VIJAY M (IT)

22IT125

CODING PRACTICES AND PROBLEMS

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

Time Complexity:O(n^2)

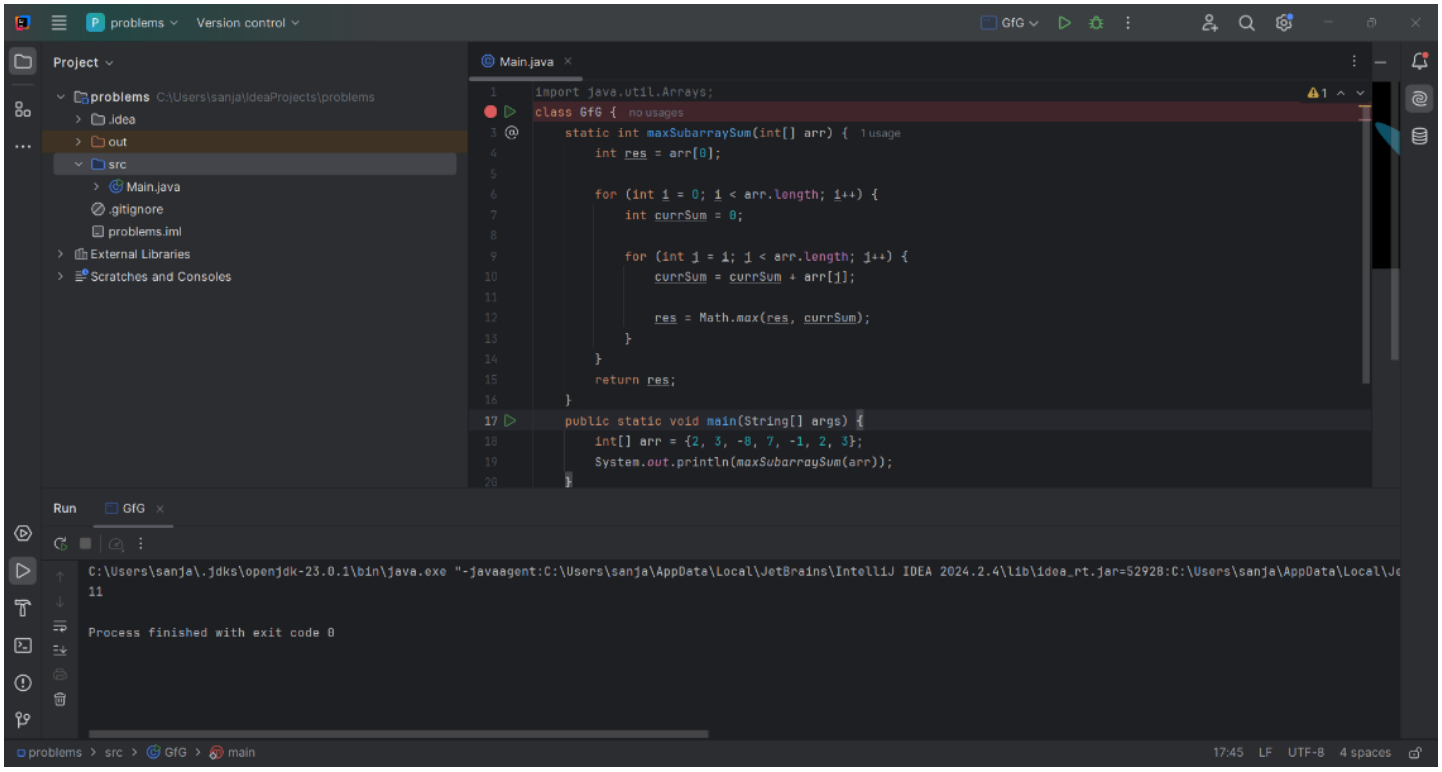
SOLUTION:

```
import java.util.Arrays;
class GfG {
    static int maxSubarraySum(int[] arr) {
        int res = arr[0];

        for (int i = 0; i < arr.length; i++) {
            int currSum = 0;

            for (int j = i; j < arr.length; j++) {
                currSum = currSum + arr[j];

                res = Math.max(res, currSum);
            }
        }
        return res;
    }
    public static void main(String[] args) {
        int[] arr = {2, 3, -8, 7, -1, 2, 3};
        System.out.println(maxSubarraySum(arr));
    }
}
```

Kadane's Algorithm:

```
import java.util.Arrays;
class GfG {
    static int maxSubarraySum(int[] arr) {
        int res = arr[0];
        int maxEnding = arr[0];
        for (int i = 1; i < arr.length; i++) {
            maxEnding = Math.max(maxEnding + arr[i], arr[i]);
            res = Math.max(res, maxEnding);
        }
        return res;
    }
    public static void main(String[] args) {
        int[] arr = {2, 3, -8, 7, -1, 2, 3};
        System.out.println(maxSubarraySum(arr));
    }
}
```

OUTPUTt:



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

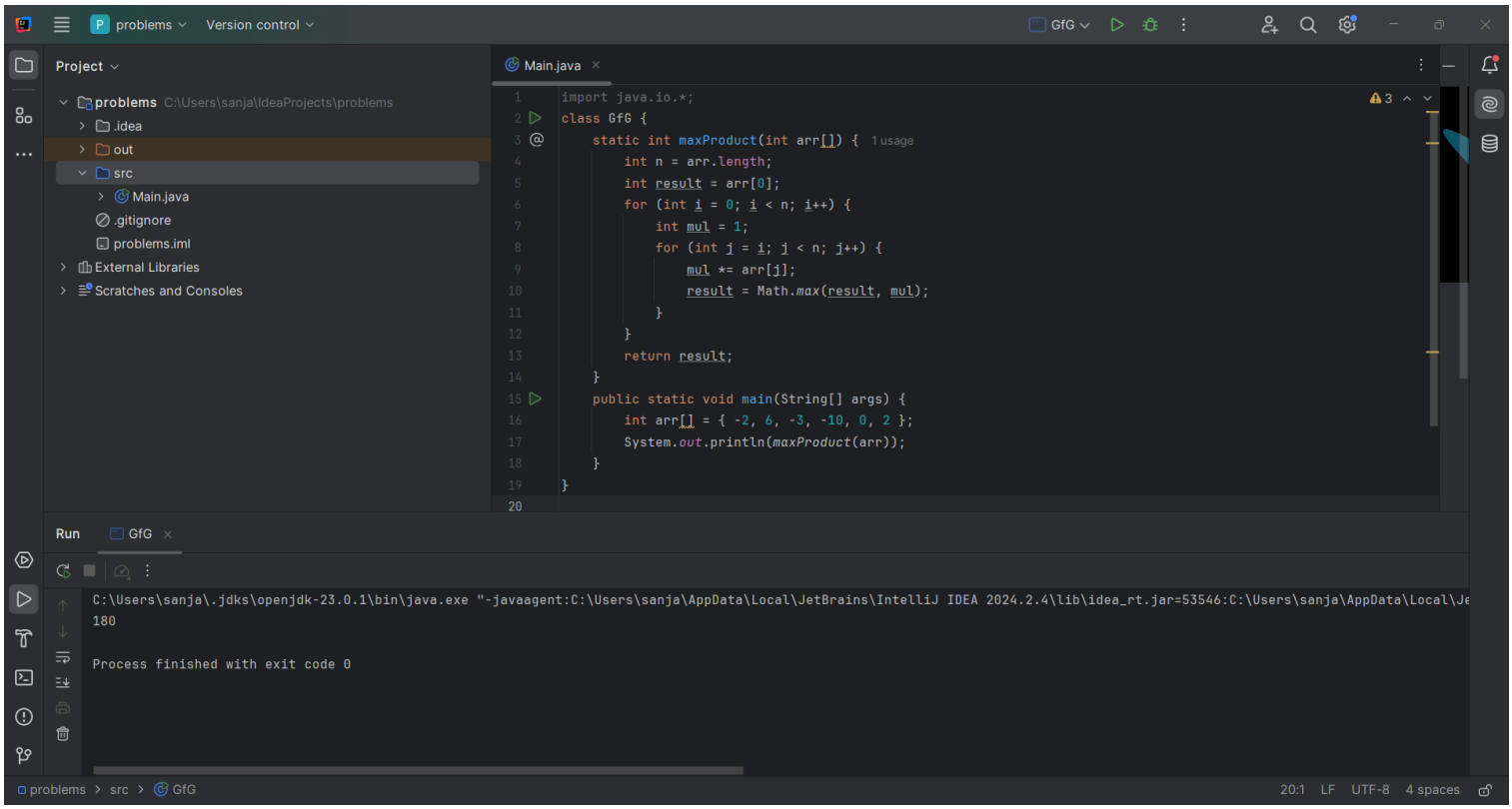
Explanation: The subarray with maximum product is {60}.

