CODING ASSIGNMENT

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.

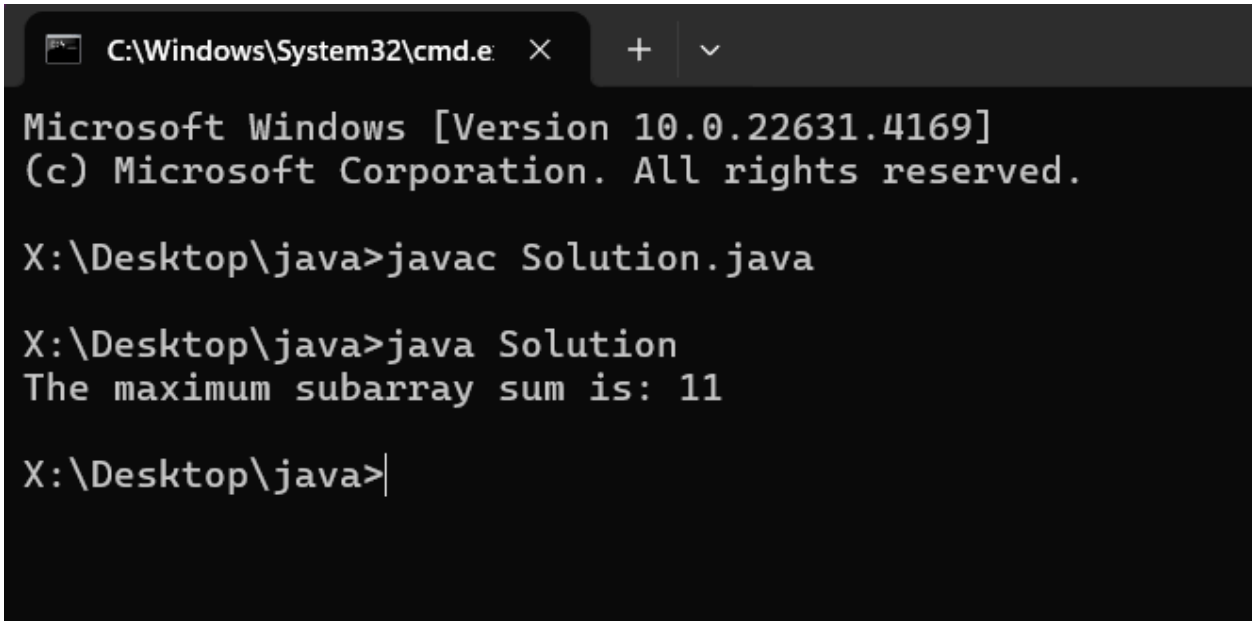
Input: arr[] = {5, 4, 1, 7, 8}
Output: 25
Explanation: The subarray {5, 4, 1, 7, 8} has the largest sum 25.

CODE:

```
public class Solution {
    public int maxSubArray(int[] nums) {
        int curSum = 0;
        int maxSum = Integer.MIN_VALUE;
        for (int i = 0; i < nums.length; i++) {
            int temp = curSum + nums[i];
            if (temp < nums[i]) {
                curSum = nums[i];
            } else {
                curSum = temp;
            }
            if (maxSum < curSum) {
                maxSum = curSum;
            }
        }
        return maxSum;
    }
    public static void main(String[] args) {
        int[] arr = {2, 3, -8, 7, -1, 2, 3};
        int result = new Solution().maxSubArray(arr);
```

```
        System.out.println("The maximum subarray sum is: " + result);
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(n)

2.Maximum Product Subarray

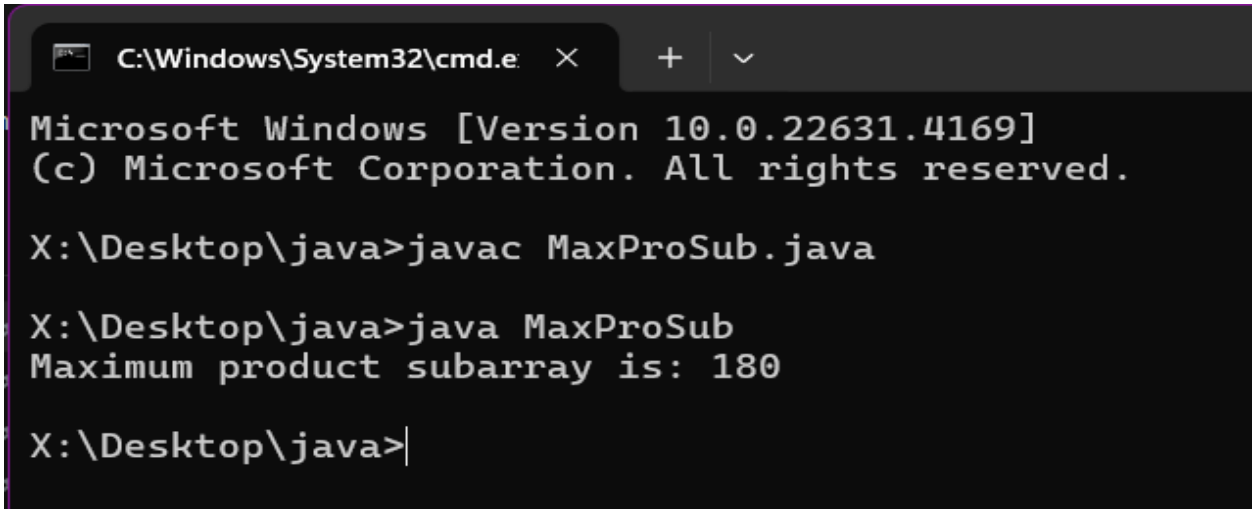
Given an integer array, the task is to find the maximum product of any subarray.
Input: arr[] = {-2, 6, -3, -10, 0, 2}
Output: 180
Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 * (-3) * (-10) = 180

CODE:

```
class MaxProSub{
    long maxProduct(int[] arr, int n) {
```

```
        long pref = 1;
        long suf = 1;
        long ans = Integer.MIN_VALUE;
        for (int i = 0; i < n; i++) {
            if (pref == 0)
                pref = 1;
            if (suf == 0)
                suf = 1;
            pref = pref * arr[i];
            suf = suf * arr[n - i - 1];
            ans = Math.max(ans, Math.max(pref, suf));
        }
        return ans;
    }
    public static void main(String[] args) {
        MaxProSub sol = new MaxProSub();
        int[] arr = {-2, 6, -3, -10, 0, 2};
        int n = arr.length;
        long result = sol.maxProduct(arr, n);
        System.out.println("Maximum product subarray is: " + result);
    }
}
```

OUTPUT:



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 the 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

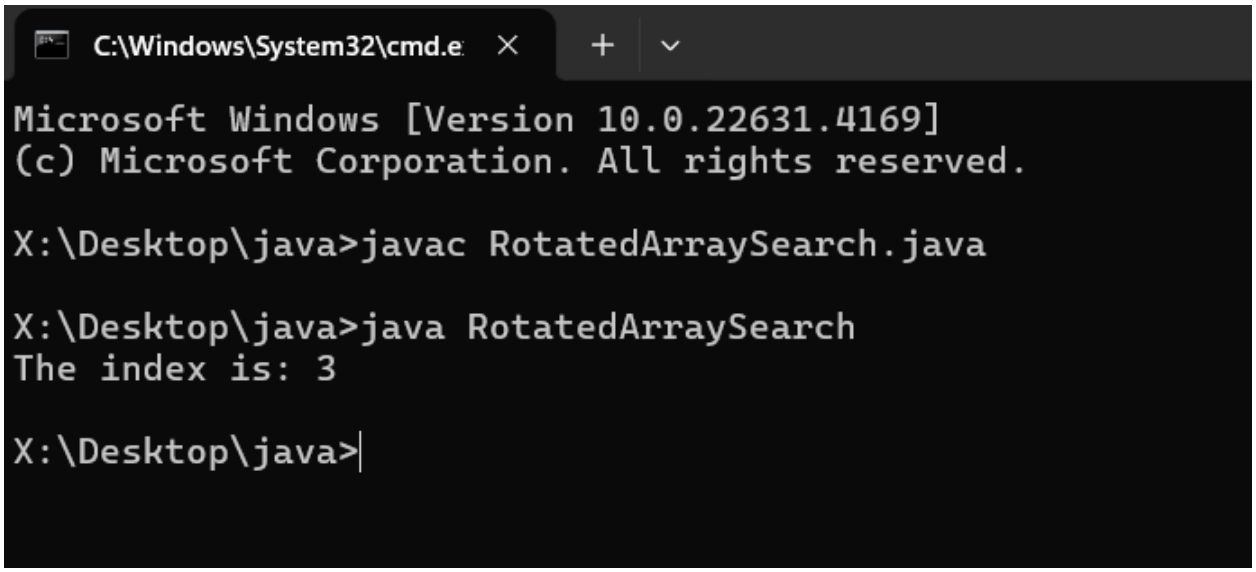
Input : arr[] = {50, 10, 20, 30, 40}, key = 10
Output : 1

CODE:

```
import java.util.*;
public class RotatedArraySearch {
    public static int search(ArrayList<Integer> arr, int size, int target) {
        int start = 0, end = size - 1;
        while (start <= end) {
            int middle = (start + end) / 2;
            if (arr.get(middle) == target)
                return middle;
            if (arr.get(start) <= arr.get(middle)) {
                if (arr.get(start) <= target && target <= arr.get(middle)) {
                    end = middle - 1;
                } else {
                    start = middle + 1;
                }
            } else {
                if (arr.get(middle) <= target && target <= arr.get(end)) {
                    start = middle + 1;
                } else {
                    end = middle - 1;
                }
            }
        }
        return -1;
    }
```

```
    public static void main(String[] args) {
        ArrayList<Integer> arr = new ArrayList<>(Arrays.asList(7, 8, 9, 1, 2, 3, 4, 5, 6));
        int size = arr.size();
        int target = 1;
        int result = search(arr, size, target);

        if (result == -1)
            System.out.println("Target is not present.");
        else
            System.out.println("The index is: " + result);
    }
}
```

OUTPUT:



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

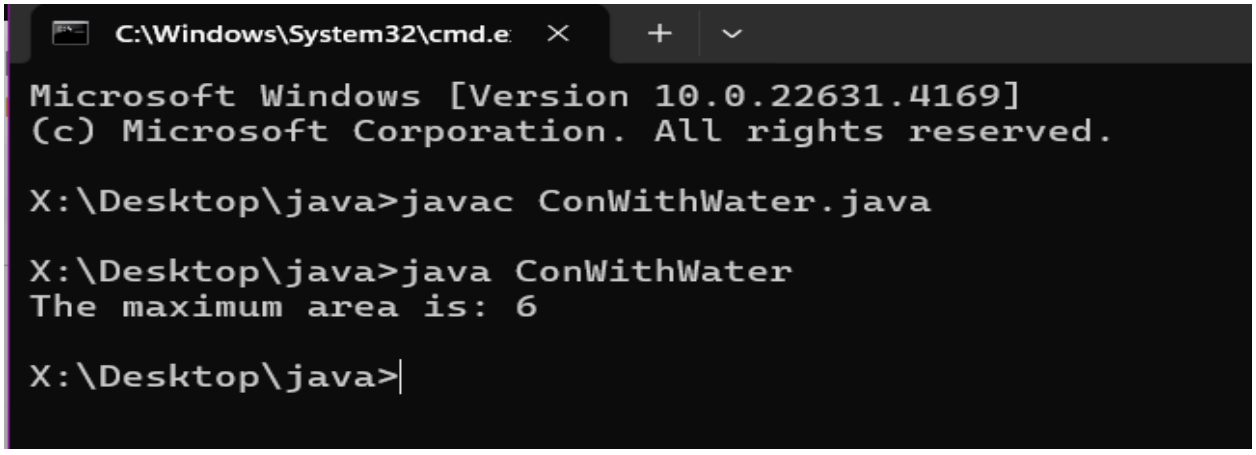
Input: arr = [3, 1, 2, 4, 5]
Output: 12
Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4.
Height of container = min(5, 3) = 3.
So total area = 4 * 3 = 12

CODE:

```
class ConWithWater {
    public int maxArea(int[] height) {
        int left = 0;
        int right = height.length - 1;
        int max = -1;
        while (left < right) {
```

```
            int w = right - left;
            int h = Math.min(height[left], height[right]);
            max = Math.max(max, w * h);
            if (height[left] < height[right])
                left++;
            else
                right--;
        }
        return max;
    }

    public static void main(String[] args) {
        int[] arr = {1, 5, 4, 3};

