**NAME: SRINTHI JAIKUMAR**

**DEPARTMENT:AIDS**

**DATE:9/11/24**

**Coding practice 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

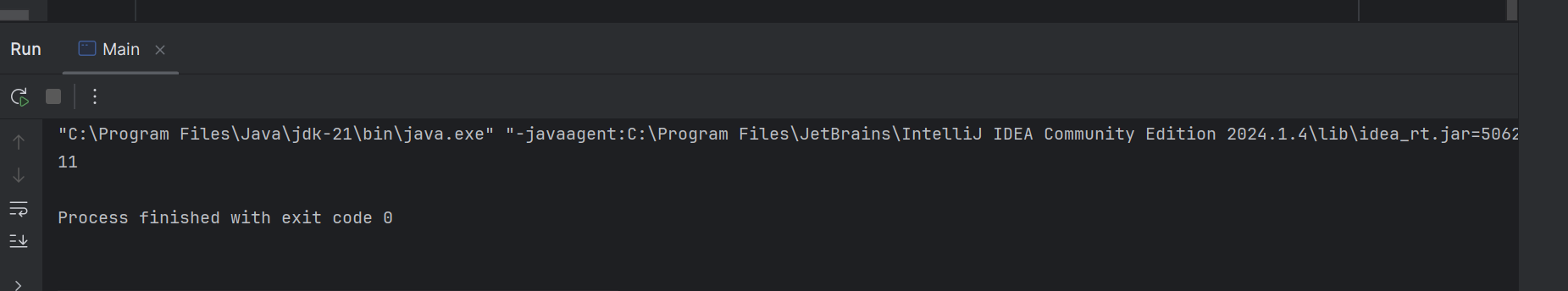
Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.

* Input: arr[] = {-2, -4}

Output: –2

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

import java.util.\*;  
  
class Main{  
 static int maxSubArrSum(int[] arr){  
 int maxSum=arr[0];  
 int currSum=0;  
  
 for(int i=0;i<arr.length;i++){  
 if (currSum<0){  
 currSum=0;  
 }  
 currSum+=arr[i];  
 maxSum=Math.*max*(currSum,maxSum);  
 }  
 return maxSum;  
 }  
  
 public static void main(String[] args) {  
 int[] arr = {2, 3, -8, 7, -1, 2, 3};  
 System.*out*.println(*maxSubArrSum*(arr));  
 }  
}



 **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)

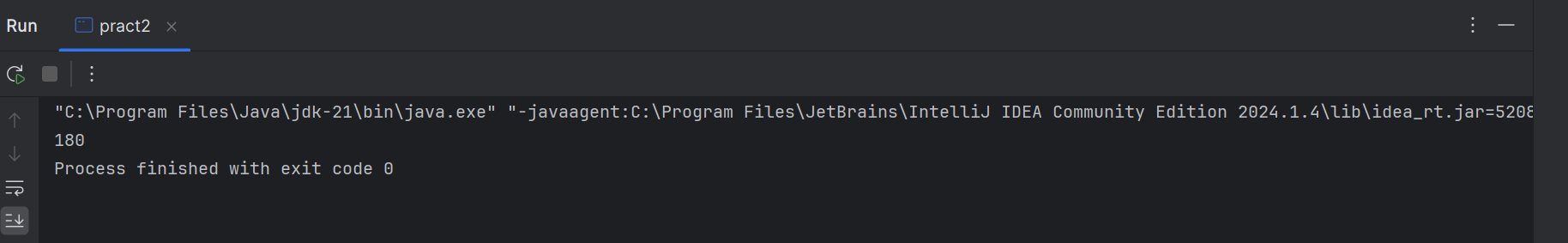
= 180

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

Output: 60

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

import java.util.\*;  
  
public class pract2 {  
 static int maxProdSubArr(int[] arr){  
 int n=arr.length;  
 int pre=1;  
 int suff=1;  
 int res=Integer.*MIN\_VALUE*;  
 for(int i=0;i<n;i++){  
 if(pre==0){  
 pre=1;  
 }  
 if(suff==0){  
 suff=1;  
 }  
 pre\*=arr[i];  
 suff\*=arr[n-i-1];  
 res=Math.*max*(res,Math.*max*(pre,suff));  
 }  
 return res;  
  
