.out.println();
    }
}
```

main method ⇒ Student obj1 = new Student();
obj1.name = "onkar shinde";
obj1.rollNo = 77;
~~obj1.for~~

```
obj1.marks[0] = 90;
obj1.marks[1] = 85;
obj1.marks[2] = 80;
obj1.getInfo();
// copy properties of obj1 to another object //
Student obj2 = new Student(obj1);
obj2.getInfo();
// change the property of obj1 //
obj1.name = "ramu";
obj1.marks[0] = 100;
obj2.getInfo();
}
```

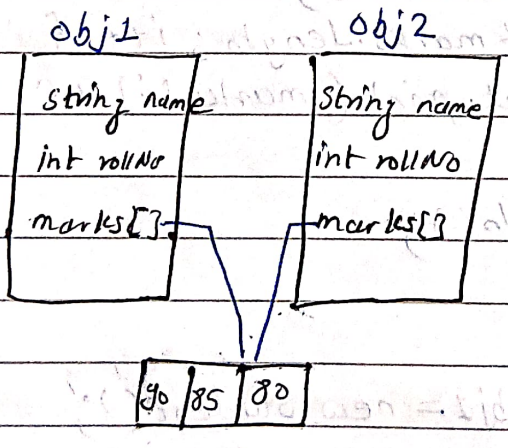
o/p:

```
onkar shinde
77
90 85 80
onkar shinde
77
90 85 80
onkar shinde
77
100 85 80
```

obj1.

obj2.

changing property of obj1 but print obj2 property but array can change because array point on obj1 array.



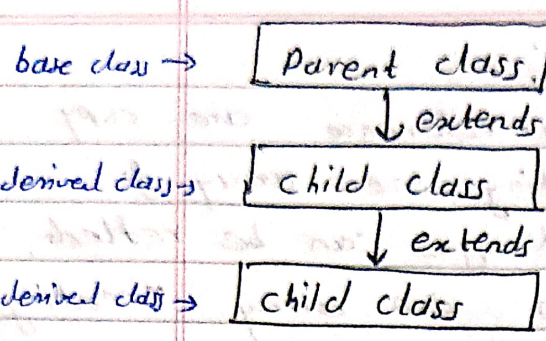
* Shallow & deep copy of array :
- Shallow copy - In these ~~only~~ reference are copy does not creating new array & new address. Changes can be reflect.
- deep copy - In these new are array creating and copy the elements. changes cannot be reflect.
- lazy copy :

* Destructor :
- It is the delete the object memory if it is does not use in programme
- In C++ function present but in java does not create destructor, because in java present garbays collector they are automatic delet memory if does not use.

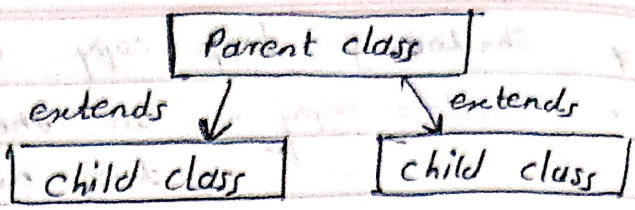
② 2) Inheritance :
- allow to access properties of ~~parte~~ parent class to child class.

ex: class Parent {
}
class Child extends Parent {
}

- single level & multilevel inheritance.



multilevel Inheritance



hierachial Inheritance

Note: multiple inheritance does not present in java. but same as present interfaces, in C++ multiple inheritance present.

3) Polymorphism:

1) Compile time polymorphism

• method overloading:

- same method name but different type of ~~argument~~ parameters. (same class).

ex: ~~sum~~ void sum (int a, int b)
void sum (float a, float b)
void sum (int a, int b, int c)

2) runtime polymorphism

• method overriding

- same method & same argument but diff. class

* Packages in Java:

- package is the ~~group~~ group of similar type of classes, interfaces & sub packages.

4) Abstraction : (Idea)

- Hiding all the unneccessary details & showing only the important parts to the users
- user does not creating object of abstract class.
- you are also write abstract or non-abstract method in abstract class.
- It can have constructor.

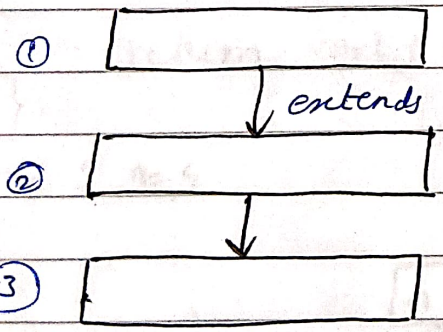
Syntax:

```
abstract class Animal {
        void eat() {
            System.out.println("animals eats");
        }
        abstract void walks();
}
```

does not implementation → abstract void walks();

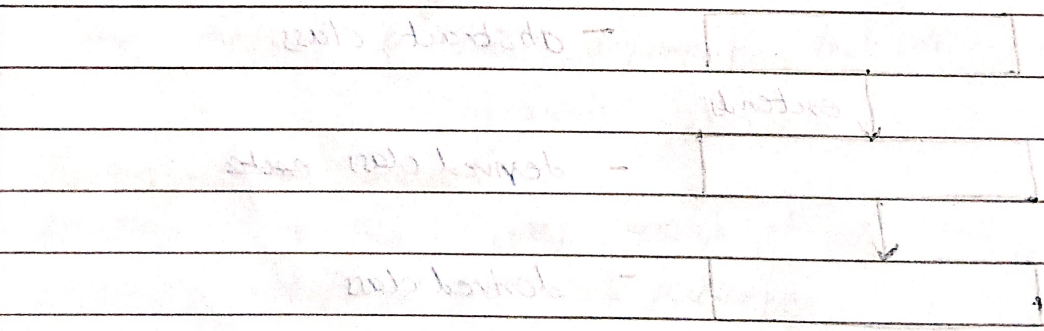
(not write insite the class)

- If Animal class extends in another class then abstract methods can be implemets & it is compulsory
- In abstract class constructor is use to by ~~d~~ ~~the~~ default value ~~but~~ ~~of~~ initialise in these class, it can change in another class.



- abstract class
- derived class ~~exte~~
- derived class

- If creating object 3rd derived class but ~~constructer~~ 1st constructor called of abstract class



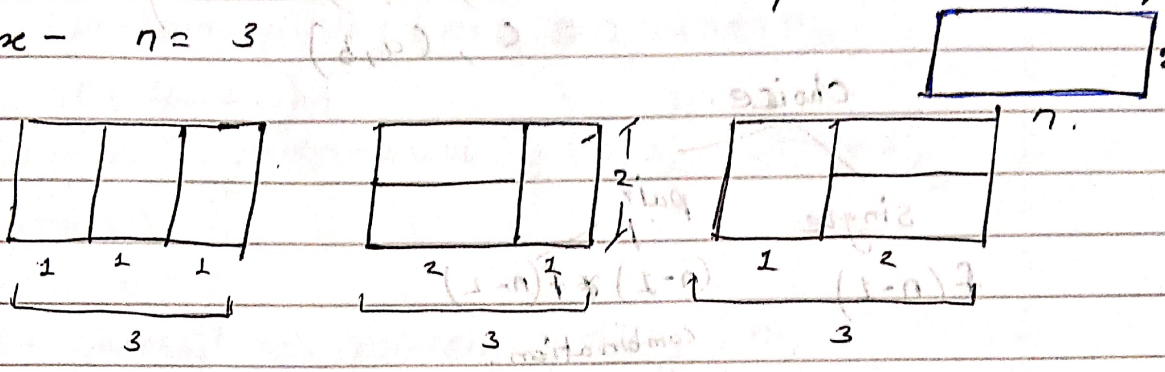
* Recursion:

- theory basic → book I — Page 26 - 41
  + questions. Page : 52 - 55.
  Page : 109 - 114

amazon

Q. Tiling problem.
Given "2 × n" board and tiles of size "2 × 1", count the number of ways to tile the given board using the 2 × 1 tiles.
(A tiles can be place: horizontally or vertically).

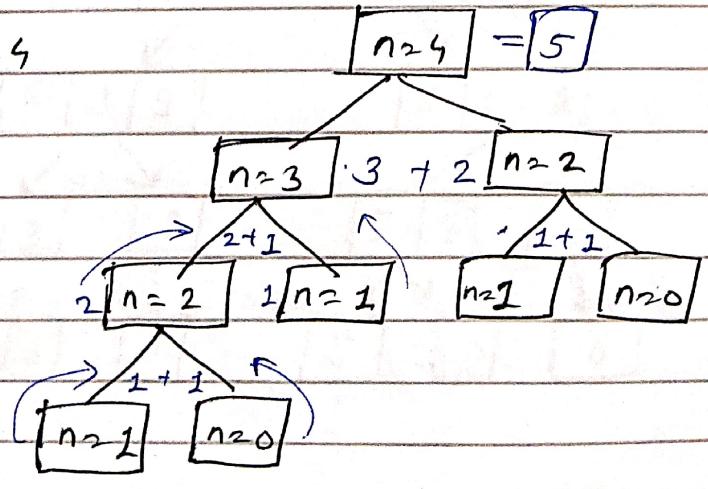
→ ex - n = 3



3 way to place tiles in 2×3 a size.
Size of tile is 2 × 1

//code// public static ~~void~~ int tilingCount (int n) {
if (n==0 || n==1) return 1;
int vert = tilingCount (n-1);
int horiz = tilingCount (n-2);
return vert + horiz;
}

ex: n=4

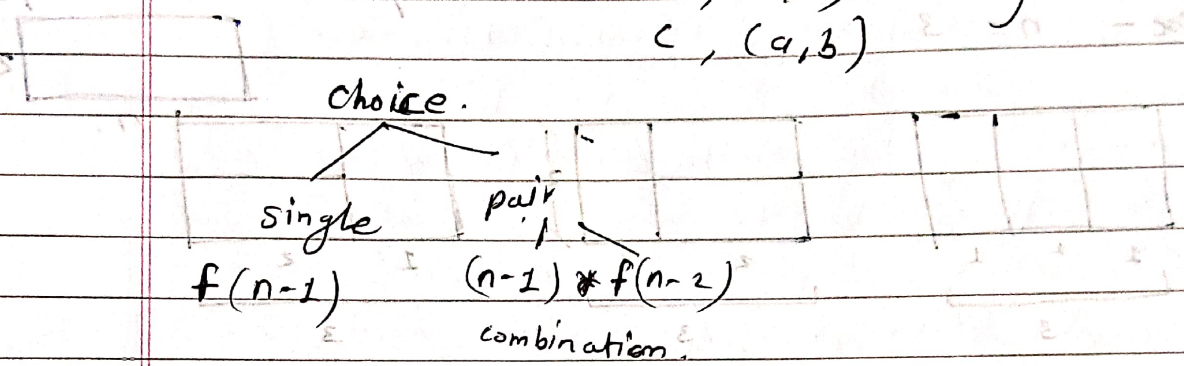


9. Friends Pairing :

- Given n friends, each one can remain single or can be paired up with some other friends. each friends can be paired only one once. Find out the total number of ways in which friends can remain single or can be paired up.

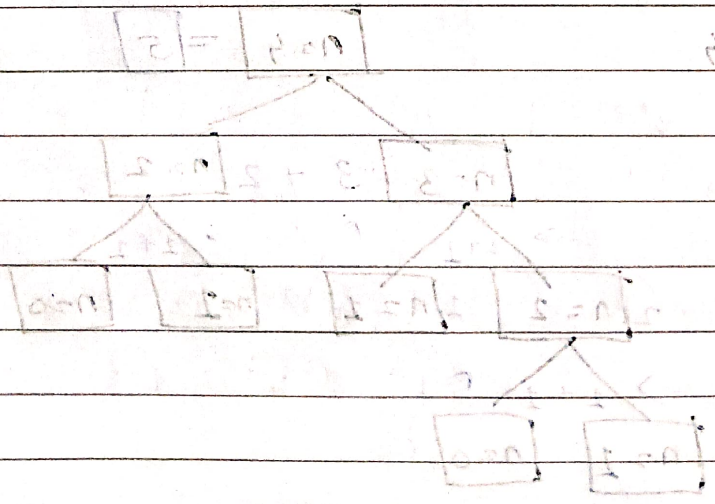
→ exe: n=2 pair → a, b; (a,b) 2 way.

n=3 pair → a, b, c
a, (b,c)
b, (a,c)
c, (a,b)
} 4 ways.



//code//

