Problems Output:

1. Kadane’s Algorithm-Maximum Subarray: (Time Complexity=O(n), space Complexity=O(n))

import java.util.\*;

public class Kadane{

public static void main(String[] args){

int arr[];

Scanner s=new Scanner(System.in);

int n = s.nextInt();

arr=new int[n];

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

arr[i]=s.nextInt();

}

int c=arr[0];

int max=arr[0];

for(int i:arr){

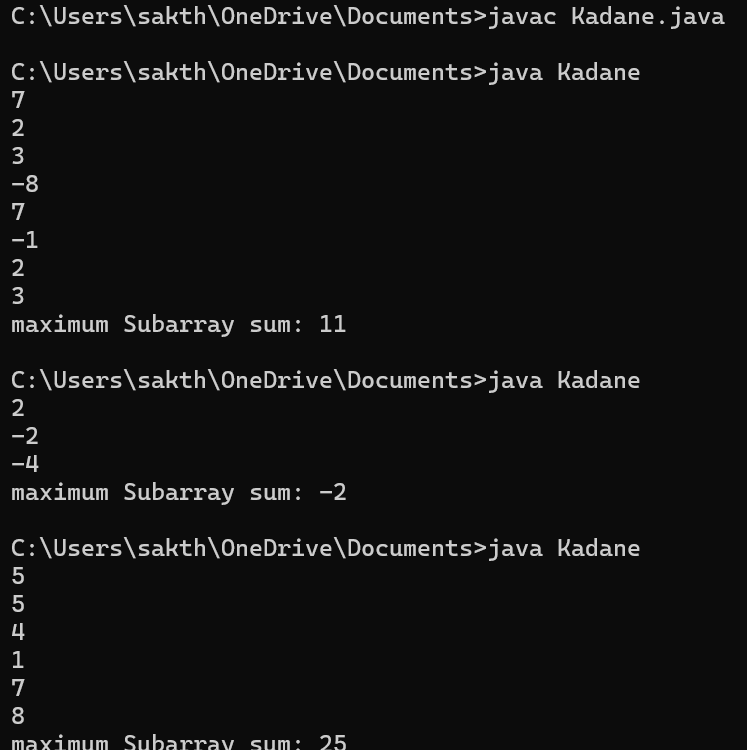
c=Math.max(arr[i],arr[i]+c);

m=Math.max(c,max);

}

System.out.println("maximum Subarray sum: "+ max);}}

Output:



2.maximum product subarray (time complexity:O(n); space complexity:O(n))

import java.util.\*;

public class Productsubarray{

public static void main(String[] args){

int arr[];

Scanner s=new Scanner(System.in);

int n = s.nextInt();

arr=new int[n];

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

arr[i]=s.nextInt();

}

int c=arr[0];

int max=arr[0];

int min=arr[0];

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

if(arr[i]<0){

int t=max;

max=min;

min=t;

}

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

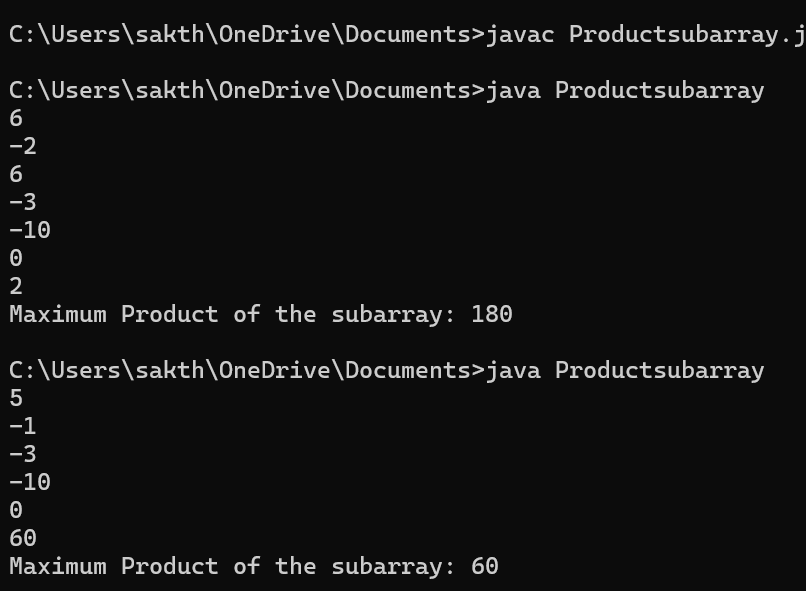
min=Math.min(arr[i],min\*arr[i]);

c=Math.max(c,max);

}

System.out.println("Maximum Product of the subarray: "+c);}}

Output:



3.Search in a Sorted array (Time Complexity: O(n), Space Complexity: O(n))

import java.util.\*;

public class Search{

public static void main(String[] args){

int arr[];