SOLUTION:

```
import java.io.*;
class GfG {
    static int maxProduct(int arr[]) {
        int n = arr.length;
        int result = arr[0];
        for (int i = 0; i < n; i++) {
            int mul = 1;
            for (int j = i; j < n; j++) {
                mul *= arr[j];
                result = Math.max(result, mul);
            }
        }
        return result;
    }
    public static void main(String[] args) {
        int arr[] = { -2, 6, -3, -10, 0, 2 };
        System.out.println(maxProduct(arr));
    }
}
```

Time Complexity:O(n)

OUTPUT:



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

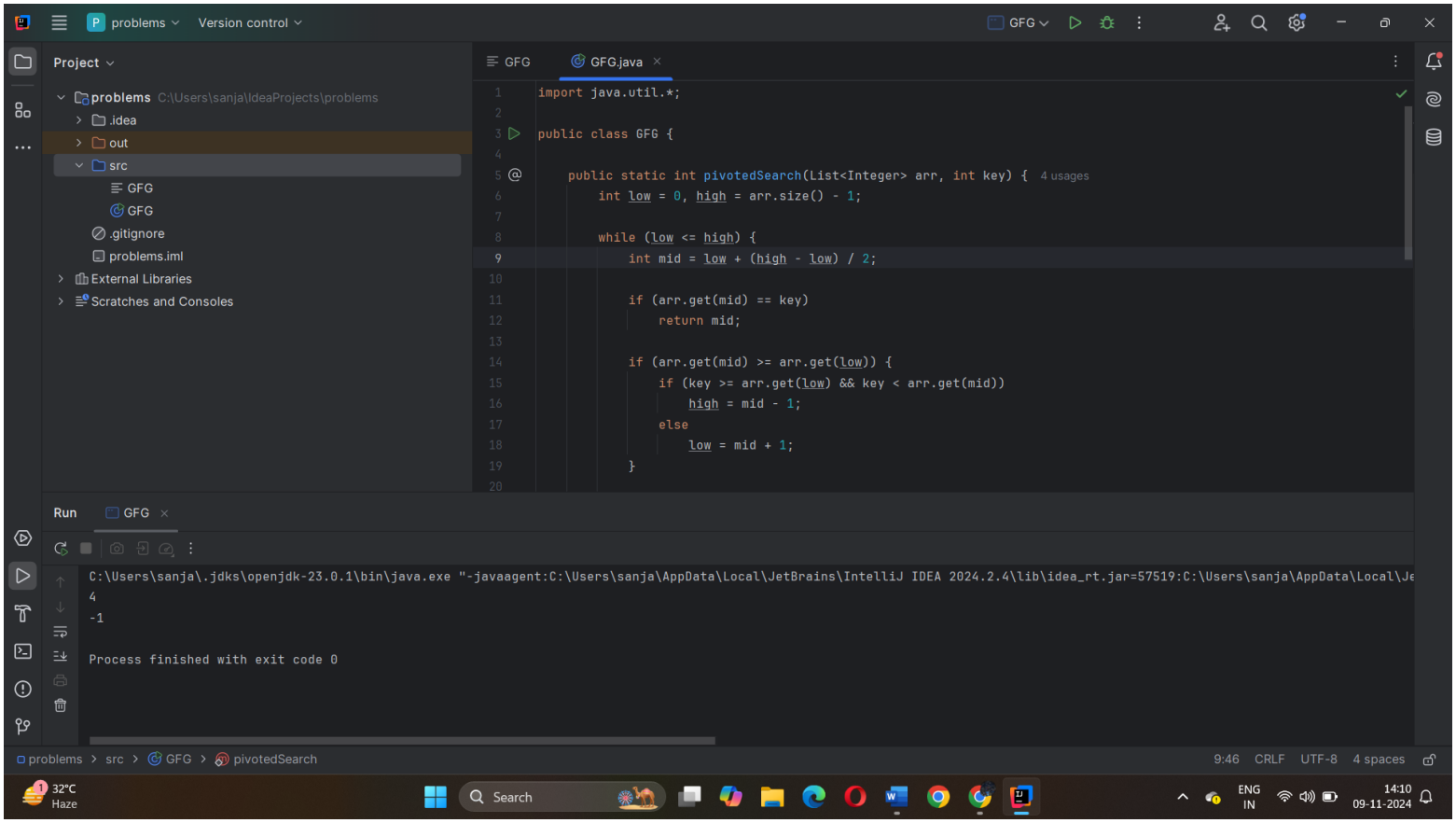
SOLUTION:

```
import java.util.*;

public class GFG {

    public static int pivotedSearch(List<Integer> arr, int key) {
        int low = 0, high = arr.size() - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (arr.get(mid) == key)
                return mid;

            if (arr.get(mid) >= arr.get(low)) {
                if (key >= arr.get(low) && key < arr.get(mid))
                    high = mid - 1;
                else
                    low = mid + 1;
            }

            else {
                if (key > arr.get(mid) && key <= arr.get(high))
                    low = mid + 1;
                else
```

```
                high = mid - 1;
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        List<Integer> arr1 = Arrays.asList(4, 5, 6, 7, 0, 1, 2);
        int key1 = 0;
        int result1 = pivotedSearch(arr1, key1);
        System.out.println(result1);

        List<Integer> arr2 = Arrays.asList(4, 5, 6, 7, 0, 1, 2);
        int key2 = 3;
        int result2 = pivotedSearch(arr2, key2);
        System.out.println(result2);
    }
}
```

Time Complexity: O(log n)

OUTPUT:



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

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

Output: 12

Explanation: 5 and 3 are distance 4 apart. So the size of the base = 4.

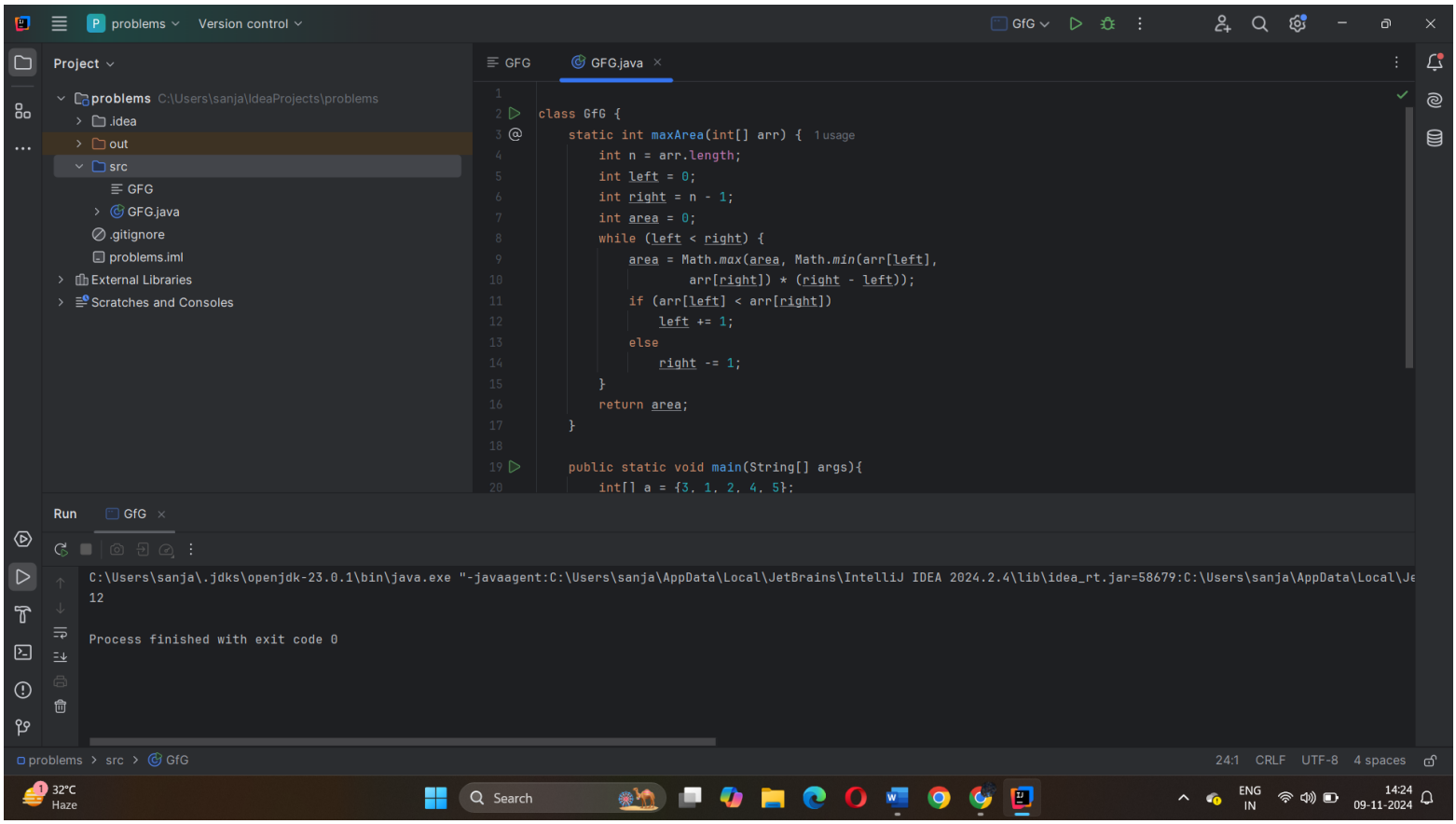
Height of container = min(5, 3) = 3. So total area = 4 * 3 = 12

SOLUTION:

```
class GfG {
    static int maxArea(int[] arr) {
        int n = arr.length;
        int left = 0;
        int right = n - 1;
        int area = 0;
        while (left < right) {
            area = Math.max(area, Math.min(arr[left],
                        arr[right]) * (right - left));
            if (arr[left] < arr[right])
                left += 1;
            else
                right -= 1;
        }
        return area;
    }

    public static void main(String[] args){
        int[] a = {3, 1, 2, 4, 5};
        System.out.println(maxArea(a));
    }
}
```

Time Complexity:O(n)

OUTPUT:



5. Find the Factorial of a large number

Input: 100

Output:
933262154439441526816992388562667004907159682643816214685929638952175999932299

156089414639761565182862536979208272237582511852109168640000000000000000000000 00

