**1.maximumsum**

**Code:**

import java.util.Scanner;

import java.util.Arrays;

class maximumsum {

static int[] maxSubarraySum(int[] arr) {

int maxSum = arr[0];

int maxEnding = arr[0];

int start = 0;

int end = 0;

int tempStart = 0;

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

if (arr[i] > maxEnding + arr[i]) {

maxEnding = arr[i];

tempStart = i;

} else {

maxEnding += arr[i];

}

if (maxEnding > maxSum) {

maxSum = maxEnding;

start = tempStart;

end = i;

}

}

int[] maxSubarray = Arrays.copyOfRange(arr, start, end + 1);

System.out.println("The maximum subarray is: " + Arrays.toString(maxSubarray));

return new int[] { maxSum };

}

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();

}

System.out.println("The input array is: " + Arrays.toString(arr));

int[] result = maxSubarraySum(arr);

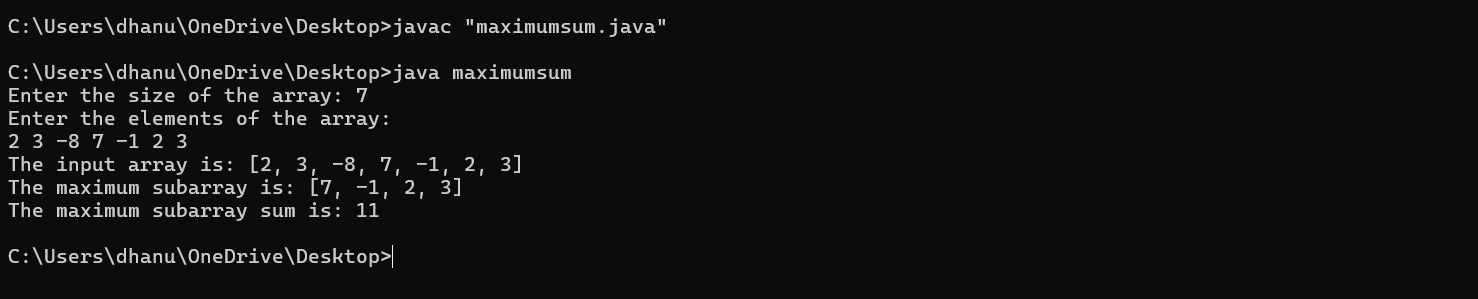
System.out.println("The maximum subarray sum is: " + result[0]);

scanner.close();

}

}

**Output:**



**Timecomplexity:O(n)**

**2. Maximum Product Subarray**

**CODE:**

import java.util.\*;

class Solution {

public int maxProduct(int[] nums) {

int max=nums[0];

int min=nums[0];

int res=nums[0];

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

int c=nums[i];

if(c<0){

int temp=max;

max=min;

min=temp;

}

max=Math.max(c,c\*max);

min=Math.min(c,c\*min);

res=Math.max(res,max);

}

return res;

}

public static void main(String[] args){

Solution ob1=new Solution();

