**DSA PRACTICE - I**

**09/11/24**

**1.MAXIMUM SUM SUBARRAY:**

import java.util.\*;

public class MaximumSubarray {

    static int maxisub(int[] arr) {

        int maxEndingHere = arr[0];

        int maxSoFar = arr[0];

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

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

            maxSoFar = Math.max(maxSoFar, maxEndingHere);

        }

        return maxSoFar;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

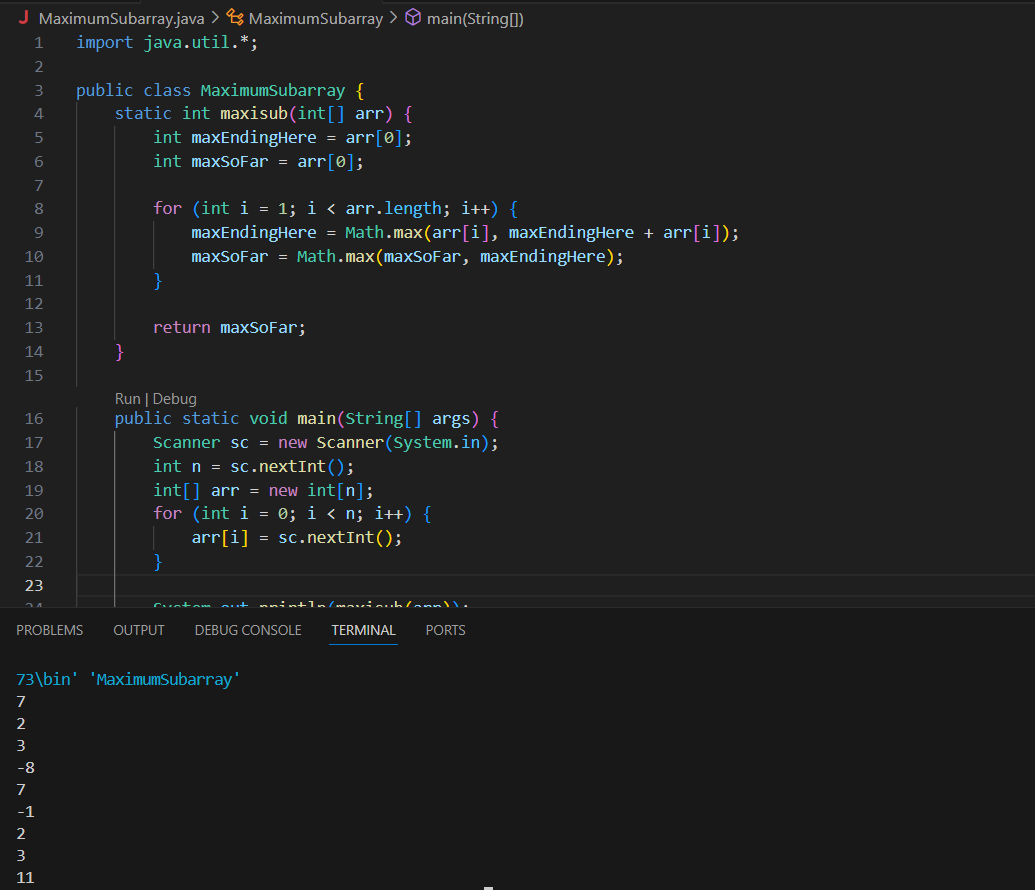
        }

        System.out.println(maxisub(arr));

        sc.close();

    }

}



**Time Complexity : O(n)**

**Space Complexity: O(1)**

**2. MAXIMUM PRODUCT SUBARRAY**

import java.util.\*;

public class MaximumProduct {

    static int maxProduct(int[] arr) {

        int maxP = arr[0];

        int minP = arr[0];

        int res = arr[0];

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

            int temp = maxP;

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

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

            res = Math.max(res, maxP);

        }

        return res;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        System.out.println(maxProduct(arr));

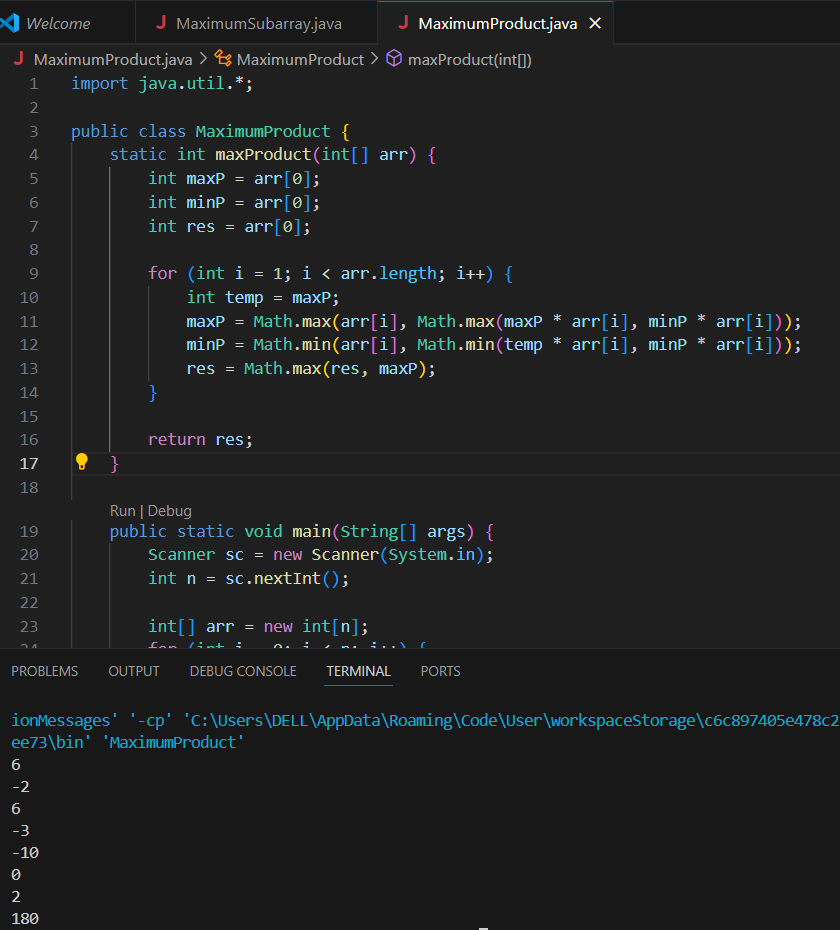
        sc.close();

