Coding practice Problems:

1. Maximum Subarray Sum – Kadane‟s Algorithm: Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.

Input: arr[] = {2, 3, -8, 7, -1, 2, 3} Output: 11

Explanation: The subarray {7, -1, 2, 3} has the largest sum 11. Input: arr[] = {-2, -4} Output: –2 Explanation: The subarray {-2} has the largest sum -2.

Input: arr[] = {5, 4, 1, 7, 8} Output: 25 Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

**CODE:**

import java.util.\*;

public class maxsubarray {

public static void main(String[] args) {

maxsubsum(new int[]{2, 3, -8, 7, -1, 2, 3});

maxsubsum(new int[]{5, 4, 1, 7, 8});

}

public static void maxsubsum(int[] arr){

System.out.print(Arrays.toString(arr) + " :");

int ans = 0;

int max = arr[0];

for(int i = 1; i<arr.length ; i++){

max = Math.max(max+arr[i],arr[i]);

ans = Math.max(ans,max);

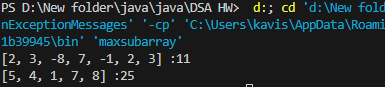
}

System.out.println(ans);

}

}

**OUTPUT :**

****

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

1. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray.

Input: arr[] = {-2, 6, -3, -10, 0, 2} Output: 180

Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 \* (-3) \* (-10) = 180

Input: arr[] = {-1, -3, -10, 0, 60} Output: 60 Explanation: The subarray with maximum product is {60}.

**CODE :**

import java.util.Arrays;

public class maxprosubarray {

    public static void main(String[] args) {

        maxproduct(new int[]{-2, 6, -3, -10, 0, 2});

        maxproduct(new int[]{-1, -3, -10, 0, 60});

    }

    public static void maxproduct(int[] arr){

        System.out.print(Arrays.toString(arr) + " :");

        int ans = 0;

        int max = arr[0];

        int min = arr[0];

        for(int i = 1;  i<arr.length ; i++){

            if(arr[i]<0){

                int temp = max;

                max = min;

                min = temp;

            }

            max = Math.max(max\*arr[i],arr[i]);

            min = Math.min(arr[i],min\*arr[i]);

            ans = Math.max(ans,max);

        }

        System.out.println(ans);

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

1. Search in a sorted and rotated Array Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1. Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0 Output : 4

Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3 Output : -1

Input : arr[] = {50, 10, 20, 30, 40}, key = 10 Output : 1

**CODE :**

import java.util.\*;

public class searchrotatedarray {

    public static void main(String[] args) {

        search(new int[]{4, 5, 6, 7, 0, 1, 2} , 0);

        search(new int[] { 4, 5, 6, 7, 0, 1, 2 } , 3);

        search(new int[]{50, 10, 20, 30, 40} , 10);

    }

    public static void search(int[] arr , int a){

        System.out.print(Arrays.toString(arr) + " : ");

        boolean flag = false;

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

            if(arr[i] == a){

                System.out.println(i);

                flag = true;

                break;

            }

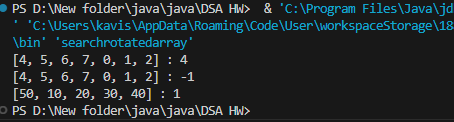
        }

        if(!flag) System.out.println(-1);

    }

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

1. Container with Most Water

Input: arr = [1, 5, 4, 3] Output: 6 Explanation: 5 and 3 are distance 2 apart. So the size of the base = 2. Height of container = min(5, 3) = 3. So total area = 3 \* 2 = 6

Input: arr = [3, 1, 2, 4, 5] Output: 12 Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4. Height of container = min(5, 3) = 3. So total area = 4 \* 3 = 12

**CODE :**

import java.util.\*;

public class mostwater {

    public static void main(String[] args) {

        area(new int[]{1, 5, 4, 3});

        area(new int[]{3, 1, 2, 4, 5});