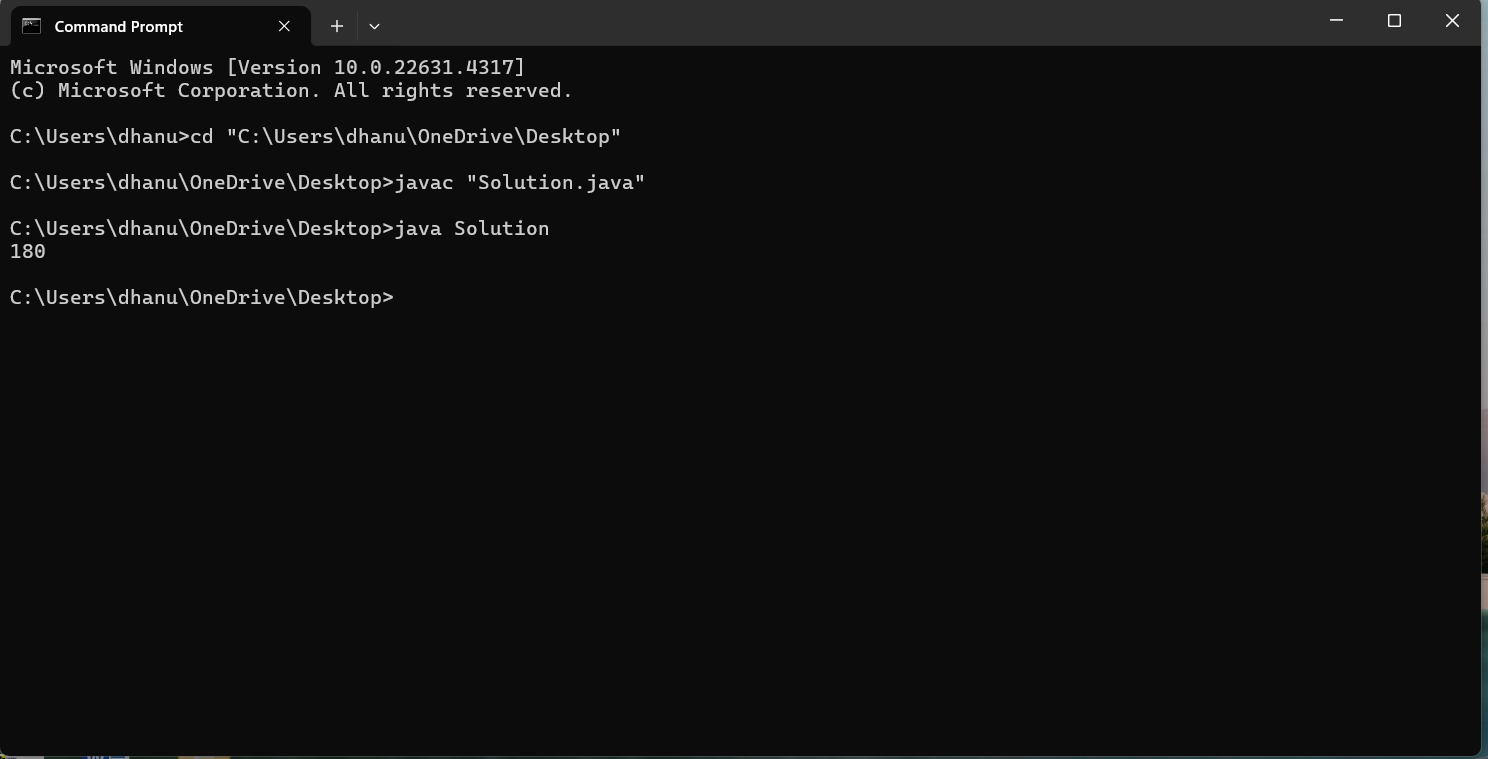
int [] nums = {-2, 6, -3, -10, 0, 2};

System.out.println(ob1.maxProduct(nums));

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(n).**

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

**CODE:**

class Array {

public int search(int[] nums, int target) {

int left=0;

int right=nums.length-1;

while(left<=right){

int mid=left+((right-left)/2);

if(target==nums[mid]){

return mid;

}

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

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

right=mid-1;

}

else{

left=mid+1;

}

}

else{

if(target>nums[mid]&&target<=nums[right]){

left=mid+1;

}

else{

right=mid-1;

}

}

}

return -1;

}

public static void main(String args[]){

Array obj=new Array();

int[] nums={50, 10, 20, 30, 40};

int target=10;

System.out.println(obj.search(nums,target));

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(log n).**

**4. Container with Most Water**

**CODE:**

class Solution {

public int MaxArea(int[] height) {

int left=0;

int right=height.length-1;

int max\_water=0;

while(left<right){

int width=right-left;

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

int area=width\*h;

max\_water=Math.max(max\_water,area);

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

left++;

}

else{

right--;

}

}

return max\_water;

}

public static void main(String args[]){

Solution obj=new Solution();

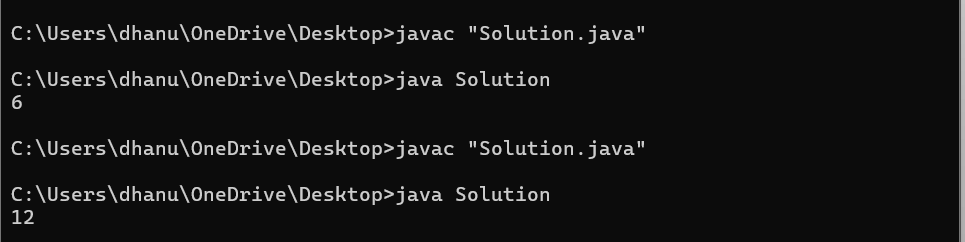
int[] height={3, 1, 2, 4, 5};

System.out.println(obj.MaxArea(height));

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(n).**

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

**CODE:**

import java.util.\*;

import java.math.BigInteger;

class Solution{

public 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[]){

Solution fact=new Solution();

