**CODING PRACTICE PROBLEMS**

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.

import java.util.LinkedList;

import java.util.Queue;

class Node{

int data;

Node left;

Node right;

Node(int val){

data = val;

left = null;

right = null;

}

}

class solution{

int maxDepth(Node root){

if(root == null){

return 0;

}

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

int level = 0;

q.add(root);

while(!q.isEmpty()){

int size = q.size();

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

Node front = q.poll();

if(front.left!=null){

q.add(front.left);

}

if(front.right!=null){

q.add(front.right);

}

}

level ++ ;

}

return level;

}

}

public class Main {

public static void main(String args[]){

Node root = new Node(12);

root.left = new Node(8);

root.right = new Node(18);

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

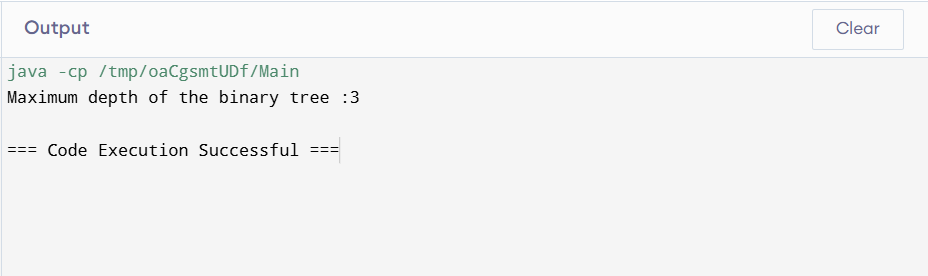
solution solution = new solution();

int depth = solution.maxDepth(root);

System.out.println("Maximum depth of the binary tree :" + depth);

}

}



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.

import java.util.\*;

class Node{

int data;

Node left;

Node right;

Node(int val){

data = val;

left = null;

right = null;

}

}

public class solution{

public static void rightview(Node curr, List<Integer> result , int currdepth){

if(curr == null){

return;

}

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

result.add(curr.data);

}

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> result = new ArrayList<>();

rightview(root,result,0);

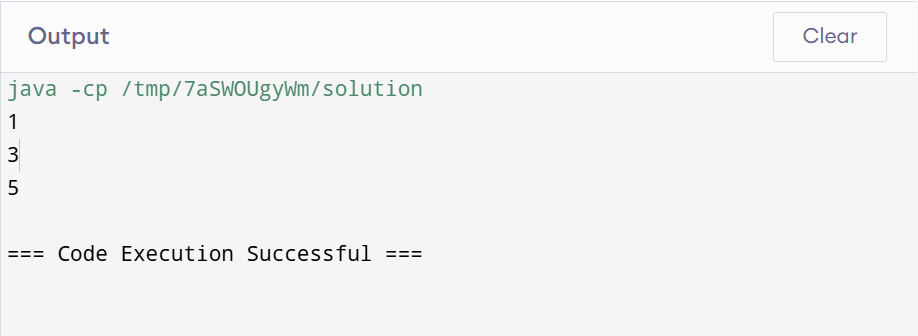
for (int i : result){

System.out.println(i);

}

}

}



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. Input: arr[] = [ 4 , 5 , 2 , 25 ]

import java.util.\*;

class HelloWorld {

public static void nextgrt(int[] arr, int[] ans){

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

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

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

st.pop();

}

if(i<arr.length){

if(!st.isEmpty()){

ans[i]=st.peek();

}

else{

ans[i]=-1;

}

}

st.push(arr[i]);

}

}

public static void main(String[] args) {

int arr[]={4,5,2,25};

int ans[]=new int[arr.length];

nextgrt(arr,ans);

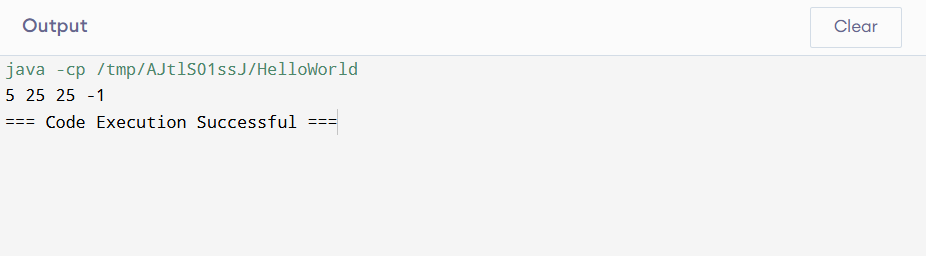
for(int i:ans){

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

}

}

}



Time complexity : O(N)

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.

import java.util.\*;

class HelloWorld {

public static void dltmid(Stack<Integer> st, int n, int curr){

if(st.isEmpty() || curr==n){

return;

}

int x=st.pop();

dltmid(st,n,curr+1);

if(curr!=n/2){

st.push(x);

}

}

public static void main(String[] args) {

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

st.push(6);

st.push(5);

st.push(4);

st.push(3);

st.push(2);

st.push(1);

int n=st.size();

dltmid(st,n,0);

while(!st.isEmpty()){

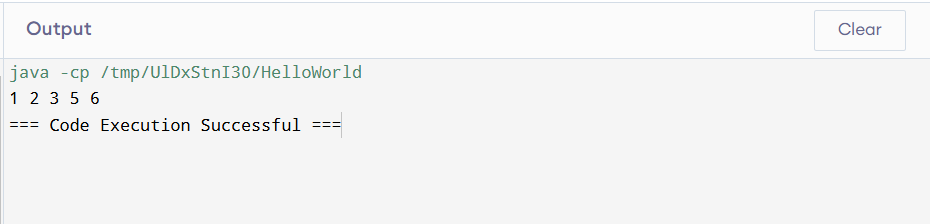
int i=st.pop();

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

}

}

}



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”.

import java.util.\*;

public class helloworld{

public static String lowestcommonprefix(String[] v){

StringBuilder ans = new StringBuilder();

Arrays.sort(v);

String first = v[0];

String last = v[v.length - 1];

for(int i = 0; i < Math.min(first.length() , last.length()) ; i++ ){

if(first.charAt(i) != last.charAt(i)){

return ans.toString();

}

ans.append(first.charAt(i));

}

return ans.toString();

}

public static void main(String args[]){

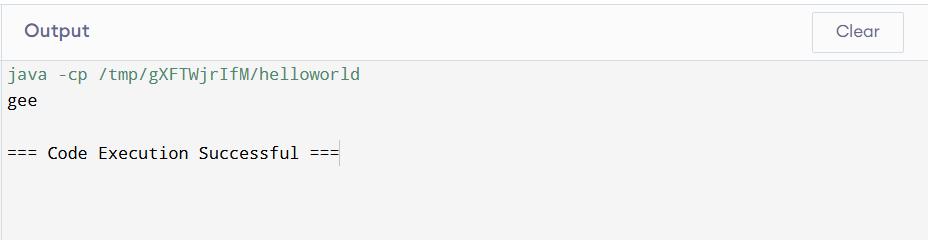
String arr[]= {"geeksforgeeks","geeks","geek","geezer"};

String st = lowestcommonprefix(arr);

System.out.println(st);

}

}



Time complexity : O(NlogN +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.

public class LongPalindrome {

public String longestPalindrome(String s) {

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

return s;

}

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

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

String odd = expandFromCenter(s, i, i);

String even = expandFromCenter(s, i, i + 1);

if (odd.length() > maxStr.length()) {

maxStr = odd;

}

if (even.length() > maxStr.length()) {

maxStr = even;

}

}

return maxStr;

}

private String expandFromCenter(String s, int left, int right) {

while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {

left--;

right++;

}

return s.substring(left + 1, right);

}

public static void main(String[] args) {

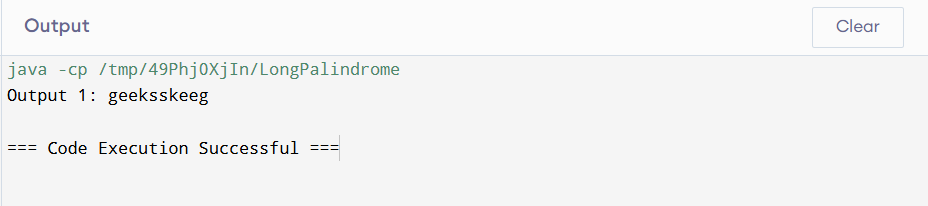
LongPalindrome sol = new LongPalindrome();

String str1 = "forgeeksskeegfor";

System.out.println("Output 1: " + sol.longestPalindrome(str1));

}

}



Time complexity : O(N^2)

**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.

import java.util.Arrays;

class Anagram {

public boolean isAnagram(String s, String t) {

if(s.length()!=t.length()){

return false;

}

char[] c1 = s.toCharArray();

char[] c2 = t.toCharArray();

Arrays.sort(c1);

Arrays.sort(c2);

for(int i = 0; i < c1.length; i++){

if(c1[i] != c2[i]){

return false;

}

}

return true;

}

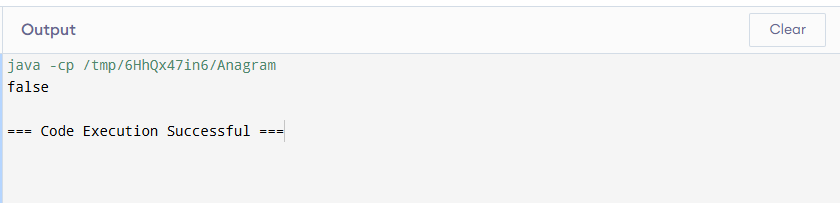
public static void main(String[] args) {

Anagram solution = new Anagram();

String s1 = "allergy";

String s2 = "allergic";

System.out.println(solution.isAnagram(s1, s2));

}}  
  


Time complexity : O(NlogN)

**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.

import java.util.Stack;

public class Parentheses {

public static String result(String str) {

Stack<Character> st1 = new Stack<>();

Stack<Character> st2 = new Stack<>();

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

if (I == ')' && !st1.isEmpty() && st1.peek() == '(') {

st1.pop();

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

st2.push(')');

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

st1.push('(');

}

}

return (st1.isEmpty() && st2.isEmpty()) ? "Balance" : "Not Balanced";

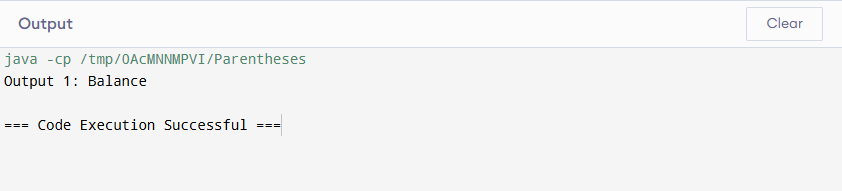
}

public static void main(String[] args) {

String str1 = "((()))()()";

System.out.println("Output 1: " + result(str1));

}

}  
  


Time complexity : O(N)  
  
**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.**

import java.util.Scanner;

public class SprialMatrix {

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 values:");

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

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

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

}

}

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

printSpiral(matrix, m, n);

scanner.close();

}

public static void printSpiral(int[][] matrix, int m, int n) {

int top = 0, bottom = m - 1, 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] + " ");

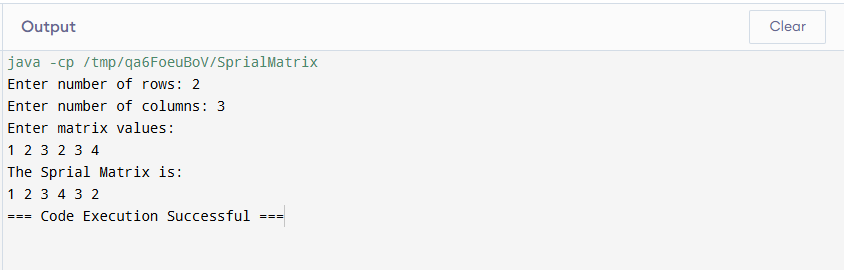
}

left++;

}

}

}

}  
  


Time complexity : O(M\*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. Input: {{1, 0}, {0, 0}}

import java.util.Scanner;

public class ModifyMatrix {

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 values (0 or 1):");

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

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

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

}

}

boolean[] rowFlag = new boolean[M];

boolean[] colFlag = new boolean[N];

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

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

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

rowFlag[i] = true;

colFlag[j] = true;

}

}

}

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

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

if (rowFlag[i] || colFlag[j]) {

mat[i][j] = 1;

}

}

}

System.out.println("Modified matrix:");

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

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

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

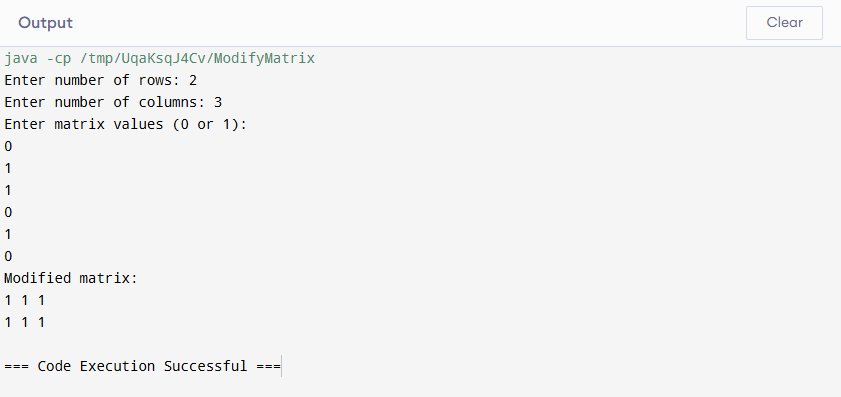
}

System.out.println();

}

scanner.close();

}

}  
  


Time complexity : O(M\*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.

import java.util.\*;

public class MergeInterval {

public static int[][] mergeIntervals(int[][] intervals) {

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

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

for (int[] interval : intervals) {

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

merged.add(interval);

} else {

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

}

}

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

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

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

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

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

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

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

}

int[][] result = mergeIntervals(intervals);

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

System.out.println(Arrays.deepToString(result));

}

}



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: Each student gets exactly one packet. The difference between the

maximum and minimum number of chocolates in the packets given to the students is minimized.

import java.util.Arrays;

import java.util.Scanner;

public class ChocolateDistribution {

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

int n=arr.length;

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

return 0;

}

if(m>n) {

return -1;

}

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];

mindiff=Math.min(mindiff, diff);

}

return mindiff;

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

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

int n=sc.nextInt();

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

int[] arr=new int[n];

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

arr[i]=sc.nextInt();

}

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

int m=sc.nextInt();

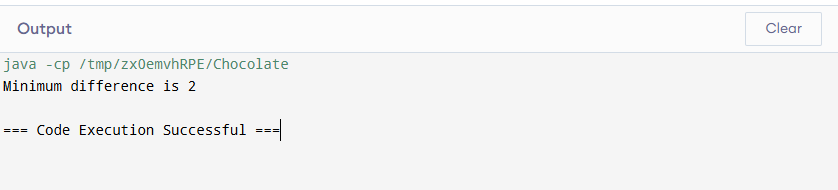
ChocolateDistribution obj=new ChocolateDistribution();

int result=obj.distribution(arr,m);

System.out.println("The Maximum Product of subarray is:" + result);

}

}

  
  
Time complexity: O(Nlog 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.**

import java.util.\*;

public class TrappedWater {

public static int trap(int[] height) {

int n = height.length;

if (n == 0) return 0;

int[] left = new int[n];

int[] right = new int[n];

int storedWater = 0;

left[0] = height[0];

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

left[i] = Math.max(left[i-1], height[i]);

}

right[n - 1] = height[n - 1];

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

right[i] = Math.max(right[i + 1], height[i]);

}

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

int minHeight = Math.min(left[i], right[i]);

storedWater += minHeight - height[i];

}

return storedWater;

}

public static void main(String[] args) {

int[] height1 = {3, 0, 1, 0, 4, 0, 2};

TrappedWater solution1 = new TrappedWater();

int trappedWater1 = TrappedWater.trap(height1);

System.out.println("Total trapped water: "+trappedWater1);

int[] height2 = {3, 0, 2, 0, 4};

TrappedWater solution2 = new TrappedWater();

int trappedWater2 = TrappedWater.trap(height2);

System.out.println("Total trapped water: "+trappedWater2);

int[] height3 = {1, 2, 3, 4};

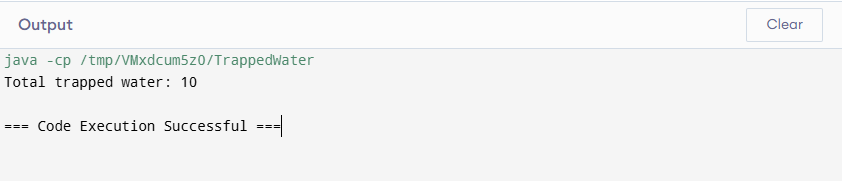
TrappedWater solution3 = new TrappedWater();

int trappedWater3 = TrappedWater.trap(height3);

System.out.println("Total trapped water: "+trappedWater3);

}

}



Time complexity : O(N)

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

import java.util.Scanner;

import java.math.BigInteger;

public class Factorial {

public static BigInteger fact(BigInteger n) {

if (n.equals(BigInteger.ONE)) {

return BigInteger.ONE;

} else if (n.compareTo(BigInteger.ONE) > 0) {

return n.multiply(fact(n.subtract(BigInteger.ONE)));

} else {

return BigInteger.valueOf(-1);

}

}

public static void main(String[] args) {

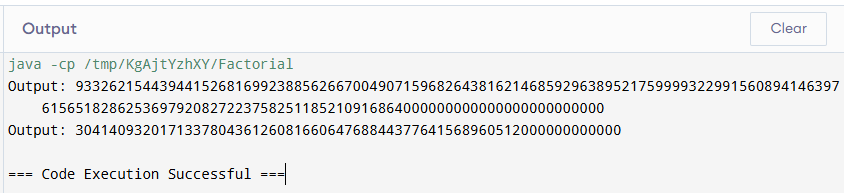
Scanner sc = new Scanner(System.in);

System.out.println("Output: " + fact(BigInteger.valueOf(100)));

System.out.println("Output: " + fact(BigInteger.valueOf(50)));

}

}



Time complexity : O(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, ai) 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.

import java.util.Scanner;

public class maxArea{

public static int maxArea(int[] height) {

int left = 0;

int right = height.length - 1;

int maxArea = 0;

while (left < right) {

int currentArea = Math.min(height[left], height[right]) \* (right - left);

maxArea = Math.max(maxArea, currentArea);

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

left++;

} else {

right--;

}

}

return maxArea;

}

public static void main(String[] args) {

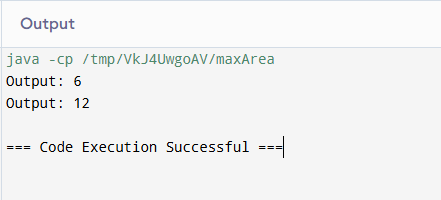
Scanner scanner = new Scanner(System.in);

System.out.println("Output: " + maxArea(new int[] {1, 5, 4, 3}));

System.out.println("Output: " + maxArea(new int[] {3, 1, 2, 4, 5}));

}

}



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.

import java.util.\*;

public class searchrotatedarray {

public static void main(String[] args) {

search(new int[]{4, 5, 6, 7, 0, 1, 2} , 0);

search(new int[] { 4, 5, 6, 7, 0, 1, 2 } , 3);

search(new int[]{50, 10, 20, 30, 40} , 10);

}

public static void search(int[] arr , int a){

System.out.print(Arrays.toString(arr) + " : ");

boolean flag = false;

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

if(arr[i] == a){

System.out.println(i);

flag = true;

break;

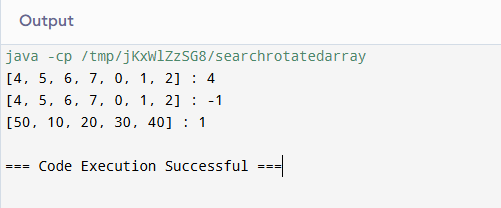
}

}

if(!flag) System.out.println(-1);

}

}



Time complexity: O(N)

2. Maximum Product Subarray Given an integer array, the task is to find the maximum

product of any subarray.

import java.util.Arrays;

public class maxprosubarray {

public static void main(String[] args) {

maxproduct(new int[]{-2, 6, -3, -10, 0, 2});

maxproduct(new int[]{-1, -3, -10, 0, 60});

}

public static void maxproduct(int[] arr){

System.out.print(Arrays.toString(arr) + " :");

int ans = 0;

int max = arr[0];

int min = arr[0];

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

if(arr[i]<0){

int temp = max;

max = min;

min = temp;

}

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

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

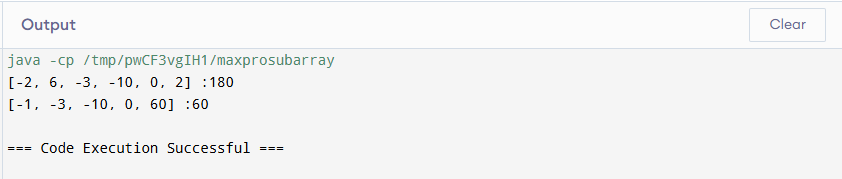
ans = Math.max(ans,max);

}

System.out.println(ans);

}

}



Time complexity: O(N)

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.  
  
import java.util.\*;

public class maxsubarray {

public static void main(String[] args) {

maxsubsum(new int[]{2, 3, -8, 7, -1, 2, 3});

maxsubsum(new int[]{5, 4, 1, 7, 8});

}

public static void maxsubsum(int[] arr){

System.out.print(Arrays.toString(arr) + " :");

int ans = 0;

int max = arr[0];

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

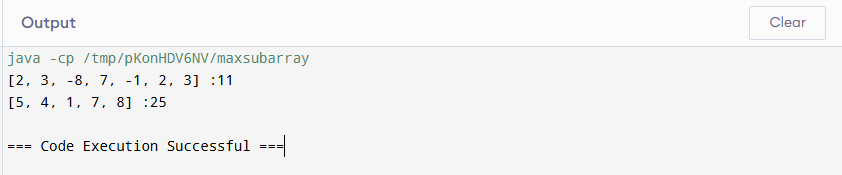
max = Math.max(max+arr[i],arr[i]);

ans = Math.max(ans,max);

}

System.out.println(ans);

}

}  
  
  
  
Time complexity : O(N)