**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.  
  
Code:**import java.util.\*;

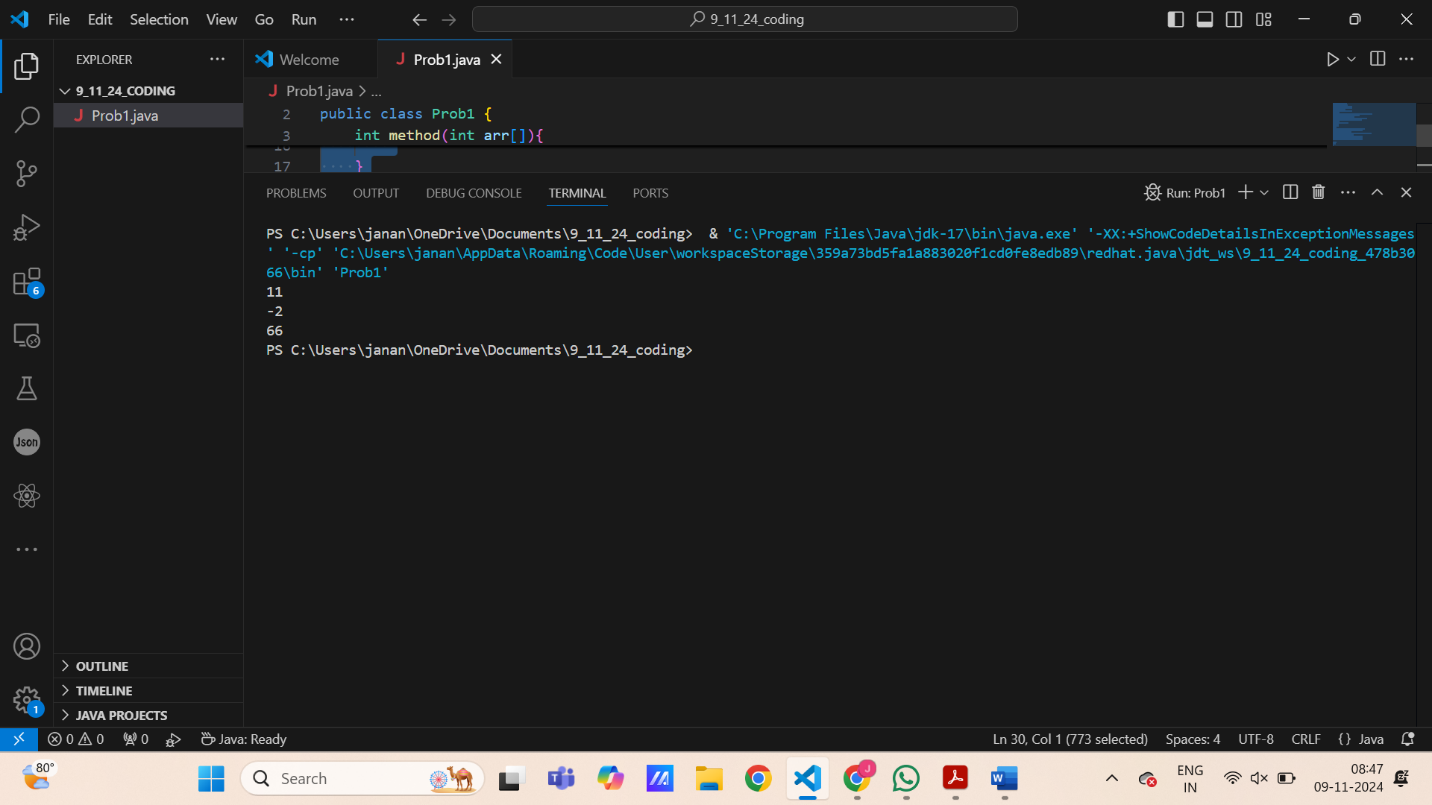
public class Prob1 {

    int method(int arr[]){

        int n= arr.length;

        if(n==1){

            return arr[0];

        }****

        int sum1=Arrays.stream(arr).min().getAsInt();

        int max\_s=Integer.MIN\_VALUE;

        for(int i=0;i<n;i++){

            sum1=Math.max(arr[i],sum1+arr[i]);

            max\_s=Math.max(max\_s,sum1);

        }

        return max\_s;

    }

    public static void main(String args[]){

        int a[]={2, 3, -8, 7, -1, 2, 3};

        int b[]={-2, -4};

        int c[]={7,-9,3,-1,9,7,4,8,6,10,20,-13,-30};

        Prob1 p=new Prob1();

        System.out.println(p.method(a));

        System.out.println(p.method(b));

        System.out.println(p.method(c));

    }

}  
  
Output:

**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}.  
  
Code:**import java.util.\*;

public class Prob2 {

    int method(int arr[]){

        int n=arr.length;

        int maxp=arr[0];

        int minp=arr[0];

        int res=arr[0];

        int temp;

        for(int i=1;i<n;i++){

            if (arr[i]<0){

                temp=maxp;

                maxp=minp;

                minp=temp;

            }

            maxp=Math.max(arr[i],maxp\*arr[i]);

            minp=Math.min(arr[i],minp\*arr[i]);

            res=Math.max(maxp,res);

        }

        return res;

    }

    public static void main(String args[]){

        int a[]={-2, 6, -3, -10, 0, 2};

        int b[]={-1, -3, -10, 0, 60};

        int c[]={9,7,3,-22,45,-66,3,4};

        Prob2 p=new Prob2();

        System.out.println(p.method(a));

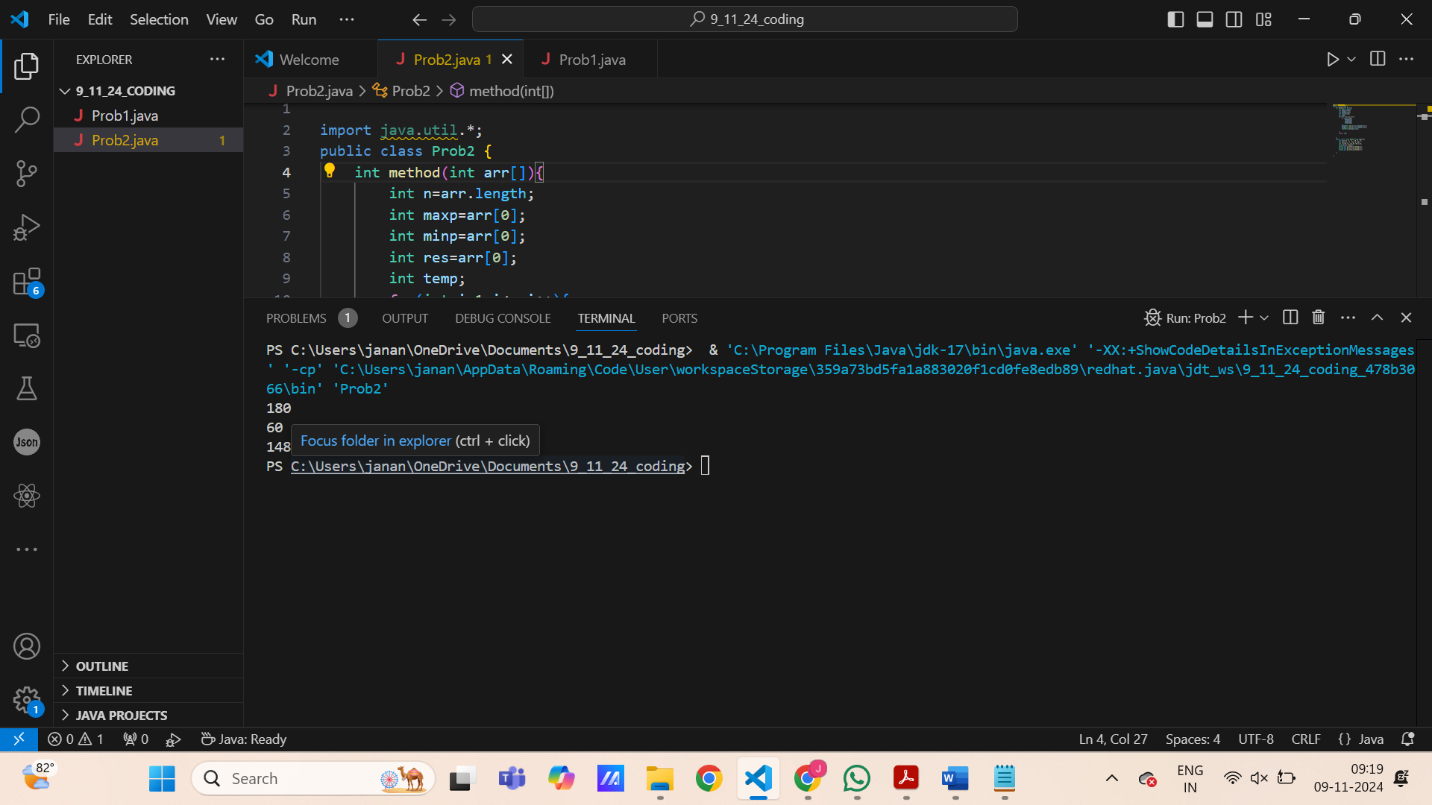
        System.out.println(p.method(b));

        System.out.println(p.method(c));

    }

}

Output:



**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**

**CODE:**

public class Prob3 {

    int method(int arr[],int key){

        int l=0;

        int r=arr.length-1;

        while(l<=r){

            int m=(l+r)/2;

            if(arr[m]==key){

                return m;

            }

            if(arr[l]<=arr[m]){

                if(arr[l]<=key && key<arr[m]){

                    r=m-1;

                }

                else{

                    l=m+1;