        ConWithWater c = new ConWithWater();
        int result = c.maxArea(arr);

        System.out.println("The maximum area is: " + result);
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(n)

5.Find the Factorial of a large number

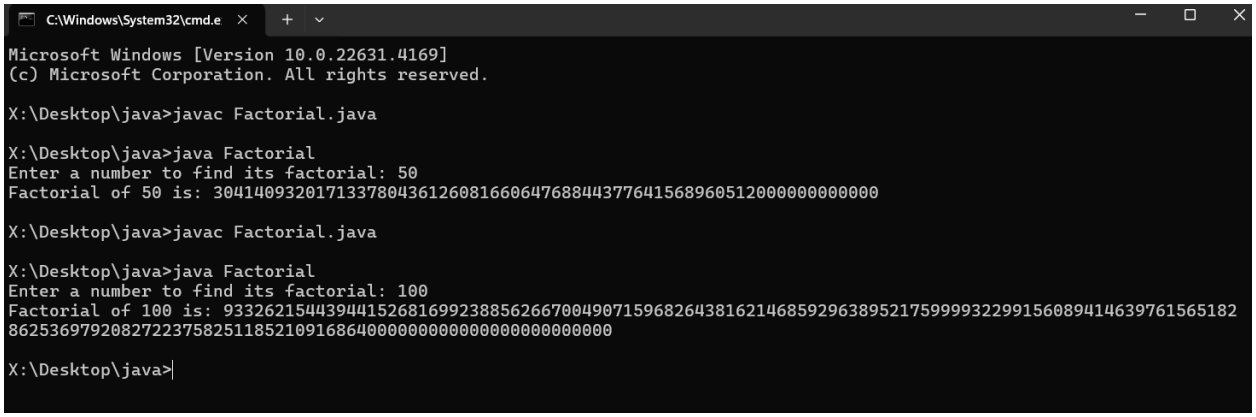
Input: 100
Output:
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000 00

Input: 50
Output:
30414093201713378043612608166064768844377641568960512000000000000

CODE:

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

class Factorial{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number to find its factorial: ");
        int number = sc.nextInt();
        BigInteger result = BigInteger.ONE;
        for (int i = 1; i <= number; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }
        System.out.println("Factorial of " + number + " is: " + result);
        sc.close();
    }
}
```

OUTPUT:



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.

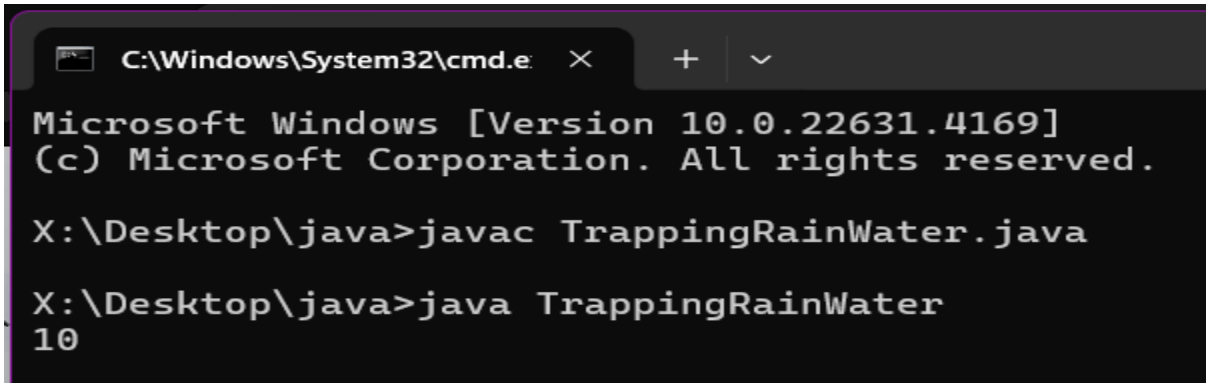
Input: arr[] = {1, 2, 3, 4}
Output: 0
Explanation : We cannot trap water as there is no height bound on both sides

CODE:

public class TrappingRainWater {

```
    public static int trap(int[] arr) {
        int n = arr.length;
        if (n == 0) return 0;
        int[] leftMax = new int[n];
        int[] rightMax = new int[n];
        int waterTrapped = 0;
        leftMax[0] = arr[0];
        for (int i = 1; i < n; i++) {
            leftMax[i] = Math.max(leftMax[i - 1], arr[i]);
        }
        rightMax[n - 1] = arr[n - 1];
        for (int i = n - 2; i >= 0; i--) {
            rightMax[i] = Math.max(rightMax[i + 1], arr[i]);
        }
        for (int i = 0; i < n; i++) {
            waterTrapped += Math.min(leftMax[i], rightMax[i]) - arr[i];
        }
        return waterTrapped;
    }
    public static void main(String[] args) {
        int[] arr = {3, 0, 1, 0, 4, 0, 2};
        System.out.println(trap(arr));
    }
}
```

OUTPUT:



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.

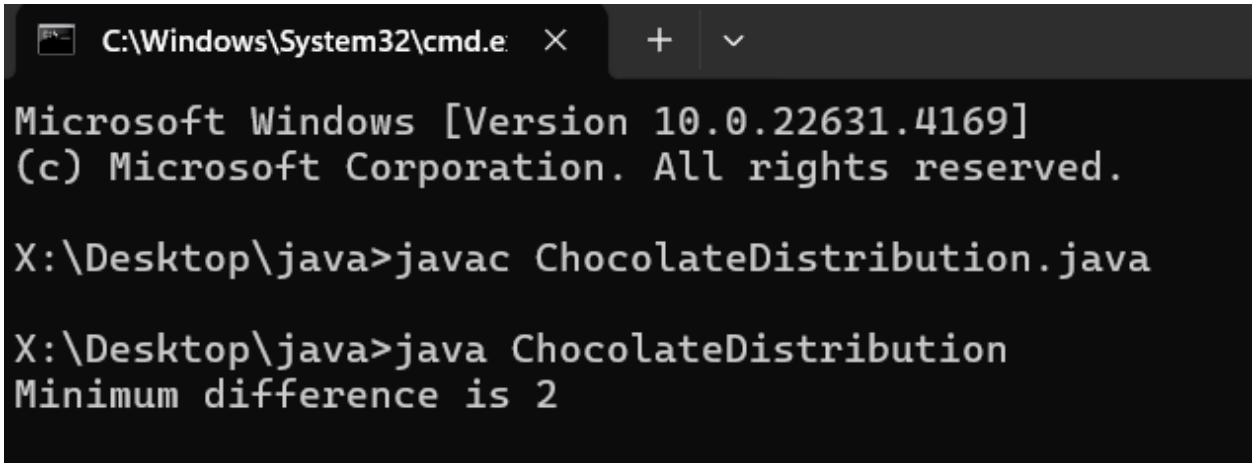
Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5
Output: 7
Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is 9 – 2 = 7.

CODE:

```
import java.util.Arrays;
public class ChocolateDistribution {
   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_diff = Integer.MAX_VALUE;
      for (int i = 0; i + m - 1 < n; i++) {
         int diff = arr[i + m - 1] - arr[i];
         min_diff = Math.min(min_diff, diff);
      }
      return min_diff;
   }
   public static void main(String[] args) {
      int arr[] = {7, 3, 2, 4, 9, 12, 56};
```

```
        int m = 3;
        int n = arr.length;
        int result = findMinDifference(arr, n, m);
        System.out.println("Minimum difference is " + result);
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(nlogn)

---

8. Merge Overlapping Intervals

Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

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

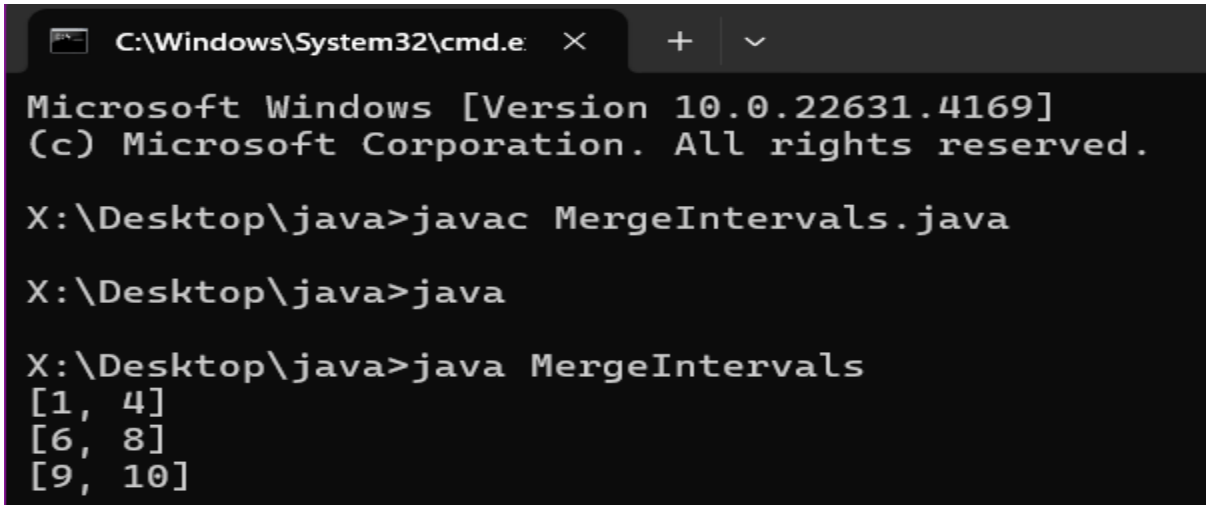
Input: arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]]
Output: [[1, 6], [7, 8]]

Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval [1, 6].

CODE:

```
import java.util.Arrays;
import java.util.ArrayList;
public class MergeIntervals {
    public static int[][] mergeIntervals(int[][] intervals) {
        if (intervals.length <= 1) {
            return intervals;
        }
        Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
        ArrayList<int[]> merged = new ArrayList<>();
        merged.add(intervals[0]);
        for (int i = 1; i < intervals.length; i++) {
            int[] last = merged.get(merged.size() - 1);
            if (last[1] >= intervals[i][0]) {
                last[1] = Math.max(last[1], intervals[i][1]);
            } else {
                merged.add(intervals[i]);
            }
        }

        return merged.toArray(new int[merged.size()][]);
    }
    public static void main(String[] args) {
        int[][] intervals = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};
        int[][] result = mergeIntervals(intervals);
        for (int[] interval : result) {
            System.out.println("[" + interval[0] + ", " + interval[1] + "]");
        }
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(nlogn)

---

9. A Boolean Matrix Question

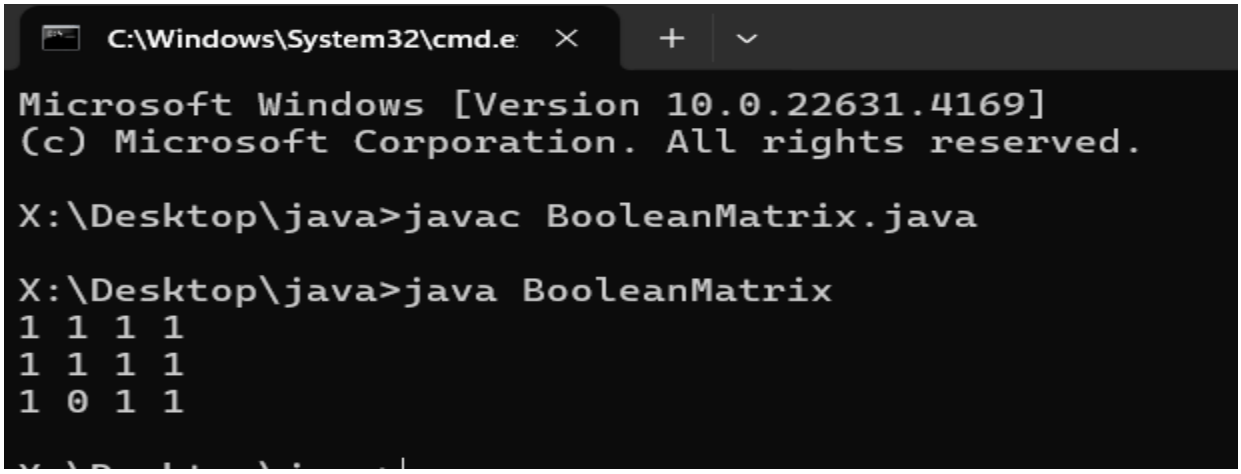
Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

