DSA PRACTICE - 1

Name : Tharani Kumar K

DEPT : IT
Date : 09/10/2024

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.

Program:

```
import java.util.Scanner;

class Solution {
    public static int maxSubArray(int[] nums) {
        int res = Integer.MIN_VALUE;
        int ans = 0;
        for (int num : nums) {
            ans += num;
            res = Math.max(res, ans);
            if (ans < 0) {
                ans = 0;
            }
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = sc.nextInt();
```

```
        }
        int result = maxSubArray(nums);
        System.out.println("Maximum subarray sum: " + result);
        sc.close();
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the size of the array:
7
Enter the elements of the array:
2 3 -8 7 -1 2 3
Maximum subarray sum: 11
```

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}

Program:

```
import java.util.*;

class Solution {
    public static int maxProduct(int[] nums) {
        int n = nums.length;
        int ans = Integer.MIN_VALUE;
        double prefixpro = 1;
        double suffixpro = 1;

        for (int i = 0; i < nums.length; i++) {
            prefixpro *= nums[i];
```

```
            suffixpro *= nums[n - i - 1];

            ans = Math.max(ans, Math.max((int) prefixpro, (int) suffixpro));

            if (prefixpro == 0) prefixpro = 1;
            if (suffixpro == 0) suffixpro = 1;
        }
        return ans;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = sc.nextInt();
        }
        int result = maxProduct(nums);
        System.out.println("Maximum product: " + result);
        sc.close();
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ
Enter the size of the array:
6
Enter the elements of the array:
-2 6 -3 -10 0 2
Maximum product: 180
```

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

Program:

```
import java.util.*;

public class Main {
   public static int search(int[] nums, int target) {
      int left = 0;
      int right = nums.length - 1;
      while(left <= right) {
         int mid = (left + right) / 2;
         if(nums[mid] == target) {
            return mid;
         }
         else if(nums[left] <= nums[mid]) {
            if(nums[left] <= target && target < nums[mid]) {
               right = mid - 1;
            }
            else {
               left = mid + 1;
            }
         }
         else {
            if(nums[mid] < target && target <= nums[right]) {
               left = mid + 1;
            }
            else {
               right = mid - 1;
            }
         }
      }
      return -1;
   }

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

```
        System.out.println("Enter the elements of the array:");
        for(int i = 0; i < n; i++) {
            nums[i] = sc.nextInt();
        }
        System.out.println("Enter the target value:");
        int target = sc.nextInt();
        int result = search(nums, target);
        System.out.println("Element found at index: " + result);
        sc.close();
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the size of the array:
7
Enter the elements of the array:
4 5 6 7 0 1 2
Enter the target value:
0
Element found at index: 4
```

Time Complexity: O(logn) (BinarySearch)

Space Complexity: O(1)

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.

Program:

```
import java.util.*;

public class Main {
   public static int maxArea(int[] height) {
      int ans = Integer.MIN_VALUE;
      int n = height.length;
      int b = n - 1;
      int left = 0;
      int right = n - 1;
      while(left <= right) {
         ans = Math.max(ans, (b * Math.min(height[left], height[right])));
         if(height[left] < height[right]) {
            left++;
         } else {
            right--;
         }
         b--;
      }
      return ans;
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the size of the array:");
      int n = sc.nextInt();
      int[] height = new int[n];
      System.out.println("Enter the heights of the bars:");
      for(int i = 0; i < n; i++) {
         height[i] = sc.nextInt();
      }
      int result = maxArea(height);
      System.out.println("Maximum area: " + result);
      sc.close();
   }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the size of the array:
9
Enter the heights of the bars:
1 8 6 2 5 4 8 3 7
Maximum area: 49
```

Time Complexity: O(n)

Space Complexity: O(1)

5) **Find the Factorial of a large number**

Input: 100

Output:

933262154439441526816992388562667004907159682643816214685929638952175999932299

156089414639761565182862536979208272237582511852109168640000000000000000000000 00

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

Program:

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

public class Main{
   public static BigInteger fun(int n) {
      BigInteger res=BigInteger.ONE;
      for(int i=1;i<=n;i++){
         res = res.multiply(BigInteger.valueOf(i));
      }
      return res;

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

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
100
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182
Process finished with exit code 0
```

Time Complexity: O(n)

Space Complexity: O(1)