 }  
  
 public static void main(String[] args) {  
 int nums[] = {-2, 6, -3, -10, 0, 2};  
 int answer =*maxProdSubArr*(nums);  
 System.*out*.print(answer);  
 }  
  
}



**Time Complexity:** O(N)

Space Complexity: O(1)

**3. Search in a sorted and rotated Array**

3)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

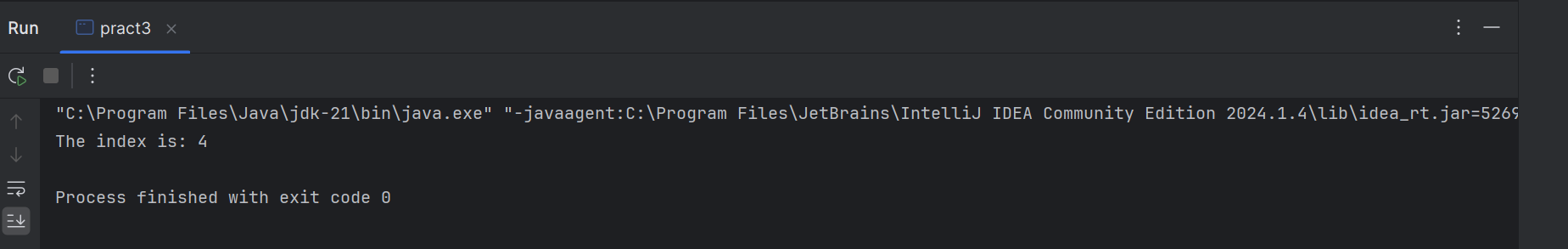
Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3

Output : -1

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

Output : 1

import java.util.\*;  
  
public class pract3 {  
 static int searchSortedArr(ArrayList<Integer> arr, int target) {  
 int l = 0;  
 int h = arr.size() - 1;  
  
 while (l <= h) {  
 int mid = (l + h) / 2;  
  
 if (arr.get(mid) == target) {  
 return mid;  
 }  
  
 // Check if the left half is sorted  
 if (arr.get(l) <= arr.get(mid)) {  
 if (arr.get(l) <= target && target <= arr.get(mid)) {  
 h = mid - 1;  
 } else {  
 l = mid + 1;  
 }  
 }  
 // Right half is sorted  
 else {  
 if (arr.get(mid) <= target && target <= arr.get(h)) {  
 l = mid + 1;  
 } else {  
 h = mid - 1;  
 }  
 }  
 }  
 return -1;  
 }  
  
 public static void main(String[] args) {  
 ArrayList<Integer> arr = new ArrayList<>(Arrays.*asList*(4, 5, 6, 7, 0, 1, 2));  
 int k = 0;  
 int ans = *searchSortedArr*(arr, k);  
 if (ans == -1)  
 System.*out*.println("Target is not present.");  
 else  
 System.*out*.println("The index is: " + ans);  
 }  
}

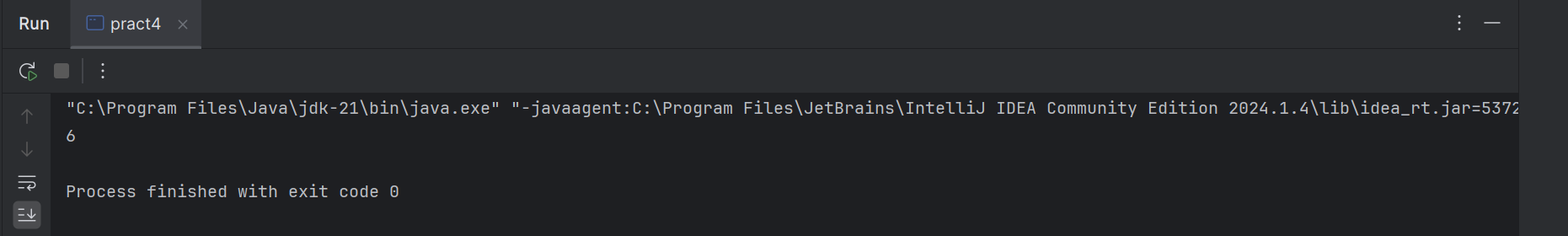


**Time Complexity:**O(logN)

**Space Complexity:**O(1)

**4.Container with Most Water**

import java.util.\*;  
  
public class pract4 {  
 public int maxArea(int[] height) {  
 int left = 0;  
 int right = height.length - 1;  
 int maxArea = 0;  
  
 while (left < right) {  
 int currentArea = Math.*min*(height[left], height[right]) \* (right - left);  
 maxArea = Math.*max*(maxArea, currentArea);  
  
 if (height[left] < height[right]) {  
 left++;  
 } else {  
 right--;  
 }  
 }  
  
 return maxArea;  
 }  
  
 public static void main(String[] args) {  
 pract4 solution = new pract4();  
 int[] arr = {1, 5, 4, 3};  
 int result = solution.maxArea(arr);  
 System.*out*.println(result);  
 }  
}