```
public static int friendPairing(int n) {
    if(n==1 || n==2) return n;
    int f1 = friendPairing(n-1);
    int f2 = friendPairing(n-2) * (n-1);
    return f1+f2;
}
```



Paytm

Q. Binary string problem

- Print all binary strings of size N without consecutive ones.

→ ex: 0000, 0101, 10101, 1001 .. etc.

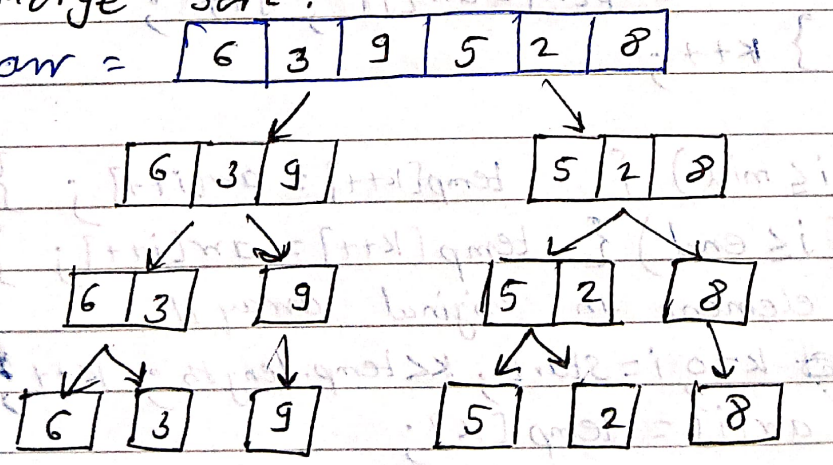
```
//code
public static void printBinaryString(int n, int lp, String str){
    if(n==0){
        System.out.println(str);
        return;
    }
    printBinaryString(n-1, 0, str+0);
    if(lp==0){
        printBinaryString(n-1, 1, str+1);
    }
}
```

main method ⇒ printBinaryString(n, 0, "");

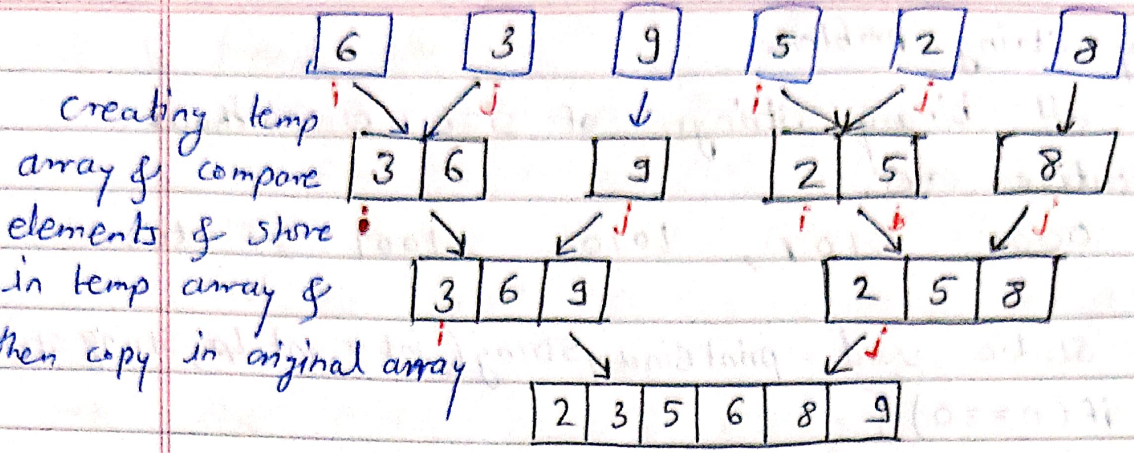
* Divide & Conquer :-

- that means divide the big problems in shortest problem and find the soln then combine the soln.

1) Merge Sort :

ex: arr = 

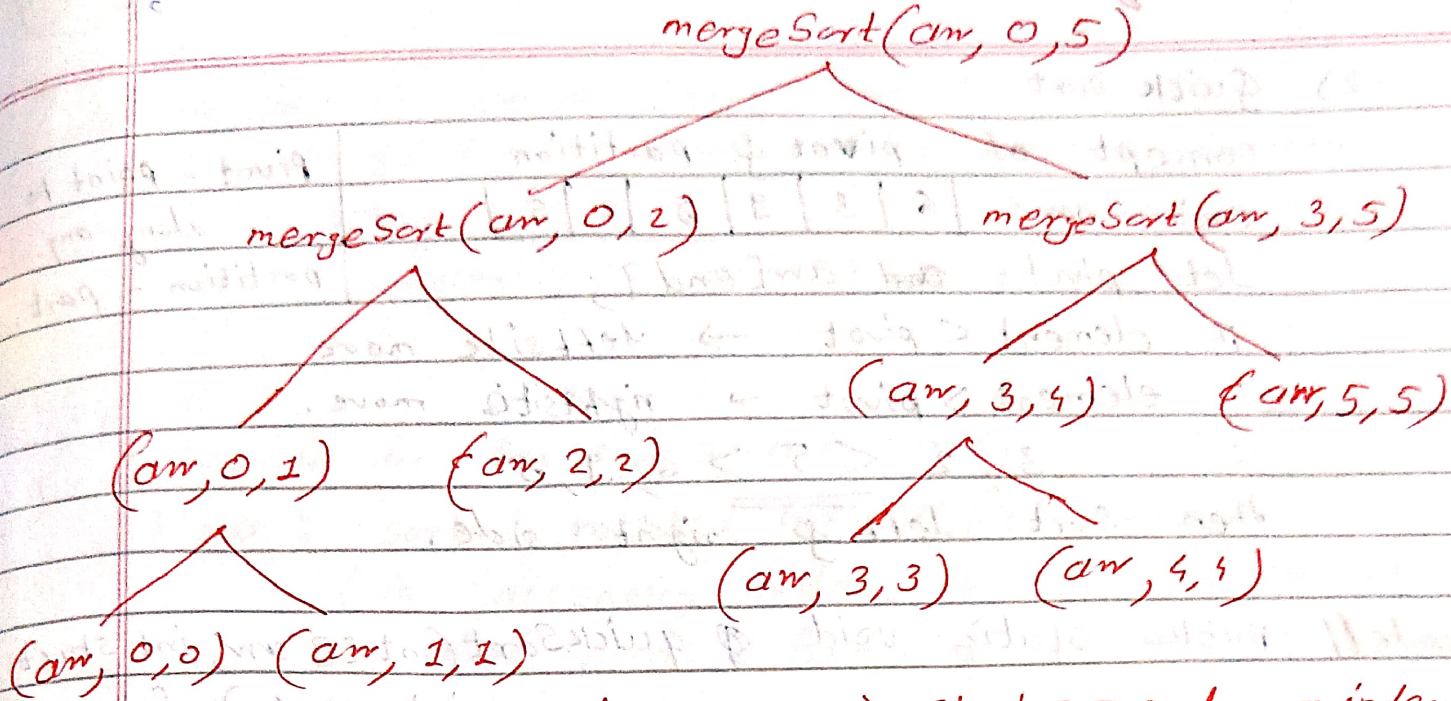
→ they are already sorted.




```
// code //  public static void mergeSort(int arr[], int start, int end){
                if (start >= end) return;
                int mid = start + (end-start)/2 ;
// time        {   mergeSort(arr, start, mid);          ---- (4)
O(log n) //     {   mergeSort(arr, mid+1, end);          ---- (5)
                    merge(arr, start, end, mid);   // time O(n) //
            }
            public static void merge(arr, start, end, mid){
                int [] temp = new int[end-start+1];
                int i = start, j = mid+1;
                int k = 0;
                while(i <= mid && j <= end){
                    if(arr[i] < arr[j]){
                        temp[k] = arr[i]; i++;
                    }else{
                        temp[k] = arr[j]; j++;
                    } k++;
                }
                while(i <= mid) { temp[k++] = arr[i++]; }
                while(j <= end) { temp[k++] = arr[j++]; }
                // copy element in original array //
                for( k=0, i=start; k<temp.length; k++, i++){
                    arr[i] = temp[k];
                }
            }
```

time complexity = O(n log n)

line 4 & 5 function calling tree.

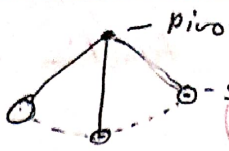


base case ⇒ start == end - index
then return

Time complexity of recursion :

Time = (number of calls) ^height of tree * other work.

= ~~(2)^logn~~



2) Quick sort

concept of pivot & partition

ex: arr = | 6 | 3 | 9 | 8 | 2 | 5 |

let pivot = arr[end];

if element < pivot → leftside move

element > pivot → rightside move.

∴ 3, 2 < 5 > 6, 9, 8

then sort left & right side.

pivot - point to along any.....

partition - part.

//code//

```
public static void quickSort(int[] arr, int start
                                    int end ) {
    if(start >= end ) return;
    int pIdx = partition(arr, start, end);
    quickSort(arr, start, pIdx-1);
    quickSort(arr, pIdx+1, end);
}
public static int partition(arr, start , end ){
    int pivot = arr[end];
    int i = start-1;
    for( int j=0; j< end ; j++){
        if( arr[j] <= pivot){
            i++;
            int temp = arr[j];
            arr[j] = arr[i];
            arr[i] = temp;
        }
    }
    i++;
    int temp= pivot;
    arr[end] = arr[i];
    arr[i] = temp;
    return i;
}
```

Note: The worst case of quick sort is when pivot is always ther smallest or the largeast element idx of array.
time complexity = $O(n^2)$

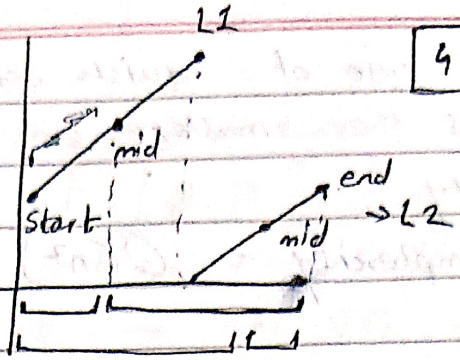
Q: Search in Rotated sorted array
Input: sorted, rotated with distict numbers (in accending order) it is rotated at a pivot point, find the index of given element.

eg: arr = | 4 | 5 | 6 | 7 | 0 | 1 | 2 |
target = 0
ans = 4

