**DSA CODING PRACTICE PROBLEMS**

**9-11-2024**

**SHALINI M – CSE**

1. **Maximum Subarray Sum – Kadane‟s Algorithm:**

import java.util.\*;

public class Problem1{

public static int maxSubarraySum(int[] arr){

int result=arr[0];

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

int curr\_sum=0;

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

curr\_sum+=arr[j];

result= Math.max(result,curr\_sum);

}

}

return result;

}

public static void main(String args[]){

Scanner scanner = new Scanner(System.in);

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

String inputLine = scanner.nextLine();

String[] inputStrings = inputLine.split(" ");

int[] arr = new int[inputStrings.length];

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

arr[i] = Integer.parseInt(inputStrings[i]);

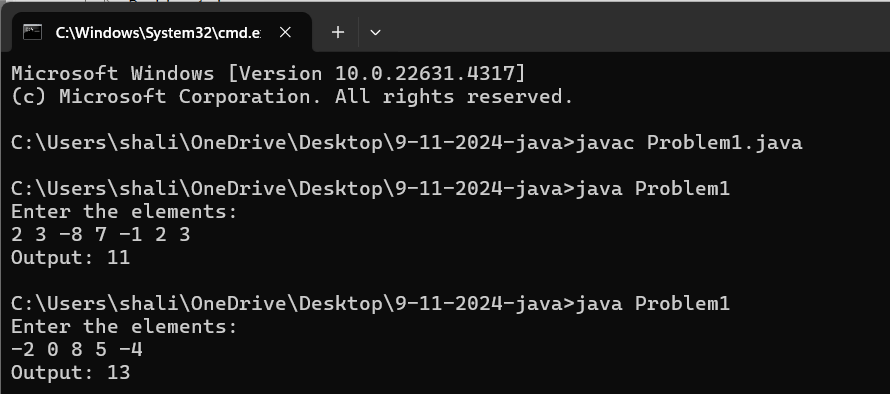
}

System.out.println("Output: "+ maxSubarraySum(arr));

scanner.close();

}

}



**Time Complexity:** O(n²)

**Space Complexity:** O(n)

1. **Maximum Product Subarray**

import java.util.\*;

public class Problem2 {

public static int maxProduct(int[] arr) {

int result = arr[0];

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

int curr\_sum = 1;

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

curr\_sum \*= arr[j];

result = Math.max(result, curr\_sum);

}

}

return result;

}

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

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

String inputLine = scanner.nextLine();

String[] inputStrings = inputLine.split(" ");

int[] arr = new int[inputStrings.length];

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

arr[i] = Integer.parseInt(inputStrings[i]);

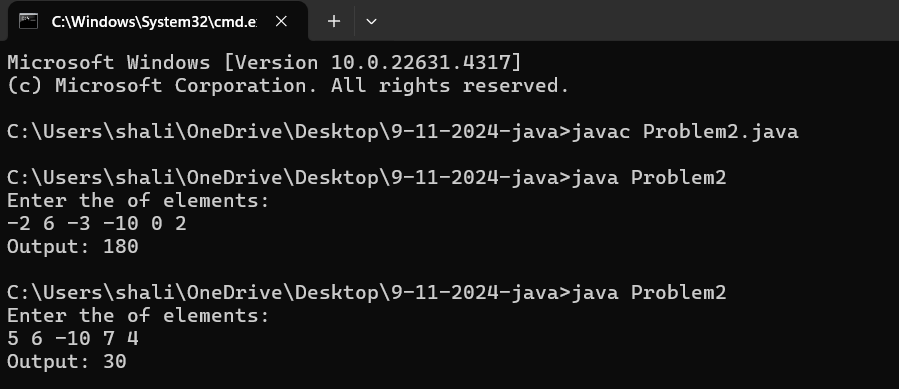
}

System.out.println("Output: " + maxProduct(arr));

scanner.close();

}

}



**Time Complexity:** O(n²)

**Space Complexity:** O(n)

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

import java.util.\*;

public class Problem3{

public static int findKey(int[] arr, int key){

int index=0;

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

if (arr[i]==key){

return i;

}

}

return -1;

}

public static void main(String args[]){

Scanner scanner =new Scanner(System.in);

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

String inputLine = scanner.nextLine();

String[] inputStrings = inputLine.split(" ");

int[] arr = new int[inputStrings.length];

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

arr[i] = Integer.parseInt(inputStrings[i]);

}

System.out.println("Enter the key:");

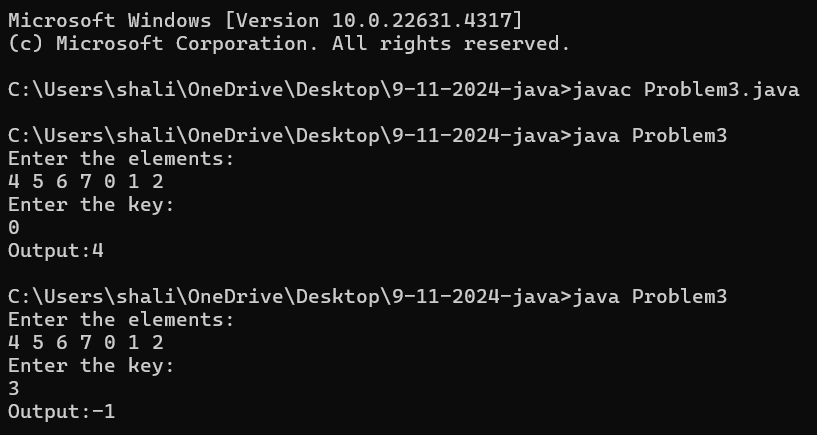
int key=scanner.nextInt();

System.out.println("Output:"+ findKey(arr,key));

scanner.close();

}

}



**Time Complexity:** O(n)

**Space Complexity:** O(n)

1. **Container with Most Water**

import java.util.\*;

public class Problem4{

public static int maxArea(int[] arr) {

int maxArea = 0;

int left = 0;

int right = arr.length - 1;

while (left < right) {

int height=Math.min(arr[left], arr[right]);

int width=right - left;

int area=height \* width;

maxArea=Math.max(maxArea, area);

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

left++;

} else {

right--;

}

}

return maxArea;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the heights:");

String inputLine = scanner.nextLine();

String[] inputStrings = inputLine.split(" ");

int[] arr = new int[inputStrings.length];

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

arr[i] = Integer.parseInt(inputStrings[i]);

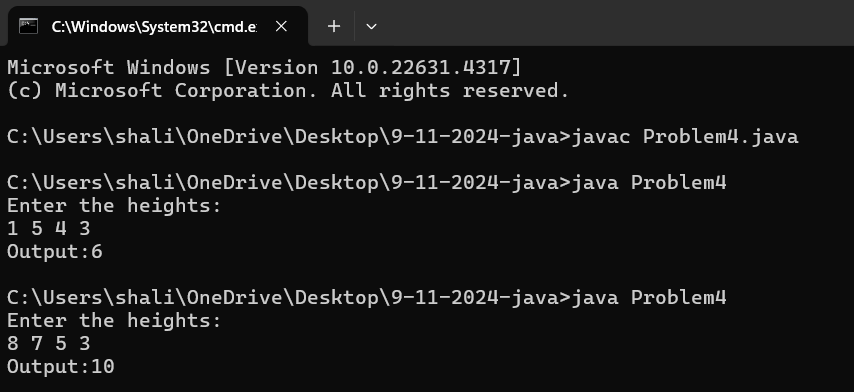
}

System.out.println("Output:"+maxArea(arr));

scanner.close();

}

}



**Time Complexity:** O(n)

**Space Complexity:** O(n)

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

import java.math.BigInteger;

import java.util.Scanner;

public class Problem5{

public 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 scanner = new Scanner(System.in);

System.out.print("Enter a number: ");

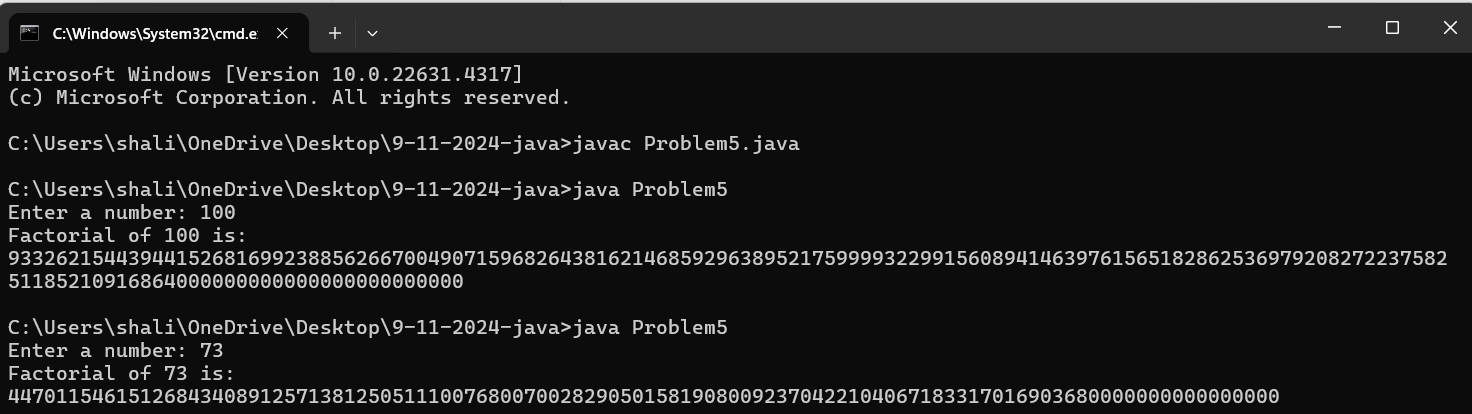
int number = scanner.nextInt();

System.out.println("Factorial of " + number + " is:\n" + factorial(number));

scanner.close();

}

}



**Time Complexity:** O(n)

**Space Complexity:** O(n)

1. **Trapping Rainwater Problem**

import java.util.\*;

public class Problem6{

public static int trapWater(int[] arr) {

int n = arr.length;

if (n == 0) {

return 0;

}

int[] leftMax = new int[n];

int[] rightMax = new int[n];

// Fill leftMax array

leftMax[0] = arr[0];

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

leftMax[i] = Math.max(leftMax[i - 1], arr[i]);

}

rightMax[n - 1] = arr[n - 1];

for (int i = n - 2; i >= 0; i--) {

rightMax[i] = Math.max(rightMax[i + 1], arr[i]);

}

int totalWater = 0;

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

// Water trapped on top of each bar is the difference between

// the height of the bar and the minimum of max heights on its left and right

totalWater += Math.min(leftMax[i], rightMax[i]) - arr[i];

}

return totalWater;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String inputLine = scanner.nextLine();

String[] inputStrings = inputLine.split(" ");

int[] arr = new int[inputStrings.length];

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

arr[i] = Integer.parseInt(inputStrings[i]);

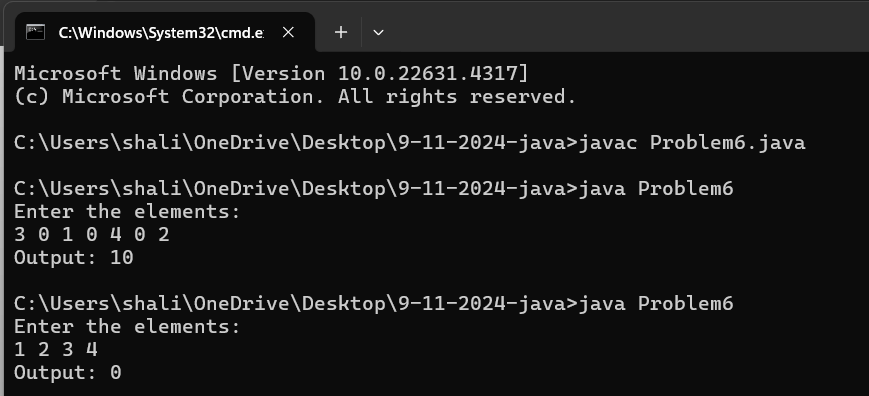
}

System.out.println("Output: "+ trapWater(arr));

scanner.close();

}

}



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

**Space Complexity:** O(n)

1. **Chocolate Distribution Problem**

import java.util.\*;

public class Problem7{

public static int minDiff(int[] arr,int m){

int n=arr.length;

if(n<m){

return -1;

}

if(n==0||m==0){

return 0;

}

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 scanner = new Scanner(System.in);

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

String inputLine = scanner.nextLine();

String[] inputStrings = inputLine.split(" ");

int[] arr = new int[inputStrings.length];

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

arr[i] = Integer.parseInt(inputStrings[i]);

}

System.out.println("Enter the nuber of students:");

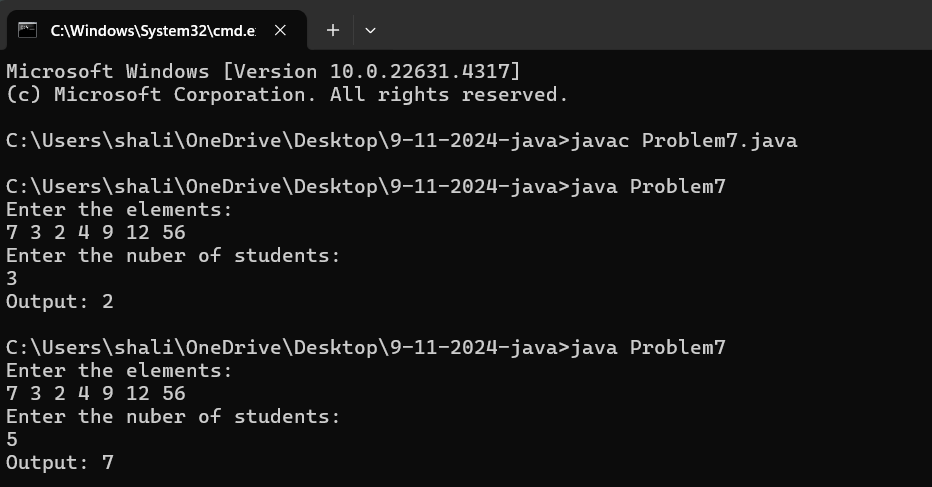
int m=scanner.nextInt();

System.out.println("Output: "+ minDiff(arr,m));

scanner.close();

}

}



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

**Space Complexity:** O(n)

1. **Merge Overlapping Intervals**

import java.util.\*;

class Problem8{

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

if (intervals.length <= 1) return intervals;

Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

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

int[] current = intervals[0];

merged.add(current);

for (int[] interval : intervals) {

if (interval[0] <= current[1]) {

current[1] = Math.max(current[1], interval[1]);

} else {

current = interval;

merged.add(current);

}

}

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

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[][] intervals = new int[n][2];

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

System.out.print("Enter start and end for interval " + (i + 1) + ": ");

intervals[i][0] = scanner.nextInt();

intervals[i][1] = scanner.nextInt();

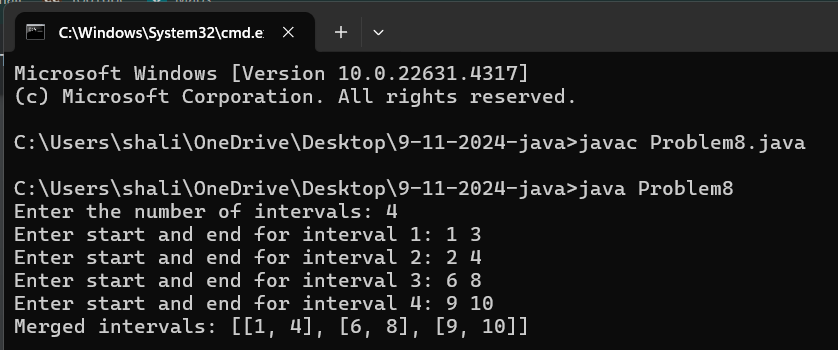
}

int[][] result = merge(intervals);

System.out.println("Merged intervals: " + Arrays.deepToString(result));

}

}



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

**Space Complexity:** O(n)

1. **A Boolean Matrix Question**

import java.util.Scanner;

public class Problem9{

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;

}

}

}

}

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

for (int[] row : mat) {

for (int cell : row) {

System.out.print(cell + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int M = scanner.nextInt();

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

int N = scanner.nextInt();

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

System.out.println("Enter the matrix elements (0 or 1):");

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

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

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

}

}

System.out.println("\nOriginal Matrix:");

printMatrix(mat);

modifyMatrix(mat);

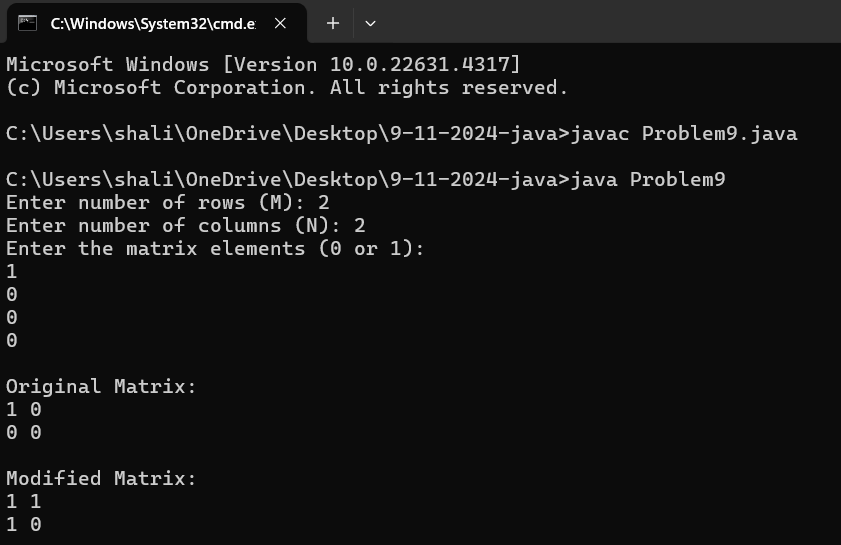
System.out.println("\nModified Matrix:");

printMatrix(mat);

scanner.close();

}

}



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

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

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

import java.util.Scanner;

public class Problem10 {

public static void printSpiral(int[][] matrix) {

int m = matrix.length;

int n = matrix[0].length;

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

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

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

System.out.print(matrix[top][i] + " ");

}

top++;

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

System.out.print(matrix[i][right] + " ");

}

right--;

if (top <= bottom) {

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

System.out.print(matrix[bottom][i] + " ");

}

bottom--;

}

if (left <= right) {

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

System.out.print(matrix[i][left] + " ");

}

left++;

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int m = scanner.nextInt();

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

int n = scanner.nextInt();

int[][] matrix = new int[m][n];

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

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

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

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

}

}

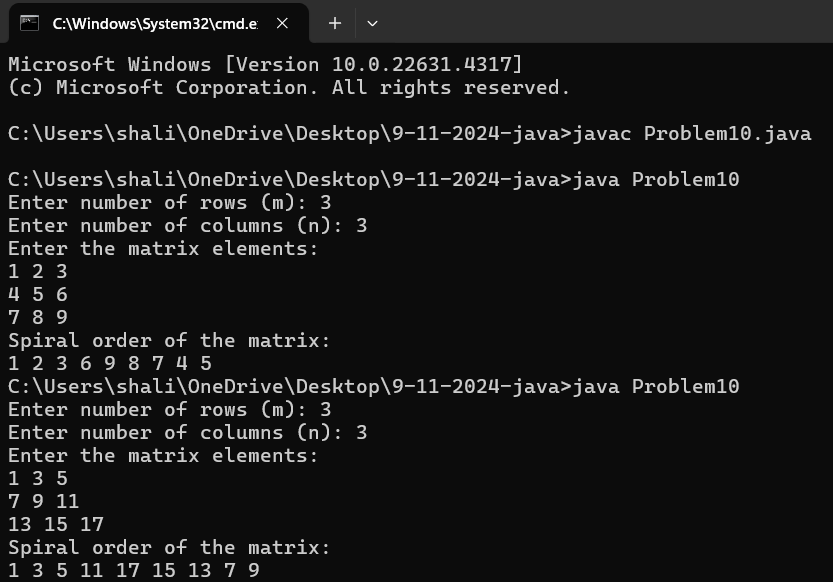
System.out.println("Spiral order of the matrix:");

printSpiral(matrix);

scanner.close();

}

}



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

**Space Complexity:** O(1)

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

import java.util.\*;

class Problem13{

public static String isBalanced(String str) {

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

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

if (ch == '(') {

stack.push(ch);

} else if (ch == ')') {

if (stack.isEmpty()){

return "Not Balanced";

}

stack.pop();

}

}

return stack.isEmpty() ? "Balanced" : "Not Balanced";

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

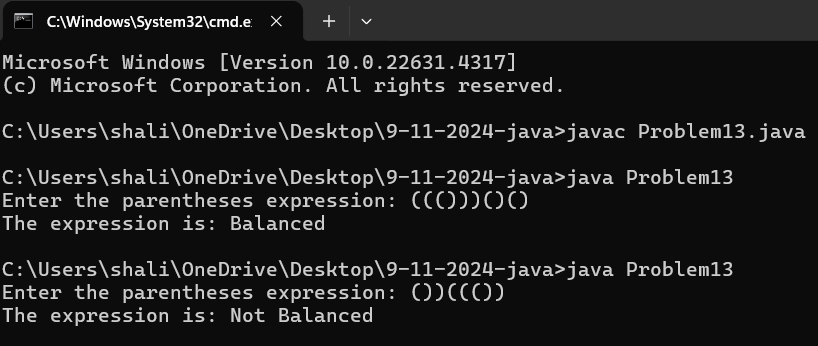
System.out.print("Enter the parentheses expression: ");

String str = scanner.nextLine();

System.out.println("The expression is: " + isBalanced(str));

}

}



**Time Complexity:** O(n)

**Space Complexity:** O(n)

1. **Check if two Strings are Anagrams of each other**

import java.util.\*;

class Problem14{

public static boolean areAnagrams(String s1, String s2) {

if (s1.length() != s2.length()) return false;

char[] arr1 = s1.toCharArray();

char[] arr2 = s2.toCharArray();

Arrays.sort(arr1);

Arrays.sort(arr2);

return Arrays.equals(arr1, arr2);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter first string: ");

String s1 = scanner.nextLine();

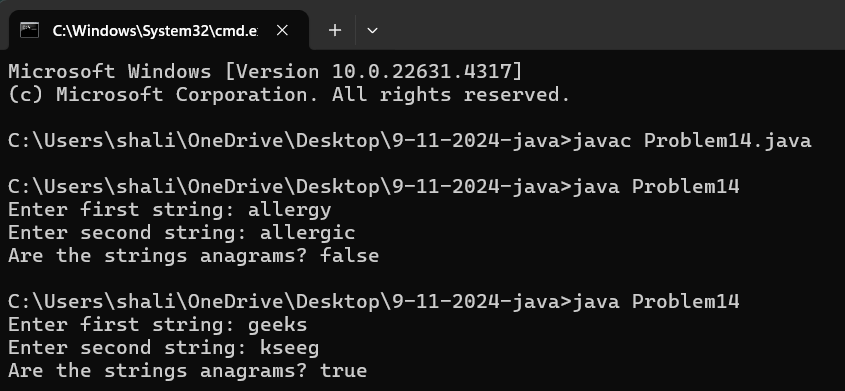
System.out.print("Enter second string: ");

String s2 = scanner.nextLine();

System.out.println("Are the strings anagrams? " + areAnagrams(s1, s2));

}

}



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

**Space Complexity:** O(n)

1. **Longest Palindromic Substring**

import java.util.Scanner;

class Problem15{

public static String longestPalindrome(String str) {

if (str == null || str.length() < 2) return str;

int start = 0, maxLength = 1;

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

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

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

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

if (len > maxLength) {

maxLength = len;

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

}

}

return str.substring(start, start + maxLength);

}

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

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

left--;

right++;

}

return right - left - 1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

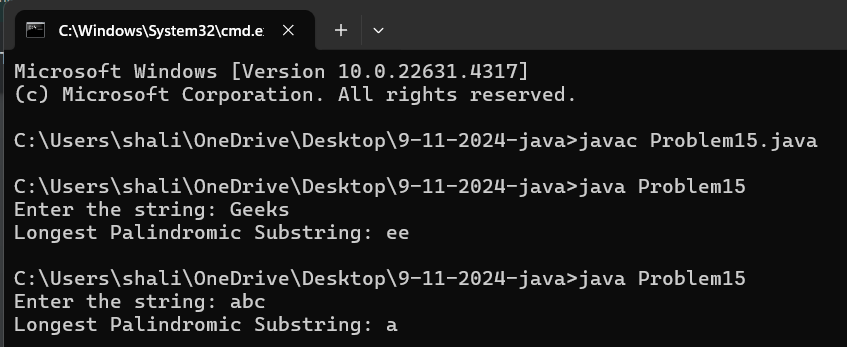
System.out.print("Enter the string: ");

String str = scanner.nextLine();

System.out.println("Longest Palindromic Substring: " + longestPalindrome(str));

}

}



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

**Space Complexity:** O(n)

1. **Longest Common Prefix using Sorting**

import java.util.\*;

class Problem16{

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 scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

scanner.nextLine();

String[] arr = new String[n];

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

System.out.print("Enter string " + (i + 1) + ": ");

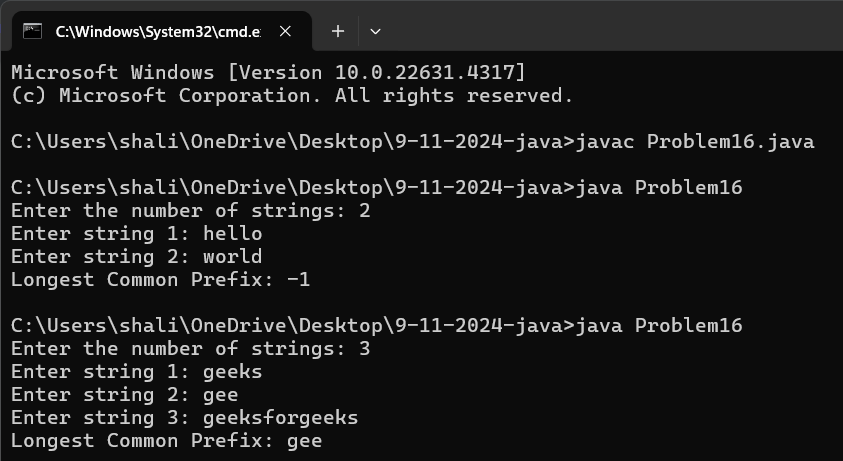
arr[i] = scanner.nextLine();

}

System.out.println("Longest Common Prefix: " + longestCommonPrefix(arr));

}

}



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

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

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

import java.util.\*;

class Problem17 {

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

if (stack.isEmpty() || currentIndex == size) return;

int element = stack.pop();

deleteMiddle(stack, size, currentIndex + 1);

if (currentIndex != size / 2) {

stack.push(element);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the stack: ");

int n = scanner.nextInt();

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

System.out.print("Enter the elements of the stack: ");

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

stack.push(scanner.nextInt());

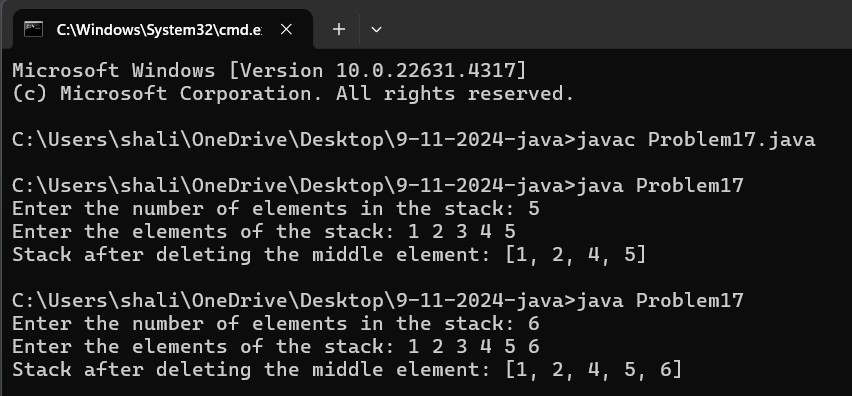
}

deleteMiddle(stack, n, 0);

System.out.println("Stack after deleting the middle element: " + stack);

}

}



**Time Complexity:** O(n)

**Space Complexity:** O(n)

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

import java.util.Stack;

import java.util.Scanner;

public class Problem18{

public static void printNextGreater(int[] arr) {

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

int n = arr.length;

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

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

int index = stack.pop();

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

}

stack.push(i);

}

while (!stack.isEmpty()) {

int index = stack.pop();

System.out.println(arr[index] + " --> -1");

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

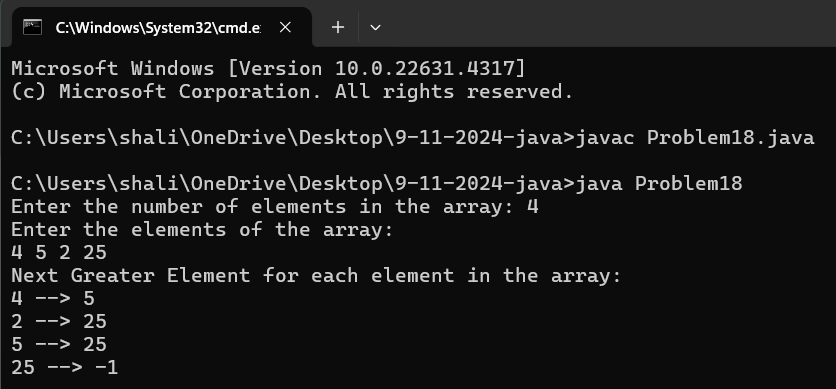
System.out.println("Next Greater Element for each element in the array:");

printNextGreater(arr);

scanner.close();

}

}



**Time Complexity:** O(n)

**Space Complexity:** O(n)

1. **Print Right View of a Binary Tree**

import java.util.ArrayList;

import java.util.Scanner;

class TreeNode {

int value;

TreeNode left, right;

TreeNode(int x) {

value = x;

}

}

public class Problem19 {

static void getRightView(TreeNode node, int level, int[] maxLevel, ArrayList<Integer> view) {

if (node == null) return;

if (level > maxLevel[0]) {

view.add(node.value);

maxLevel[0] = level;

}

getRightView(node.right, level + 1, maxLevel, view);

getRightView(node.left, level + 1, maxLevel, view);

}

static ArrayList<Integer> rightView(TreeNode root) {

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

getRightView(root, 0, new int[]{-1}, rightSideView);

return rightSideView;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

TreeNode root = new TreeNode(scanner.nextInt());

root.left = new TreeNode(scanner.nextInt());

root.right = new TreeNode(scanner.nextInt());

root.right.left = new TreeNode(scanner.nextInt());

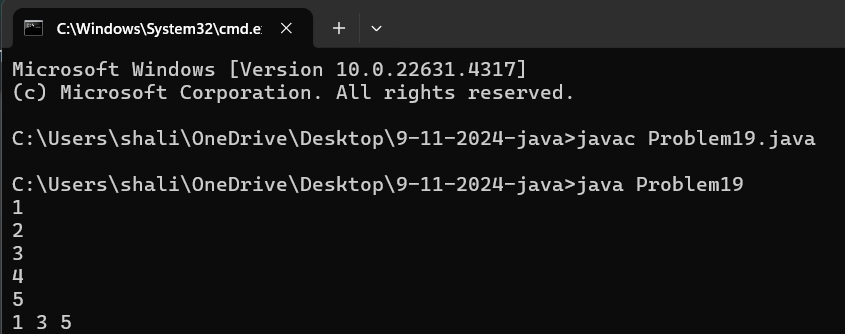
root.right.right = new TreeNode(scanner.nextInt());

ArrayList<Integer> result = rightView(root);

result.forEach(v -> System.out.print(v + " "));

}

}



**Time Complexity:** O(n)

**Space Complexity:** O(h)

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

import java.util.Scanner;

class TreeNode {

int value;

TreeNode left, right;

TreeNode(int x) {

value = x;

}

}

public class Problem20 {

static int findHeight(TreeNode node) {

if (node == null) return 0;

return Math.max(findHeight(node.left), findHeight(node.right)) + 1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

TreeNode root = new TreeNode(scanner.nextInt());

buildTree(root, scanner);

System.out.println("Maximum Depth/Height of the tree: " + findHeight(root));

}

static void buildTree(TreeNode node, Scanner scanner) {

int leftValue = scanner.nextInt();

if (leftValue != -1) {

node.left = new TreeNode(leftValue);

buildTree(node.left, scanner);

}

int rightValue = scanner.nextInt();

if (rightValue != -1) {

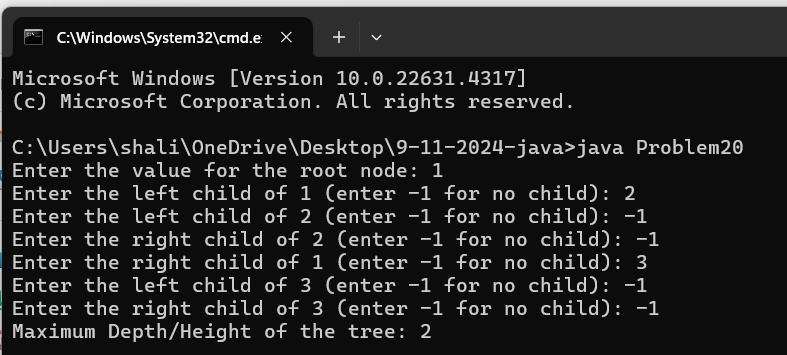
node.right = new TreeNode(rightValue);

buildTree(node.right, scanner);

}

}

}



**Time Complexity:** O(n)

**Space Complexity:** O(h)