    }

}

**Time Complexity: O(n)**

**Space Complexity: O(1)**



**3. SEARCH IN SORTED AND ROTATED ARRAY**

import java.util.\*;

public class SearchInRotatedArray {

    static int search(int[] arr, int key) {

        int left = 0, right = arr.length - 1;

        while (left <= right) {

            int mid = left + (right - left) / 2;

            if (arr[mid] == key) return mid;

            if (arr[left] <= arr[mid]) {

                if (arr[left] <= key && key < arr[mid]) {

                    right = mid - 1;

                } else {

                    left = mid + 1;

                }

            }

            else {

                if (arr[mid] < key && key <= arr[right]) {

                    left = mid + 1;

                } else {

                    right = mid - 1;

                }

            }

        }

        return -1;

    }

    public static void main(String[] args) {

        Scanner sc= new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        int key = sc.nextInt();

        int result = search(arr, key);

        if (result != -1) {

            System.out.println("Key found at index: " + result);

        } else {

            System.out.println("Key not found in the array.");

        }

        sc.close();

    }

}



**Time Complexity : O(log n)**

**Space Complexity: O(1)**

**4. CONTAINER WITH MOST WATER**

import java.util.\*;

public class ContainerwithMostWater {

    static int maxArea(int[] arr) {

        int left = 0;

        int right = arr.length - 1;

        int maxArea = 0;

        while (left < right) {

            maxArea = Math.max(maxArea, (right - left) \* Math.min(arr[left], arr[right]));

            if (arr[left] < arr[right]) {

                left++;

            } else {

                right--;

            }

        }

        return maxArea;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

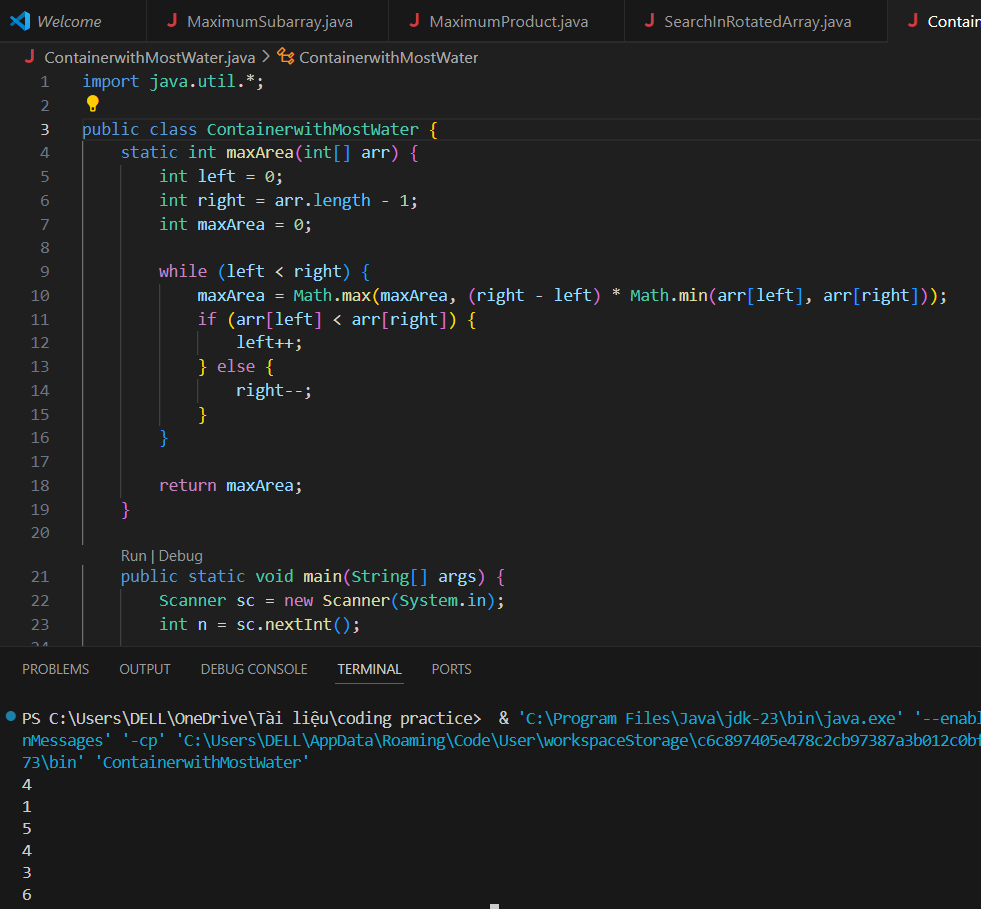
        }

        System.out.println(maxArea(arr));

        sc.close();

    }

}



**Time Complexity: O(n)**

**Space Complexity:O(1)**

**5. FACTORIAL OF LARGE NUMBER**

import java.util.\*;

import java.math.\*;

public class Factorial {

    static BigInteger factorial(int n) {

        BigInteger result = BigInteger.ONE;

        for (int i = 2; i <= n; i++) {

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

        }

        return result;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

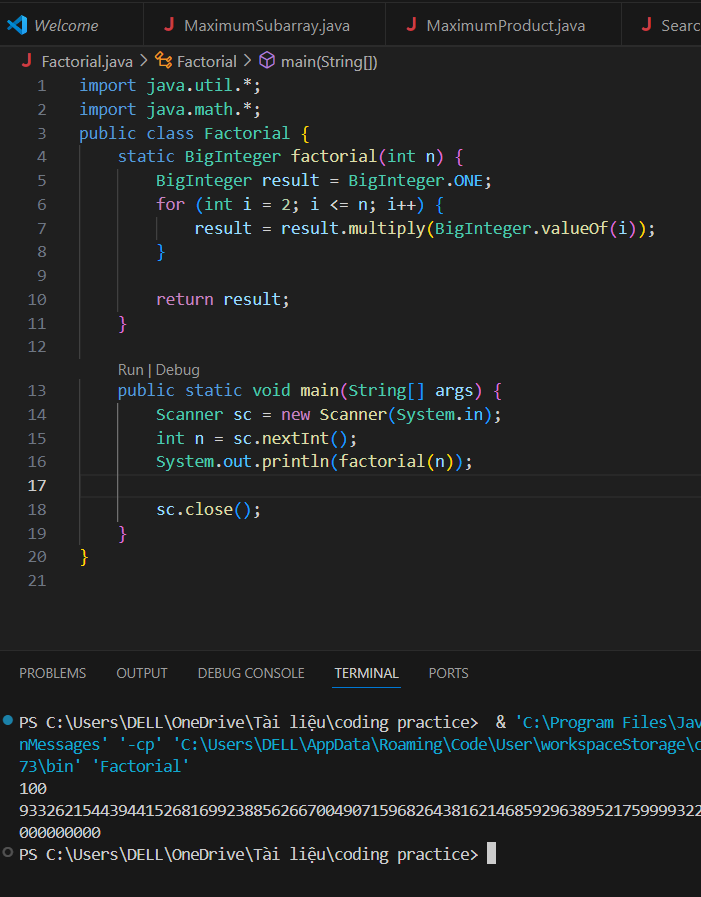
        int n = sc.nextInt();

        System.out.println(factorial(n));

        sc.close();

    }

}



**Time Complexity : O(n)**

**Space Complexity: O(1)**

**5. TRAPPING RAIN WATER**

import java.util.\*;

public class TrappingRainWater{

    static int maxArea(int[] arr){

        int left=0;

        int right=arr.length-1;

        int leftmax=arr[left];

        int rightmax=arr[right];

        int water=0;

        while(left<right){

            if(arr[left]<arr[right]){

                left++;

                leftmax=Math.max(leftmax, arr[left]);

                water += leftmax - arr[left];

            }

            else{

                right--;

                rightmax=Math.max(rightmax, arr[right]);

                water += rightmax - arr[right];

            }

        }

        return water;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        System.out.println(maxArea(arr));

        sc.close();

    }

}



**Time Complexity: O(n)**

**Space Complexity: O(1)**

**7. CHOCOLATE DISTRIBUTION PROBLEM**

import java.util.\*;

public class ChocolateProblem{

    static int distribution(int[] arr, int m){

        int n = arr.length;

        if(m>n){

            return -1;

        }

        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];

            minDiff = Math.min(minDiff, diff);

        }

        return minDiff;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

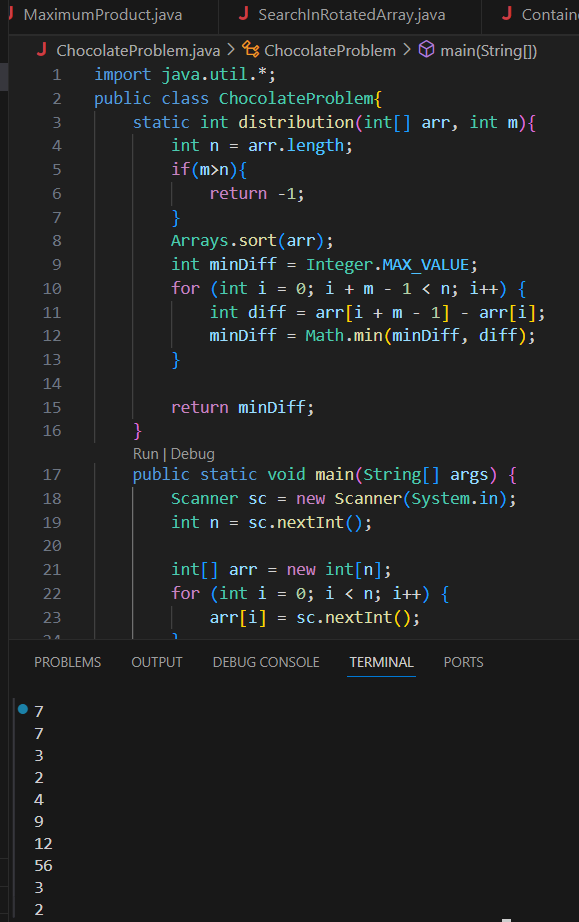
        int m = sc.nextInt();

        System.out.println(distribution(arr,m));

        sc.close();

    }

}



**Time Complexity: O(nlog(n))**

**Space Complexity: O(1)**

**8. MERGE INTERVAL**

import java.util.\*;

public class MergeIntervals {

    static int[][] merge(int[][] arr) {

        PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[0]));

        ArrayList<int[]> res = new ArrayList<>();

        for (int[] i : arr) {

            pq.add(i);

        }

        while (pq.size() > 1) {

            int[] first = pq.poll();

            int[] second = pq.poll();

            if (first[1] < second[0]) {

                res.add(first);

            } else {

                second[0] = Math.min(first[0], second[0]);

                second[1] = Math.max(first[1], second[1]);

            }

            pq.add(second);

        }

        while (!pq.isEmpty()) {

            res.add(pq.poll());

        }

        return res.toArray(new int[0][]);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[][] arr = new int[n][2];

        for (int i = 0; i < n; i++) {

            arr[i][0] = sc.nextInt();

            arr[i][1] = sc.nextInt();

        }

        int[][] mergedIntervals = merge(arr);

        for (int[] interval : mergedIntervals) {

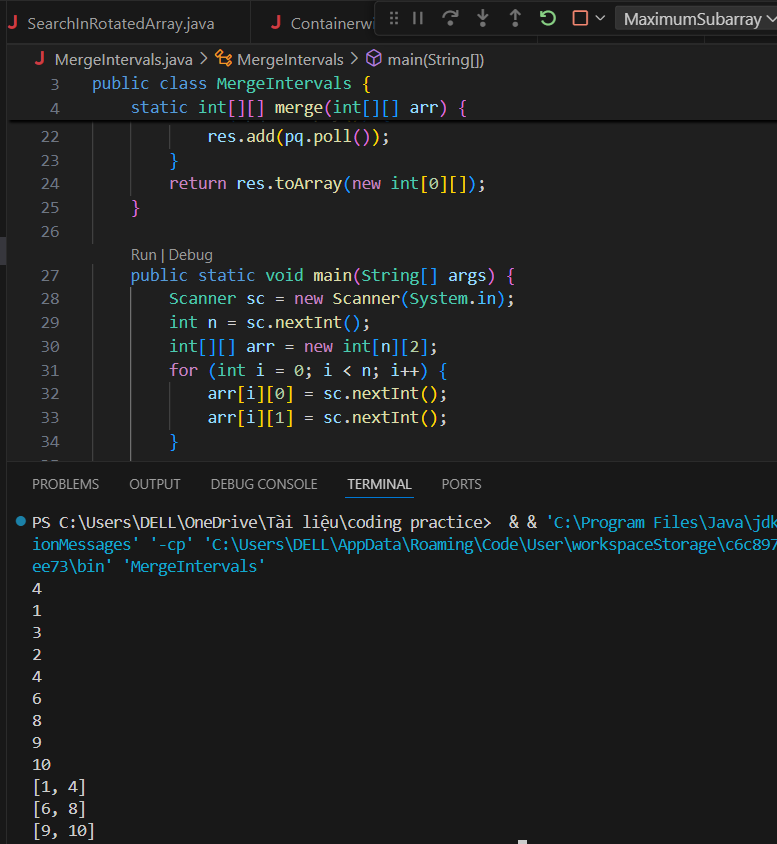
            System.out.println(Arrays.toString(interval));

        }

        sc.close();

    }

}



**Time Complexity : O(nlogn)**

**Space Complexity: O(n)**

**9. BOOLEAN MATRIX**

import java.util.\*;

public class BooleanMatrix {

    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) {

        Scanner sc = new Scanner(System.in);

        int M = sc.nextInt();

        int N = sc.nextInt();

        int[][] mat = new int[M][N];

        for (int i = 0; i < M; i++) {

            for (int j = 0; j < N; j++) {

                mat[i][j] = sc.nextInt();

            }

        }

        modifyMatrix(mat);

        for (int[] row : mat) {

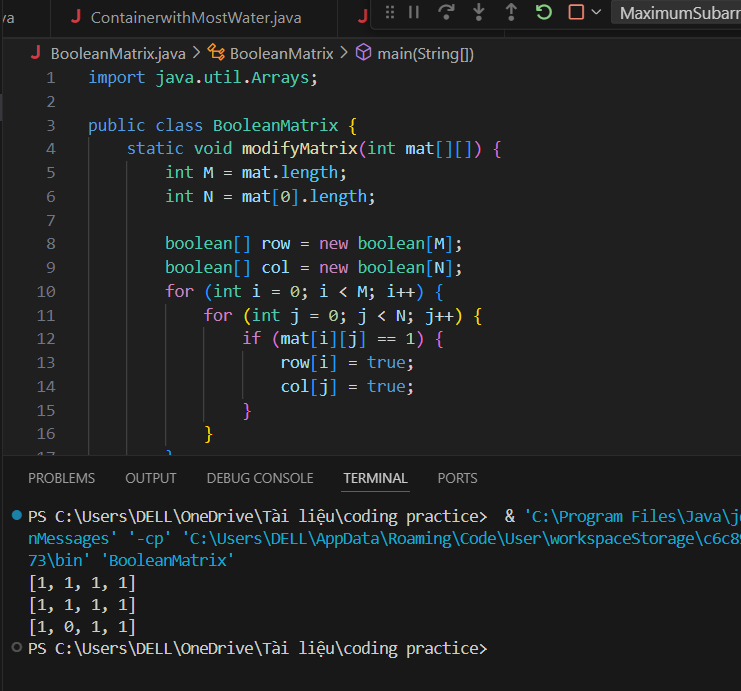
            System.out.println(Arrays.toString(row));

        }

        sc.close();

    }

}



**Time Complexity: O(m\*n)**

**Space Complexity: O(m+n)**

**10. SPIRAL MATRIX**

import java.util.\*;

public class SpiralMatrix {

    static List<Integer> spiralOrder(int[][] matrix) {

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

        if (matrix.length == 0) return result;

        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++) {

                result.add(matrix[top][i]);

            }

            top++;

            for (int i = top; i <= bottom; i++) {

                result.add(matrix[i][right]);

            }

            right--;

            if (top <= bottom) {

                for (int i = right; i >= left; i--) {

                    result.add(matrix[bottom][i]);

                }

                bottom--;

            }

            if (left <= right) {

                for (int i = bottom; i >= top; i--) {

                    result.add(matrix[i][left]);

                }

                left++;

            }

        }

        return result;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int M = sc.nextInt();

        int N = sc.nextInt();

        int[][] matrix = new int[M][N];

        for (int i = 0; i < M; i++) {

            for (int j = 0; j < N; j++) {

                matrix[i][j] = sc.nextInt();

            }

        }

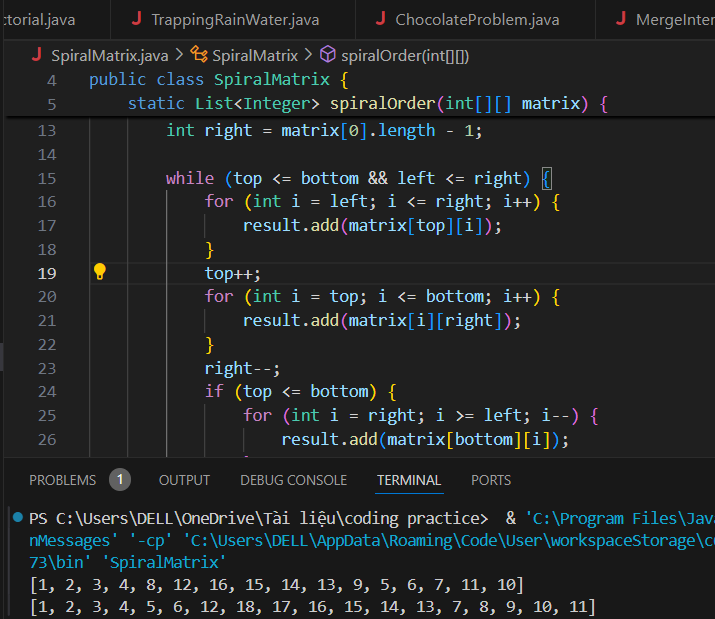
        List<Integer> spiralOrder = spiralOrder(matrix);

        System.out.println(spiralOrder);

        sc.close();

    }

}



**Time Complexity : O(m\*n)**

**Space Complexity : O(m\*n)**

**11. CHECK PARENTHESIS**

import java.util.Scanner;

public class Paranthesis{

    static Boolean check(String str){

        int balance=0;

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

            if (str.charAt(i)=='('){

                balance++;

            }

            else{

                balance--;

            }

            if(balance<0){

                return false;

            }

        }

        return balance==0;

    }

    public static void main(String[] args) {

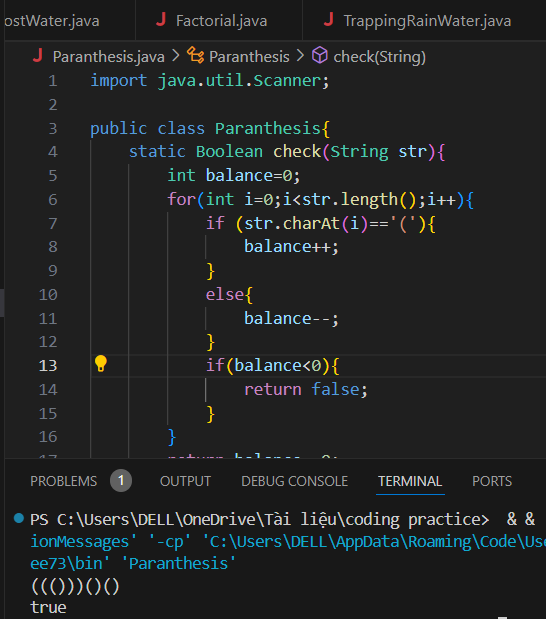
        Scanner sc = new Scanner(System.in);

        String str=sc.nextLine();

        System.out.println(check(str));

    }

}



**Time Complexity: O(n)**

**Space Complexity: O(1)**

**12. ANAGRAM**

import java.util.\*;

public class AnagramCheck {

    static boolean areAnagrams(String s1, String s2) {

        if (s1.length() != s2.length()) {

            return false;

        }

        int[] count = new int[26];

        for (int i = 0; i < s1.length(); i++) {

            count[s1.charAt(i) - 'a']++;

            count[s2.charAt(i) - 'a']--;

        }

        for (int c : count) {

            if (c != 0) {

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String s1 = sc.nextLine();

        String s2 = sc.nextLine();

        if (areAnagrams(s1, s2)) {

            System.out.println("The strings are anagrams.");

        } else {

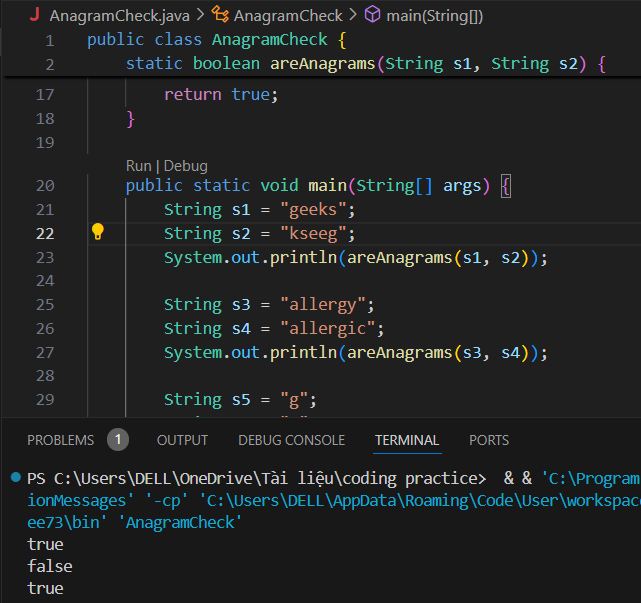
            System.out.println("The strings are not anagrams.");

        }

        sc.close();

    }

}



**Time Complexity : O(n)**

**Space Complexity : O(1)**

**13. LONGEST PALINDROMIC SUBSTRING**

import java.util.\*;

public class LongestPalindromicSubstring {

    public static String longestPalindrome(String s) {

        if (s == null || s.length() < 1) return "";

        int start = 0, end = 0;

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

            int len1 = expandFromCenter(s, i, i);

            int len2 = expandFromCenter(s, i, i + 1);

            int len = Math.max(len1, len2);

            if (len > end - start) {

                start = i - (len - 1) / 2;

                end = i + len / 2;

            }

        }

        return s.substring(start, end + 1);

    }

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

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

            left--;

            right++;

        }

        return right - left - 1;

    }

    public static void main(String[] args) {

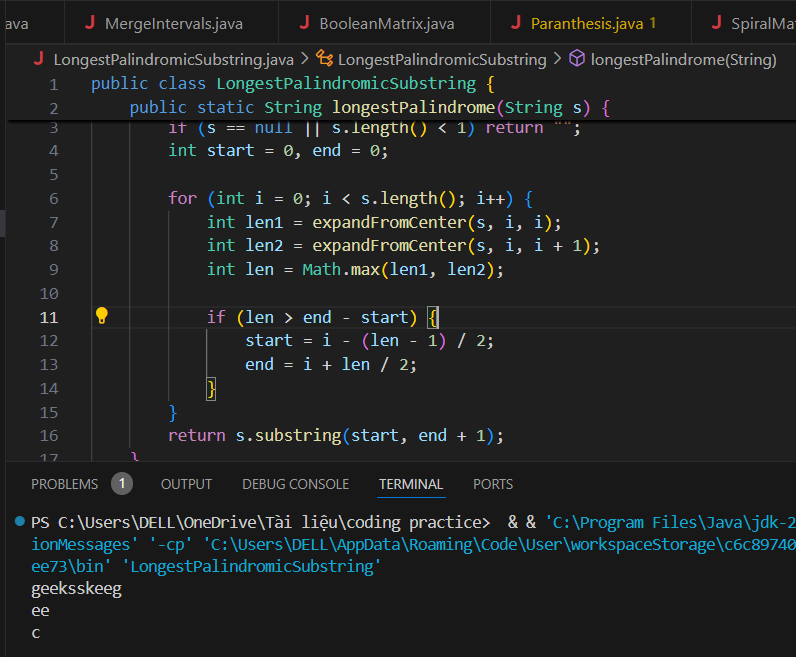
        Scanner sc = new Scanner(System.in);

        String str=sc.nextLine();

        System.out.println(longestPalindrome(str));

    }

}



**Time Complexity : O(n^2)**

**Space Complexity : O(1)**

**14.LONGEST COMMON PREFIX**

import java.util.\*;

public class LongestCommonPrefix {

    public static String longestCommonPrefix(String[] arr) {

        if (arr == null || arr.length == 0) {

            return "-1";

        }

        Arrays.sort(arr);

        String first = arr[0];

        String last = arr[arr.length - 1];

        int i = 0;

        while (i < first.length() && i < last.length() && first.charAt(i) == last.charAt(i)) {

            i++;

        }

        if (i == 0) {

            return "-1";

        }

        return first.substring(0, i);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        sc.nextLine();

        String[] arr = new String[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextLine();

        }

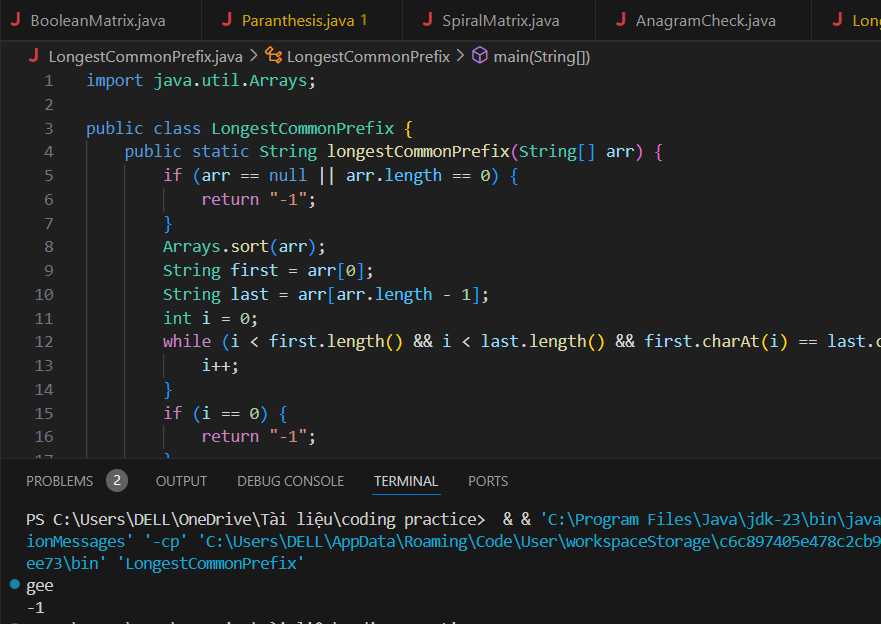
        String result = longestCommonPrefix(arr);

        System.out.println(result);

        sc.close();

    }

}



**Time Complexity : O(nlogn+M)**

**Space Complexity : O(1)**

**15. DELETE MIDDLE ELEMENT FROM STACK**

import java.util.\*;

public class Stackdelete {

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

        if (current == size / 2) {

            stack.pop();

            return;

        }

        int top = stack.pop();

        deleteMiddle(stack, size, current + 1);

        stack.push(top);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

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

        for (int i = 0; i < n; i++) {

            stack.push(sc.nextInt());

        }

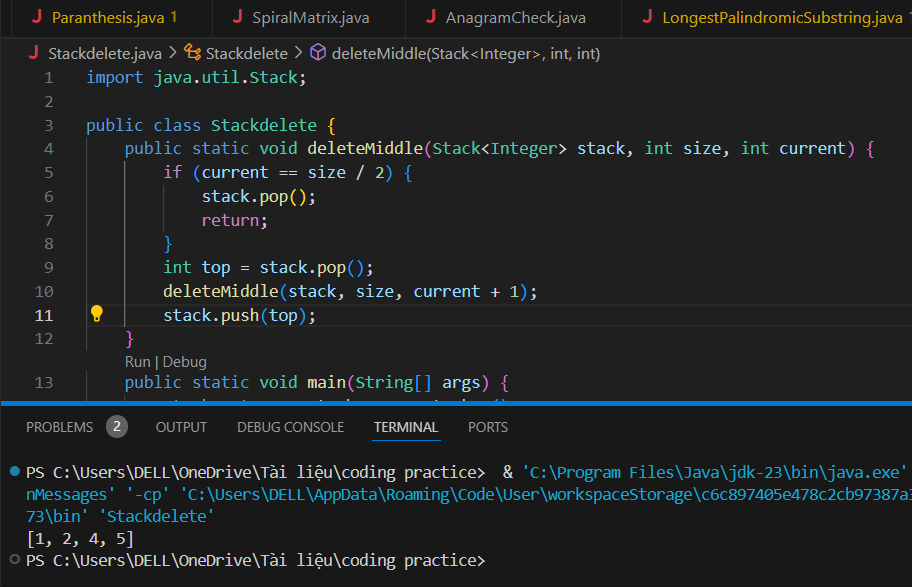
        int size = stack.size();

        deleteMiddle(stack, size, 0);

        System.out.println(stack);

    }

}



**Time Complexity : O(n)**

**Space Complexity : O(n)**

**16. NEXT GREATER ELEMENT**

import java.util.Scanner;

import java.util.Stack;

public class NextGreaterElement {

    public static void printNextGreaterElements(int[] arr) {

        int n = arr.length;

        int[] result = new int[n];

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

        for (int i = 0; i < n; i++) {

            result[i] = -1;

        }

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

            while (!stack.isEmpty() && stack.peek() <= arr[i]) {

                stack.pop();

            }

            if (!stack.isEmpty()) {

                result[i] = stack.peek();

            }

            stack.push(arr[i]);

        }

        for (int i = 0; i < n; i++) {

            System.out.println(arr[i] + " --> " + (result[i] != -1 ? result[i] : "-1"));

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

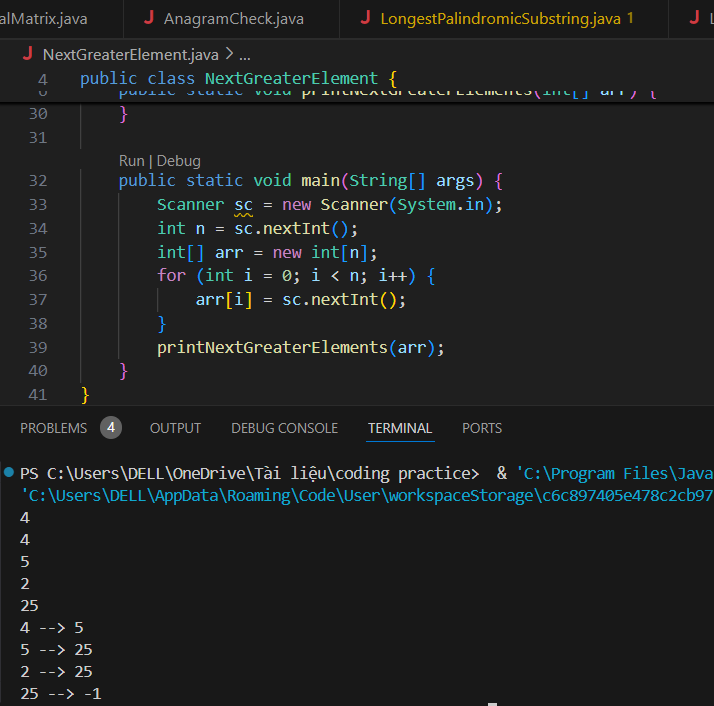
            arr[i] = sc.nextInt();

        }

        printNextGreaterElements(arr);

    }

}



**Time Complexity : O(n)**

**Space Complexity : O(n)**

**17. RIGHT VIEW OF BINARY TREE**

import java.util.\*;

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int val) {

        this.val = val;

        left = right = null;

    }

}

public class RightViewBinaryTree {

    public static ArrayList<Integer> rightView(TreeNode root) {

        ArrayList<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) {

        TreeNode root1 = new TreeNode(1);

        root1.left = new TreeNode(2);

        root1.right = new TreeNode(3);

        root1.right.left = new TreeNode(4);

        root1.right.right = new TreeNode(5);

        ArrayList<Integer> rightView1 = rightView(root1);

        System.out.println(rightView1);

        TreeNode root2 = new TreeNode(1);

        root2.left = new TreeNode(2);

        root2.right = new TreeNode(3);

        root2.left.left = new TreeNode(4);

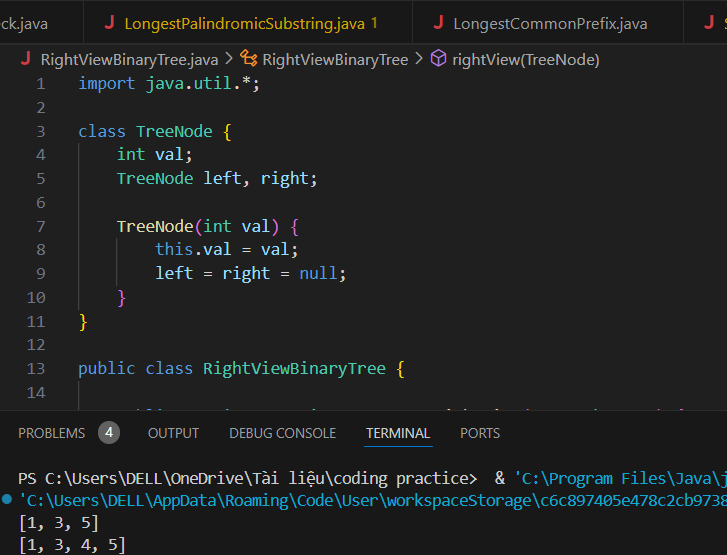
        root2.left.left.right = new TreeNode(5);

        ArrayList<Integer> rightView2 = rightView(root2);

        System.out.println(rightView2);

    }

}



**Time Complexity : O(n)**

**Space Complexity : O(n)**

**18. BINARY TREE HEIGHT**

class TreeNode {

    int val;

    TreeNode left, right;

    public TreeNode(int val) {

        this.val = val;

        left = right = null;

    }

}

public class BinaryTreeHeight {

    public static 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) {

        TreeNode root1 = new TreeNode(12);

        root1.left = new TreeNode(8);

        root1.right = new TreeNode(18);

        root1.left.left = new TreeNode(5);

        root1.left.right = new TreeNode(11);

        System.out.println(maxDepth(root1));

        TreeNode root2 = new TreeNode(1);

        root2.left = new TreeNode(2);

        root2.right = new TreeNode(3);

        root2.left.left = new TreeNode(4);

        root2.right.left = new TreeNode(5);

        root2.right.left.left = new TreeNode(6);

        root2.right.left.right = new TreeNode(7);

        System.out.println(maxDepth(root2));

    }

}



**Time Complexity : O(n)**

**Space Complexity: O(h)**