```
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:

```
public class BooleanMatrix {
    public static void modifyMatrix(int[][] mat) {
        int M = mat.length;
        int N = mat[0].length;
        boolean[] row = new boolean[M];
        boolean[] col = new boolean[N];
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                if (mat[i][j] == 1) {
                    row[i] = true;
                    col[j] = true;
                }
            }
        }
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++) {
                if (row[i] || col[j]) {
                    mat[i][j] = 1;
                }
            }
        }
    }
    public static void main(String[] args) {
        int[][] mat = {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}};
        modifyMatrix(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();
        }
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(mn)

---

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 }}

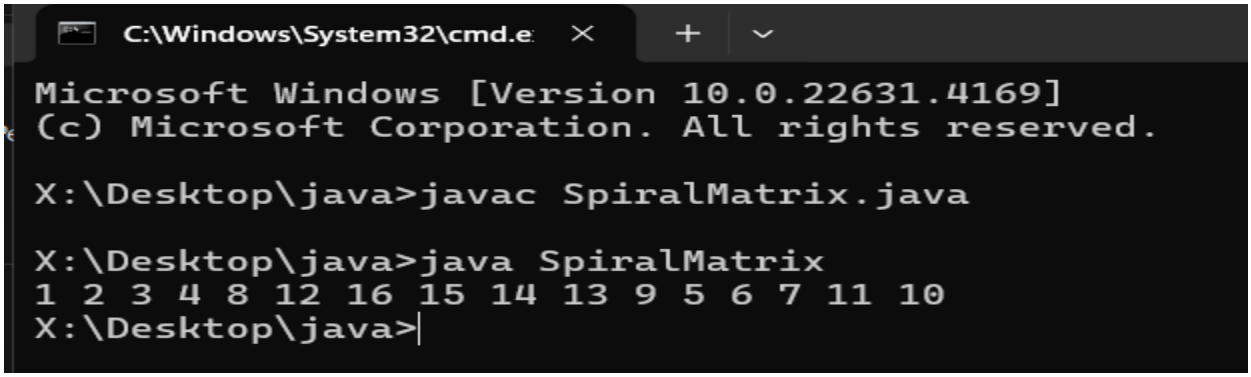
Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

CODE:

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

OUTPUT:



TIME COMPLEXITY: O(n)

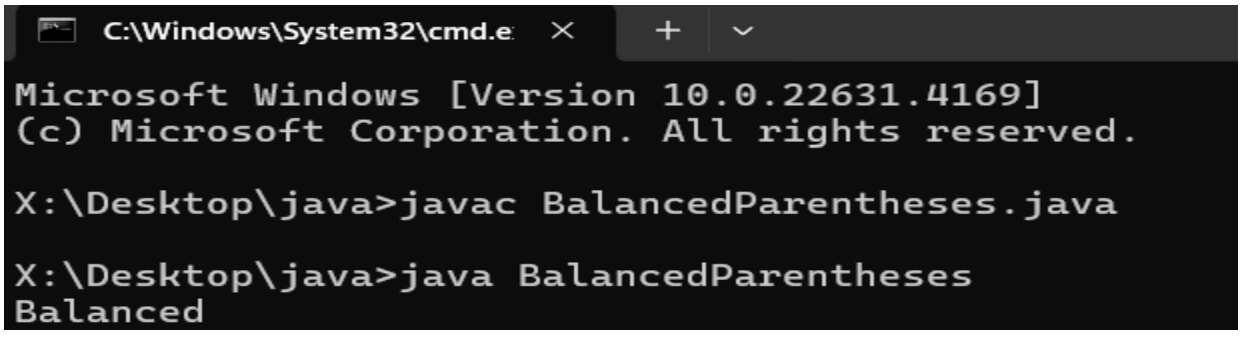
---

13. Check if given Parentheses expression is balanced or not
Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not.
Input: str = “((()))()()”
Output: Balanced

CODE:

```
public class BalancedParentheses {
    public static String checkBalanced(String str) {
        int count = 0;\
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == '(') {
                count++;
            } else if (str.charAt(i) == ')') {
                count--;
            }
            if (count < 0) {
                return "Not Balanced";
            }
        }
        return count == 0 ? "Balanced" : "Not Balanced";
    }
    public static void main(String[] args) {
        String str1 = "((()))()()";
        System.out.println(checkBalanced(str1));
    }
}
```

OUTPUT



TIME COMPLEXITY: O(n)

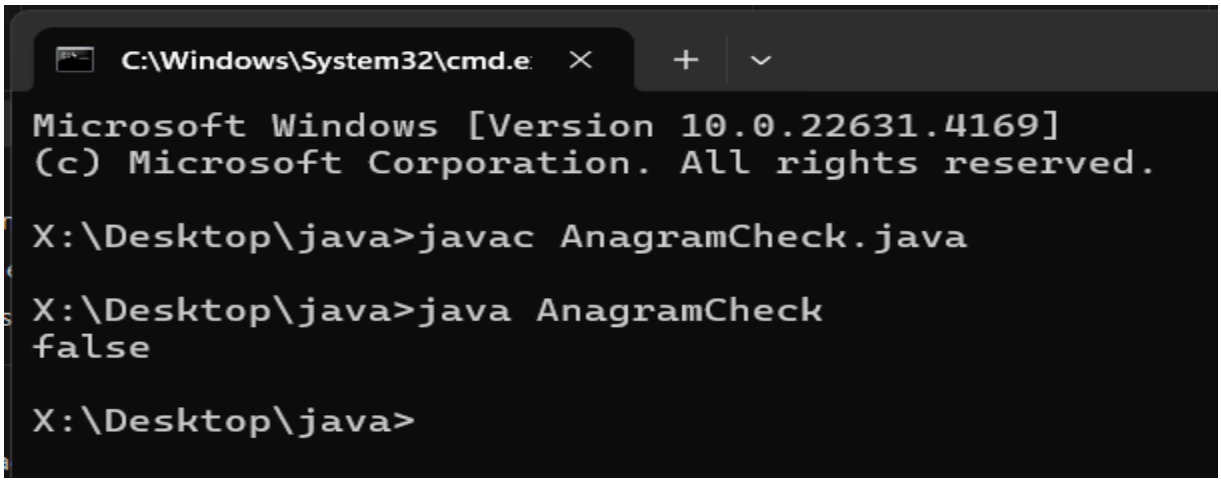
14. Check if two Strings are Anagrams of each other

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 strings have the same characters with the same frequency. So, they are anagrams.

CODE:

```
import java.util.Arrays;
public class AnagramCheck {
    public static boolean areAnagrams(String s1, String s2) {
        if (s1.length() != s2.length()) {
            return false;
        }
        char[] arr1 = s1.toCharArray();
        char[] arr2 = s2.toCharArray();

        Arrays.sort(arr1);
        Arrays.sort(arr2);
        return Arrays.equals(arr1, arr2);
    }
    public static void main(String[] args) {
        String s1 = "allergy";
        String s2 = "allergic";
    System.out.println(areAnagrams(s1, s2));
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(nlogn)

---

15. **Longest Palindromic Substring**

Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

Input: str = “forgeeksskeegfor”

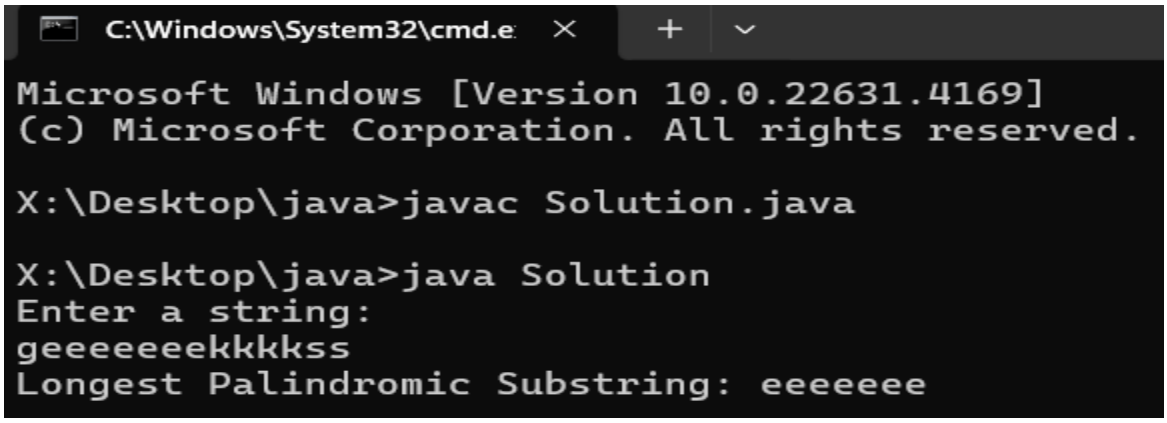
Output: “geeksskeeg” Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.

CODE:

```
import java.util.Scanner;
public class Solution {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a string: ");
        String s = sc.nextLine();
        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);

        sc.close();
    }
}
```

OUTPUT:



TIME COMPLEXITY: O(n^2)

16. Longest Common Prefix using Sorting

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

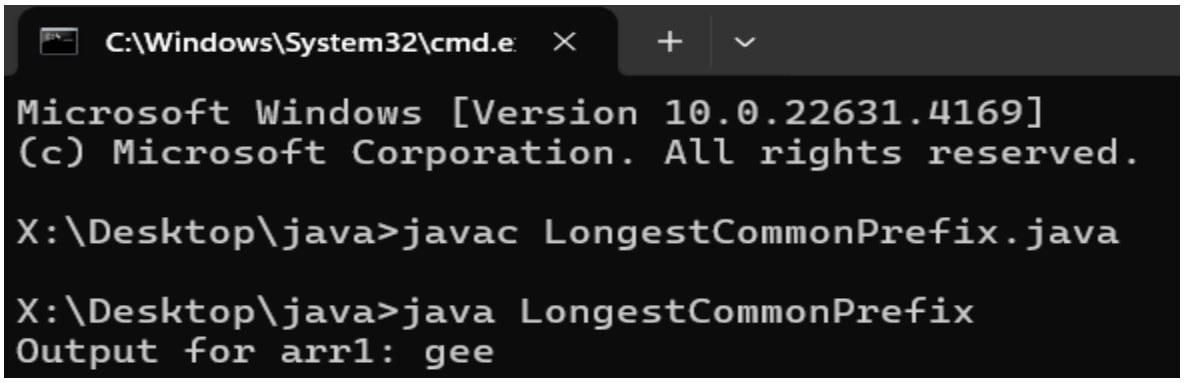
Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”]

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

CODE:

```
import java.util.Arrays;

public class LongestCommonPrefix {
    public 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 i = 0;
        while (i < first.length() && i < last.length() && first.charAt(i) ==
last.charAt(i)) {
            i++;
        }
        String prefix = first.substring(0, i);
        return prefix.isEmpty() ? "-1" : prefix;
    }

    public static void main(String[] args) {
        LongestCommonPrefix lcp = new LongestCommonPrefix();
        String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"};
        System.out.println("Output for arr1: " +
lcp.longestCommonPrefix(arr1)); // Expected Output: "gee"

    }
}
```

OUTPUT:



TIME COMPLEXITY: O(n*logm+n)

---

17. Delete middle element of a stack

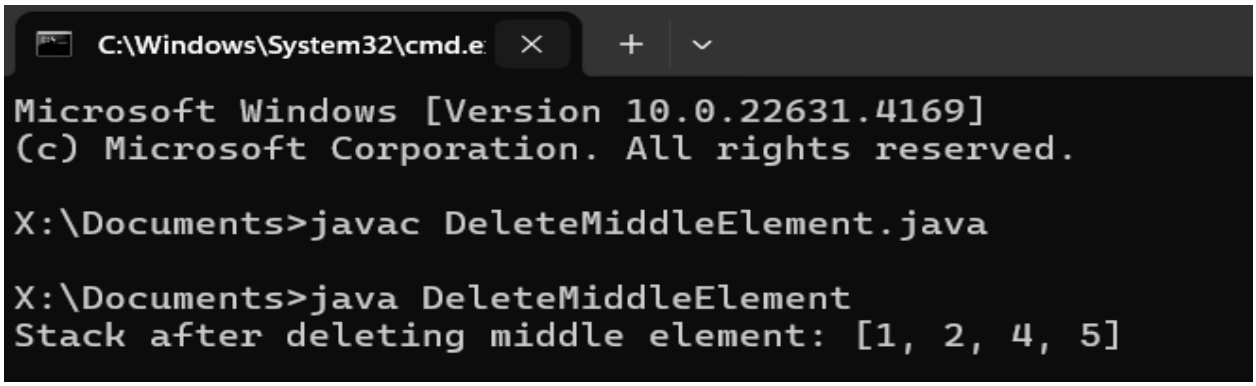
Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.
Input : Stack[] = [1, 2, 3, 4, 5]
Output : Stack[] = [1, 2, 4, 5]

CODE:

```
import java.util.Stack;
public class DeleteMiddleElement {
    public void deleteMiddle(Stack<Integer> stack, int size) {
        if (stack.isEmpty() || size == 0) return;
        int midIndex = size / 2;
        deleteMiddleHelper(stack, midIndex);
    }
    private void deleteMiddleHelper(Stack<Integer> stack, int currentIndex) {
        if (currentIndex == 0) {
            stack.pop();
            return;
        }
```

```
        int topElement = stack.pop();
        deleteMiddleHelper(stack, currentIndex - 1);
        stack.push(topElement);
    }
    public static void main(String[] args) {
        DeleteMiddleElement dme = new DeleteMiddleElement();
        Stack<Integer> stack = new Stack<>();
        stack.push(1);
        stack.push(2);
        stack.push(3);
        stack.push(4);
        stack.push(5);
        dme.deleteMiddle(stack, stack.size());
        System.out.println("Stack after deleting middle element: " + stack);
    }
}
```

OUTPUT:



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 exists, consider the next greater element as -1.

Input: arr[] = [4 , 5 , 2 , 25]
Output: 4 –> 5
5 –> 25
2 –> 25
25 –> -1
Explanation: Except 25 every element has an element greater than them present on the right side.

CODE:

```
import java.util.Stack;
import java.util.Arrays;

public class NextGreaterElement {
    public int[] findNextGreaterElements(int[] arr) {
        int[] result = new int[arr.length];
        Arrays.fill(result, -1);
        Stack<Integer> stack = new Stack<>();

        for (int i = 0; i < arr.length; i++) {
            while (!stack.isEmpty() && arr[stack.peek()] < arr[i]) {
                result[stack.pop()] = arr[i];
            }
            stack.push(i);
        }

        return result;
    }

    public static void main(String[] args) {
        NextGreaterElement nge = new NextGreaterElement();

        int[] arr1 = {4, 5, 2, 25};
        System.out.println("Array: " + Arrays.toString(arr1));
```

```
        System.out.println("Next Greater Elements: " +
Arrays.toString(nge.findNextGreaterElements(arr1)));

        int[] arr2 = {13, 7, 6, 12};
        System.out.println("Array: " + Arrays.toString(arr2));
        System.out.println("Next Greater Elements: " +
Arrays.toString(nge.findNextGreaterElements(arr2)));
    }
}
```

OUTPUT:

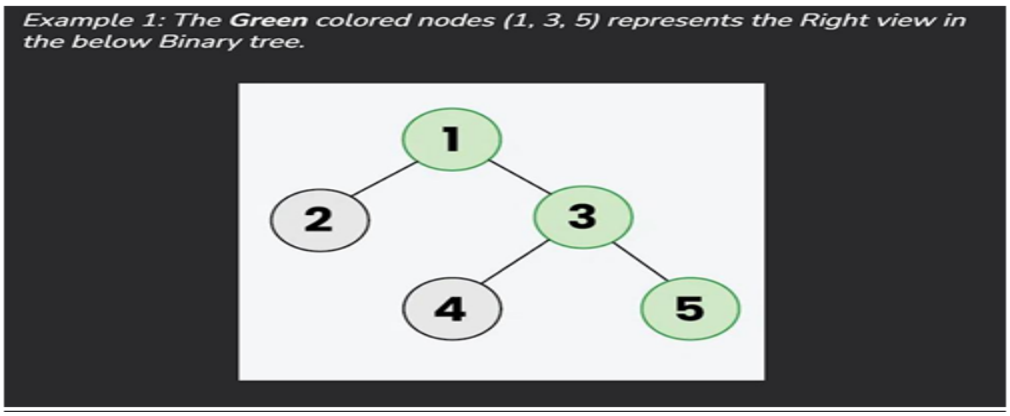
```
Microsoft Windows [Version 10.0.22631.4169]
(c) Microsoft Corporation. All rights reserved.

X:\Documents>javac NextGreaterElement.java

X:\Documents>java NextGreaterElement
Array: [4, 5, 2, 25]
Next Greater Elements: [5, 25, 25, -1]
Array: [13, 7, 6, 12]
Next Greater Elements: [-1, 12, 12, -1]
```

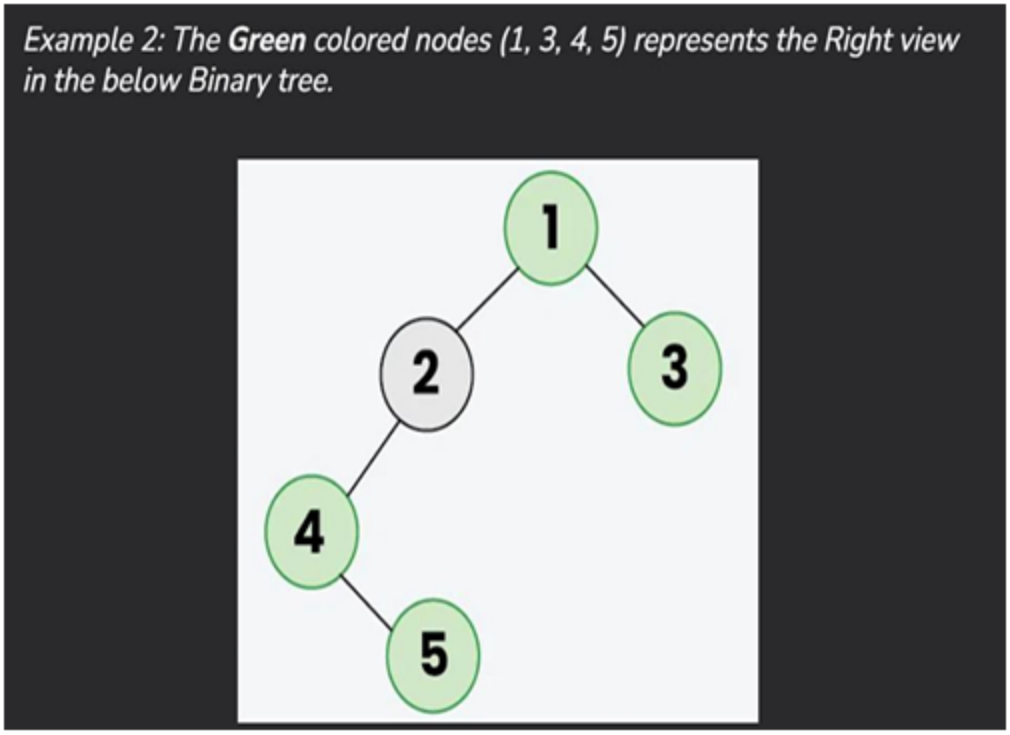
TIME COMPLEXITY: O(n)

---

19. Print Right View of a Binary Tree

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





CODE:

```
import java.util.*;

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

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

class Problem19 {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
```

```
        rightView(root, result, 0);
        return result;
    }

    public void rightView(TreeNode curr, List<Integer> result, int currDepth)
    {
        if(curr == null) {
            return;
        }
        if(currDepth == result.size()) {
            result.add(curr.val);
        }
        rightView(curr.right, result, currDepth + 1);
        rightView(curr.left, result, currDepth + 1);
    }

    public static void main(String[] args)
    {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.right.right = new TreeNode(6);
        root.left.left.left = new TreeNode(7);
        Problem19 solution = new Problem19();
        List<Integer> rightViewList = solution.rightSideView(root);
        System.out.println("Right View of the Binary Tree:");
        for (Integer val : rightViewList)
        {
            System.out.print(val + " ");
        }
    }
}
```

OUTPUT:

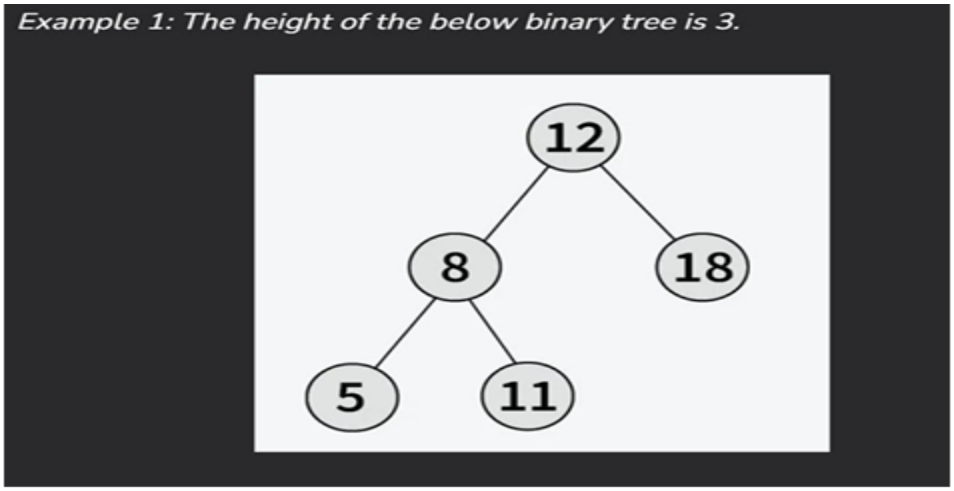
```
Right View of the Binary Tree:
1 3 6 7
```

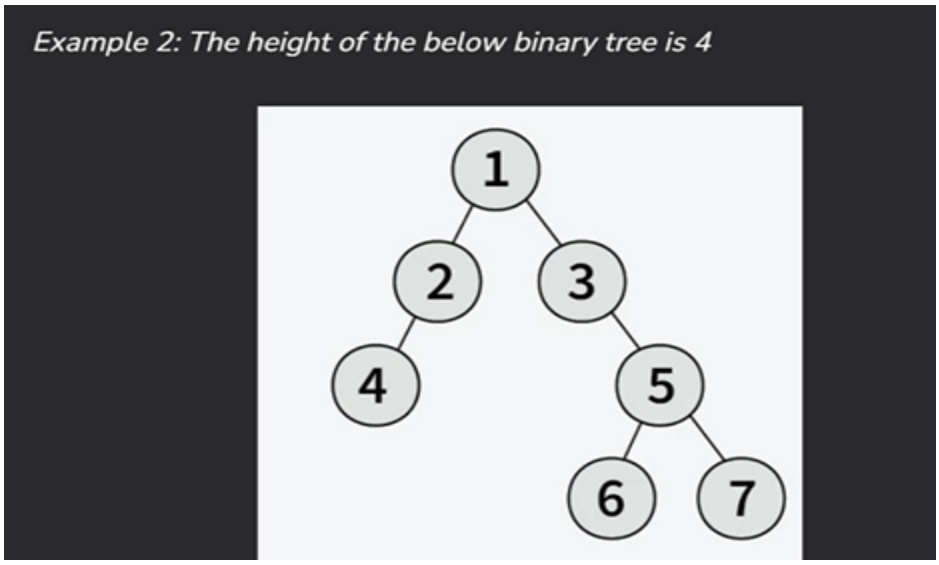
TIME COMPLEXITY: O(n)

20.

Maximum Depth or Height of Binary Tree

Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.





CODE:

```
class TreeNode {
    int value;
    TreeNode leftChild;
    TreeNode rightChild;

    TreeNode(int value) {
        this.value = value;
        this.leftChild = null;
        this.rightChild = null;
    }
}

class BinaryTree {
    public int getMaxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int leftHeight = getMaxDepth(root.leftChild);
        int rightHeight = getMaxDepth(root.rightChild);
        return 1 + Math.max(leftHeight, rightHeight);
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.leftChild = new TreeNode(2);
        root.rightChild = new TreeNode(3);
        root.leftChild.leftChild = new TreeNode(4);
        root.leftChild.rightChild = new TreeNode(5);
        root.rightChild.rightChild = new TreeNode(6);
        root.leftChild.leftChild.leftChild = new TreeNode(7);

        BinaryTree tree = new BinaryTree();
        int maxDepth = tree.getMaxDepth(root);
```

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

OUTPUT:

```
Maximum Depth of the Binary Tree: 4
```

TIME COMPLEXITY: O(n)