Input: 50

Output:
30414093201713378043612608166064768844377641568960512000000000000

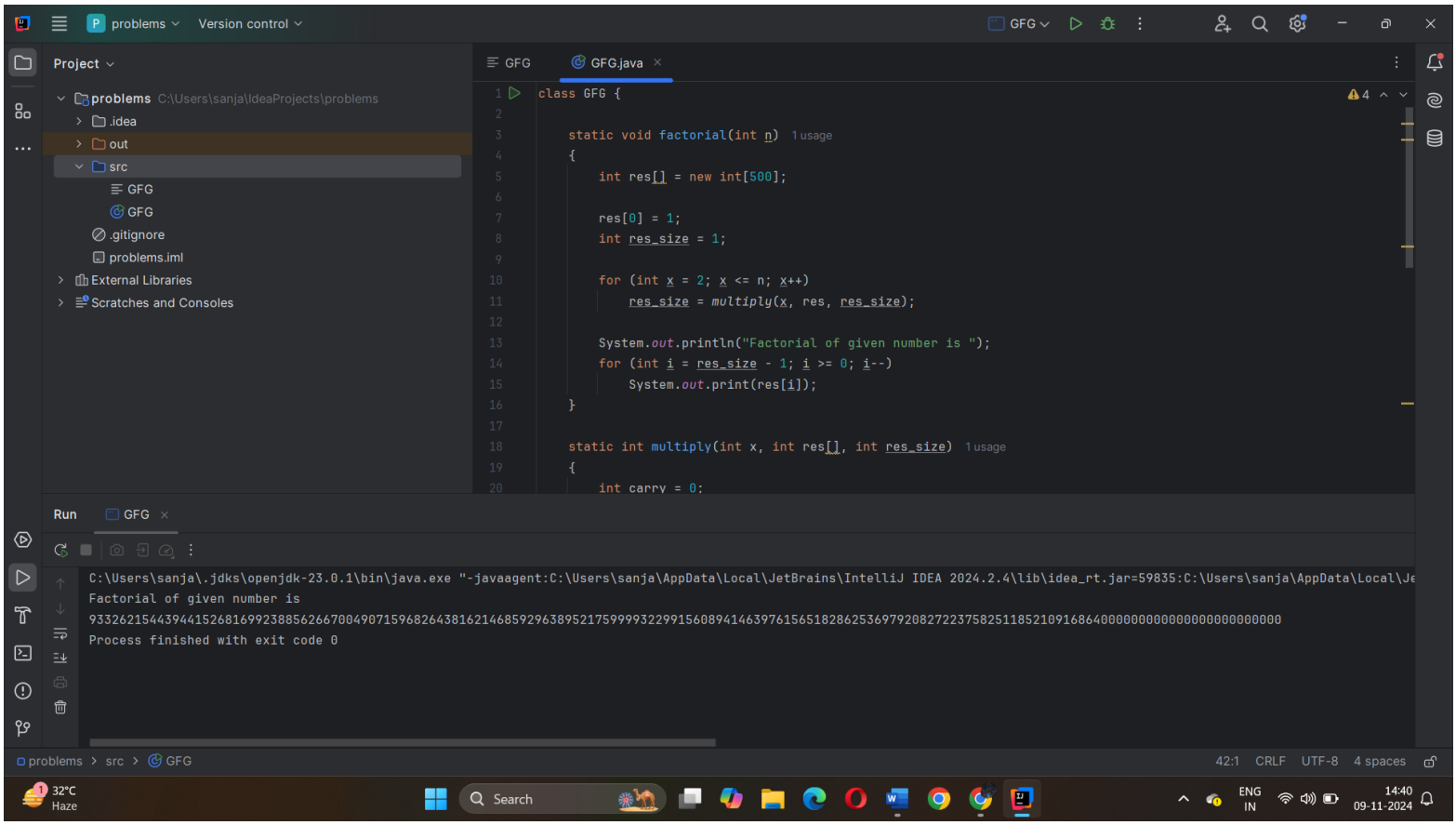
SOLUTION:

```
class GFG {

   static void factorial(int n)
   {
      int res[] = new int[500];

      res[0] = 1;
      int res_size = 1;

      for (int x = 2; x <= n; x++)
         res_size = multiply(x, res, res_size);

      System.out.println("Factorial of given number is ");
      for (int i = res_size - 1; i >= 0; i--)
         System.out.print(res[i]);
   }

   static int multiply(int x, int res[], int res_size)
   {
      int carry = 0;

      for (int i = 0; i < res_size; i++) {
         int prod = res[i] * x + carry;
```

```
        res[i] = prod % 10;

        carry = prod / 10;
    }

    while (carry != 0) {
        res[res_size] = carry % 10;
        carry = carry / 10;
        res_size++;
    }
    return res_size;
  }

  public static void main(String args[])
  {
    factorial(100);
  }
}
```

Time Complexity:O(n)

OUTPUT:



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

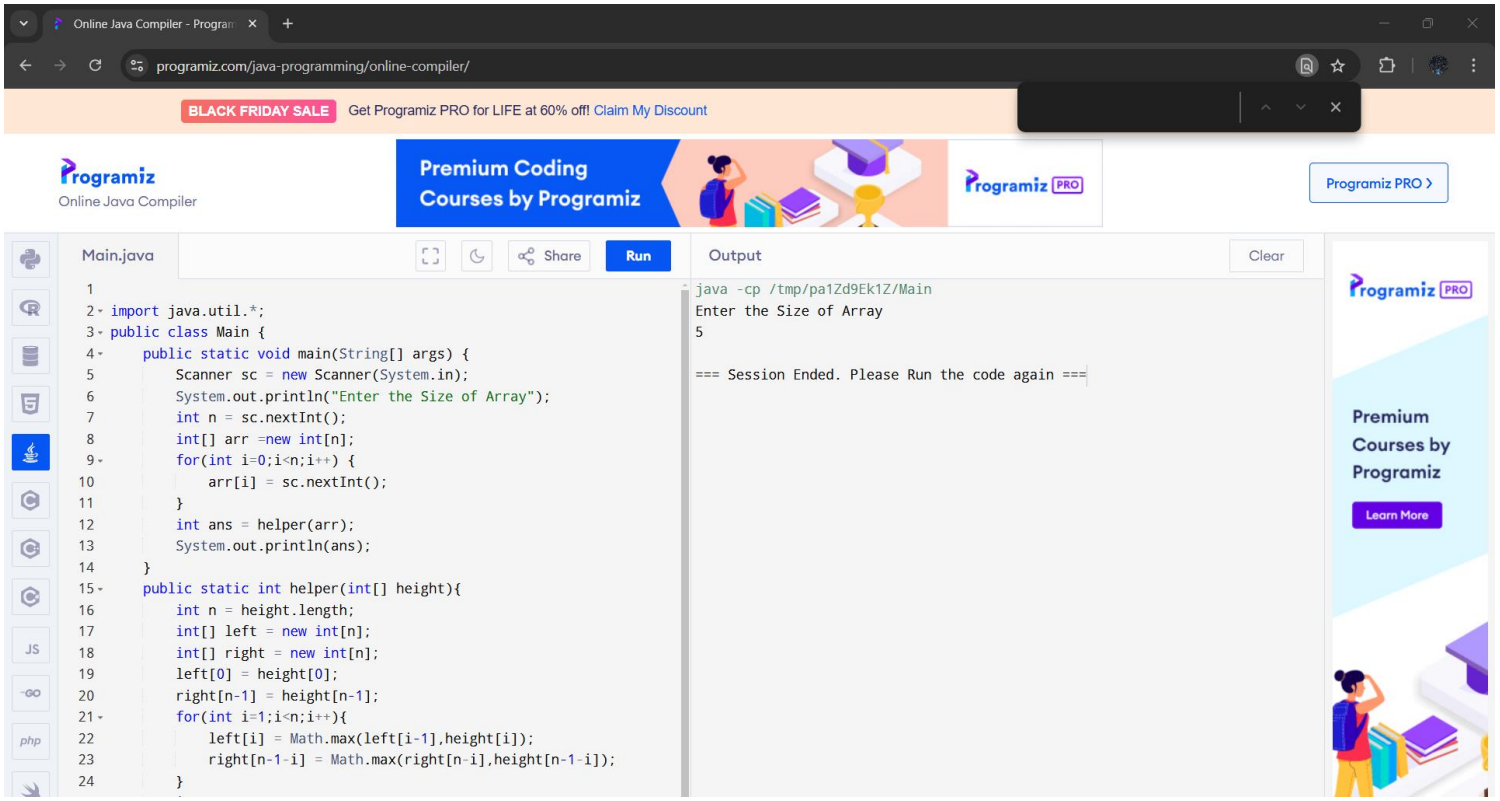
SOLUTION:

```
import java.util.*;
public class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the Size of Array");
      int n = sc.nextInt();
      int[] arr =new int[n];
      for(int i=0;i<n;i++) {
         arr[i] = sc.nextInt();
      }
      int ans = helper(arr);
      System.out.println(ans);
   }
   public static int helper(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;
    }
}
```

Time complexity: O(n)

OUTPUT:



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

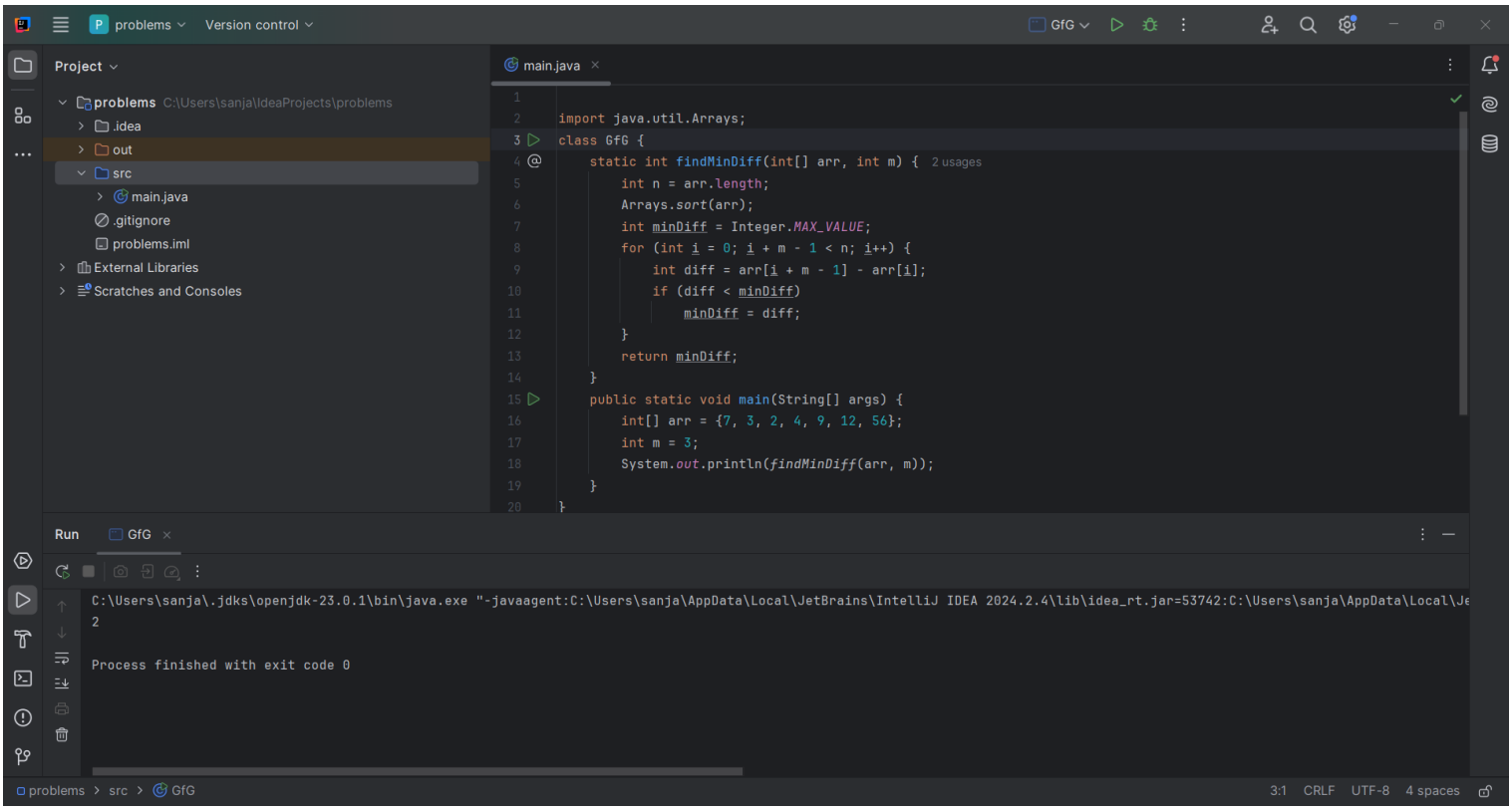
Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum difference, that is 9 – 2 = 7.

SOLUTION:

```
import java.util.Arrays;
class GfG {
    static int findMinDiff(int[] arr, int m) {
        int n = arr.length;
        Arrays.sort(arr);
        int minDiff = Integer.MAX_VALUE;
        for (int i = 0; i + m - 1 < n; i++) {
            int diff = arr[i + m - 1] - arr[i];
            if (diff < minDiff)
                minDiff = diff;
        }
        return minDiff;
    }
    public static void main(String[] args) {
        int[] arr = {7, 3, 2, 4, 9, 12, 56};
        int m = 3;
        System.out.println(findMinDiff(arr, m));
    }
}
```

Time Complexity: n*log(n)

OUTPUT:



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].

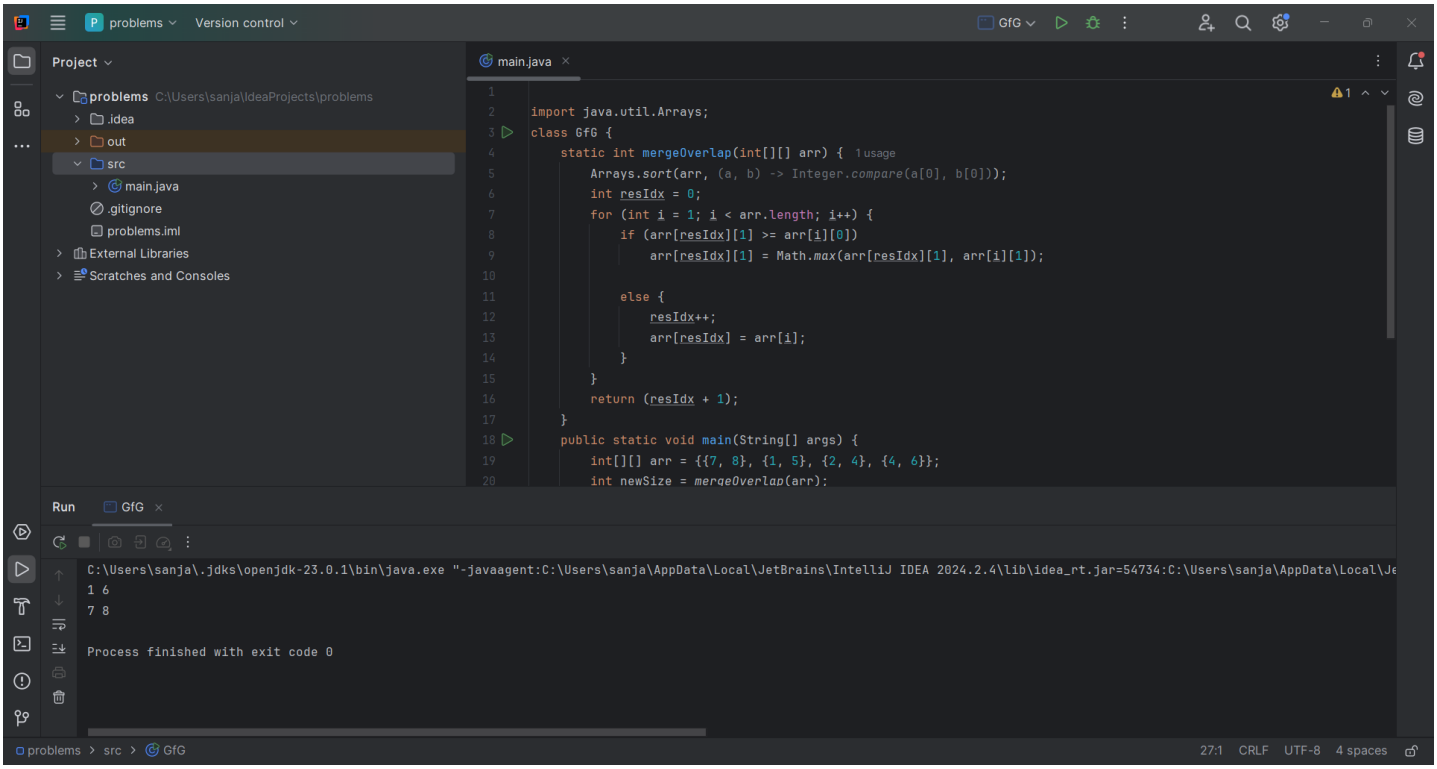
SOLUTION

```
import java.util.Arrays;
class GfG {
    static int mergeOverlap(int[][] arr) {
        Arrays.sort(arr, (a, b) -> Integer.compare(a[0], b[0]));
        int resIdx = 0;
        for (int i = 1; i < arr.length; i++) {
            if (arr[resIdx][1] >= arr[i][0])
                arr[resIdx][1] = Math.max(arr[resIdx][1], arr[i][1]);

            else {
                resIdx++;
                arr[resIdx] = arr[i];
            }
        }
        return (resIdx + 1);
    }
    public static void main(String[] args) {
        int[][] arr = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};
        int newSize = mergeOverlap(arr);

        for (int i = 0; i < newSize; i++) {
```

```
        System.out.println(arr[i][0] + " " + arr[i][1]);
    }
  }
}
```

Time Complexity: O(n)

OUTPUT:



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}}

SOLUTION:

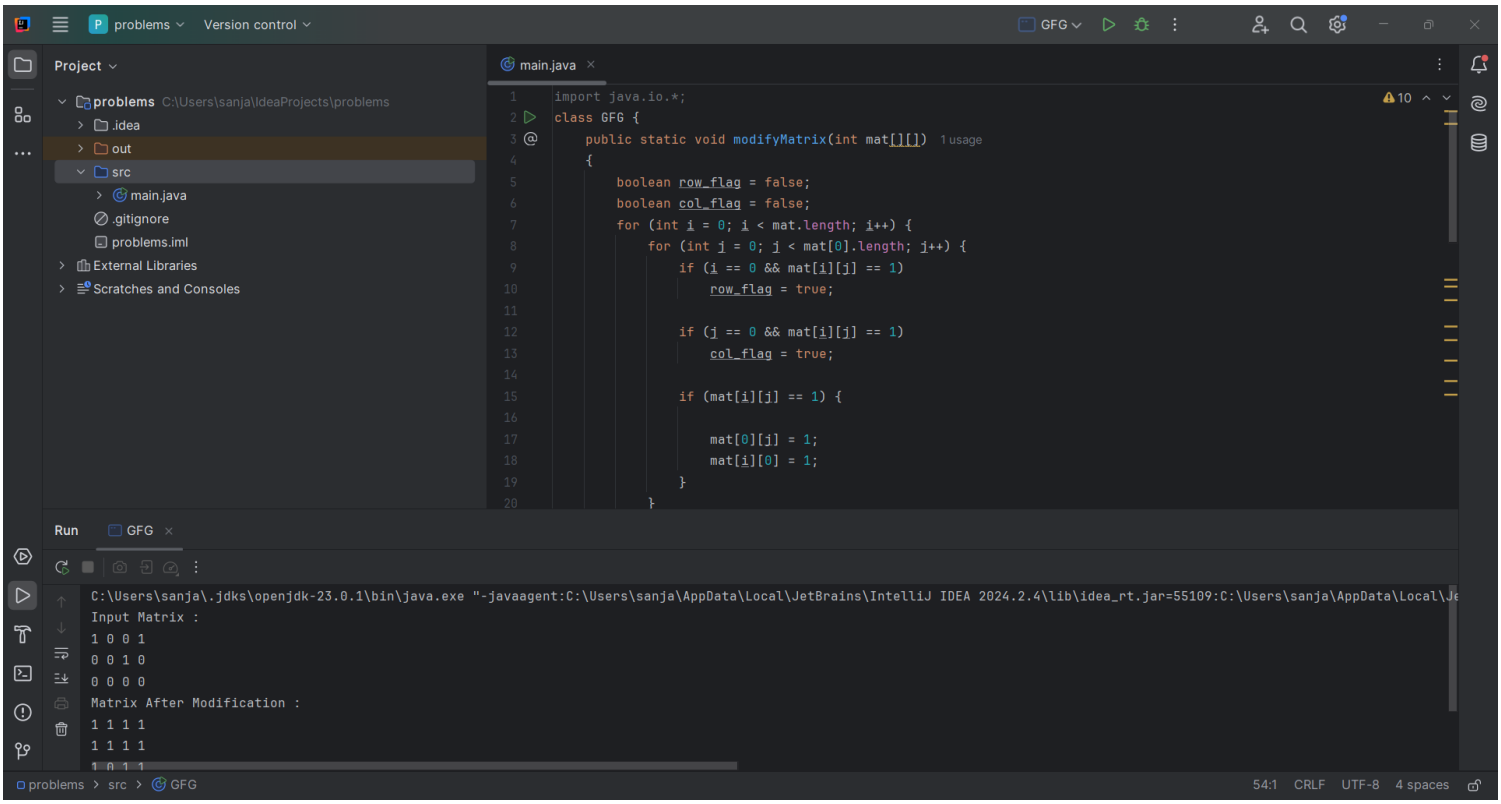
```
import java.io.*;
class GFG {
    public static void modifyMatrix(int mat[][])
    {
        boolean row_flag = false;
        boolean col_flag = false;
        for (int i = 0; i < mat.length; i++) {
            for (int j = 0; j < mat[0].length; j++) {
                if (i == 0 && mat[i][j] == 1)
                    row_flag = true;

                if (j == 0 && mat[i][j] == 1)
                    col_flag = true;

                if (mat[i][j] == 1) {

                    mat[0][j] = 1;
                    mat[i][0] = 1;
```

```
            }
        }
    }
    for (int i = 1; i < mat.length; i++)
        for (int j = 1; j < mat[0].length; j++)
            if (mat[0][j] == 1 || mat[i][0] == 1)
                mat[i][j] = 1;
    if (row_flag == true)
        for (int i = 0; i < mat[0].length; i++)
            mat[0][i] = 1;
    if (col_flag == true)
        for (int i = 0; i < mat.length; i++)
            mat[i][0] = 1;
}
public static void printMatrix(int mat[][])
{
    for (int i = 0; i < mat.length; i++) {
        for (int j = 0; j < mat[0].length; j++)
            System.out.print(mat[i][j] + " ");
        System.out.println("");
    }
}
public static void main(String args[])
{
    int mat[][] = { { 1, 0, 0, 1 },
                    { 0, 0, 1, 0 },
```

```
                { 0, 0, 0, 0 } };

        System.out.println("Input Matrix :");
        printMatrix(mat);
        modifyMatrix(mat);
        System.out.println("Matrix After Modification :");
        printMatrix(mat);
    }
}
```

Time Complexity:O((N*M)*(N + M)). O(N*M)

OUTPUT:



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.

SOLUTION:

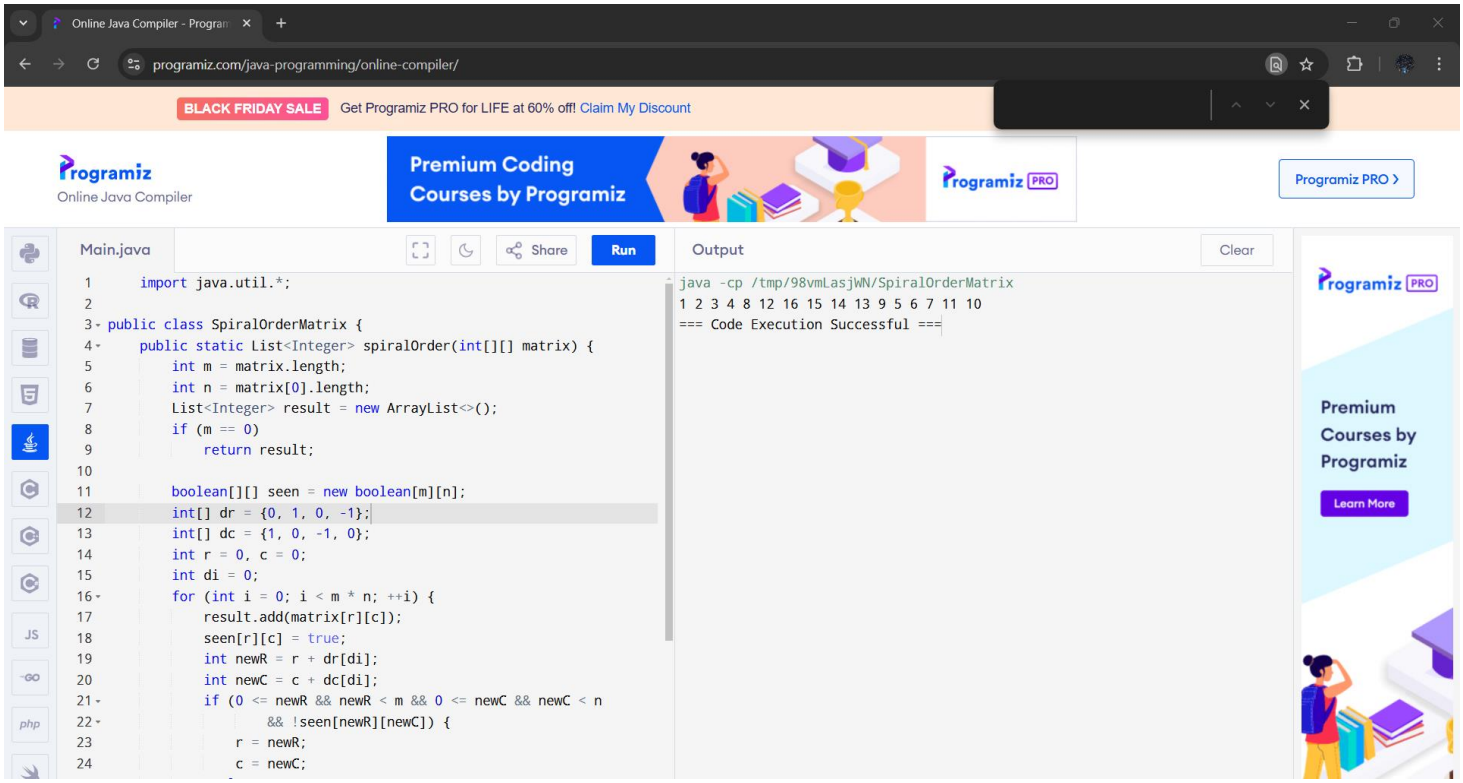
```
import java.util.*;


public class SpiralOrderMatrix {
    public static List<Integer> spiralOrder(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;
        List<Integer> result = new ArrayList<>();
        if (m == 0)
            return result;
```

```
    boolean[][] seen = new boolean[m][n];
    int[] dr = {0, 1, 0, -1};
    int[] dc = {1, 0, -1, 0};
    int r = 0, c = 0;
    int di = 0;
    for (int i = 0; i < m * n; ++i) {
        result.add(matrix[r][c]);
        seen[r][c] = true;
        int newR = r + dr[di];
        int newC = c + dc[di];
        if (0 <= newR && newR < m && 0 <= newC && newC < n
                && !seen[newR][newC]) {
            r = newR;
            c = newC;
        } else {
            di = (di + 1) % 4;
            r += dr[di];
            c += dc[di];
        }
    }
    return result;
}
public static void main(String[] args) {
    int[][] matrix = {
            { 1, 2, 3, 4 },
            { 5, 6, 7, 8 },
```

```
        { 9, 10, 11, 12 },
        { 13, 14, 15, 16 }
    };
    List<Integer> result = spiralOrder(matrix);
    for (int num : result) {
        System.out.print(num + " ");
    }
  }
}
```

Time Complexity: O(m*n)

OUTPUT:



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

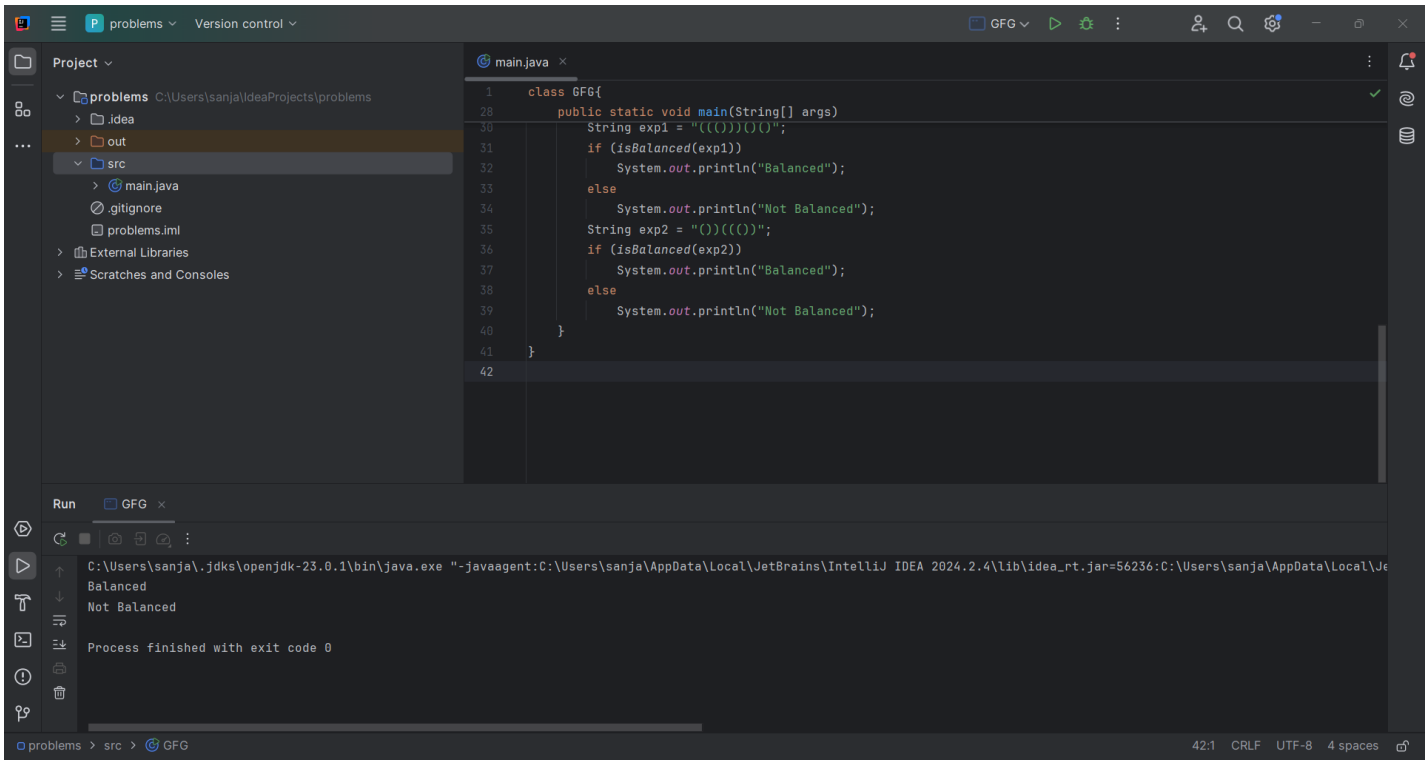
SOLUTION:

```
class GFG{
public static boolean isBalanced(String exp)
{
    boolean flag = true;
    int count = 0;
    for(int i = 0; i < exp.length(); i++)
    {
        if (exp.charAt(i) == '('
        {
            count++;
        }
        else
        {
            count--;
        }
        if (count < 0)
```

```
            {
                flag = false;
                break;
            }
        }
        if (count != 0)
        {
            flag = false;
        }
        return flag;
}
public static void main(String[] args)
{
        String exp1 = "(((()))()()";
        if (isBalanced(exp1))
            System.out.println("Balanced");
        else
            System.out.println("Not Balanced");
        String exp2 = "())((())";
        if (isBalanced(exp2))
            System.out.println("Balanced");
        else
            System.out.println("Not Balanced");
}
}
```

Time Complexity:O(N)

OUTPUT:



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.

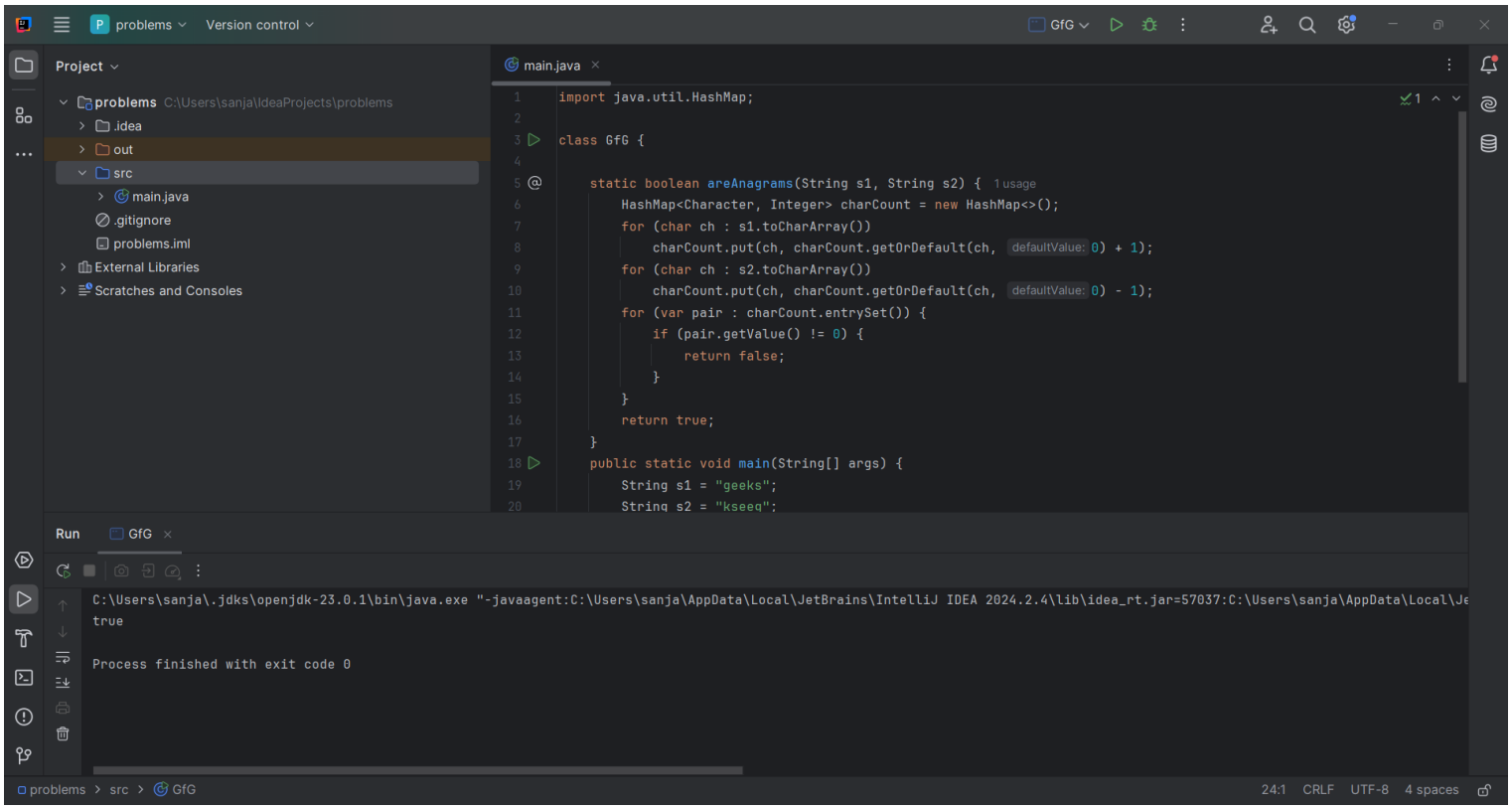
SOLUTION:

```
import java.util.HashMap;

class GfG {

  static boolean areAnagrams(String s1, String s2) {
    HashMap<Character, Integer> charCount = new HashMap<>();
    for (char ch : s1.toCharArray())
      charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);
    for (char ch : s2.toCharArray())
      charCount.put(ch, charCount.getOrDefault(ch, 0) - 1);
    for (var pair : charCount.entrySet()) {
      if (pair.getValue() != 0) {
        return false;
      }
```

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

Time Complexity:O(m + n)

OUTPUT:



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: “”