Scanner s=new Scanner(System.in);

int n = s.nextInt();

arr=new int[n];

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

arr[i]=s.nextInt();

}

int k=s.nextInt();

int res=-1;

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

if(arr[i]==k){

res=i;

break;

}

}

if(res!=-1){

System.out.println("The key is found at index: "+ res);

}

else{

System.out.println("The key is found at index: "+ -1);

}

s.close();

}

}

Output:



5.Factorial of a large number: (Time Complexity: O(N^3), space complexity:O(1))

import java.util.\*;

import java.math.BigInteger;

public class Factorial{

public BigInteger fact(int n)

{

if(n<2)

{

return BigInteger.ONE;

}

else

{

return BigInteger.valueOf(n).multiply(fact(n-1));

}

}

public static void main(String[] args)

{

Scanner s=new Scanner(System.in);

Factorial f = new Factorial();

int t=s.nextInt();

BigInteger res=f.fact(t);

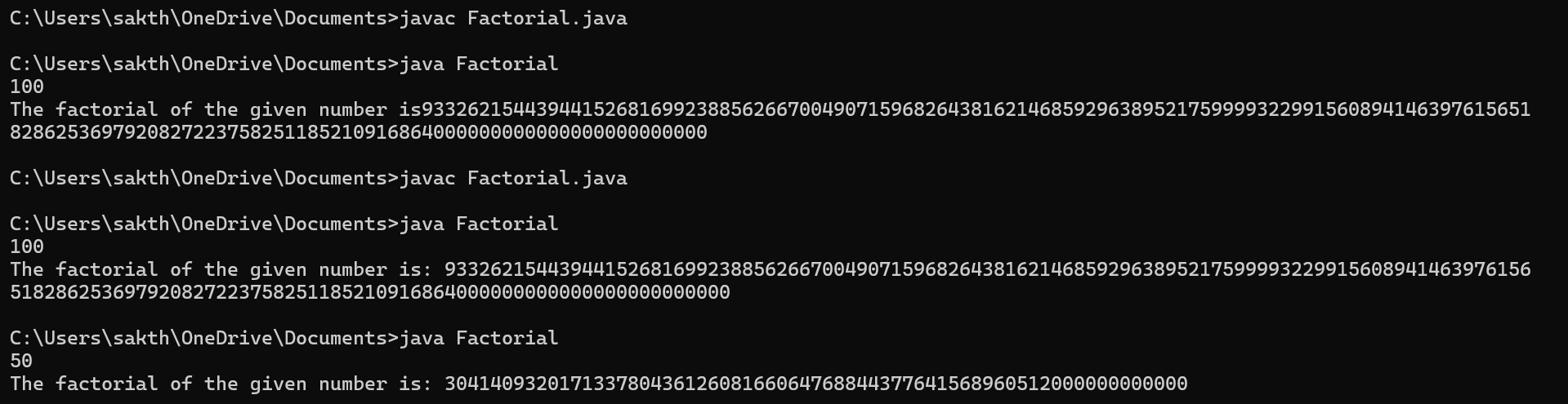
System.out.println("The factorial of the given number is: "+ res);

s.close();

}

}

Output:



14. Anagram: time Complexity:O(n)

import java.util.\*;

public class Anagram{

public String Sort(String a){

char arr[]=a.toCharArray();

Arrays.sort(arr);

return new String(arr);

}

public static void main(String[] args)

{

Scanner s=new Scanner(System.in);

String s1=s.nextLine();

String s2=s.nextLine();

Anagram a=new Anagram();

if(a.Sort(s1).equals(a.Sort(s2)))

{

System.out.println("true");

}

else

{

System.out.println("false");

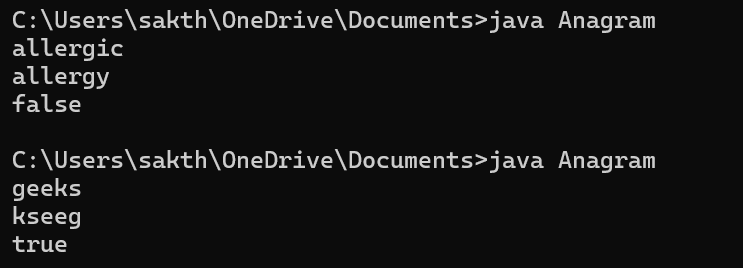
}

s.close();

}

}

Output:



13. Check whether Parenthesis is Balanced or not: time Complexity:O(n)

import java.util.\*;

public class Parenthesis

{

public boolean Valid(String s){

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

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

if(s.charAt(i)=='(')

{

st.push('(');

}

else

{

if(!st.isEmpty()){

st.pop();