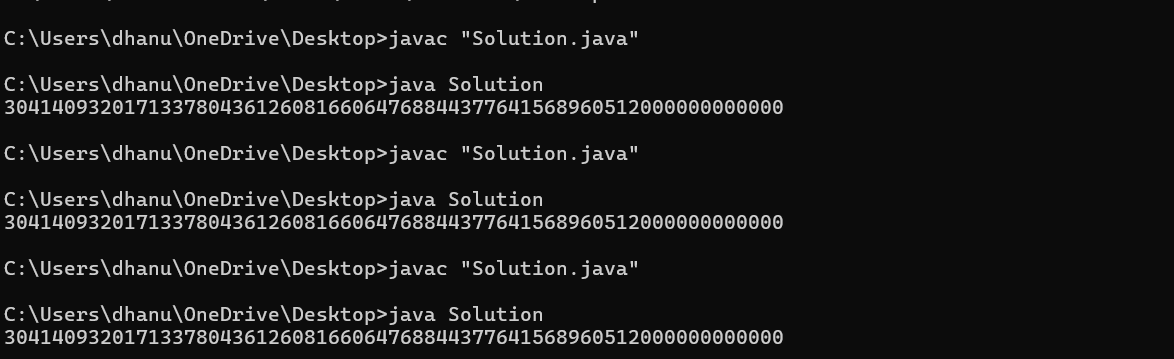
int n=50;

System.out.println(fact.factorial(n));

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(n).**

**6.** **Trapping Rainwater Problem**

**CODE:**

import java.util.\*;

class Rain{

public int TrappingRainwater(int arr[]){

int left=0;

int right=arr.length-1;

int max\_left=arr[left];

int max\_right=arr[right];

int water=0;

while(left<right){

if(max\_left<max\_right){

left++;

max\_left=Math.max(max\_left,arr[left]);

water+=max\_left-arr[left];

}

else{

right--;

max\_right=Math.max(max\_right,arr[right]);

water+=max\_right-arr[right];

}

}

return water;

}

public static void main(String args[]){

Rain trap=new Rain();

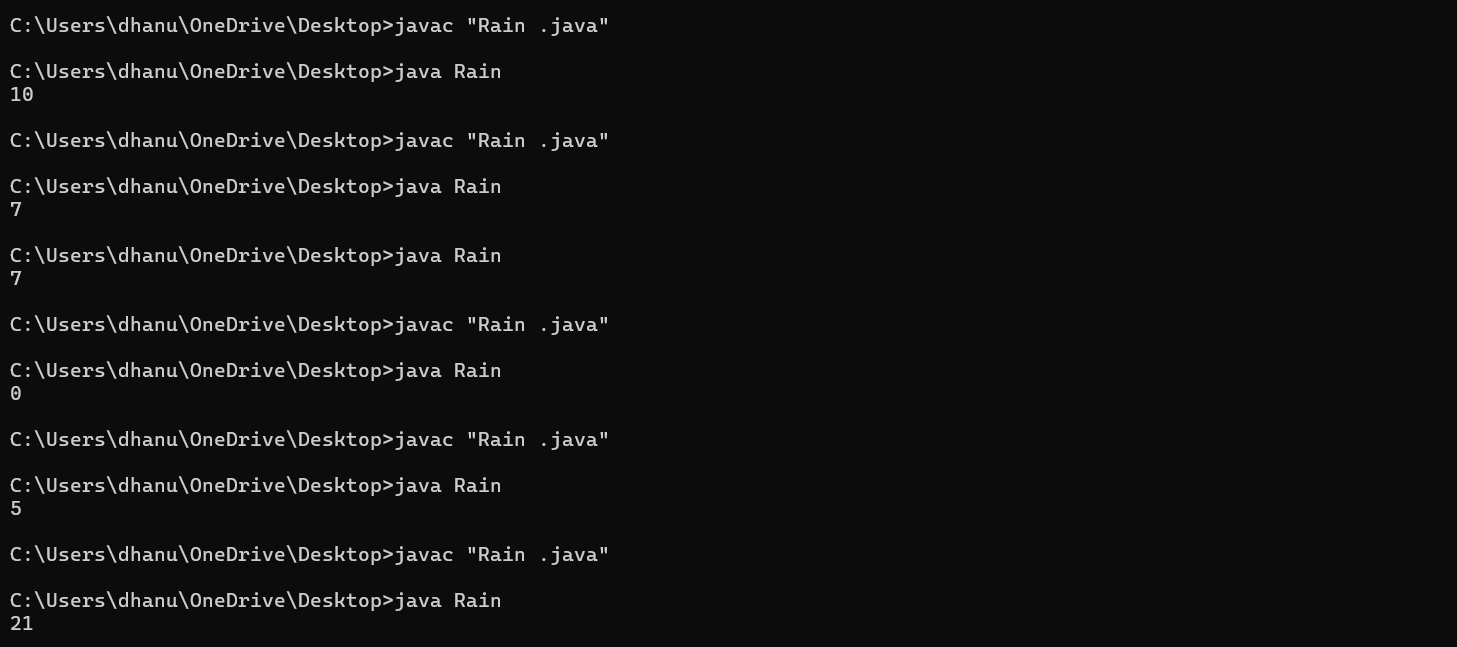
int[] arr={11,23,10,15,30,43};

System.out.println(trap.TrappingRainwater(arr));

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(n).**

**7.** **Chocolate Distribution Problem**

import java.util.\*;

class Chocolate{

public int Distribution(int[] arr, int m){

int n=arr.length;

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

return 0;

}

Arrays.sort(arr);

if(n<m){

return -1;

}

int min=Integer.MAX\_VALUE;

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

int diff=arr[i+m-1]-arr[i];

min=Math.min(min,diff);

}

return min;

}

public static void main(String args[]){

Chocolate choco=new Chocolate();

int[] arr={24,34,28,19,43,23,41,47};

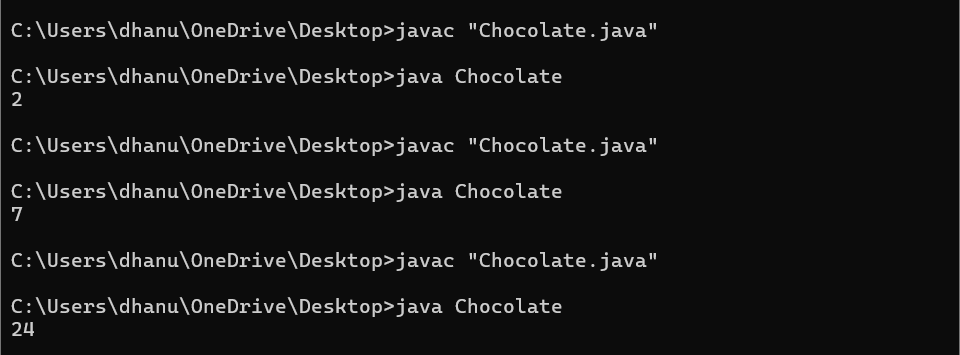
int m=7;

System.out.println(choco.Distribution(arr,m));

}

}

**OUTPUT:**

****

**TIME COMPLEXITY:** **O(n log n).**

**8.mergeovelapping**

import java.util.\*;

class Merge {

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

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

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

int[] prev = intervals[0];

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

int[] interval = intervals[i];

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

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

} else {

merged.add(prev);

prev = interval;

}

}

merged.add(prev);

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];

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

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

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

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

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

}

Merge mergeObj = new Merge();

int[][] mergedIntervals = mergeObj.merge(intervals);

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

for (int[] interval : mergedIntervals) {

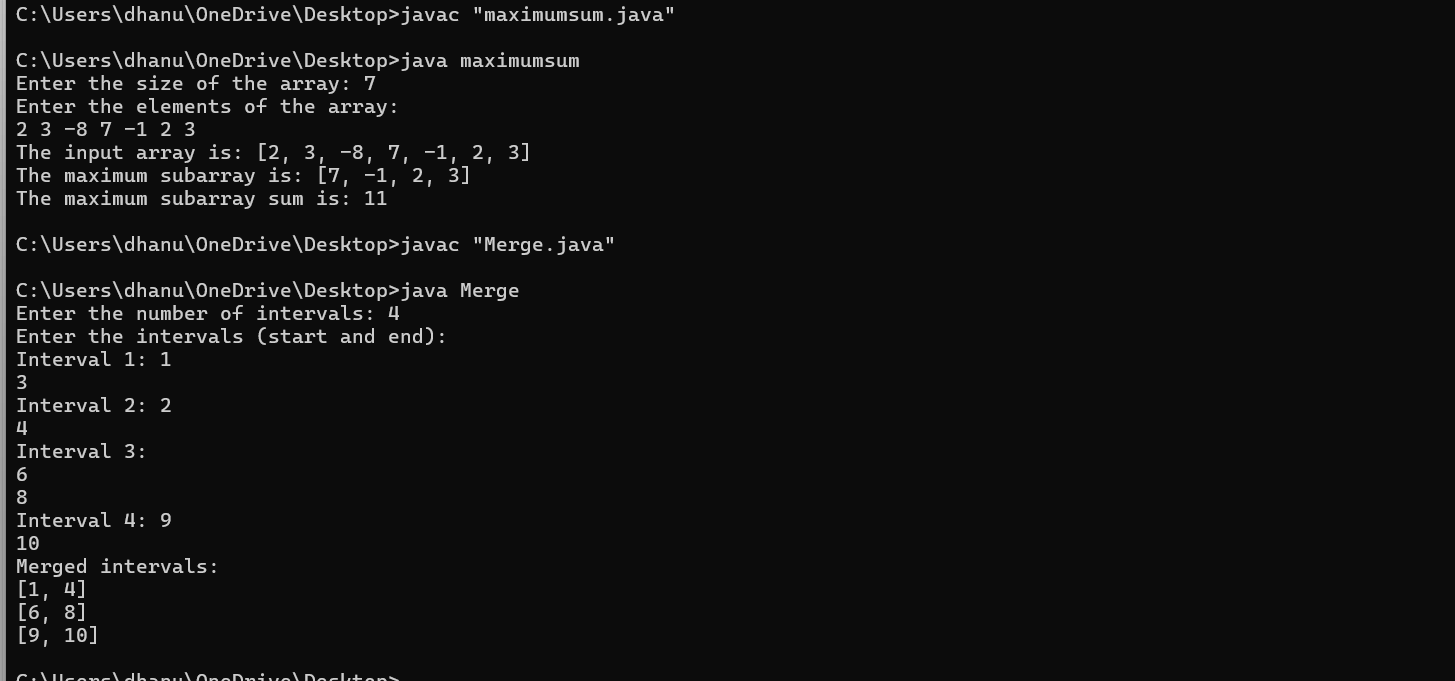
System.out.println(Arrays.toString(interval));

scanner.close();

}

}

}



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

**9.booleanmatrix**

import java.util.Scanner;

class BooleanMatrix {

public static void modifyMatrix(int mat[][], int R, int C) {

int row[] = new int[R];

int col[] = new int[C];

int i, j;

for (i = 0; i < R; i++)

row[i] = 0;

for (i = 0; i < C; i++)

col[i] = 0;

for (i = 0; i < R; i++) {

for (j = 0; j < C; j++) {

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

row[i] = 1;

col[j] = 1;

}

}

}

for (i = 0; i < R; i++) {

for (j = 0; j < C; j++) {

if (row[i] == 1 || col[j] == 1)

mat[i][j] = 1;

}

}

}

public static void printMatrix(int mat[][], int R, int C) {

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

for (int j = 0; j < C; j++)

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

System.out.println();

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int R = scanner.nextInt();

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

int C = scanner.nextInt();

int[][] mat = new int[R][C];

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

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

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

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

}

}

System.out.println("Matrix Initially:");

printMatrix(mat, R, C);

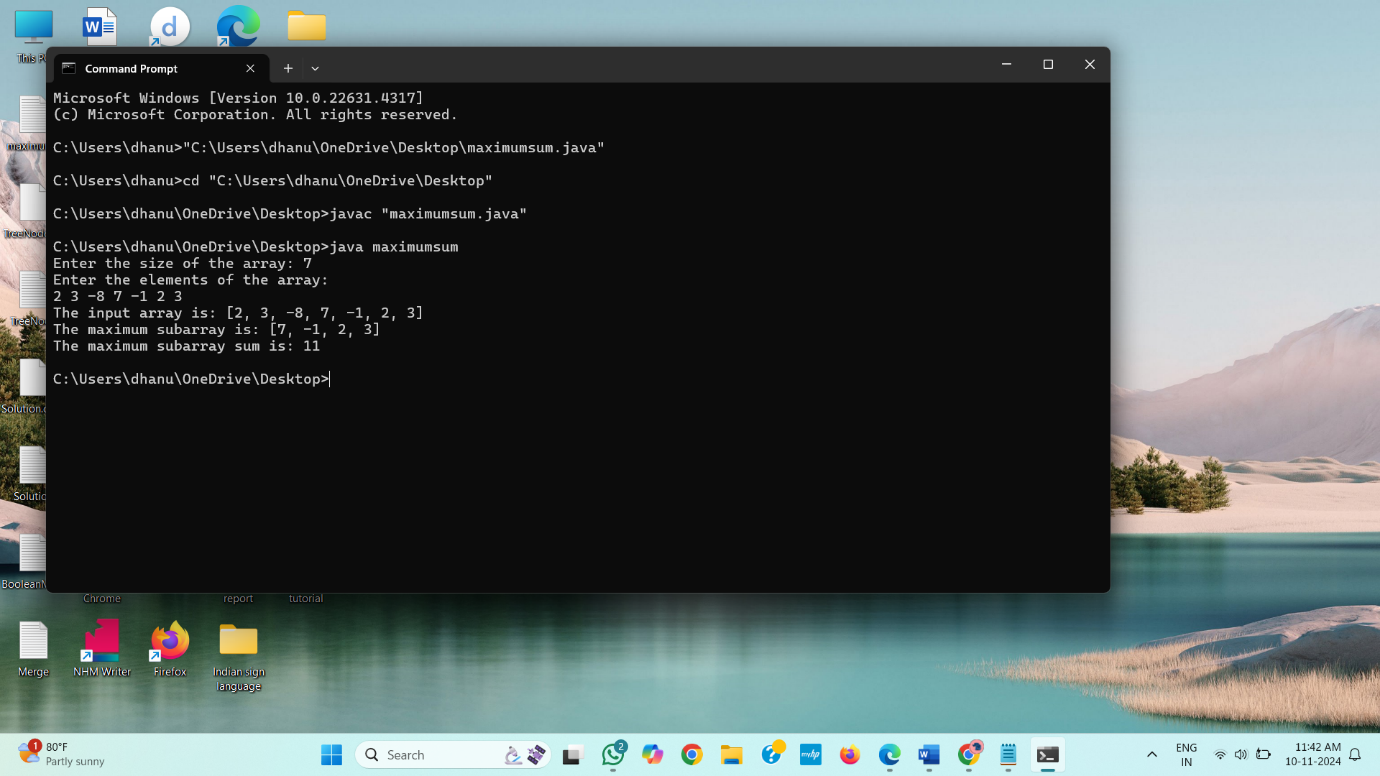
modifyMatrix(mat, R, C);

System.out.println("Matrix after modification:");

printMatrix(mat, R, C);

}

}

****

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

**CODE:**

import java.util.\*;

class Matrix {

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

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

int row=0;

int rowend=matrix.length-1;

int col=0;

int colend=matrix[0].length-1;

while(row<=rowend&&col<=colend){

for(int i=col;i<=colend;i++){

result.add(matrix[row][i]);

}

row++;

for(int i=row;i<=rowend;i++){

result.add(matrix[i][colend]);

}

colend--;

if(row<=rowend){

for(int i=colend;i>=col;i--){

result.add(matrix[rowend][i]);

}

rowend--;

}

if(col<=colend){

for(int i=rowend;i>=row;i--){

result.add(matrix[i][col]);

}

col++;

}

}

return result;

}

public static void main(String args[]){

Matrix spiral=new Matrix();

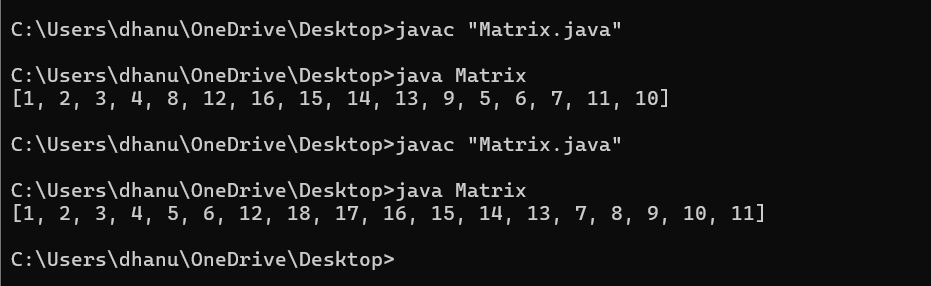
int[][] matrix={{1, 2, 3, 4, 5, 6},{7, 8, 9, 10, 11, 12},{13, 14, 15, 16, 17, 18}};

System.out.println(spiral.SpiralOrder(matrix));

}

}

**OUTPUT:**

****

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

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

**CODE:**import java.util.Stack;

class Paranthesis{

public Boolean Balanced(String str){

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

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

char ch=str.charAt(i);

if(ch=='('){

stack.push(ch);

}

else if(ch==')'){

if(stack.isEmpty()){

return false;

}

stack.pop();

}

}

return stack.isEmpty();

}

public static void main(String args[])

{

Paranthesis para=new Paranthesis();

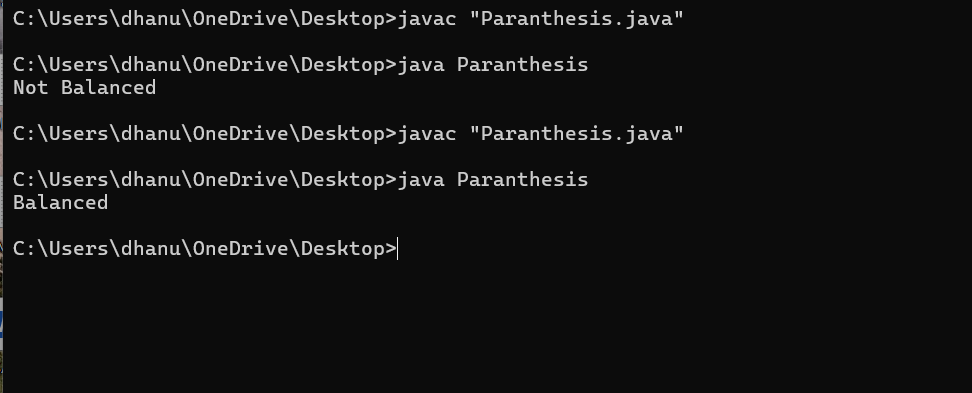
String str="((()))()()";

System.out.println(para.Balanced(str)?"Balanced":"Not Balanced");

}

}

**Output:**

****

**Time Complexity: O(n)**

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

**CODE:**

import java.util.\*;

class Solution{

public Boolean Anagram(String s1, String s2){

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

return false;

}

char[] charArray1=s1.toCharArray();

char[] charArray2=s2.toCharArray();

Arrays.sort(charArray1);

Arrays.sort(charArray2);

return Arrays.equals(charArray1,charArray2);

}

public static void main(String args[]){

Solution arr=new Solution();

String s1="geeks";

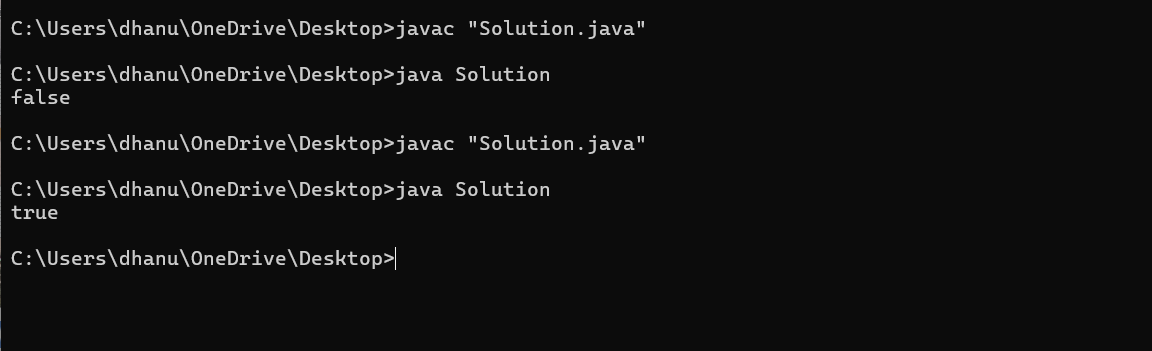
String s2="kseeg";

System.out.println(arr.Anagram(s1,s2));

}

}

**OUTPUT:**

****

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

**15.** **Longest Palindromic Substring**

**CODE:**

public class Solution {

public static String LongestPalindrome(String str) {

int n = str.length();

if (n < 2) return str;

boolean[][] dp = new boolean[n][n];

int start = 0, maxLength = 1;

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

dp[i][i] = true;

}

for (int length = 2; length <= n; length++) {

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

int j = i + length - 1;

if (str.charAt(i) == str.charAt(j)) {

if (length == 2 || dp[i + 1][j - 1]) {

dp[i][j] = true;

if (length > maxLength) {

start = i;

maxLength = length;

}

}

}

}

}

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

}

public static void main(String[] args) {

Solution st=new Solution();

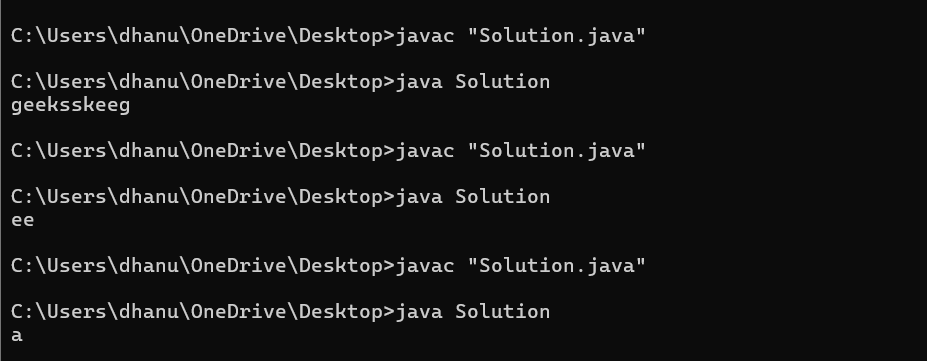
String str1 = "abc";

System.out.println(st.LongestPalindrome(str1));

