Practice_Set_1

Name: Sanjay S, Roll No:22IT093

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

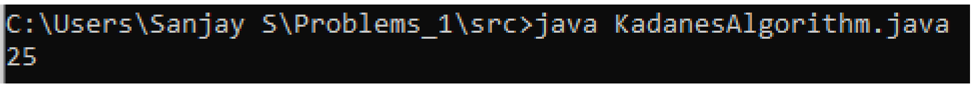
Explanation: The subarray {-2} has the largest sum -2.

Input: arr[] = {5, 4, 1, 7, 8}

Output: 25

Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

```
public class KadanesAlgorithm {
        public static void main(String args[]) {
                int[] nums= {5, 4, 1, 7, 8};
                int n=nums.length;
                int max=nums[0];
                int currSum=nums[0];
                for(int i=1;i<n;i++) {
                        currSum=Math.max(currSum+nums[i], nums[i]);
                        max=Math.max(max, currSum);
                }
                System.out.println(max);
        }
}
```



```
C:\Users\Sanjay S\Problems_1\src>java KadanesAlgorithm.java
25
```

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

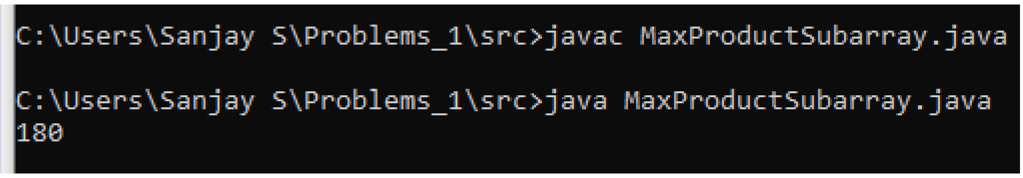
Input: arr[] = {-1, -3, -10, 0, 60}

Output: 60

Explanation: The subarray with maximum product is {60}.

```
public class MaxProductSubarray {
          public static void main(String args[]) {
                    int[] nums= {-2, 6, -3, -10, 0, 2};
                    int n=nums.length;
      int max = Integer.MIN_VALUE;
      int leftProd = 1;
      int rightProd = 1;
      for (int i = 0; i < n; i++) {
         if (leftProd == 0) {
            leftProd = 1;
         }
         leftProd = leftProd * nums[i];
         if (rightProd == 0) {
            rightProd = 1;
         }
         rightProd = rightProd * nums[n - 1 - i];
         max = Math.max(max, Math.max(leftProd, rightProd));
      }
      System.out.println(max);
```

```
        }
}
```



```
C:\Users\Sanjay S\Problems_1\src>javac MaxProductSubarray.java

C:\Users\Sanjay S\Problems_1\src>java MaxProductSubarray.java
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

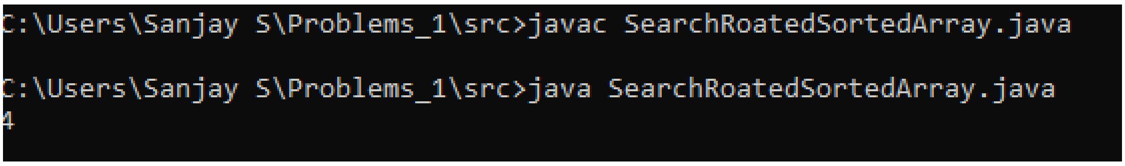
Output : -1

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

Output : 1

```
public class SearchRoatedSortedArray {

        public static void main(String args[]) {

                int[] arr= {4, 5, 6, 7, 0, 1, 2};

                int n=arr.length;

                int key=0;

                int left=0;

                int right=n-1;

                int ans=-1;

                while(left<=right) {

                        int mid=(left+right)/2;

                        if(arr[mid]==key) {

                                ans= mid;

                                break;

                        }
```

```
            if(arr[left]<=arr[mid]) {
                if(key>=arr[left] && key<arr[mid]) {
                    right=mid-1;
                }else {
                    left=mid+1;
                }
            }else {
                if(key>arr[mid] && key<=arr[right]) {
                    left=mid+1;
                }else {
                    right=mid-1;
                }
            }
        }
        System.out.println(ans);
    }
}
```



```
C:\Users\Sanjay S\Problems_1\src>javac SearchRoatedSortedArray.java

C:\Users\Sanjay S\Problems_1\src>java SearchRoatedSortedArray.java
4
```

Time complexity : O(log 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