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

Program:

```
import java.util.Scanner;

public class Main {
    public static int trap(int[] height) {
        int n = height.length;
        int[] left = new int[n];
        int[] right = new int[n];
        left[0] = height[0];
        right[n-1] = height[n-1];
        for(int i=1;i<n;i++){
            left[i] = Math.max(left[i-1],height[i]);
            right[n-1-i] = Math.max(right[n-i],height[n-1-i]);
        }
        int ans = 0;
        for(int i=1;i<n;i++){
            ans+= Math.min(left[i],right[i])-height[i];
        }
```

```
        return ans;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = sc.nextInt();
        int[] height = new int[n];
        System.out.println("Enter the heights of the bars:");
        for(int i = 0; i < n; i++) {
            height[i] = sc.nextInt();
        }
        int result = trap(height);
        System.out.println("Trapped rainwater: " + result);
        sc.close();
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the size of the array:
12
Enter the heights of the bars:
0 1 0 2 1 0 1 3 2 1 2 1
Trapped rainwater: 6
```

Time Complexity: O(n)

Space 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.

Program:

```
import java.util.*;

class Solution {
   public static int helper(int[] arr, int m) {
      Arrays.sort(arr);
      if (arr.length < m) return -1;
      int min = Integer.MAX_VALUE;
      for (int i = m - 1; i < arr.length; i++) {
         min = Math.min(arr[i] - arr[i - m + 1], min);
      }
      return min;
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the Size of Array");
      int n = sc.nextInt();
      System.out.println("Enter the No.of Students");
      int m = sc.nextInt();
      System.out.println("The Elements of Array");
      int[] arr = new int[n];
      for (int i = 0; i < n; i++) {
         arr[i] = sc.nextInt();
      }
      System.out.print(helper(arr, m));
   }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the Size of Array7
Enter the No.of Students3
The Elements of Array7 3 2 4 9 12 56
2
```

Time Complexity: O(nlogn) (due to sorting)

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].

Program:

```
import java.util.*;

public class Main {

   public static int[][] helper(int[][] intervals) {
      if(intervals.length == 1) {
         return intervals;
      }

      Arrays.sort(intervals, (arr1, arr2) -> arr1[0] - arr2[0]);

      List<int[]> res = new ArrayList<>();
      int[] newInterval = intervals[0];
      res.add(newInterval);

      for(int[] interval : intervals) {
         if(newInterval[1] >= interval[0]) {
            newInterval[1] = Math.max(newInterval[1], interval[1]);
         } else {
            newInterval = interval;
            res.add(newInterval);
         }
      }

      return res.toArray(new int[res.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[][] intervals = new int[n][2];

        System.out.println("Enter the intervals :");
        for (int i = 0; i < n; i++) {
            intervals[i][0] = sc.nextInt();
            intervals[i][1] = sc.nextInt();
        }
        int[][] mergedIntervals = helper(intervals);
        System.out.println("Merged intervals:");
        for (int[] interval : mergedIntervals) {
            System.out.println("[" + interval[0] + ", " + interval[1] + "]");
        }
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the number of intervals:
4
Enter the intervals :
1 3
2 4
5 7
6 8
Merged intervals:
[1, 4]
[5, 8]
```

Time Complexity: O(nlogn) (due to sorting)

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}

Program:

```
import java.util.*;

public class Main{
   public static void helper(int[][] mat) {
      int M = mat.length;
      int N = mat[0].length;
      boolean[] rowFlags = new boolean[M];
      boolean[] colFlags = new boolean[N];
      for (int i = 0; i < M; i++) {
         for (int j = 0; j < N; j++) {
            if (mat[i][j] == 1) {
               rowFlags[i] = true;
               colFlags[j] = true;
            }
         }
      }
      for (int i = 0; i < M; i++) {
         for (int j = 0; j < N; j++) {
            if (rowFlags[i] || colFlags[j]) {
               mat[i][j] = 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.println("Enter number of rows (M): ");
      int M = sc.nextInt();
      System.out.println("Enter number of columns (N): ");
      int N = sc.nextInt();
      int[][] mat = new int[M][N];
      System.out.println("Enter the elements of the matrix (0 or 1): ");
      for (int i = 0; i < M; i++) {
         for (int j = 0; j < N; j++) {
            mat[i][j] = sc.nextInt();
```

```
        }
      }
      helper(mat);
      System.out.println("Modified Matrix:");
      printMatrix(mat);
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ
Enter number of rows (M):
3
Enter number of columns (N):
4
Enter the elements of the matrix (0 or 1):
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: O(rXc)

Space Complexity: O(1)

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.

Program:

```
import java.util.*;

public class Main {
   public static List<Integer> helper(int[][] matrix) {
      List<Integer> ans = new ArrayList<>();
      int rowbegin = 0;
      int rowend = matrix.length-1;
      int colbegin = 0;
```