}

}

**OUTPUT:**

****

**16.** **Longest Common Prefix using Sorting.**

**CODE:**

import java.util.Arrays;

class Solution{

public static String CommonPrefix(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++;

}

String prefix = first.substring(0, i);

return prefix.isEmpty() ? "-1" : prefix;

}

public static void main(String[] args) {

Solution cp=new Solution();

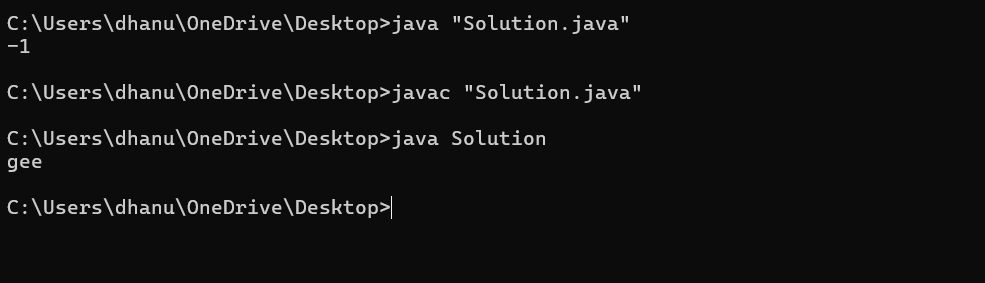
String[] arr1 = {"geeksforgeeks", "geeks", "geek", "geezer"} ;

System.out.println(cp.CommonPrefix(arr1));

}

}

**OUTPUT:**



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

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

**CODE:**

import java.util.Stack;

public class Nge {

public int[] NextGreater(int[] arr) {

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

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

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

arr[stack.pop()] = arr[i];

}

stack.push(i);

}

while (!stack.isEmpty()) {

arr[stack.pop()] = -1;

}

return arr;

}

public static void main(String[] args) {

Nge ob = new Nge();

int[] arr1 = {13 , 7, 6 , 12};

int[] result = ob.NextGreater(arr1);

for (int num : result) {

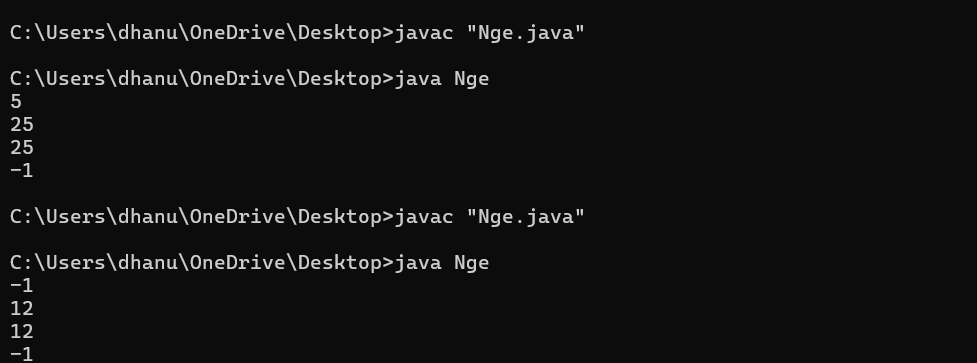
System.out.println(num);

}

}

}

**OUTPUT:**

****

**Time Complexity: O(n)**

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

**CODE:**

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

public TreeNode(int val) {

this.val = val;

left = right = null;

}

}

public class Node {

public List<Integer> rightSideView(TreeNode root) {

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

rightView(root, result, 0);

return result;

}

public void rightView(TreeNode curr, List<Integer> result, int currDepth) {

if (curr == null) {

return;

}

if (currDepth == result.size()) {

result.add(curr.val);

}

rightView(curr.right, result, currDepth + 1);

rightView(curr.left, result, currDepth + 1);

}

public static void main(String[] args) {

Node ob = new Node();

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

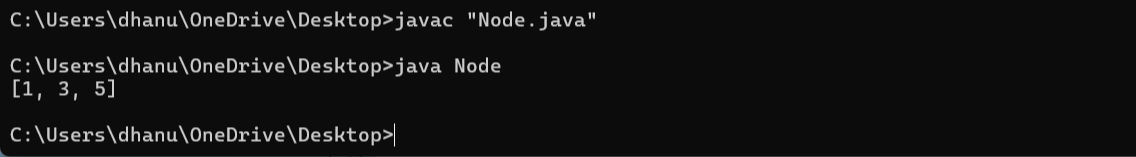
root.left.right = new TreeNode(5);

System.out.println(ob.rightSideView(root));

}

}

**OUTPUT:** ****



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

****