}

else {

return false;

}

}

}

return st.isEmpty();

}

public static void main(String[] args)

{

Scanner s=new Scanner(System.in);

String str=s.nextLine();

Parenthesis p=new Parenthesis();

if(p.Valid(str))

{

System.out.println("balanced");

}

else{

System.out.println("not balanced");

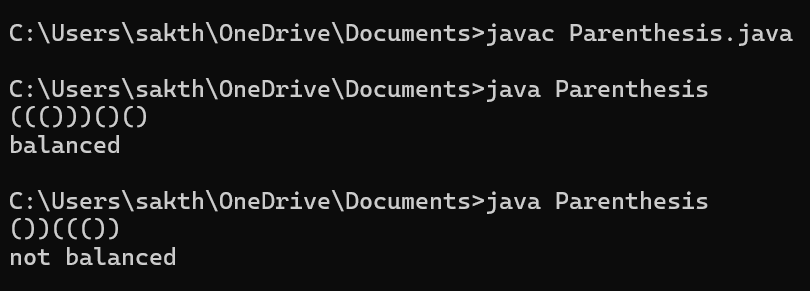
};

s.close();

}

}

Output:



15.Longest Comman Prefix: time Complexity: O(n)

import java.util.\*;

class HelloWorld {

public static void main(String[] args) {

Scanner s=new Scanner(System.in);

String res="";

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

int n=s.nextInt();

s.nextLine();

String arr[]=new String[n];

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

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

arr[i]=s.nextLine();

}

Arrays.sort(arr);

for(int i=0;i<Math.min(arr[0].length(),arr[arr.length-1].length());i++){

if(arr[0].charAt(i)!=(arr[arr.length-1].charAt(i))){

break;

}

res+=arr[0].charAt(i);

}

if(res.length()>0){

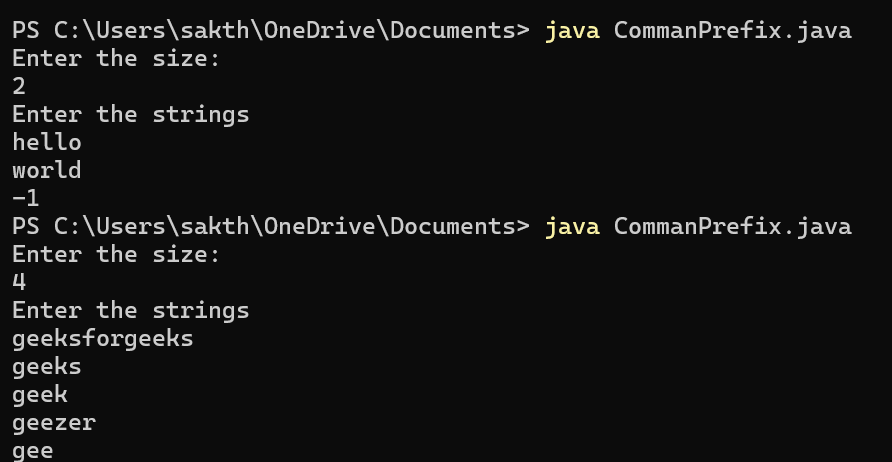
System.out.println(res);}

else System.out.println(-1);

}

}

Output:



4. Container with most water: time complexity: O(n)

import java.util.\*;

class MaxWater{

public static void main(String[] args) {

Scanner s=new Scanner(System.in);

int n = s.nextInt();

int height[]=new int[n];

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

height[i]=s.nextInt();

}

int l=0;

int r=n-1;

int m=0;

while(l<r){

int w=Math.min(height[l],height[r]);

int b=r-l;

int c=w\*b;

m=Math.max(m,c);

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

l++;

}

else r--;