```
//code// public static int searchInRotatedSortedArr(int arr[],
                                    int start, int end, int target){
        int mid = start + (end-start)/2;
        if( target == arr[mid] ) return mid;
        // case //
        if(arr[start] <= arr[mid]){
            if( arr[start] <= target && target <= arr[mid]){
                return searchIn(arr, start, mid-1, target);
            }else{
                return searchIn(arr, mid+1, end, target);
            }
        }else{
            if(arr[end] >= target && target >= arr[mid]){
                return searchIn(arr, mid+1, end, target);
            }else{
                return searchIn(arr, start, mid-1, target);
            }
        }
}
```

explaination :

1) start ≤ target ≤ mid (L1)
2) mid ≤ target ≤ end (L2)
otherwise search in remaining part.



4	5	6	7	0	1	2

Q. Practice Set question → Inversion count.

Ex: 1) arr = { 2, 4, 1, 3, 5 }; cond" arr[i] > arr[j]

ans = 3 ⇒ (2,1), (4,1), (4,3).

2) arr = { 5, 4, 3, 2, 1 }

ans = 10 ⇒ (5,4), (5,3), (5,2), (5,1),
(4,3), (4,2), (4,1),
(3,2), (3,1), (2,1).

* BackTracking :

Type of BackTracking

1) Decision Decision :
2) Optimization :
3) Enumeration : (possible soln list)

- backtracking is also recusion, but do work by recursion when it will be return for ex. given.

Q. add element in array & subtract by 2 each value when it will be return.

→ arr = new int [5]

after adding ele → | 1 | 2 | 3 | 4 | 5 |

when return the (-2) each ele → | -1 | 0 | 1 | 2 | 3 |

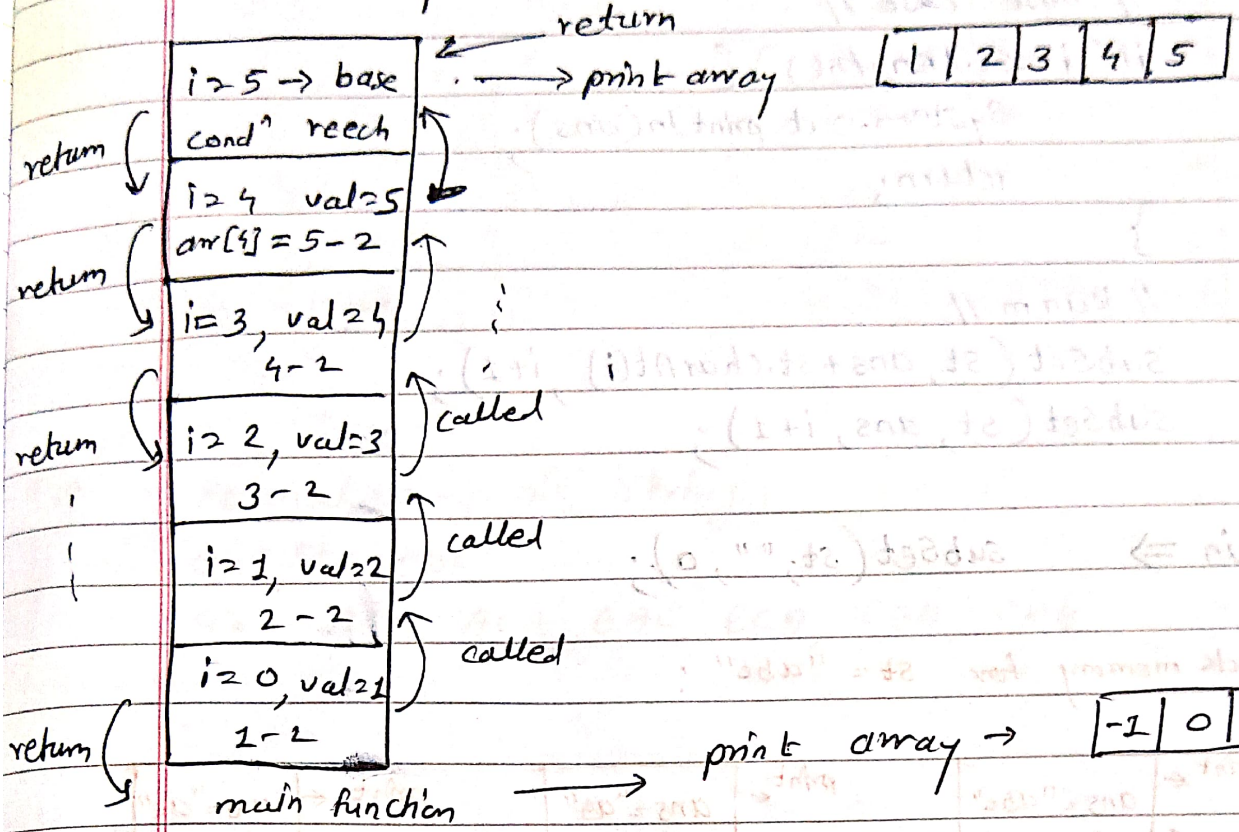
// code //
```
public static void addEleInArr(int[] arr, int i, int val){
    // base case //
    if( i == arr.length)
        printArr(arr);                    - (4)
        return;
    }
    arr[i] = val;
    addEleInArr(arr, i+1, val+1);         - (8)
    arr[i] = arr[i]-2;                    - (9)
}
```

- line (4) is the print arr ele when adding ele (function)
- line (8) is called function
- line (9) is subtract each value by 2 when it will be return

main ⇒ addEleInArr(arr, 0, 1);

O/P : 1, 2, 3, 4, 5
-1, 0, 1, 2, 3

Stack memory

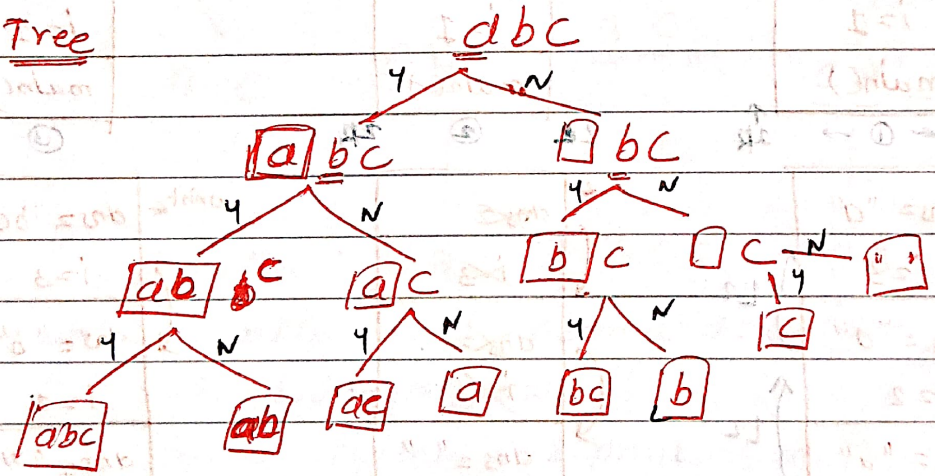


Q. subset of string.

→ ex: st = "abc" → abc, ab, ac, a, bc, b, c, ... ∅

total = 8 subset (2^n).

Tree



creating ~~two~~ function & called when add substring

& called when No add

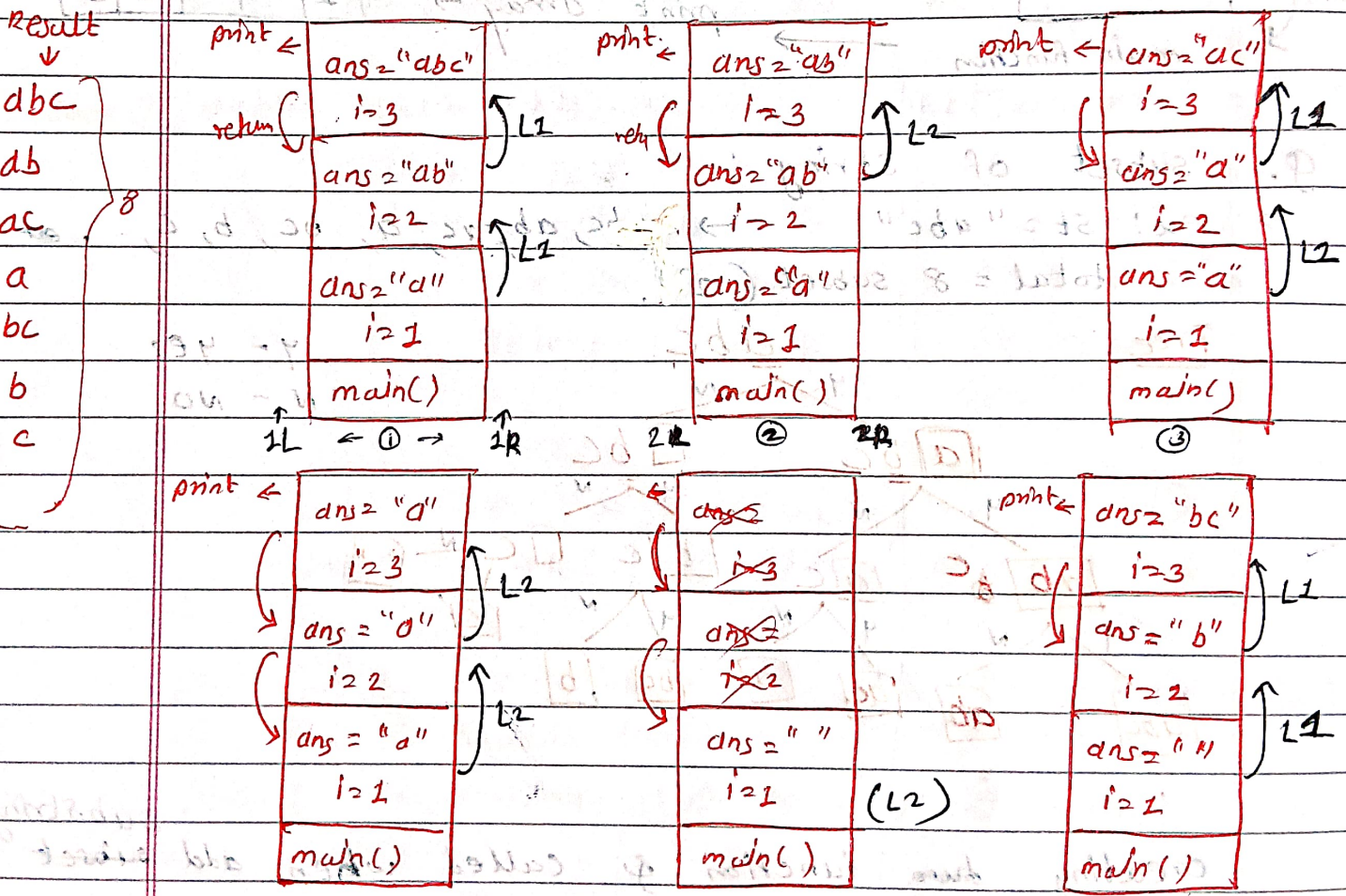
subset (st, ans + st.charAt(i), i+1)

subset (st, ans, i+1);

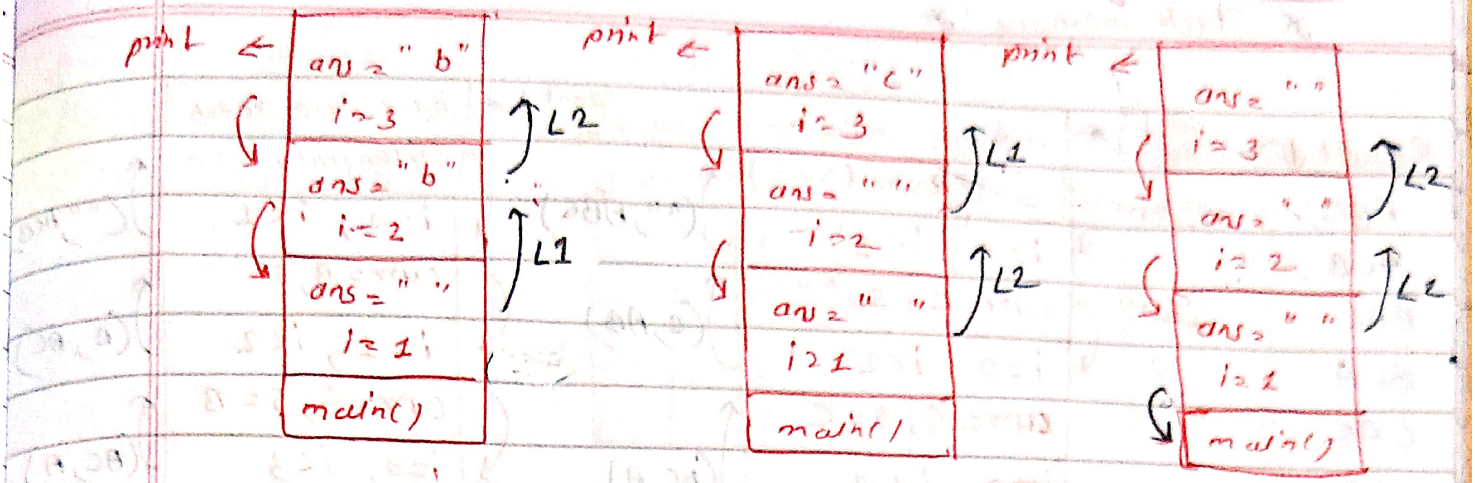
//code//
```
public static void subset(String st, String ans, int i){
    // base case //
    if(i==st.length()){
        System.out.println(ans);
        return;
    }
    // kaam //
    subset(st, ans+st.charAt(i), i+1);
    subset(st, ans, i+1);
}
```

main ⇒ `subset(st, "", 0);`

Stack memory for st = "abc" :



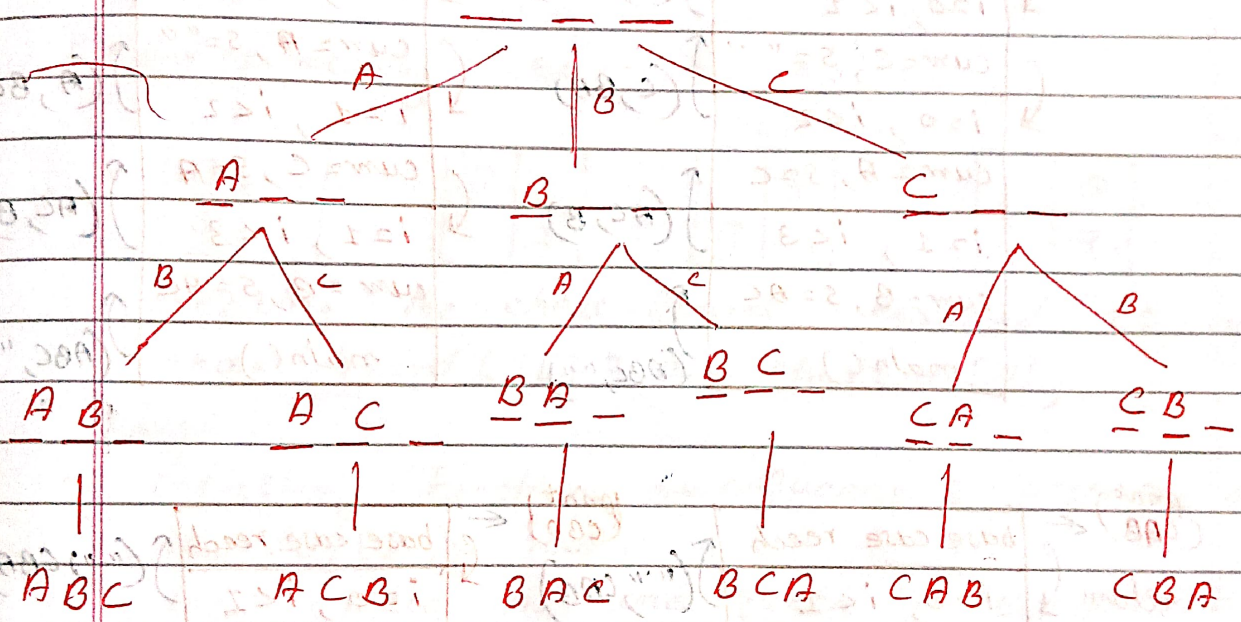
now read this stack :→ !)



**Q. Permutation of String.

→ Ex: St = "ABC"

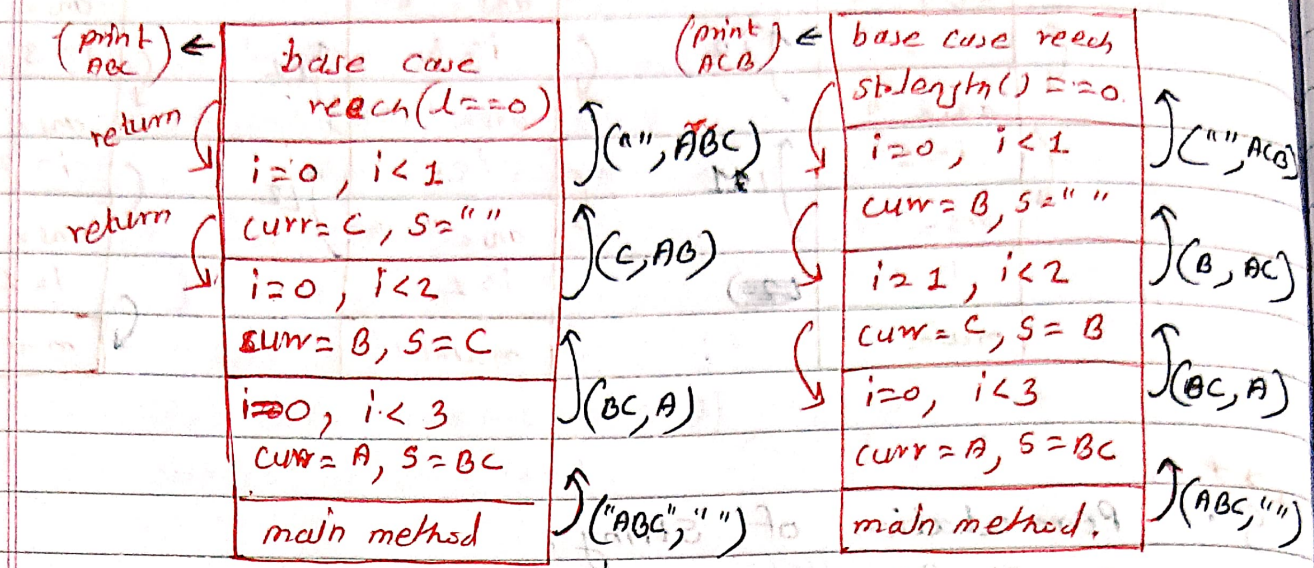
O/P: ABC, ACB, BAC, BCA, CBA, CAB. ... (n!)

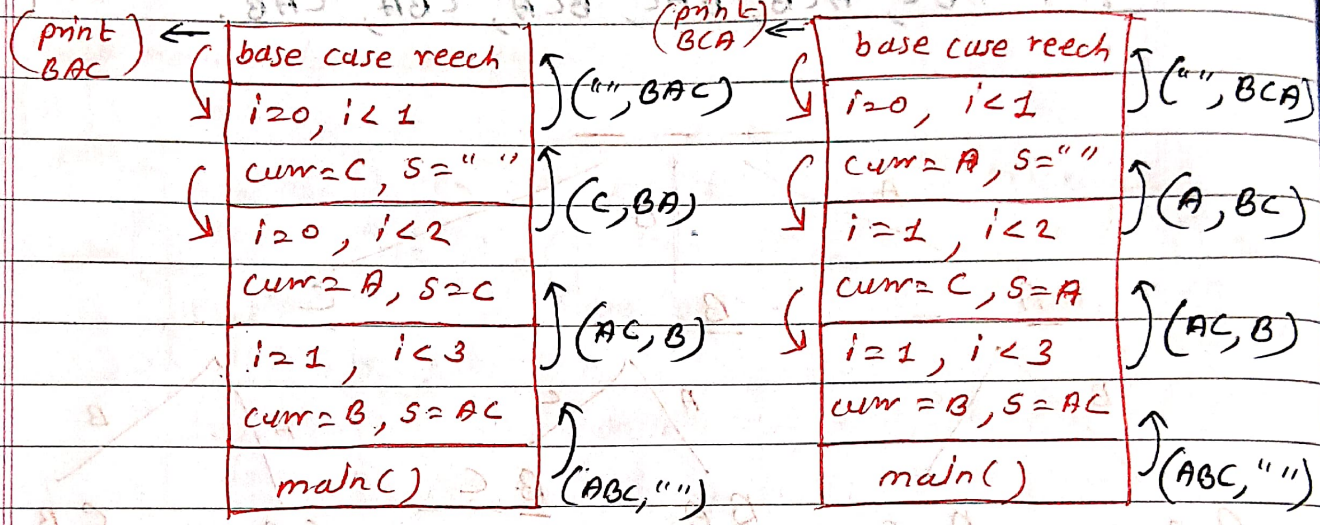


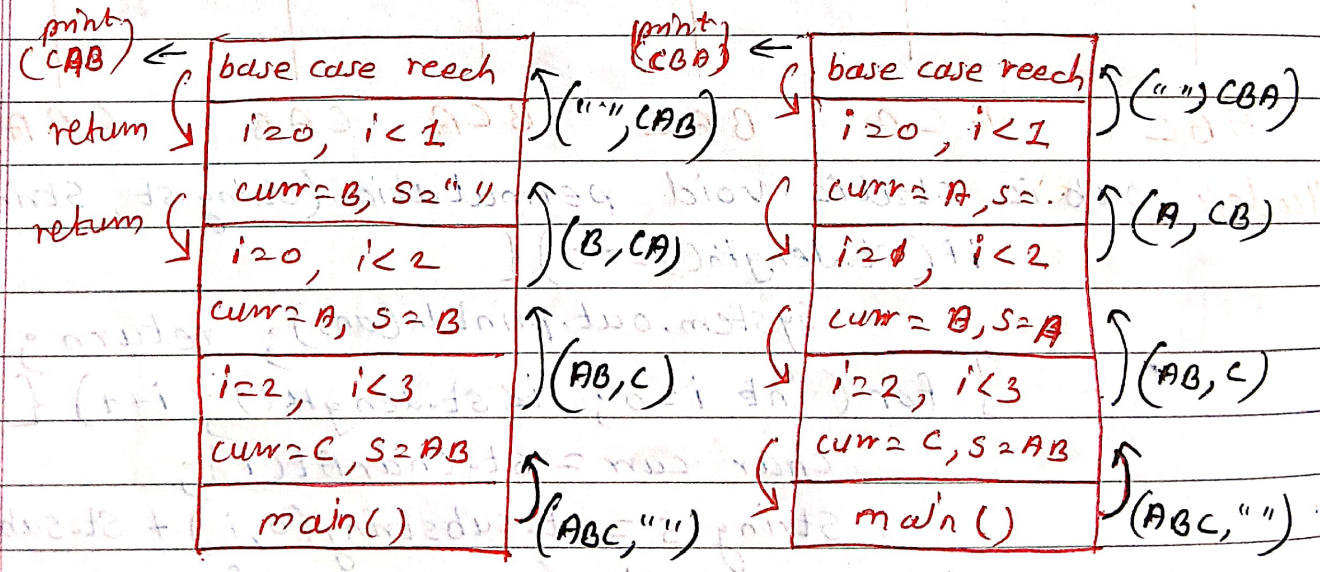

```
//code  public static void permutation(String st, String ans){
        if(st.length()==0){
            System.out.println(ans); return;
        } for(int i=0; i<st.length(); i++){
            char curr = st.charAt(i);
            String s = st.substring(0,i) + st.substring(i+1);
            permutation(s, ans+curr);
        }
}
```

* Stack memory : *

Result ↓
ABC
ACB
BAC
BCA
CAB
CBA



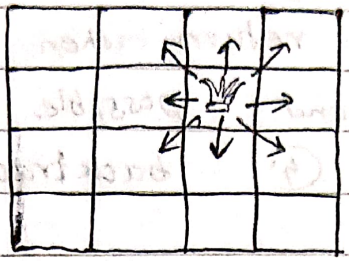




(end recursion)

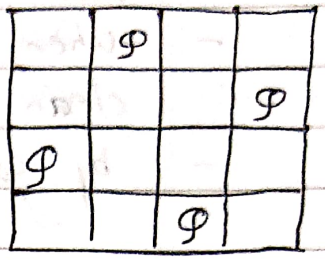
Q. N-queens :

- Place ~~N*N~~ N-queens on an N×N chessboard such that no 2 queens can attack each other.
- ex : N = 4



direction of queen can be move

ans :

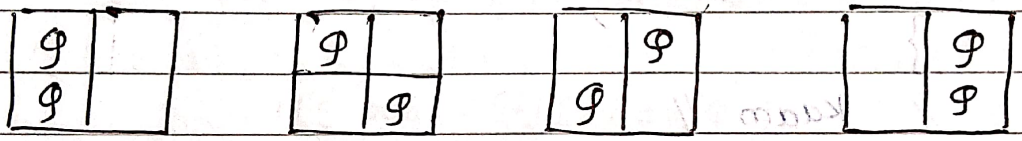


does not attack to each other.

Approach

1) Set one queens in each row but diff. column.
2) Check which place can be safe for queen.
3) print no. of ways of sol^n.

line 1 meaning : ex - N = 2



creating the chess board required 2D-array.

```
char board[][] = new char[N][N];
```

logic :

- creating function ~~fn~~ nQueens & pass the parameter board[][], & row.
- then place the one queen in every row ~~but diff, column~~.

```
① for (int j=0; j< board.length; j++) {
②       board[row][j] = 'Q';
③       nQueens(board, row+1);      - recursion
④       board[row][j] = '-';        - backtrack
    }
```

line ③ → place the N-queens, by calling function when base reech then print board & return then board is

- when function return then required board is clean for second possible ~~set~~ seating queen
- by using line ④ backtrack for board clean.
- write down the code for check every queen can be safe to place in every place.
- this cond^n apply before seating the queen.

```
//code//
public static void nQueens(char board[][], int row){
    // base case //
    if(row == board.length){
        printBard(board);          — function
        return;
    }
    // kaam //
    for(int j=0; j<board.length; j++){
        if(isSafe(board, row, j)){
            board[row][j] = 'Q';
            nQueens(board, row+1);
            board[row][j] = '-';
        }
    }
}
```

function → printBoard (char board[][]){
    // nested loop to print 2D array //
}

time complexity = O(n!)

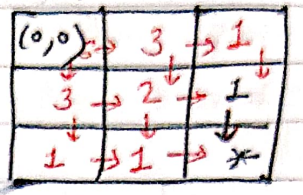
```
// function to check queen is safe //
public static boolean isSafe(board, row, col){
    // In column //
    for(int i=row-1; i>=0; i--){
        if(board[i][col]=='Q'){
            return false;
        }
    }
    // In left upper diagonal //
    for(int i=row-1, j=col-1; i>=0 && j>=0; i--, j--){
        if(board[i][j]=='Q'){
            return false;
        }
    }
    // In right upper diagonal //
    for(int i=row-1, j=col+1; i>=0 && j<board.length
                                  i--, j++){
        if(board[i][j]=='Q'){
            return false;
        }
    }
    return true;
}
public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            board[i][j] = '-';
        }
    }
    nQueens(board, 0);
}
```

Q. Grid ways :

- find numbers of ways to reach from (0,0) to (N-1, M-1) in a NXM Grid Grid.
- Allowed Moves - right or down.

→ ex: N = 3
M = 3

total = f(i, j+1) + f(i+1, j)



6 ways

```
// code //
public static int gridWays(int i, int j, int right, int down){
    if(i==right-1 && j==down-1){
        return 1;
    }else if(i==right || j==down){
        return 0;
    }
// kaam //
    int w1 = gridWays(i, j+1, right, down);
    int w2 = gridWays(i+1, j, right, down);
    return w1+w2;
}
```

time complexity = $O(2^{n+m})$ right turn = no. of col = m
down turn = no. of row = n

shortcut trick : $ans = \frac{(n-1+m-1)!}{(n-1)!\,(m-1)!}$

ex: n=3, m=3

$ans = \frac{(3-1+3-1)!}{2!\,.\,2!} = \frac{4!}{4} = 6$ ways

g. write the function to complete sudoku.

– rule: same column & same row does not reapet number.

* Arraylist :

* Multidimentional Arraylist

↳ ArrayList < ArrayList < Integer >> list = new ArrayList<>()

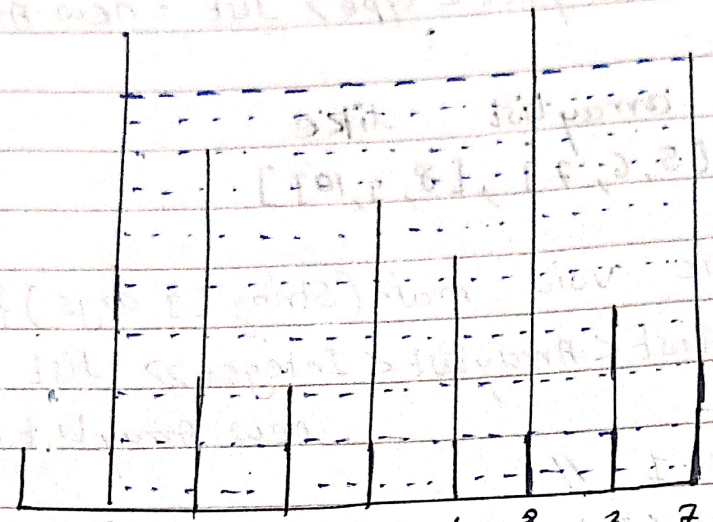
Normal → 1D = ArrayList < type > list = new ArrayList<>();

Q. create 2D arraylist like
[[2,3,4], [5,6,7], [8,9,10]]

```
public static void main(String [] args){
        ArrayList < ArrayList < Integer >> list =
                                    new ArrayList <> ();
    // list 1 //
    ArrayList < Integer > list1 = new ArrayList <> ();
    list1.add (2);
    list1.add (3);
    list1.add (4);
    // list 2 //
    ArrayList < Integer > list2 = new ArrayList <> ();
    list2.add (5);
    list2.add (6);
    list2.add (7);
    ArrayList < Integer > list3 = new ArrayList <> ();
    list3.add (8);
    list3.add (9);
    list3.add (10);
    // add all list in 2D list //.
    list.add(list1);
    list.add (list2);
    list.add (list3);
}
```

Q. Calculate stored water in container. given height of container.

→ height = [1, 8, 6, 2, 5, 4, 8, 3, 7]



height → 1 8 6 2 5 4 8 3 7

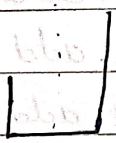
idx → (0) (1) (2) (3) (4) (5) (6) (7) (8)

→ Solve this question by ~~assume~~ any two pole & assume two pole is container then calculate the max store water.

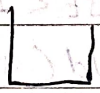
- bet^n two pole weidth is 1.

assume: for ex:

1) bet^n i=0 & i=2 →



2) bet^n i=1 to i=5 →



3) bet^n i=3 to i=4 →

... etc selecting any two pole & calculate all container area & compare area with currarea and print max area.

for this approch use the concept of possible pair of Array.

//code//
```
public static void storedWater(ArrayList<Integer> height)
{
    // brute force approch //
    int l = height.size();
    int maxWater = 0;
    int b1=0, b2=0;
    for(int i=0; i<l; i++){
        for(int j=i+1; j<l; j++){
            int hei = Math.min(height.get(i), height.get(j));
            int wei = j-i;
            int curWater = hei * wei;
            if(maxWater < curWater){
                b1 = height.get(i);
                b2 = height.get(j);
                maxWater = curWater;
            }
        }
    }
    System.out.println("Max water height bandries :" +
                                        b1+", "+b2);
    System.out.println("Max water contain : "+maxWater);
}
```

time complexity = O(n²)

```
    // two pointer approach //.
public static void storedWater(ArrayList ...){
    int l=height.size(), lp=0, rp=0, storedWater=0;
    while(lp<rp){
        if(height.get(lp) < height.get(rp)){
            lp++;
        }else{
            rp--;
        }
        int hei = Math.min(height.get(lp), height.get(rp));
        int wei = rp-lp;
        int curWater = hei * wei;
```

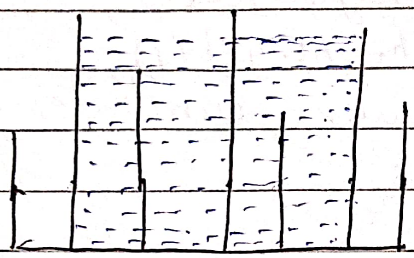
//code//

time complexity
O(n)
linear t.c.

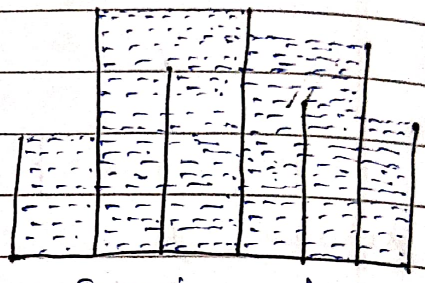
```
            if(storedWater < curWater){
                storedWater = curWater;
            }
        }
        System.out.println("Max stored water : " + storedWater);
    }
```

* Trapping rain water → stared all water betn all poles
* contain max stored water → water stored betn two poles.

ex:



water store container.



Trapping rain water

Q. Pair sum to equal target number in sorted list

→ ex: list = [1, 2, 3, 4, 5] target = 8

ans = 2, 4 → idx.

//code// brute force approach $O(n^2)$.

