DSA Practice 1 (09-11-2024)

NAME: Atchayaa T

DEPARTMENT: B.Tech CSBS

REGISTER NO.: 22CB004

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.

   Code:

```
import java.lang.reflect.Array;
import java.util.*;

public class KadaneAlgorithm{
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int n;
        System.out.println("Enter number of elements: ");
        n = s.nextInt();
        int[] nums = new int[n];

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

        int sum = Integer.MIN_VALUE;
        int l=0; int r=0;
        int cs = nums[0];

        while (r<=nums.length && l<=r){
            sum = Math.max(cs,sum);
            if (cs<0){
                l = r+1;
                r+=1;
                if (l<nums.length){
                    cs = nums[l];
                    sum = Math.max(sum,cs);
                }

            }
            else{
                r+=1;
```

```
                if (r<nums.length)
                    cs+=nums[r];
            }

        }

        System.out.println(Math.max(cs,sum));

        s.close();
    }
}
```

Output:

Enter number of elements:

7

2 3 -8 7 -1 2 3

11

Enter number of elements:

2

-2 -4

-2

Enter number of elements:

5

5 4 1 7 8

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

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

   Code:

```
import java.util.Scanner;
public class Max_product_subarray {

    public static int solution(int[] nums, int n){
        int r = nums[0];
        int p1 = nums[0];
        int p2 = nums[0];
```

```
        for (int i=1;i<n;i++){
            int temp = Math.max(nums[i],Math.max(p1*nums[i],p2*nums[i]));
            p2 = Math.min(nums[i],Math.min(p1*nums[i],p2*nums[i]));
            p1 = temp;

            r = Math.max(r,p1);
        }

        return r;
    }
    public static void main(String[] args) {
        int n;
        Scanner s = new Scanner(System.in);
        System.out.println("Enter number of elements: ");
        n = s.nextInt();

        int[] nums = new int[n];
        for (int i=0;i<n;i++){
            nums[i] = s.nextInt();
        }

        System.out.println(solution(nums,n));

    }
}
```

Output:

1. Enter number of elements:
   5
   -1 -3 -10 0 60
   60

2. Enter number of elements:
   6
   -2 6 -3 -10 0 2
   180

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.

   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.Scanner;

public class SearchInRotatedSortedArray {
    public static void main(String[] args) {
        int n;
        Scanner s = new Scanner(System.in);
        n = s.nextInt();

        int nums[] = new int[n];
        for (int i=0;i<n;i++){
            nums[i] = s.nextInt();
        }

        int t = s.nextInt();

        int l=0;
        int h=n-1;
        int mid=0;
        Boolean flag=false;

        while (l<=h)
        {
            mid = (l+h)/2;
            if (t==nums[mid])
            {System.out.println(mid);
            flag = true;
             break;}

             if (nums[l]<=nums[mid]){
                    if (t>=nums[l] && t<=nums[mid]){
                        h = mid-1;
                    }
                    else{
                        l = mid+1;
                    }
             }
             else{
                if (t>=nums[mid+1] && t<=nums[h]){
                    l=mid+1;
                }
                else{
                    h=mid-1;
                }
             }
        }
        if (flag==false){
        System.out.println(-1);}
    }
```

```
}
```

Output:

7

4 5 6 7 0 1 2

2

6

Time Complexity: O(log n)

4. Container with Most Water

Given n non-negative integers $a_1, a_2, \ldots, a_n$ where each represents a point at coordinate (i, a_i). ' n ' vertical lines are drawn such that the two endpoints of line i is at (i, a_i) and $(i, 0)$. Find two lines, which together with x-axis forms a container, such that the container contains the most water.

The program should return an integer which corresponds to the maximum area of water that can be contained (maximum area instead of maximum volume sounds weird but this is the 2D plane we are working with for simplicity).

Note: You may not slant the container.

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.Scanner;

public class ContainerWithMostWater
```

```
{
    public static int solution(int[] height) {
        int m = 0;
        int l = 0;
        int r = height.length - 1;

        while (l < r) {
            m = Math.max(m, (r - l) * Math.min(height[l], height[r]));

            if (height[l] < height[r]) {
                l++;
            } else {
                r--;
            }
        }

        return m;
    }

    public static void main(String[] args) {
        System.out.println("Enter n: ");
        int n;
        Scanner s = new Scanner(System.in);
        n = s.nextInt();

        int nums[] = new int[n];
        System.out.println("Enter values: ");
        for (int i=0;i<n;i++){
            nums[i] = s.nextInt();
        }
        System.out.println(solution(nums));

    }
}
```