                }

            }

            else{

                if(arr[m]<key && key<=arr[r]){

                    l=m+1;

                }

                else{

                    r=m-1;

                }

            }

        }

        return -1;

    }

    public static void main(String args[]){

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

        int k1=0;

        int k2=3;

        int b[]={50, 10, 20, 30, 40};

        int k3=10;

        Prob3 p=new Prob3();

        System.out.println(p.method(a,k1));

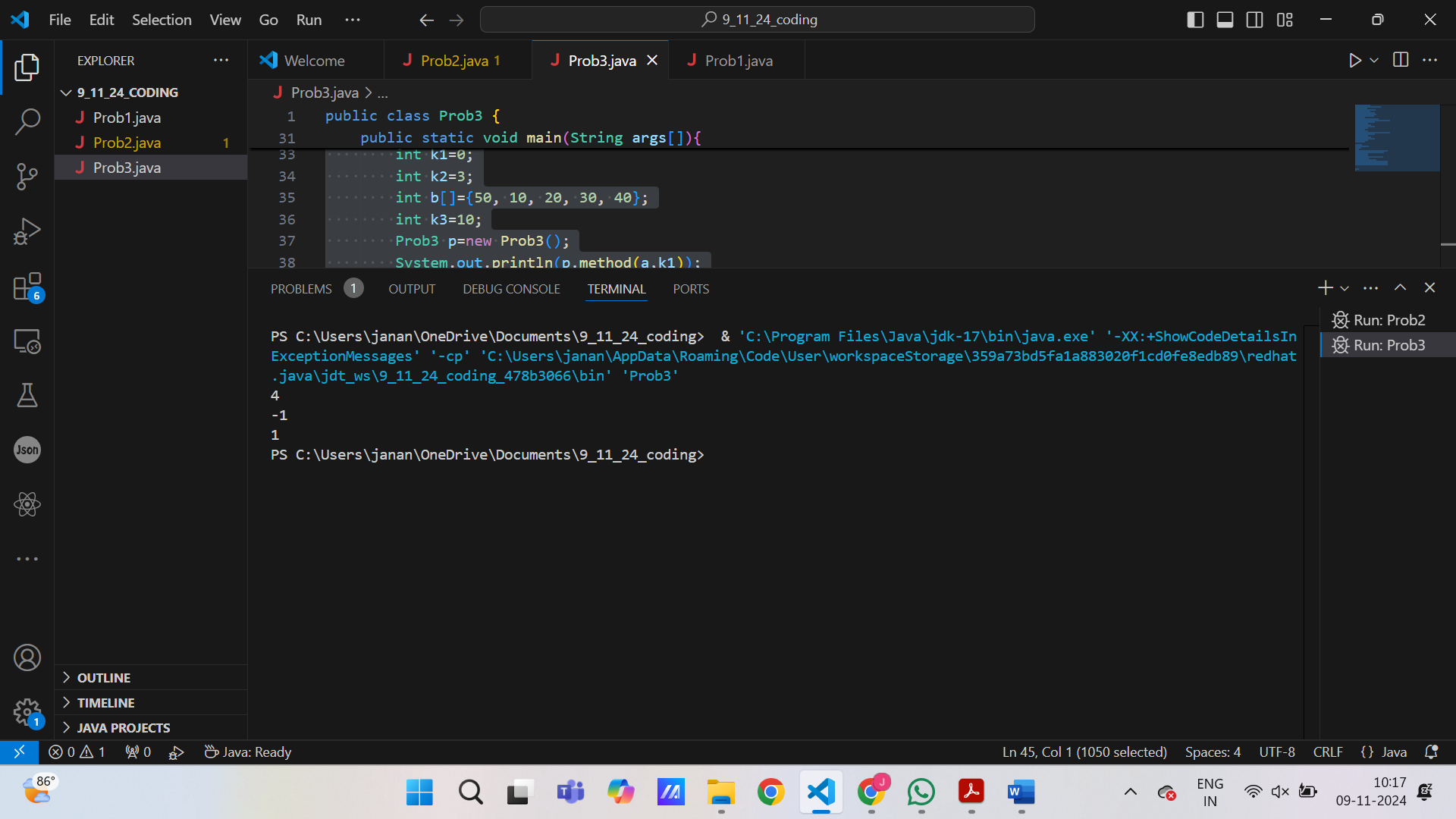
        System.out.println(p.method(a,k2));

        System.out.println(p.method(b,k3));

    }

}

**Output:**



**TIME COMPLEXITY:** O(logn)

**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**

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

**Output: 12**

**Explanation:**

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

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

**CODE:**

import java.util.\*;

public class Prob4 {

    int method(int arr[]){

        int i=0;

        int j=arr.length-1;

        int curr\_vol=1;

        int max\_vol=Integer.MIN\_VALUE;

        while(i!=j){

                curr\_vol=(j-i)\*Math.min(arr[i],arr[j]);

                max\_vol=Math.max(curr\_vol,max\_vol);

                if(arr[i]<arr[j]){

                    i++;

                }

                else{

                    j--;

                }

            }

        return max\_vol;

        }

    public static void main(String args[]){

        int a[]={1, 5, 4, 3};

        int b[]={3, 1, 2, 4, 5};

        Prob4 p=new Prob4();

        System.out.println(p.method(a));

        System.out.println(p.method(b));

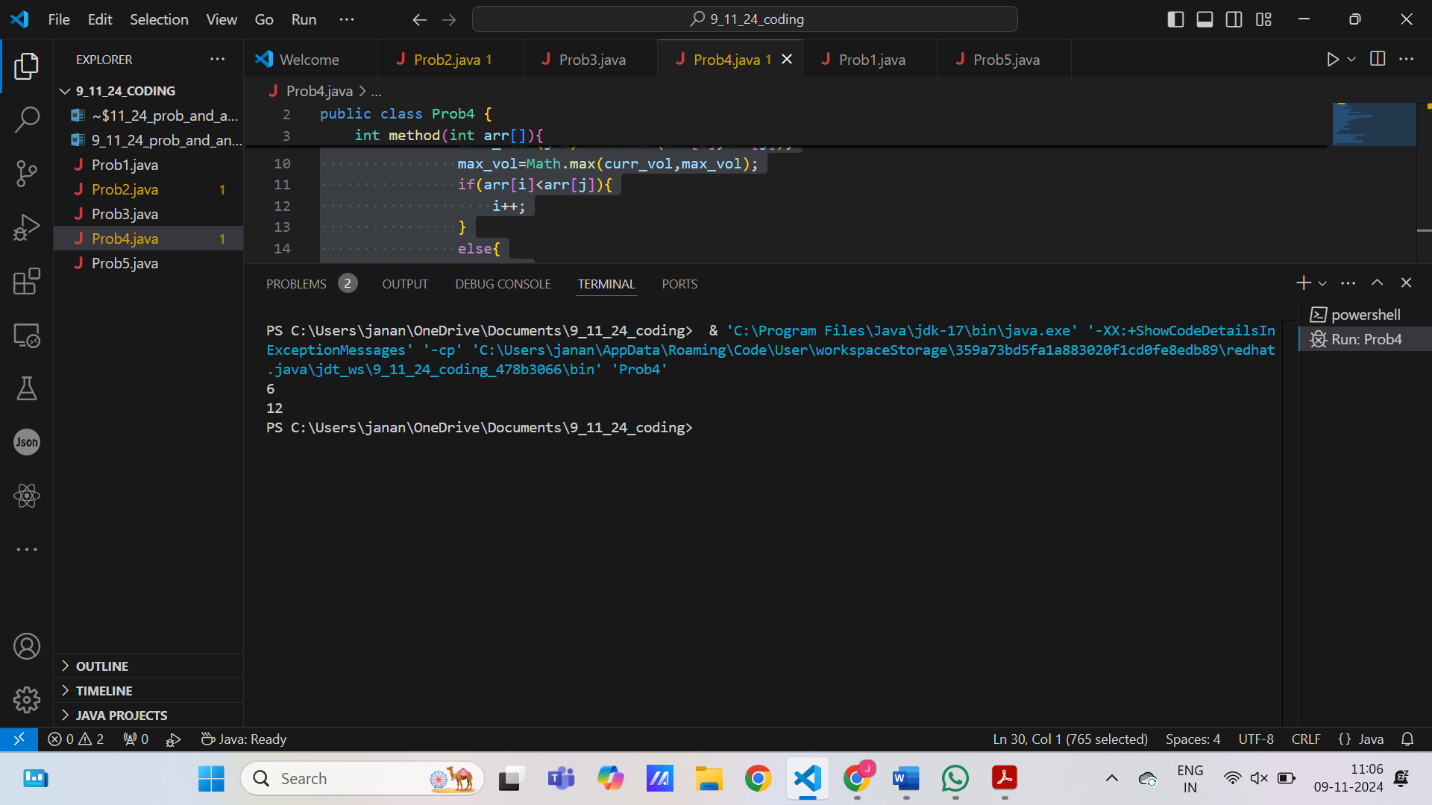
    }

}

**TIME COMPLEXITY:** O(n)

**SPACE COMPLEXITY:** O(1)

**OUTPUT:**



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

**Input: 100**

**Output: 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000**

**Input: 50**

**Output: 30414093201713378043612608166064768844377641568960512000000000000**

**CODE:**

import java.math.BigInteger;

public class Prob5 {

    BigInteger method(int n){

        BigInteger start=BigInteger.ONE;

        for(int i=2;i<n+1;i++){

            start=start.multiply(BigInteger.valueOf(i));

        }

        return start;

    }

    public static void main(String args[]){

        int n1=100;

        int n2=50;

        int n3=700;

        Prob5 p=new Prob5();

        System.out.println(p.method(n1));

        System.out.println();

        System.out.println(p.method(n2));

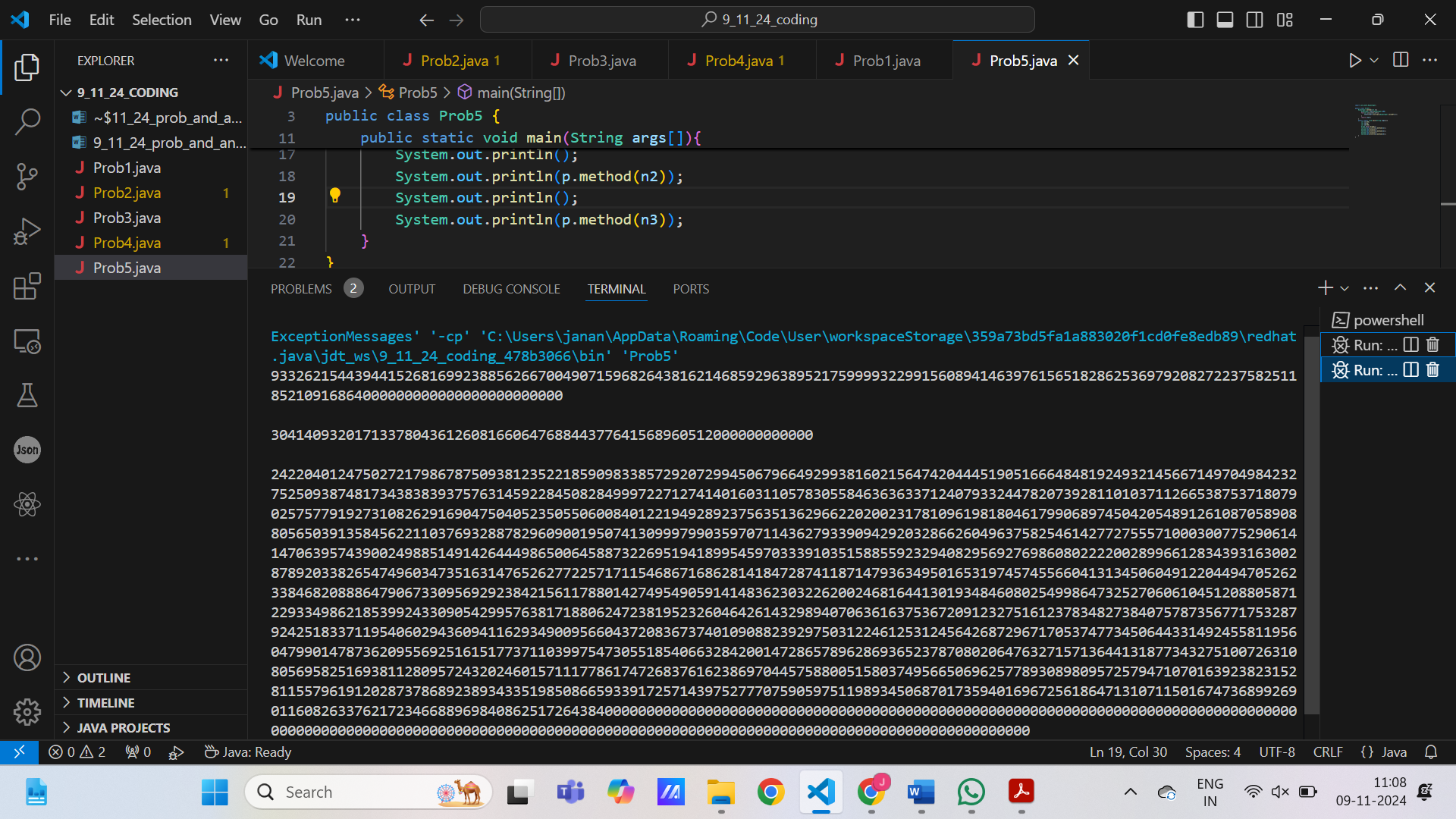
        System.out.println();

        System.out.println(p.method(n3));

    }

}

**OUTPUT:**



**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**

**CODE:**

public class Prob6 {

    int method(int[] arr) {

        int n=arr.length;

        int l\_m[]=new int[n];

        l\_m[0]=arr[0];

        for(int i=1;i<n;i++){

            l\_m[i]=Math.max(l\_m[i-1],arr[i]);

        }

        int r\_m[]=new int[n];

        r\_m[n-1]=arr[n-1];

        for(int i=n-2;i>=0;i--){

            r\_m[i]=Math.max(arr[i],r\_m[i+1]);

        }

        int t\_w=0;

        for(int i=0;i<n;i++){

            int w\_l=Math.min(l\_m[i],r\_m[i]);

            t\_w += w\_l-arr[i];

        }

        return t\_w;

    }

    public static void main(String[] args) {

        Prob6 p=new Prob6();

        int[] arr1={3, 0, 1, 0, 4, 0, 2};

        int[] arr2={3, 0, 2, 0, 4};

        int[] arr3={1, 2, 3, 4};

        int[] arr4={10, 9, 0, 5};

        System.out.println(p.method(arr1));

        System.out.println(p.method(arr2));

        System.out.println(p.method(arr3));

        System.out.println(p.method(arr4));

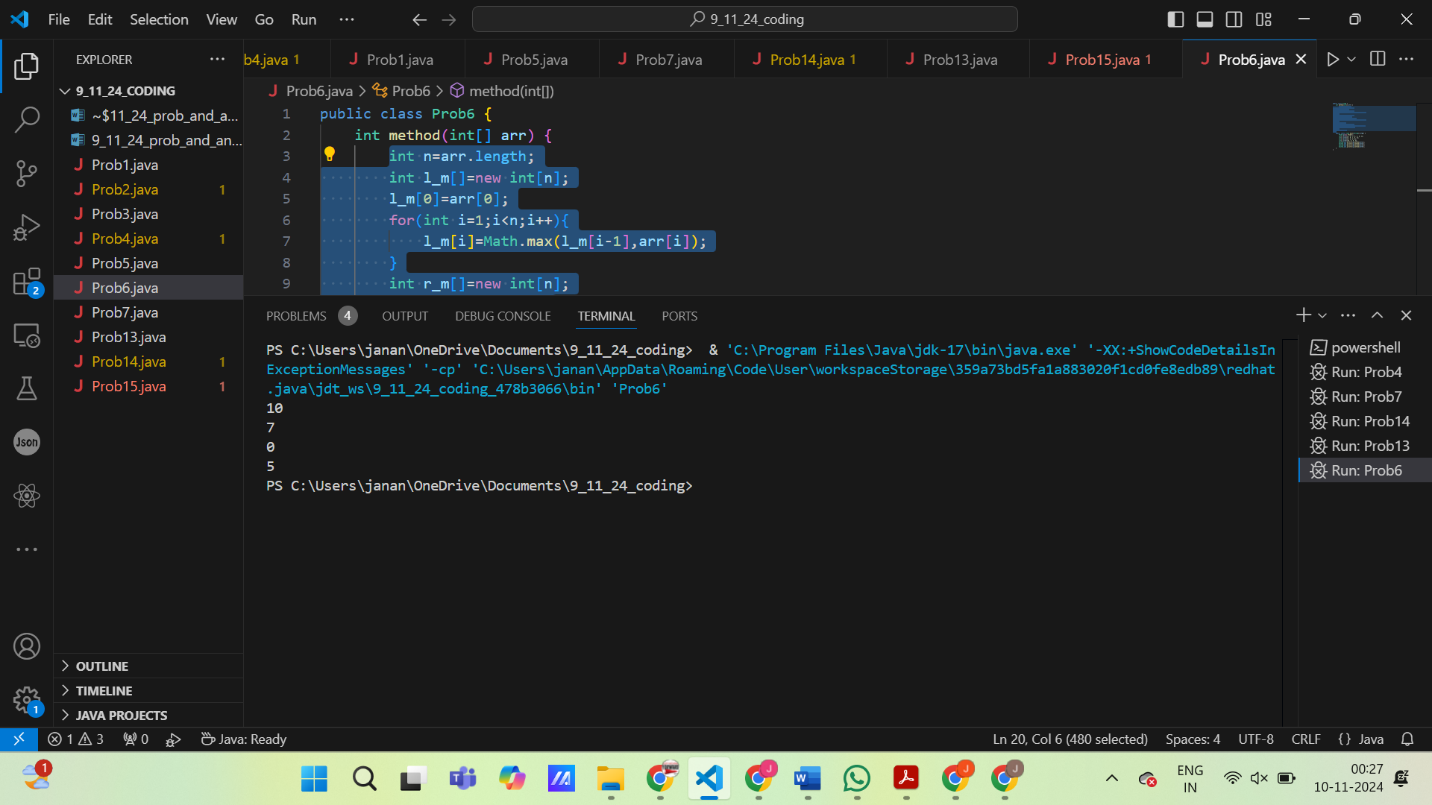
    }

}

**TIME COMPLEXITY:** O(n)

**SPACE 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.**

**CODE:**import java.util.Arrays;

public class Prob7 {

    int method(int[] arr,int m){

        int n=arr.length;

        int diff;

        int min\_diff=Integer.MAX\_VALUE;

        for(int i=0;i<n;i++){

            if((i+m)>=n){

                return min\_diff;

            }

            int[] a=Arrays.copyOfRange(arr, i,i+m);

            diff=Arrays.stream(a).max().getAsInt()-Arrays.stream(a).min().getAsInt();

            min\_diff=Math.min(diff,min\_diff);

        }

        return min\_diff;

    }

    public static void main(String  args[]){

        int[] a={7, 3, 2, 4, 9, 12, 56};

        int m1=3;

        int m2=5;

        Prob7 p=new Prob7();

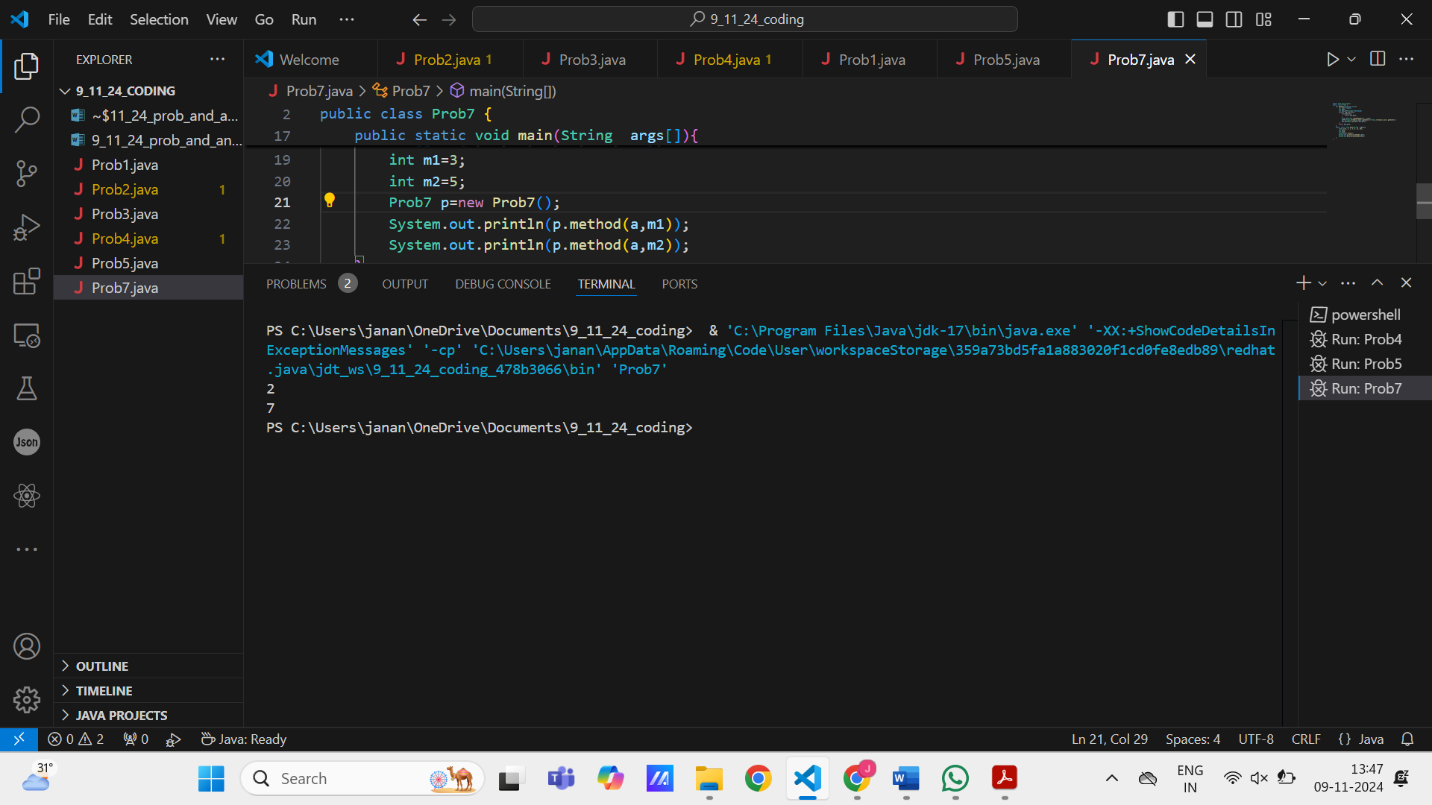
        System.out.println(p.method(a,m1));

        System.out.println(p.method(a,m2));

    }

}

**OUTPUT:**

****

**TIME COMPLEXITY:** O(n×m) **SPACE COMPLEXITY:** O(m)

**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.\*;

public class Prob8 {

    int[][] method(int[][] arr) {

        Arrays.sort(arr, Comparator.comparingInt(a -> a[0]));

        Stack<int[]> stack = new Stack<int[]>();

        stack.push(new int[]{arr[0][0], arr[0][1]});

        for(int i = 1; i<arr.length;i++){

            int[] u = stack.peek();

            if(arr[i][0] <= u[1]) {

                int[] new\_i = new int[]{u[0], Math.max(arr[i][1], u[1])};

                stack.pop();

                stack.push(new\_i);

            } else {

                stack.push(new int[]{arr[i][0], arr[i][1]});

            }

        }

        return stack.toArray(new int[stack.size()][]);

    }

    public static void main(String[] args) {

        Prob8 p=new Prob8();

        int[][] arr1={{1, 3}, {2, 4}, {6, 8}, {9, 10}};

        int[][] arr2={{7, 8}, {1, 5}, {2, 4}, {4, 6}};

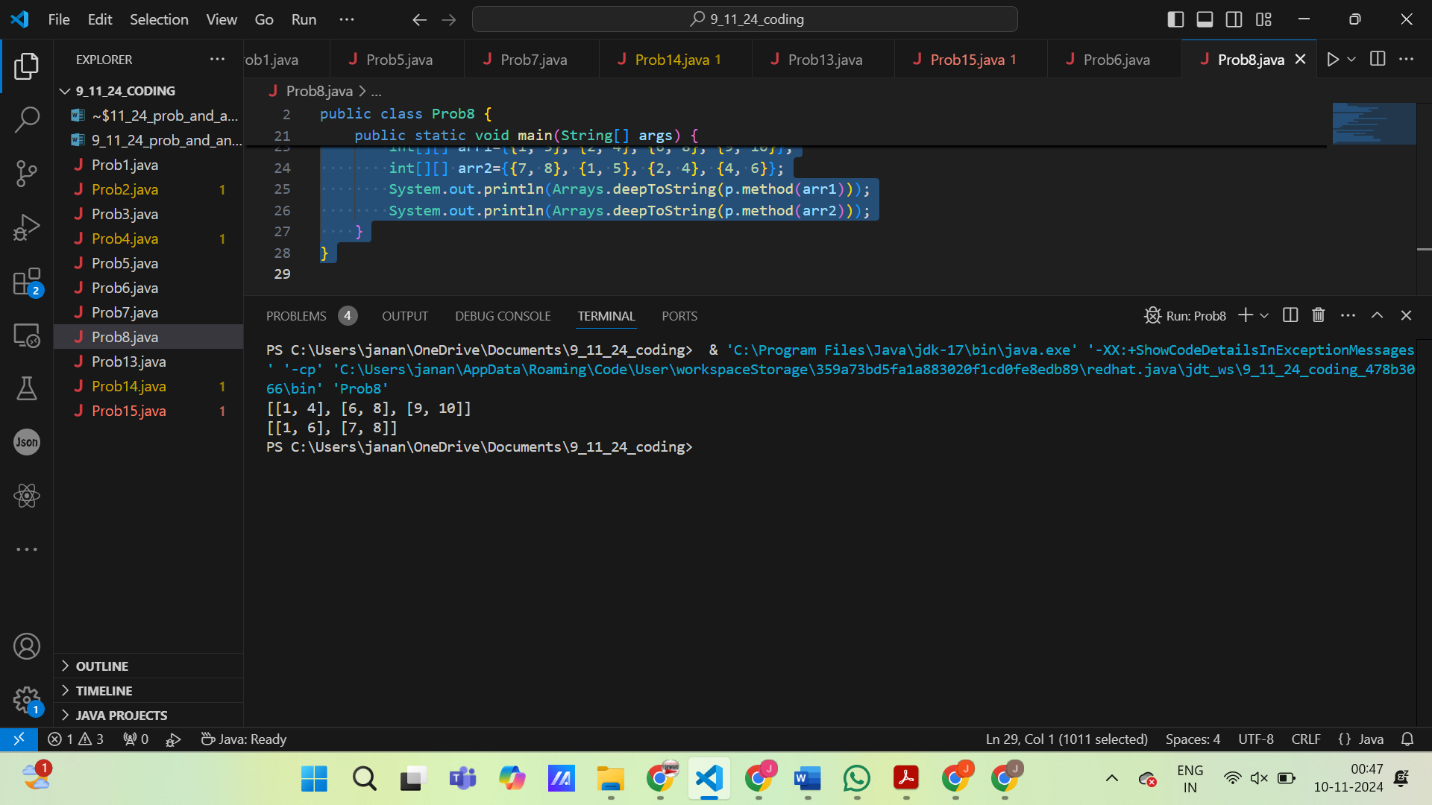
        System.out.println(Arrays.deepToString(p.method(arr1)));

        System.out.println(Arrays.deepToString(p.method(arr2)));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(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}}**

**CODE:**

import java.util.Arrays;

public class Prob9 {

    int[][] method(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;

        return mat;

    }

    public static void main (String args[]){

        Prob9 p = new Prob9();

        int[][] mat1={{1, 0},{0, 0}};

        int[][] mat2={{0, 0, 0},{0, 0, 1}};

        int[][] mat3={{1, 0, 0, 1},{0, 0, 1, 0},{0, 0, 0, 0}};

        System.out.println(Arrays.deepToString(p.method(mat1)));

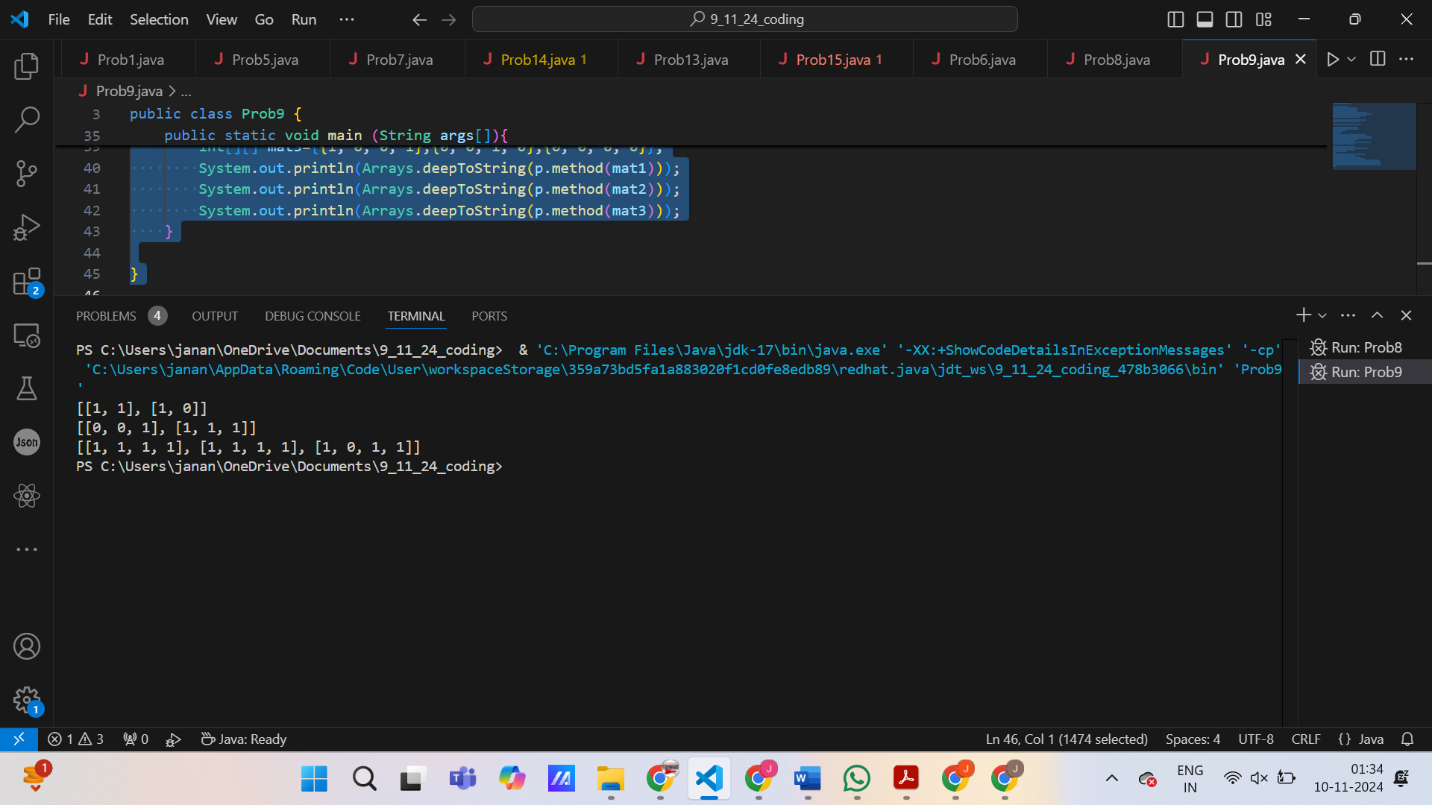
        System.out.println(Arrays.deepToString(p.method(mat2)));

        System.out.println(Arrays.deepToString(p.method(mat3)));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n^2)

**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.**

**CODE:**

import java.util.\*;

public class Prob10 {

    List<Integer> method(int[][] matrix) {

        int d = matrix.length-1;

        int r = matrix[0].length-1;

        int u=0,l=0,i=0,j=0,n=0;

        int a = (r+1)\*(d+1);

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

        while(true){

            while(j<=r){

                list.add(matrix[i][j]);

                n++;

                j++;

            }

            if(n==a){

                break;

            }

            u++;

            j--;

            i=u;

            while(i<=d){

                list.add(matrix[i][j]);

                n++;

                i++;

            }

            if(n==a){

                break;

            }

            r--;

            i--;

            j=r;

            while(j>=l){

                list.add(matrix[i][j]);

                n++;

                j--;

            }

            if(n==a){

                break;

            }

            d--;

            j++;

            i=d;

            while(i>=u){

                list.add(matrix[i][j]);

                n++;

                i--;

            }

            if(n==a){

                break;

            }

            l++;

            i++;

            j=l;

        }

        return list;

    }

    public static void main(String[] args) {

        Prob10 p=new Prob10();

        int[][] m1 = {{1, 2, 3, 4},

        {5, 6, 7, 8},

        {9, 10, 11, 12},

        {13, 14, 15, 16 }};

        int[][] m2 = { {1, 2, 3, 4, 5, 6},

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

{13, 14, 15, 16, 17, 18}};

        System.out.println(p.method(m1));

        System.out.println(p.method(m2));

    }

}

**OUTPUT:**

****

**TIME COMPLEXITY:** O(m x n)

**SPACE COMPLEXITY:** O(m x n)

**13. Check if given Parentheses expression is balanced or not**

**Given a string str of length N, consisting of „(„ and „)„ only, the task is to check whether it is balanced or not.**

**Input: str = “((()))()()”**

**Output: Balanced**

**Input: str = “())((())”**

**Output: Not Balanced**

**CODE:**public class Prob13 {

    String method(String s){

        int n=s.length();

        int l=0;

        int r=0;

        for(int i=0;i<n;i++){

            if(s.charAt(i)=='('){

                l++;

            }

            if(s.charAt(i)==')'){

                r++;

            }

            if (r>l){

                return "NOT BALANCED";

            }

        }

        return "BALANCED";

    }

    public static void main(String args[]){

        Prob13 p=new Prob13();

        String s1="((()))()()";

        String s2="())((())";

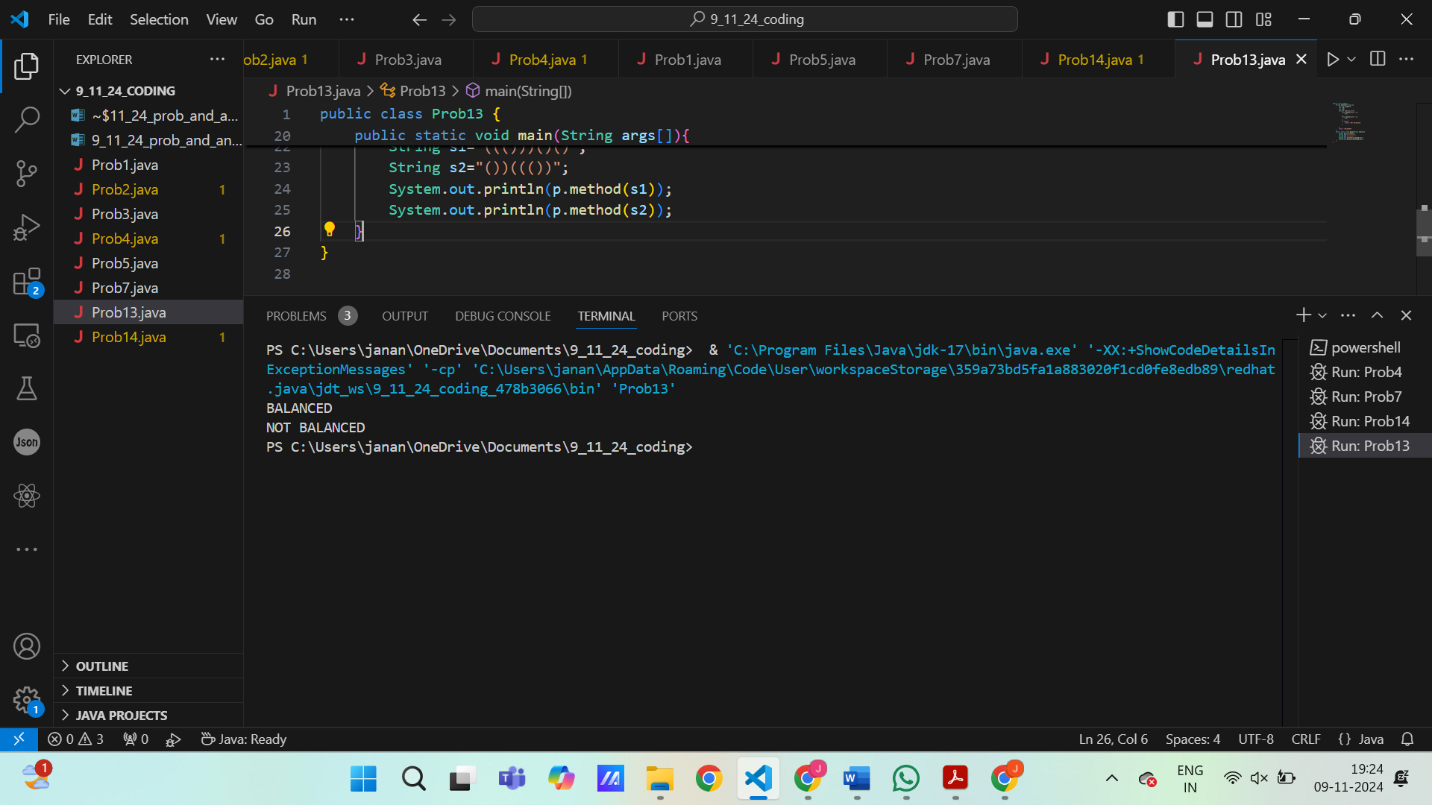
        System.out.println(p.method(s1));

        System.out.println(p.method(s2));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n)

**SPACE COMPLEXITY**: O(1)

**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.**

**CODE:**

import java.util.\*;

import java.lang.\*;

public class Prob14 {

    boolean method(String s1,String s2){

        HashMap<Character,Integer> h1= new HashMap<>();

        HashMap<Character,Integer> h2= new HashMap<>();

        int n1=s1.length();

        int n2=s2.length();

        if(n1!=n2){

            return false;

        }

        for(int i=0;i<n1;i++){

            h1.put(s1.charAt(i),h1.getOrDefault(h1,0)+1);

        }

        for(int i=0;i<n2;i++){

            h2.put(s2.charAt(i),h2.getOrDefault(h2,0)+1);

        }

        return h1.equals(h2);

    }

    public static void main(String args[]){

        Prob14 p=new Prob14();

        String s1="geeks";

        String s2="kseeg";

        String s3="allergy";

        String s4="allergic";

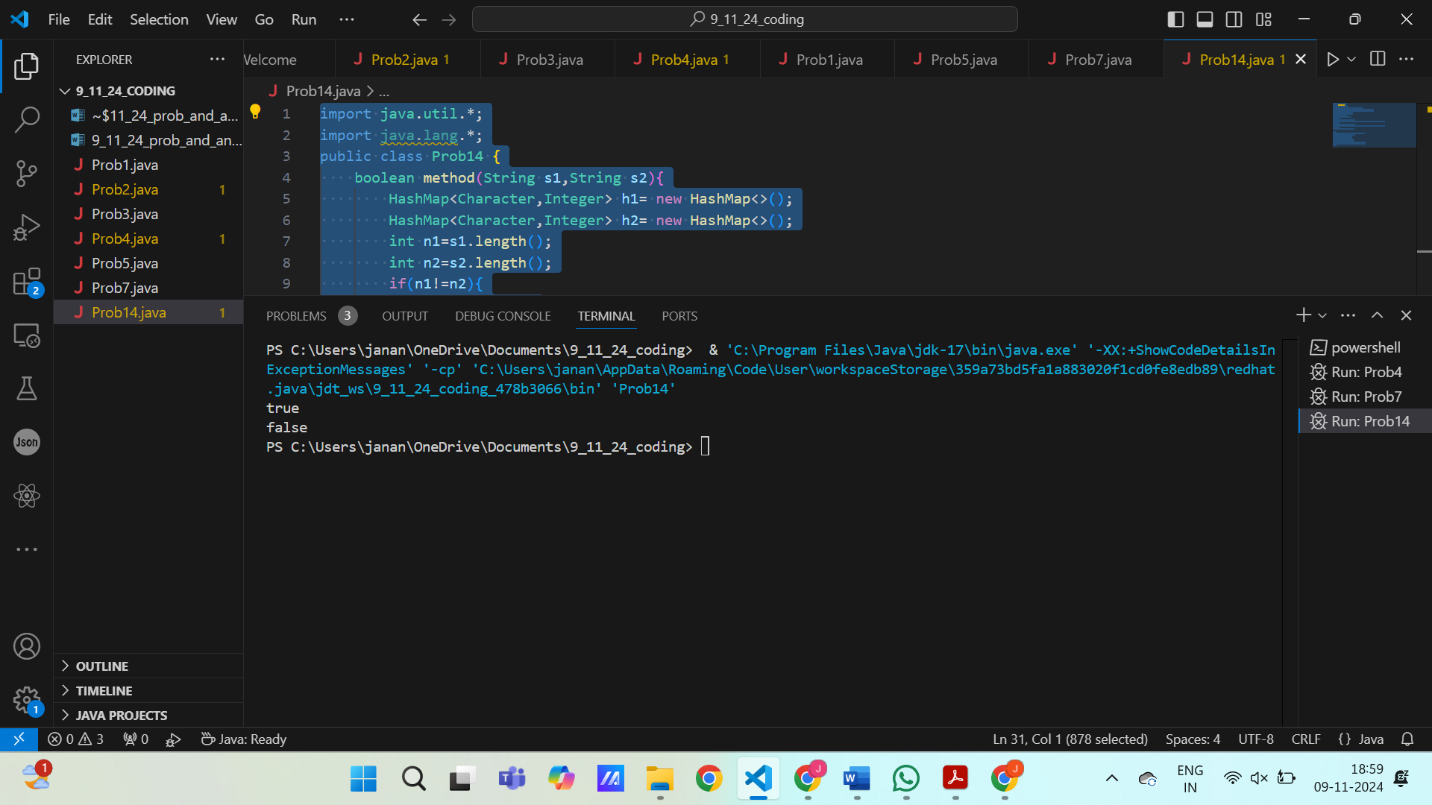
        System.out.println(p.method(s1,s2));

        System.out.println(p.method(s3,s4));

    }

}

**OUTPUT:**

****

**TIME COMPLEXITY:** O(n)

**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 = “**

**CODE:**

import java.util.\*;

public class Prob15 {

        public String method(String s) {

            if (s.length() <= 1) {

                return s;

            }

            String maxStr = s.substring(0, 1);

            for (int i = 0; i < s.length() - 1; i++) {

                String odd = expandFromCenter(s, i, i);

                String even = expandFromCenter(s, i, i + 1);

                if (odd.length() > maxStr.length()) {

                    maxStr = odd;

                }

                if (even.length() > maxStr.length()) {

                    maxStr = even;

                }

            }

            return maxStr;

        }

        private String expandFromCenter(String s, int left, int right) {

            while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {

                left--;

                right++;

            }

            return s.substring(left + 1, right);

        }

    public static void main(String args[]){

        Prob15 p=new Prob15();

        String str1 = "forgeeksskeegfor";

        String str2 = "Geeks";

        String str3 = "abc";

        String str4="";

        System.out.println(p.method(str1));

        System.out.println(p.method(str2));

        System.out.println(p.method(str3));

        System.out.println(p.method(str4));

    }

}

**OUTPUT:**



**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.**

**CODE:**

public class Prob16 {

    String method(String[] arr){

        if (arr == null || arr.length == 0) return "";

        String P\_f = arr[0];

        for (int i = 1; i < arr.length; i++) {

            while (!arr[i].startsWith(P\_f)) {

                P\_f = P\_f.substring(0, P\_f.length() - 1);

                if (P\_f.isEmpty()) return "-1";

            }

        }

        return P\_f;

    }

    public static void main(String[] args) {

        Prob16 p=new Prob16();

        String arr1[] = {"geeksforgeeks", "geeks", "geek", "geezer"};

        String arr2[] = {"hello", "world"};

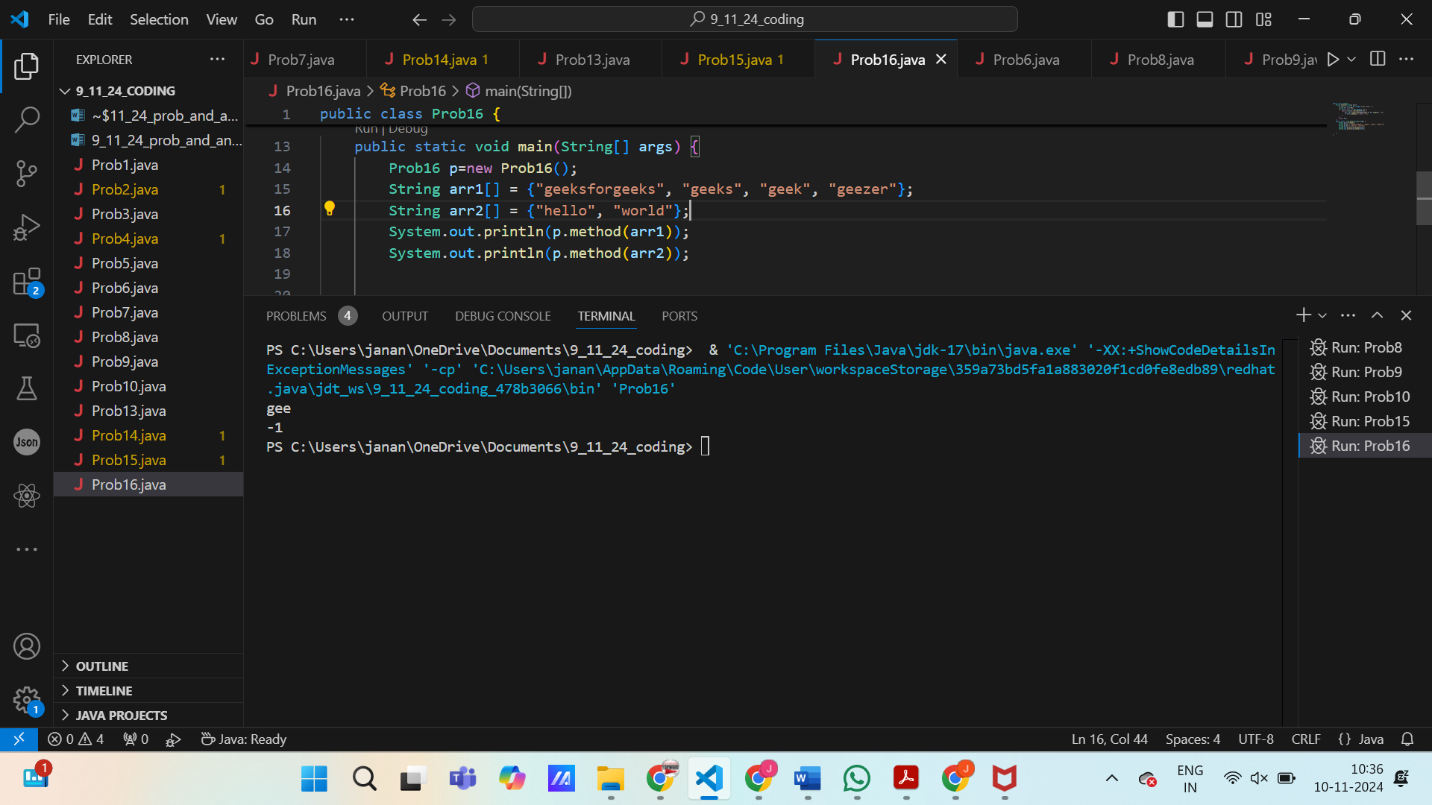
        System.out.println(p.method(arr1));