```
public static void pairSum(list, target){
    int l = list.size();
    for(int i=0; i<l; i++){
        for(int j=i+1; j<l; j++){
            if(list.get(i) + list.get(j) == target){
                printf("Index : %d, %d", i, j);
                break;
            }
        }
    }
}
```

this approach use sorted & unsorted list/array.

```
// two pointer approach // O(n)
public static void pairSum(List, int target){
    int l = list.size();
    int lp=0, rp=l-1;
    while(list.get(lp)+list.get(rp) != target){
        if(list.get(lp)+list.get(rp) < target){
            lp++;
        }else{
            rp--;
        }
    }
    System.out.printf("Index: %d, %d", lp, rp);
}
```

Q. Pair sum of Sorted Roteted List.

→ ex: list = [4, 5, 6, 1, 2, 3]. target = 10 → (0, 2).

Step 1: first find pivot point

Step 2: then point the lp & rp

Step 3: use formula $lp = (lp+1) \% l$ & $rp = (rp-1+l) \% l$ } for cyclic move.

Note: % is use in diff. place in competitive programming (search it).

//code//

```
public static void pairSumRotatedList(list, target){
    int lp=0, rp=0;
    int l = list.size();
    for(int i=0; i<l-1; i++){
        if(list.get(i) > list.get(i+1)){
            rp=i; lp=i+1;
            break;
        }
    }
}
```

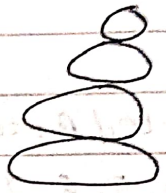
O(n).

```
while(list.get(lp) + list.get(rp) != target){
    if(                          < target) {
        lp = (lp + 1) % l;
    } else {
        rp = (rp - 1 + l) % l;
    }
}
System.out.printf("Idx : %d, %d", lp, rp);
}
```

* Stack :

- ex: coins, books in stack.

coins



stones.

* operation :

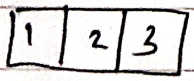
1) Push : O(1)
2) pop : O(1)
3) peek : O(1)

- LIFO → Last in first out

* Implementation :

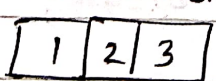
Array

fix size

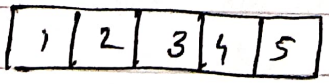


ArrayList

variable size



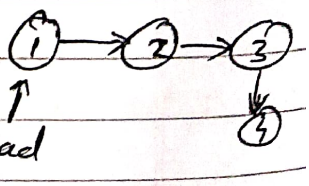
top



top

LinkedList

variable size



Note: for implemente stack best is ArrayList & LinkedList.

1) Emplement stack by using ArrayList.

first import java.util.ArrayList;

//code//

```
static class Stack {
        public static boolean isEmpty() {
                return list.size()==0;
        }
        // push //
        public static void push(int data) {
                list.add(data);
        }
        // peek //
        public static int peek() {
                return list.get(list.size()-1);
        }
        // pop //
        public static int pop() {
                int top = list.get(list.size()-1);
                list.remove(list.size()-1);
                return top;
        }
}
public static void main(String [] args) {
        Stack stack = new Stack();
        stack.push(1); stack.push(2); stack.push(3);
        while(!stack.isEmpty()) {
                System.out.println(stack.peek());
                stack.pop();
        }
}
```

o/p: 3
2
1

OYO
Q* By using linkedlist implement Stack

//code//
```
static class Node {
    int data;
    Node next;
    Node (int data) {
        this.data = data;
        this.next = null;
    }
}
static class Stack {
    static Node head = null;
    // empty //
    public static boolean isEmpty () {
        return head == null;
    }
    // push //
    public static void push (int val) {
        Node temp = new Node(val);
        if ( head == null) {
            head = temp;
            return;
        }
        temp.next = head;
        head = temp;
        return;
    }
    // peek //
    public static int peek () {
        return head.data;
    }
```

```
// pop //
public static int pop (){
        int top = head.data;
        head = head.next;
        return top;
    }
}
public static void main (String [] args) {
    Stack s = new Stack();
    s.push (1);
    s.push (2);
    s.push (3);
    while (!isEmpty()){
        System.out.println (stack.peek());
        stack.pop();
    }
}
```

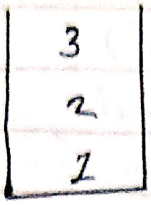
*. Stack using framework:

```
//code// import java.util.*;
public static void main (String args []) {
    Stack <Integer> s = new Stack();
    s.push (1);
    s.push (2);
    s.push (3);
    while (!isEmpty()){
        System.out.println (s.peek());
        s.pop();
    }
}
```

Amazon

Q. Push at the bottom of the stack.

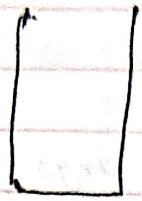
→ stack = Ex:



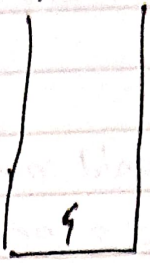
and add 4 in bottom

- first to do empty of stack then add 4 then add push add data.

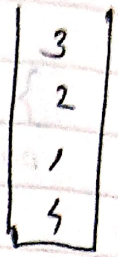
empty -



add 4 →



add old data →



- These question also solve by using loop. but better ans by recursively.

// code//

```
public static void putBottomOfStack (Stack<Integer>
                                     S1, int num ) {
    // base case //
    if(S1.isEmpty()){
        S1.push(num);
        return;
    }
    // kaam //
    int data = S1.pop();
    putBottomOfStack (S1, num);   — called
    S1.push (data);               — backtrack
}
public static void main (String[] args){
    Stack<Integer> S1 = new Stack();
    S1.push(1); S1.push(2); S1.push(3);
    putBottomOfStack (S1, 4);
    while(!S1.isEmpty()){
        System.out.println(S1.pop());
    }
}
```

O(n)

O/P: 3
2
1
4

Imp

Q. Reverse the string by using stack.

→ In method pass String as parameter & create Character Stack in method & stored each character in stack and print.

Ex: St = "abc" → | c | b | a | → "c b a"

- when then create String builder and append each character (in present of stack) by using. sb.append(s.pop()). (use while loop).
- In last convert stringbuilder to string [sb.toString();]

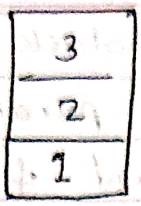
```
//code//  public static void reverseString(Stack String st) {
            Stack<Character> s = new Stack();
            int i=0, l = st.length();
            while (i != l) {
                s.push(st.charAt(i));
                i++;
            }
            StringBuilder sb = new StringBuilder("");
            while (!s.isEmpty()) {
                sb.append(s.pop());
            }
            return sb.toString();
        }
```

Imp

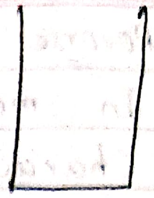
Q. Reverse stack. (by recursion).

→ - by using recursion to do empty stack.
- when recursion is return then backtrack by function → putAtBottom();
- Using this function ~~each~~ each element go to the bottom when recursion is return.

ex: a = 1 2 3

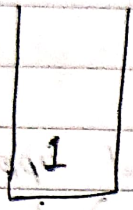


→ by using recursion → (but store each value in top variable)



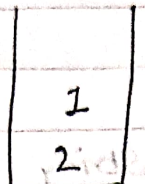
} empty

1) when return

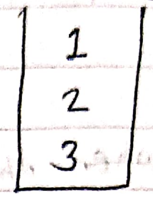


→

2) when return



3) when return



call by putAtbottom(top)   reverse Stack

//code//

```
public static void putAtbottom(Stack<Integer> s, int num){
    // base case //
    if(s.isEmpty()){
        s.push(num);
        return;
    }
    int top = s.pop();
    putAtBottom(s, num);
    s.push(top);
}

public static void reverseStack(Stack<Integer> s){
    // base case //
    if(s.isEmpty()){
        return;
    }
    int top = s.pop();
    reverseStack(s);
    putAtbottom(s, top);
}
```

very Imp Logic

classmate
Date 14-7-24
Page 65

9. Next greater Element in array (time comp = O(n)).

→ this question also solve by nested loop but time complexity is $O(n^2)$. hence, these question solve by stack.

ex: arr =

8	6	4	7	4	9	10	8	12

ans =

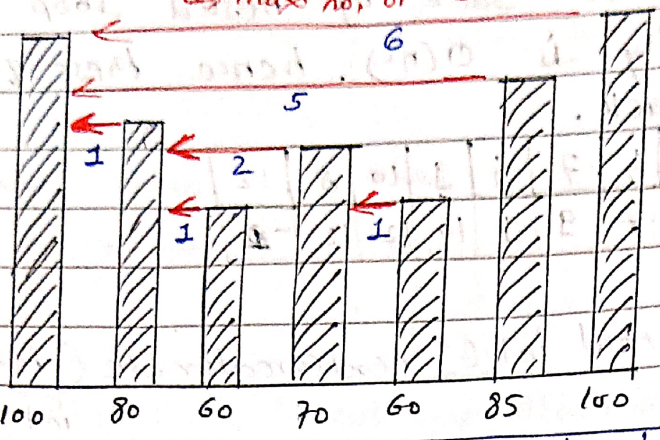
9	7	7	9	9	10	12	12	-1

```
public static ~~void~~ int[] nextGreaterEle(int arr[],
                                            int ans[]){
    Stack<Integer> s = new Stack();
    int l = arr.length;
    for(int i=0; i<l; i++){                          arr[i]
        while(!s.isEmpty() && arr[s.peek()]<1){
            ans[s.peek()] = arr[i];
            s.pop();
        }if(s.isEmpty() || i==l-1) ans[i] = -1;
        s.push(i);
    }
    return ans;
}
```

O(n)

Q. Stock span problem.

↳ max no. of consecutive day.



time complexity O(n)

Stock [] =	100	80	60	70	60	85	100
Span []	1	1	1	2	1	5	6

← return this ans

Note: these question also solve by nested loop

Step 1) creat stack of Integer type & push 0.
2) stored 1 in span array at zeroth index.
3) while stack is not empty & prices < todayprice that mean stock[s.peek()] < stock[i] then pop out of the top value.
4) Store in span → i - s.peek() in out of while loop
5) push each index in stack s.push(i);
6) print / return span array.

```
//code//  public static int[] stockSpan(int[] stock){
            Stack<Integer> s = new Stack();
            int span[] = new int[stock.length];
            s.push(0);
            span[0] = 1;
            for(int i=1; i<stock.length; i++){
                while(stock[s.peek()]<stock[i]){
                    s.pop();
                } span[i] = i - s.peek();
                s.push(i);
            } return span;
        }
```

9. Valid Parentheses :

- Given a string a containing just the characters '(', ')', '{', '}', '[', ']', determine if the input string is valid.
- An input string is valid if :

1) Open brackets must be closed by the same type of brackets.
2) Open brackets must be closed is the correct order.
3) Every close bracket has a corresponding open bracket of the same type.

ex: st = "[](())" - valid
st = "{[]}" - Invalid

```
//code//  public static void String validParentheses (String st) {
            Stack <Character> s = new Stack();
            for (int i=0; i<st.length(); i++) {
                Char ch = st.charAt(i);
                if (ch == '[' || ch == '(' || ch == '{') {
                    s.push(ch);
                } else {
                    if (s.isEmpty()) {
                        return "Invalid";
                    }
                    if (s.peek() == '(' && ch == ')' ||
                        s.peek() == '[' && ch == ']' ||
                        s.peek() == '{' && ch == '}' ) {
                        s.pop();
                    } else {
                        return "Invalid";
                    }
                }
            } if (s.isEmpty()) return "Valid";
              else return "Invalid".
```

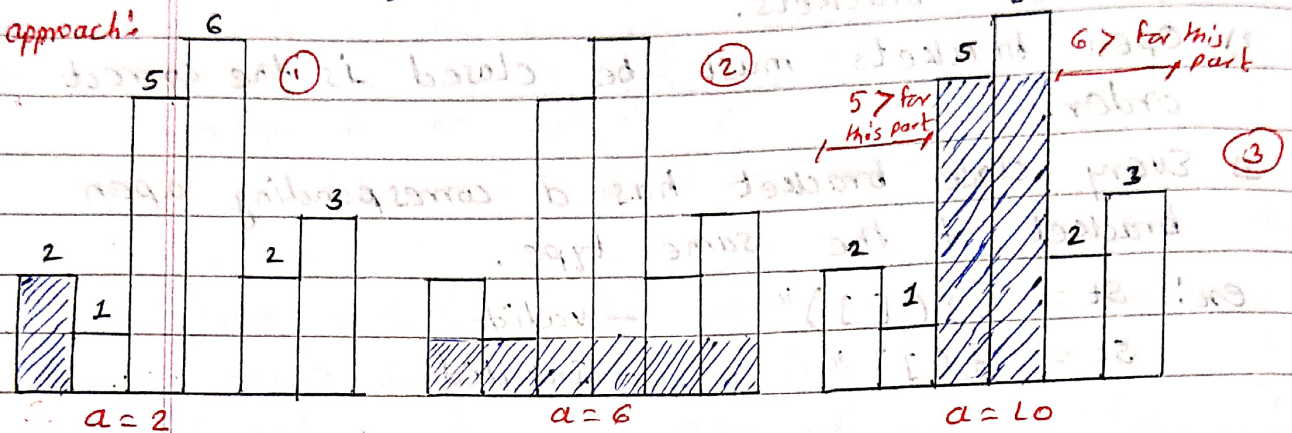
* * * hard.

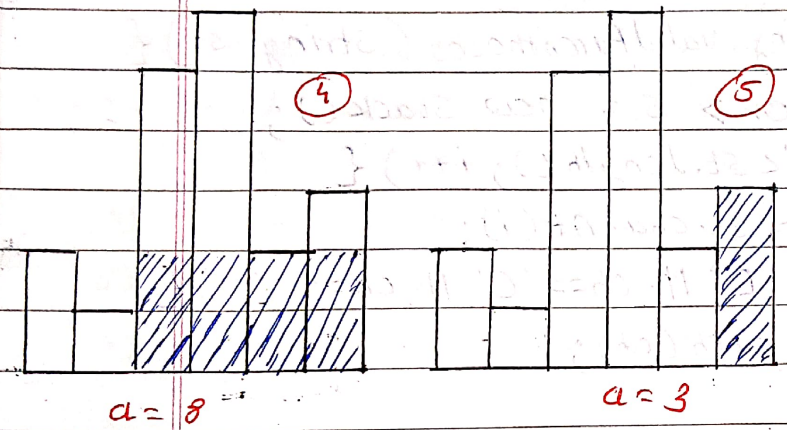
Q. Max Area in Histogram : (use logic of next greater number)

Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.

→ ex: height [] = 2, 1, 5, 6, 2, 3

approach:





∴ area = hei × wei

wei

next smaller right (j) idx — next smaller left (i) idx

∴ wei = j - i - 1

	-1	-1	1	2	2	4
∴ next smaller left (nsl) i =	-1	-1	1	5	1	2
∴ next smaller right (nsr) j =	1	6	2	2	6	6
	1	1 (6)	4	4	1 (6)	1 (6)

for understanding stored value but actual stored the index.

for ①:
a = h × w
a = 2 × (j - i - 1)
a = 2 × (1 + 1 - 1)
a = 2

for ②:
a = h × w
a = 1 × (6 + 1 - 1)
a = 6

for ③:
a = h × w
a = 5 × (4 - 1 - 1)
a = 10

for ④:
a = 6 × (4 - 2 - 1)
a = 6

for ⑤:
a = 3 × (6 - 4 - 1)
a = 3

//code//

```
public static void maxArea(int arr[]) {          // O(n).
    int maxArea = 0, l = arr.length;
    int nsl = new int[l];
    int nsr = new int[l];
    // next smaller right //
    Stack<Integer> s = new Stack();
    for(int i=l-1; i>=0; i--) {
        while(!s.isEmpty() && arr[s.peek()] >= arr[i]) {
            s.pop();
        }
        if(s.isEmpty()) {
            nsr[i] = l;
        } else {
            nsr[i] = s.peek();
        }
        s.push(i);
    }
    // next smaller left //
    s = new Stack<>();
    for(int i=0; i<l; i++) {
        while(!s.isEmpty() && arr[s.peek()] >= arr[i]) {
            s.pop();
        }
        if(s.isEmpty()) {
            nsl[i] = -1;
        } else {
            nsl[i] = s.peek();
        }
        s.push(i);
    }

    // curr area = width = j - i - 1 = nsr[i] - nsl[i] - 1 //
```

O(n)

O(n)

$O(n)$

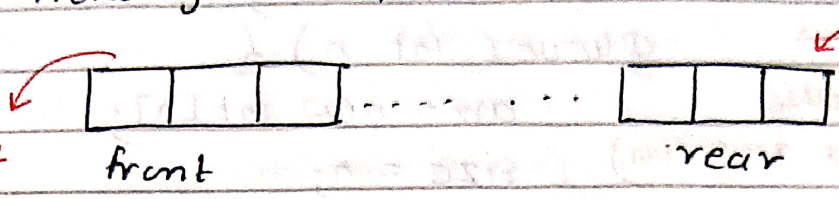
```
for( int i=0; i<l ; i++) {
    int hei = arr[i];
    int wid = nsr[i] - nsl[i] - 1;
    int curArea = hei * wid;
    maxArea = Math.max(curArea, maxArea);
}
System.out.println("Max area in histogram is ");
System.out.println( maxArea );
```

Date 18-7-24
Page 21

Queue:

- queue is base on FIFO me logic:
- it is linear data d structure that used to stored elements
- there are two end of the queue collection i.e. frond & rear,

ex:



- The queue is an interface In the java that belongs to java.util package. It is also extends the Collection interface
- there are two different classes that are used to implement the queue interface. these class are LinkedList & PriorityQueue.

- Implementation:

Array	LinkedList	Stack.
• fixed size	• does not	
• time complexity of remove() is O(n)	fix size	
- circular queue	• time complexity for remove O(1)	

Note: queue data structure also present in java collection.

import java.util.*; it interface, does not make obj

∴ Queue<Integer> q = new LinkedList<>();
Queue<Integer> q2 = new ArrayDeque<>();

Note: find difference bet' them,

```
public static class QueueImp {
    static class Queue {
        static int arr[];
        static int size;
        static int rear;

        Queue(int n) {
            arr = new int[n];
            size = n;
            rear = -1;
        }

        public static boolean isEmpty() {
            return rear == -1;
        }

        // add function //
        public static void add(int data) {
            if(rear == size-1) {
                println("queue is full");
                return;
            }
            rear = rear+1;
            arr[rear] = data;
        }

        // remove function //
        public static int remove() {
            if(isEmpty()) {
                println("queue is empty");
                return -1;
            }
            int front = arr[0];
            for(i=0; i<rear; i++) {
                arr[i] = arr[i+1];
            }
            rear = rear-1;
            return front;
        }
```

constructor → (by the fault value to create obj of queue class)

O(1)

O(n)

```
    // peek function //
    public static int peek(){
        if(isEmpty()){
            println("empty queue");
            return -1;
        }
        return arr[0];
    }
}
public static void main(String[] args){
    Queue q = new Queue(5);
    q.add(2);
    q.add(3);
    q.add(4);
    while(!q.isEmpty()){
        System.out.println(q.peek()); q.remove();
    }
}
}
```

o/p: 2
3
4

* Circular queue:

```
public static class Queue{
    static int size;
    static int arr[];
    static int rear;
    static int front;

    // constructor //
```

```
queue(int n){
    size = n;
    arr = new int[n];
    rear = -1;
    front = -1;
}
// is empty //
public static boolean isEmpty(){
    return front == -1 && rear == -1;
}
// is Full //
public static boolean isFull(){
    return (rear+1) % size == front;
}
// add ele //
public static void add(int data){
    if(isFull()){
        System.out.println("queue is Empty");
        return;
    }
    if(front == -1){
        front = 0;
    }
    rear = (rear+1) % size;
    arr[rear] = data;
}
// peek ele //
public static int peek(){
    if(isEmpty()){
        println("queue empty");
        return -1;
    }
    return arr[front];
}
```

```
// remove element //
public static int remove(){
    if(isEmpty()){
        println("queue empty");
        return -1;
    }
    int result = arr[front];
    if( front == rear){
        front = rear = -1;
    }else{
        front = (front+1) % size;
    }
    return result;
  }
}
public static void main(String[] args){
    Queue q = new Queue(4);
    q.add(1);
    q.add(2);
    q.add(3);
    q.add(4);
    while(!q.isEmpty()){
        System.out.println(q.peek());
        q.remove();
    }
}
```

for all method time complexity O(1).

* Queue using Linked_List

```
//code//  public static void
Static class Node {
        int data;
        Node next;
        Node (int data) {          — Constructor.
            this.data = data;
            next = null;
        }
}
static class queue {               — class.
        Static Node head = null;
        Static Node tail = null;
        // null list //
        public static boolean isEmpty() {
            return head == null && tail == null;
        }
        // add element //
        public static void add(int data) {
            Node temp = new Node (data);
            if(head == null) {
                head = temp;
            }else {
                tail.next = temp;
            }
            tail = temp;
        }
        // peek element //
        public static int peek() {
            return head.data;
        }
```

```
// remove head / first ele //
public static int remove(){
    if(isEmpty()){
        System.out.println("Empty queue");
        return -1;
    }
    Node temp = head;
    head = head.next;
    return temp.data;
}
}
public static void main(String [] args){
    Queue q = new Queue();
    q.add(1);
    q.add(2);
    q.add(3);
    while(!q.isEmpty()){
        System.out.println(q.peek());
        q.remove();
    }
}
```

o/p: 1
2
3

* Queue Implement by using java collection.

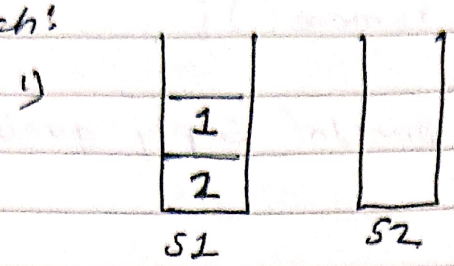
//code//

```
public static void main(String [] args){
    Queue<Integer> q = new LinkedList();
    q.add(11);
    q.add(22);
    q.add(33);
    // also implement by //
    Queue<Integer> q2 = new ArrayDeque();
}
```

IMP

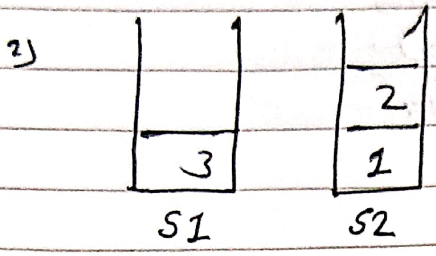
Q. Queue using two stack.

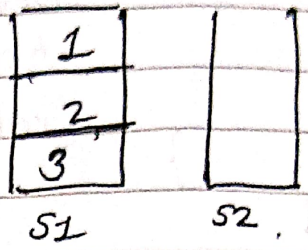
approach:



For add 3

1) all ele push in S2 stack
2) push new data
3) push all data in S1 from S2.





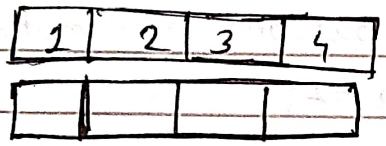
//code// static class queue {

```
    static Stack<Integer> S1 = new Stack<>();
    static Stack<Integer> S2 = new Stack<>();
    // is empty //  - O(1)
    public static boolean isEmpty() {
        return S1.isEmpty();
    }
    // add //  - O(n)
    public static void add(int data) {
        while (!S1.isEmpty()) {
            S2.push(S1.pop());
        }
        S1.push(data);
        while (!S2.isEmpty()) {
            S1.push(S2.pop());
        }
    }

    // peek //  - O(1)
```

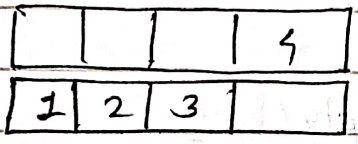
```
public static int peek(){
    if(s1.isEmpty()){
        System.out.println("Empty queue");
        return -1;
    }
    return s1.peek();
}
// remove // - O(1)
public static int remove(){
    if(s1.isEmpty()){
        System.out.println("Empty queue");
        return -1;
    }
    return s1.pop();
}
}
```

Q. Stack using two queue.

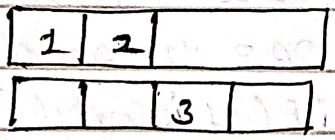
ex: 

1) add all element in second queue, if last ele then return

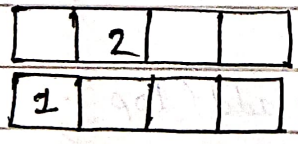
approach.

1) 

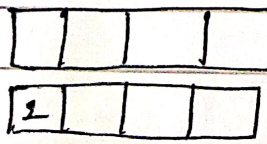
— return (4).

2) 

— return (3).

3) 

— return (2)

4) 

— return. (1).

O/p: 4
3
2
1

// code //

```
public static class Stack {
    static Queue<Integer> q1 = new LinkedList<>();
    static Queue<Integer> q2 = new LinkedList();

    public static boolean isEmpty() {
        return q1.isEmpty() && q2.isEmpty();
    }
    // push function //
    public static void push(int data) {
        if(q1.isEmpty()){
            q1.add(data);
        }else{
            q2.add(data);
        }
    }
    // pop function //
O(n) public static int pop() {
        if(isEmpty()){
            System.out.println("Empty Stack");
            return -1;
        }
        int top= -1;
        // case 1 //
        if(!q1.isEmpty()){
            while(!q1.isEmpty()){
                top = q1.remove();
                if(q1.isEmpty()){
                    break;
                }
                q2.add(top);
            }
        }
```

```
        else{
            while(!q2.isEmpty()){
                top = q2.remove();
                if(q2.isEmpty()){
                    break;
                }
                q2.add(top);
            }
        }
        return top;
    }
    public static int peek(){
O(n)    if(isEmpty()){
            println("empty stack");
            return -1;
        }
        int top=-1;
        if(!q1.isEmpty()){
            while(!q1.isEmpty()){
                top = q1.remove();
                q2.add(top);
            }
        }else{
            while(!q2.isEmpty()){
                top = q2.remove();
                q1.add(top);
            }
        }
        return top;
    }
}
```

Note: In any competative programming contest question is steam of character/etc these are solve by max. time by queue.

Q. first non-repeating letter in a stream of character

ex: str = aabccxb ans = x.

Stream = y d l ε, flow,

approach:

① create freq array to track elem character how many times repeat.

② create queue to store current character,

③ for (int i=0; i<st.length) {

ch = str.charAt(i)

q.add(ch);

freq[ch-'a']++; ← Imp logic

q.remove() ← this process till 1st non repeat element.

if(isempty()) → return -1;

}

return q.peek();

//code// public static void firstNonRepeatingEle(String str){

int freq[] = new int[26];

queue<character> q = new LinkedList();

for(int i=0; i<str.length(); i++){

char ch = str.charAt(i);

q.add(ch);

freq[ch-'a']++;

// check ele is repeat //

while(!q.isEmpty() && freq[q.peek()-'a']>1){

q.remove();

}

~~if(q.isEmpty()){~~

println(q.peek()); ~~out.print(-1+" ");~~

~~}else{~~

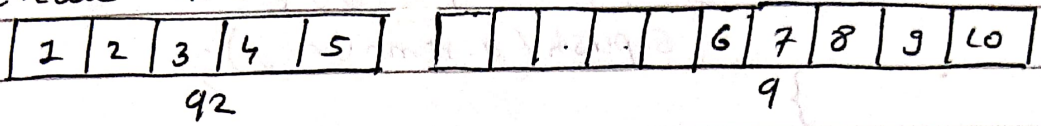
~~System.out.print(q.peek()+" ");~~

o/p : a -1 b b b x ;

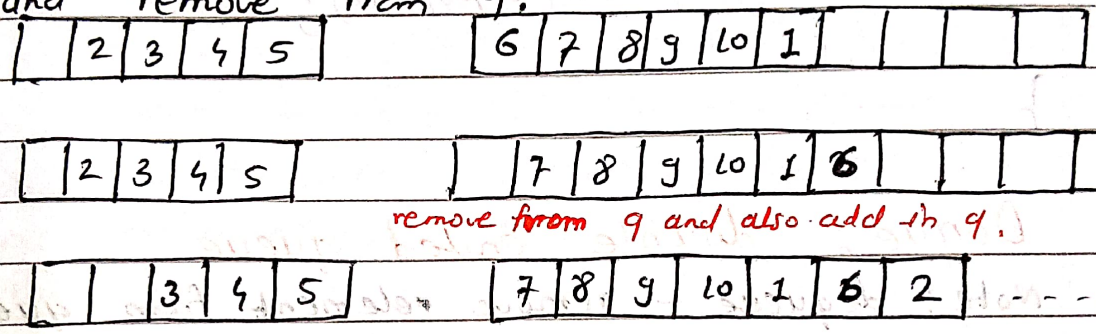
Q. Interleave 2 halves of a queue (even no. length)
→ ex: Input queue → 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
break this queue bet^n them and add given type
ans = 1, 6, 2, 7, 3, 8, 4, 9, 5, 10.

approach:

1) create queue and stored first half data.



2) add simultaneously element in q (original I/p queue) and remove from q.



3) return q.

```
//code
public static Queue<Integer> interleaveHalves(
                                Queue<Integer> q ) {
    Queue<Integer> q2 = new LinkedList<>();
    int l = q.size();
    for( int i=0; i< l/2 ; i++) {
        q2.add(q.remove());
    }
    for( int i=0; i< l ; i++) {
        if( i%2==0) {
            q.add(q2.remove());
        }else{
            q.add(q.remove());
        }
    }return q;
}
```

(*) Note: learn & study of how to implement vector in java & what is your use and difference of vector, ArrayList etc.

Q. Reverse of queue :
ex: q = 1, 2, 3, 4, 5 ans = 5 4 3 2 1

approch: 1) create new stack
2) push all ele in stack from queue.
3) add all ele in queue from stack

//code
```
public static void reverseQueue (Queue<Integer> q){
    Stack<Integer> s = new Stack();
    while (!q.isEmpty()){
        s.push(q.remove());
    }
    while (!s.isEmpty()){
        q.add(s.pop());
    }
}
```

(*) Deque : double ended queue

Note: dequeue - remove element from queue (action)
- Deque & Dequeue are different.

- first import java.util.*;

methods: 1) addFirst(data);
2) addLast(data);
3) getFirst();
4) getLast();
5) removeFirst();
6) removeLast();

//code
```
Deque<Integer> deq = new LinkedList();
deq.add(11);
deq.add(22);
deq.add(33);
System.out.println(deq);
}
```

o/p: [11, 22, 33]

Q. Stack & Queue using deque.

//code//

```
static class Stack {
    static Deque<Integer> d = new LinkedList();
    public static void push(int data) {
        d.addLast(data);
    }
    public static int peek() {
        return d.getLast();
    }
    public static int pop() {
        return d.removeLast();
    }
}
public static void main(String[] args) {
    Stack s = new Stack();
    s.push(11);
    s.push(22);
    s.push(33);
    System.out.println(s.pop());
    System.out.println(s.pop());
    System.out.println(s.pop());
}
```

o/p: 33
22
11

Q. Queue using Deque.

//code//

```
static class Queue {
    static Deque<Integer> d = new LinkedList();
    public static void add(int data) {
        d.addLast(data);
    }
    public static void remove() {
        return d.removeFirst();
    }
}
```

Greedy Algorithm:

Greedy - लालची (english meaning)

- In Java: Greedy algorithm is the problem solving technique where we make the locally optimum choice at each stage & hope to achieve a global optimum.

Pros: Simple & Easy good enough TC (time. complex).
Cons: A lot of time, global optimal is not achieved.

***

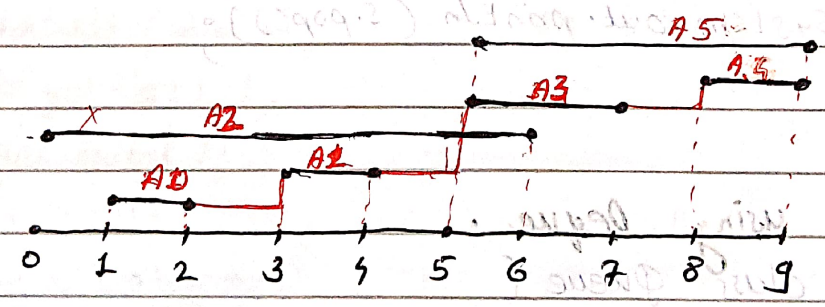
Q. Activity selection: Imp logic

- Same logic use → disjoint set, max meeting in room etc.

- You are given n activities with their start and end times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time. Activities are sorted according to end time.

ex: start[] = [1, 3, 0, 5, 8, 5]
end[] = [2, 4, 6, 7, 9, 9]

A0 A1 A2 A3 A4 A5



ans: A0 A1 A3 A4 total = 4;

approach:

```
lastend = end[0]
if(lastend <= start[i]) {        loop for i=1 to l.
    count++;
```

- first process is end then second process start.
- ~~after end of the fins~~ does not consider the second process is start before end of the first process.

//code//
```
public static int maxActivity(int[] start, int[] end){
    int l = start.length;
    ArrayList<Integer> ans = new ArrayList();
    ans.add(0);
    int maxAct = 1;
    int lastEnd = end[0];
    for(int i=1; i<l; i++){
        if(lastEnd <= start[i]){
            maxAct++;
            lastEnd = end[i];
            ans.add(i);
        }
    }
    for(int i=0; i<ans.size(); i++){
        System.out.print("A"+ans.get(i)+" ");
    }
    System.out.println();
    return maxAct;
}
```

O/P: A0 A1 A3 A4
Max Activity : 4.

Note: for this question end time is sorted madnatory. start time any type sorted or unsorted they will be ok.

- If end time is not sorted then sort the end time.

approch: create 2D array of nx3 size store each column of idx, start, end time.

Note: read the comparator material of java in course elements

for ex: start = [0, 1, 3, 5, 5, 8]
end = [6, 2, 4, 7, 9, 9]

index	start	end
0	0	6
1	1	2
2	3	4
3	5	7
4	5	9
5	8	9

by using in built function sort array by choose column they have sort.

Arrays.sort(ArrayName, Comparator.comparingDouble(o -> o[2]));
o alphabet, hyphen, greater than

- this is lambda function (shortcut of big functions)
- In java Comparator is interface for sorting java objects.

```
//code1// public static int maxActivity2[int start[], int[] end[]){
        int l = end.length;
        ArrayList<Integer> ans = new ArrayList();
        // for sorting //
        int activities[][] = new int[l][3];
        for(int i=0; i<l; i++){
            activities[i][0] = i;
            activities[i][1] = start[i];
            activities[i][2] = end[i];
        }
        // lambda function //
O(nlogn) → Arrays.sort(activities, Comparator.comparingDouble(o -> o[2]));
        ans.add(activities[0][0]);
        int maxAct = 1;
        int lastEnd = activities[0][2];
```

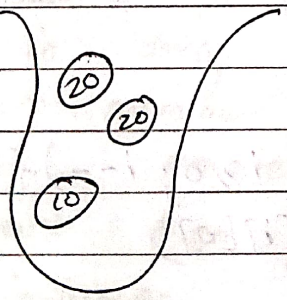
```
for (int i=0; i<l; i++) {
        if (lastEnd <= activities[i][1]) {
            maxAct++;
            lastEnd = activities[i][2];
            ans.add(activities[i][0]);
        }
    }
    for (int i=0; i<ans.size(); i++) {
        System.out.print("A"+ans.get(i)+" ");
    }
    System.out.println();
    return maxAct;
}
```

Q. Fraction Knapsack (bag):

- Given the weights and values of N items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

(Note; 0-1 knapsack problem solve by DP).
(This question solve by greedy method).

→ Ex: cost = [60, 100, 120] capacity (W) = 50 kg.
weight = [10, 20, 30]



capacity = 10 + 20 + 20
cost = 60 + 100 + 4 × 20
cost = 240

max cost = 240.

- A man is greedy, bag capacity is 50 but ~~put~~ like ~~this~~ that max cost get & capacity is complete.

approach : 1) first find ratio of cost & weight.

∴ ratio[i] = cost[i] / wei[i]

& because | cost ↑ & weight ↓ |

2) check always weight[i] ≤ capacity then subtract weight from capacity & add cost in object.

~~3) if otherwise add cost.~~

3) if space is remain but weight is max.
then cost = ratio[i] * capacity

4) break & return.

Note: ratio[][]; because in 0th column stored the index, they are help to trace the possition of weight & cost.

```
// code // public static void maxCost(int wei[], int cost[], int cap) {
        float ratio[][] = new float[cost.length][2];
        for(int i=0; i< ratio.length; i++) {
            ratio[i][0] = i;
            ratio[i][1] = cost[i] / wei[i];
        }
        // sort the ratio //
sort. →  Arrays.sort(ratio, Comparator.comparingDouble(o -> o[1]));
O(nlogn)
        // but you required max ratio and array is //
        // sorted in decre. increasing order, hence start //
        // loop to end //
        float maxCost = 0;
        for(int i= ratio.length-1; i>=0; i--) {
type casting → int idx = (int) ratio[i][0];
            if( wei[idx] ≤ cap ) {
                cap -= wei[idx];
                maxCost += cost[idx];
            }
```

```
            else {
                maxCost += (ratio[i][1] * cap);
                break;
            }
        }
        System.out.println("Max Cost : " maxCost);
    }
```

o/p: Max Cost : 240.0

Q. Min Absolute Differences Pairs :

- Given two Array A & B of equal length n. pair each element of array A to an element in Array B, such that sum S of absolute differences of all the pair is minimum.

ex: A = [1, 2, 3]
B = [2, 1, 3]

case 1: |1-2| + |2-1| + |3-3| = 1+1+0 = 2
case 2: |1-1| + |2-3| + |3-2| = 0+1+1 = 2
case 3: |1-3| + |2-2| + |3-1| = 2+0+2 = 4
case 4: |1-1| + |2-2| + |3-3| = 0 — ans.

approach:

- If any two number maximum closed to each other then their difference is minimum
  ex: 1-3 & 2-10 ⇒ diff. 2 & 8.
- but in array in case one number is closed to two number then logic is wrong.
- for these problem solve by, given two array are sort then find difference simultaneously of indexes.

ex: A = [1, 2, 3]
B = [1, 2, 3]
diff. = 0+0+0 = 0

both min is oneside & maximum in another side there are closed to each other.

//code//
```
public static int minAbsoluteVal(int A[], int B[]){
    int l = A.length;
    Arrays.sort(A);
    Arrays.sort(B);
    int minVal = 0;
    for(int i=0; i<l; i++){
        minVal = Math.abs(A[i] - B[i]);
    }
    return minVal;
}
```

Q. Max length of Chain of pairs

You are given n pairs of numbers. In every pair, the first number is always smaller than the second number. A pair (c,d) can come after pair (a,b) if b<c. Find the longest chain which can be formed from a given set of pairs.

Ex: pairs = (5,24)
(39,60)
(5,28)
(27,40)
(50,90)

5	24
39	60
5	28
27	40
50	90

1) Sort the array (base on 2nd column).
2) one pair can be select.
3) apply loop from (i) & cond" is
second pair (first index) > first pair (last index)

Note: also print the pairs a stored the index in 2D arrayList like Max Activity problem, and, also print given written code.

	column → 0	1
row → 0	5	24
→ 1	5	28
2	27	40
3	39	60
4	50	90

[5, 24] → [27, 40] → [50, 90].
max chain = 3.

// code //

```
public static void maxLengthOfChainPair(int pairs[][]){
    // first sort array //
    Arrays.sort(pairs, Comparator.comparingDouble(o->o[1]));
    int count = 1;                          // because also one pair.
    int prevIdx = pairs[0][1];
    System.out.print("["+pairs[0][0]+","+pairs[0][1]+"]");
    for(int i=1; i<pairs.length; i++){
        int nextIdx = pairs[i][0];
        if(prevIdx < nextIdx){
            System.out.println("["+pairs[i][0]+","...);
            count++;
            prevIdx = pairs[i][1];
        }
    }
    System.out.println();
    System.out.println("Max Pair : "+count);
}
```

Q. Indian coins :

- we are given an infinite supply of denominations [1, 2, 5, 10, 20, 50, 100, 500, 2000]. find min no. of coins/notes to make change for a value V.

Ex: rupee = 590 ans: 500 50 20 20
total min coin : 4.

approach:
coin array is in decreasing order and apply loop to if(coin[i] ≤ rupee).