Output:

1. Enter n:
   5
   Enter values:
   3 1 2 4 5
   12
2. Enter n:
   4
   Enter values:
   1 5 4 3
   6

Time Complexity: O(n)

5. Find the Factorial of a large number

   Input: 100

   Output: 33262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000

   Input: 50

   Output: 30414093201713378043612608166064768844377641568960512000000000000

   Code:

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

public class Factorial {
    public static BigInteger factorial(int n) {
        BigInteger result = BigInteger.ONE;
        for (int i = 2; i <= n; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }
        return result;
    }

    public static void main(String[] args) {
        System.out.println("Enter n: ");
        int n;
        Scanner s = new Scanner(System.in);
        n = s.nextInt();
        System.out.println(factorial(n));
    }
}
```

   Output:

   Enter n:

   100

   93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

   Enter n:

   50

   30414093201713378043612608166064768844377641568960512000000000000

   Enter n:

   300

   30605751221644063603537046129726862938858880417357699941677674125947653317671686746551529142247757334993914788870172636886426390775900315422684292790697455984122547693027195460400801221577625217685425596535690350678872526432189626429936520457644883038890975394348962543605322598077652127082243763944912012867867536830571229368194364995646049816645022771650018517654646934011222603472972406633325858350687015016979416885035375213755491028912640715715483028228493795263658014523523315693648223343679925459 4

09527682060806223281238738388081704960000000000000000000000000000000000000000000000
000000000000000000000000000

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.
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.Scanner;

class TrappingRainWater {
    public static int solution(int[] height) {
        int l = 0;
        int r = height.length - 1;
        int lmax = height[l];
        int rmax = height[r];
        int w = 0;

        while (l < r) {
            if (lmax < rmax) {
                l++;
                lmax = Math.max(lmax, height[l]);
                w += lmax - height[l];
            } else {
                r--;
                rmax = Math.max(rmax, height[r]);
                w += rmax - height[r];
            }
        }

        return w;
    }

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

```
        System.out.println("Enter n: ");
        int n;
        Scanner s = new Scanner(System.in);
        n = s.nextInt();

        int nums[] = new int[n];
        System.out.println("Enter values: ");
        for (int i=0;i<n;i++){
            nums[i] = s.nextInt();
        }
        System.out.println(solution(nums));
        s.close();

    }
}
```

Output:

1. Enter n:

   5

   Enter values:

   3 0 2 0 4

   7

2. Enter n:

   4

   Enter values:

   1 2 3 4

   0

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.

   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:

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

class ChocolateDistributionProblem {
    public static int solution(int[] arr, int m) {
        int n = arr.length;

        Arrays.sort(arr);
        int t = Integer.MAX_VALUE;
        for (int i = 0; i + m - 1 < n; i++) {
            int diff = arr[i + m - 1] - arr[i];
            if (diff < t)
                t = diff;
        }
        return t;
    }

    public static void main(String[] args) {
        System.out.println("Enter n: ");
            int n;
            Scanner s = new Scanner(System.in);
            n = s.nextInt();

            int nums[] = new int[n];
            System.out.println("Enter values: ");
            for (int i=0;i<n;i++){
                nums[i] = s.nextInt();
            }
            System.out.println("Enter m: ");
            int m = s.nextInt();
            System.out.println(solution(nums,m));
            s.close();
    }
}
```

Output:

1. Enter n:

   7

   Enter values:

   7 3 2 4 9 12 56

   Enter m:

   3

   2

2. Enter n:

   7

Enter values:

7 3 2 4 9 12 56

Enter m:

5

7

Time Complexity: O(n log n)

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:

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

public class MergeOverlappingIntervals {

    public static List<int[]> solution(int[][] arr) {
        Arrays.sort(arr, (a, b) -> Integer.compare(a[0], b[0]));

        List<int[]> res = new ArrayList<>();
        res.add(new int[]{arr[0][0], arr[0][1]});

        for (int i = 1; i < arr.length; i++) {
            int[] last = res.get(res.size() - 1);
            int[] curr = arr[i];
            if (curr[0] <= last[1])
                last[1] = Math.max(last[1], curr[1]);
            else
                res.add(new int[]{curr[0], curr[1]});
```

```
        }

        return res;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter n: ");
        int n = s.nextInt();

        int[][] arr = new int[n][2];
        System.out.println("Enter intervals: ");
        for (int i = 0; i < n; i++) {
            System.out.print("Enter start " + (i + 1) + ": ");
            arr[i][0] = s.nextInt();
            System.out.print("Enter end  " + (i + 1) + ": ");
            arr[i][1] = s.nextInt();
        }

        List<int[]> res = solution(arr);

        System.out.println("Merged intervals are:");


        for (int i = 0; i < res.size(); i++) {
            int[] interval = res.get(i);
            System.out.println(interval[0] + " " + interval[1]);
        }


        s.close();
    }
}
```

Output:

Enter n: 4

Enter intervals:

Enter start 1: 1

Enter end 1: 3

Enter start 2: 2

Enter end 2: 4

Enter start 3: 6

Enter end 3: 8

Enter start 4: 9

Enter end 4: 10

Merged intervals are:

1 4

6 8

9 10

Time Complexity: O(n logn)

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}}

   Code:

```
import java.util.*;

class BooleanMatrix {
    public static void main(String[] args) {
        int[][] mat = {{1, 0}, {0, 0}};
        ArrayList<Integer> row = new ArrayList<>();
        ArrayList<Integer> col = new ArrayList<>();
        int r = mat.length;
        int c = mat[0].length;

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                if (mat[i][j] == 1) {
                    row.add(i);
                    col.add(j);
                }
            }
        }

        for (int i = 0; i < row.size(); i++) {
            for (int j = 0; j < c; j++) {
                int q = row.get(i);
                mat[q][j] = 1;
            }
        }

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < col.size(); j++) {
                int p = col.get(j);
```

```
                mat[i][p] = 1;
            }
        }

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                System.out.print(mat[i][j]);
            }
            System.out.println();
        }
    }
}
```

Output:

11

10

Time 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.

    Code:

```
import java.util.*;

public class Spiral {
    public static void spiralMatrix(int m, int n, int[][] a) {
        int top = 0, bottom = m - 1, left = 0, right = n - 1;
        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; ++i) {
                System.out.print(a[top][i] + " ");
            }
            top++;
            for (int i = top; i <= bottom; ++i) {
                System.out.print(a[i][right] + " ");
            }
            right--;
            if (top <= bottom) {
                for (int i = right; i >= left; --i) {
                    System.out.print(a[bottom][i] + " ");
                }
                bottom--;
            }
            if (left <= right) {
```

```
                for (int i = bottom; i >= top; --i) {
                    System.out.print(a[i][left] + " ");
                }
                left++;
            }
        }
    }

    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3, 4},
            {5, 6, 7, 8},
            {9, 10, 11, 12},
            {13, 14, 15, 16}
        };
        spiralMatrix(matrix.length, matrix[0].length, matrix);
    }
}
```

Output:
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
Time Complexity: O(m*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:

```
import java.util.*;

class checkparantheses {
    public static void main(String Args[]) {
        Scanner sc = new Scanner(System.in);
        String str = sc.nextLine();
        int count = 0;
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == ')') {
                count -= 1;
            } else {
                count++;
            }
        }
        if (count == 0) {
            System.out.println("Balanced");
        } else {
            System.out.println("Not balanced");
        }
    }
}
```

Output:

(()))()()

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.

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.*;

class anagram {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str1 = sc.nextLine();
        String str2 = sc.nextLine();

        if (str1.length() != str2.length()) {
            System.out.println("false");
        } else {
            char[] s1 = str1.toCharArray();
            char[] s2 = str2.toCharArray();

            Arrays.sort(s1);
            Arrays.sort(s2);

            int c = s2.length;
            for (int i = 0; i < s1.length; i++) {
                if (s1[i] == s2[i]) {
                    c -= 1;
                }
                if (c == 0) {
                    System.out.println("true");
                    return;
                }
            }
        }
```

```
    }
}
```

Output:

allergy

allergic

false

Time Complexity: O(n log 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: “”

Code:

```
import java.util.*;

class longestpalsubstring {

    public static void main(String[] args) {
        String s = "abc";
        String res = "";
        int resLen = 0;

        for (int i = 0; i < s.length(); i++) {
            int l = i, r = i;
            while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
                if ((r - l + 1) > resLen) {
                    res = s.substring(l, r + 1);
                    resLen = r - l + 1;
                }
                l--;
                r++;
            }
            l = i;
            r = i + 1;
            while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {
                if ((r - l + 1) > resLen) {
```

```
                    res = s.substring(l, r + 1);
                    resLen = r - l + 1;
                }
                l--;
                r++;
            }
        }

        System.out.println("Longest Palindromic Substring: " + res);
    }
}
```

Output:

Longest Palindromic Substring: a

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.

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

Code:

```
import java.util.*;

class longcommonpre {
    public static void main(String[] args) {
        String[] strs = {"hello","world"};

        if (strs.length == 1) {
            System.out.println(strs[0]);
        }

        Arrays.sort(strs);

        int i = 0;
        String a = strs[0];
        String b = strs[strs.length - 1];

        while (i < a.length() && i < b.length()) {
            if (a.charAt(i) == b.charAt(i)) {
                i++;
            } else {
                break;
            }
```

```
        }

        System.out.println(a.substring(0, i));
    }
}
```

Output:

-1

Time Complexity: O(n logn)

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]

Code:

```
import java.util.*;

class stack1 {
    public static void main(String[] args) {
        Stack<Integer> st = new Stack<>();
        st.push(1); st.push(2); st.push(3); st.push(4); st.push(5); st.push(6);

        int n = st.size() / 2;
        Stack<Integer> temp = new Stack<>();

        for (int i = 0; i < n; i++) {
            temp.push(st.pop());
        }

        st.pop();

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

        for (int i : st) {
            System.out.println(i);
        }
    }
}
```

Output:

1

2

4

5

6

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

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

Code:

```
import java.util.*;

class nextgrtelement {
    public static void main(String[] args) {
        int[] arr = {4, 5, 2, 25};

        for (int i = 0; i < arr.length; i++) {
            int greatest = -1;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] > arr[i]) {
                    greatest = arr[j];
                    break;
                }
            }
            System.out.println(arr[i] + " -> " + greatest);
        }
    }
}
```

Output:

4 -> 5

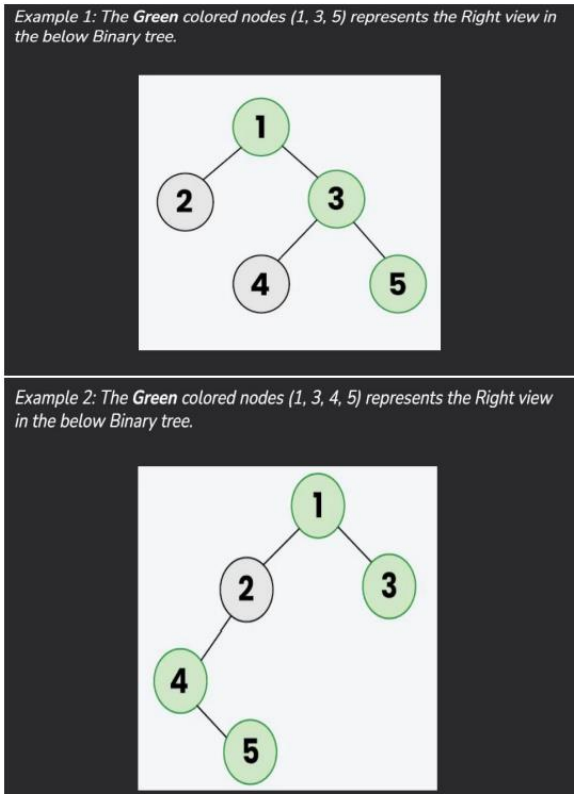
5 -> 25

2 -> 25

25 -> -1

Time Complexity: O(n^2)

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 val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
        left = null;
        right = null;
    }
}

class rightviewofbinarytree {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        rightView(root, result, 0);
        return result;
```

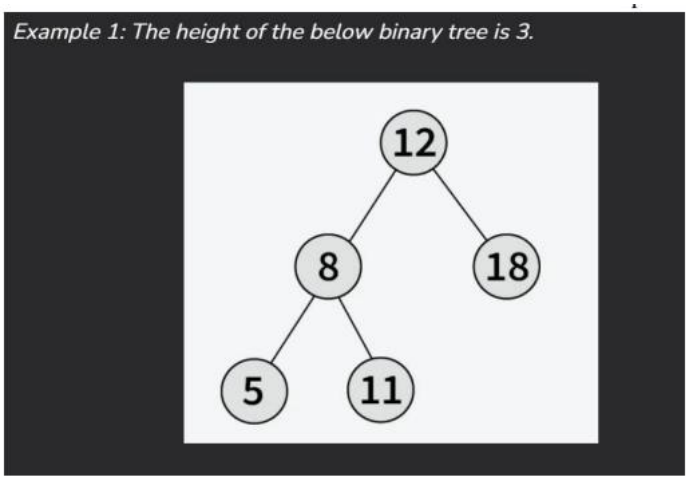
```
    }

    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.right.left = new TreeNode(4);
        root.right.right = new TreeNode(5);

        rightviewofbinarytree solution = new rightviewofbinarytree();
        List<Integer> result = solution.rightSideView(root);
        System.out.println(result);
    }
}
```

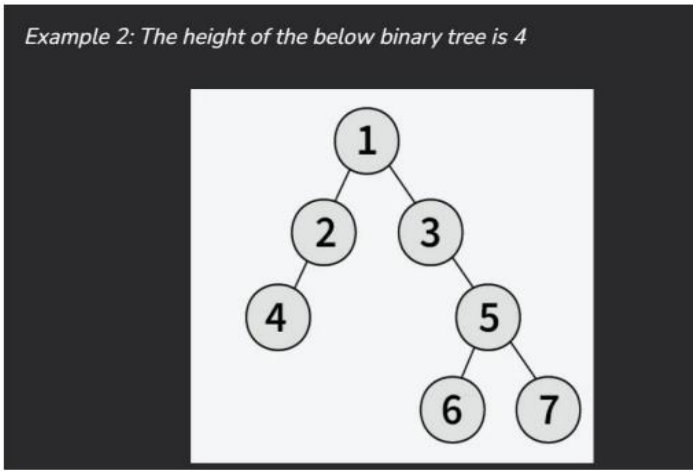
Output:

[1, 3, 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:

```java
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

class maxdepthofbinarytree {
    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(6);
        root.right.right = new TreeNode(7);

        maxdepthofbinarytree main = new maxdepthofbinarytree();
        int depth = main.maxDepth(root);
        System.out.println("Maximum Depth of the Tree: " + depth);
    }
}
```

Output:

Maximum Depth of the Tree: 3

Time Complexity: O(n)