**PAVADARENI R AIDS**

**09-11-2024**

**Coding practice Problems**

1. 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 que1{

public static void main(String[] args){

Scanner s = new Scanner(System.in);

System.out.print("Enter the array size:");

int n = s.nextInt();

int[] arr = new int[n];

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

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

arr[i] = s.nextInt();

}

int maxElement = arr[0];

int result = arr[0];

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

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

result=Math.max(maxElement,result);

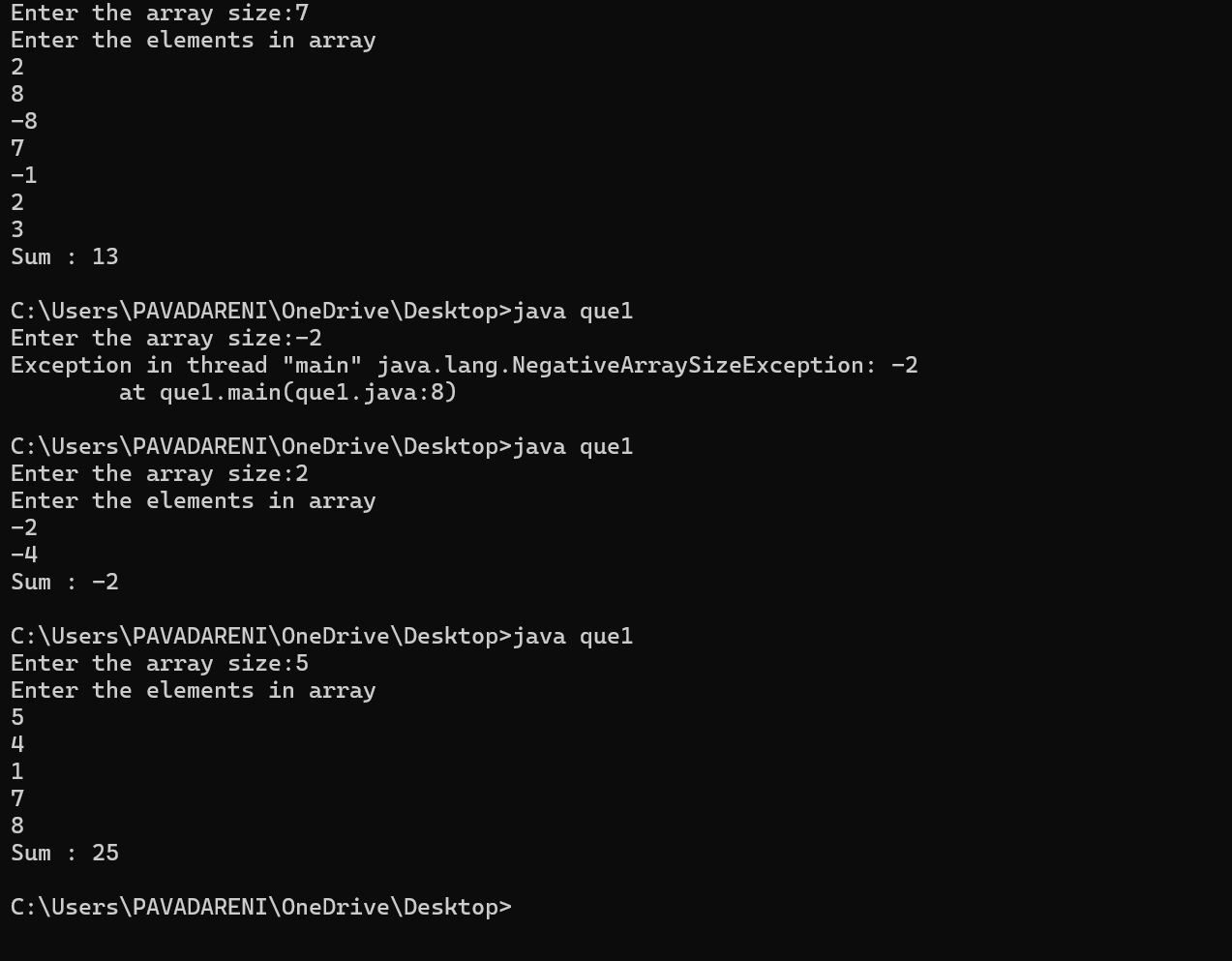
}

System.out.println("Sum : " + result);

}

}

**OUTPUT**



**TIME COMPLEXITY**

O(n)   
One time Traversal

The basic operation in **Sum.** The number of times it is done is **one.** Hence Time complexity is **O(n)** where n is the size of the array

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

**CODE**

import java.util.\*;

class que2{

public static void main(String[] args){

Scanner s = new Scanner(System.in);

System.out.print("Enter the array size:");

int n = s.nextInt();

int[] arr = new int[n];

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

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

arr[i] = s.nextInt();

}

int result = arr[0];

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

int mul = 1;

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

mul \*= arr[j];

result=Math.max(mul ,result);

}

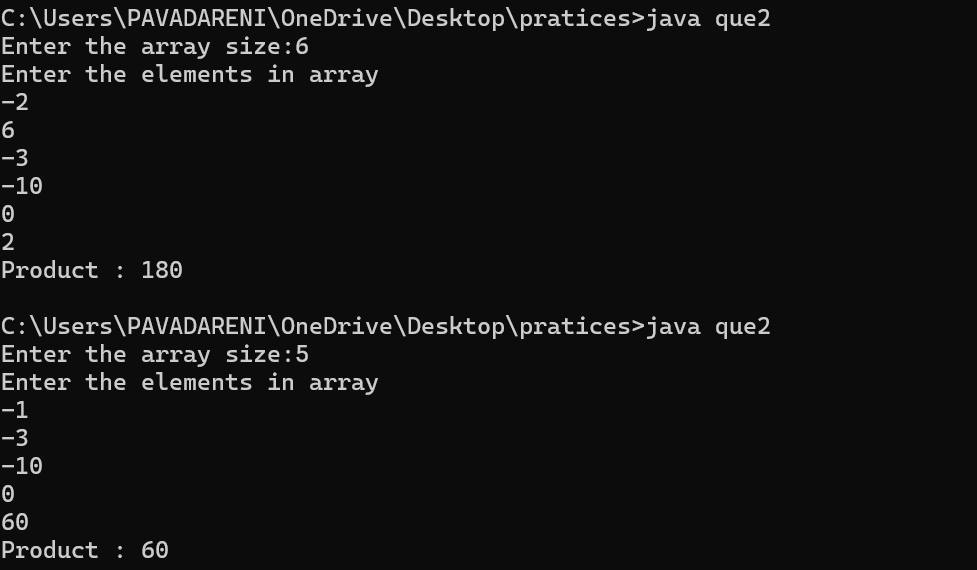
}

System.out.println("Product : " + result);

}

}

**OUTPUT**



**TIME COMPLEXITY**

O(n^2)

The basic operation in **Product.** The elements are traversed more than one time… Every element is visited at least 2 times. The n\*n times the loop runs. Hence the time complexity is O(n^2)

1. Search in a sorted and rotated Array

**CODE**

import java.util.Scanner;

class que3 {

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

System.out.print("Enter the rotated array size: ");

int n = s.nextInt();

int[] arr = new int[n];

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

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

arr[i] = s.nextInt();

}

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

int x = s.nextInt();

int l = 0;

int r = n - 1;

int mid;

boolean found = false;

while (l <= r) {

mid = (l + r) / 2;

if (arr[mid] == x) {

System.out.println(mid);

found = true;

break;

}

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

if (arr[l] <= x && x < arr[mid]) {

r = mid - 1;

} else {

l = mid + 1;

}

} else {

if (arr[mid] < x && x <= arr[r]) {

l = mid + 1;

} else {

r = mid - 1;

}

}

}

if (!found) {

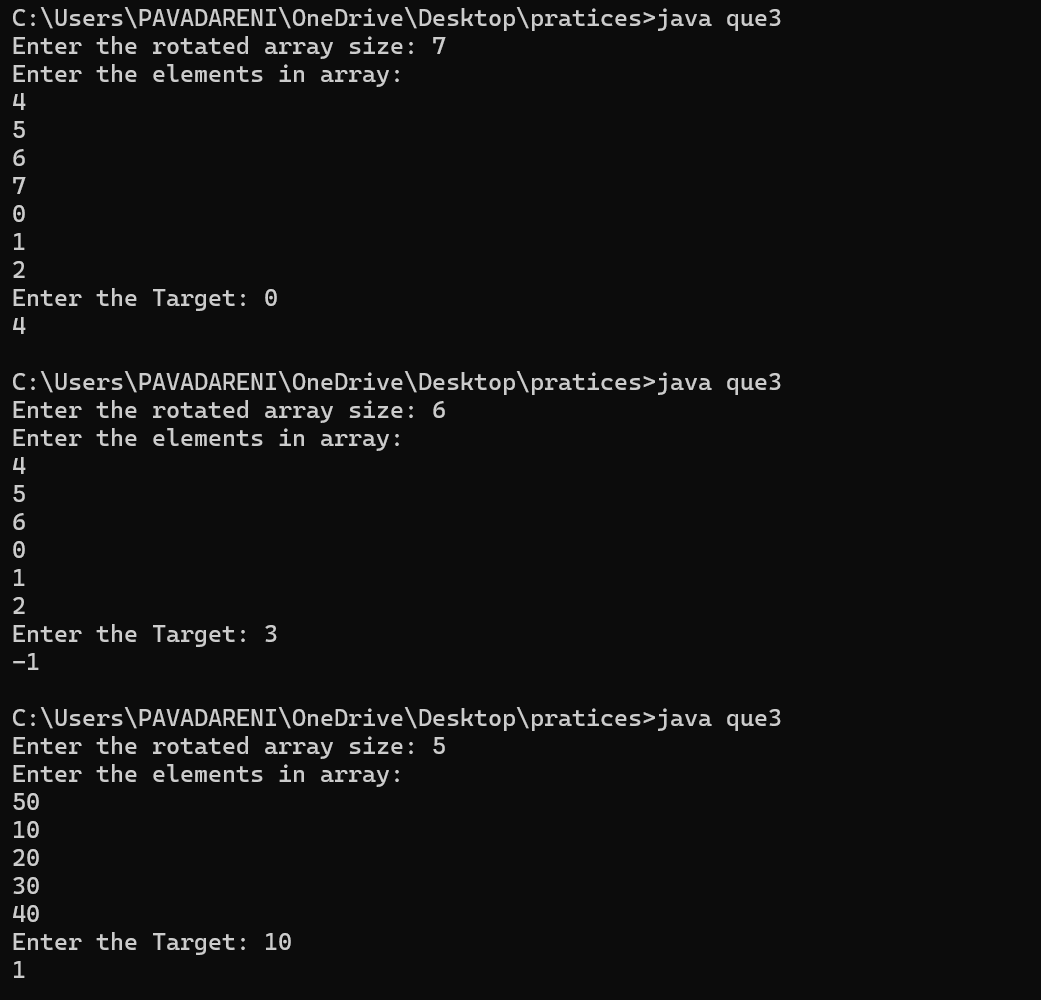
System.out.println(-1);

}

}

}

**OUTPUT**

****

**Time Complexity**

Binary Search approach is used.. Hence time complexity is **O(log n)** Iteation reduce by **half of N**

1. Container with Most Water

**CODE**

import java.util.Scanner;

class que4 {

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

System.out.print("Enter the array size:");

int n = s.nextInt();

int[] arr = new int[n];

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

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

arr[i] = s.nextInt();

}

int left = 0;

int right = n - 1;

int area = 0;

while (left < right) {

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

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

left += 1;

else

right -= 1;

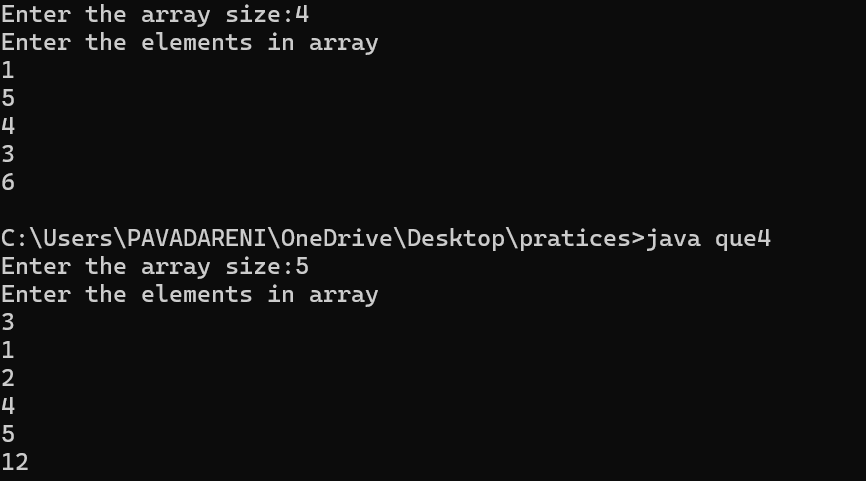
}

System.out.println(area);

}

}

**OUTPUT**

****

**TIME COMPLEXITY : O(n)**

The array is traversed only one time. Hence the complexity remains O(n).

1. Find the Factorial of a large number

**CODE**

import java.util.\*;

import java.math.\*;

class que5{

public static void main(String args[]){

Scanner s = new Scanner(System.in);

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

int n=s.nextInt();

BigInteger result = BigInteger.ONE;

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

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

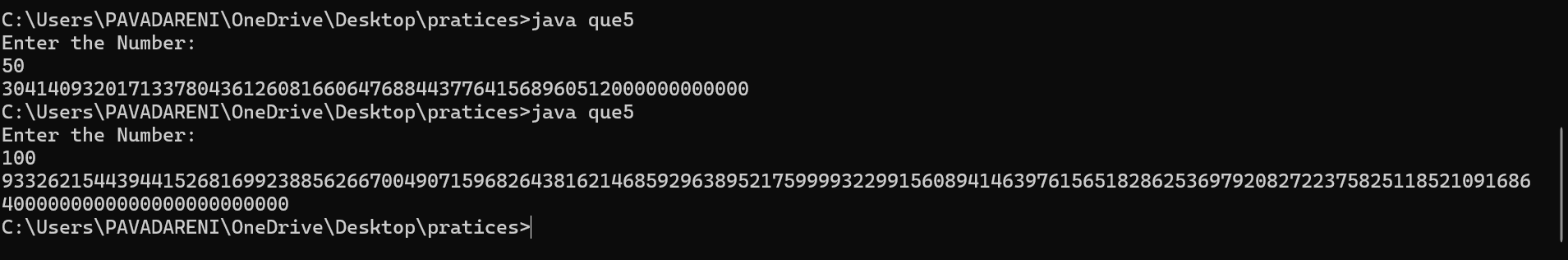
}

System.out.print(result);

}

}

**OUTPUT**

****

**TIME COMPLEXITY: O(n)**

The array is traversed only one time. Hence the complexity remains O(n).

1. Trapping Rainwater Problem

**CODE**

import java.util.Scanner;

import java.util.Stack;

class que6 {

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

System.out.print("Enter the array size:");

int n = s.nextInt();

int[] arr = new int[n];

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

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

arr[i] = s.nextInt();

}

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

int res = 0;

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

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

int pop\_height = arr[st.pop()];

if (st.isEmpty())

break;

int distance = i - st.peek() - 1;

int water = Math.min(arr[st.peek()], arr[i]);

water -= pop\_height;

res += distance \* water;

}

st.push(i);

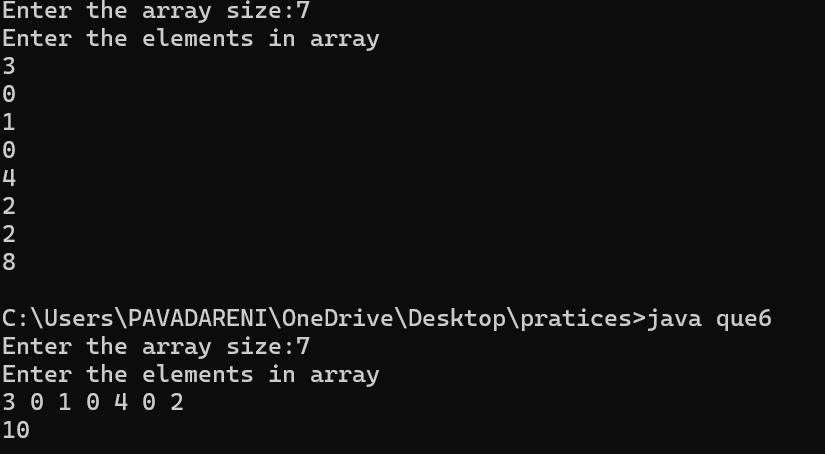
}

System.out.println(res);

}

}

**OUTPUT**

****

**TIME COMPLEXITY: O(n)**

The time complexity remains O(n) as the traversal is done once

1. Chocolate Distribution Problem

**CODE**

import java.util.\*;

class que7 {

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

System.out.print("Enter the rotated array size: ");

int n = s.nextInt();

int[] arr = new int[n];

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

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

arr[i] = s.nextInt();

}

System.out.print("Enter the m students: ");

int m = s.nextInt();

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];

if (diff < minDiff)

minDiff = diff;

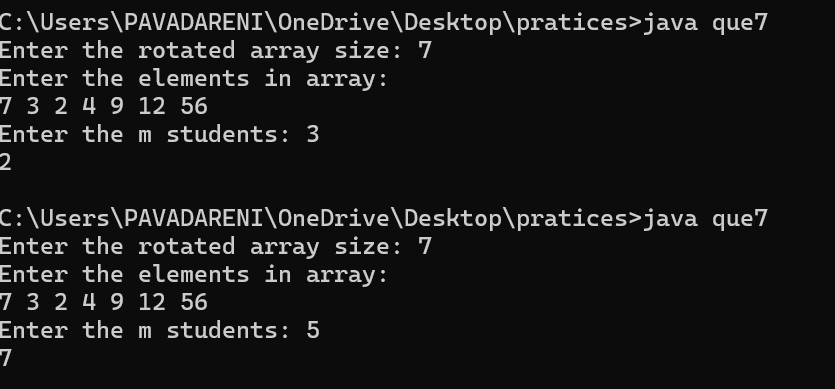
}

System.out.println(minDiff);

}

}

**OUTPUT**

****

**TIME COMPLEXITY: nlog(n)**

The code contain nlogn time complexity as the complexity of the sort () is nlog(n)

1. Merge Overlapping Intervals

**CODE**

import java.util.\*;

public class que8 {

static List<int[]> mergeOverlap(int[][] arr) {

int n = arr.length;

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

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

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

int start = arr[i][0];

int end = arr[i][1];

if (!res.isEmpty() && res.get(res.size() - 1)[1] >= start) {

res.get(res.size() - 1)[1] = Math.max(res.get(res.size() - 1)[1], end);

} else {

res.add(new int[] { start, end });

}

}

return res;

}

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

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

int n = s.nextInt();

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

System.out.println("Enter the intervals (start and end): ");

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

arr[i][0] = s.nextInt();

arr[i][1] = s.nextInt();

}

List<int[]> res = mergeOverlap(arr);

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

for (int[] interval : res) {

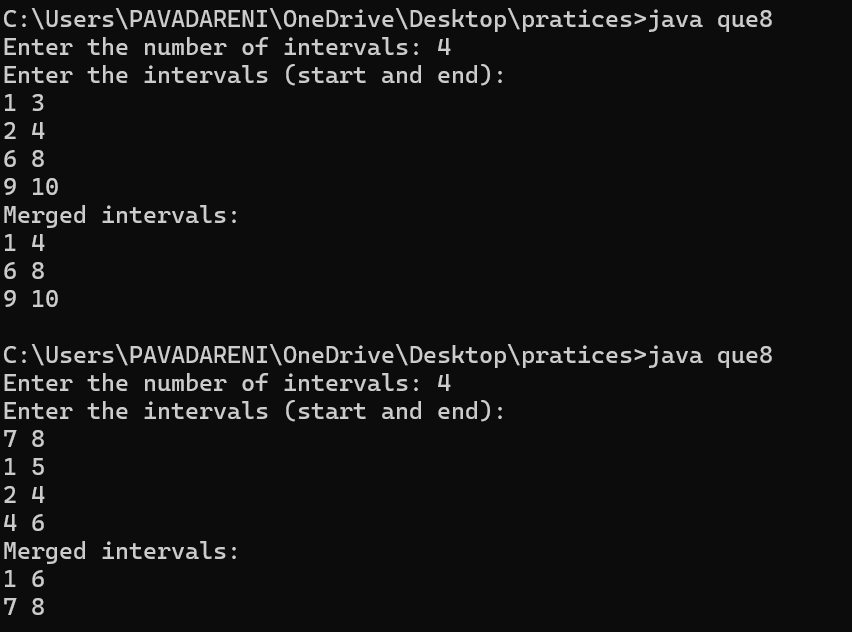
System.out.println(interval[0] + " " + interval[1]);

}

}

}

**OUTPUT**

****

**TIME COMPLEXITY**

**O(n log(n))**

1. A Boolean Matrix Question

**CODE**

class que9 {

    static void setZeroes(int[][] matrix) {

        int rows = matrix.length;

        int cols = matrix[0].length;

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

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

                if (matrix[i][j] == 1) {

                    int ind = i - 1;

                    while (ind >= 0) {

                        if (matrix[ind][j] != 1) {

                            matrix[ind][j] = -1;

                        }

                        ind--;

                    }

                    ind = i + 1;

                    while (ind < rows) {

                        if (matrix[ind][j] != 1) {

                            matrix[ind][j] = -1;

                        }

                        ind++;

                    }

                    ind = j - 1;

                    while (ind >= 0) {

                        if (matrix[i][ind] != 1) {

                            matrix[i][ind] = -1;

                        }

                        ind--;

                    }

                    ind = j + 1;

                    while (ind < cols) {

                        if (matrix[i][ind] != 1) {

                            matrix[i][ind] = -1;

                        }

                        ind++;

                    }

                }

            }

        }

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

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

                if (matrix[i][j] < 0) {

                    matrix[i][j] = 1;

                }

            }

        }

    }

    public static void main(String[] args) {

        int[][] arr = { { 1, 0, 0, 1 },

                { 0, 0, 1, 0 },

                { 0, 0, 0, 0 } };

        setZeroes(arr);

        System.out.println("The Final Matrix is:");

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

            for (int j = 0; j < arr[0].length; j++) {

                System.out.print(arr[i][j] + " ");

            }

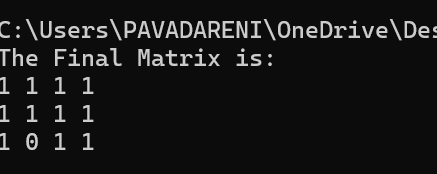
            System.out.println();

        }

    }

}

**OUTPUT**

****

**TIME COMPLEXITY : O(n+m)^2**

O((N\*M)\*(N + M)). O(N\*M) for traversing through each element and (N+M)for traversing to row and column of elements having value 1.

1. Print a given matrix in spiral form

**CODE**

import java.util.\*;

public class que10 {

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

        int m = matrix.length;

        int n = matrix[0].length;

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

        if (m == 0)

            return result;

        boolean[][] seen = new boolean[m][n];

        int[] dr = { 0, 1, 0, -1 };

        int[] dc = { 1, 0, -1, 0 };

        int r = 0, c = 0;

        int di = 0;

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

            result.add(matrix[r][c]);

            seen[r][c] = true;

            int newR = r + dr[di];

            int newC = c + dc[di];

            if (0 <= newR && newR < m && 0 <= newC && newC < n

                    && !seen[newR][newC]) {

                r = newR;

                c = newC;

            } else {

                di = (di + 1) % 4;

                r += dr[di];

                c += dc[di];

            }

        }

        return result;

    }

    public static void main(String[] args) {

        int[][] matrix = {

                { 1, 2, 3, 4 },

                { 5, 6, 7, 8 },

                { 9, 10, 11, 12 },

                { 13, 14, 15, 16 }

        };

        List<Integer> result = spiralOrder(matrix);

        for (int num : result) {

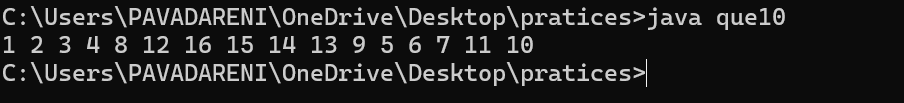
            System.out.print(num + " ");

        }

    }

}

**OUTPUT**



**TIME COMPLIXITY O(m\*n)**

O(m\*n), where m and n are the number of rows and columns of the given matrix respectively.

13) Check if given Parentheses expression is balanced or not

**CODE**

class GFG {

    public static boolean isBalanced(String exp) {

        boolean flag = true;

        int count = 0;

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

            if (exp.charAt(i) == '(') {

                count++;

            } else {

                count--;

            }

            if (count < 0) {

                flag = false;

                break;

            }

        }

        if (count != 0) {

            flag = false;

        }

        return flag;

    }

    public static void main(String[] args) {

        String exp1 = "((()))()()";

        if (isBalanced(exp1))

            System.out.println("Balanced");

        else

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

        String exp2 = "())((())";

        if (isBalanced(exp2))

            System.out.println("Balanced");

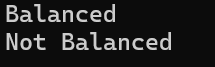
        else

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

    }

}

**OUTPUT**

****

**Time complexity: O(N)**

14. Check if two Strings are Anagrams of each other

**CODE**

import java.util.Arrays;

class que14 {

    public static void main(String[] args) {

        String s1 = "geeks";

        String s2 = "kseeg";

        char[] s1Array = s1.toCharArray();

        char[] s2Array = s2.toCharArray();

        Arrays.sort(s1Array);

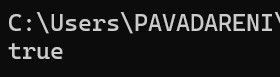
        Arrays.sort(s2Array);

        System.out.println(Arrays.equals(s1Array, s2Array));

    }

}

**OUTPUT**

****

**TIME COMPLEXITY O(nlogn)**

The sort function is nlogn of time complexity

15) Longest Palindromic Substring

**CODE**

public class java15 {

    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) {

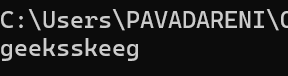
        String s = "forgeeksskeegfor";

        System.out.println(longestPalSubstr(s));

    }

}

**OUTPUT**

****

**TME COMPLEXITY O(n^3)**

As the string is traversed 3 time. The time complexity remains n^3

16. Longest Common Prefix using Sorting

**CODE**

import java.util.Arrays;

class quee16 {

    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 minLength = Math.min(first.length(), last.length());

        int i = 0;

        while (i < minLength

                && first.charAt(i) == last.charAt(i)) {

            i++;

        }

        if (i == 0)

            return "-1";

        return first.substring(0, i);

    }

    public static void main(String[] args) {

        String[] arr = { "geeksforgeeks", "geeks", "geek",

                "geezer" };

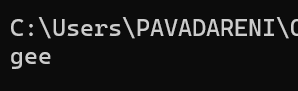
        System.out.println("The longest common prefix is: "

                + longestCommonPrefix(arr));

    }

}

**OUTPUT**

****

**TIME COMPLEXITY O(n)**

17) Delete middle element of a stack

**CODE**

import java.util.\*;

public class que17 {

    static void deleteMid(Stack<Character> st,

            int n, int curr) {

        if (st.empty() || curr == n)

            return;

        char x = st.pop();

        deleteMid(st, n, curr + 1);

        if (curr != n / 2)

            st.push(x);

    }

    public static void main(String args[]) {

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

        st.push('1');

        st.push('2');

        st.push('3');

        st.push('4');

        st.push('5');

        st.push('6');

        st.push('7');

        deleteMid(st, st.size(), 0);

        while (!st.empty()) {

            char p = st.pop();

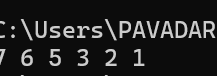
            System.out.print(p + " ");

        }

    }

}

**Output**



**Time complexity O(n)**

O(n) For the while loop

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

**CODE**

class que18 {

    static void printNGE(int arr[], int n) {

        int next, i, j;

        for (i = 0; i < n; i++) {

            next = -1;

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

                if (arr[i] < arr[j]) {

                    next = arr[j];

                    break;

                }

            }

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

        }

    }

    public static void main(String args[]) {

        int arr[] = { 11, 13, 21, 3 };

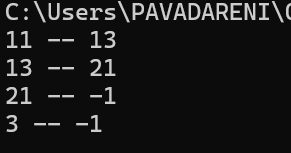
        int n = arr.length;

        printNGE(arr, n);

    }

}

**OUTPUT**

****

**TIME COMPLEXITY: O(n)**

19. Print Right View of a Binary Tree

**CODE**

// Java program to print right view of binary tree

// using Recursion

import java.util.ArrayList;

class Node {

    int data;

    Node left, right;

    Node(int x) {

        data = x;

        left = right = null;

    }

}

class que19 {

    static void RecursiveRightView(Node root, int level,

            int[] maxLevel, ArrayList<Integer> result) {

        if (root == null)

            return;

        if (level > maxLevel[0]) {

            result.add(root.data);

            maxLevel[0] = level;

        }

        RecursiveRightView(root.right, level + 1,

                maxLevel, result);

        RecursiveRightView(root.left, level + 1,

                maxLevel, result);

    }

    static ArrayList<Integer> rightView(Node root) {

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

        int[] maxLevel = new int[] { -1 };

        RecursiveRightView(root, 0, maxLevel, result);

        return result;

    }

    static void printArray(ArrayList<Integer> arr) {

        for (int val : arr) {

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

        }

        System.out.println();

    }

    public static void main(String[] args) {

        Node root = new Node(1);

        root.left = new Node(2);

        root.right = new Node(3);

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

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

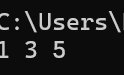
        ArrayList<Integer> result = rightView(root);

        printArray(result);

    }

}

**OUTPUT**

****

**TIME COMPLEXITY O(n)**

O(n), We traverse all nodes of the binary tree exactly once, where n is the number of nodes

20. Maximum Depth or Height of Binary Tree

**CODE**

import java.util.LinkedList;

import java.util.Queue;

class Node {

    int key;

    Node left, right;

    Node(int val) {

        key = val;

        left = null;

        right = null;

    }

}

class java20 {

    static int height(Node root) {

        if (root == null)

            return 0;

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

        q.add(root);

        int h = 0;

        while (!q.isEmpty()) {

            int size = q.size();

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

                Node temp = q.poll();

                if (temp.left != null)

                    q.add(temp.left);

                if (temp.right != null)

                    q.add(temp.right);

            }

            h++;

        }

        return h;

    }

    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);

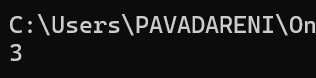
        root.left.right = new Node(5);

        System.out.println(height(root));

    }

}

**OUTPUT**

****

**TIME COMPLEXITY O(n)**

O(n), where n is the number of nodes in the tree.