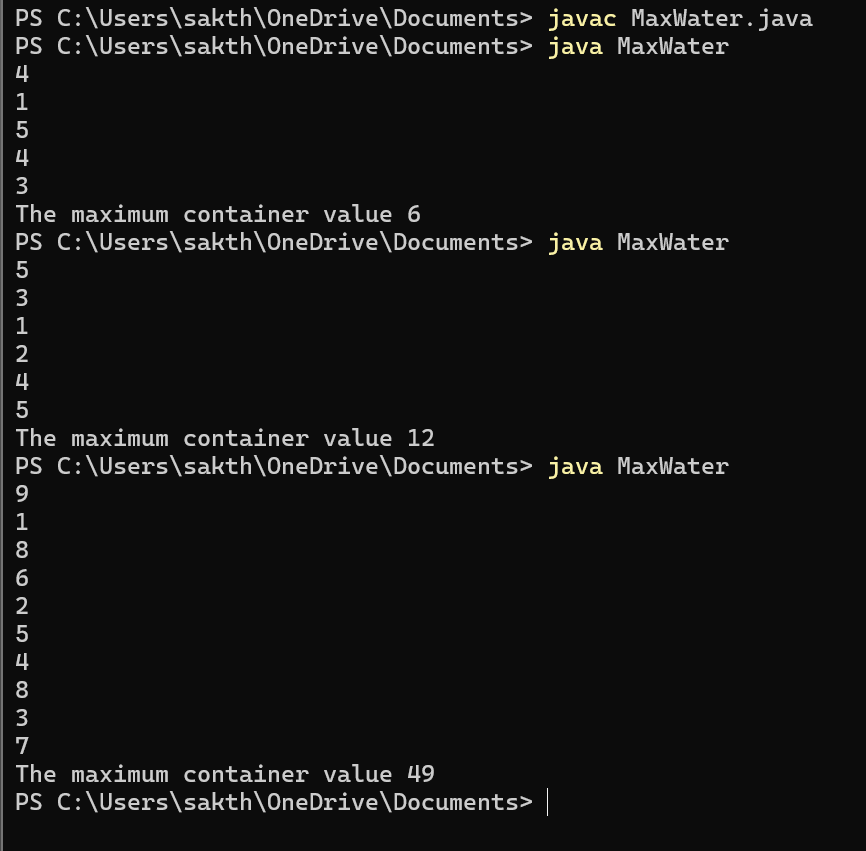
}

System.out.print("The maximum container value "+m);

}

}

Output:



7.Chocolate Distribution Problem: Time Complexity: O(n)

import java.util.\*;

public class Chocolate{

public static void main(String[] args) {

Scanner s=new Scanner(System.in);

System.out.println("enter the no of students: ");

int m=s.nextInt();

s.nextLine();

System.out.println("enter the no of chocolates in each packet");

int n=s.nextInt();

s.nextLine();

int a[]=new int[n];

System.out.println("enter the chocolates in each packet");

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

a[i]=s.nextInt();

}

Arrays.sort(a);

int i=0;

int j=m-1;

int min=Integer.MAX\_VALUE;

while(j<n){

min=Math.min(min,a[j]-a[i]);

i++;

j++;

}

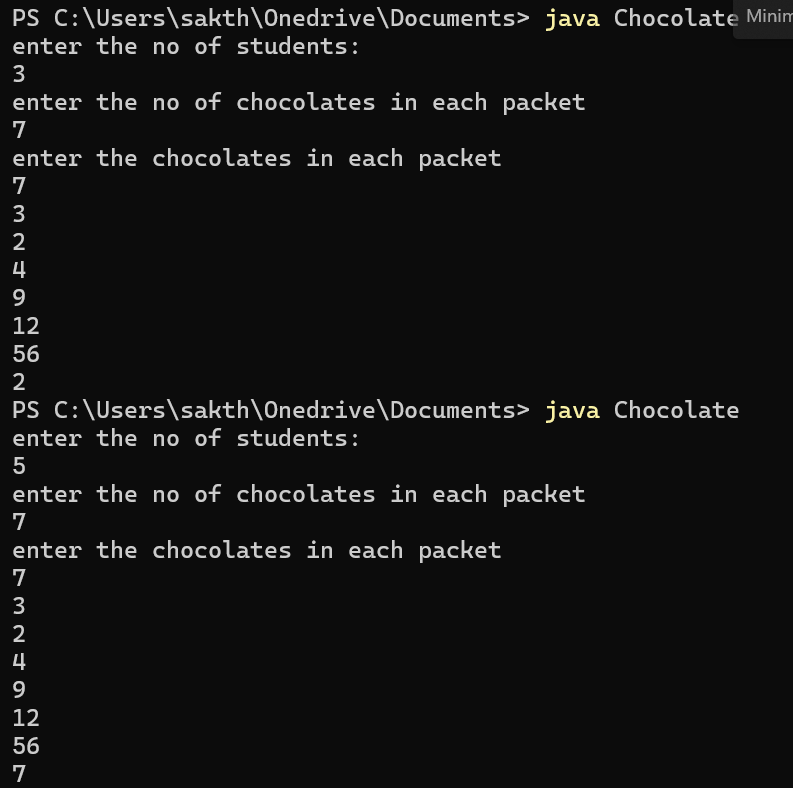
System.out.println(min);

s.close();

}

}

Output:



8.Merger Intervals: Time Complexity:O(n)

import java.util.\*;

public class IntervalMerger {

public static int[][] merger(int[][] arr) {

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

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

for (int[] i : arr) {

if (merged.isEmpty() || merged.get(merged.size() - 1)[1] < i[0]) {

merged.add(i);

} else {

merged.get(merged.size() - 1)[1] = Math.max(merged.get(merged.size() - 1)[1], i[1]);

}

}

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

}

private static void printIntervals(int[][] arr) {

for (int[] i : arr) {

System.out.print(Arrays.toString(i) + " ");

}

System.out.println();

}

public static void main(String[] args) {

int[][] arr1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};

int[][] arr2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};

int[][] res1 = merger(arr1);

int[][] res2 = merger(arr2);

System.out.println("arr1 Merge: ");

printIntervals(res1);

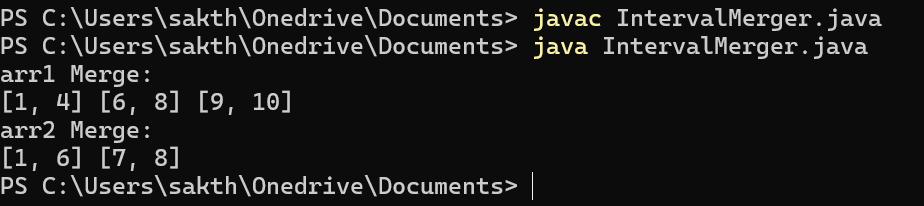
System.out.println("arr2 Merge: ");

printIntervals(res2);

}

}

Output:



9.Boolean Matrix: Time Complexity: O(n)

import java.util.\*;

public class boolMatrix{

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

int a = mat.length;

int b = mat[0].length;

boolean[] r = new boolean[a];

boolean[] c = new boolean[b];

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

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

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

r[i] = true;

c[j] = true;

}

}

}

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

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

if (r[i] || c[j]) {

mat[i][j] = 1;

}

}

}

}

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

for (int[] r : mat) {

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

}

}

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

System.out.print("enter no of rows ");

int r = s.nextInt();

System.out.print("enter no of cols ");

int c = s.nextInt();

int[][] mat = new int[r][c];

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

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

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

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

}

}

modifyMatrix(mat);

System.out.println("res mat:");

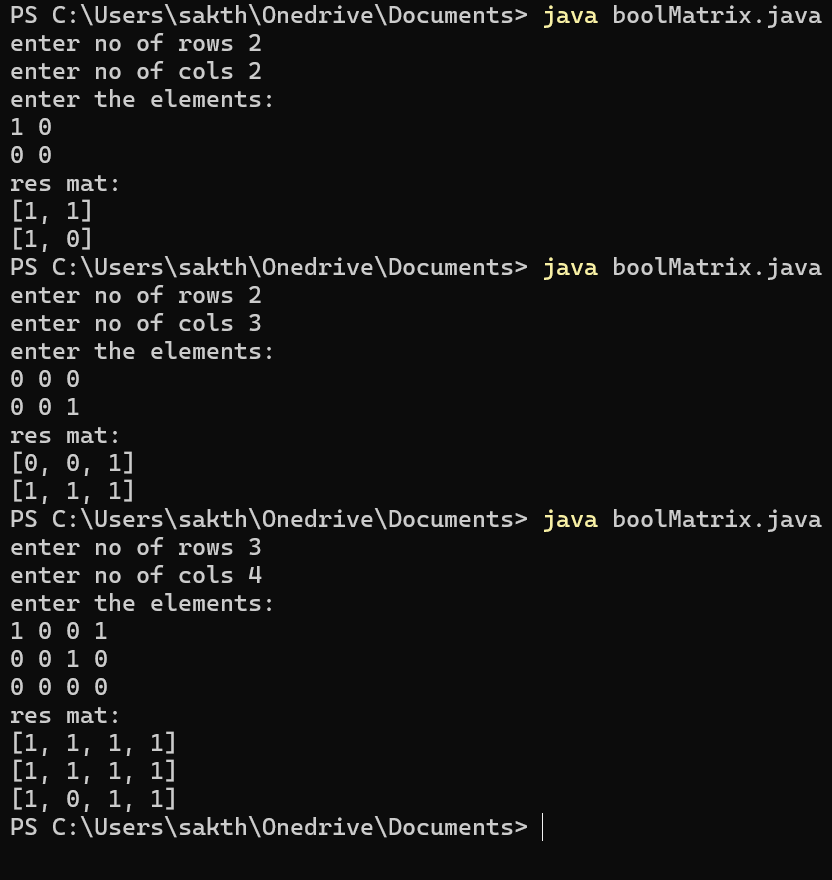
printMatrix(mat);

s.close();

}

}

Output:



10.Spiral Matrix: Time Complexity: O(n2)

import java.util.\*;

public class SpiralMatrix {

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

System.out.print("enter no of rows: ");

int r = s.nextInt();

System.out.print("enter no of cols: ");

int c = s.nextInt();

int[][] mat = new int[r][c];

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

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

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

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

}

}

int l = 0, r\_bound = c - 1;

int t = 0, b = r - 1;

while (t <= b && l <= r\_bound) {

for (int i = l; i <= r\_bound; i++) {

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

}

t++;

for (int i = t; i <= b; i++) {

System.out.print(mat[i][r\_bound] + " ");

}

r\_bound--;

if (t <= b) {

for (int i = r\_bound; i >= l; i--) {

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

}

b--;

}

if (l <= r\_bound) {

for (int i = b; i >= t; i--) {

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

}

l++;

}

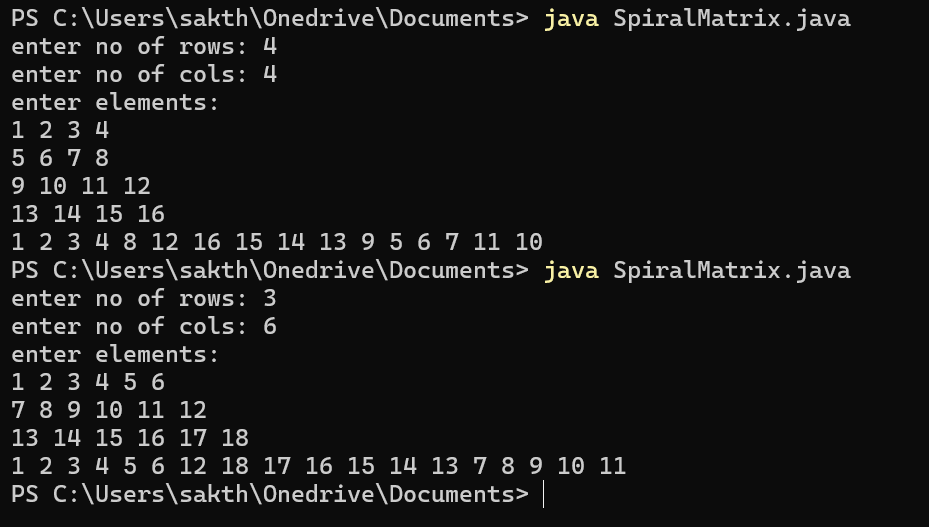
}

s.close();

}

}

Output:



16.Longest Palindromic substring: Time Complexity(O(n))

import java.util.\*;

public class LongPalindrome {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

String str = sc.nextLine();

int n = str.length();

int max = 1, s = 0;

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

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

int l = i, h = j;

boolean isPalindrome = true;

while (l < h) {

if (str.charAt(l) != str.charAt(h)) {

isPalindrome = false;

break;

}

l++;

h--;

}

if (isPalindrome && (j - i + 1) > max) {

s = i;

max = j - i + 1;

}

}

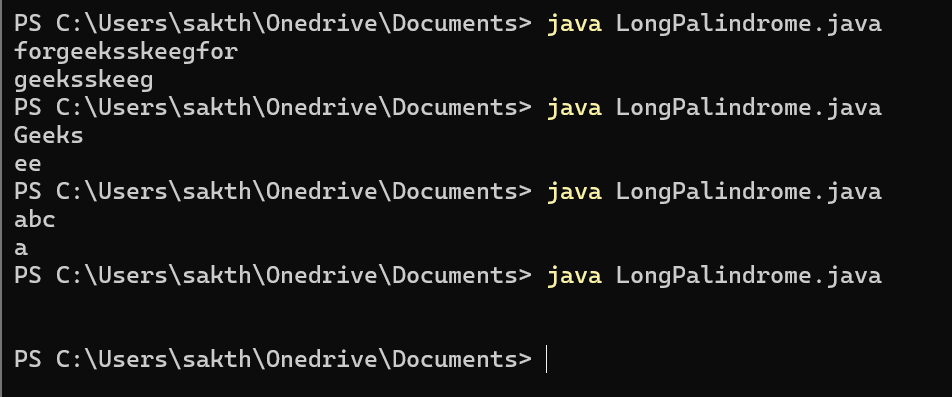
}

System.out.println(str.substring(s, s + max));

}

}

Output:



17.Delete the middle element of the stack: Time Complexity: O(n)

import java.util.\*;

public class midStack{

public static void main(String[] args)

{

Scanner s=new Scanner(System.in);

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

System.out.println("enter size of stack ");

int size=s.nextInt();

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

{

int a = s.nextInt();

st.push(a);

}

Vector<Integer> v=new Vector<>();

while(!st.empty())

{

v.add(st.pop());

}

int n=v.size();

if(n%2==0)

{

int mid=n/2;

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

{

if(i==mid){

continue;

}

st.push(v.get(i));

}

}

else{

int mid=(int) Math.ceil(n/2);

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

{

if(i==mid){

continue;

}

st.push(v.get(i));