```
public class ContainerWithMostWater {
    public static void main(String args[]) {
        int[] height = {1, 5, 4, 3};
  int n = height.length;
  int i = 0;
  int j = n - 1;
```

```
        int max = 0;
        while (i < j) {
            int area = Math.min(height[i], height[j]) * (j - i);
            if (height[i] <= height[j]) {
                i++;
            } else {
                j--;
            }
            max = Math.max(area, max);
        }
       System.out.println(max);;
    }
}
```

Time complexity : O(n)

Space complexity : O(1)

5. Find the Factorial of a large number

Input: 100

Output:

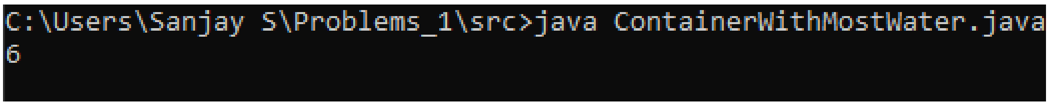
9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828625369792082722375825118521091686400000000000000000000000000

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

```
import java.math.BigInteger;
public class FactorialLargeNumbers {
             public static void main(String args[]) {
                          int num=100;
                          BigInteger val=BigInteger.ONE;
```

```
		for(int i=1;i<=num;i++) {
			val=val.multiply(BigInteger.valueOf(i));
		}
		System.out.println(val);
	}
}
```



```
C:\Users\Sanjay S\Problems_1\src>java ContainerWithMostWater.java
6
```

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

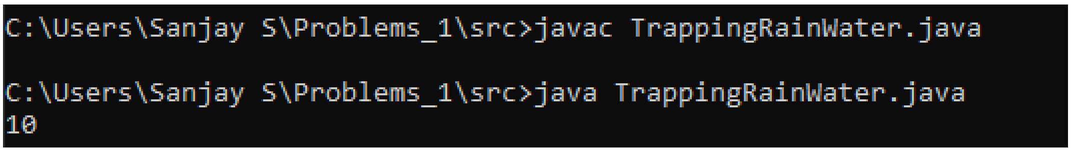
Input: arr[] = {10, 9, 0, 5}

Output: 5

Explanation : We trap 0 + 0 + 5 + 0 = 5

```
public class TrappingRainWater {
	public static void main(String[] args) {
		int[] height= {3, 0, 1, 0, 4, 0, 2};
		int n = height.length;
```

```
        int[] prefixMax = new int[n];// maximum height from left
        int[] suffixMax = new int[n];// maximum height from right
        prefixMax[0] = height[0];
        for (int i = 1; i < n; i++) {
            prefixMax[i] = Math.max(prefixMax[i - 1], height[i]);
        }
        suffixMax[n - 1] = height[n - 1];
        for (int i = n - 2; i >= 0; i--) {
            suffixMax[i] = Math.max(suffixMax[i + 1], height[i]);
        }
        int tot = 0;
        for (int i = 0; i < n; i++) {
            int leftMax = prefixMax[i];
            int rightMax = suffixMax[i];
            tot += Math.min(leftMax, rightMax) - height[i];
        }
        System.out.println(tot);
             }
}
```



```
C:\Users\Sanjay S\Problems_1\src>javac TrappingRainWater.java

C:\Users\Sanjay S\Problems_1\src>java TrappingRainWater.java
10
```

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

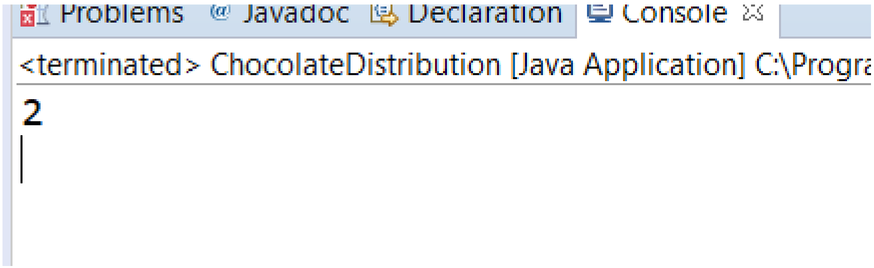
Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference, that is 2.

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

```
import java.util.Arrays;

public class ChocolateDistribution {

    public static int findMinDifference(int arr[], int n, int m) {
        if (m == 0 || n == 0) {
            return 0;
        }