    }

    public static void area(int[] arr){

        System.out.print(Arrays.toString(arr)+ " : ");

        int i = 0 ;

        int j = 1 ;

        int ans = Integer.MIN\_VALUE;

        while(i<arr.length){

            ans = Math.max((j-i)\*Math.min(arr[i],arr[j]) , ans);

            if(j!=arr.length-1) j++;

            else i++;

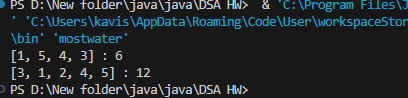
        }

        System.out.println(ans);

    }

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

1. Find the Factorial of a large number

Input: 100 Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000 00

Input: 50 Output: 30414093201713378043612608166064768844377641568960512000000000000

**CODE :**

import java.math.BigInteger;

public class factoriallarge {

    public static void main(String[] args) {

        System.out.println("factorial of 100  : " + fact(100));

        System.out.println("factorial of 50  : " + fact(50));

    }

    public static BigInteger fact(int n){

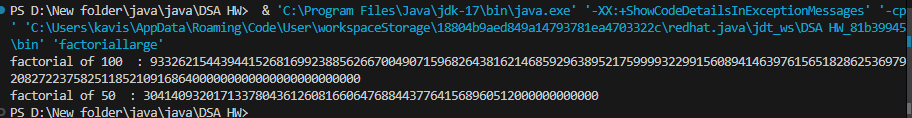
        if(n == 1) return BigInteger.ONE;

        return BigInteger.valueOf(n).multiply(fact(n-1));

    }

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

1. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.

Input: arr[] = {3, 0, 1, 0, 4, 0, 2} Output: 10 Explanation: The expected rainwater to be trapped is shown in the above image.

Input: arr[] = {3, 0, 2, 0, 4} Output: 7 Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units.

Input: arr[] = {1, 2, 3, 4} Output: 0 Explanation : We cannot trap water as there is no height bound on both sides

Input: arr[] = {10, 9, 0, 5} Output: 5 Explanation : We trap 0 + 0 + 5 + 0 = 5

**CODE :**

import java.util.Arrays;

public class traprainwater {

    public static void main(String[] args) {

        trap(new int[]{3, 0, 1, 0, 4, 0, 2});

        trap(new int[]{3, 0, 2, 0, 4});

        trap(new int[]{1, 2, 3, 4});

    }

    public static void trap(int[] arr){

        System.out.print(Arrays.toString(arr) + " : ");

        int i = 0 , j = arr.length-1 ;

        int left = arr[i] , right = arr[j];

        int ans = 0;

        while(i<j){

            if(left<right){

                i++;

                left = Math.max(left,arr[i]);

                ans += left - arr[i];

            }else{

                j--;

                right = Math.max(right,arr[j]);

                ans += right - arr[j];

            }

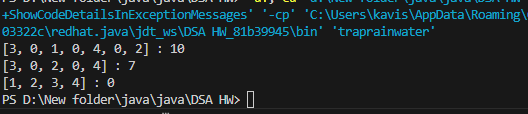
        }

        System.out.println(ans);

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

1. Chocolate Distribution Problem Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized.

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3 Output: 2 Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5 Output: 7 Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is 9 – 2 = 7.

**CODE :**

import java.util.\*;

public class chocolate {

    public static void main(String[] args) {

        choco(new int[]{7, 3, 2, 4, 9, 12, 56}, 3);

        choco(new int[]{7, 3, 2, 4, 9, 12, 56},5);

    }

    public static void choco(int[] arr , int m){

        System.out.print(Arrays.toString(arr) + " : ");

        int ans = Integer.MAX\_VALUE;

        for(int i = 0 ; i<arr.length - m +1; i++){

            int max = arr[i];

            int min = arr[i];

            for(int j = i ; j<i+m ; j++){

                max = Math.max(max , arr[j]);

                min = Math.min(min , arr[j]);

            }

            ans = Math.min(ans ,max-min);