}

}

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

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

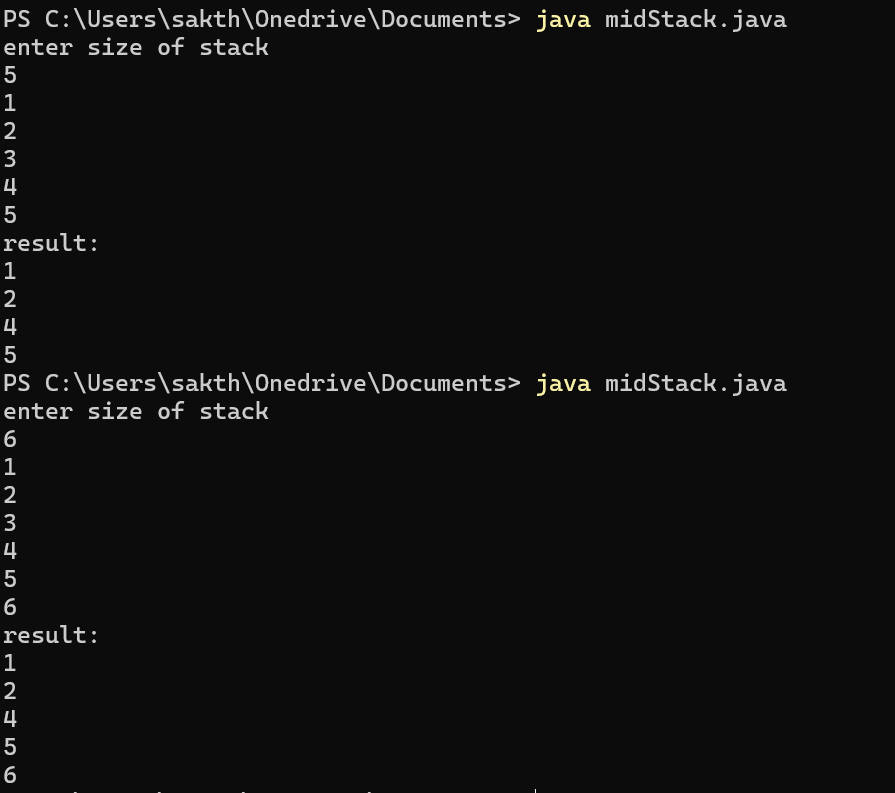
System.out.println(st.pop());

}

}

}

Output:



18.Next Greater Element : Time Complexity: O(n2)

import java.util.\*;

public class NGE{

public static void main(String[] args){

Scanner s=new Scanner(System.in);

System.out.println("enter array size");

int n=s.nextInt();

s.nextLine();

int arr[]=new int[n];

System.out.println("enter array elements");

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

arr[i]=s.nextInt();

}

int checker[]=new int[n];

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

{

checker[i]=-1;

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

{

if(arr[j]>arr[i])

{

checker[i]=arr[j];

break;

}

}

}

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

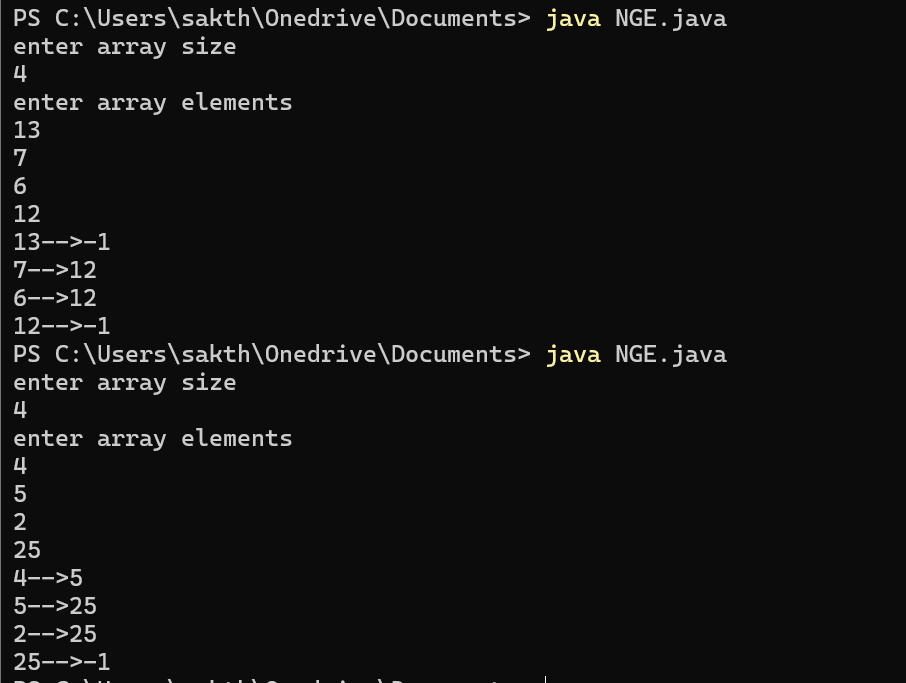
System.out.println(arr[i]+"-->"+checker[i]);