```
    int colend = matrix[0].length-1;
    while(rowbegin<=rowend && colbegin<=colend){
      for(int i=colbegin;i<=colend;i++){
        ans.add(matrix[rowbegin][i]);
      }
      rowbegin++;

      for(int j=rowbegin;j<=rowend;j++){
        ans.add(matrix[j][colend]);
      }
      colend--;

      if(rowbegin<=rowend){
        for(int k=colend;k>=colbegin;k--){
          ans.add(matrix[rowend][k]);
        }
      }
      rowend--;

      if(colbegin<=colend){
        for(int l=rowend;l>=rowbegin;l--){
          ans.add(matrix[l][colbegin]);
        }
      }
      colbegin++;

    }
    return ans;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the number of rows:");
    int rows = sc.nextInt();
    System.out.println("Enter the number of columns:");
    int cols = sc.nextInt();
    int[][] matrix = new int[rows][cols];
    System.out.println("Enter the matrix elements row by row:");
    for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
        matrix[i][j] = sc.nextInt();
      }
    }
    List<Integer> result = helper(matrix);
    System.out.println("Spiral Order of the matrix: " + result);
```

```
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the number of rows:
3
Enter the number of columns:
3
Enter the matrix elements row by row:
1 2 3
4 5 6
7 8 9
Spiral Order of the matrix: [1, 2, 3, 6, 9, 8, 7, 4, 5]
```

Time Complexity: O(rXc)

Space Complexity: O(rXc)

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

Program:

```
import java.util.*;


public class Main{
    public static boolean helper(String str) {
        Stack<Character> st = new Stack<>();
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == '(' || str.charAt(i) == '{' || str.charAt(i) == '[') {
                st.push(str.charAt(i));
```

```
            } else {
                if (!st.empty() &&
                        ((st.peek() == '(' && str.charAt(i) == ')') ||
                                (st.peek() == '{' && str.charAt(i) == '}') ||
                                (st.peek() == '[' && str.charAt(i) == ']'))) {
                    st.pop();
                } else {
                    return false;
                }
            }
        }
        return st.empty();
    }

    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the String");
        String str=sc.next();
        System.out.println(helper(str) ? "Balanced" : "Unbalanced");
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the String
(((()))()()
Balanced
```

Time Complexity: O(n)

Space 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.

Program:

```
import java.util.*;

public class Main {

   public static boolean helper(String str1, String str2) {

      char[] arr1 = str1.toCharArray();

      char[] arr2 = str2.toCharArray();

      Arrays.sort(arr1);

      Arrays.sort(arr2);

      return Arrays.equals(arr1, arr2);

   }

   public static void main(String[] args) {

      Scanner sc=new Scanner(System.in);
```

```
        System.out.println("Ënter the String 1");
        String s1 = sc.next();
        System.out.println("Ënter the String 2");
        String s2 = sc.next();
        System.out.println(helper(s1, s2));
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Ënter the String 1
allergy
Ënter the String 2
allergic
false
```

Time Complexity: O(nlogn) (due to sorting)

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: “”

Program:

```
import java.util.*;
```

```
public class Main {
    private static String helper(String str, int left, int right) {
        while (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {
            left--;
            right++;
        }
        return str.substring(left + 1, right);
    }
    public static String longestPalindrome(String str) {
        if (str == null || str.length() == 0) {
            return "";
        }
        String longest = "";
        for (int i = 0; i < str.length(); i++) {
            String oddPalindrome = helper(str, i, i);
            if (oddPalindrome.length() > longest.length()) {
                longest = oddPalindrome;
            }
            String evenPalindrome = helper(str, i, i + 1);
            if (evenPalindrome.length() > longest.length()) {
                longest = evenPalindrome;
            }
        }
        return longest;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter a string: ");
        String str = sc.nextLine();
        String result = longestPalindrome(str);
        System.out.println("Longest Palindromic Substring: " + result);
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter a string:  "Geeks"
Longest Palindromic Substring: ee

Process finished with exit code 0
```

Time Complexity: O(n^3)

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.

Program:

```
import java.util.*;
public class Main {
    public static String longestCommonPrefix(String[] arr) {
```