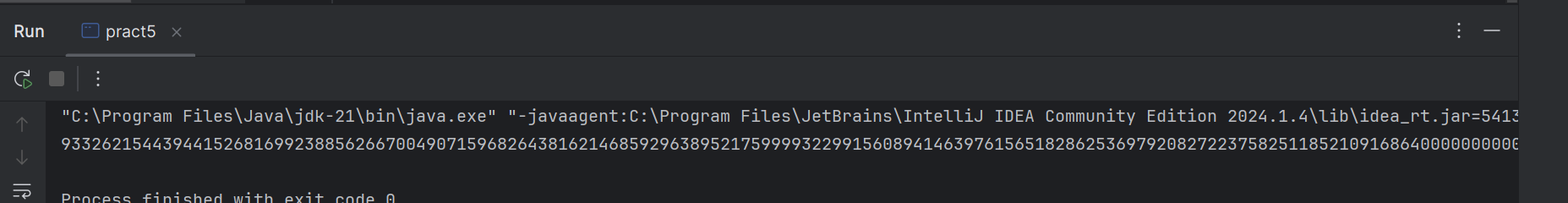


**Time Complexity:**O(N)

**Space Complexity:**O(1)

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

import java.math.BigInteger;  
import java.util.Scanner;  
  
public class pract5 {  
  
 static BigInteger factorial(int N)  
 {  
 BigInteger f  
 = new BigInteger("1");  
  
 for (int i = 2; i <= N; i++)  
 f = f.multiply(BigInteger.*valueOf*(i));  
  
 return f;  
 }  
  
 public static void main(String args[]) throws Exception  
 {  
 int N = 100;  
 System.*out*.println(*factorial*(N));  
 }  
}

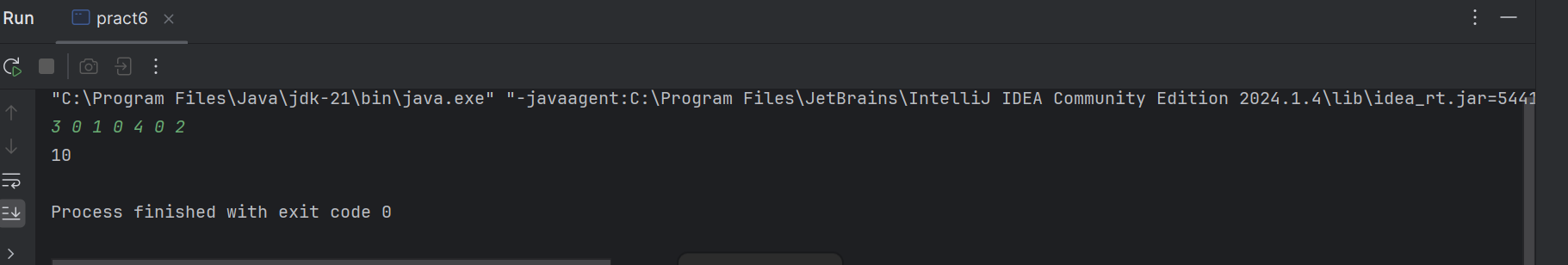


**Time Complexity:**O(N)

**Space Complexity:**O(1)

6. Trapping Rainwater Problem

package com;  
import java.util.Scanner;  
class pract6 {  
 static int trapWater(int[] arr) {  
 int n = arr.length;  
 if (n <= 2) return 0;  
 int[] l = new int[n];  
 int[] r = new int[n];  
 int ans = 0;  
 l[0] = arr[0];  
 for (int i = 1; i < n; i++) {  
 l[i] = Math.*max*(l[i - 1], arr[i]);  
 }  
  
 r[n - 1] = arr[n - 1];  
 for (int i = n - 2; i >= 0; i--) {  
 r[i] = Math.*max*(r[i + 1], arr[i]);  
 }  
 for (int i = 1; i < n - 1; i++) {  
 ans += Math.*min*(l[i], r[i]) - arr[i];  
 }  
 return ans;  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 String input = scanner.nextLine();  
 String[] parts = input.split(" ");  
 int[] arr = new int[parts.length];  
 for (int i = 0; i < parts.length; i++) {  
 arr[i] = Integer.*parseInt*(parts[i]);  
 }  
 System.*out*.println(*trapWater*(arr));  
 scanner.close();  
 }  
}

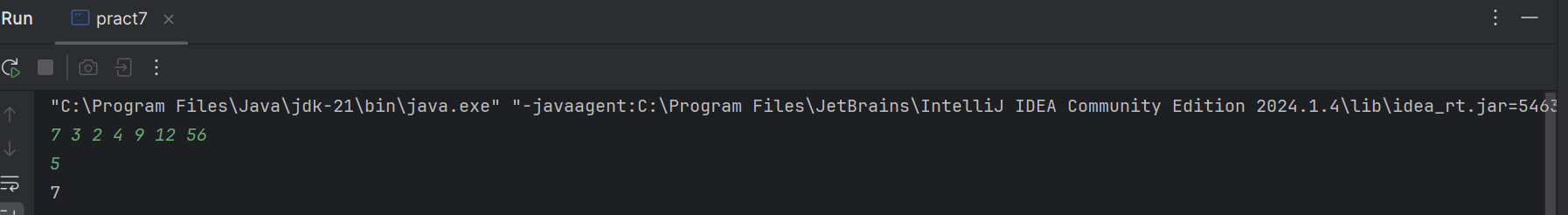


**Time Complexity:**O(N)

**Space Complexity:**O(1)

**7. Chocolate Distribution Problem**

package com;  
import java.util.Scanner;  
import java.util.Arrays;  
  
public class pract7 {  
 public static int chocolate(int[] arr, int m) {  
 int n = arr.length;  
 if (n < m) {  
 return -1;  
 }  
 Arrays.*sort*(arr);  
 int md = Integer.*MAX\_VALUE*;  
  
 for (int i = 0; i + m - 1 < n; i++) {  
  
 int diff = arr[i + m - 1] - arr[i];  
 md = Math.*min*(md, diff);  
 }  
 return md;  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 String input = scanner.nextLine();  
 String[] parts = input.split(" ");  
 int[] arr = new int[parts.length];  
 for (int i = 0; i < parts.length; i++) {  
 arr[i] = Integer.*parseInt*(parts[i]);  
 }  
 int m = scanner.nextInt();  
 System.*out*.println(*chocolate*(arr, m));  
 scanner.close();  
 }  
}

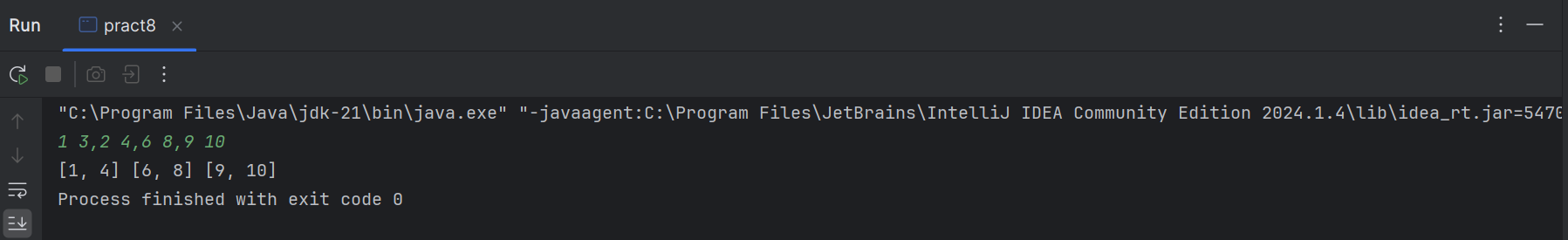


**Time Complexity:**O(N)

**Space Complexity:**O(1)

**8. Merge Overlapping Intervals**

package com;  
import java.util.\*;  
  
public class pract8 {  
 public static int[][] mergeint(int[][] arr) {  
 if (arr.length <= 1) return arr;  
 Arrays.*sort*(arr, (a, b) -> Integer.*compare*(a[0], b[0]));  
 List<int[]> merged = new ArrayList<>();  
 merged.add(arr[0]);  
 for (int i = 1; i < arr.length; i++) {  
 int[] lastInterval = merged.get(merged.size() - 1);  
 if (arr[i][0] <= lastInterval[1]) {  
 lastInterval[1] = Math.*max*(lastInterval[1], arr[i][1]);  
 } else {  
 merged.add(arr[i]);  
 }  
 }  
 return merged.toArray(new int[merged.size()][]);  
  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 String input = scanner.nextLine();  
 String[] parts = input.split(",\\s\*");  
 int[][] arr = new int[parts.length][2];  
 for (int i = 0; i < parts.length; i++) {  
 String[] interval = parts[i].split("\\s+");  
 arr[i][0] = Integer.*parseInt*(interval[0]);  
 arr[i][1] = Integer.*parseInt*(interval[1]);  
 }  
 int[][] result = *mergeint*(arr);  
 for (int[] interval : result) {  
 System.*out*.print(Arrays.*toString*(interval) + " ");  
 }  
 scanner.close();  
 }  
}

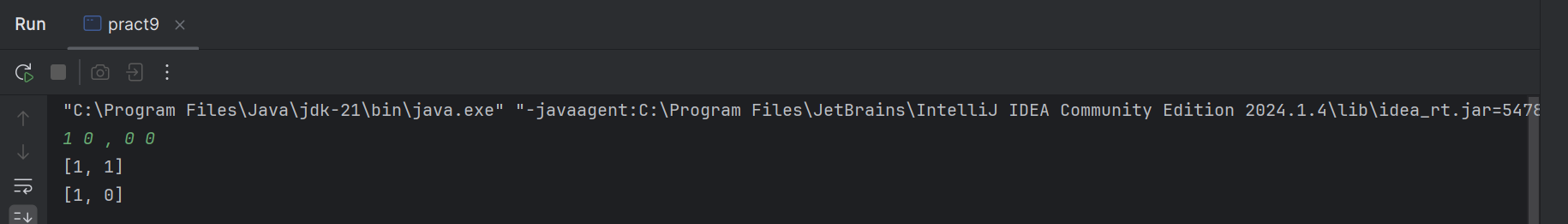


**Time Complexity:**O(N)

**Space Complexity:**O(1)

**9. A Boolean Matrix Question**

package com;  
import java.util.\*;  
  
public class pract9 {  
 public static void modifyMatrix(int[][] mat) {  
 int m = mat.length;  
 int n = mat[0].length;  
 boolean[] rowFlag = new boolean[m];  
 boolean[] colFlag = new boolean[n];  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < n; j++) {  
 if (mat[i][j] == 1) {  
  
 rowFlag[i] = true;  
 colFlag[j] = true;  
 }  
 }  
 }  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < n; j++) {  
 if (rowFlag[i] || colFlag[j]) {  
 mat[i][j] = 1;  
 }  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 String input = scanner.nextLine();  
 String[] parts = input.split(",\\s\*");  
 int numIntervals = parts.length;  
 int[][] mat = new int[numIntervals][2];  
 for (int i = 0; i < numIntervals; i++) {  
 String[] interval = parts[i].split("\\s+");  
 mat[i][0] = Integer.*parseInt*(interval[0].trim());  
 mat[i][1] = Integer.*parseInt*(interval[1].trim());  
 }  
 *modifyMatrix*(mat);  
 for (int[] row : mat) {  
 System.*out*.println(Arrays.*toString*(row));  
 }  
 scanner.close();  
 }  
}

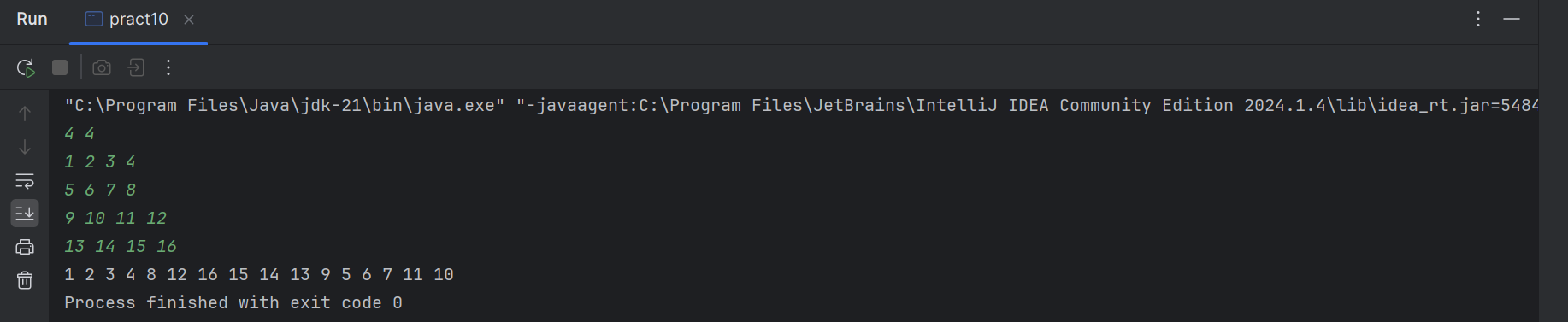
****

**Time Complexity:**O(N)

**Space Complexity:**O(1)

**10. Print a given matrix in spiral form**

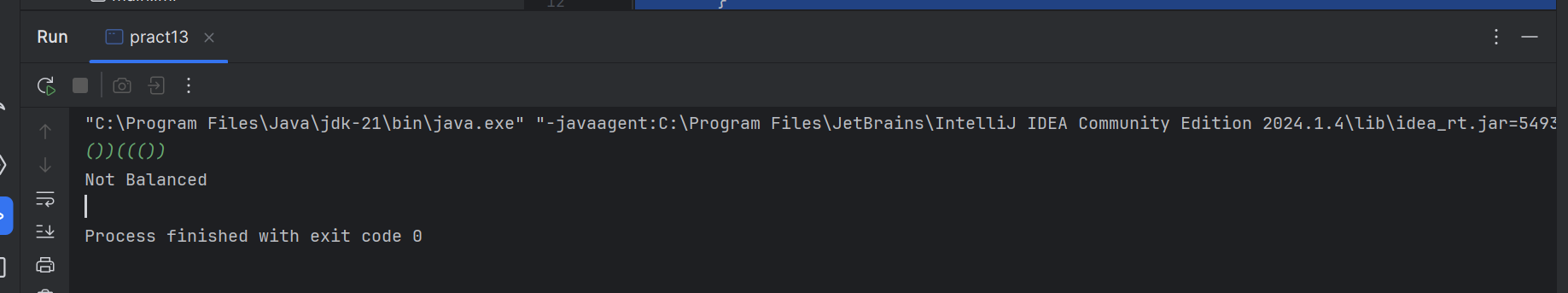
package com;  
import java.util.Scanner;  
  
public class pract10 {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int m = sc.nextInt();  
 int n = sc.nextInt();  
 int[][] matrix = new int[m][n];  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < n; j++) {  
 matrix[i][j] = sc.nextInt();  
 }  
 }  
 *printSpiral*(matrix, m, n);  
 }  
  
 public static void printSpiral(int[][] matrix, int m, int n) {  
  
 int t = 0, b = m - 1, l = 0, r = n - 1;  
  
 while (t <= b && l <= r) {  
 for (int i = l; i <= r; i++) {  
 System.*out*.print(matrix[t][i] + " ");  
 }  
 t++;  
  
 for (int i = t; i <= b; i++) {  
 System.*out*.print(matrix[i][r] + " ");  
 }  
 r--;  
  
 if (t <= b) {  
 for (int i = r; i >= l; i--) {  
 System.*out*.print(matrix[b][i] + " ");  
 }  
 b--;  
 }  
  
 if (l <= r) {  
 for (int i = b; i >= t; i--) {  
 System.*out*.print(matrix[i][l] + " ");  
 }  
 l++;  
 }  
 }  
 }  
}

****

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

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

package com;  
import java.util.Scanner;  
import java.util.Stack;  
public class pract13 {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 String str = sc.next();  
 if (*isBalanced*(str)) {  
 System.*out*.println("Balanced");  
 } else {  
 System.*out*.println("Not Balanced");  
 }  
 }  
  
 public static boolean isBalanced(String str) {  
 Stack<Character> stack = new Stack<>();  
 for (int i = 0; i < str.length(); i++) {  
 char ch = str.charAt(i);  
 if (ch == '(') {  
 stack.push(ch);  
 } else if (ch == ')') {  
 if (stack.isEmpty()) {  
  
 return false;  
 }  
 stack.pop();  
 }  
 }  
 return stack.isEmpty();  
 }  
}

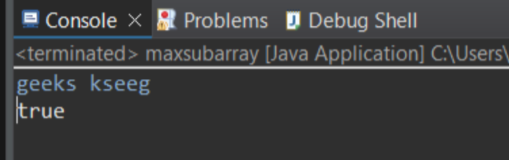
****

Time complexity: O(n)

14. Check if two Strings are Anagrams of each other

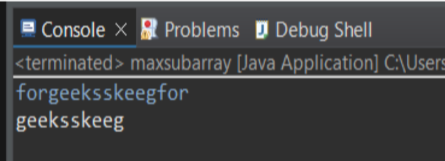
package com;  
import java.util.Scanner;  
  
public class pract14 {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 String s1 = sc.next();  
 String s2 = sc.next();  
  
 if (*areAnagrams*(s1, s2)) {  
 System.*out*.println("true");  
 } else {  
 System.*out*.println("false");  
 }  
 }  
  
 public 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 i = 0; i < 26; i++) {  
 if (count[i] != 0) {  
 return false;  
 }  
 }  
 return true;  
 }  
}

Time complexity: O(n)



**15. Longest Palindromic Substring**

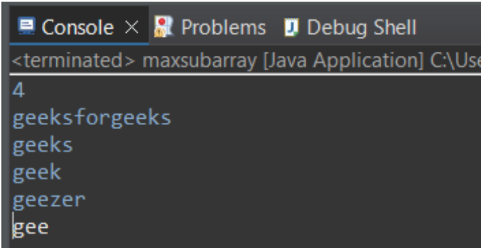
package com;  
import java.util.Scanner;  
public class pract15 {  
 static boolean checkPal(String s, int low, int high) {  
 while (low < high) {  
 if (s.charAt(low) != s.charAt(high))  
 return false;  
 low++;  
 high--;  
 }  
 return true;  
 }  
 static String longestPalSubstr(String s) {  
 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) {  
 Scanner sc = new Scanner(System.*in*);  
 String s = sc.nextLine();  
 System.*out*.println(*longestPalSubstr*(s));  
 sc.close();  
 }  
}

****

**Time Complexity: O(N3)**

**16. Longest Common Prefix using Sorting**

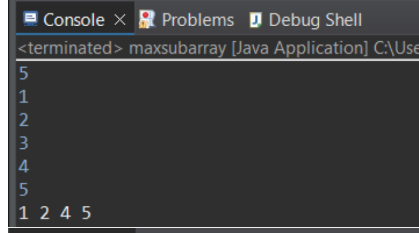
package com;  
import java.util.Scanner;  
import java.util.Arrays;  
  
public class java16 {  
 static String longestCommonPrefix(String[] arr) {  
 Arrays.*sort*(arr);  
 String first = arr[0];  
 String last = arr[arr.length - 1];  
 int n = Math.*min*(first.length(), last.length());  
 int i = 0;  
 while (i < n && 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();  
 }  
 System.*out*.println(*longestCommonPrefix*(arr));  
 sc.close();  
 }  
}

****

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

**17. Delete middle element of a stack**

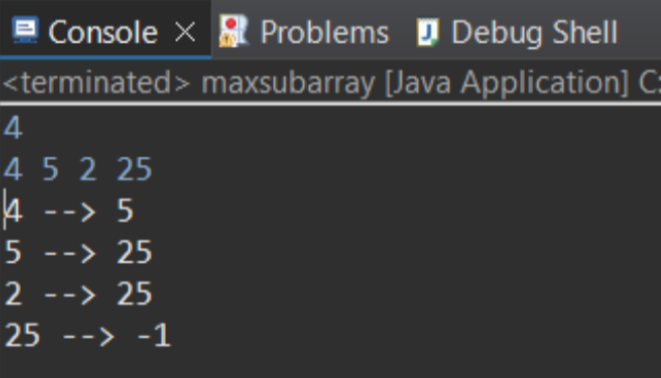
package com;  
import java.util.Stack;  
import java.util.Vector;  
import java.util.Scanner;  
import java.util.List;  
import java.util.ArrayList;  
import java.util.Collections;  
public class pract17 {  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt();  
 sc.nextLine();  
 Stack<Character> st = new Stack<Character>();  
 for (int i = 0; i < n; i++) {  
 st.push(sc.nextLine().charAt(0));  
 }  
 if (st.isEmpty()) {  
 System.*out*.println("Stack is empty.");  
 return;  
 }  
 Vector<Character> v = new Vector<Character>();  
 while (!st.empty()) {  
 v.add(st.pop());  
 }  
  
 int midIndex = v.size() / 2;  
 v.remove(midIndex);  
 List<Character> sortedList = new ArrayList<>(v);  
 Collections.*sort*(sortedList);  
 for (Character c : sortedList) {  
 System.*out*.print(c + " ");  
 }  
  
 sc.close();  
 }  
}

****

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

**18. Next Greater Element (NGE) for every element in given Array**

package com;  
import java.util.Stack;  
import java.util.Scanner;  
  
public class pract18 {  
 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();  
 }  
  
 Stack<Integer> stack = new Stack<>();  
 int[] nge = new int[n];  
 for (int i = 0; i < n; i++) {  
 nge[i] = -1;  
 }  
 for (int i = n - 1; i >= 0; i--) {  
 while (!stack.isEmpty() && stack.peek() <= arr[i]) {  
 stack.pop();  
 }  
 if (!stack.isEmpty()) {  
 nge[i] = stack.peek();  
 }  
 stack.push(arr[i]);  
 }  
 for (int i = 0; i < n; i++) {  
 System.*out*.println(arr[i] + " --> " + nge[i]);  
 }  
 sc.close();  
 }  
}

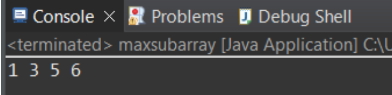
****

**Time complexity: O(n)**

**19. Print Right View of a Binary Tree Given a Binary Tree,**

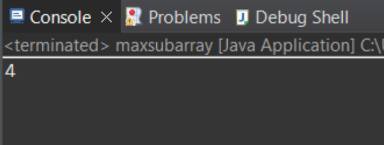
package com;  
import java.util.LinkedList;  
import java.util.Queue;  
class Node {  
 int data;  
 Node left, right;  
 Node(int data) {  
 this.data = data;  
 left = right = null;  
 }  
}  
public class pract19 {  
 Node root;  
 public void printRightView() {  
 if (root == null) return;  
 Queue<Node> queue = new LinkedList<>();  
 queue.offer(root);  
 while (!queue.isEmpty()) {  
 int size = queue.size();  
 for (int i = 0; i < size; i++) {  
 Node node = queue.poll();  
 if (i == size - 1) {  
 System.*out*.print(node.data + " ");  
 }  
 if (node.left != null) queue.offer(node.left);  
 if (node.right != null) queue.offer(node.right);  
 }  
 }  
 }  
 public static void main(String[] args) {  
 maxsubarray tree = new maxsubarray();  
 tree.root = new Node(1);  
 tree.root.left = new Node(2);  
 tree.root.right = new Node(3);  
 tree.root.left.right = new Node(4);  
 tree.root.left.right.right = new Node(6);  
 tree.root.right.right = new Node(5);  
 tree.printRightView();

**Time complexity: O(n)**

****

**20. Maximum Depth or Height of Binary Tree**

package com;  
class Node {  
 int data;  
 Node left, right;  
 public Node(int item) {  
 data = item;  
 left = right = null;  
 }  
}  
class pract20 {  
 Node root;  
 int maxDepth(Node node) {  
 if (node == null)  
 return 0;  
 else {  
 int leftDepth = maxDepth(node.left);  
 int rightDepth = maxDepth(node.right);  
 return Math.*max*(leftDepth, rightDepth) + 1;  
 }  
 }  
 public static void main(String[] args) {  
 maxsubarray tree = new maxsubarray();  
 tree.root = new Node(1);  
 tree.root.left = new Node(2);  
 tree.root.right = new Node(3);  
 tree.root.left.left = new Node(4);  
 tree.root.left.right = new Node(5);  
 tree.root.left.left.left = new Node(6);  
 System.*out*.println(tree.maxDepth(tree.root));  
  
 }  
}

****

**Time complexity: O(n)**