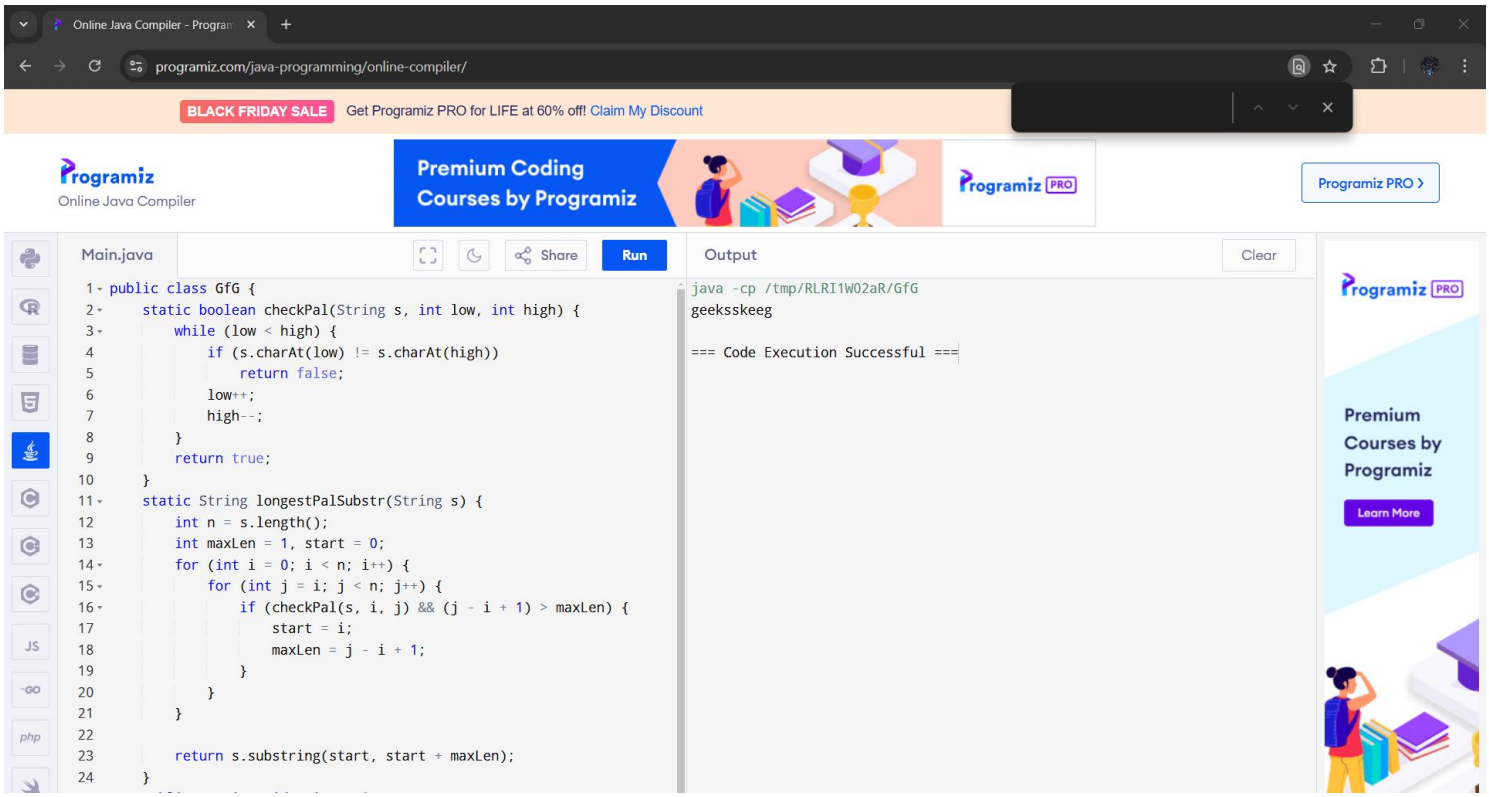
SOLUTION:

```
public class GfG {
    static boolean checkPal(String s, int low, int high) {
        while (low < high) {
            if (s.charAt(low) != s.charAt(high))
                return false;
            low++;
```

```
            high--;
        }
        return true;
    }
    static String longestPalSubstr(String s) {
        int n = s.length();
        int maxLen = 1, start = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i; j < n; j++) {
                if (checkPal(s, i, j) && (j - i + 1) > maxLen) {
                    start = i;
                    maxLen = j - i + 1;
                }
            }
        }

        return s.substring(start, start + maxLen);
    }
    public static void main(String[] args) {
        String s = "forgeeksskeegfor";
        System.out.println(longestPalSubstr(s));
    }
}
```

Time Complexity:O(N3)

OUTPUT:



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

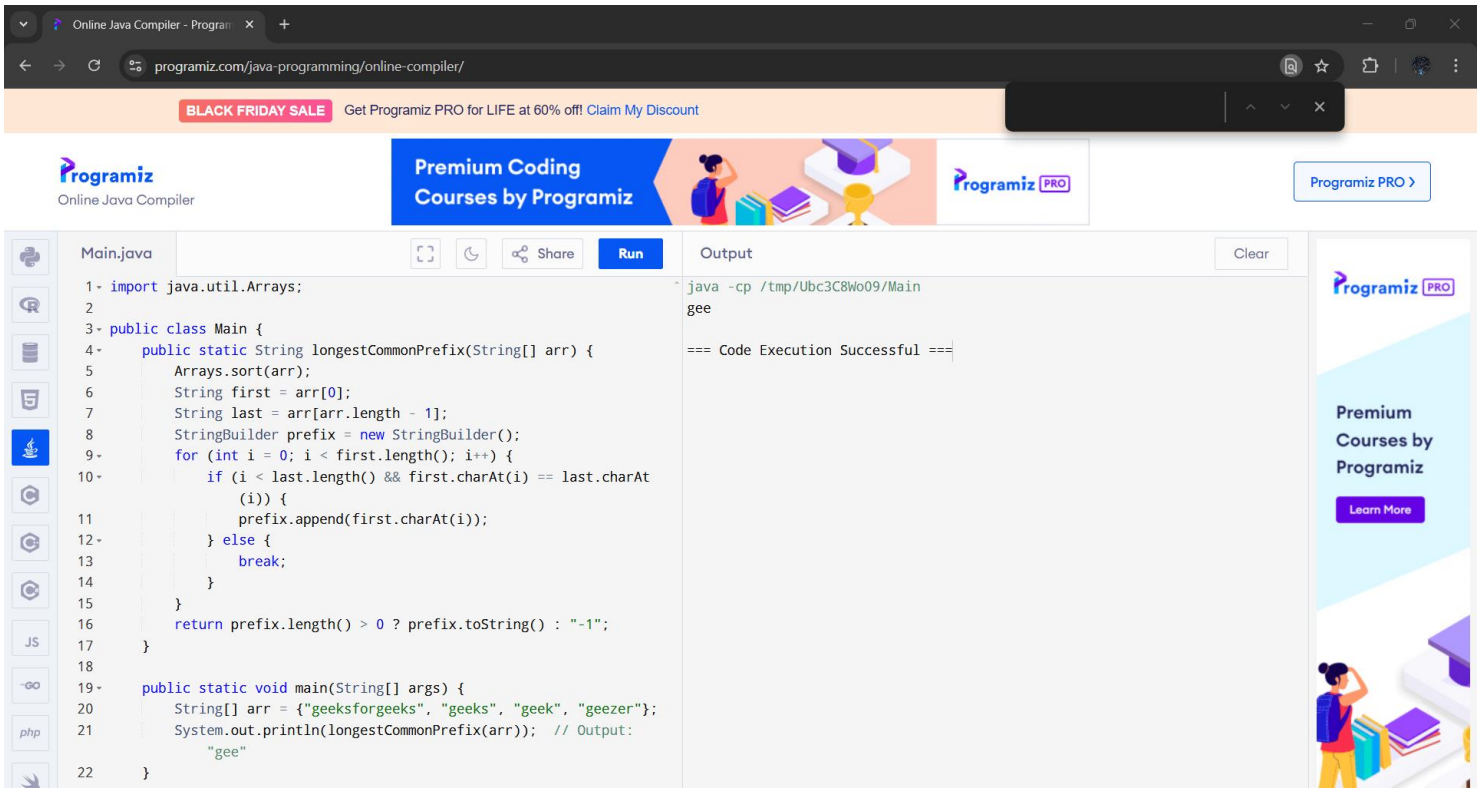
Explanation: There‟s no common prefix in the given strings.

SOLUTION:

```
import java.util.Arrays;

public class Main {
    public static String longestCommonPrefix(String[] arr) {
        Arrays.sort(arr);
        String first = arr[0];
        String last = arr[arr.length - 1];
        StringBuilder prefix = new StringBuilder();
        for (int i = 0; i < first.length(); i++) {
            if (i < last.length() && first.charAt(i) == last.charAt(i)) {
                prefix.append(first.charAt(i));
            } else {
                break;
            }
        }
        return prefix.length() > 0 ? prefix.toString() : "-1";
    }

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

Time Complexity:O(n log n+m)

OUTPUT:



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]

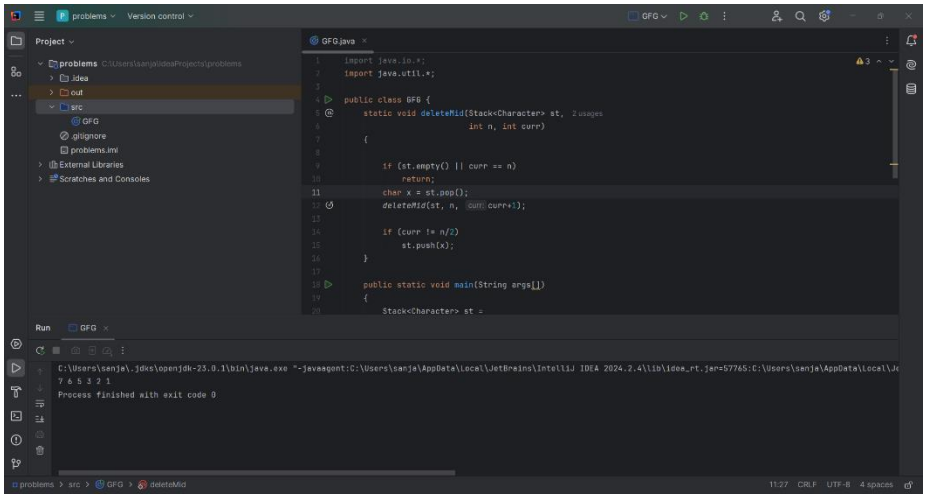
SOLUTION:

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

public class GFG {
    static void deleteMid(Stack<Character> st,
                                    int n, int curr)
    {

        if (st.empty() || curr == n)
            return;
        char x = st.pop();
        deleteMid(st, n, curr+1);

        if (curr != n/2)
            st.push(x);
    }

    public static void main(String args[])
    {
        Stack<Character> st =
                    new Stack<Character>();
        st.push('1');
        st.push('2');
```

```
        st.push('3');
        st.push('4');
        st.push('5');
        st.push('6');
        st.push('7');

        deleteMid(st, st.size(), 0);

        while (!st.empty())
        {
            char p=st.pop();
            System.out.print(p + " ");
        }
    }
}
```