}

}

}

Output:



6. Trapping Rain Water: (Time Complexity: O(n))

import java.util.\*;

public class TrappingRainWater{

public static void main(String[] args)

{

Scanner s=new Scanner(System.in);

System.out.println("Enter blocks count");

int n=s.nextInt();

s.nextLine();

int heights[]=new int[n];

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

heights[i]=s.nextInt();

}

int l=0,r=heights.length-1,l\_max=heights[0],r\_max=heights[heights.length-1],res=0;

while(l<r){

if(l\_max<=r\_max){

res+=(l\_max-heights[l]);

l++;

l\_max=Math.max(l\_max,heights[l]);

}

else{

res+=(r\_max-heights[r]);

r--;

r\_max=Math.max(r\_max,heights[r]);

}

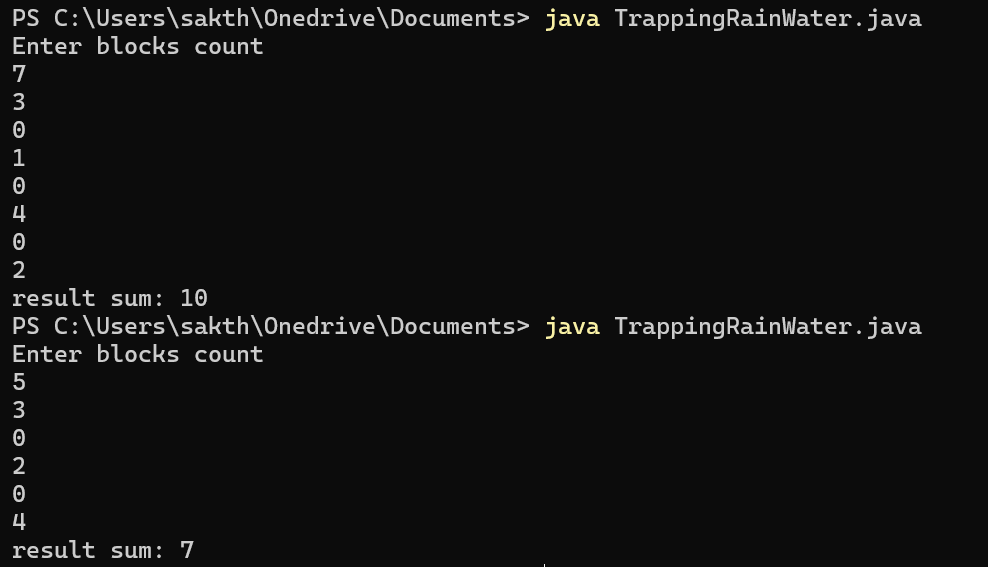
}

System.out.println(res);

}

}

Output:



19. Right View Of a Binary Tree: (Time Complexity:O(n))

import java.util.\*;

class Node

{

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class RightViewBT{

Node root;

void rightView() {

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) {

RightViewBT bt = new RightViewBT();

bt.root = new Node(1);

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

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

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

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

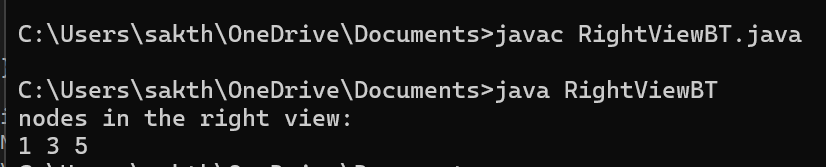
System.out.println("nodes in the right view:");

bt.rightView();

}

}

Output:



20. Maximum Height of a Binary Tree: (Time Complexity: O(n))

import java.util.\*;

class Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

public class MaxDepthBT {

int maxDepth(Node root) {

if (root == null) {

return 0;

}

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

queue.add(root);

int depth = 0;

while (!queue.isEmpty()) {

int nodeCount = queue.size();

depth++;

while (nodeCount > 0) {

Node currentNode = queue.poll();

if (currentNode.left != null) {

queue.add(currentNode.left);

}

if (currentNode.right != null) {

queue.add(currentNode.right);

}

nodeCount--;

}

}

return depth;

}

public static void main(String[] args) {

MaxDepthBT bt = new MaxDepthBT();

bt.root = new Node(12);

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

bt.root.right = new Node(18);

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

bt.root.left.right = new Node(11);

System.out.println("maximum depth of the binary tree: " + bt.maxDepth(bt.root));

}

Node root;

}

Output:

