**09-11-2024 DSA PRACTICE PROBLEMS SET 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.**

**CODE:**

import *java.util.\**;

*class* MaxSubArraySum {

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

*ArrayList*<*Integer*> arr = new *ArrayList*<>();

*Scanner* scan = new Scanner(System.in);

*int* n = scan.nextInt();

        for(*int* i=0;i<n;i++) {

*int* ele = scan.nextInt();

            arr.add(ele);

        }

*int* sum=0;

*int* max= Integer.MIN\_VALUE;

        for(*int* i=0;i<n;i++) {

            sum+=arr.get(i);

            if (sum<0){

                sum=0;

            }

            max = Math.max(max, sum);

        }

*int* cnt=0;

        for (*int* i=0;i<n;i++) {

            if (arr.get(i)<0) {

                cnt++;

            }

        }

        if (cnt==n) {

            System.out.println("Max Subarray Sum:"+ Collections.max(arr));

        }

        else {

            System.out.println("Max Subarray Sum:" + max);

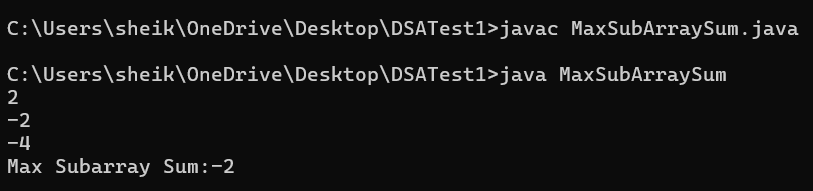
        }

        scan.close();

    }

}

**OUPUT:**



**TIME COMPLEXITY: O(N)**

**2. Maximum Product Subarray**

Given an integer array, the task is to find the maximum product of any subarray.

**CODE:**

import java.util.\*;

class MaxProduct {

    public static void main(String[] args) {

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

        Scanner scan = new Scanner(System.in);

        int n = scan.nextInt();

        for(int i = 0; i < n; i++) {

            int ele = scan.nextInt();

            nums.add(ele);

        }

        int prod1 = nums.get(0);

        int prod2 = nums.get(0);

        int ans = nums.get(0);

        for (int i = 1; i < nums.size(); i++) {

            int temp = Math.max(nums.get(i), Math.max(prod1 \* nums.get(i), prod2 \* nums.get(i)));

            prod2 = Math.min(nums.get(i), Math.min(prod1 \* nums.get(i), prod2 \* nums.get(i)));

            prod1 = temp;

            ans = Math.max(prod1, ans);

        }

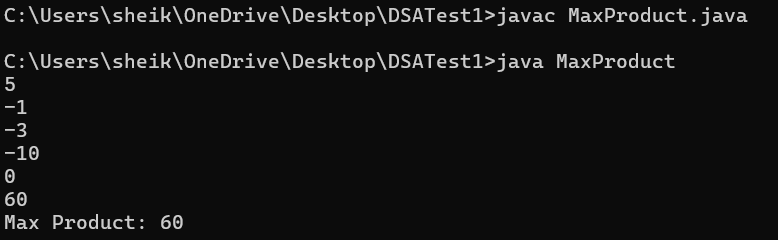
        System.out.println("Max Product: " + ans);

        scan.close();

    }

}

**OUTPUT:**



**TIME COMPLEXITY: O(N)**

**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.

**CODE:**

import java.util.\*;

class SearchInSorted {

    public static void main(String[] args) {

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

        Scanner scan = new Scanner(System.in);

        int n = scan.nextInt();

        for(int i = 0; i < n; i++) {

            int ele = scan.nextInt();

            nums.add(ele);

        }

        int target = scan.nextInt();

        int result = searchInRotatedArray(nums, target);

        if (result!=-1) {

        System.out.println("Index of Target: " + result);

        }

        else{

            System.out.println("Target not found");

        }

        scan.close();

    }

    public static int searchInRotatedArray(ArrayList<Integer> nums, int target) {

        int low = 0;

        int high = nums.size() - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            if (nums.get(mid) == target) {

                return mid;

            }

            if (nums.get(low) <= nums.get(mid)) {

                if (nums.get(low) <= target && target <= nums.get(mid)) {

                    high = mid - 1;

                } else {

                    low = mid + 1;

                }

            } else {

                if (nums.get(mid) <= target && target <= nums.get(high)) {

                    low = mid + 1;

                } else {

                    high = mid - 1;

                }

            }

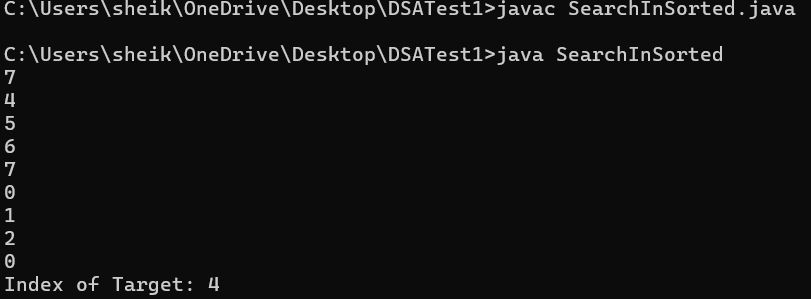
        }

        return -1;

    }

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(log(N))**

**4.Container with Most Water**

Given n non-negative integers a1, a2,..., an where each represents a point at coordinate (i, ai). ‘n ‘vertical lines are drawn such that the two endpoints of line i is at (i, a;) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

The program should return an integer which corresponds to the maximum area of water that can be contained (maximum area instead of maximum volume sounds weird but this is the 2D plane we are working with for simplicity).

Note: You may not slant the container.

**CODE:**

import java.util.\*;

class WaterContainer {

    public static void main(String[] args) {

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

        Scanner scan = new Scanner(System.in);

        int n = scan.nextInt();

        for(int i = 0; i < n; i++) {

            int ele = scan.nextInt();

            nums.add(ele);

        }

        int i=0; int j=n-1; int m=0;

        while(i<j){

            int curr=(j-i)\*Math.min(nums.get(i),nums.get(j));

            m = Math.max(curr,m);

            if (nums.get(i)<nums.get(j)){

                i++;

            }

            else{

                j--;

            }

        }

        System.out.println("Maximum:"+m);

    }

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

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

**CODE:**

import java.math.BigInteger;

import java.util.Scanner;

class Factorial {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

BigInteger[] dp = new BigInteger[n + 1];

dp[0] = BigInteger.ONE;

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

dp[i] = dp[i - 1].multiply(BigInteger.valueOf(i));

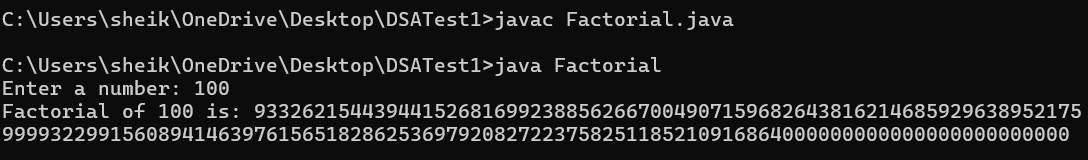
}

System.out.println("Factorial of " + n + " is: " + dp[n]);

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**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.

**CODE:**

import java.util.ArrayList;

import java.util.Scanner;

class TrapRain {

public static void main(String[] args) {

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

Scanner scan = new Scanner(System.in);

int n = scan.nextInt();

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

int ele = scan.nextInt();

arr.add(ele);

}

int[] psum = new int[n];

int[] ssum = new int[n];

psum[0] = arr.get(0);

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

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

}

ssum[n - 1] = arr.get(n - 1);

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

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

}

int water= 0;

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

water += Math.min(psum[i], ssum[i]) - arr.get(i);

}

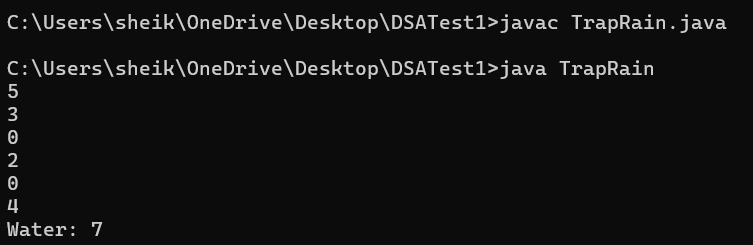
System.out.println("Water: "+water);

scan.close();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**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**:**

**CODE:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Scanner;

class Chocolate {

public static void main(String[] args) {

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

Scanner scan = new Scanner(System.in);

int n = scan.nextInt();

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

int ele = scan.nextInt();

arr.add(ele);

}

System.out.print("Enter no. of students: ");

int m = scan.nextInt();

Collections.sort(arr);

int mini = Integer.MAX\_VALUE;

for(int i=0;i+m-1<n;i++){

int temp = arr.get(i+m-1) - arr.get(i);

if (temp<mini){

mini = temp;

}

}

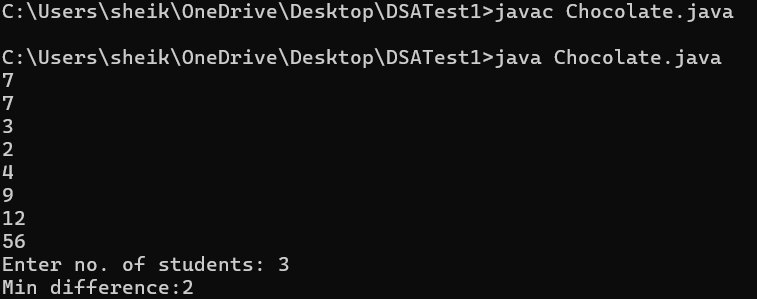
System.out.println("Min difference:"+mini);

scan.close();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N\*log(N))**

**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.

**CODE:**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

class MergedInterval {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

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

int n = scan.nextInt();

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

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

System.out.print("Enter pair :");

int first = scan.nextInt();

int second = scan.nextInt();

pair.add(first);

pair.add(second);

intervals.add(pair);

}

intervals.sort((a, b) -> Integer.compare(a.get(0), b.get(0)));

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

ArrayList<Integer> current = intervals.get(0);

for (int i = 1; i < intervals.size(); i++) {

ArrayList<Integer> interval = intervals.get(i);

if (interval.get(0) <= current.get(1)) {

current.set(1, Math.max(current.get(1), interval.get(1)));

} else {

merge.add(current);

current = interval;

}

}

merge.add(current);

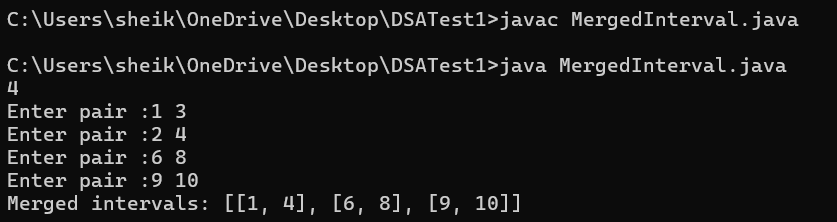
System.out.println("Merged intervals: "+merge);

scan.close();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N\*log(N)) + 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.

**CODE:**

import java.util.Scanner;

public class BoolMatrix {

public 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 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: ");

int M = scanner.nextInt();

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

int N = scanner.nextInt();

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

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

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:");

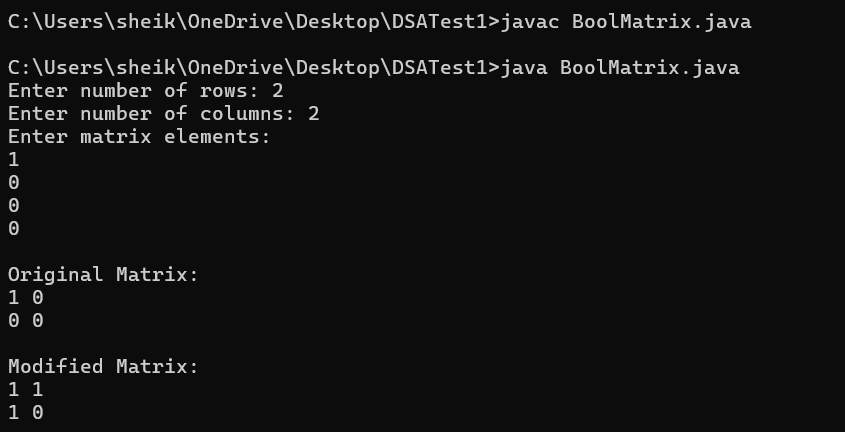
printMatrix(mat);

scanner.close();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(MXN)**

**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.

**CODE:**

import java.util.Scanner;

class SpiralMatrix {

public static void printSpiral(int[][] matrix, int M, int N) {

int top = 0, bottom = M - 1;

int 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++;

}

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int M = scanner.nextInt();

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

int N = scanner.nextInt();

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

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

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

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

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

}

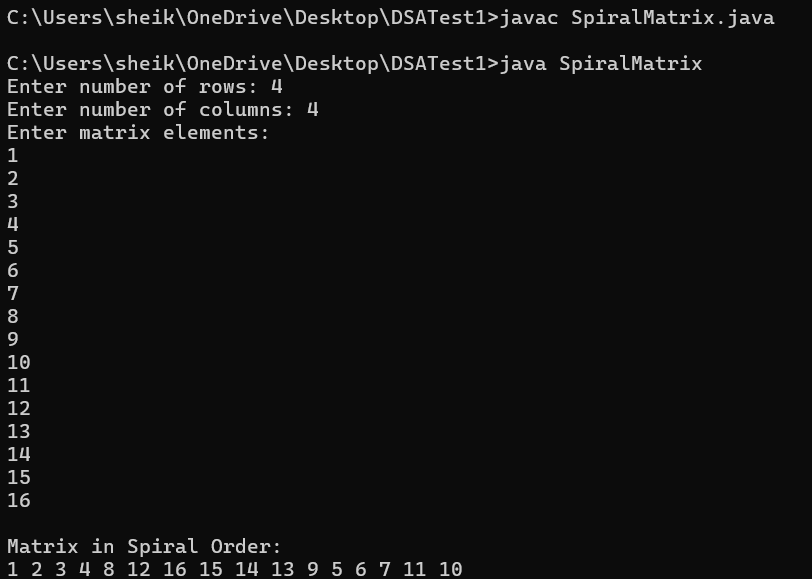
}

System.out.println("\nMatrix in Spiral Order:");

printSpiral(matrix, M, N);

scanner.close();

}

**OUTPUT:**

**TIME COMPLEXITY: O(MXN)**

**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.

**CODE:**

import java.util.Scanner;

public class BalancedParentheses {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a parentheses string: ");

String str = scanner.nextLine();

int cnt = 0;

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

if (ch == '(') {

cnt++;

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

cnt--;

}

if (cnt < 0) {

System.out.println("Not Balanced");

}

}

if (cnt == 0) {

System.out.println("Balanced");

} else {

System.out.println("Not Balanced");

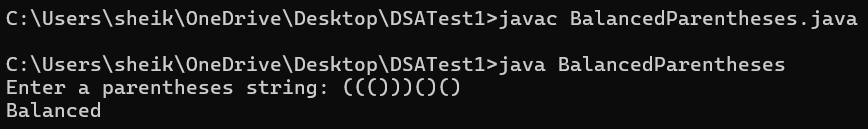
}

scanner.close();

}

}

**OUTPUT:**

****

**TIME 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.

**CODE:**

import java.util.Scanner;

public class AnagramChecker {

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

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

return false;

}

int[] charCount = new int[26];

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

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

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

}

for (int count : charCount) {

if (count != 0) {

return false;

}

}

return true;

}

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();

if (areAnagrams(s1, s2)) {

System.out.println("True");

} else {

System.out.println("False");

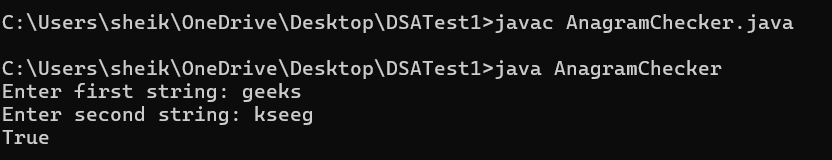
}

scanner.close();

}

}

**OUTPUT:**

****

**TIME 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.

**CODE:**

import java.util.Scanner;

public class LongPalindrome {

public String longestPalindrome(String s) {

if (s.length() <= 1) {

return s;

}

int maxLen = 1;

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

s = "#" + s.replaceAll("", "#") + "#";

int[] dp = new int[s.length()];

int center = 0;

int right = 0;

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

if (i < right) {

dp[i] = Math.min(right - i, dp[2 \* center - i]);

}

while (i - dp[i] - 1 >= 0 && i + dp[i] + 1 < s.length() && s.charAt(i - dp[i] - 1) == s.charAt(i + dp[i] + 1)) {

dp[i]++;

}

if (i + dp[i] > right) {

center = i;

right = i + dp[i];

}

if (dp[i] > maxLen) {

maxLen = dp[i];

maxStr = s.substring(i - dp[i], i + dp[i] + 1).replaceAll("#", "");

}

}

return maxStr;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String input = scanner.nextLine();

LongPalindrome sol = new LongPalindrome();

String result = sol.longestPalindrome(input);

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

scanner.close();

}

}

**OUTPUT:**

****

**TIME 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”.

**CODE:**

import java.util.Arrays;

import java.util.Scanner;

public class LongPrefix {

public String longestCommonPrefix(String[] arr) {

if (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 first.substring(0, i);

} else {

return "-1";

}

}

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];

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

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

arr[i] = scanner.nextLine();

}

LongPrefix longpre = new LongPrefix();

String result = longpre.longestCommonPrefix(arr);

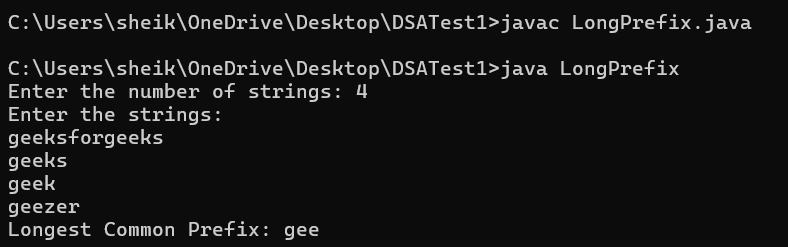
System.out.println("Longest Common Prefix: " + result);

scanner.close();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(N\*log(N) + M)**

**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.

**CODE:**

import java.util.Stack;

import java.util.Scanner;

public class StackMiddle {

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

if (currentIndex == size / 2) {

stack.pop();

return;

}

int top = stack.pop();

deleteMiddle(stack, currentIndex + 1, size);

stack.push(top);

}

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.println("Enter the elements of the stack:");

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

stack.push(scanner.nextInt());

}

System.out.println("Original Stack: " + stack);

deleteMiddle(stack, 0, stack.size());

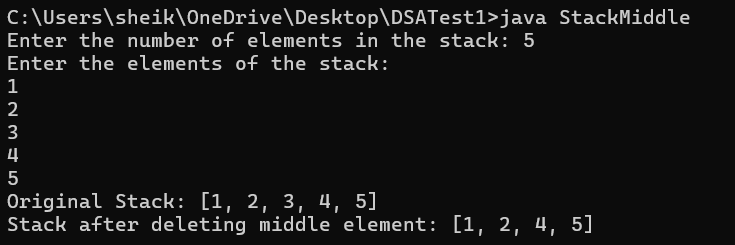
System.out.println("Stack after deleting middle element: " + stack);

scanner.close();

}

}

**OUTPUT:**

****

**TIME 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.

**CODE:**

import java.util.Stack;

import java.util.Scanner;

import java.util.ArrayList;

import java.util.List;

public class NGE {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of 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();

}

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

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

for (int i = arr.length - 1; i >= 0; i--) {

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

stack.pop();

}

if (!stack.isEmpty()) {

result.add(arr[i] + " --> " + stack.peek());

} else {

result.add(arr[i] + " --> -1");

}

stack.push(arr[i]);

}

for (int i = result.size() - 1; i >= 0; i--) {

System.out.println(result.get(i));

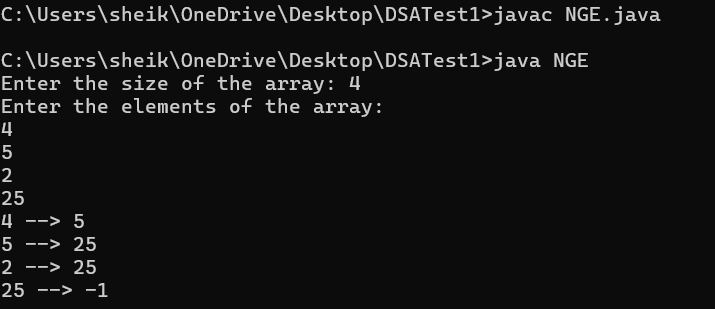
}

scanner.close();

}

}

**OUTPUT:**

****

**TIME 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 Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

public class BinaryTree {

Node root;

public void rightView() {

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) {

BinaryTree tree = new BinaryTree();

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.right.right = new Node(6);

tree.root.left.left.left = new Node(7);

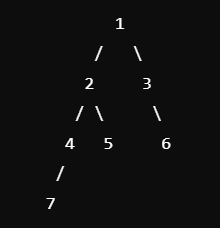
System.out.print("Right View of the Binary Tree: ");

tree.rightView();

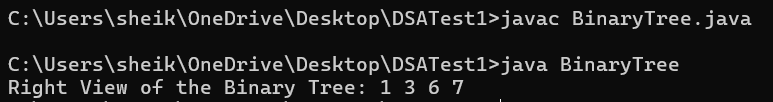
}

}

**INPUT:**

****

**OUTPUT:**

****

**TIME 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 Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

public class MaxDepth {

Node root;

public int maxDepth(Node node) {

if (node == null) {

return 0;

}

int leftHeight = maxDepth(node.left);

int rightHeight = maxDepth(node.right);

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

}

public static void main(String[] args) {

MaxDepth tree = new MaxDepth();

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.right.left = new Node(6);

tree.root.right.right = new Node(7);

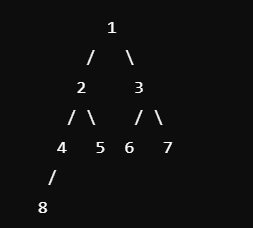
tree.root.left.left.left = new Node(8);

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

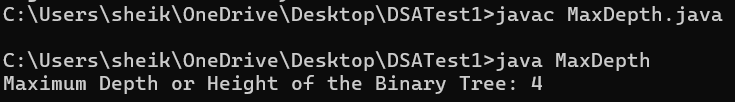
}

}

**INPUT:**

****

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**