```
        if (arr == null || arr.length == 0) {
            return "-1";
        }
        Arrays.sort(arr);
        String first = arr[0];
        String last = arr[arr.length - 1];
        int minLength = Math.min(first.length(), last.length());
        int i = 0;
        while (i < minLength && first.charAt(i) == last.charAt(i)) {
            i++;
        }
        String commonPrefix = first.substring(0, i);
        return commonPrefix.isEmpty() ? "-1" : commonPrefix;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of strings in the array: ");
        int n = sc.nextInt();
        sc.nextLine();
        String[] arr = new String[n];
        System.out.println("Enter each string:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextLine();
        }
        System.out.println("Longest Common Prefix: " + longestCommonPrefix(arr));
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the number of strings in the array: 4
Enter each string:
geeksforgeeks
geeks
geek
geezer
Longest Common Prefix: gee
```

Time Complexity: O(nlogn+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]

Program:

```
import java.util.*;

public class Main {

    public static void deleteMiddle(Stack<Integer> stack, int currentIndex) {

        if (stack.isEmpty() || currentIndex == 0) {

            stack.pop();

            return;

        }

        int top = stack.pop();

        deleteMiddle(stack, currentIndex - 1);
```

```
        stack.push(top);
    }
    public static void deleteMiddleElement(Stack<Integer> stack) {
        if (stack.isEmpty()) return;
        int middleIndex = stack.size() / 2;
        deleteMiddle(stack, middleIndex);
    }



    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the Stack");
        int n = sc.nextInt();
        System.out.println("Enter the Elements of the Stack");
        Stack<Integer> st = new Stack<>();
        for(int i=0;i<n;i++) {
            st.push(sc.nextInt());
        }
        System.out.println("Original Stack: " + st);
        deleteMiddleElement(st);
        System.out.println("Stack after deleting middle element: " + st);
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the size of the Stack
5
Enter the Elements of the Stack
1 2 3 4 5
Original Stack: [1, 2, 3, 4, 5]
Stack after deleting middle element: [1, 2, 4, 5]
```

Time Complexity: O(n)

Space Complexity: O(n) (recursive stack space)

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

Program:

```
import java.util.*;

public class Main {

    public static void printNextGreaterElements(int[] arr) {

        int n = arr.length;

        int[] nge = new int[n];

        Stack<Integer> stack = new Stack<>();

        for (int i = n - 1; i >= 0; i--) {
```

```
        while (!stack.isEmpty() && stack.peek() <= arr[i]) {
            stack.pop();
        }
        nge[i] = stack.isEmpty() ? -1 : stack.peek();
        stack.push(arr[i]);
    }
    for (int i = 0; i < n; i++) {
        System.out.println(arr[i] + " --> " + nge[i]);
    }
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int[] arr1 = new int[n];
    System.out.println("Enter the Elements");
    for(int i=0;i<n;i++){
        arr1[i] = sc.nextInt();
    }
    System.out.println("Next Greater Elements for arr1:");
    printNextGreaterElements(arr1);
  }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
4
Enter the Elements
4 5 2 25
Next Greater Elements for arr1:
4 --> 5
5 --> 25
2 --> 25
25 --> -1
```

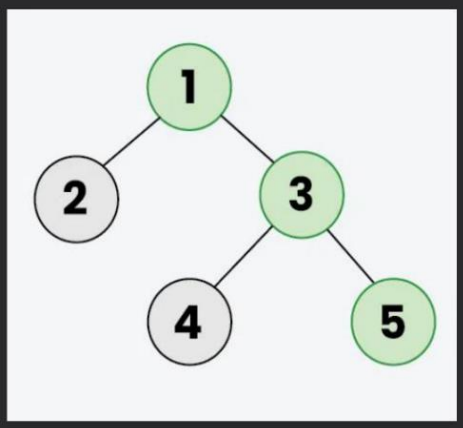
Time Complexity: O(n^2)

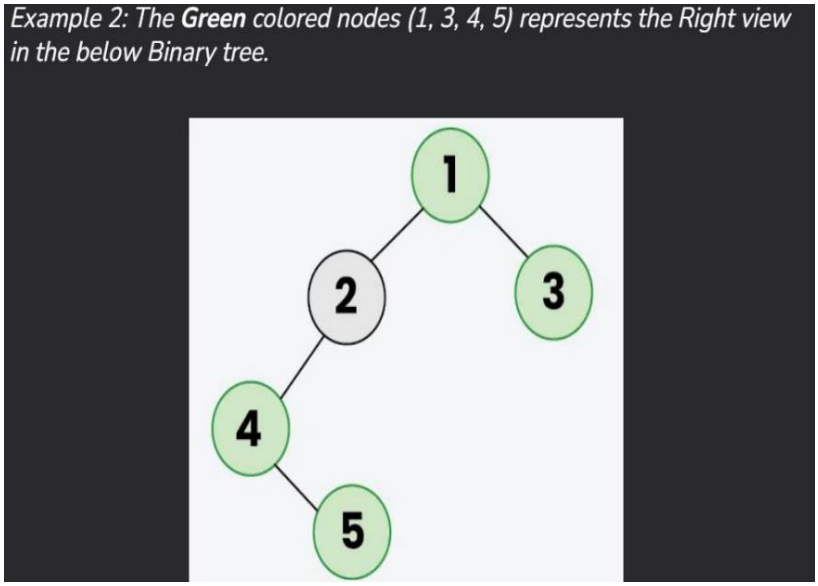
Space Complexity: O(1)

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.

*Example 1: The **Green** colored nodes (1, 3, 5) represents the Right view in the below Binary tree.*





Program:

```
import java.util.*;
class TreeNode {
   int val;
   TreeNode left, right;

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

public class Main {
   public static void helper(TreeNode root, int level,List<Integer> list){
      if(root == null){
         return ;
```

```
        }
        if(list.size() == level){
            list.add(root.val);
        }
        helper(root.right,level+1,list);
        helper(root.left,level+1,list);
    }
    public static List<Integer> rightSideView(TreeNode root) {
        int level = 0;
        List<Integer> list = new LinkedList<>();
        helper(root,level,list);
        return list;
    }

    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);
        List<Integer> ans = rightSideView(root);
        System.out.println("Right view of the binary tree:");
        for(int num:ans){
            System.out.print(num+" ");
        }
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
Right view of the binary tree:
1 3 4
Process finished with exit code 0
```

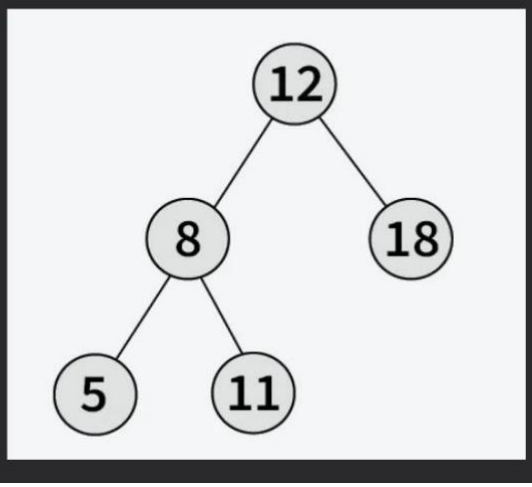
Time Complexity: O(n)

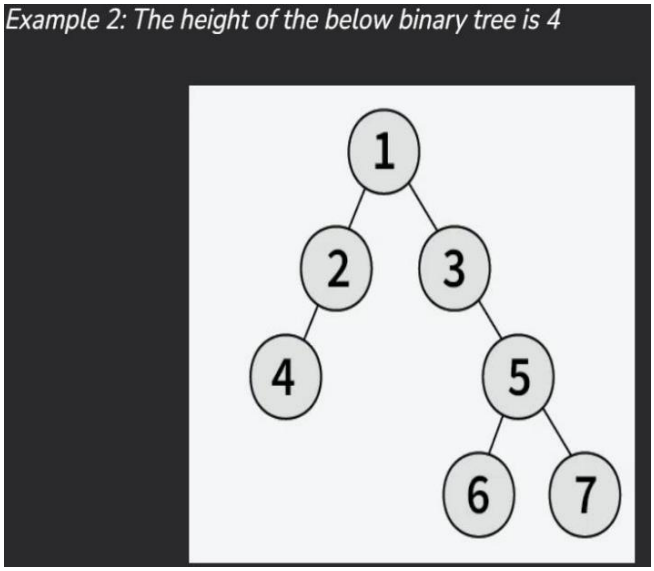
Space Complexity: O(h) (h- height of the tree)

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.

Example 1: The height of the below binary tree is 3.





Program:

```
import java.util.*;
class TreeNode {
   int val;
   TreeNode left, right;

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

public class Main {
   public static int maxDepth(TreeNode root) {
      if(root==null){
         return 0;
      }
      int l=maxDepth(root.left);
```

```
        int r=maxDepth(root.right);


        return 1+Math.max(l,r);
    }


    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);


        System.out.println("Right view of the binary tree:"+maxDepth(root));
    }
}
```

Output:

```
C:\Users\thara\.jdks\openjdk-23\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
The height of the binary tree:3

Process finished with exit code 0
```

Time Complexity: O(n)

Space Complexity: O(h)