Time Complexity:

OUTPUT:



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

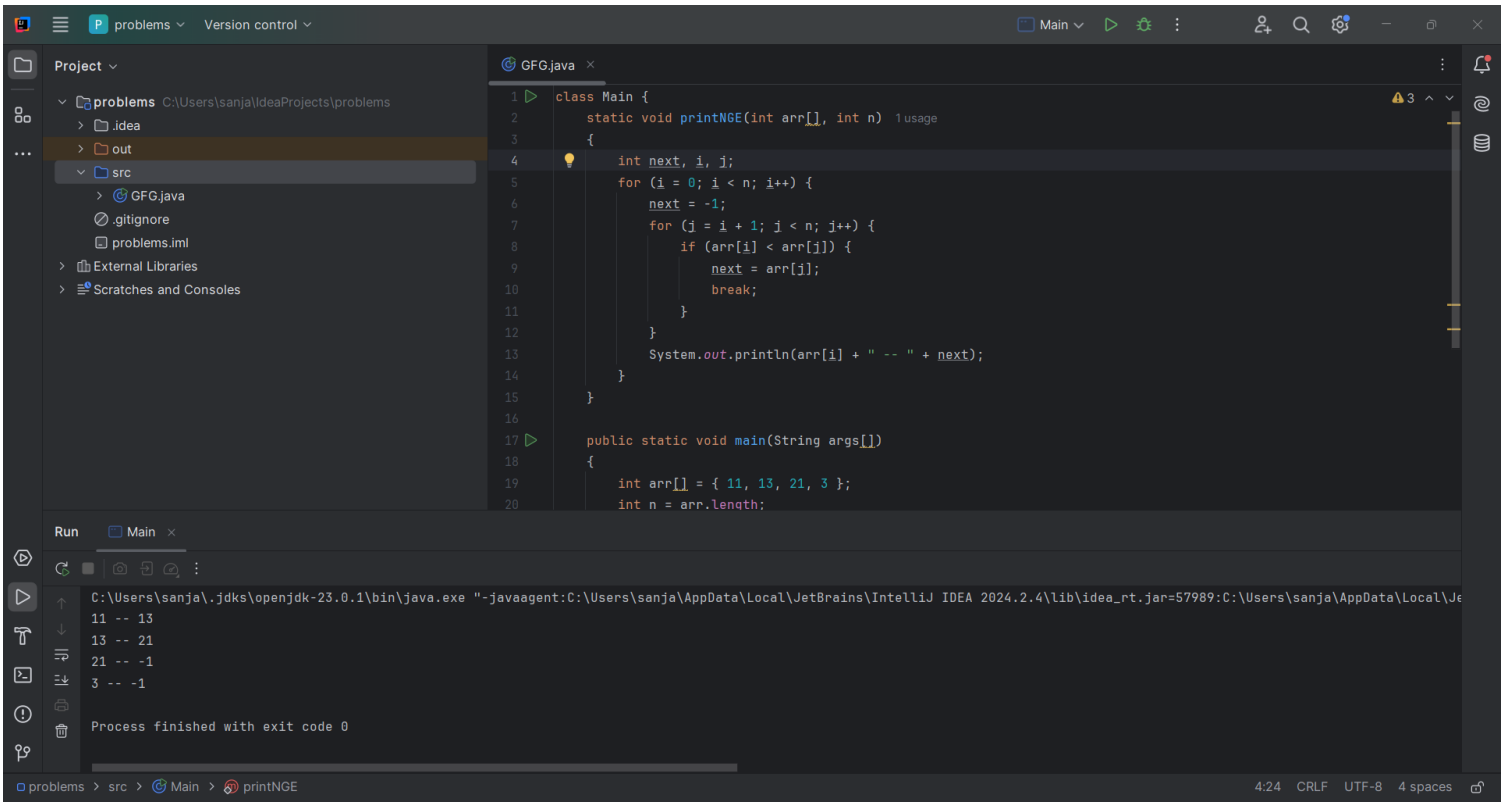
SOLUTION:

```
class Main {
  static void printNGE(int arr[], int n)
  {
     int next, i, j;
     for (i = 0; i < n; i++) {
        next = -1;
```

```
        for (j = i + 1; j < n; j++) {
            if (arr[i] < arr[j]) {
                next = arr[j];
                break;
            }
        }
        System.out.println(arr[i] + " -- " + next);
    }
  }

  public static void main(String args[])
  {
    int arr[] = { 11, 13, 21, 3 };
    int n = arr.length;
    printNGE(arr, n);
  }
}
```

Time Complexity:O(N^)

OUTPUT:



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.

SOLUTION:

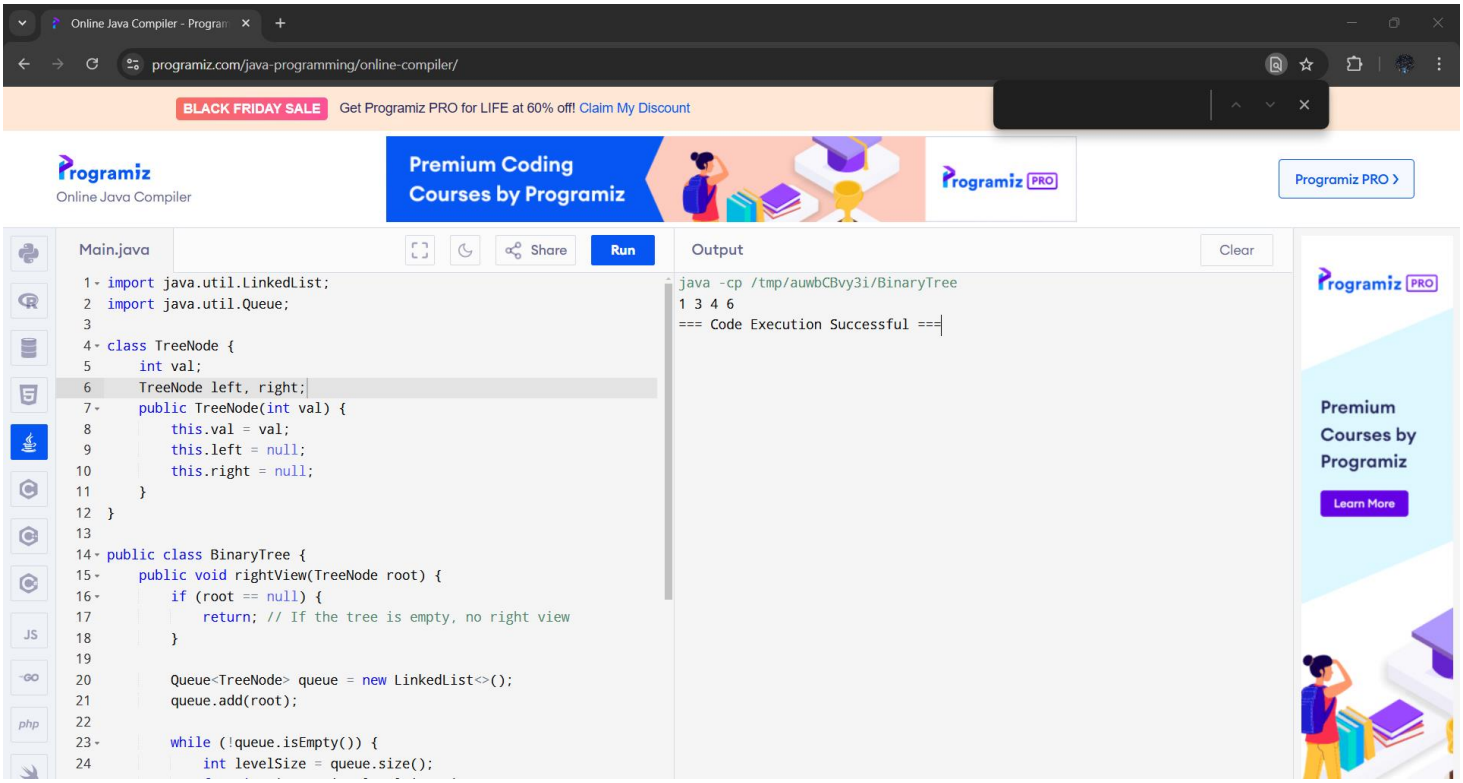
```
import java.util.LinkedList;
import java.util.Queue;


class TreeNode {
    int val;
    TreeNode left, right;
    public TreeNode(int val) {
        this.val = val;
```

```
        this.left = null;
        this.right = null;
    }
}

public class BinaryTree {
    public void rightView(TreeNode root) {
        if (root == null) {
            return; // If the tree is empty, no right view
        }

        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            for (int i = 0; i < levelSize; i++) {
                TreeNode node = queue.poll();
                if (i == levelSize - 1) {
                    System.out.print(node.val + " ");
                }
                if (node.left != null) {
                    queue.add(node.left);
                }
                if (node.right != null) {
                    queue.add(node.right);
```

```
            }
        }
    }
}

public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();

    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);
    root.left.right.left = new TreeNode(6);
    tree.rightView(root);  // Output: 1 3 4 6
}
}
```

Time Complexity:O(n)

OUTPUT:



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.. Maximum Depth or Height of Binary Tree

SOLUTION:

```
class TreeNode {
    int val;
    TreeNode left, right;
    public TreeNode(int val) {
        this.val = val;
        this.left = null;
```

```
        this.right = null;
    }
}

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

        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);
        root.left.right.left = new TreeNode(6);
        System.out.println("Maximum Depth of the Tree: " +
tree.maxDepth(root));  // Output: 4
    }
}
```

Time Complexity: O(N)

OUTPUT: