Date:09-11-2024

Coding Practice Problem

NAME : SURIYA R

DEPARTMENT : INFORMATION TECHNOLOGY

DSA PRACTICE 1

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.

Solution:

import java.util.\*;

public class Main{

public static void main(String[] aer){

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 maxsum=arr[0];

int res=arr[0];

for(int i=1;i<n;i++){

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

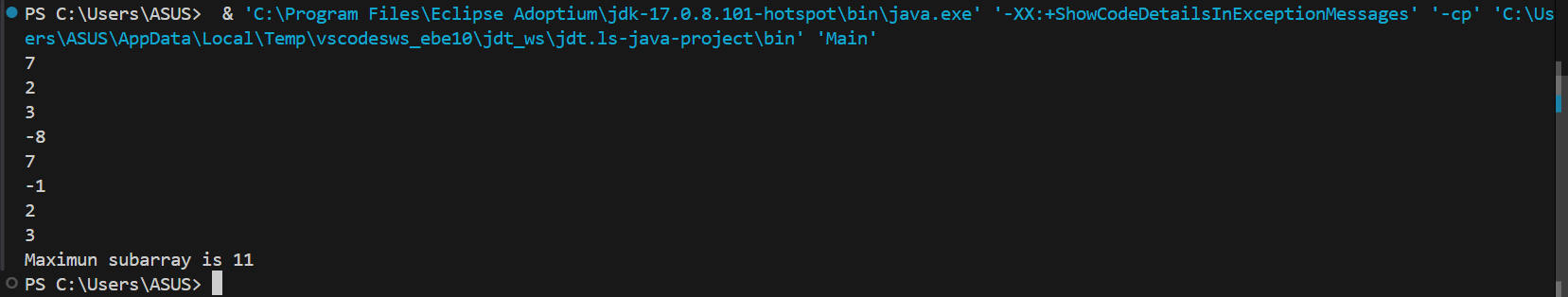
res=Math.max(res,maxsum);

}

System.out.println("Maximun subarray is "+res);

}

}  
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}.

Solution:

import java.util.\*;

class HelloWorld {

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 maxproduct = arr[0];

int minproduct = arr[0];

int ans = arr[0];

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

if (arr[i] < 0) {

int temp = maxproduct;

maxproduct = minproduct;

minproduct = temp;

}

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

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

ans = Math.max(ans, maxproduct);

}

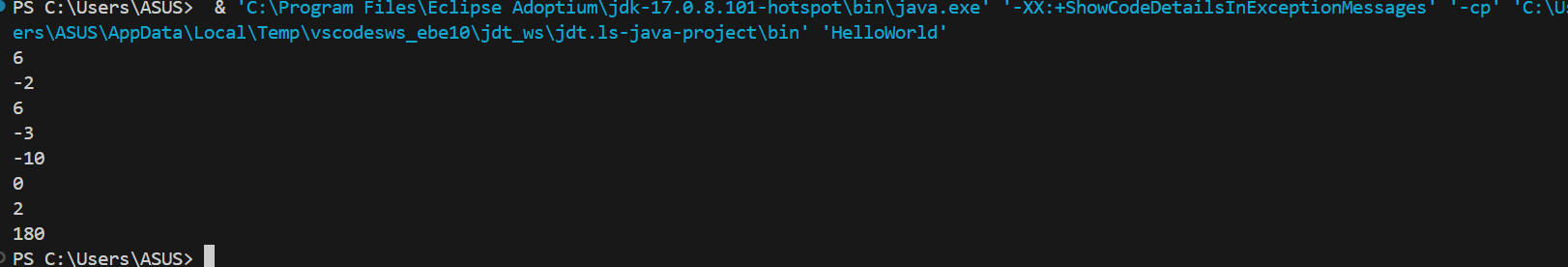
System.out.println(ans);

sc.close();

}

}

Output:



Time Complexity: O(n)

Space complexity: O(1)

1. 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 Main{

public static void main(String[] aer){

Scanner sc=new Scanner(System.in);

System.out.println("Size of the array is");

int n=sc.nextInt();

int[] nums=new int[n];

System.out.println("Inputs are");

for(int i=0;i<n;i++){

nums[i]=sc.nextInt();

}

System.out.println("Key is");

int key=sc.nextInt();

int left=0;

int right=n-1;

boolean found=false;

while(left<=right){

int mid=(left+right)/2;

if(nums[mid]==key){

found=true;

System.out.println(mid);

break;

}

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

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

right=mid-1;

}

else left=mid+1;

}

else{

if(nums[mid+1]<=key && nums[right]>=key){

left=mid+1;

}

else right=mid-1;

}

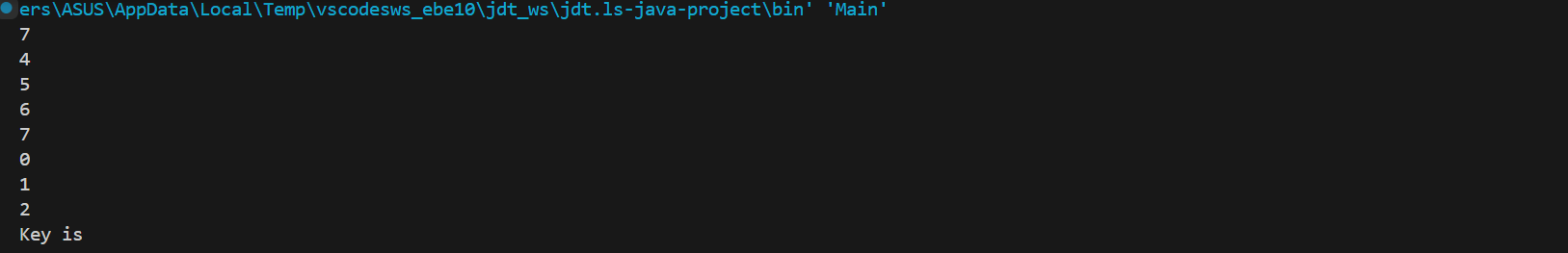
}

if(!found) System.out.println(-1);

}

}

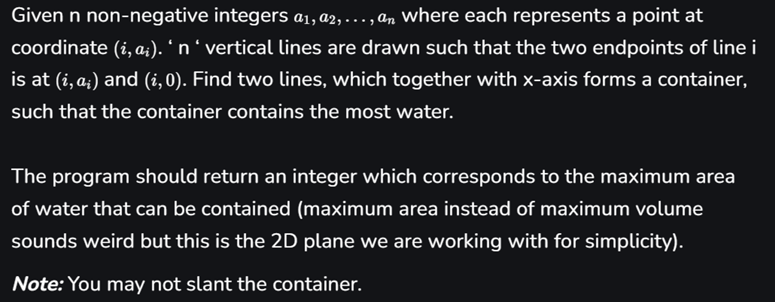
Output:



Time Complexity: O(log n)

Space Complexity:O(1)

1. 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 Main{

    public static int maxArea(int[] height) {

        int l = 0;

        int r = height.length - 1;

        int ma = 0;

        while (l < r) {

            int ca = Math.min(height[l], height[r]) \* (r - l);

            ma = Math.max(ca, ma);

            if (height[l] < height[r]) {

                l++;

            } else {

                r--;

            }

        }

        return ma;

    }

    public static void main(String[] args) {

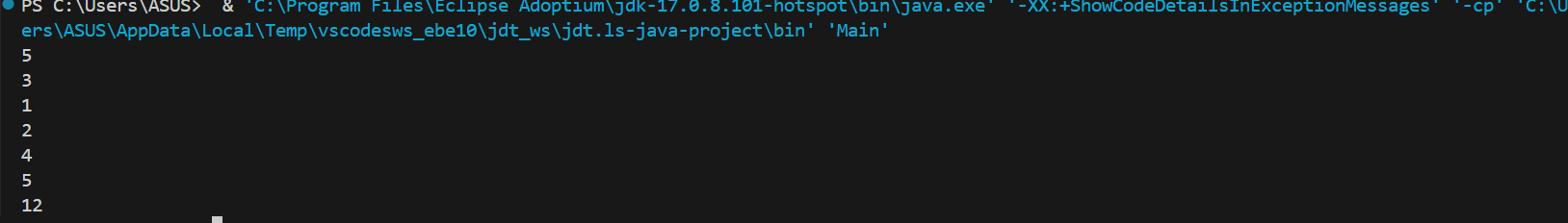
        int[] arr = {3,1,2,4,5};

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

    }

}

Output:



**Time Complexity: O(n)**

**Space Complexity:O(1)**

1. Find the Factorial of a large number

Input: 100

Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

Solution:

import java.math.BigInteger;

public class Main {

    public static void main(String[] args) {

        BigInteger num =  BigInteger.ONE;

        int n=100;

        for(int i = 1 ; i<=n ; i++) {

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

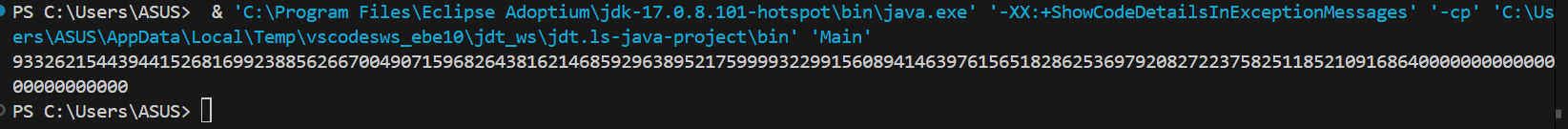
        }

        System.out.println(num);

    }

}

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

Solution:

class Main {

    public static int trap(int[] height) {

        int l=0;

        int r=height.length-1;

        int lm=height[l];

        int rm=height[r];

        int w=0;

        while(l<r){

            if(lm<rm){

                l++;

                lm=Math.max(height[l],lm);

                w+=(lm-height[l]);

            }else{

                r--;

                rm=Math.max(height[r],rm);

                w+=(rm-height[r]);

            }

        }

        return w;

        }

        public static void main(String[] args) {

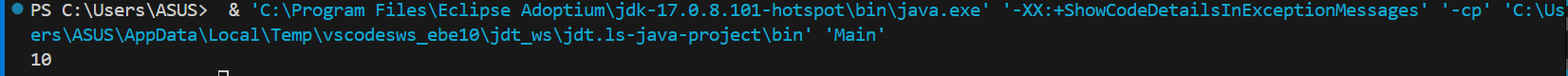
            int[] tra={3, 0, 1, 0, 4, 0, 2};

            System.out.println(trap(tra));

        }

}

Output:



**Time Complexity: O(n)**

**Space Complexity:O(1)**

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;

public class ChocolateDistribution {

    public static int findMinDiff(int[] arr, int n, int m) {

        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) {

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

        int m = 3;

        int n = arr.length;

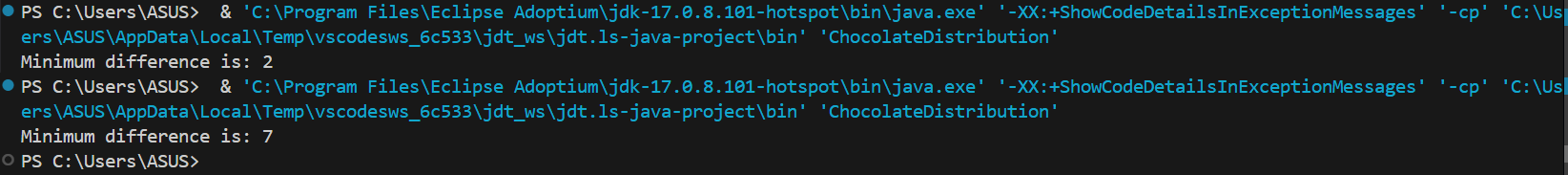
        int result = findMinDiff(arr, n, m);

        System.out.println("Minimum difference is: " + result);

    }

}

Output:



**Time Complexity: O(nLogn)**

**Space Complexity:O(1)**

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.\*;

public class MergeIntervals {

    public static int[][] merge(int[][] intervals) {

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

        List<int[]> merged = new ArrayList<>();

        int[] currentInterval = intervals[0];

        merged.add(currentInterval);

        for (int[] interval : intervals) {

            int currentEnd = currentInterval[1];

            int nextStart = interval[0];

            int nextEnd = interval[1];

            if (currentEnd >= nextStart) {

                currentInterval[1] = Math.max(currentEnd, nextEnd);

            } else {

                currentInterval = interval;

                merged.add(currentInterval);

            }

        }

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

    }

    public static void main(String[] args) {

        int[][] intervals1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};

        int[][] mergedIntervals1 = merge(intervals1);

        System.out.println("Merged Intervals 1: ");

        for (int[] interval : mergedIntervals1) {

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

        }

        int[][] intervals2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};

        int[][] mergedIntervals2 = merge(intervals2);

        System.out.println("Merged Intervals 2: ");

        for (int[] interval : mergedIntervals2) {

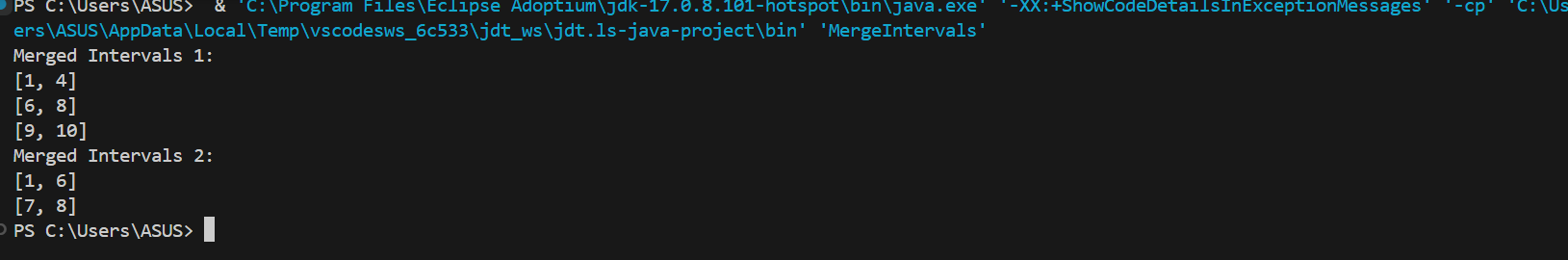
            System.out.println(Arrays.toString(interval));

        }

    }

}

Output:



**Time Complexity: O(nLogn)**

**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}}

Solution:

import java.util.\*;

public class Main {

  public static void modifyMatrix(int[][] mat) {

        int M = mat.length;

        int N = mat[0].length;

        boolean[] rows = new boolean[M];

        boolean[] cols = new boolean[N];

        for (int i = 0; i < M; i++) {

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

                if (mat[i][j] == 1) {

                    rows[i] = true;

                    cols[j] = true;

                }

            }

        }

        for (int i = 0; i < M; i++) {

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

                if (rows[i] || cols[j]) {

                    mat[i][j] = 1;

                }

            }

        }

        for (int[] row : mat) {

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

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of rows: ");

        int M = sc.nextInt();

        System.out.print("Enter number of columns: ");

        int N = sc.nextInt();

        int[][] mat = new int[M][N];

        System.out.println("Enter elements:");

        for (int i = 0; i < M; i++) {

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

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

            }

        }

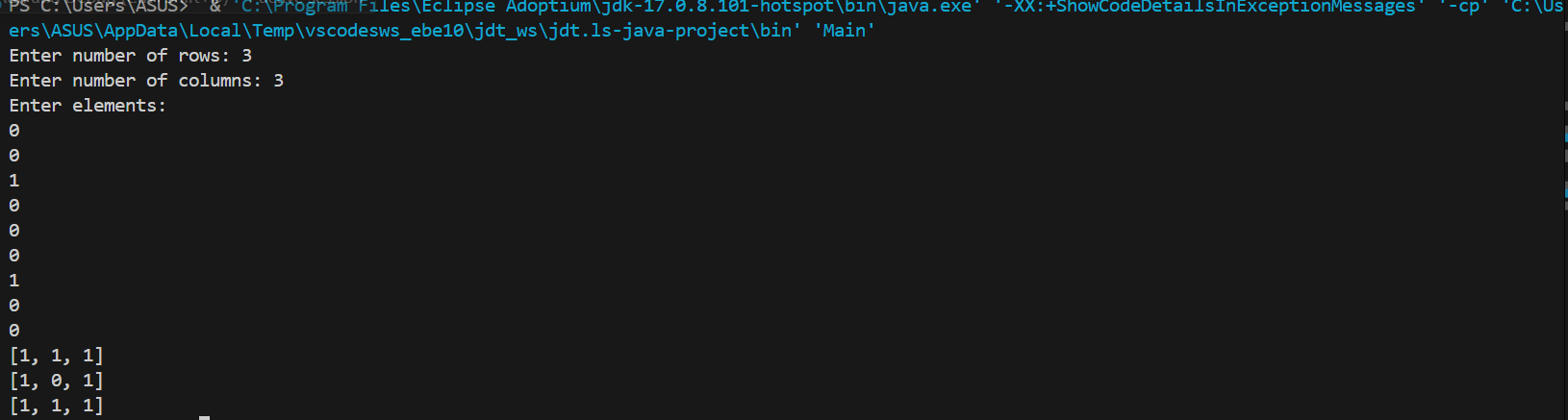
        modifyMatrix(mat);

        sc.close();

    }

}

Output:



Time complexity :O(N\*M)

Space Complexity :O(M + N).

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.\*;

class Main {

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

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

        int n = matrix.length;

        int m = matrix[0].length;

        int left = 0, top = 0, bottom = n - 1, right = m - 1;

        while (top <= bottom && left <= right) {

            for (int i = left; i <= right; i++) {

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

            }

            top++;

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

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

            }

            right--;

            if (top <= bottom) {

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

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

                }

                bottom--;

            }

            if (left <= right) {

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

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

                }

                left++;

            }

        }

        return ans;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of rows: ");

        int M = sc.nextInt();

        System.out.print("Enter number of columns: ");

        int N = sc.nextInt();

        int[][] mat = new int[M][N];

        System.out.println("Enter elements:");

        for (int i = 0; i < M; i++) {

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

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

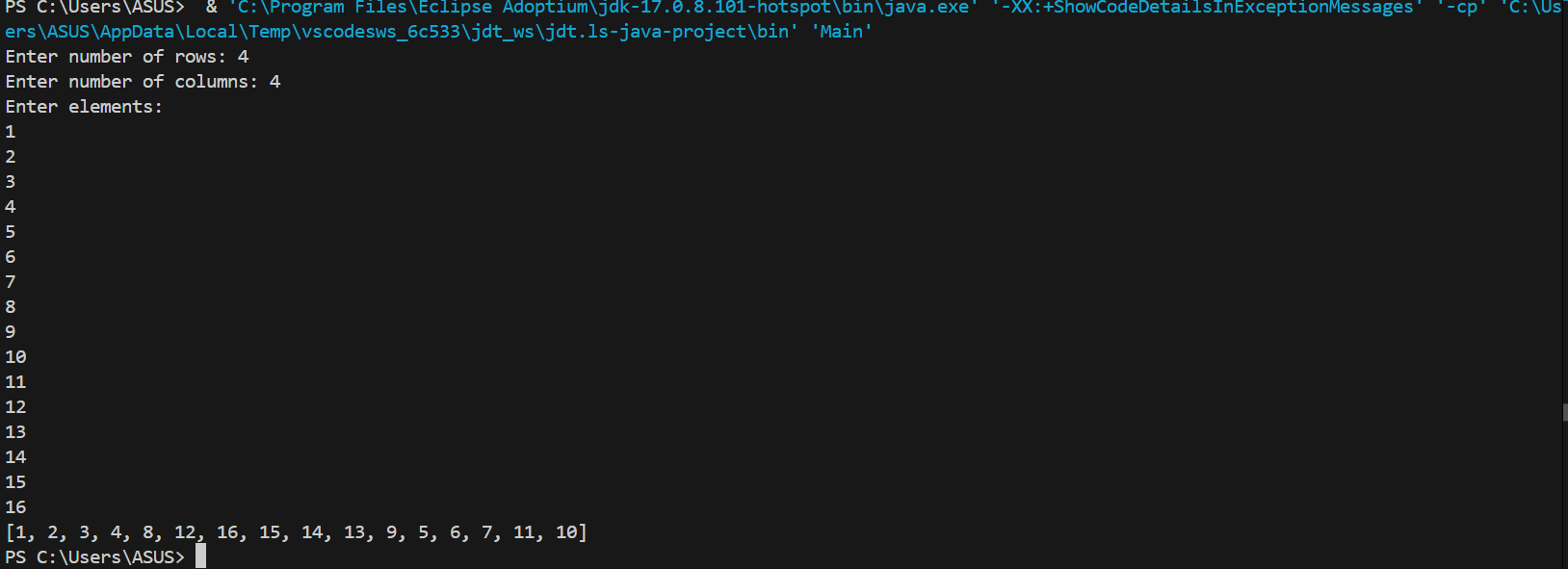
            }

        }

        System.out.println(spiralOrder(mat));

    }

}

Output:

**Time Complexity: O(M\*N)**

**Space Complexity:O(M\*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

Solution:

import java.util.\*;

public class Main{

    public static String isBalanced(String str) {

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

        for (char ch : str.toCharArray()) {

            if (ch == '(') stack.push(ch);

            else if (ch == ')' && (stack.isEmpty() || stack.pop() != '(')) {

                return ("Not Balanced");

        }}

        return "Balanced";

    }

    public static void main(String[] args) {

        String[] testStrings = {"((()))()()" , "())((())"};

        for (String str : testStrings) {

            System.out.println("Input: " + str);

            System.out.println("Balanced: " + isBalanced(str));

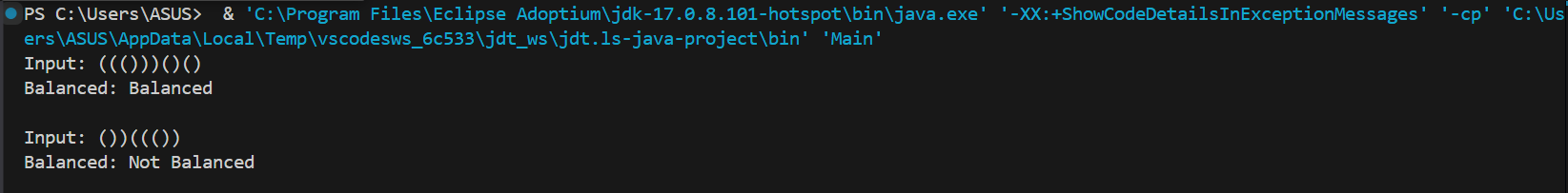
            System.out.println();

        }

    }

}

Output:



**Time Complexity: O(n)**

**Space Complexity:O(n)**

14.Check if two Strings are Anagrams of each other

Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the

two given strings are anagrams of each other or not. An anagram of a string is another string that

contains the same characters, only the order of characters can be different.

Input: s1 = “geeks” s2 = “kseeg”

Output: true

Explanation: Both the 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.\*;

public class Main {

    public static  boolean isAnagram(String s, String t) {

        char[] str1 = s.toCharArray();

        char[] str2 = t.toCharArray();

        Arrays.sort(str1);

        Arrays.sort(str2);

        String arr = new String(str1);

        String arr1 = new String(str2);

        return arr.equals(arr1);

    }

    public static void main(String[] args) {

        String s1 = "geeks";

        String t1 = "kseeg";

        String s2 = "hello";

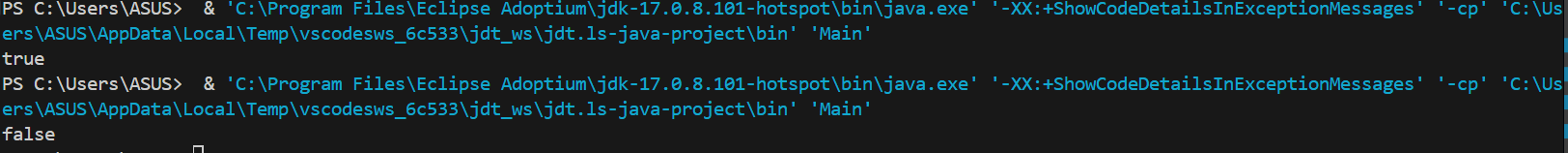
        String t2 = "world";

        System.out.println(isAnagram(s1, t1));

        System.out.println(isAnagram(s2, t2));     }

}

Output:



**Time Complexity: O(nLogn)**

**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 = “”

Output: “”

Solution:

public class Main  {

    private static String expandAroundCenter(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 String longestPalindrome(String s) {

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

        String longest = "";

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

            String oddPalindrome = expandAroundCenter(s, i, i);

            String evenPalindrome = expandAroundCenter(s, i, i + 1);

            if (oddPalindrome.length() > longest.length()) {

                longest = oddPalindrome;

            }

            if (evenPalindrome.length() > longest.length()) {

                longest = evenPalindrome;

            }

        }

        return longest;

    }

    public static void main(String[] args) {

        String str1 = "forgeeksskeegfor";

        String str2 = "Geeks";

        String str3 = "abc";

        String str4 = "";

        System.out.println(longestPalindrome(str1));

        System.out.println(longestPalindrome(str2));

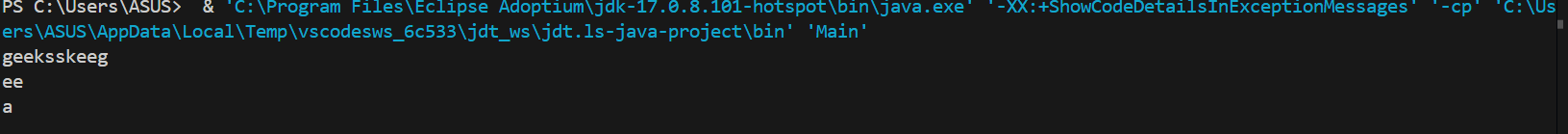
        System.out.println(longestPalindrome(str3));

        System.out.println(longestPalindrome(str4));

    }

}

Output:



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

**Space Complexity:O(n)**

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.\*;

public class Main {

    public static void main(String[] args) {

        String[] words1 = {"flower", "flow", "flight"};

        Arrays.sort(words1);

        String first = words1[0];

        String last = words1[words1.length - 1];

        StringBuilder ans = new StringBuilder();

        for (int i = 0; i < Math.min(first.length(), last.length()); i++) {

            if (first.charAt(i) != last.charAt(i)) {

                break;

            }

            ans.append(first.charAt(i));

        }

        if (ans.length() == 0) {

            System.out.println("-1");

        } else {

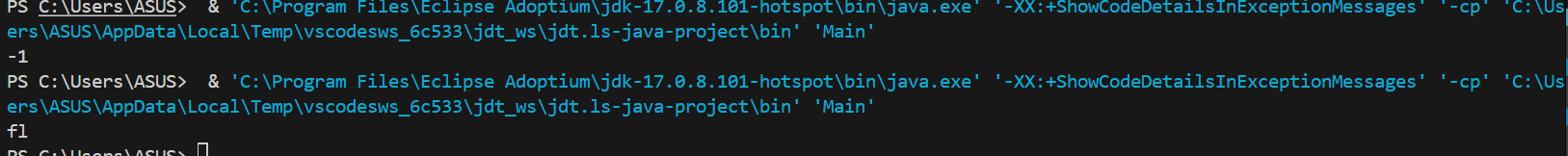
            System.out.println(ans.toString());

        }

    }

}

Output:



**Time Complexity: O(nLogn)**

**Space Complexity:O(n)**

17. Delete middle element of a stack

Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element

of it without using any additional data structure.

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

Output : Stack[] = [1, 2, 4, 5]

Solution:

import java.util.\*;

class JavaSample {

public static void fun(Stack<Integer> st, int indx , int sz) {

if (st.isEmpty()) return;

if (indx == sz / 2) {

st.pop();

return;

}

int temp = st.pop();

fun(st, indx + 1,sz);

st.push(temp);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

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

for (int i = 0; i < n; i++) {

st.push(sc.nextInt());

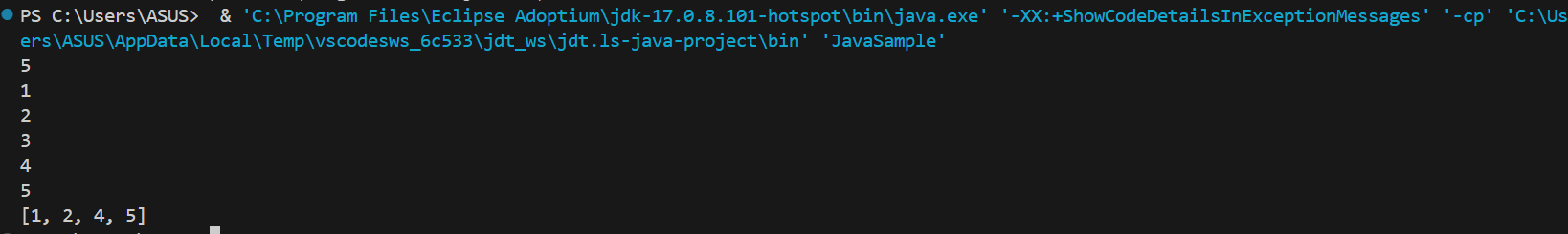
}

fun(st, 0 , n);

System.out.println(st);

}}

Output:



**Time Complexity: O(nLogn)**

**Space Complexity:O(1)**

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

Solution:

import java.util.Stack;

public class NextGreaterElement {

public static void printNextGreaterElement(int[] arr) {

int n = arr.length;

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

int[] nge = new int[n];

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

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

stack.pop();

}

if (stack.isEmpty()) {

nge[i] = -1;

} else {

nge[i] = stack.peek();

}

stack.push(arr[i]);

}

for (int i = 0; i < n; i++) {

System.out.println(arr[i] + " --> " + nge[i]);

}

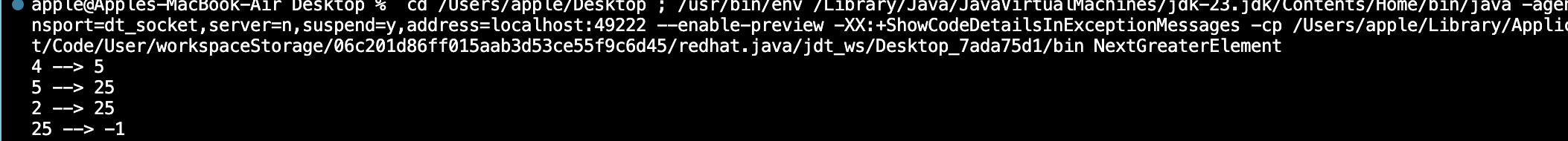
}

public static void main(String[] args) {

int[] arr1 = {4, 5, 2, 25};

printNextGreaterElement(arr1);

}}

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.

Solution:

import java.util.\*;

class BinaryTree {

static class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public static void printRightView(Node root) {

if (root == null) {

return;

}

Queue<Node> queue = new LinkedList<>();

queue.add(root);

while (!queue.isEmpty()) {

int nodeCount = queue.size();

for (int i = 1; i <= nodeCount; i++) {

Node currentNode = queue.poll();

if (i == nodeCount) {

System.out.print(currentNode.data + " ");

}

if (currentNode.left != null) {

queue.add(currentNode.left);

}

if (currentNode.right != null) {

queue.add(currentNode.right);

}

}

}

}

public static void main(String[] args) {

Node root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

root.left.right = new Node(5);

root.right.right = new Node(4);

printRightView(root);

}

}

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.**

**Solution:**

**class BinaryTree {**

**static class Node {**

**int data;**

**Node left, right;**

**Node(int item) {**

**data = item;**

**left = right = null;**

**}**

**}**

**public static int maxDepth(Node root) {**

**if (root == null) {**

**return 0;**

**}**

**int leftHeight = maxDepth(root.left);**

**int rightHeight = maxDepth(root.right);**

**return Math.max(leftHeight, rightHeight) + 1;**

**}**

**public static void main(String[] args) {**

**Node root = new Node(1);**

**root.left = new Node(2);**

**root.right = new Node(3);**

**root.left.left = new Node(4);**

**root.left.right = new Node(5);**

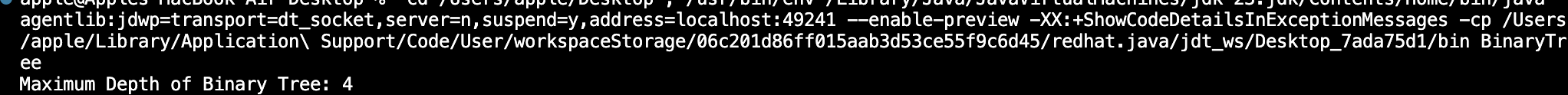
**root.left.left.left = new Node(6);**

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

**}**

**}**

**Output:**

****

**Time Complexity: O(n)**

**Space Complexity:O(logn)**