**Gate technical training coding problem practice**

**By Jashwanth SA, CSE-C**

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

import java.util.\*;

class one {

static int kadane(int[] arr){

int res=arr[0], me=arr[0];

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

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

res=Math.max(me,res);

}

return res;

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] arr=new int[n];

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

arr[i]=sc.nextInt();

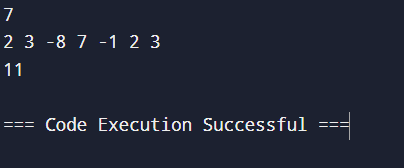
}

System.out.println(kadane(arr));

}

}

Output:



Time complexity: O(n)

1. **Maximum Product Subarray:**

import java.util.\*;

class two {

static int maxprodsub(int[] arr){

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

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

int s=arr[i],x=arr[i];

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

s=s\*arr[j];

x=Math.max(s,x);

}

res.add(x);

}

return Math.max(arr[arr.length-1],Collections.max(res));

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] arr=new int[n];

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

arr[i]=sc.nextInt();

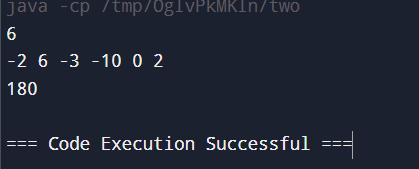
}

System.out.println(maxprodsub(arr));

}

}

Output:



Time complexity: O(n^2)

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

import java.util.\*;

class three {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

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

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

arr.add(sc.nextInt());

}

int key=sc.nextInt();

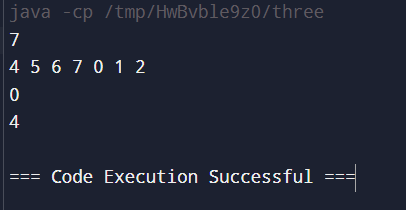
if(arr.contains(key)){System.out.println(arr.indexOf(key));}

else{System.out.println(-1);}

}

}

Output:



Time complexity: O(n)

1. **Container with Most Water**

import java.util.\*;

class four {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] arr=new int[n];

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

arr[i]=sc.nextInt();

}

int l=0, r=n-1, res=0;

while(l<r){

int vol=Math.min(arr[l],arr[r])\*(r-l);

res=Math.max(vol,res);

if(arr[l]<arr[r]){l+=1;}

else{r-=1;}

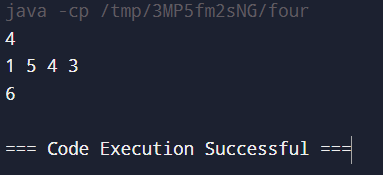
}

System.out.println(res);

}

}

Output:



Time complexity: O(n)

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

import java.util.\*;

import java.math.\*;

class five {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

BigInteger big=new BigInteger("1");

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

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

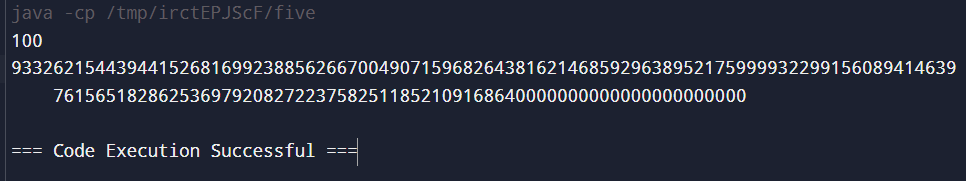
}

System.out.println(big);

}

}

Output:



Time complexity: O(n)

1. **Trapping Rainwater Problem:**

import java.util.\*;

class six {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] arr=new int[n];

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

arr[i]=sc.nextInt();

}

int l=0, r=n-1,ans=0,lmax=Integer.MIN\_VALUE,rmax=Integer.MIN\_VALUE;

while(l<r){

lmax=Math.max(lmax,arr[l]);

rmax=Math.max(rmax,arr[r]);

if(lmax<rmax){

ans=ans+lmax-arr[l];

l+=1;

}else{

ans=ans+rmax-arr[r];

r-=1;

}

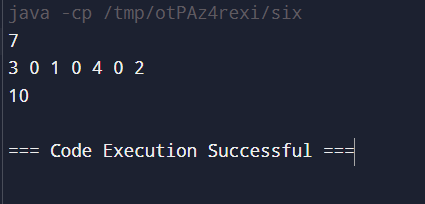
}

System.out.println(ans);

}

}

Output:



Time complexity: O(n)

1. **Chocolate Distribution Problem:**

import java.util.\*;

class seven {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

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

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

arr.add(sc.nextInt());

}

int m=sc.nextInt();

if(m==0||n==0){System.out.println(0);}

else{

Collections.sort(arr);

long res=Integer.MAX\_VALUE;

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

long diff = arr.get(i + m - 1) - arr.get(i);

if (diff < res) {

res = diff;

}

}

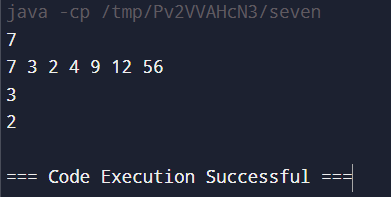
System.out.println(res);

}

}

}

Output:



Time complexity: O(n logn)

1. **Merge Overlapping Intervals:**

import java.util.\*;

public class eight {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

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

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

int a = scanner.nextInt();

int b = scanner.nextInt();

arr.add(new int[]{a, b});

}

Collections.sort(arr, (x, y) -> x[0] - y[0]);

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

int[] interval = arr.get(0);

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

if (arr.get(i)[0] >= interval[0] && arr.get(i)[0] <= interval[1]) {

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

} else {

res.add(interval);

interval = arr.get(i);

}

}

res.add(interval);

for (int[] pair : res) {

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

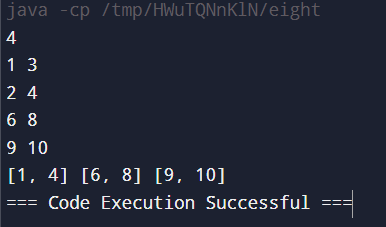
}

scanner.close();

}

}

Output:



Time complexity: O(n logn)

1. **A Boolean Matrix Question:**

import java.util.\*;

class nine {

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] + " ");

}

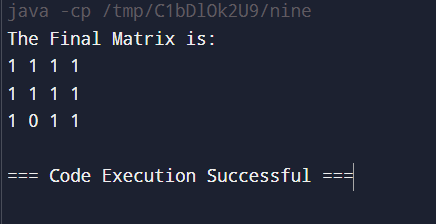
System.out.println();

}

}

}

Output:



Time complexity: O(m\*n)

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

import java.util.\*;

class ten {

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

int rows = matrix.length;

int cols = matrix[0].length;

int x = 0;

int y = 0;

int dx = 1;

int dy = 0;

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

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

res.add(matrix[y][x]);

matrix[y][x] = -101;

if (!(0 <= x + dx && x + dx < cols && 0 <= y + dy && y + dy < rows) || matrix[y+dy][x+dx] == -101) {

int temp = dx;

dx = -dy;

dy = temp;

}

x += dx;

y += dy;

}

return res;

}

public static void main(String[] args){

int[][] arr={ {1, 2, 3, 4, 5, 6},

{7, 8, 9, 10, 11, 12},

{13, 14, 15, 16, 17, 18}};

List<Integer> res=spiralOrder(arr);

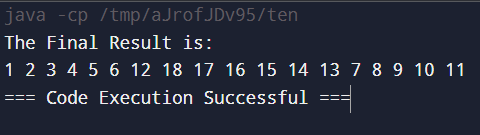
System.out.println("The Final Result is:");

for(int i:res){System.out.print(i+" ");}

}

}

Output:



Time complexity: O(n)

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

import java.util.\*;

public class thirteen {

public static boolean ispar(String s) {

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

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

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

stk.push(s.charAt(i));

}

else {

if (!stk.empty() &&

((stk.peek() == '(' && s.charAt(i) == ')'))) {

stk.pop();

}

else {

return false;

}

}

}

return stk.empty();

}

public static void main(String[] args) {

String s1 = "((()))()()";

if (ispar(s1))

System.out.println("Balanced");

else

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

String s2 ="())((())";

if (ispar(s2))

System.out.println("Balanced");

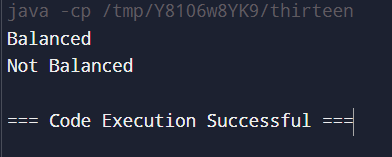
else

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

}

}

Output:



Time complexity: O(n)

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

import java.util.\*;

public class fourteen {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String s1 = scanner.nextLine();

String s2 = scanner.nextLine();

boolean f = true;

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

HashSet<Character> chars = new HashSet<>();

for (char c : s1.toCharArray()) {

chars.add(c);

}

for (char ch : chars) {

long count1 = s1.chars().filter(c -> c == ch).count();

long count2 = s2.chars().filter(c -> c == ch).count();

if (count1 != count2) {

f = false;

break;

}

}

} else {

f = false;

}

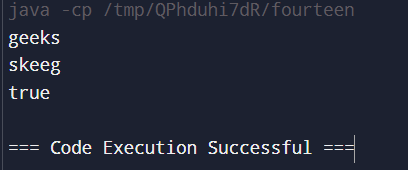
System.out.println(f);

scanner.close();

}

}

Output:



Time complexity: O(n^2)

1. **Longest Palindromic Substring:**

import java.util.\*;

public class fifteen {

public static String longestPalindrome(String s) {

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

return s;

}

int maxLen = 1;

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

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

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

if (j - i > maxLen && isPalindrome(s.substring(i, j))) {

maxLen = j - i;

maxStr = s.substring(i, j);

}

}

}

return maxStr;

}

private static boolean isPalindrome(String str) {

int left = 0;

int right = str.length() - 1;

while (left < right) {

if (str.charAt(left) != str.charAt(right)) {

return false;

}

left++;

right--;

}

return true;

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

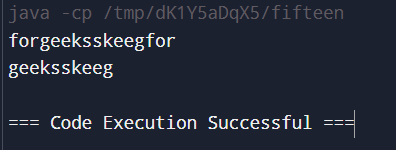
String s=sc.nextLine();

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

}

}

Output:



Time complexity: O(n^3)

1. . **Longest Common Prefix using Sorting:**

import java.util.Arrays;

class sixteen {

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));

String[] arr1={"hello", "world"};

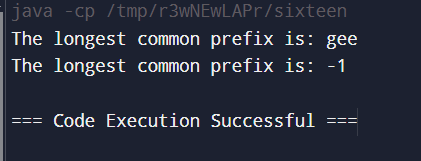
System.out.println("The longest common prefix is: "

+ longestCommonPrefix(arr1));

}

}

Output:



Time complexity: O(n logn + m)

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

import java.util.\*;

public class seventeen {

public static void main(String[] args) {

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

for(int i=1;i<=6;i++){st.push(i);}

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

while (!st.empty()) {

v.add(st.pop());

}

int n = v.size();

if (n % 2 == 0) {

int target = (n / 2);

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

if (i == target) continue;

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

}

} else {

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

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

if (i == target) continue;

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

}

}

System.out.print("Printing stack after deletion of middle: ");

while (!st.empty()) {

int p = st.pop();

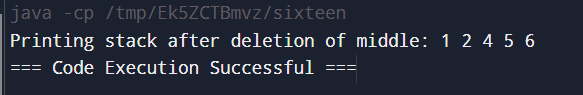
System.out.print(p + " ");

}

}

}

Output:



Time complexity: O(n)

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

import java.util.\*;

public class eighteen {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] arr=new int[n];

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

arr[i]=sc.nextInt();

}

int[] res=new int[n];

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

int f=0;

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

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

res[i]=arr[j];

f=1;

break;

}

}

if(f==0){res[i]=-1;}

}

res[n-1]=-1;

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

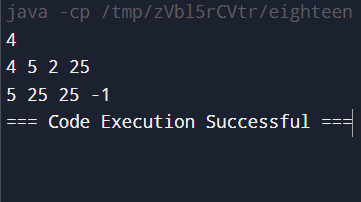
System.out.print(res[i] + " ");

}

}

}

Output:



Time complexity: O(n^2)

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

import java.util.\*;

public class nineteen {

static class Node {

int val;

Node left;

Node right;

Node() {}

Node(int val) { this.val = val; }

Node(int val, Node left, Node right) {

this.val = val;

this.left = left;

this.right = right;

}

}

public static List<Integer> rightSideView(Node root) {

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

rightView(root, result, 0);

return result;

}

public static void rightView(Node 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 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);

List<Integer> res = rightSideView(root);

for (int val : res) {

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

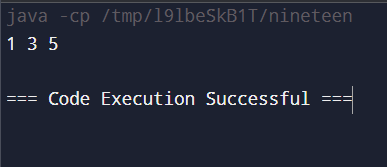
}

System.out.println();

}

}

Output:



Time complexity: O(n)

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

class twenty {

static class Node {

int data;

Node left, right;

Node(int val) {

data = val;

left = null;

right = null;

}

}

static int maxDepth(Node node) {

if (node == null)

return 0;

int lDepth = maxDepth(node.left);

int rDepth = maxDepth(node.right);

return Math.max(lDepth, rDepth) + 1;

}

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);

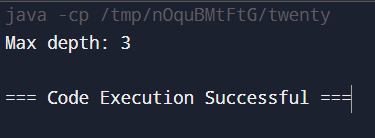
root.left.right = new Node(5);

System.out.println("Max depth: "+maxDepth(root));

}

}

Output:



Time complexity: O(n)