        Arrays.sort(arr);

        if (n < m) {
            return -1;
        }

        int min = Integer.MAX_VALUE;

        for (int i = 0; i + m - 1 < n; i++) {
            int diff = arr[i + m - 1] - arr[i];
            min = Math.min(min, diff);
        }

        return min;
    }
    public static void main(String[] args) {
        int arr[] = {7, 3, 2, 4, 9, 12, 56};
        int m = 3;
        int n = arr.length;

        System.out.println( + findMinDifference(arr, n,m));

    }
}
```



```
<terminated> ChocolateDistribution [Java Application] C:\Progra
2
```

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]]

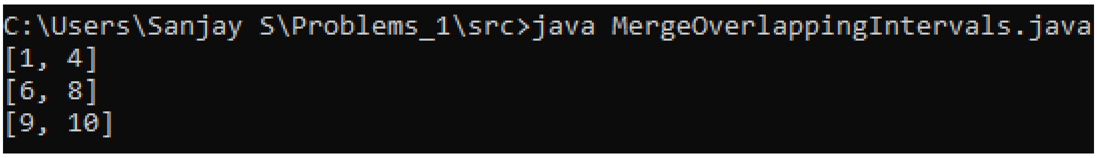
Explanation: In the given intervals, we have only two overlapping intervals [1, 3] and [2, 4]. Therefore, we will merge these two and return [[1, 4}], [6, 8], [9, 10]].

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 MergeOverlappingIntervals {


        public static void main(String[] args) {
                int[][] arr= {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
                int n = arr.length;
     Arrays.sort(arr, (a, b) -> a[0] == b[0] ? a[1] - b[1] : a[0] - b[0]);
     List<int[]> merged = new ArrayList<>();
     for (int i = 0; i < n; i++) {
       if (!merged.isEmpty() && merged.get(merged.size() - 1)[1] >= arr[i][0]) {
         int[] intr = merged.get(merged.size() - 1);
         merged.remove(merged.size() - 1);
         merged.add(new int[] { intr[0], Math.max(intr[1], arr[i][1]) });
       } else {
         merged.add(arr[i]);
       }
     }

     int[][] res = new int[merged.size()][2];
     for (int i = 0; i < merged.size(); i++) {
       res[i] = merged.get(i);
     }
     for (int[] interval : res) {
       System.out.println(Arrays.toString(interval));
     }
        }


}
```



```
C:\Users\Sanjay S\Problems_1\src>java MergeOverlappingIntervals.java
[1, 4]
[6, 8]
[9, 10]
```

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},

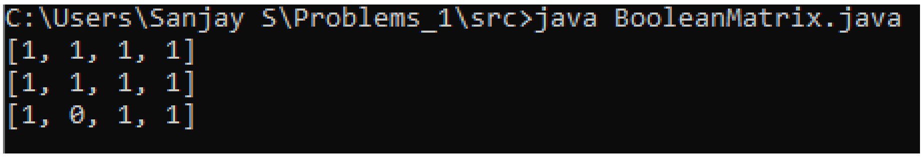
{0, 0, 0, 0}}

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

{1, 1, 1, 1},

{1, 0, 1, 1}}

```
import java.util.Arrays;


public class BooleanMatrix {
        public static void main(String args[]){
                int[][] matrix={{1, 0, 0, 1},
                                 {0, 0, 1, 0},
                                 {0, 0, 0, 0}};


                int row=matrix.length;
     int col=matrix[0].length;
    boolean[] rowFlag=new boolean[row];
    boolean[] colFlag=new boolean[col];
    for(int i=0;i<row;i++){
            for(int j=0;j<col;j++){
                if(matrix[i][j]==1){
                    rowFlag[i]=true;
                    colFlag[j]=true;
                }
            }
        }
        for(int i=0;i<row;i++){
            for(int j=0;j<col;j++){
                if(rowFlag[i] || colFlag[j]){
                    matrix[i][j]=1;
                }
            }
        }
        for(int[] entry:matrix) {
                System.out.println(Arrays.toString(entry));
        }
              }
}
```



```
C:\Users\Sanjay S\Problems_1\src>java BooleanMatrix.java
[1, 1, 1, 1]
[1, 1, 1, 1]
[1, 0, 1, 1]
```

*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},

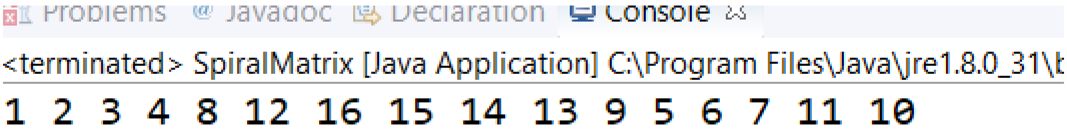
{7, 8, 9, 10, 11, 12},

{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.

```
public class SpiralMatrix {


    public static void printSpiral(int[][] matrix) {
        if (matrix.length == 0) {
            return;
        }


        int top = 0;
        int bottom = matrix.length - 1;
        int left = 0;
        int right = matrix[0].length - 1;


        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) {
                System.out.print(matrix[top][i] + " ");
            }
            top++;


            for (int i = top; i <= bottom; i++) {
                System.out.print(matrix[i][right] + " ");
            }
            right--;


            if (top <= bottom) {
                for (int i = right; i >= left; i--) {
                    System.out.print(matrix[bottom][i] + " ");
                }
                bottom--;
            }

            if (left <= right) {
                for (int i = bottom; i >= top; i--) {
                    System.out.print(matrix[i][left] + " ");
                }
                left++;
            }
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int[][] matrix1 = {
            {1, 2, 3, 4},
            {5, 6, 7, 8},
            {9, 10, 11, 12},
            {13, 14, 15, 16}
        };

        printSpiral(matrix1);

        System.out.println("Spiral order of matrix2:");
        printSpiral(matrix2);
    }
}
```



```
<terminated> SpiralMatrix [Java Application] C:\Program Files\Java\jre1.8.0_31\b
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

Time complexity : O(n)

Space complexity : O(1)

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.

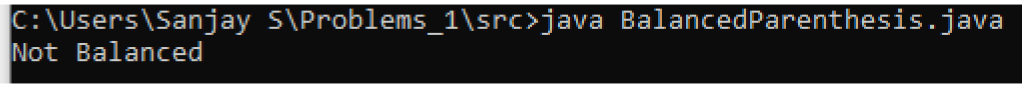
Input: str = “((()))()()”

Output: Balanced

Input: str = “())((())”

Output: Not Balanced

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

        public static void main(String[] args) {
                String str="())((())";
                Stack<Character> st = new Stack<>();
    for (char c : str.toCharArray()) {
      if (!st.isEmpty() && c == ')' && st.peek() == '(') {
        st.pop();
      } else {
        st.push(c);
      }
    }
    System.out.println(st.isEmpty()?"Balanced":"Not Balanced");

        }

}
```



```
C:\Users\Sanjay S\Problems_1\src>java BalancedParenthesis.java
Not 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.

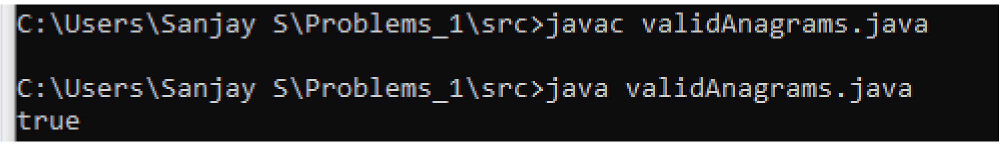
Input: s1 = “g”, s2 = “g”

Output: true

Explanation: Characters in both the strings are same, so they are anagrams

```
public class validAnagrams {
        public static boolean areAnagrams(String s1, String s2) {

            // Your code here
            int[] freq=new int[26];
            for(char c:s1.toCharArray()){
                freq[c-'a']++;
            }
            for(char c:s2.toCharArray()){
                freq[c-'a']--;
            }
            for(int i=0;i<26;i++){
                if(freq[i]!=0){
                    return false;
                }
            }
```

```
                return true;
            }
        public static void main(String args[]) {
                String s1="geeks";
                String s2="kseeg";
                System.out.println(areAnagrams(s1,s2));
        }
}
```



```
C:\Users\Sanjay S\Problems_1\src>javac validAnagrams.java

C:\Users\Sanjay S\Problems_1\src>java validAnagrams.java
true
```

Time complexity : O(n + m)

Space complexity : O(1)

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”

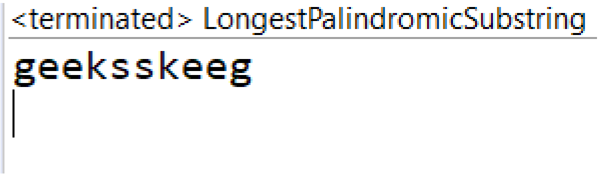
Output: “a”

Input: str = “”

Output: “”

```
public class LongestPalindromicSubstring {

    public static String expandAroundCenter(String str, int left, int right) {
        while (left >= 0 && right < str.length() && str.charAt(left) == str.charAt(right)) {
```

```java
            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 = expandAroundCenter(str, i, i);
            String evenPalindrome = expandAroundCenter(str, i, i + 1);
            if (oddPalindrome.length() > longest.length()) {
                longest = oddPalindrome;
            }
            if (evenPalindrome.length() > longest.length()) {
                longest = evenPalindrome;
            }
        }

        return longest;
    }

    public static void main(String[] args) {
        String str = "forgeeksskeegfor";
        System.out.println(longestPalindrome(str));
    }
}
```



```
<terminated> LongestPalindromicSubstring
geeksskeeg
```

Time complexity : O(n ^2)

Space complexity : O(1)

16. Longest Common Prefix using Sorting

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

Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”]

Output: gee

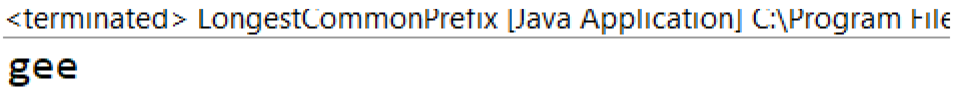
Explanation: “gee” is the longest common prefix in all the given strings.

Input: arr[] = [“hello”, “world”]

Output: -1

Explanation: There‟s no common prefix in the given strings

```
import java.util.Arrays;

public class LongestCommonPrefix {

    public static String longestCommonPrefix(String[] arr) {
        if (arr.length == 0) {
            return "-1";
        }

        Arrays.sort(arr);

        String start = arr[0];
        String end = arr[arr.length - 1];

        int min = Math.min(start.length(), end.length());
        int i = 0;

        while (i < min && start.charAt(i) == end.charAt(i)) {
            i++;
        }

        if (i == 0) {
            return "-1";
        }

        return start.substring(0, i);
    }

    public static void main(String[] args) {
        String[] arr = {"geeksforgeeks", "geeks", "geek", "geezer"};
        System.out.println(longestCommonPrefix(arr));
    }
}
```



```
<terminated> LongestCommonPrefix [Java Application] C:\Program Fil
gee
```

*Time complexity O(n * m * log(n))*

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.

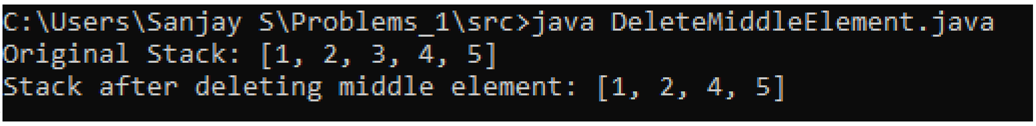
Input : Stack[] = [1, 2, 3, 4, 5]

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

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

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

```
import java.util.Stack;


public class DeleteMiddleElement {
    public static void deleteMid(Stack<Integer> s, int sizeOfStack) {
        Stack<Integer> tempStack = new Stack<>();
        int mid = sizeOfStack % 2 == 0 ? sizeOfStack / 2 : sizeOfStack / 2 + 1;
        while (s.size() > mid) {
            tempStack.push(s.pop());
        }
        s.pop();
        while (!tempStack.isEmpty()) {
            s.push(tempStack.pop());
        }
    }


    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        System.out.println("Original Stack: " + stack);
        deleteMid(stack, stack.size());
        System.out.println("Stack after deleting middle element: " + stack);
    }
}
```



```
C:\Users\Sanjay S\Problems_1\src>java DeleteMiddleElement.java
Original Stack: [1, 2, 3, 4, 5]
Stack after deleting middle element: [1, 2, 4, 5]
```

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

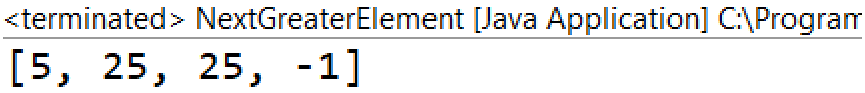
7 –> 12

6 –> 12

12 –> -1

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

```
import java.util.ArrayList;

import java.util.Stack;


class NextGreaterElement {

    public static ArrayList<Integer> nextLargerElement(int[] arr) {

        int n = arr.length;

        ArrayList<Integer> ans = new ArrayList<>();

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

            ans.add(-1);

        }

        Stack<Integer> st = new Stack<>();
        for (int i = n - 1; i >= 0; i--) {
            while (!st.isEmpty() && st.peek() <= arr[i]) {
                st.pop();
            }
            if (!st.isEmpty()) {
                ans.set(i, st.peek());
            }
            st.push(arr[i]);
        }
        return ans;
    }

    public static void main(String[] args) {

        int[] arr = {4 , 5 , 2 , 25};

        ArrayList<Integer> result =nextLargerElement(arr);

        System.out.println(result);
    }
}
```

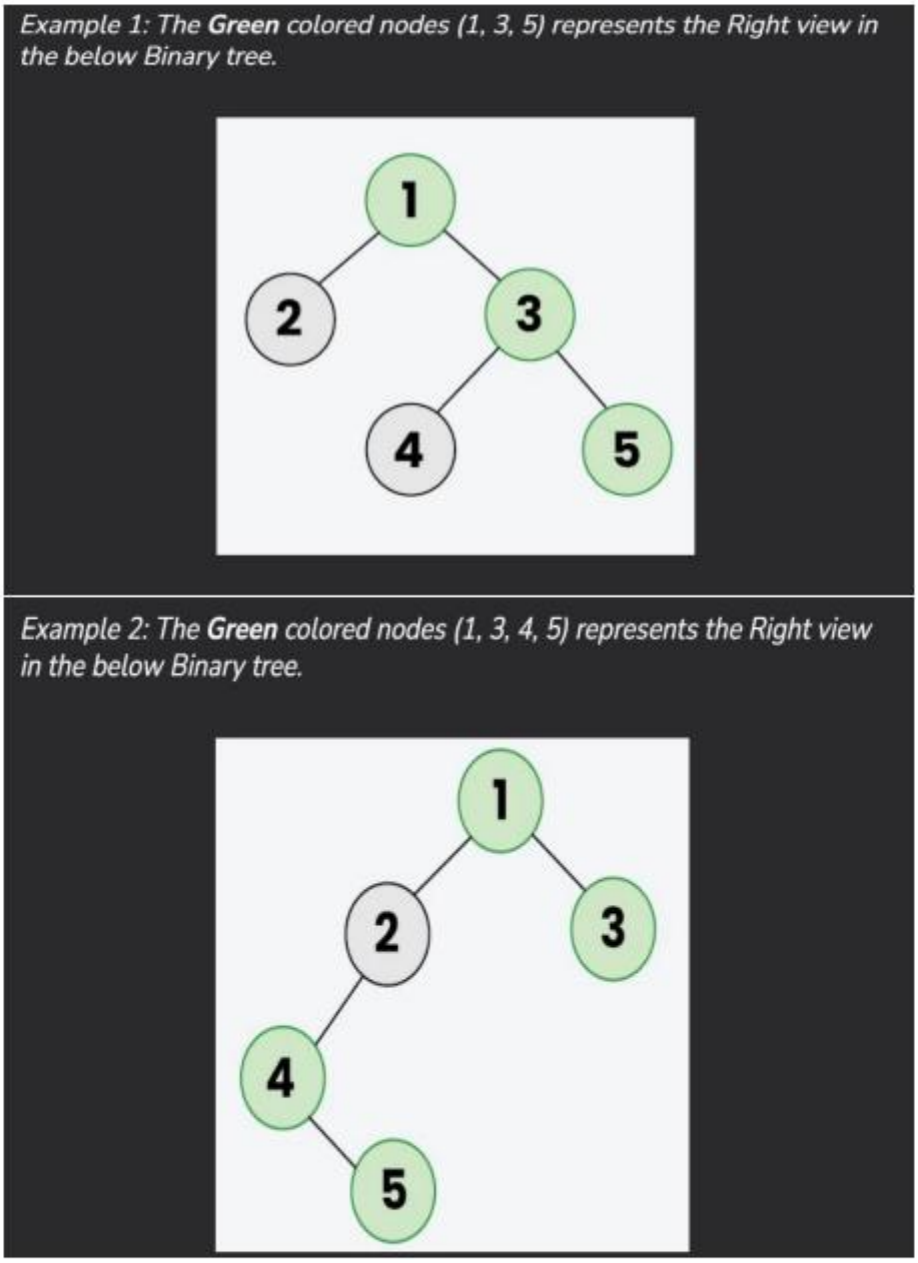


```
<terminated> NextGreaterElement [Java Application] C:\Progran
[5, 25, 25, -1]
```

Time complexity O(n)

Space complexity : O(n)

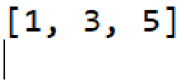
19.Print Right View of a Binary TreeGiven a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level



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

Example 2: The ***Green*** *colored nodes (1, 3, 4, 5) represents the Right view in the below Binary tree.*

```
import java.util.*;


class TreeNode {
    int val;
    TreeNode left, right;


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

public class RightView {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> rightView = new ArrayList<>();
        Queue<TreeNode> queue = new LinkedList<>();
        if (root == null)
            return rightView;
        queue.add(root);
        while (!queue.isEmpty()) {
            int size = queue.size();
            TreeNode rightNode = null;
            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                rightNode = node;
                if (node.left != null)
                    queue.add(node.left);
                if (node.right != null)
                    queue.add(node.right);
            }
            rightView.add(rightNode.val);
        }
        return rightView;
    }

    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);


        RightView solution = new RightView();
        List<Integer> rightView = solution.rightSideView(root);


        System.out.println(rightView);
    }
}
```



```
[1, 3, 5]
```

Time complexity O(n)

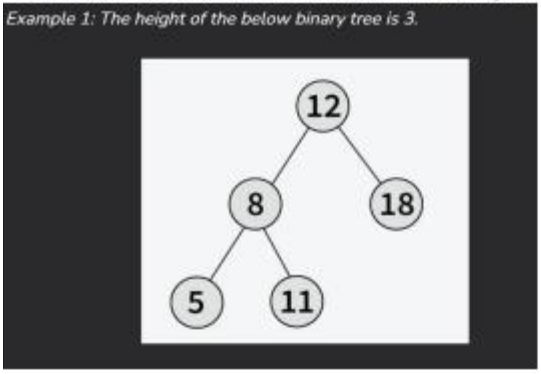
Space complexity : O(w)

20. Maximum Depth or Height of Binary Tree

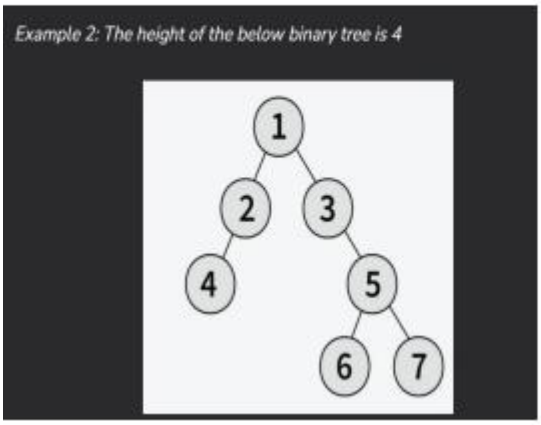
Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the

tree is the number of vertices in the tree from the root to the deepest node

[illegible]

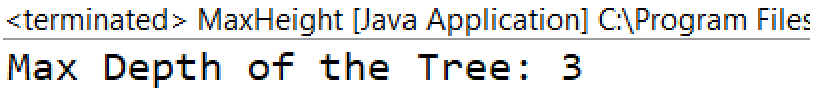


Example 1: The height of the below binary tree is 3.



Example 2: The height of the below binary tree is 4

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

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

public class MaxHeight {
    public int maxDepth(TreeNode root) {
        return findHeight(root);
    }

    public int findHeight(TreeNode node) {
        if (node == null) {
            return 0;
        }
        int left = findHeight(node.left);
        int right = findHeight(node.right);
        return Math.max(left, right) + 1;
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(12);
        root.left = new TreeNode(8);
        root.right = new TreeNode(18);
        root.left.left = new TreeNode(5);
        root.left.right = new TreeNode(11);
        MaxHeight solution = new MaxHeight();
        int maxDepth = solution.maxDepth(root);

        System.out.println("Max Depth of the Tree: " + maxDepth);
    }
}
```



```
<terminated> MaxHeight [Java Application] C:\Program Files
Max Depth of the Tree: 3
```

Time complexity O(n)

Space complexity : O(log n)