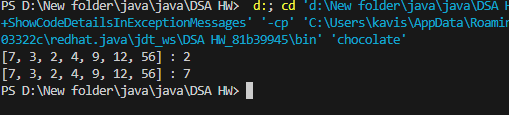
        }

        System.out.println(ans);

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(n\*m)

SPACE COMPLEXITY : O(1)

1. Merge Overlapping Intervals Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

Input: arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]] Output: [[1, 4], [6, 8], [9, 10]] Explanation: In the given intervals, we have only two overlapping intervals [1, 3] and [2, 4]. Therefore, we will merge these two and return [[1, 4}], [6, 8], [9, 10]].

Input: arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]] Output: [[1, 6], [7, 8]] Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval [1, 6].

**CODE :**

import java.util.\*;

public class mergeintervals {

    public static void main(String[] args) {

        merge(new int[][]{{1,3},{2,4},{6,8},{9,10}});

        merge(new int[][]{{7,8},{1,5},{2,4},{4,6}});

    }

    public static void merge(int[][] interval){

        System.out.print(Arrays.deepToString(interval) + "---->");

        Arrays.sort(interval , (a,b)->(a[0]-b[0]));

        List<List<Integer>> l = new ArrayList<>();

        int s = interval[0][0] , e = interval[0][1];

        for(int[] i : interval){

            if(e<i[0]){

                l.add(Arrays.asList(s, e));

                s = i[0];

                e = i[1];

            }else{

                s = Math.min(i[0],s);

                e = Math.max(i[1],e);

            }

            if(i==interval[interval.length-1]){

                l.add(Arrays.asList(s, e));

            }

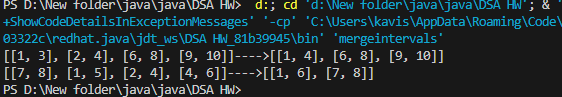
        }

        System.out.println(l);

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(n log n)

SPACE COMPLEXITY : O(n)

1. A Boolean Matrix Question Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as

**CODE :**

import java.util.\*;

public class booleanmatrix {

    public static void main(String[] args) {

        matrix(new int[][]{{1, 0},{0, 0}});

        matrix(new int[][]{{0, 0, 0},{0, 0, 1}});

        matrix(new int[][]{{1, 0, 0, 1},{0, 0, 1, 0},{0, 0, 0, 0}});

    }

    public static void matrix(int[][] arr){

        System.out.print(Arrays.deepToString(arr) + "---->");

        int n = arr.length , m = arr[0].length;

        Set<Integer> si = new HashSet<>();

        Set<Integer> sj = new HashSet<>();

        for(int i = 0 ; i<n ; i++){

            for(int j = 0 ; j<m ; j++){

                if(arr[i][j] == 1){

                    si.add(i);

                    sj.add(j);

                }

            }

        }

        for(int i = 0 ; i<n ; i++){

            for(int j = 0 ; j<m ; j++){

                if(si.contains(i)){

                    arr[i][j] = 1;

                }else if(sj.contains(j)){

                    arr[i][j] = 1;

                }else{

                    arr[i][j] = 0;

                }

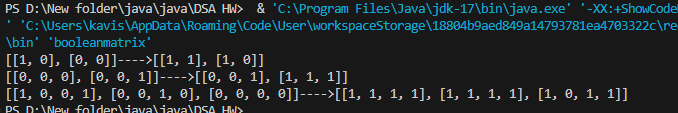
            }

        }

        System.out.println(Arrays.deepToString(arr));

    }

}

**OUTPUT : **

TIME COMPLEXITY : O(n\*m)

SPACE COMPLEXITY : O(n+m)

1. Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form.

Input: matrix = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}, {13, 14, 15, 16 }} Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Input: matrix = { {1, 2, 3, 4, 5, 6}, {7, 8, 9, 10, 11, 12}, {13, 14, 15, 16, 17, 18}} Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11 Explanation: The output is matrix in spiral format.

**CODE :**

import java.util.\*;

public class spiralmatrix {

    public static void main(String[] args) {

        spiral(new int[][]{{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}});

        spiral(new int[][]{{1,2,3,4,5,6},{7,8,9,10,11,12},{13,14,15,16,17,18}});

    }

    public static void spiral(int[][] arr){

        int top= 0;

        int bottom = arr.length-1;

        int left = 0;

        int right = arr[0].length-1;

        int size = arr.length\*arr[0].length;

        List<Integer> l = new ArrayList<>();

        int i= 0;

        while(i!= size){

            for(int n = left; n <= right && (i!= size); n++){

                l.add(arr[top][n]);

                i++;

            }

            for(int n = top; n <bottom && (i!= size); n++){

                l.add(arr[n+1][right]);

                i++;

            }

            for(int n = right-1; n>= left && (i!= size); n--){

                l.add(arr[bottom][n]);

                i++;

            }

            for(int n = bottom-1; n >= top+1 && (i!= size); n--){

                l.add(arr[n][left]);

                i++;

            }

            top++;

            bottom--;

            left++;

            right--;

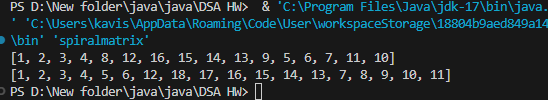
        }

        System.out.println(l);

    }

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

13. Check if given Parentheses expression is balanced or not Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not.

Input: str = “((()))()()” Output: Balanced

Input: str = “())((())” Output: Not Balanced

**CODE :**

public class paranthesis {

    public static void main(String[] args) {

        paran("((()))()()");

        paran("())((())");

        paran("((())()()))()");

    }

    public static void paran(String s){

        System.out.print(s + " : ");

        int c =0 ;

        boolean flag = false;

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

            if(s.charAt(i) == '('){

                c++;

            }else{

                c--;

            }

            if(c<0){

                flag = true;

            }

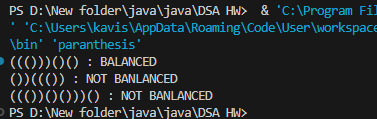
        }

        System.out.println(flag?"NOT BANLANCED" : "BALANCED");

    }

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

14. Check if two Strings are Anagrams of each other Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

Input: s1 = “geeks” s2 = “kseeg” Output: true Explanation: Both the string have same characters with same frequency. So, they are anagrams.

Input: s1 = “allergy” s2 = “allergic” Output: false Explanation: Characters in both the strings are not same. s1 has extra character „y‟ and s2 has extra characters „i‟ and „c‟, so they are not anagrams. Input: s1 = “g”, s2 = “g” Output: true Explanation: Characters in both the strings are same, so they are anagrams.

**CODE :**

import java.util.\*;

public class anagaram {

    public static void main(String[] args) {

        anagra("geeks","kseeg");

        anagra("allergy","allergic");

        anagra("g","g");

    }

    public static void anagra(String s1 ,String s2){

        Map<Character,Integer> hm = new HashMap<>();

        for(char c: s1.toCharArray()){

            hm.put(c,hm.getOrDefault(c,0)+1);

        }

        Map<Character,Integer> hm1 = new HashMap<>();

        for(char c:s2.toCharArray()){

            hm1.put(c,hm1.getOrDefault(c,0)+1);

        }

        System.out.println(hm.equals(hm1));

    }

}

**OUTPUT :**



TIME COMPLEXITY : O(n+m)

SPACE COMPLEXITY : O(n+m)

15. Longest Palindromic Substring Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

Input: str = “forgeeksskeegfor” Output: “geeksskeeg” Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.

**CODE :**

public class palinsubstring {

    public static void main(String[] args) {

        palin("forgeeksskeegfor");

        palin("Geeks");

        palin("abc");

    }

    public static void palin(String s){

        System.out.print(s + " : ");

        int start = 0;

        int end = 0;

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

            int o = find(s, i, i);

            int e = find(s, i, i + 1);

            int max = Math.max(o, e);

            if (max > end - start) {

                start = i - (max - 1) / 2;

                end = i + max / 2;

            }

        }

        String ans = s.substring(start, end + 1);

        System.out.println(ans.length() > 1 ? ans : s.charAt(0));

    }

    public static int find(String s , int l , int r){

        while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {

            l--;

            r++;

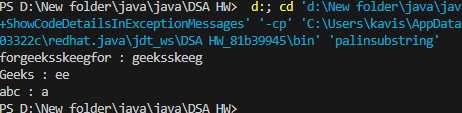
        }

        return r - l - 1;

    }

}

**OUTPUT :**



TIME COMPLEXITY : O(n^2)

SPACE COMPLEXITY : O(1)

16. Longest Common Prefix using Sorting Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”.

Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”] Output: gee Explanation: “gee” is the longest common prefix in all the given strings.

Input: arr[] = [“hello”, “world”] Output: -1 Explanation: There‟s no common prefix in the given strings.

**CODE :**

import java.util.\*;

public class longestprefix {

    public static void main(String[] args) {

        prefix(new String[]{"geeksforgeeks", "geeks", "geek", "geezer"});

        prefix(new String[]{"hello", "world"});

    }

    public static void prefix(String[] arr){

        System.out.print(Arrays.toString(arr) + " : ");

        int l = arr[0].length();

        for(int i = 1 ; i<arr.length ; i++){

            l = Math.min(l , arr[i].length());

            for(int j = 0 ; j<l; j++){

                char c= arr[0].charAt(j);

                char d = arr[i].charAt(j);

                if(c!=d){

                    l = j;

                    break;

                }

            }

        }

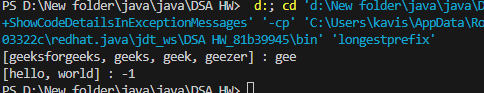
        String ans = arr[0].substring(0,l);

        System.out.println(ans.length() > 0 ? ans : -1 );

    }

}

**OUTPUT :**



TIME COMPLEXITY : O(n\*m)

SPACE COMPLEXITY : O(1)

17. Delete middle element of a stack Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

Input : Stack[] = [1, 2, 3, 4, 5] Output : Stack[] = [1, 2, 4, 5]

Input : Stack[] = [1, 2, 3, 4, 5, 6] Output : Stack[] = [1, 2, 4, 5, 6]

**CODE :**

import java.util.\*;

public class stacks {

    public static void main(String[] args) {

        remove(stack(new int[]{1, 2, 3, 4, 5}));

        remove(stack(new int[]{1, 2, 3, 4, 5,6}));

    }

    public static void remove(Stack s){

        int m = s.size()/2;

        Stack ss = new Stack<>();

        int i = 0;

        while(i<m){

            ss.push(s.pop());

            i++;

        }

        s.pop();

        while(!ss.isEmpty()){

            s.push(ss.pop());

        }

        System.out.println(s );

    }

    public static Stack stack(int[] arr){

        Stack s = new Stack<>();

        for(int i : arr){

            s.push(i);

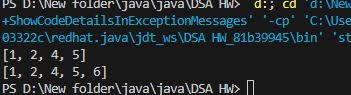
        }

        return s;

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

18. Next Greater Element (NGE) for every element in given Array Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.

Input: arr[] = [ 4 , 5 , 2 , 25 ] Output: 4 5 2 –> 5 –> 25 –> 25 25 –> -1 Explanation: Except 25 every element has an element greater than them present on the right side

Input: arr[] = [ 13 , 7, 6 , 12 ] Output: 13 –> 7 -1 –> 12 6 12 –> 12 –> -1 Explanation: 13 and 12 don‟t have any element greater than them present on the right side

**CODE :**

import java.util.\*;

public class nextgreater {

    public static void main(String[] args) {

        greater(new int[]{4 , 5 , 2 , 25});

        greater(new int[]{13 , 7, 6 , 12});

        greater(new int[]{14,10,7,11,13,12,5});

    }

    public static void greater(int[] arr){

        System.out.print(Arrays.toString(arr) + " : ");

        int[] dp = new int[arr.length];

        int i = arr.length-2;

        int j = arr.length-1;

        dp[dp.length-1] = arr[arr.length-1];

        for(int k = dp.length-2 ; k>= 0 ; k-- ){

            dp[k] = arr[j]>arr[i] ? arr[j] : dp[k+1] > arr[i] ? dp[k+1] : -1;

            // System.out.println(arr[j] + " " + arr[i]+ " " + dp[k] + " " + dp[k+1]);

            i--;

            j--;

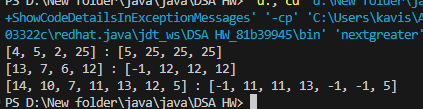
        }

        System.out.println(Arrays.toString(dp));

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

19. Print Right View of a Binary Tree Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.

**CODE :**

import java.util.\*;

class TreeNode{

    int data;

    TreeNode left ;

    TreeNode right ;

    TreeNode(int data){

        this.data = data;

        left = null;

        right = null;

    }

}

public class binaryrightview {

    public static void main(String[] args) {

        TreeNode root = new TreeNode(1);

        root.left = new TreeNode(2);

        root.right = new TreeNode(3);

        root.right.left = new TreeNode(4);

        root.right.right = new TreeNode(5);

        view(root);

        TreeNode root1 = new TreeNode(1);

        root1.left = new TreeNode(2);

        root1.right = new TreeNode(3);

        root1.left.left = new TreeNode(4);

        root1.left.left.right = new TreeNode(5);

        view(root1);

    }

    public static void view(TreeNode root){

        Queue<TreeNode> q = new LinkedList<>();

        q.add(root);

        List<Integer> l = new ArrayList<>();

        while(!q.isEmpty()){

            TreeNode curr = q.poll();

            if(curr.left != null){

                q.add(curr.left);

            }

            if(curr.right != null){

                if(!l.contains(curr.data)) l.add(curr.data);

                l.add(curr.right.data);

                q.add(curr.right);

            }

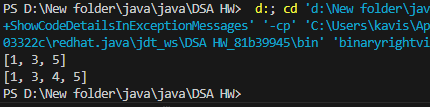
        }

        System.out.println(l);

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

20. Maximum Depth or Height of Binary Tree Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.  
**CODE :**

import java.util.LinkedList;

import java.util.Queue;

class TreeNode{

    int data;

    TreeNode left ;

    TreeNode right ;

    TreeNode(int data){

        this.data = data;

        left = null;

        right = null;

    }

}

public class depthbinarytree {

    public static void main(String[] args) {

        TreeNode root = new TreeNode(12);

        root.right = new TreeNode(18);

        root.left = new TreeNode(8);

        root.left.left = new TreeNode(5);

        root.left.right = new TreeNode(11);

        depth(root);

        TreeNode r = new TreeNode(1);

        r.left = new TreeNode(2);

        r.left.left = new TreeNode(4);

        r.right = new TreeNode(3);

        r.right.right = new TreeNode(5);

        r.right.right.right  = new TreeNode(7);

        r.right.right.left = new TreeNode(6);

        depth(r);

    }

    public static void depth(TreeNode root){

        Queue<TreeNode> q = new LinkedList<>();

        q.add(root);

        int d = 0;

        while(!q.isEmpty()){

            int n = q.size();

            d++;

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

                TreeNode curr = q.poll();

                if(curr.left != null){

                    q.add(curr.left);

                }

                if(curr.right != null){

                    q.add(curr.right);

                }

            }

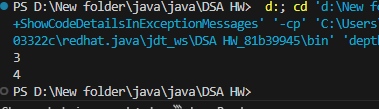
        }

        System.out.println(d);

    }

}

**OUTPUT :**

****

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)