```
//code// public static void minNumberCoin(int rupee){
        Integer coin[] = {2000, 500, 100, 50, 20, 10, 5, 2, 1};
        ArrayList<Integer> ans = new ArrayList();
        int count = 0;
        for(int i=0; i<coin.length; i++){
            if(rupee>=coin[i]){
                while(rupee>=coin[i]){
                    ans.add(coin[i]);
                    rupee -= coin[i];
                    count++;
                }
            }
        }
        for(int i=0; i<ans.size(); i++){
            System.out.print(ans.get(i)+" ");
        }
        System.out.println();
        System.out.println("total Min coin used : "+count);
    }
```

Q. Job sequencing problem:

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

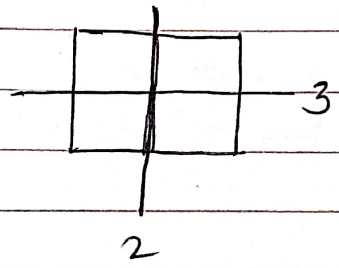
Hard

Q Chocolate problem // Min cost to cut board into square.

- we are given a bar of chocolate composed of m×n square pieces. One should break the chocolate into single squares, each break of a part of the chocolate is charged a cost expressed by a positive integer. This cost does not depend on the size of the part that is being broken but only depends on the line the gre break goes along. Let us denote the costs of breaking along consecutive vertical lines with x_1, x_2 $x_3 \dots x_{m-1}$, and along horizontal lines with $y_1, y_2, y_3, \dots y_{n-1}$.

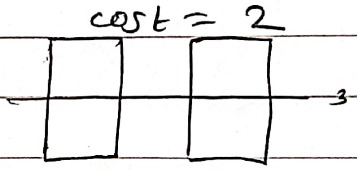
Compute the min cost of breaking the whole chocolate into single squares.

ex:

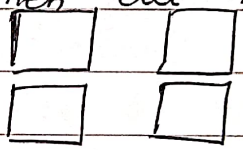


Case 1: 1) cut first vertical

cost = 2



2) then cut horizontal



cost = 2+3+3 = 8.

approach:

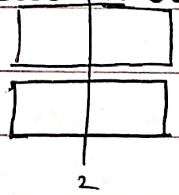
1) use all cuts because cuts in square in all possible

2) first cut expensive cuts. then cut cheep cuts. for this sort array in decreasing order.

3) cost of vertical cut = hp × x.
cost of horizontal cut = vp × y

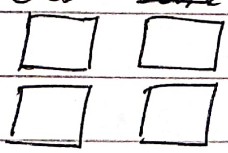
4) then apply two point of V & H cuts array & compare which is greater the cut

Case 2: 1) cut first Horizontal.



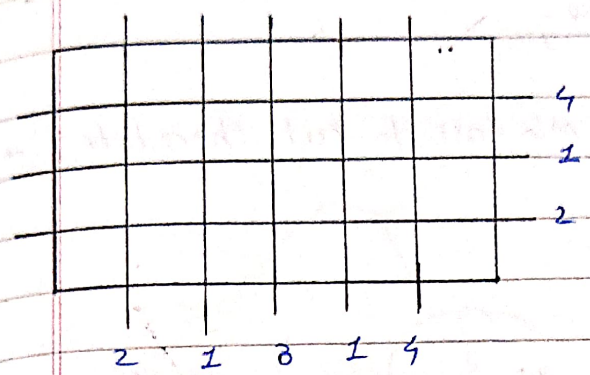
cost = 3

2) cut ~~Horiz~~ vertical.



cost = 3+2+2 = 7.

ex: cost of vertical cuts = [4, 3, 2, 1, 1] } sorted.
cost of Horizontal cuts = [4, 2, 1] .




```
int h=0, v=0;
int hp=1, vp=1;
while(h< hori.length && v< length)
    if ( h<v)
        vertical cut.
    }else {
        Horizontal cut. }
```

//code// public static void minCost(Integer [] VertiPrices, Integer [] HoriPrice) {

```
// first sort prices //
Arrays.sort(vertPrice, Collection.reverseOrder());
Arrays.sort(hariPrice, Collection.reverseOrder() );
int vp=1, hp=1 ; int v=0, h=0, cost=0 ;
while(v< vertPrice.length && h<hariPrice.length ){
    if( vertPrice[v] < hariPrice[h] ) {
        cost += (vp * hariPrice[h]);
        h++; hp++;
    } else {
        cost += (hp * vertPrice [v]);
        v++; vp++;
    }
}
// remaining cuts in vertical //
while (v< vertPrice.length) {
    cost += (hp * vertPrice[v]);
    v++; vp++;
}.
// remaing cuts in horizontal //
```

```
        while ( h < horiPrice.length) {
            cost += (vp * horiPrice[h]);
            h++; hp++;
        }
        System.out.println("Min cost to cut chocolate : " + cost);
    }
```

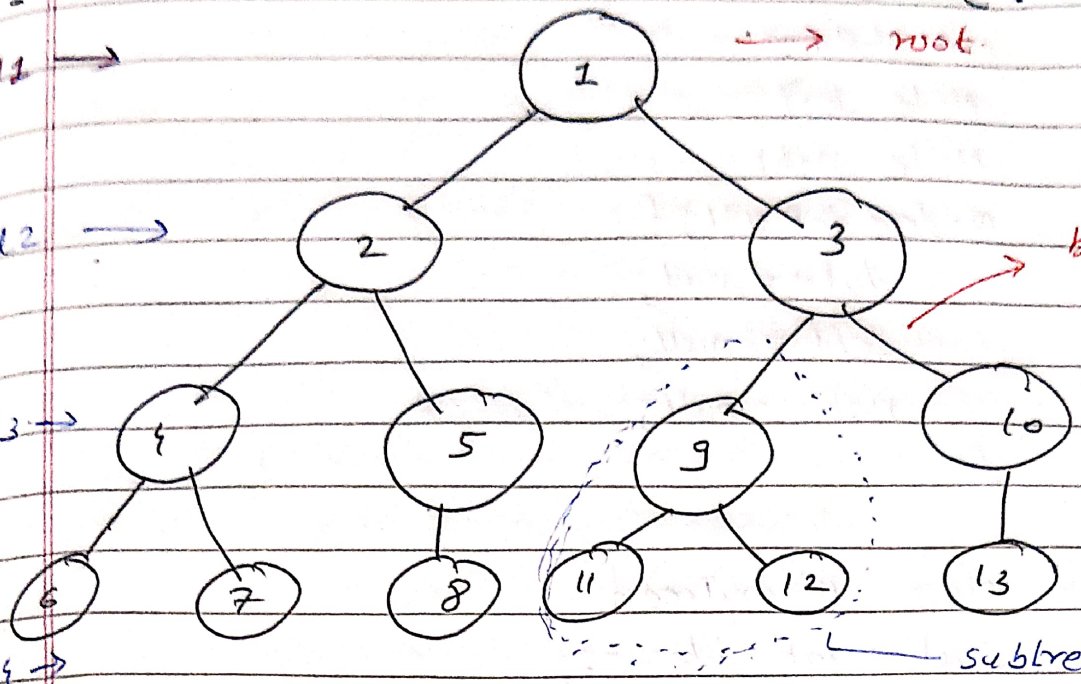
- Summary :

1) for max activity end time are sorted.
2) fraction knapsack - max cost → sort by ratio for cost ↑ & weight ↓.
3) ~~for~~ Min absolute pair → sort two array.
4) max length chain pair → sort by start index.
5) Indian coin → sorted coins.
6)
7) Chocola problem → sort by cost of cuts horizontal & vertical.

Date 25-7-24
Page 99

Binary Trees: (Part-1)

It is Hierarchical data structure.



- depth = 4.

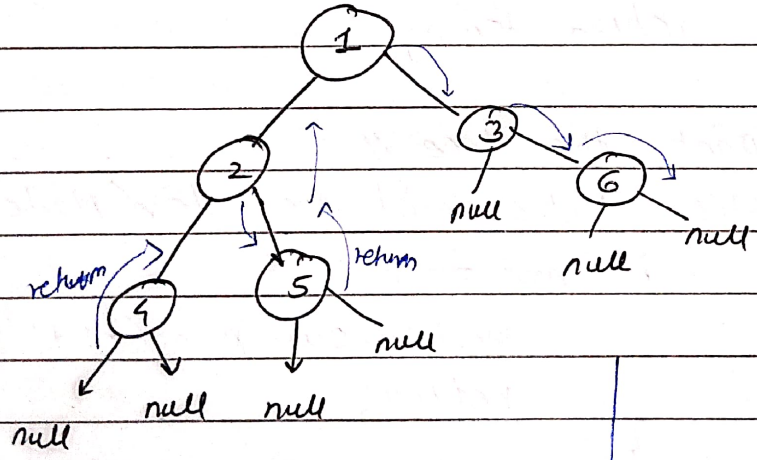
- Ancestors : all parent node.

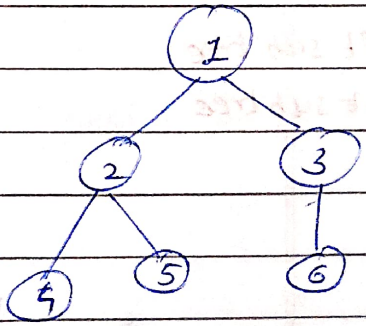
ex: ancestors of 8 : {5, 2, 1}.

* Build Tree Preorder:

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

→





// all code in public class //

// code //

```
Static class Node {
    int data;
    Node left;
    Node right;
    Node(int val) {
        data = val;
        left = null;
        right = null;
    }
}

Static class BinaryTree {
    Static int idx = -1;
    public static Node buildTree(int [] nodes) {
        idx++;
        if(nodes[idx] == -1) {
            return null;
        }
        Node temp = new Node(nodes[idx]);
        temp.left = buildTree(nodes);
        temp.right = buildTree(nodes);
        return temp;
    }
    // print the tree //
    public static void preOrder(Node root) {
        if(root == null) {
            System.out.print(-1 + " ");
            return;
        }
        System.out.print(root.data + " ");
        preOrder(root.left);
        preOrder(root.right);
    }
```

root
left sub tree
right sub tree

```
// print tree in order //
public static void inOrder(Node root) {
        if(root == null){
            return;
        }
        inOrder(root.left);
        System.out.print(root.data+" ");
        inOrder(root.right);
    }
    // post order traversal //
    public static void postOrder(Node root) {
        if(root == null){
            return;
        }
        postOrder(root.left);
        postOrder(root.right);
        System.out.print(root.data+" ");
    }
}
```

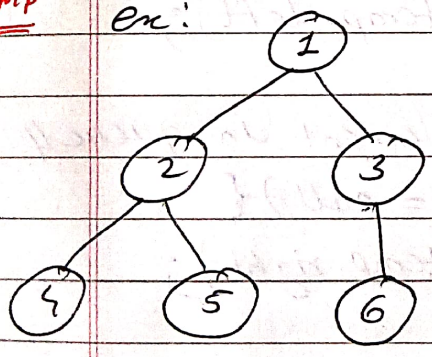
leftsubtree
root
rightsubtree

leftsubtree
rightsubtree
root.

**Q. Level order of Binary tree.

Imp

ex:



approach:

1) use the Stack because they are work LIFO.
2) first add root node & null. | 1 | null | |
3) after each level print nextline.

o/p: 1
2 3
4 5 6

IMP.

// this code in BinaryTree class.// prev page.

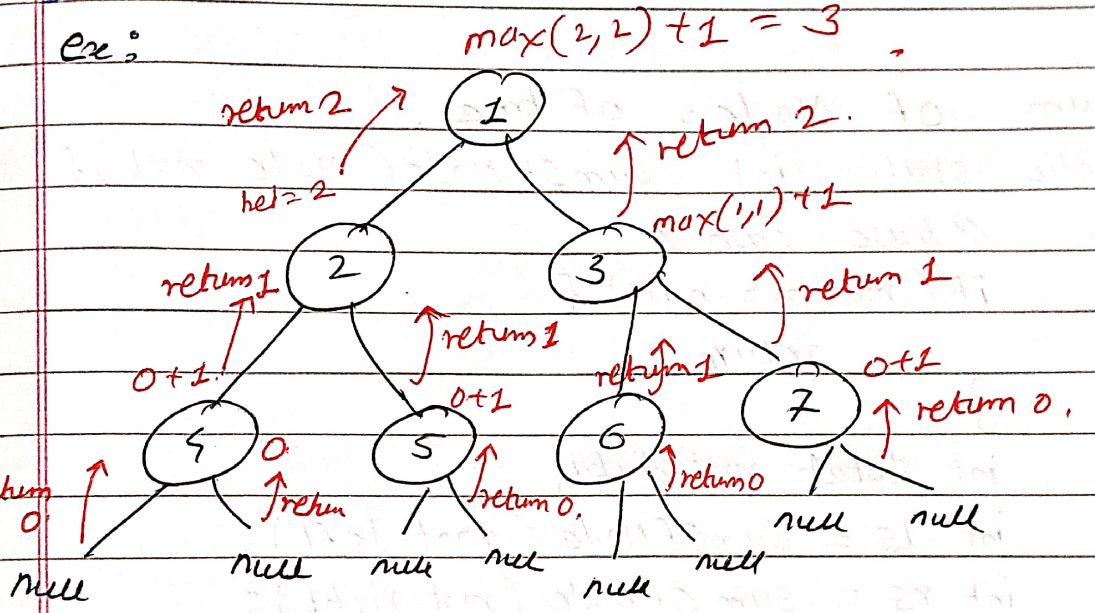
//code//

```
public static void levelOrder(Node root){
    if(root == null){
        return null;
    }
    Queue<Node> q = new LinkedList();
    q.add(root);
    q.add(null);
    while(!q.isEmpty()){
        Node temp = q.remove();
        // temp == null  print nextline //
        if(temp == null){
            System.out.println();
            if(q.isEmpty()){
                break;
            }else{
                q.add(null);
            }
        }else{
            System.out.print(temp.data + " ");
            // add left element in queue //
            if(temp.left != null){
                q.add(temp.left);
            }
            // add right element in queue //
            if(temp.right != null){
                q.add(temp.right);
            }
        }
    }
}
```

Height of a tree :

```
//model1
public static int height(Node root){
        // base case //
        if(root == null){
                return 0;
        }
        // recursion //
        int lh = height(root.length);
        int rh = height(root.right);
        int hei = Math.max(lh, rh) + 1;
        return hei;
}
```

ex:




```
public static void main(String args[]){
        Node root = new Node(1);
        root.left = new Node(2);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right = new Node(3);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        System.out.println("Height of tree :" + height(root));
}
```

Q Count of Nodes of Tree :

// code //

```
public static int countNode(Node root){
    // base case //
    if(root == null){
        return 0;
    }
    // recursion // kaam //
    int lc = countNode(root.left);
    int rc = countNode(root.right);
    int totalCount = lc + rc + 1;
    return totalCount;
}
```

Q. Sum of Nodes of tree.

// code //

```
public static int SumOfNode(Node root){
    // base case //
    if(root == null){
        return 0;
    }
    int sum = root.data;
    int ls = SumOfNode(root.left);
    int rs = SumOfNode(root.right);
    int total = ls + rs + sum;
    return total;
}
```