Name: Pradeep S

Dept: CSBS

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.

```
import java.util.*;


public class Main{
    public static void main(String args[]){
            Scanner sc = new Scanner(System.in);
            int n = sc.nextInt();
            int[] arr = new int[n];
            for (int idx = 0; idx < arr.length; idx++) {
                    arr[idx] = sc.nextInt();


            }
            int b = 0;
            int a =Integer.MIN_VALUE;
            for(int num:arr){
```

```
                b+=num;
                a = Math.max(b,a);
                if(b<0){
                        b = 0;
                }
            }
            System.out.println(a);


    }
}
```

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

2. 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}.

```
import java.util.*;

public class Maxprod{
   public static void main(String args[]){
```

```
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int idx = 0; idx < arr.length; idx++) {
           arr[idx] = sc.nextInt();

        }
        int max = arr[0], min = arr[0], ans = arr[0];

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

           int temp = max;

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

           ans = Math.max(ans,max);
        }
        System.out.println(ans);



    }
}
```

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

3.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

```
import java.util.*;

public class Example3 {
   public static int BinarySearch(int[] arr, int tar, int s, int e) {

         if (s > e) {
      return -1;
    }

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

    if (arr[m] == tar) {
      return m;
    }


    if (arr[s] <= arr[m]) {
      if (tar >= arr[s] && tar < arr[m]) {
        return BinarySearch(arr, tar, s, m - 1);
      } else {
        return BinarySearch(arr, tar, m + 1, e);
      }
```

```
    }


    else {
      if (tar > arr[m] && tar <= arr[e]) {
        return BinarySearch(arr, tar, m + 1, e);
      } else {
        return BinarySearch(arr, tar, s, m - 1);
      }
    }
  }

  public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the value for N:");
        int n = sc.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
      arr[i] = sc.nextInt();
    }
        System.out.println("Enter the element to search");
    int tar = sc.nextInt();
    int s = 0, e = n - 1;
    int result = BinarySearch(arr, tar, s, e);
        System.out.println(result);


  }
}
```

TIME COMPLEXITY : O(n)
SPACE COMPLEXITY : O(1)

4. 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

```
import java.util.*;
public class Water{
   public static void main(String ar[]){
      Scanner sc = new Scanner(System.in);
      int n = sc.nextInt();
      int[] arr = new int[n];

      for(int i=0;i<n;i++){
         arr[i] = sc.nextInt();
      }

      int i = 0,j = n-1;
      int area = 0;

      while(i<j){
         int m = Math.min(arr[i],arr[j]);
         int h = j-i;
         area = Math.max(ans,m*h);
         if(i<j) i++;
         else j--;
      }

      System.out.println(ans);




   }
}
```

TIME COMPLEXITY : O(n)
SPACE COMPLEXITY : O(1)

5. Find the Factorial of a large number
Input: 100
Output:
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000

Input: 50
Output:
30414093201713378043612608166064768844377641568960512000000000000

```
import java.util.*;

class Main {
  static int mul(int x, int arr[], int size) {
    int carry = 0;
    for (int i = 0; i < size; i++) {
      int prod = arr[i] * x + carry;
      arr[i] = prod % 10;
      carry = prod / 10;
    }
    while (carry != 0) {
      arr[size] = carry % 10;
      carry = carry / 10;
      size++;
    }
    return size;
  }

  static void factorial(int n) {
    int[] arr = new int[400];
    arr[0] = 1;
    int size = 1;
    for (int i = 2; i <= n; i++) {
      size = mul(i, arr, size);
    }
    System.out.println("factorial:");
    for (int i = size - 1; i >= 0; i--) {
      System.out.print(arr[i]);
    }
```

```
    }

    public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int n = sc.nextInt();
      factorial(n);
    }
}
```

TIME COMPLEXITY : O(n^2 log n)
SPACE COMPLEXITY : O(n log n)

6. 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

```
    import java.util.*;
public class TrapingWater{
  public static void main(String args[]){
    Scanner sc = new Scanner(System.in);
```

```
        int n = sc.nextInt();
        int[] arr = new int[n];
        for(int i=0;i<n;i++){
            arr[i] = sc.nextInt();
        }
        int l = 0,r = 0;
        int[] max_left = new int[n];
        int[] max_right = new int[n];
        for(int i=0;i<n;i++){
            int j = n-i-1;
            max_left[i] = l;
            max_right[j] = r;
            l = Math.max(l,arr[i]);
            r = Math.max(r,arr[j]);
        }
        int sum = 0;
        for(int i=0;i<n;i++){
            int pot = Math.min(max_left[i],max_right[i]);
            sum+=Math.max(0,pot-arr[i]);
        }
        System.out.println(sum);
    }
}
```

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

7. 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

```
import java.util.*;
public class ChocolateDistribution{
        public static void main(String args[]){


            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the Number of Elements");
            int n = sc.nextInt();
            System.out.println("Enter the value for m");
            int m = sc.nextInt();
            int[] arr = new int[n];
            System.out.println("Enter the values for the array");
```

```
            for(int i=0;i<n;i++){
                arr[i] = sc.nextInt();
            }
            Arrays.sort(arr);
            int ans = Integer.MAX_VALUE;
            for(int i=0;i<n-m;i++){
                int diff = arr[i+m-1]-arr[i];
                ans = Math.min(ans,diff);
            }
            System.out.println(ans);
        }
}
```

TIME COMPLEXITY : O(n log n)

SPACE COMPLEXITY : O(1)

8. 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].

```
import java.util.*;

public class Merge {
   public static int[][] merge(int[][] intervals) {
      Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));
      List<int[]> arr = new ArrayList<>();
      int[] prev = intervals[0];

      for (int i = 1; i < intervals.length; i++) {
         int[] v = intervals[i];
         if (prev[1] >= v[0]) {
            prev[1] = Math.max(prev[1], v[1]);
         } else {
            arr.add(prev);
            prev = intervals[i];
         }
      }
      arr.add(prev);

      return arr.toArray(new int[arr.size()][]);
   }

   public static void main(String args[]) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the number of intervals:");
      int n = sc.nextInt();
      int[][] arr = new int[n][2];
```

```
        System.out.println("Enter the intervals (start and end):");
        for (int i = 0; i < n; i++) {
            arr[i][0] = sc.nextInt();
            arr[i][1] = sc.nextInt();
        }

        int[][] result = merge(arr);

        System.out.println("Merged intervals:");
        for (int[] interval : result) {
            System.out.println(Arrays.toString(interval));
        }
    }
}
```

TIME COMPLEXITY : O(n log n)

SPACE COMPLEXITY : O(n)

9. 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 1.

Input: {{1, 0}, {0, 0}}

Output: {{1, 1} {1, 0}}

Input: {{0, 0, 0}, {0, 0, 1}}

Output: {{0, 0, 1}, {1, 1, 1}}

Input: {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}}

Output: {{1, 1, 1, 1}, {1, 1, 1, 1}, {1, 0, 1, 1}}

```
import java.util.*;

class BooleanMatrix{

   static void Matrix(int[][] matrix) {
      int rows = matrix.length;
      int cols = matrix[0].length;

      for (int i = 0; i < rows; i++) {
         for (int j = 0; j < cols; j++) {
            if (matrix[i][j] == 1) {
               int ind = i - 1;
               while (ind >= 0) {
                  if (matrix[ind][j] != 1) {
                     matrix[ind][j] = -1;
                  }
                  ind--;
               }
               ind = i + 1;
               while (ind < rows) {
                  if (matrix[ind][j] != 1) {
                     matrix[ind][j] = -1;
                  }
                  ind++;
               }

               ind = j - 1;
```

```
            while (ind >= 0) {
                if (matrix[i][ind] != 1) {
                    matrix[i][ind] = -1;
                }
                ind--;
            }
            ind = j + 1;
            while (ind < cols) {
                if (matrix[i][ind] != 1) {
                    matrix[i][ind] = -1;
                }
                ind++;
            }
        }
    }
  }

  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      if (matrix[i][j] < 0) {
        matrix[i][j] = 1;
      }
    }
  }
}

public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
```

```
        int rows = sc.nextInt();

        int cols = sc.nextInt();

        int[][] arr = new int[rows][cols];



        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                arr[i][j] = sc.nextInt();
            }
        }

        Matrix(arr);

        System.out.println("The Final Matrix is:");
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[0].length; j++) {
                System.out.print(arr[i][j] + " ");
            }
            System.out.println();
        }

        sc.close();
    }
}
```

TIME COMPLEXITY : O(n*m)

SPACE COMPLEXITY : O(n+m)

10. 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.

```
import java.util.*;


class SpiralMatrix{
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);


      int rows = sc.nextInt();


      int cols = sc.nextInt();


      int[][] matrix = new int[rows][cols];



      for (int i = 0; i < rows; i++) {
         for (int j = 0; j < cols; j++) {
            matrix[i][j] = sc.nextInt();
         }
      }
```

```
        int x = 0, y = 0, dx = 1, dy = 0;
        List<Integer> res = new ArrayList<>();

        for (int i = 0; i < rows * cols; i++) {
            res.add(matrix[y][x]);
            matrix[y][x] = -101;

            if (!(0 <= x + dx && x + dx < cols && 0 <= y + dy && y + dy < rows) || matrix[y + dy][x + dx] == -101) {
                int temp = dx;
                dx = -dy;
                dy = temp;
            }

            x += dx;
            y += dy;
        }


        for (int num : res) {
            System.out.print(num + " ");
        }

        sc.close();
    }
}
```

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

```
import java.util.*;
public class ValidParentheses{

        public static Boolean isvalid(String s){

        Stack<Character>a = new Stack<>();
        for(int i=0;i<s.length();i++){
            if(s.charAt(i)=='('){
                    a.push(s.charAt(i));                                    // “(((()))()()”
“())((())”

            }
            else{
                    if(a.isEmpty()){
                            return false;
                    }
                    else{
                            if(a.peek()=='('){
                                    a.pop();
                            }
```

```
                    else{
                        return false;
                    }

                }
            }
        }

        return a.isEmpty();

        }

        public static void main(String args[]){
            Scanner sc = new Scanner(System.in);
            String s = sc.nextLine();
            Boolean ans = isvalid(s);
            if(ans){
                System.out.println("Balanced");
            }
            else{
                System.out.println("Not Balanced");
            }
        }
}
```

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.

```
import java.util.*;


class Anagrams{
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);



    String a = sc.nextLine();
          String b = sc.nextLine();


    System.out.println(ValidAnagrams(a,b));
  }
```

```
  public static boolean ValidAnagrams(String a, String b) {
    if (a.length() != b.length()) {
      return false;
    }

    char[] arr1 = a.toCharArray();
    char[] arr2 = b.toCharArray();

    Arrays.sort(arr1);
    Arrays.sort(arr2);

    Boolean ans = Arrays.equals(arr1, arr2);
            if(ans) return true;
            else return false;
  }
}
```

TIME COMPLEXITY : O(n log n)

SPACE COMPLEXITY : O(n)

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.

Input: str = “Geeks”

Output: “ee”

Input: str = “abc”

Output: “a” Input: str = “” Output: “”

```
import java.util.Scanner;


public class LongestPalindrome{
   public static String Palindrome(String s) {
      if (s.length() <= 1) {
         return s;
      }


      int maxLen = 1;
      String maxStr = s.substring(0, 1);
      s = "#" + s.replaceAll("", "#") + "#";
      int[] dp = new int[s.length()];
      int center = 0;
      int right = 0;


      for (int i = 0; i < s.length(); i++) {
         if (i < right) {
            dp[i] = Math.min(right - i, dp[2 * center - i]);
         }


         while (i - dp[i] - 1 >= 0 && i + dp[i] + 1 < s.length() && s.charAt(i - dp[i] - 1) ==
s.charAt(i + dp[i] + 1)) {
            dp[i]++;
         }
```

```
        if (i + dp[i] > right) {
            center = i;
            right = i + dp[i];
        }

        if (dp[i] > maxLen) {
            maxLen = dp[i];
            maxStr = s.substring(i - dp[i], i + dp[i] + 1).replaceAll("#", "");
        }
      }

      return maxStr;
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      String str = sc.nextLine();
                System.out.println(Palindrome(str));

   }
}
```

TIME COMPLEXITY : O(n)

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.

```
import java.util.*;

public class Prefix{
   public static String CommonPrefix(String[] v) {
      StringBuilder ans = new StringBuilder();
      Arrays.sort(v);
      String first = v[0];
      String last = v[v.length - 1];
      for (int i = 0; i < Math.min(first.length(), last.length()); i++) {
         if (first.charAt(i) != last.charAt(i)) {
            return ans.toString();
         }
         ans.append(first.charAt(i));
      }
      return ans.toString();
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
```

```
        int n = sc.nextInt();
        sc.nextLine();
        String[] arr = new String[n];

        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextLine();
        }

        System.out.println(CommonPrefix(arr));

    }
}
```

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,

```
import java.util.*;

public class Stack1 {

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      int n = sc.nextInt();
      Stack<Character> st = new Stack<>();

      for (int i = 0; i < n; i++) {
         st.push(sc.next().charAt(0));
      }

      int size = st.size();
      int middleIndex = size / 2;

      Stack<Character> tempStack = new Stack<>();
      int currentIndex = 0;


      while (!st.isEmpty()) {
         char currentElement = st.pop();
         if (currentIndex != middleIndex) {
            tempStack.push(currentElement);
         }
         currentIndex++;
```

```
    }



    while (!tempStack.isEmpty()) {
      st.push(tempStack.pop());
    }


            while (!st.isEmpty()) {
      System.out.print(st.pop() + " ");
    }
  }
}
```

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 5 –> 25 2 –> 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 –> -1 7 –> 12 6 –> 12 12 –> -1

Explanation: 13 and 12 don‟t have any element greater than them present on the right side

```
import java.util.*;


public class NextGreaterElement {


  public static void Element(int[] arr) {
    Stack<Integer> stack = new Stack<>();
    int n = arr.length;


    for (int i = n - 1; i >= 0; i--) {
      while (!stack.isEmpty() && stack.peek() <= arr[i]) {
        stack.pop();
      }


      if (stack.isEmpty()) {
        System.out.println(arr[i] + " -1");
      } else {
        System.out.println(arr[i] + "  " + stack.peek());
      }


      stack.push(arr[i]);
    }
  }


  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
```

```
        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        Element(arr);
    }
}
```

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

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

```
import java.util.*;
class TreeNode{
int val;
TreeNode left;
TreeNode right;
TreeNode(int val) {
this.val = val;
left = null;
right = null;
}
}
```

```
class Rightview{
public List<Integer> rightSideView(TreeNode root) {
List<Integer> result = new ArrayList<Integer>();
rightView(root, result, 0);
return result;
}
public void rightView(TreeNode curr, List<Integer> result, int currDepth) {
if (curr == null) {
return;
}
if (currDepth == result.size()) {
result.add(curr.val);
}
rightView(curr.right, result, currDepth + 1);
rightView(curr.left, result, currDepth + 1);
}
public static void main(String[] args) {
TreeNode root = new TreeNode(1);
root.left = new TreeNode(2);
root.right = new TreeNode(3);
root.left.right = new TreeNode(5);
root.right.right = new TreeNode(4);
Question19 solution = new Question19();
List<Integer> result = solution.rightSideView(root);
System.out.println(result);
}
}
```

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

```
import java.util.*;
class TreeNode {
 int val;
 TreeNode left;
 TreeNode right;
 TreeNode(int val) {
 this.val = val;
 this.left = null;
 this.right = null;
 }
}
class Height{
 public int maxDepth(TreeNode root) {
 if (root == null) {
 return 0;
 }
 int left = maxDepth(root.left);
 int right = maxDepth(root.right);
 return Math.max(left, right) + 1;
 }
```

```
 public static void main(String[] args) {
 TreeNode root = new TreeNode(1);
 root.left = new TreeNode(2);
 root.right = new TreeNode(3);
 root.left.left = new TreeNode(4);
 root.left.right = new TreeNode(5);
 root.right.right = new TreeNode(6);
 Question20 main = new Question20();
 int depth = main.maxDepth(root);
 System.out.println("Maximum Depth of the Tree: " + depth);
 }
}
```

TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)