DSA Practice Solutions – 09.11.24

NAME: N J Meenakshi

DEPARTMENT: CSBS

ROLL NO.: 22CB028

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.

   Code:

```java
import java.util.*;
public class Max_subarray_sum{
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter n: ");
        int n=sc.nextInt();
        int[] nums=new int[n];
        for (int i=0;i<n;i++){
            nums[i]=sc.nextInt();
        }
        int l=0;
        int r=0;
        int cs=nums[r];
        int res=Integer.MIN_VALUE;
        while(l<=r && r<nums.length){
            res=Math.max(res,cs);
            if (cs<0){
                l=r+1;
                r+=1;
                if (l<nums.length){
                    cs=nums[l];
                    res=0;
                }
            }
            else{
                r+=1;
                if (r<nums.length) cs+=nums[r];
            }
        }
        System.out.println("Max sum = "+Math.max(res,cs));
        sc.close();
    }
}
```

Output:

```
Enter n:
7
2    3 -8 7 -1 2 3
Max sum = 11

Enter n:
2
-2 -4
Max sum = -2

Enter n:
5
5 4 1 7 8
Max sum = 25
```

Time Complexity – O(N)

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

   Code:

```java
import java.util.Scanner;

public class Max_subarray_pdt {
    public static int solution(int arr[], int n){
        int p1=arr[0],p2=arr[0],res=arr[0];
        for(int i=1;i<n;i++){
            int temp=Math.max(arr[i],Math.max(p1*arr[i],p2*arr[i]));
            p2=Math.min(arr[i],Math.min(arr[i]*p1,arr[i]*p2));
            p1=temp;
            res=Math.max(res,p1);
        }
        return res;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        int nums[]=new int[n];
        for(int i=0;i<n;i++){
            nums[i]=sc.nextInt();
        }
        System.out.println("Max product: "+solution(nums,n));
        sc.close();
    }
}
```

Output:
Enter n: 6
-2 6 -3 -10 0 2
Max product: 180

Enter n: 5
-1 -3 -10 0 60
Max product: 60

Time Complexity – O(N)

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.

   Code:

```java
import java.util.Scanner;

public class Search_in_rotated_arr {
    public static int solution(int arr[], int target){
        int l=0,h=arr.length-1;
        while (l<=h){
            int mid=(l+h)/2;
            if (target==arr[mid]) return mid;
            if(arr[l]<=arr[mid]){
                if(target>=arr[l] && target<=arr[mid]){
                    h=mid-1;
                }
                else{
                    l=mid+1;
                }
            }
            else{
                if(target>=arr[mid] && target<=arr[h]){
                    l=mid+1;
                }
                else{
                    h=mid-1;
                }
            }
        }
        return -1;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        int arr[]=new int[n];
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Enter target: ");
        int tar=sc.nextInt();
```

```
        System.out.print(solution(arr, tar));
        sc.close();
    }
}
```

Output:

Enter n: 7
4 5 6 7 0 1 2
Enter target: 0
4

Enter n: 7
4 5 6 7 0 1 2
Enter target: 3
-1

Enter n: 5
50 10 20 30 40
Enter target: 10
1

4. Container with Most Water

Code:

```
import java.util.Scanner;

public class Container_with_most_water {
    public static int solution(int arr[]){
        int l=0,r=arr.length-1,res=0,currarea=0;
        while(l<r){
            currarea=Math.min(arr[l],arr[r])*(r-l);
            res=Math.max(res,currarea);
            if(arr[l]<arr[r]) l+=1;
            else r-=1;
        }
        return res;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner((System.in));
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        int arr[] = new int[n];
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        System.out.print("Maximum area is: "+solution(arr));
        sc.close();
    }
}
```

Output:

Enter n: 4
1 5 4 3
Maximum area is: 6

Enter n: 5
3 1 2 4 5
Maximum area is: 12

Time Complexity – O(N)

5. Find the Factorial of a large number

Code:

```
import java.math.BigInteger;
import java.util.Scanner;

public class Factorial_of_large_no {
    public static BigInteger solution(int n){
        BigInteger factorial = BigInteger.ONE;
        for (int i = 1; i <= n; i++) {
            factorial = factorial.multiply(BigInteger.valueOf(i));
        }
        return factorial;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        System.out.print("Factorial is: "+solution(n));
        sc.close();
    }
}
```

Output:
Enter n: 100
Factorial is:
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000

Enter n: 50
Factorial is: 30414093201713378043612608166064768844377641568960512000000000000

Time Complexity – O(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.

Code:

```java
import java.util.Scanner;

public class Trapping_rainwater {
    public static int solution(int arr[]){
        int l=0,r=arr.length-1,lMax=0,rMax=0,total=0;
        while(l<r){
            if (arr[l]<=arr[r]){
                if (lMax>arr[l]) total+=lMax-arr[l];
                else lMax=arr[l];
                l+=1;
            }
            else{
                if(rMax>arr[r]) total+=rMax-arr[r];
                else rMax=arr[r];
                r-=1;
            }
        }
        return total;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        int nums[]=new int[n];
        for(int i=0;i<n;i++){
            nums[i]=sc.nextInt();
        }
        System.out.print("Rainwater Trapped: "+solution(nums));
        sc.close();
    }
}
```

Output:

Enter n: 7
3 0 1 0 4 0 2
Rainwater Trapped: 10

Enter n: 5
3 0 2 0 4
Rainwater Trapped: 7

Enter n: 4
1 2 3 4
Rainwater Trapped: 0

Enter n: 4
10 9 0 5
Rainwater Trapped: 5

Time Complexity - O(N)

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.

   Code:

```java
import java.util.Arrays;
import java.util.Scanner;

public class Chocolate_dist {
    public static int solution(int chocolate[], int m){
        int res=Integer.MAX_VALUE;
        Arrays.sort(chocolate);
        for (int i=0;i<(chocolate.length-m+1);i++){
            int diff=chocolate[i+m-1]-chocolate[i];
            res=Math.min(res,diff);
        }
        return res;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        int nums[]=new int[n];
        for(int i=0;i<n;i++){
            nums[i]=sc.nextInt();
        }
        System.out.print("Enter m: ");
        int m=sc.nextInt();
        System.out.print("Minimum difference : "+solution(nums, m));
        sc.close();
    }
}
```

   Output:

   Enter n: 7
   7 3 2 4 9 12 56
   Enter m: 3
   Minimum difference : 2

   Enter n: 7
   7 3 2 4 9 12 56
   Enter m: 5
   Minimum difference : 7

   Time Complexity: O(NlogN)

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].

   Code:

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class MergeIntervals {
    public static List<int[]> solution(int[][] arr){
        List<int[]> res=new ArrayList<>();
        Arrays.sort(arr, (a,b) -> Integer.compare(a[0], b[0]));
        int[] prev=arr[0];
        for(int i=1;i<arr.length;i++){
            int[] current = arr[i];
            if(current[0]<=prev[1]){
                prev[1]=Math.max(current[1], prev[1]);
            }
            else{
                res.add(prev);
                prev=current;
            }
        }
        res.add(prev);
        return res;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        int [][] intervals = new int[n][2];
        for(int i=0;i<n;i++){
            intervals[i][0]=sc.nextInt();
            intervals[i][1]=sc.nextInt();
        }
        System.out.print("Merged Intervals: ");
        List<int[]> mergedIntervals = solution(intervals);
        for(int[] interval:mergedIntervals){
            System.out.println(Arrays.toString(interval));
        }
        sc.close();
    }
}
```

Output:

Enter n: 4
1 3
2 4
6 8
9 10
Merged Intervals: [1, 4]
[6, 8]
[9, 10]

Enter n: 4
7 8
1 5
2 4
4 6
Merged Intervals: [1, 6]
[7, 8]

Time Complexity: O(NlogN)

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.

Code:

```
import java.util.Scanner;

public class Boolean_Matrix {
    public static void modifyMat(int mat[][]){
        int m=mat.length;
        int n=mat.length;
        boolean firstR =false, firstC=false;
        for(int j=0;j<n;j++){
            if(mat[0][j]==1){
                firstR=true;
                break;
            }
        }
        for(int j=0;j<m;j++){
            if(mat[0][j]==1){
                firstC=true;
                break;
            }
        }
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                if (mat[i][j] == 1) {
                    mat[i][0] = 1;
                    mat[0][j] = 1;
                }
            }
        }
```

```
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                if (mat[i][0] == 1 || mat[0][j] == 1) {
                    mat[i][j] = 1;
                }
            }
        }
        if (firstR) {
            for (int j = 0; j < n; j++) {
                mat[0][j] = 1;
            }
        }
        if (firstC) {
            for (int i = 0; i < m; i++) {
                mat[i][0] = 1;
            }
        }
    }
    public static void printMatrix(int mat[][]) {
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[0].length; j++) {
                System.out.print(mat[i][j] + " ");
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter m: ");
        int m = sc.nextInt();
        System.out.print("Enter n: ");
        int n = sc.nextInt();
        int[][] mat = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                mat[i][j] = sc.nextInt();
            }
        }
        modifyMat(mat);
        System.out.println("Modified Matrix: ");
        printMatrix(mat);
        sc.close();
    }
}
```

Output:

Enter m: 2

Enter n: 2

1 0

0 0

Modified Matrix:

1 1

1 0

Enter m: 2
Enter n: 3
0 0 0
0 0 1
Modified Matrix:
0 0 1
1 1 1

Enter m: 3
Enter n: 4
1 0 0 1
0 0 1 0
0 0 0 0
Modified Matrix:
1 1 1 1
1 1 1 1
1 0 1 1

Time Complexity:

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.

Code:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Spiral_matrix {
    public static List<Integer> solution(int[][] matrix){
        List<Integer> res=new ArrayList<>();
        int n=matrix.length, m=matrix[0].length;
        int top=0, bottom=n-1, left=0, right=m-1;
        while (res.size()<n*m){
            for(int i=left;i<=right;i++){
                res.add(matrix[top][i]);
            }
            top+=1;
            for(int i=top;i<=bottom;i++){
                res.add(matrix[i][right]);
            }
            right-=1;
            if (top<=bottom){
                for(int i=right;i>=left;i--){
                    res.add(matrix[bottom][i]);
                }
                bottom-=1;
            }
            if(right>=left){
                for(int i=bottom;i>=top;i--){
                    res.add(matrix[i][left]);
                }
```

```
                left+=1;
            }
        }
        return res;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        System.out.print("Enter m: ");
        int m=sc.nextInt();
        int [][] matrix = new int[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                matrix[i][j]=sc.nextInt();
            }
        }
        System.out.print("Spiral Matrix: "+solution(matrix));
        sc.close();
    }
}
```

Output:

Enter n: 4
Enter m: 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
Spiral Matrix: [1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10]

Enter n: 3
Enter m: 6
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
Spiral Matrix: [1, 2, 3, 4, 5, 6, 12, 18, 17, 16, 15, 14, 13, 7, 8, 9, 10, 11]

Time Complexity: O(n*m)

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.

    Code:

```
import java.util.Scanner;
import java.util.Stack;

public class Balanced_Stack {
    public static boolean solution(String s){
        Stack<Character> st=new Stack<>();
        for(char ch:s.toCharArray()){
```

```
            if (ch=='(' || ch=='[' || ch=='{'){
                st.push(ch);
            }
            else{
                if (!st.isEmpty()){
                    char top=st.pop();
                    if((ch==')' && top!='(')|| (ch=='}' && top!='{') || (ch==']' &&
top!='[')){
                        return false;
                    }
                }
                else{
                    return false;
                }
            }
        }
        return st.isEmpty();
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter s: ");
        String s=sc.next();
        boolean res=solution(s);
        if (res==true) System.out.print("Balanced");
        else System.out.print("Not Balanced");
        sc.close();
    }
}
```

Output:
Enter s: ((()))()()
Balanced

Enter s: ())((())
Not Balanced

Time Complexity: O(N)

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.
    Code:

```
import java.util.Scanner;

public class Anagrams {
    public static boolean solution(String s, String t){
        if (s.length()!=t.length()) return false;
        int[] freq=new int[26];
        for(char ch:s.toCharArray()){
            freq[ch-'a']++;
        }
        for(char ch:t.toCharArray()){
```

```
            freq[ch-'a']--;
        }
        for(int count:freq){
            if (count!=0) return false;
        }
        return true;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter s: ");
        String s=sc.next();
        System.out.print("Enter t: ");
        String t=sc.next();
        System.out.print("Result: "+solution(s, t));
        sc.close();
    }
}
```

Output:
Enter s: geeks
Enter t: kseeg
Result: true

Enter s: allergy
Enter t: allergic
Result: false

Enter s: g
Enter t: g
Result: true

Time 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.

    Code:

```
import java.util.Scanner;

public class Longest_Pal_Substring {
    public static String solution(String s){
        if (s==null || s.length()<1) return "";
        String res="";
        for (int i=0;i<s.length();i++){
            String sub1=expandAtCenter(s,i,i);
            if(sub1.length()>res.length()) res=sub1;
            String sub2=expandAtCenter(s,i,i+1);
```

```
            if(sub2.length()>res.length()) res=sub2;
        }
        return res;
    }
    public static String expandAtCenter(String s, int left, int right){
        while((left>=0 && right<s.length()) && (s.charAt(left)==s.charAt(right))){
            left--;
            right++;
        }
        return s.substring(left+1,right);
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter s: ");
        String s=sc.next();
        System.out.print("Longest Palindromic Substring: "+solution(s));
        sc.close();
    }
}
```

Output:
Enter s: forgeeksskeegfor
Longest Palindromic Substring: geeksskeeg

Enter s: Geeks
Longest Palindromic Substring: ee

Enter s: abc
Longest Palindromic Substring: a

Enter s:
""
Longest Palindromic Substring: ""

Time Complexity: O(N^2)

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.

    Code:

```
import java.util.Scanner;

public class Longest_common_prefix {
    public static String longestCommonPrefix(String[] strs){
        if(strs==null || strs.length==0) return "";
        String prefix=strs[0];
        for(int i=1;i<strs.length;i++){
            while (strs[i].indexOf(prefix)!=0){
```

```
                prefix=prefix.substring(0,prefix.length()-1);
                if (prefix.isEmpty()){
                    return "-1";
                }
            }
        }
        if (prefix=="") return "-1";
        return prefix;
    }
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        String[] s=new String[n];
        for(int i=0;i<n;i++){
            s[i]=sc.next();
        }
        System.out.print("Longest Common Prefix: "+longestCommonPrefix(s));
        sc.close();
    }
}
```

Output:
Enter n: 4
geeksforgeeks geeks geek geezer
Longest Common Prefix: gee

Enter n: 2
hello world
Longest Common Prefix: -1

Time Complexity: O(n*m)

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.

    Code:

```
import java.util.Scanner;
import java.util.Stack;
public class Delete_middle_of_stk {
    public static void solution(Stack<Integer> st){
        int size=st.size();
        if(size==0) return;
        int mid=size/2;
        for(int i=0;i<size;i++){
            int ele=st.pop();
            if(i!=mid){
                st.push(ele);
            }
        }
    }
```

```
    public static void main(String args[]){
        Scanner sc= new Scanner(System.in);
        Stack<Integer> stack=new Stack<>();
        System.out.print("Enter n: ");
        int n=sc.nextInt();
        System.out.print("Enter Stack: ");
        for(int i=0;i<n;i++){
            stack.push(sc.nextInt());
        }
        System.out.print("Stack after deleting middle element is: ");
        solution(stack);
        System.out.print(stack);
        sc.close();
    }
}
```

Output:
Enter n: 5
Enter Stack: 1 2 3 4 5
Stack after deleting middle element is: [1, 2, 3, 4]

Time 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

Code:

```
import java.util.*;
public class Next_Greatest_Element {
    public static int[] solution(int[] nums){
        int[] res=new int[nums.length];
        int n=nums.length;
        Arrays.fill(res,-1);
        Deque<Integer> s=new ArrayDeque<>();
        for(int i=0;i<2*nums.length;i++){
            while(!s.isEmpty() && nums[i%n]>nums[s.peek()]){
                res[s.pop()]=nums[i%n];
            }
            s.push(i%n);
        }
        return res;
    }
    public static void main(String args[]){
        Scanner sc= new Scanner(System.in);
        System.out.print("Enter n: ");
        int n=sc.nextInt();
```

```
        int[] nums=new int[n];
        System.out.print("Enter array: ");
        for(int i=0;i<n;i++){
            nums[i]=sc.nextInt();
        }
        int[] res=solution(nums);
        System.out.print("The result is: ");
        for (int i=0;i<n;i++){
            System.out.print(res[i]+" ");
        }
        sc.close();
    }
}
```

Output:
Enter n: 4
Enter array: 4 5 2 25
The result is: 5 25 25 -1

Enter n: 4
Enter array: 13 7 6 12
The result is: -1 12 12 13

Time 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.ArrayList;
import java.util.List;

class Node{
    int data;
    Node left;
    Node right;
    Node(int val){
        data=val;
        left=right=null;
    }
}
public class Right_View_of_BT {
    public static List<Integer> rightView(Node root){
        List<Integer> res=new ArrayList<>();
        recursionRight(root,0,res);
        return res;
    }
    public static void recursionRight(Node root, int lev, List<Integer> res){
        if(root==null) return;
        if (res.size()==lev) res.add(root.data);
        recursionRight(root.right, lev+1, res);
```

```
        recursionRight(root.left,lev+1,res);
    }
    public static void main(String args[]){
        Node root= new Node(1);
        root.left=new Node(2);
        root.right=new Node(3);
        root.left.left=new Node(4);
        root.left.left.right=new Node(5);

        List<Integer> rightView=rightView(root);
        System.out.print("Right View Traversal: ");
        for(int node:rightView){
            System.out.print(node+" ");
        }
    }
}
```

Output:
Right View Traversal: 1 3 5

Right View Traversal: 1 3 4 5

Time 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:

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

public class Max_Depth_of_BT {
    public static int maxDepth(Node root) {
        if (root == null) {
            return 0;
        }
        int leftDepth = maxDepth(root.left);
        int rightDepth = maxDepth(root.right);
        return Math.max(leftDepth, rightDepth) + 1;
    }
    public static void main(String args[]) {
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
```

```
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);
        int depth = maxDepth(root);
        System.out.println("Maximum Depth of the Binary Tree: " + depth);
    }
}
```

Output:
Maximum Depth of the Binary Tree: 3
Maximum Depth of the Binary Tree: 4

Time Complexity: O(N)