        System.out.println(p.method(arr2));

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n x 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]**

**CODE:**

import java.util.Stack;

public class Prob17 {

    private void deleteMiddle(Stack<Integer> stack, int currentIndex, int middleIndex) {

        if (currentIndex == middleIndex) {

            stack.pop();

            return;

        }

        int topElement = stack.pop();

        deleteMiddle(stack, currentIndex + 1, middleIndex);

        stack.push(topElement);

    }

    public void deleteMiddleElement(Stack<Integer> stack) {

        int middleIndex = stack.size() / 2;

        deleteMiddle(stack, 0, middleIndex);

    }

    public static void main(String[] args) {

        Prob17 p=new Prob17();

        Stack<Integer> s1 = new Stack<>();

        s1.push(1);

        s1.push(2);

        s1.push(3);

        s1.push(4);

        s1.push(5);

        p.deleteMiddleElement(s1);

        System.out.println("After deleting middle element: " + s1);

        Stack<Integer> s2 = new Stack<>();

        s2.push(1);

        s2.push(2);

        s2.push(3);

        s2.push(4);

        s2.push(5);

        s2.push(6);

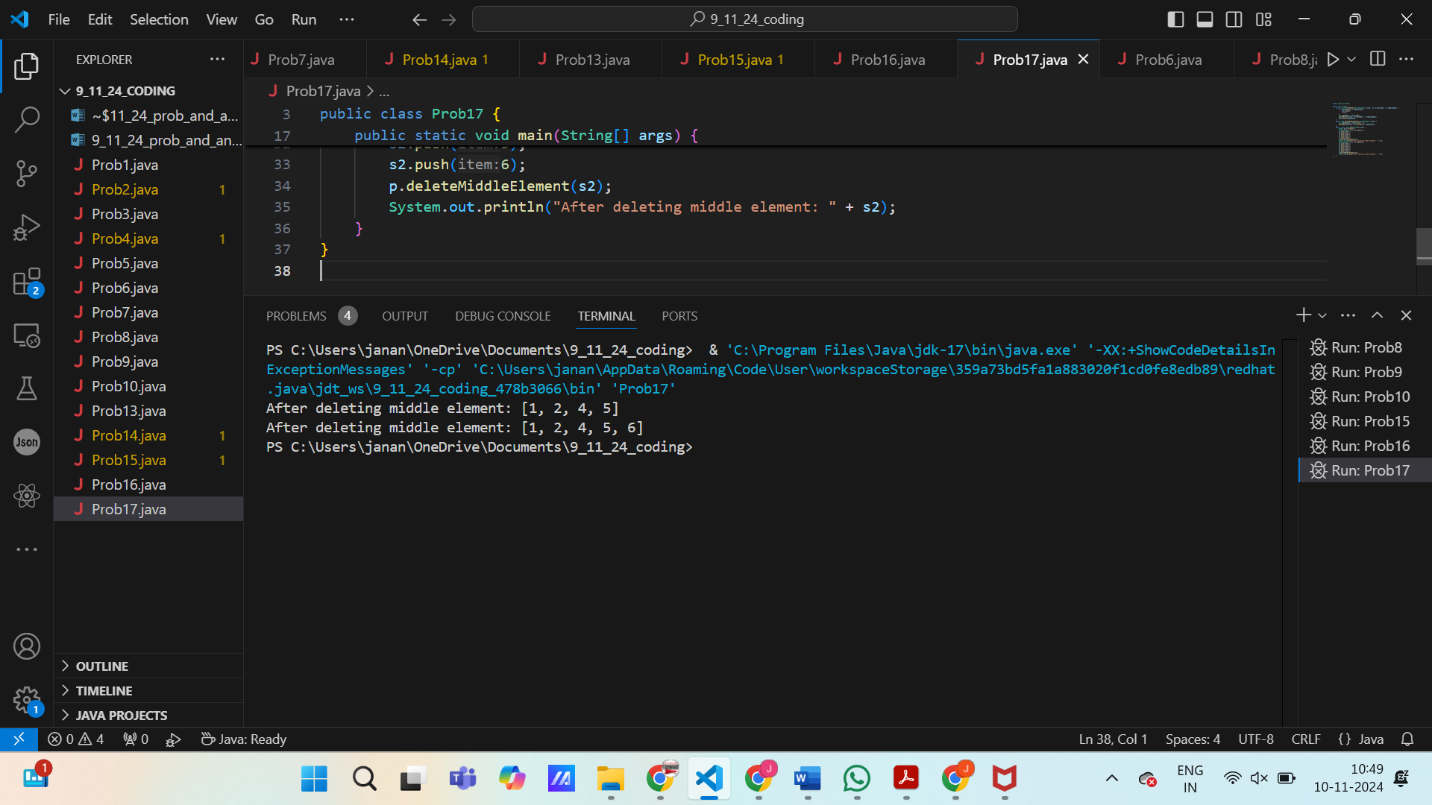
        p.deleteMiddleElement(s2);

        System.out.println("After deleting middle element: " + s2);

    }

}

**OUTPUT:**

****

**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**

**CODE:**

import java.util.Arrays;

import java.util.Stack;

public class Prob18 {

    int[] method ( 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]);

        }

        return nge;

    }

    public static void main(String args[]){

        Prob18 prob = new Prob18();

        int[] arr1 = {4, 5, 2, 25};

        int[] r1 = prob.method (arr1);

        System.out.println("Array: " + Arrays.toString(arr1));

        for (int i = 0; i < arr1.length; i++) {

            System.out.println(arr1[i] + " -> " + r1[i]);

        }

        int[] arr2 = {13, 7, 6, 12};

        int[] r2 = prob.method (arr2);

        System.out.println("\nArray: " + Arrays.toString(arr2));

        for (int i = 0; i < arr2.length; i++) {

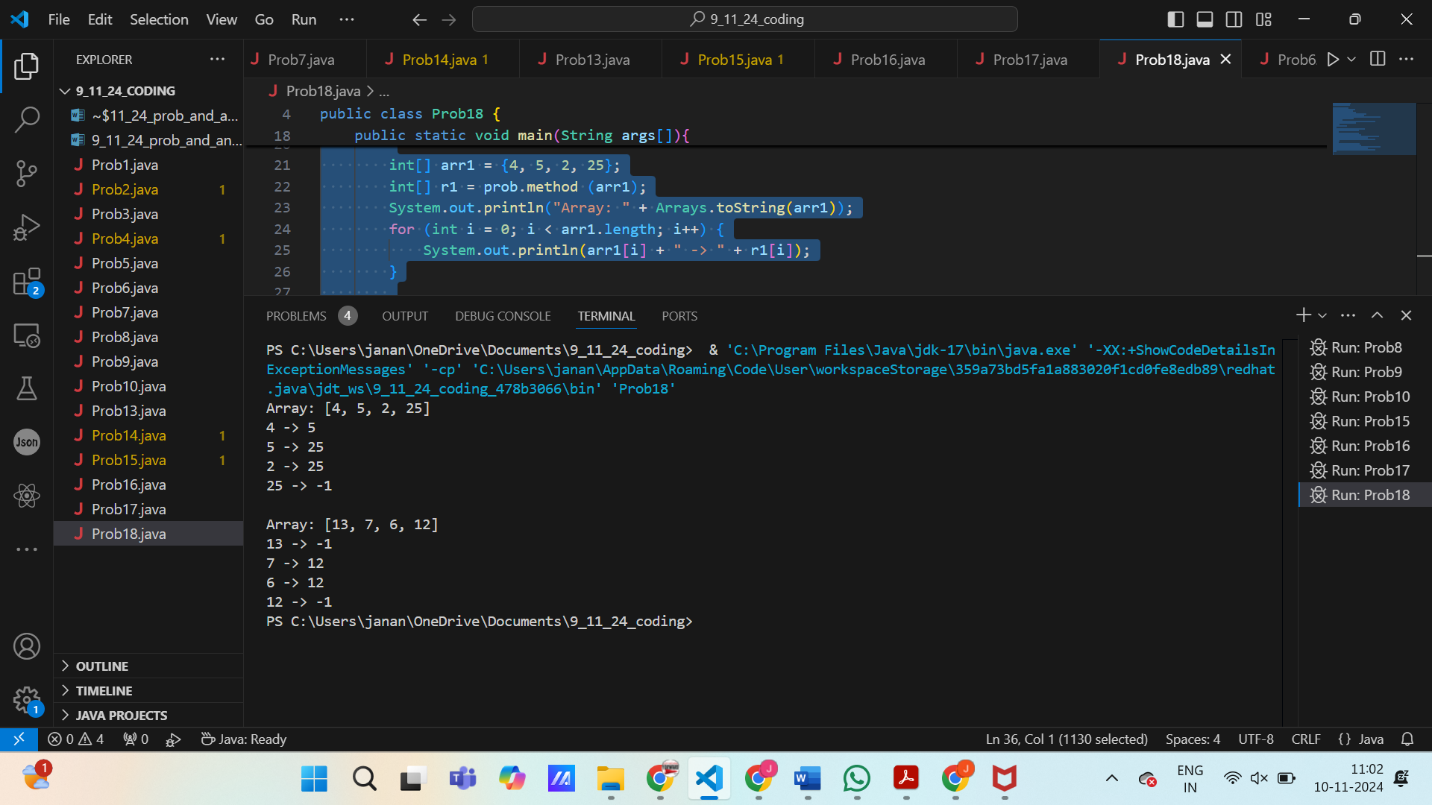
            System.out.println(arr2[i] + " -> " + r2[i]);

    }

}

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n)

**SPACE 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, right;

    public TreeNode(int val) {

        this.val = val;

        left = right = null;

    }

}

public class Prob19{

    public List<Integer> rightView(TreeNode root) {

        List<Integer> result = new ArrayList<>();

        if (root == null) return result;

        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) {

                    result.add(node.val);

                }

                if (node.left != null) {

                    queue.add(node.left);

                }

                if (node.right != null) {

                    queue.add(node.right);

                }

            }

        }

        return result;

    }

    public static void main(String[] args) {

        Prob19 tree = new Prob19();

        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);

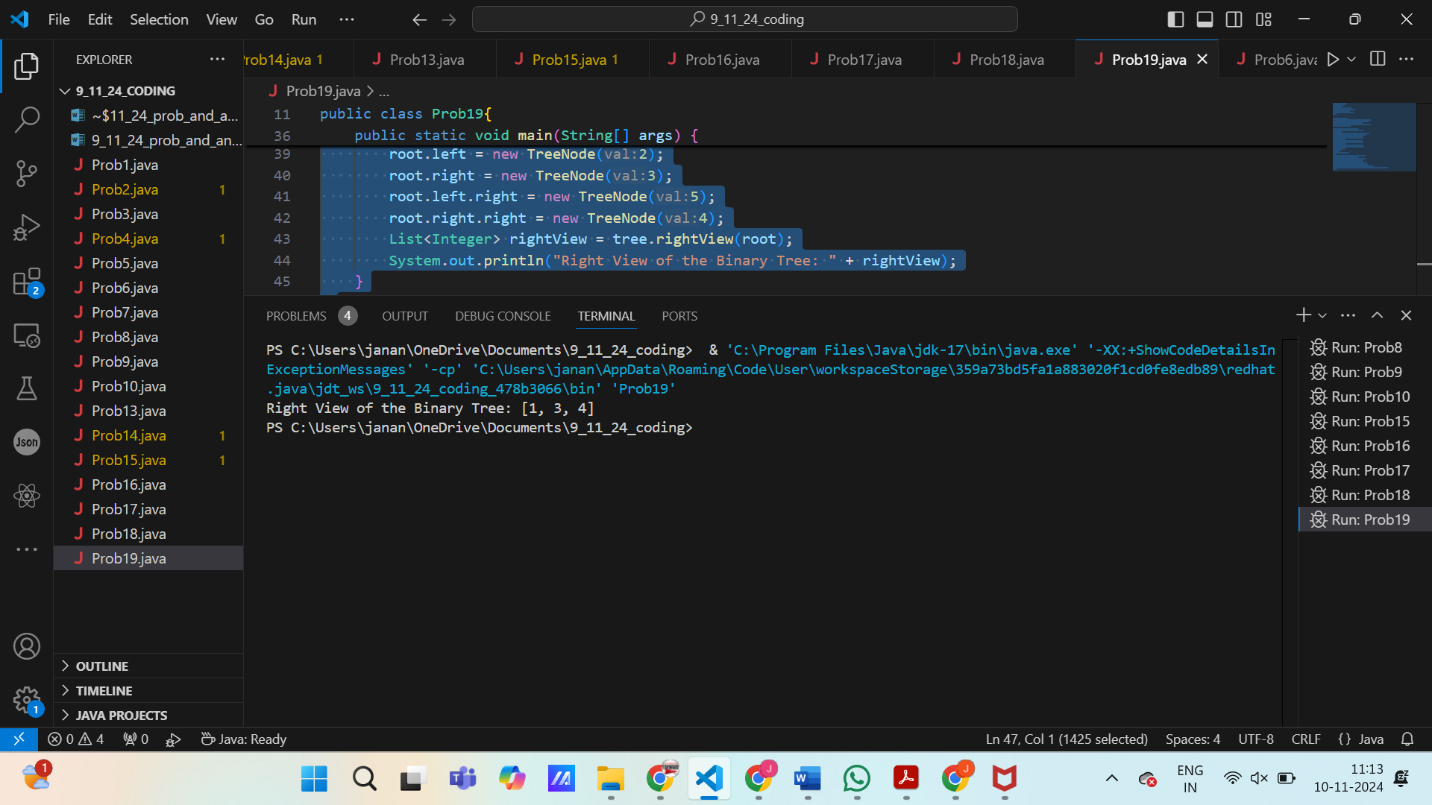
        List<Integer> rightView = tree.rightView(root);

        System.out.println("Right View of the Binary Tree: " + rightView);

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n)

**SPACE 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 val;

    TreeNode left, right;

    public TreeNode(int val) {

        this.val = val;

        left = right = null;

    }

}

public class Prob20 {

    public int maxDepth(TreeNode root) {

        if (root == null) return 0;

        int leftDepth = maxDepth(root.left);

        int rightDepth = maxDepth(root.right);

        return 1 + Math.max(leftDepth, rightDepth);

    }

    public static void main(String[] args) {

        Prob20 tree = new Prob20();

        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.right.right.left = new TreeNode(7);

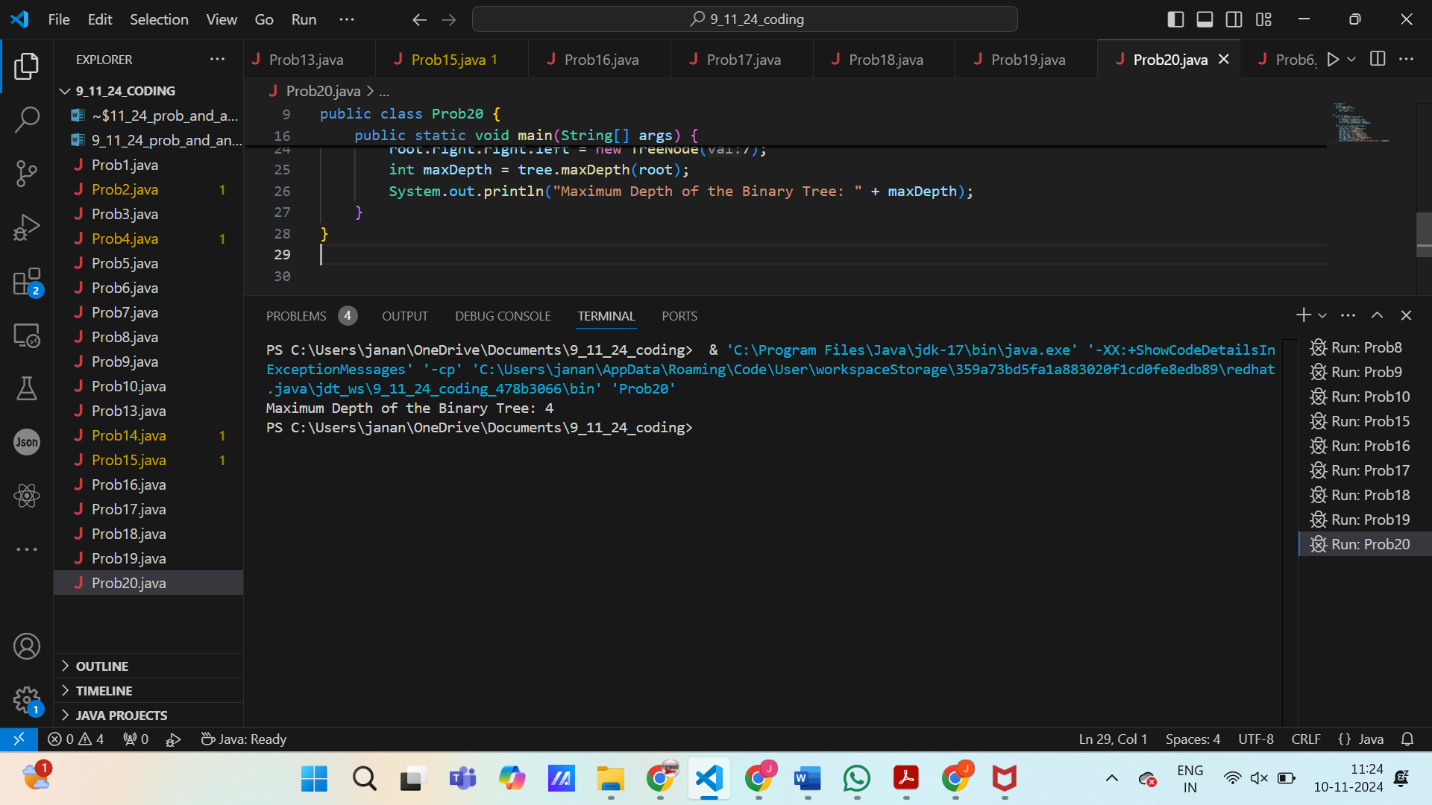
        int maxDepth = tree.maxDepth(root);

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

    }

}

**OUTPUT:**



**TIME COMPLEXITY:** O(n)

**SPACE COMPLEXITY:** O(h) h-height of the tree