SDE ASSIGNMENT – 1

T. Shankar Sai(22AD144)

9-11-2024

QUESTIONS:

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.

Input: arr[] = {2, 3, -8, 7, -1, 2, 3}

Output: 11

Explanation: The subarray {7, -1, 2, 3} has the largest sum 11.

Input: arr[] = {-2, -4}

Output: –2

Explanation: The subarray {-2} has the largest sum -2.

Input: arr[] = {5, 4, 1, 7, 8}

Output: 25

**class HelloWorld {**

**public void kadane(int[] arr){**

**int ms=Integer.MIN\_VALUE,cs=0,minn=Integer.MIN\_VALUE;**

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

**cs+=arr[i];**

**if(arr[i]>minn) minn=arr[i];**

**if(cs<0) cs=0;**

**ms=Math.max(cs,ms);**

**}**

**if(ms>0) System.out.println(ms);**

**else System.out.println(minn);**

**return;**

**}**

**public static void main(String[] args) {**

**int[] arr1 = {2, 3, -8, 7, -1, 2, 3};**

**int[] arr2 = {-2, -4};**

**int[] arr3 = {5, 4, 1, 7, 8};**

**HelloWorld obj=new HelloWorld();**

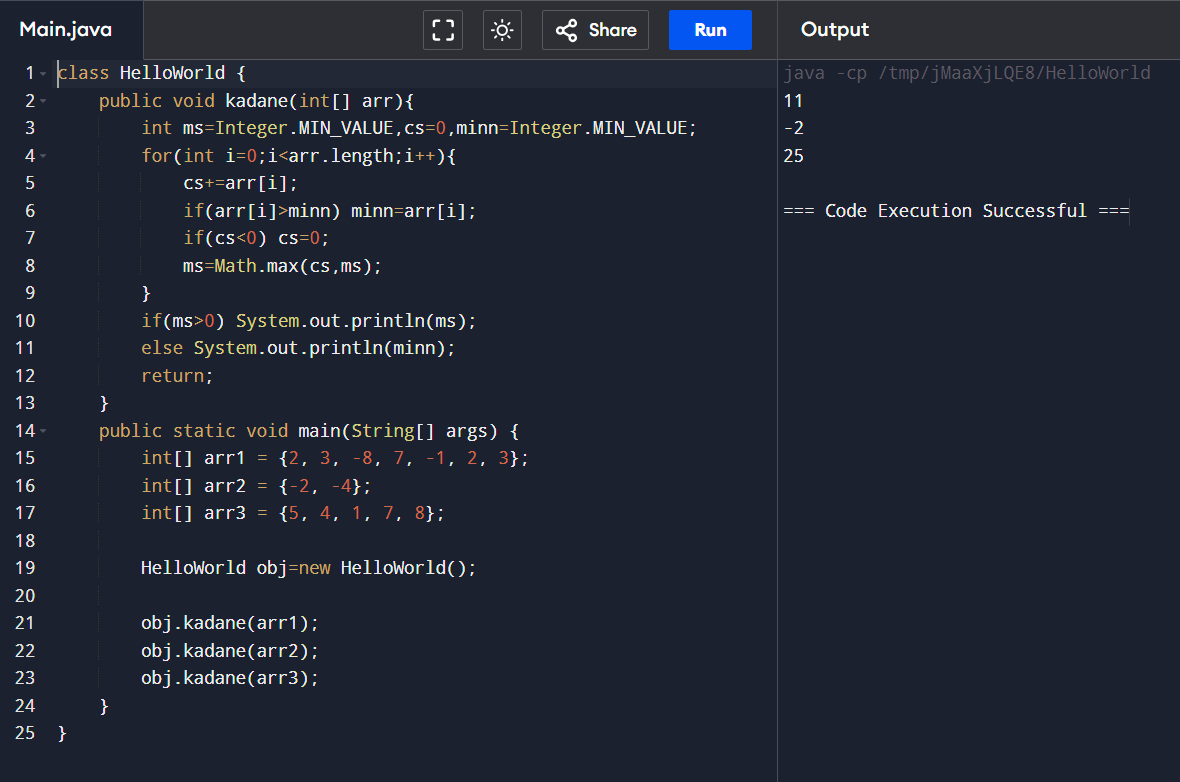
**obj.kadane(arr1);**

**obj.kadane(arr2);**

**obj.kadane(arr3);**

**}**

**}**



**Time Complexity: Big O(n), were n = len(arr)**

2. Maximum Product Subarray

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

Input: arr[] = {-2, 6, -3, -10, 0, 2}

Output: 180

Explanation: The subarray with maximum product is {6, -3, -10} with product = 6 \* (-3) \* (-10)

= 180

Input: arr[] = {-1, -3, -10, 0, 60}

Output: 60

Explanation: The subarray with maximum product is {60}.

**class HelloWorld {**

**public void optimal\_maxProd(int[] arr){**

**//ArrayList<Integer> mada = new ArrayList<>();**

**int maxx=arr[0],minn=arr[0],res=arr[0];**

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

**int temp=Math.max(arr[i],Math.max(maxx\*arr[i],minn\*arr[i]));**

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

**maxx=temp;**

**res=Math.max(maxx,res);**

**}**

**System.out.println(res);**

**return;**

**}**

**public void maxProd(int[] arr){**

**int maxx=Integer.MIN\_VALUE;**

**int curr;**

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

**curr=arr[i];**

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

**maxx=Math.max(curr,maxx);**

**curr\*=arr[j];**

**}**

**maxx=Math.max(maxx,curr);**

**}System.out.println(maxx);return;**

**}**

**public static void main(String[] args) {**

**int[] arr1 = {2, 3, -8, 7, -1, 2, 3};**

**int[] arr2 = {-2, -4};**

**int[] arr3 = {5, 4, 1, 7, 8};**

**int[] arr4 = {-2, 6, -3, -10, 0, 2};**

**int[] arr5 = {-1, -3, -10, 0, 60};**

**HelloWorld obj=new HelloWorld();**

**obj.optimal\_maxProd(arr4);**

**obj.optimal\_maxProd(arr5);**

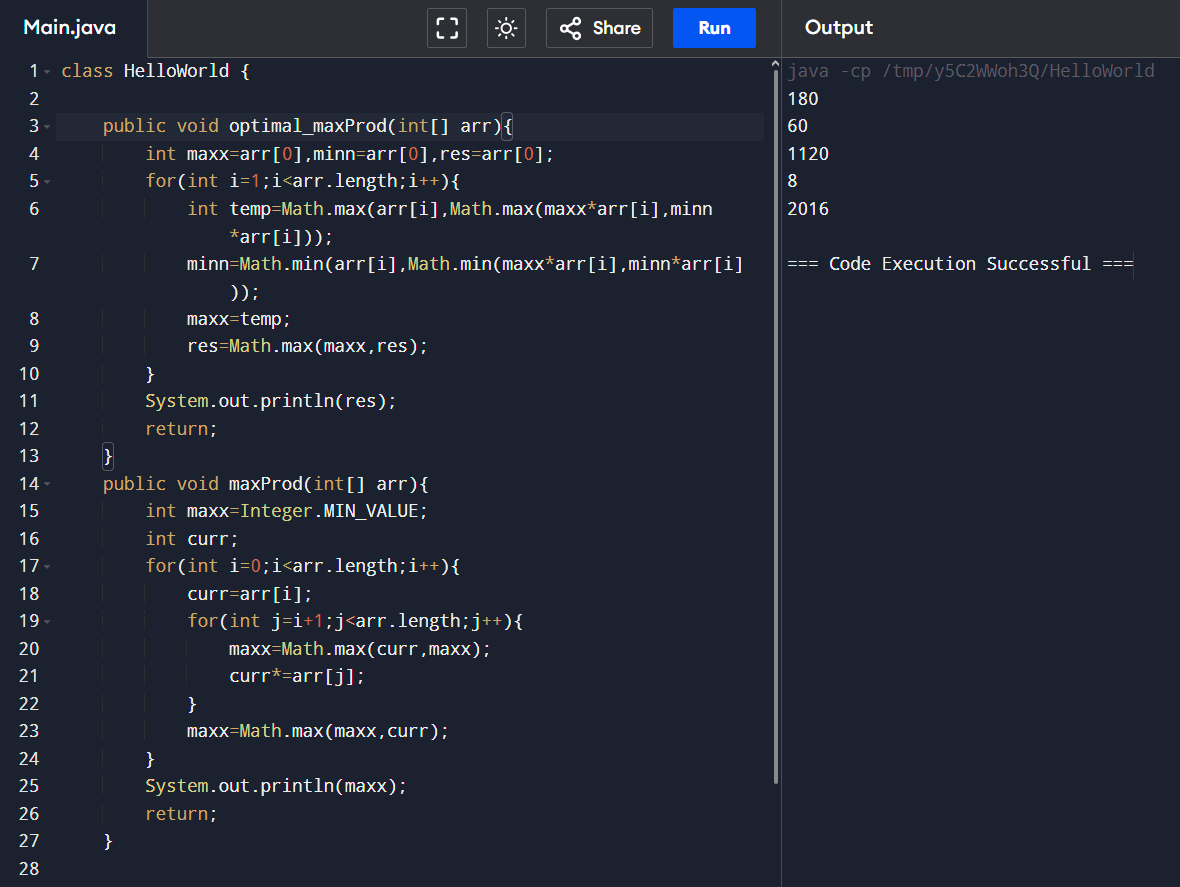
**obj.optimal\_maxProd(arr3);**

**obj.optimal\_maxProd(arr2);**

**obj.optimal\_maxProd(arr1);**

**}**

**}**



**Time Complexity : Big O(n), were n= arr.length-1**

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.

Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0

Output : 4

Input : arr[] = { 4, 5, 6, 7, 0, 1, 2 }, key = 3

Output : -1

Input : arr[] = {50, 10, 20, 30, 40}, key = 10

Output : 1

**class HelloWorld {**

**public void key(int[] arr,int key){**

**int mad=0;boolean falg=false;**

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

**if(arr[i]==key && falg==false){**

**falg=true;**

**}**

**if(falg==true) mad++;**

**}**

**if(falg==false){**

**System.out.println(-1);**

**}else{**

**System.out.println(arr.length-mad);**

**}**

**return;**

**}**

**public static void main(String[] args) {**

**int[] arr1 = {4, 5, 6, 7, 0, 1, 2};**

**int[] arr2 = { 4, 5, 6, 7, 0, 1, 2 };**

**int[] arr3 = {50, 10, 20, 30, 40};**

**int[] arr4 = {-2, 0, 2, 6,-10, -3,};**

**HelloWorld obj=new HelloWorld();**

**obj.key(arr1,0);**

**//obj.optimal\_maxProd(arr5);**

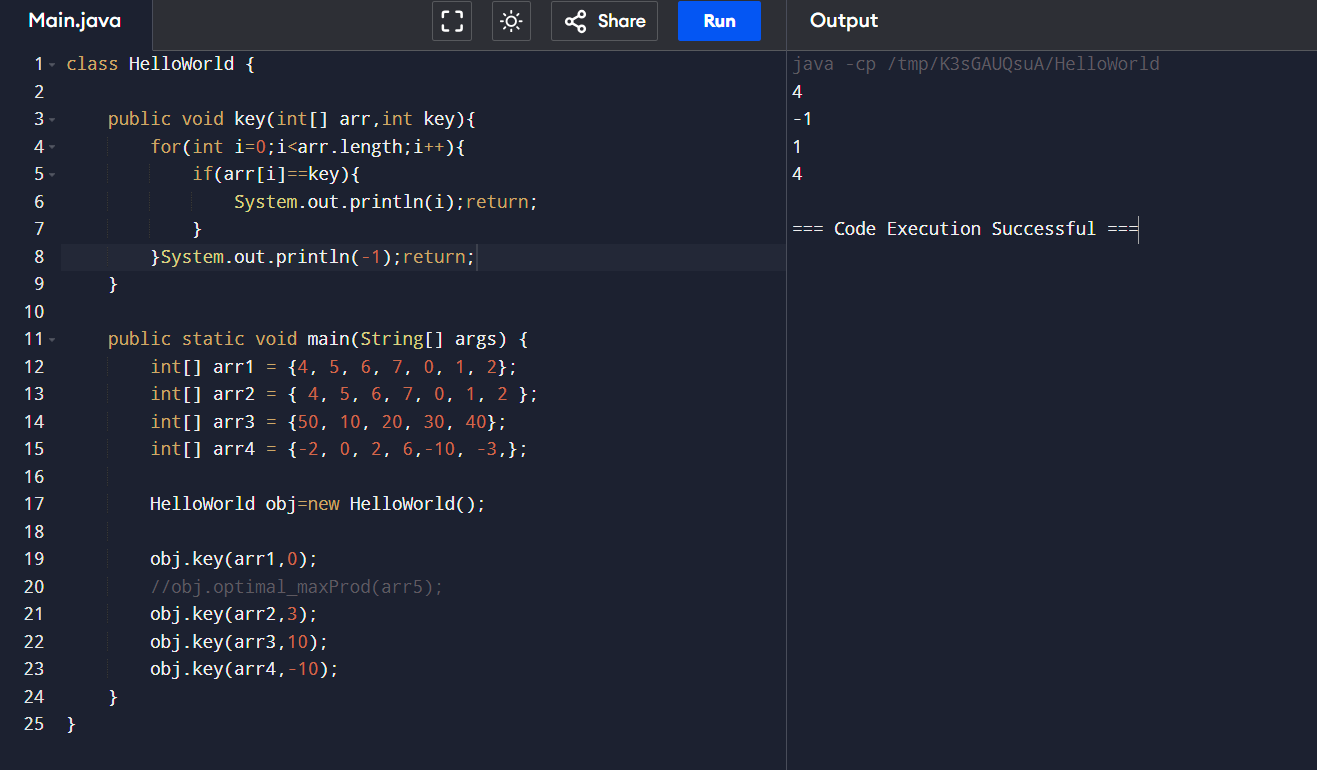
**obj.key(arr2,3);**

**obj.key(arr3,10);**

**obj.key(arr4,-10);**

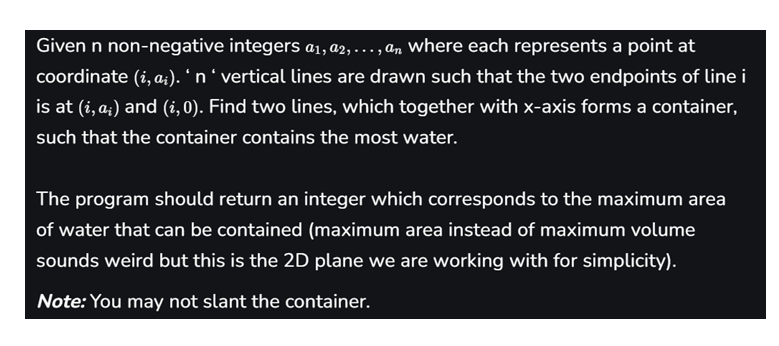
**}**

**}**



**Time Complexity: O(n) were, n= arr.length**

4. Container with Most Water



Input: arr = [1, 5, 4, 3]

Output: 6

Explanation:

5 and 3 are distance 2 apart. So the size of the base = 2.

Height of container = min(5, 3) = 3. So total area = 3 \* 2 = 6

Input: arr = [3, 1, 2, 4, 5]

Output: 12

Explanation:

5 and 3 are distance 4 apart. So the size of the base = 4.

Height of container = min(5, 3) = 3. So total area = 4 \* 3 = 12

**class HelloWorld {**

**public int container(int[] arr){**

**int left=0,right=arr.length-1,maxx=0;**

**while(left<right){**

**int width=right-left;**

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

**maxx=Math.max(maxx,width\*height);**

**if(arr[left]<arr[right]) left++;**

**else right--;**

**}**

**return maxx;**

**}**

**public int suboptimal\_container(int[] arr){**

**int left=0,right=arr.length-1,maxx=-1;**

**for(int i=0;i<arr.length/2;i++){**

**left=0;**

**while(left<right){**

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

**left++;**

**}right--;**

**}return maxx;**

**}**

**public static void main(String[] args) {**

**HelloWorld obj = new HelloWorld();**

**int[] arr1={1, 5, 4, 3};**

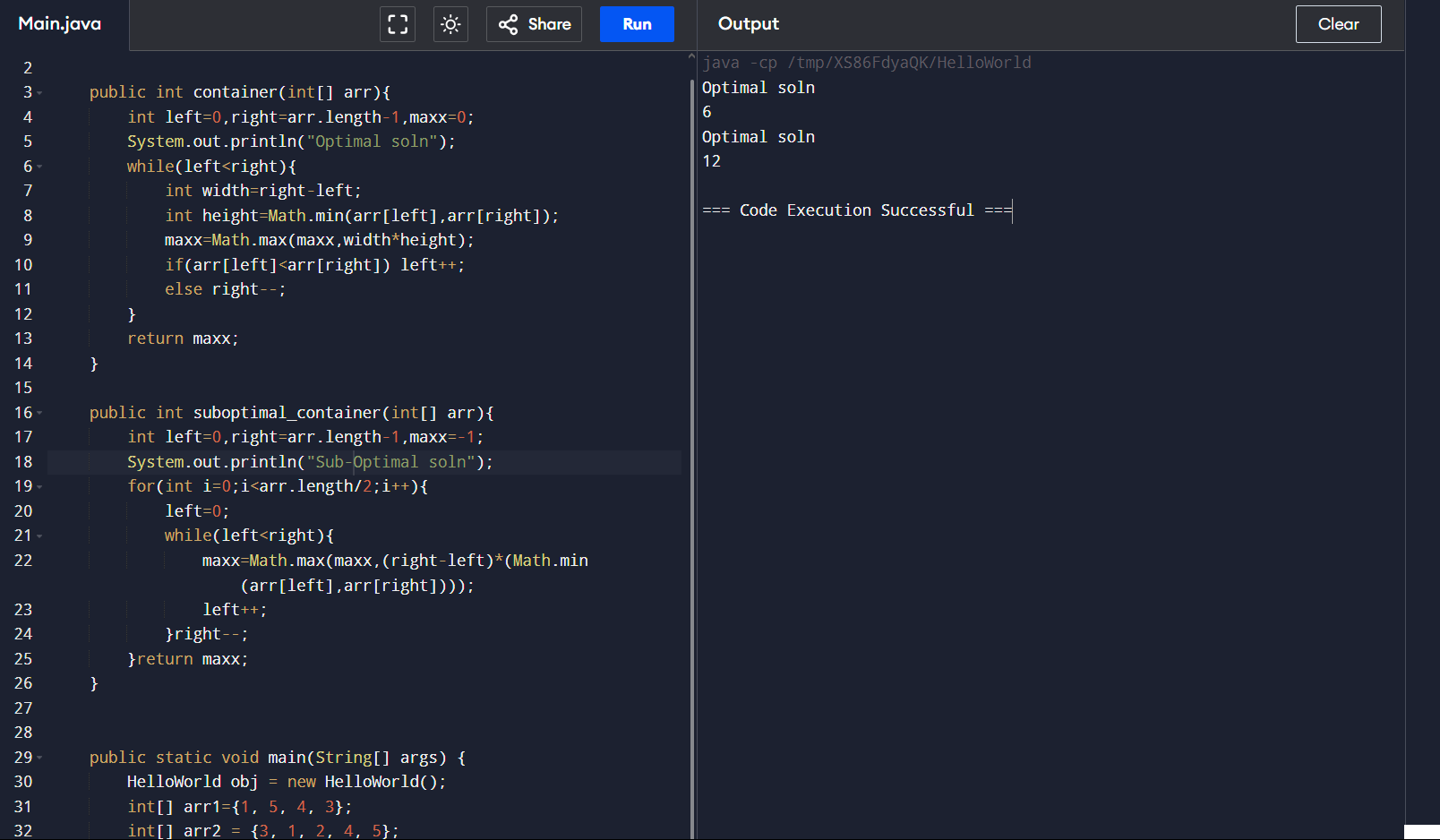
**int[] arr2 = {3, 1, 2, 4, 5};**

**System.out.println(obj.container(arr1));**

**System.out.println(obj.container(arr2));**

**}**

**}**



**The time complexity of container is O(n), and for suboptimal\_container, it is O(n^2).**

5. Find the Factorial of a large number

Input: 100

Output:

933262154439441526816992388562667004907159682643816214685929638952175999932299

156089414639761565182862536979208272237582511852109168640000000000000000000000

00

Input: 50

Output: 30414093201713378043612608166064768844377641568960512000000000000

**import java.math.BigInteger;**

**class HelloWorld {**

**public BigInteger factorial(int fact){**

**BigInteger res = BigInteger.valueOf(fact);**

**if(fact>1){**

**res=res.multiply(factorial(fact-1));**

**}else{**

**System.out.println(res);**

**}**

**return res;**

**}**

**public static void main(String[] args) {**

**HelloWorld obj = new HelloWorld();**

**int num1 = 100;**

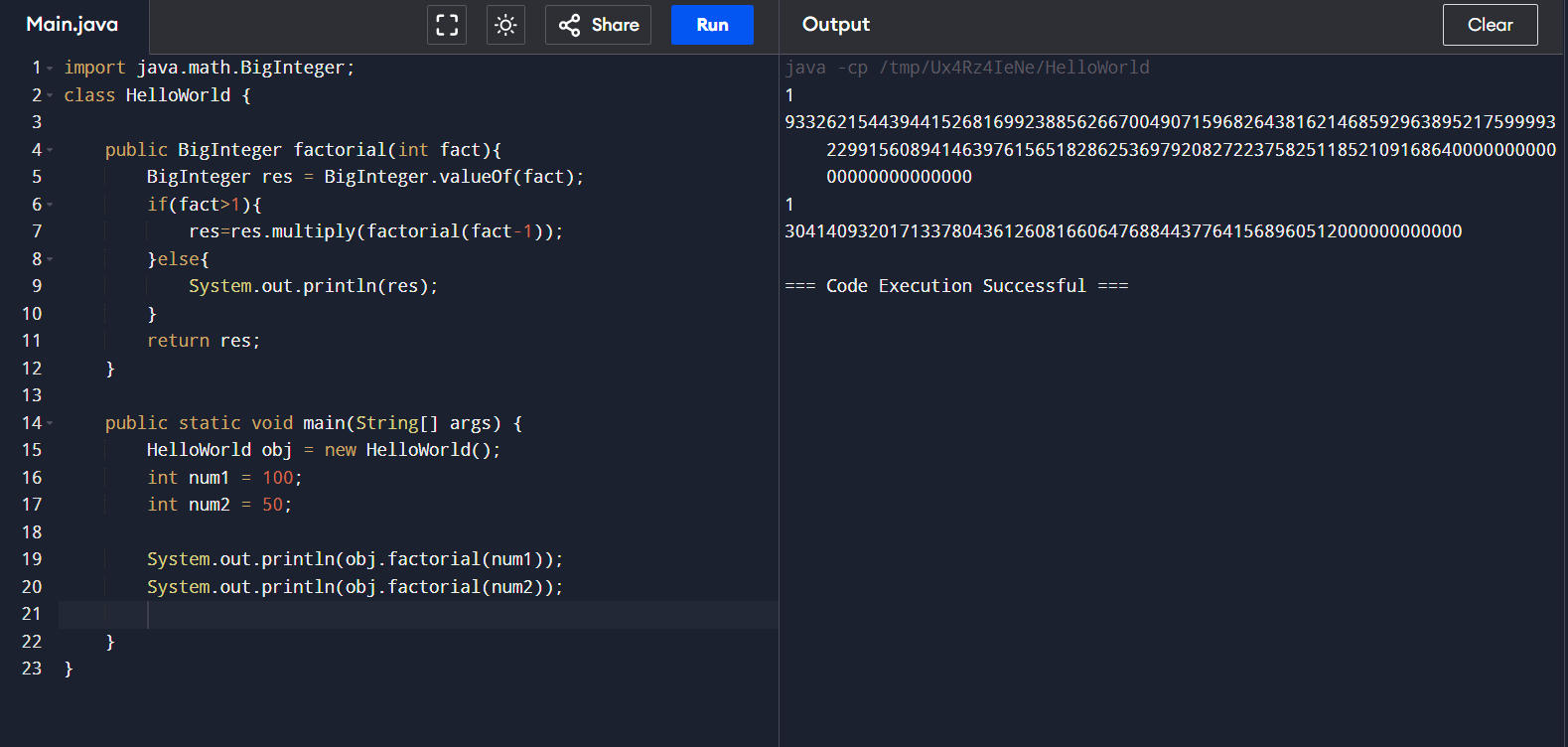
**int num2 = 50;**

**System.out.println(obj.factorial(num1));**

**System.out.println(obj.factorial(num2));**

**}**

**}**



**Time Complexity: Big O(n), were n is the passed int input.**

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.

Input: arr[] = {3, 0, 1, 0, 4, 0, 2}

Output: 10

Explanation: The expected rainwater to be trapped is shown in the above image.

Input: arr[] = {3, 0, 2, 0, 4}

Output: 7

Explanation: We trap 0 + 3 + 1 + 3 + 0 = 7 units.

Input: arr[] = {1, 2, 3, 4}

Output: 0

Explanation : We cannot trap water as there is no height bound on both sides

Input: arr[] = {10, 9, 0, 5}

Output: 5

Explanation : We trap 0 + 0 + 5 + 0 = 5

**class HelloWorld {**

**public int rainwater(int[] arr) {**

**int left = 0, right = arr.length - 1;**

**int leftMax = 0, rightMax = 0;**

**int rain = 0;**

**while (left < right) {**

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

**if (arr[left] >= leftMax) {**

**leftMax = arr[left];**

**} else {**

**rain += leftMax - arr[left];**

**}**

**left++;**

**} else {**

**if (arr[right] >= rightMax) {**

**rightMax = arr[right];**

**} else {**

**rain += rightMax - arr[right];**

**}**

**right--;**

**}**

**}**

**return rain;**

**}**

**public static void main(String[] args) {**

**HelloWorld obj = new HelloWorld();**

**int[] arr1={3, 0, 1, 0, 4, 0, 2};**

**int[] arr2 = {3, 0, 2, 0, 4};**

**int[] arr3={1, 2, 3, 4} ;**

**int[] arr4= {1, 5, 4, 3} ;**

**int[] arr5={3, 1, 2, 4, 5};**

**System.out.println(obj.rainwater(arr1));**

**System.out.println(obj.rainwater(arr2));**

**System.out.println(obj.rainwater(arr3));**

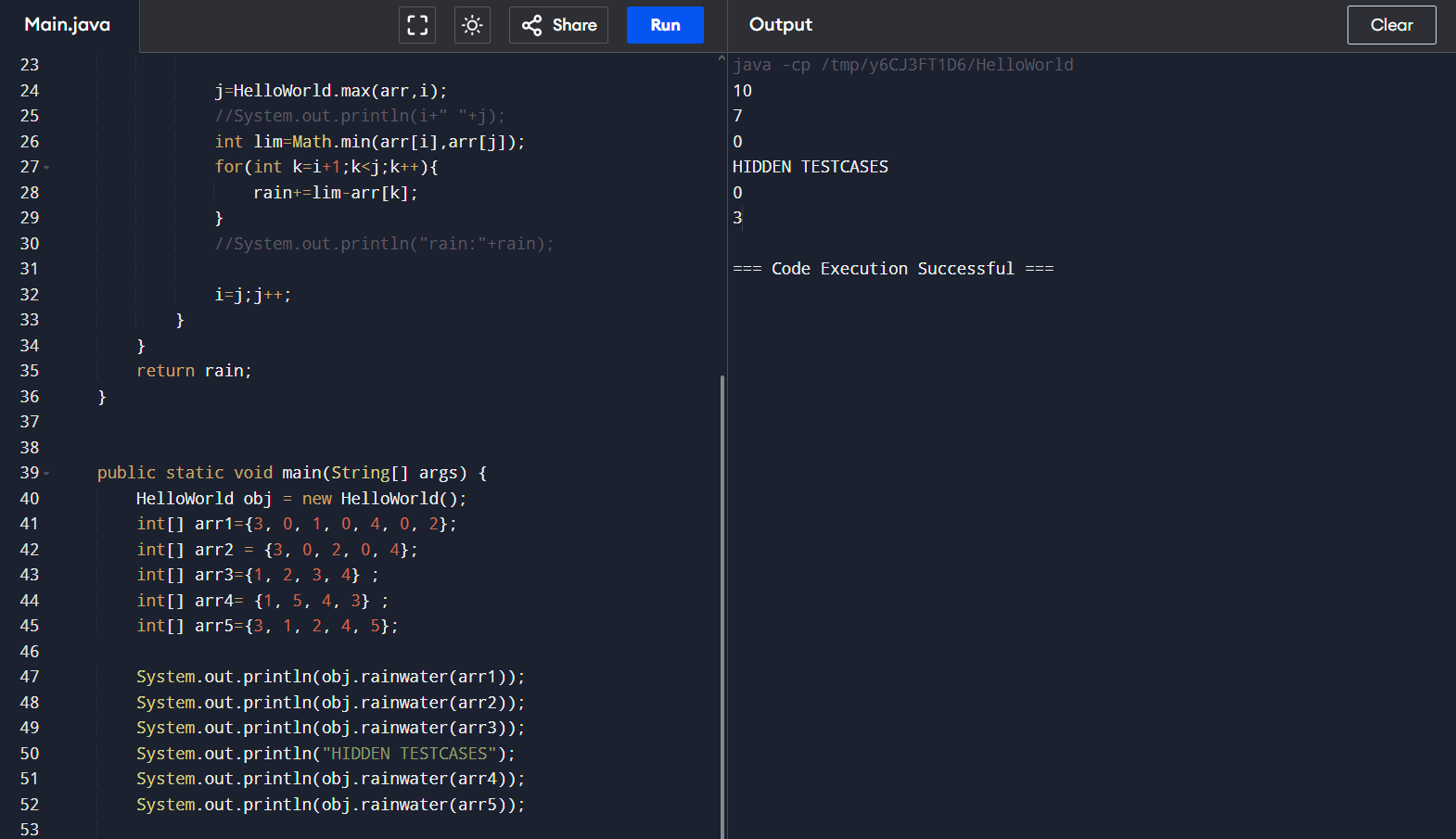
**System.out.println("HIDDEN TESTCASES");**

**System.out.println(obj.rainwater(arr4));**

**System.out.println(obj.rainwater(arr5));**

**}**

**}**



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

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.

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3

Output: 2

Explanation: If we distribute chocolate packets {3, 2, 4}, we will get the minimum difference,

that is 2.

Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 5

Output: 7

Explanation: If we distribute chocolate packets {3, 2, 4, 9, 7}, we will get the minimum

difference, that is 9 – 2 = 7.

**import java.util.Arrays;**

**class HelloWorld {**

**public int minSubArray(int[] arr,int m){**

**if (m > arr.length) return -1;**

**Arrays.sort(arr);**

**int curr=0,mindiff=Integer.MAX\_VALUE,i=arr.length-m;**

**while(i>=0){**

**curr=Math.abs(arr[i]-arr[i+m-1]);**

**//System.out.println(arr[i]+" "+arr[i+m-1]);**

**mindiff=Math.min(mindiff,curr);**

**i--;**

**}return mindiff;**

**}**

**public static void main(String[] args) {**

**HelloWorld obj=new HelloWorld();**

**int[] arr1={7, 3, 2, 4, 9, 12, 56};**

**int m1=3;**

**int m2=5;**

**int m3=7;**

**int m4=1;**

**System.out.println(obj.minSubArray(arr1,m1));**

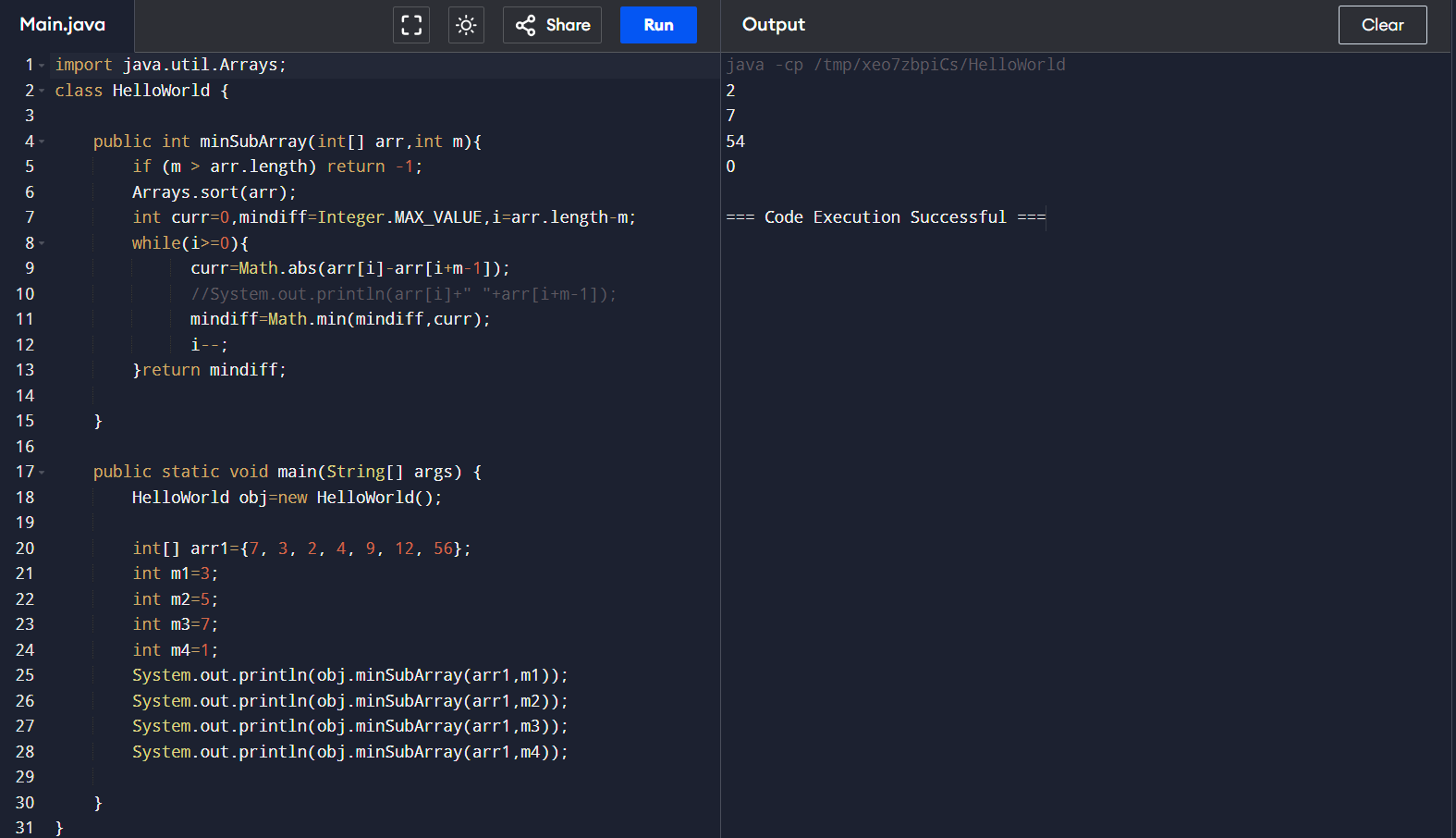
**System.out.println(obj.minSubArray(arr1,m2));**

**System.out.println(obj.minSubArray(arr1,m3));**

**System.out.println(obj.minSubArray(arr1,m4));**

**}**

**}**



**Time Complexity: O(n) , were n=arr.length**

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.

Input: arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]]

Output: [[1, 4], [6, 8], [9, 10]]

Explanation: In the given intervals, we have only two overlapping intervals [1, 3] and [2, 4].

Therefore, we will merge these two and return [[1, 4}], [6, 8], [9, 10]].

Input: arr[] = [[7, 8], [1, 5], [2, 4], [4, 6]]

Output: [[1, 6], [7, 8]]

Explanation: We will merge the overlapping intervals [[1, 5], [2, 4], [4, 6]] into a single interval

[1, 6].

**import java.util.\*;**

**class IntervalMerger {**

**public static List<int[]> combineIntervals(int[][] timeIntervals) {**

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

**if (timeIntervals == null || timeIntervals.length == 0) {**

**return mergedIntervals;**

**}**

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

**mergedIntervals.add(timeIntervals[0]);**

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

**int[] current = timeIntervals[i];**

**int[] lastMerged = mergedIntervals.get(mergedIntervals.size() - 1);**

**if (current[0] <= lastMerged[1]) {**

**lastMerged[1] = Math.max(lastMerged[1], current[1]);**

**} else {**

**mergedIntervals.add(current);**

**}**

**}**

**return mergedIntervals;**

**}**

**public static void main(String[] args) {**

**int[][] intervals1 = {{1, 3}, {2, 4}, {6, 8}, {9, 10}};**

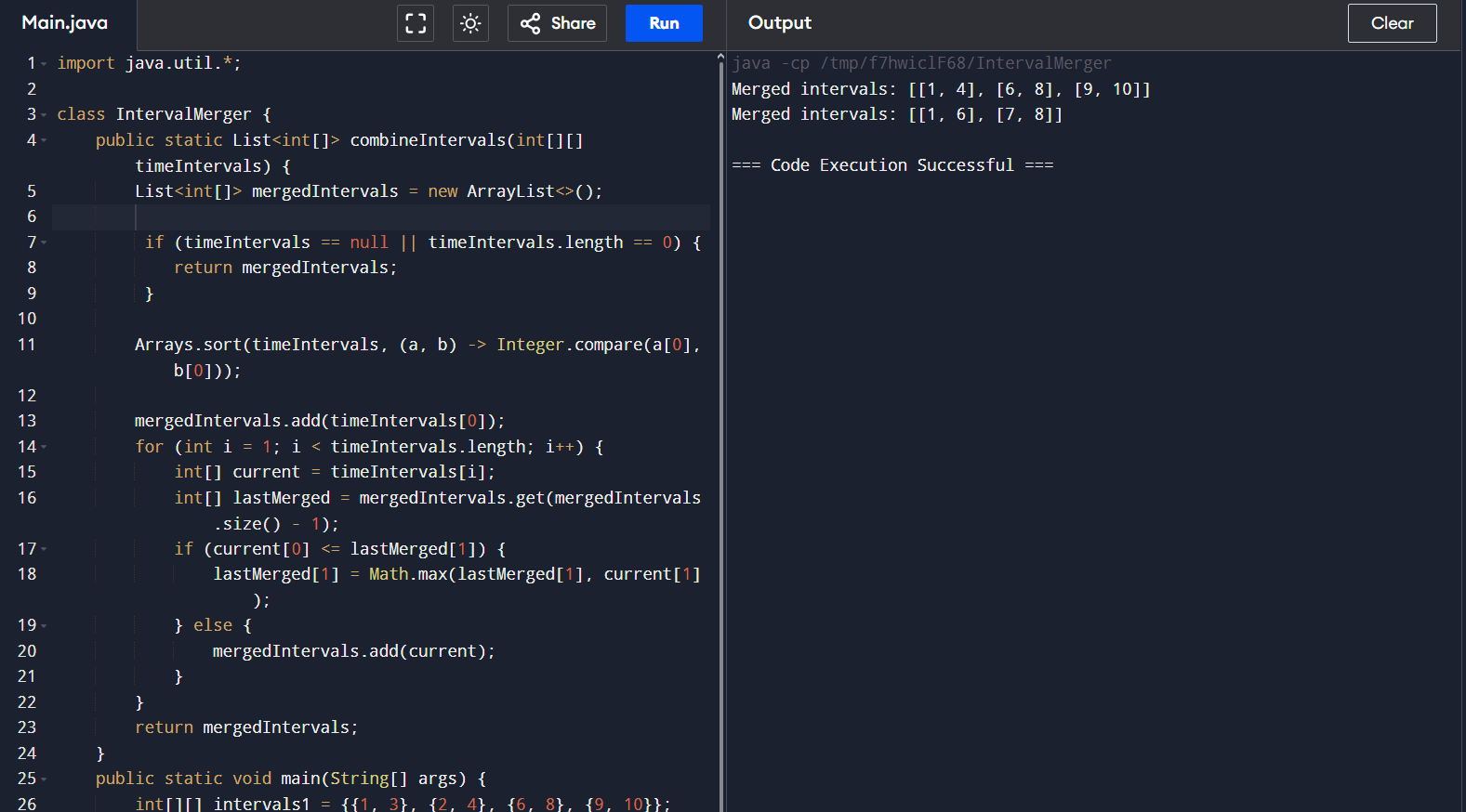
**System.out.println("Merged intervals: " + Arrays.deepToString(combineIntervals(intervals1).toArray()));**

**int[][] intervals2 = {{7, 8}, {1, 5}, {2, 4}, {4, 6}};**

**System.out.println("Merged intervals: " + Arrays.deepToString(combineIntervals(intervals2).toArray()));**

**}**

**}**



**Time Complexity: O(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}}

Output: {{1, 1}

{1, 0}}

Input: {{0, 0, 0},

{0, 0, 1}}

Output: {{0, 0, 1},

{1, 1, 1}}

Input: {{1, 0, 0, 1},

{0, 0, 1, 0},

{0, 0, 0, 0}}

Output: {{1, 1, 1, 1},

{1, 1, 1, 1},

{1, 0, 1, 1}}

**import java.util.\*;**

**class HelloWorld {**

**public String boolmat(int[][] arr){**

**int[] a=new int[arr[0].length];**

**int[] b=new int[arr.length];**

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

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

**if(arr[i][j]==1){**

**a[j]=1;b[i]=1;**

**}**

**}**

**}**

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

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

**if( b[i]==1 || a[j]==1) arr[i][j]=1;**

**}**

**}**

**return Arrays.deepToString(arr);**

**}**

**public static void main(String[] args) {**

**int[][] arr1={{1, 0},{0, 0}};**

**int[][] arr2={{0, 0, 0},{0, 0, 1}};**

**int[][] arr3= {{1, 0, 0, 1},**

**{0, 0, 1, 0},**

**{0, 0, 0, 0}};**

**HelloWorld obj=new HelloWorld();**

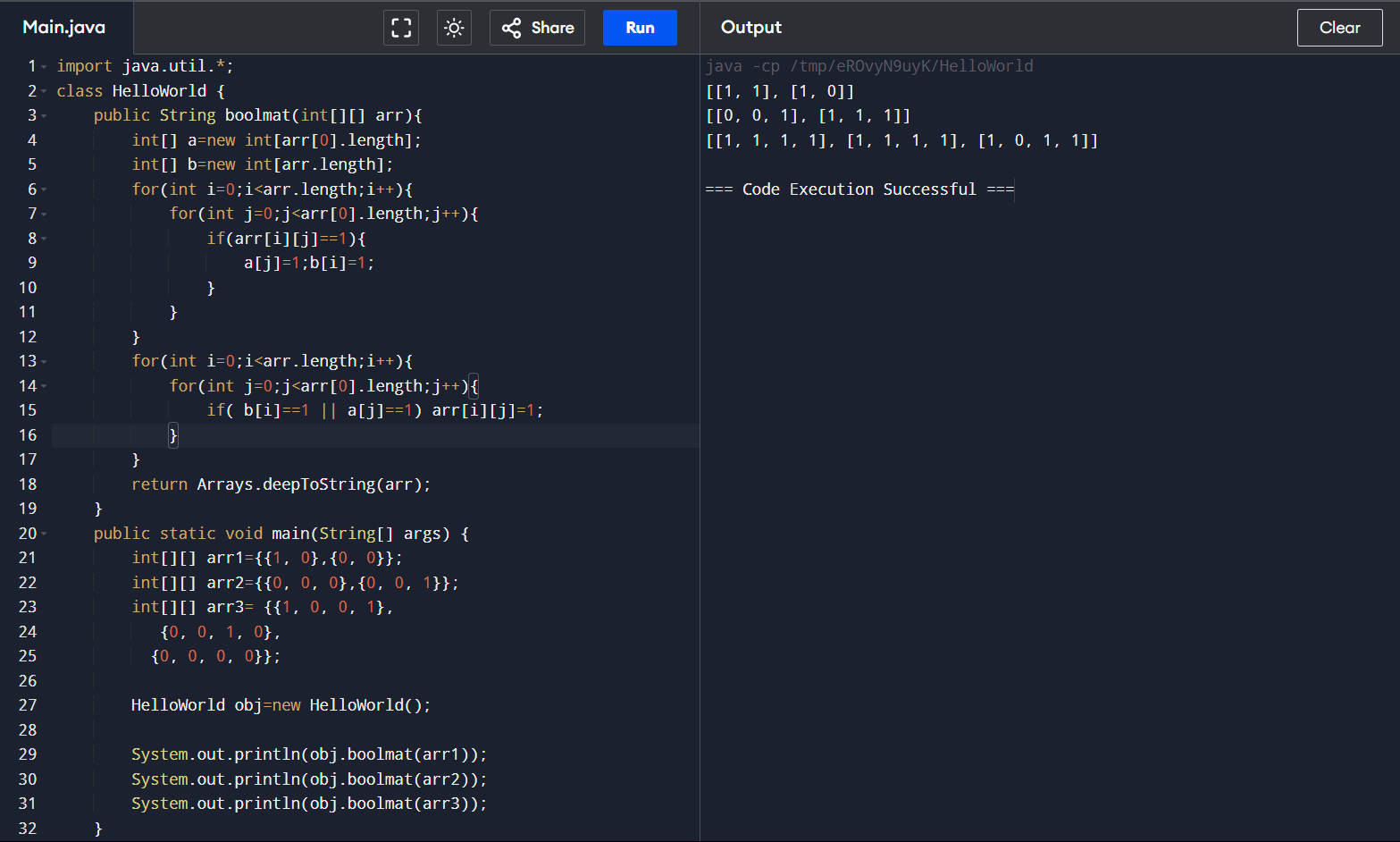
**System.out.println(obj.boolmat(arr1));**

**System.out.println(obj.boolmat(arr2));**

**System.out.println(obj.boolmat(arr3));**

**}**

**}**



**Time Complexity : O (n\*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.

Input: matrix = {{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16 }}

Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Input: matrix = { {1, 2, 3, 4, 5, 6},

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

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

Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11

Explanation: The output is matrix in spiral format.

**import java.util.\*;**

**class HelloWorld {**

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

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

**if (matrix == null || matrix.length == 0) return result;**

**int top = 0, bottom = matrix.length - 1, left = 0, right = matrix[0].length - 1;while (top <= bottom && left <= right) {**

**for (int i = left; i <= right; i++) result.add(matrix[top][i]);**

**top++;**

**for (int i = top; i <= bottom; i++) result.add(matrix[i][right]);**

**right--;**

**if (top <= bottom) {**

**for (int i = right; i >= left; i--) result.add(matrix[bottom][i]);**

**bottom--;**

**}**

**if (left <= right) {**

**for (int i = bottom; i >= top; i--) result.add(matrix[i][left]);**

**left++;**

**}**

**}return result;**

**}**

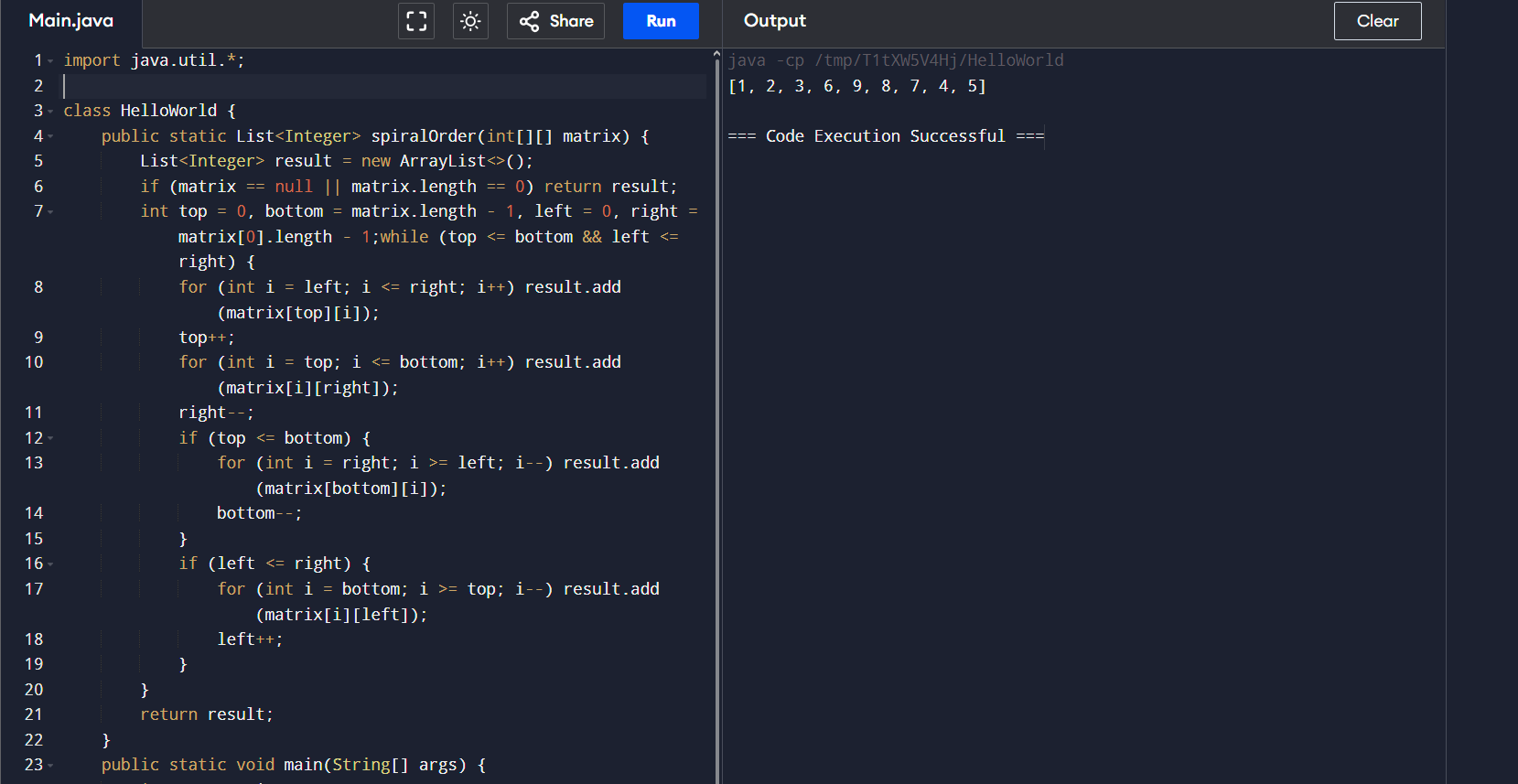
**public static void main(String[] args) {**

**int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};**

**List<Integer> result = spiralOrder(matrix);**

**System.out.println(result);**

**}**

**}**

**Time Complexity: O(n)**

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.

Input: str = “((()))()()”

Output: Balanced

Input: str = “())((())”

Output: Not Balanced

**import java.util.Arrays;**

**class HelloWorld {**

**public String parenthesis(String str){**

**int count=0;**

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

**if(str.charAt(i)=='(') count++;**

**else if(str.charAt(i)==')') count--;**

**if(count<0) return "Unbalanced";**

**}**

**return "Balanced";**

**}**

**public static void main(String[] args) {**

**HelloWorld obj=new HelloWorld();**

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

**String str2="())((())";**

**String str3="(())))(";**

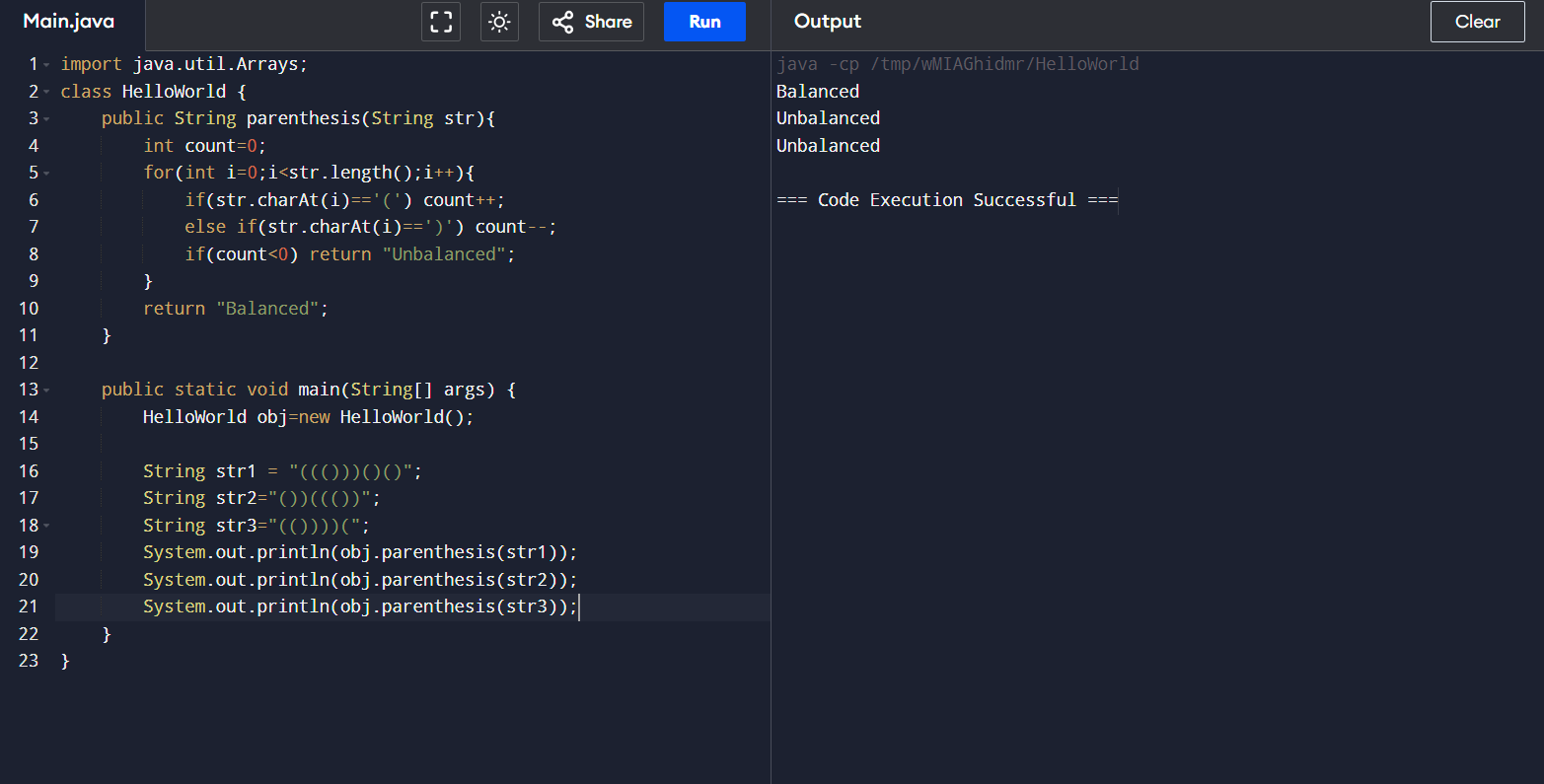
**System.out.println(obj.parenthesis(str1));**

**System.out.println(obj.parenthesis(str2));**

**System.out.println(obj.parenthesis(str3));**

**}**

**}**



**Time complexity: O(n), were n=str.length()**

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.

Input: s1 = “geeks” s2 = “kseeg”

Output: true

Explanation: Both the string have same characters with same frequency. So, they are anagrams.

Input: s1 = “allergy” s2 = “allergic”

Output: false

Explanation: Characters in both the strings are not same. s1 has extra character „y‟ and s2 has

extra characters „i‟ and „c‟, so they are not anagrams.

Input: s1 = “g”, s2 = “g”

Output: true

Explanation: Characters in both the strings are same, so they are anagrams**.**

**import java.util.Arrays;**

**class HelloWorld {**

**public boolean anagrams(String str1,String str2){**

**if (str1.length() != str2.length()) return false;**

**char[] a=str1.toCharArray();**

**char[] b=str2.toCharArray();**

**Arrays.sort(a);**

**Arrays.sort(b);**

**return Arrays.equals(a,b);**

**}**

**public static void main(String[] args) {**

**HelloWorld obj=new HelloWorld();**

**String s1 = "geeks",s2 = "kseeg";**

**String s3 = "allergy" , s4 = "allergic";**

**String s5="g",s6="g";**

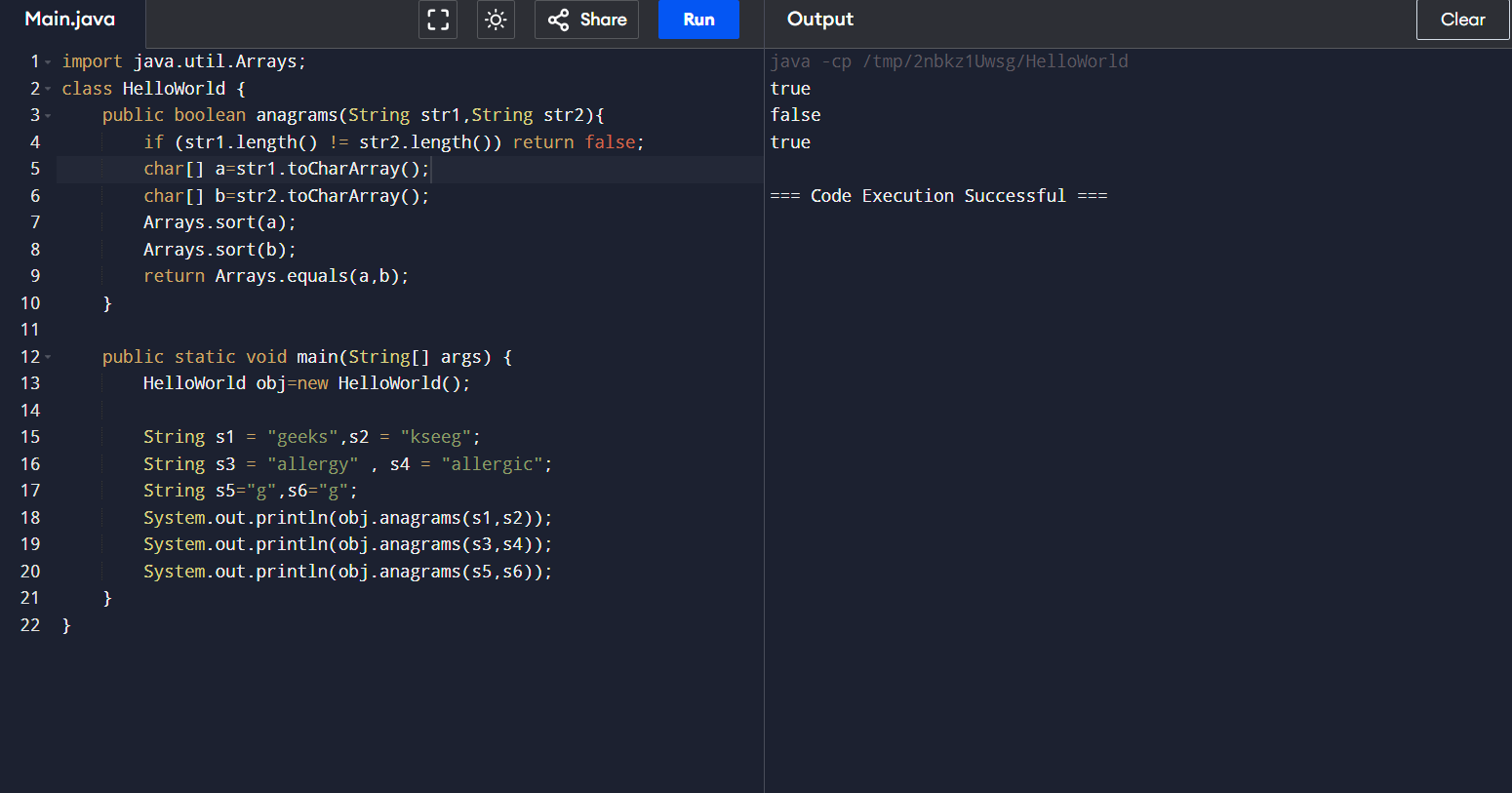
**System.out.println(obj.anagrams(s1,s2));**

**System.out.println(obj.anagrams(s3,s4));**

**System.out.println(obj.anagrams(s5,s6));**

**}**

**}**



**Time complexity: O(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.

Input: str = “forgeeksskeegfor”

Output: “geeksskeeg”

Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc.

But the substring “geeksskeeg” is the longest among all.

Input: str = “Geeks”

Output: “ee”

Input: str = “abc”

Output: “a”

Input: str = “”

Output: “”

**import java.util.\*;**

**class HelloWorld {**

**public String longestPalindrome(String str){**

**if (str == null || str.length() < 2) return str;**

**int maxLength = 1;**

**int start = 0;**

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

**st.push(0);**

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

**if (!st.isEmpty() && str.charAt(st.peek()) == str.charAt(i)) {**

**st.pop();**

**if (st.isEmpty()) {**

**start = 0;**

**maxLength = i + 1;**

**} else {**

**int length = i - st.peek();**

**if (length > maxLength) {**

**start = st.peek() + 1;**

**maxLength = length;**

**}**

**}**

**} else {**

**st.push(i);**

**}**

**}**

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

**}**

**public static void main(String[] args) {**

**HelloWorld obj=new HelloWorld();**

**String s1 = "Geeks";**

**String s2 = "abc" ;**

**String s3="";**

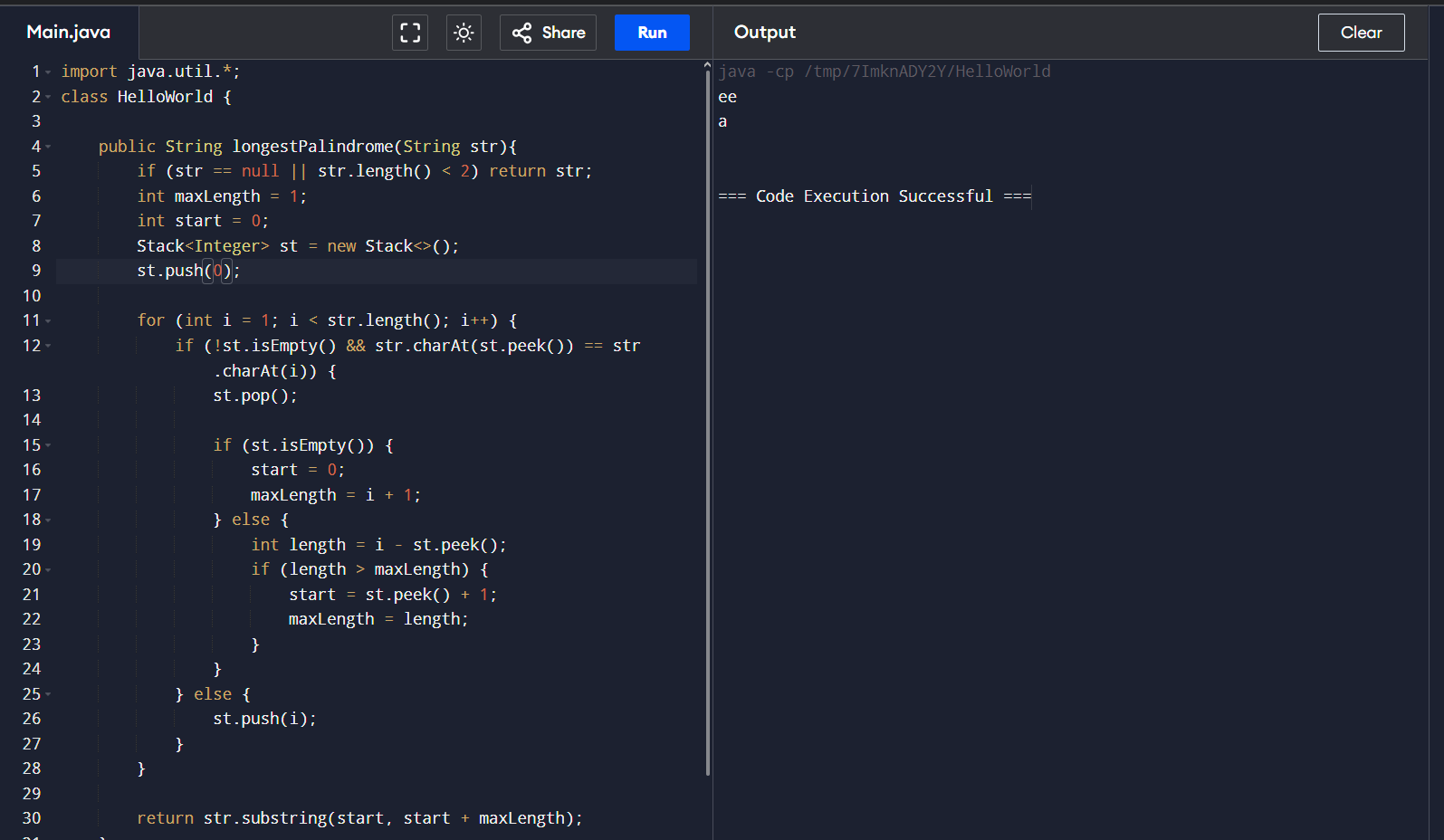
**System.out.println(obj.longestPalindrome(s1));**

**System.out.println(obj.longestPalindrome(s2));**

**System.out.println(obj.longestPalindrome(s3));**

**}**

**}**



**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”.

Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”]

Output: gee

Explanation: “gee” is the longest common prefix in all the given strings.

Input: arr[] = [“hello”, “world”]

Output: -1

Explanation: There‟s no common prefix in the given strings.

**import java.util.\*;**

**class HelloWorld {**

**public String commonPrefix(String[] arr){**

**if(arr==null || arr.length==0) return "-1";**

**int len=arr.length-1;**

**Arrays.sort(arr);**

**int minlen=Math.min(arr[0].length(),arr[len].length());**

**int i=0;**

**while(i<minlen && arr[0].charAt(i)==arr[len].charAt(i)){**

**i++;**

**}**

**if (i == 0) return "-1";**

**return arr[0].substring(0,i);**

**}**

**public static void main(String[] args) {**

**HelloWorld obj=new HelloWorld();**

**String[] s1 = {"geeksforgeeks", "geeks", "geek", "geezer"};**

**String[] s2 = {"hello", "world"} ;**

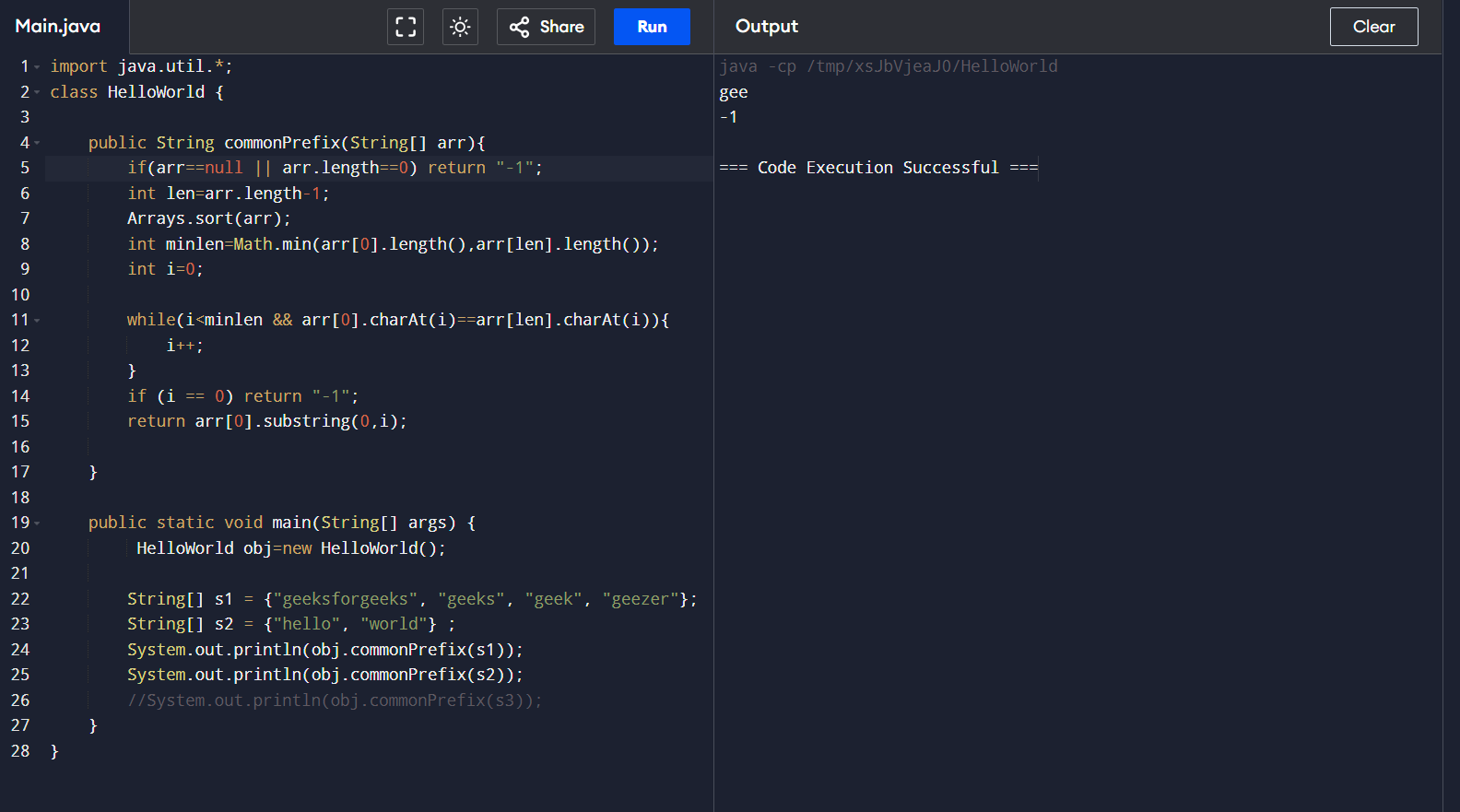
**System.out.println(obj.commonPrefix(s1));**

**System.out.println(obj.commonPrefix(s2));**

**//System.out.println(obj.commonPrefix(s3));**

**}**

**}**



**Time Complexity : O(n log n + m), sorting and m elements**

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.

Input : Stack[] = [1, 2, 3, 4, 5]

Output : Stack[] = [1, 2, 4, 5]

Input : Stack[] = [1, 2, 3, 4, 5, 6]

Output : Stack[] = [1, 2, 4, 5, 6]

**import java.util.Stack;**

**class HelloWorld {**

**public void midremove(Stack<Integer> stack) {**

**int n = stack.size();**

**if (n == 0) return;**

**int middleIndex = n / 2;**

**pushrem(stack, middleIndex);**

**}**

**private void pushrem(Stack<Integer> stack, int index) {**

**if (index == 0) {**

**stack.pop();**

**return ;**

**}**

**int temp = stack.pop();**

**pushrem(stack, index - 1);**

**stack.push(temp);**

**}**

**public static void main(String[] args) {**

**HelloWorld operations = new HelloWorld();**

**Stack<Integer> stack1 = new Stack<>();**

**stack1.push(1);**

**stack1.push(2);**

**stack1.push(3);**

**stack1.push(4);**

**stack1.push(5);**

**System.out.println("Before Stack: " + stack1);**

**operations.midremove(stack1);**

**System.out.println("After removal" + stack1);**

**Stack<Integer> stack2 = new Stack<>();**

**stack2.push(1);**

**stack2.push(2);**

**stack2.push(3);**

**stack2.push(4);**

**stack2.push(5);**

**stack2.push(6);**

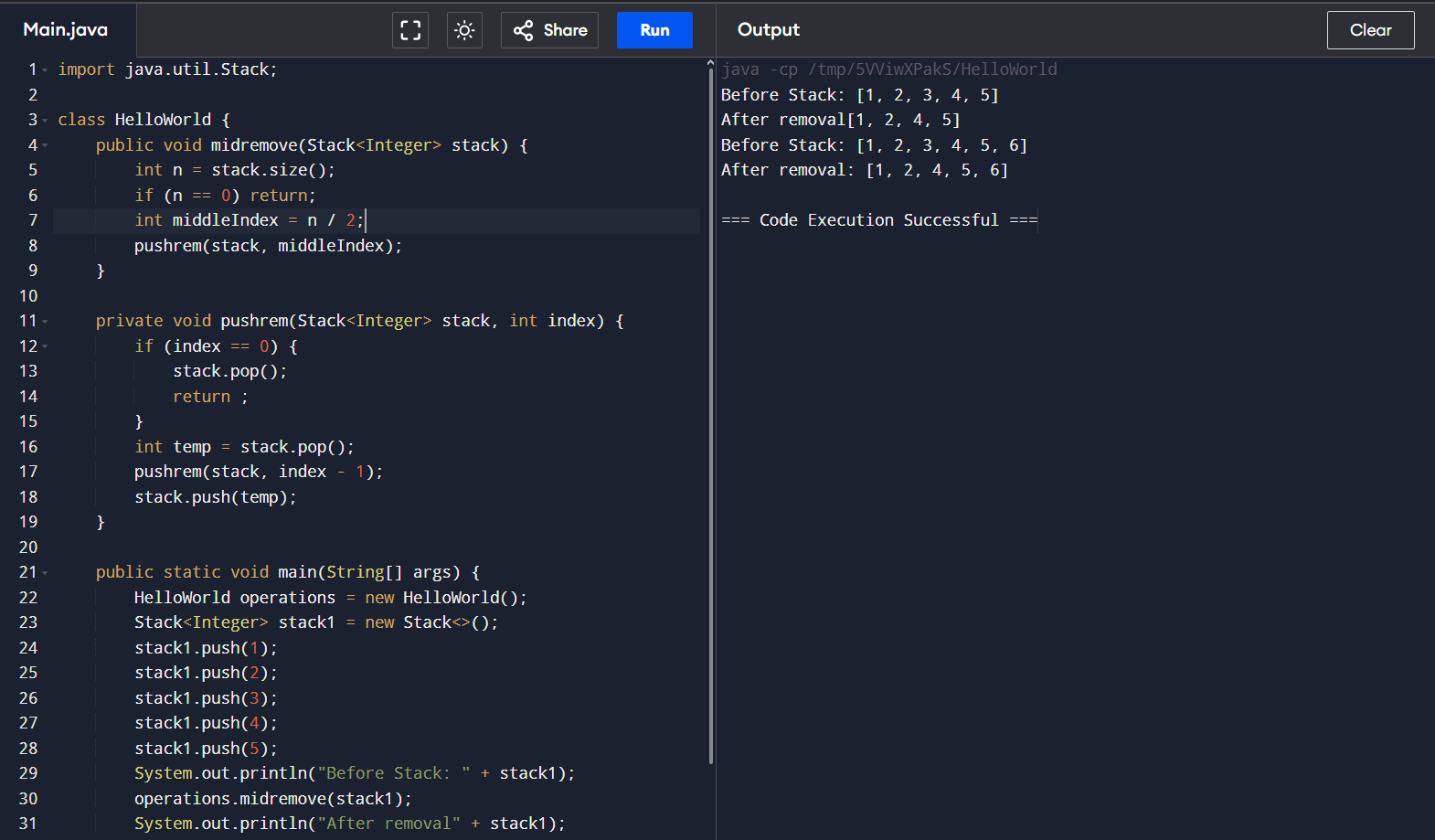
**System.out.println("Before Stack: " + stack2);**

**operations.midremove(stack2);**

**System.out.println("After removal: " + stack2);**

**}**

**}**



**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 ]

Output: 4

5

2 –>

5 –> 25 –> 25

25 –> -1

Explanation: Except 25 every element has an element greater than them present on the right side

Input: arr[] = [ 13 , 7, 6 , 12 ]

Output: 13 –>

7 -1 –> 12

6

12 –> 12 –> -1

Explanation: 13 and 12 don‟t have any element greater than them present on the right side

**class HelloWorld {**

**public void greaterNum(int[] arr) {**

**int n = arr.length;**

**int[] b = new int[n];**

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

**b[i] = -1;**

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

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

**b[i] = arr[j];**

**break;**

**}**

**}**

**}**

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

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

**}**

**return;**

**}**

**public static void main(String[] args) {**

**HelloWorld obj = new HelloWorld();**

**int[] arr1={4 , 5 , 2 , 25 };**

**int[] arr2={13 , 7, 6 , 12};**

**System.out.println("Output 1:");**

**obj.greaterNum(arr1);**

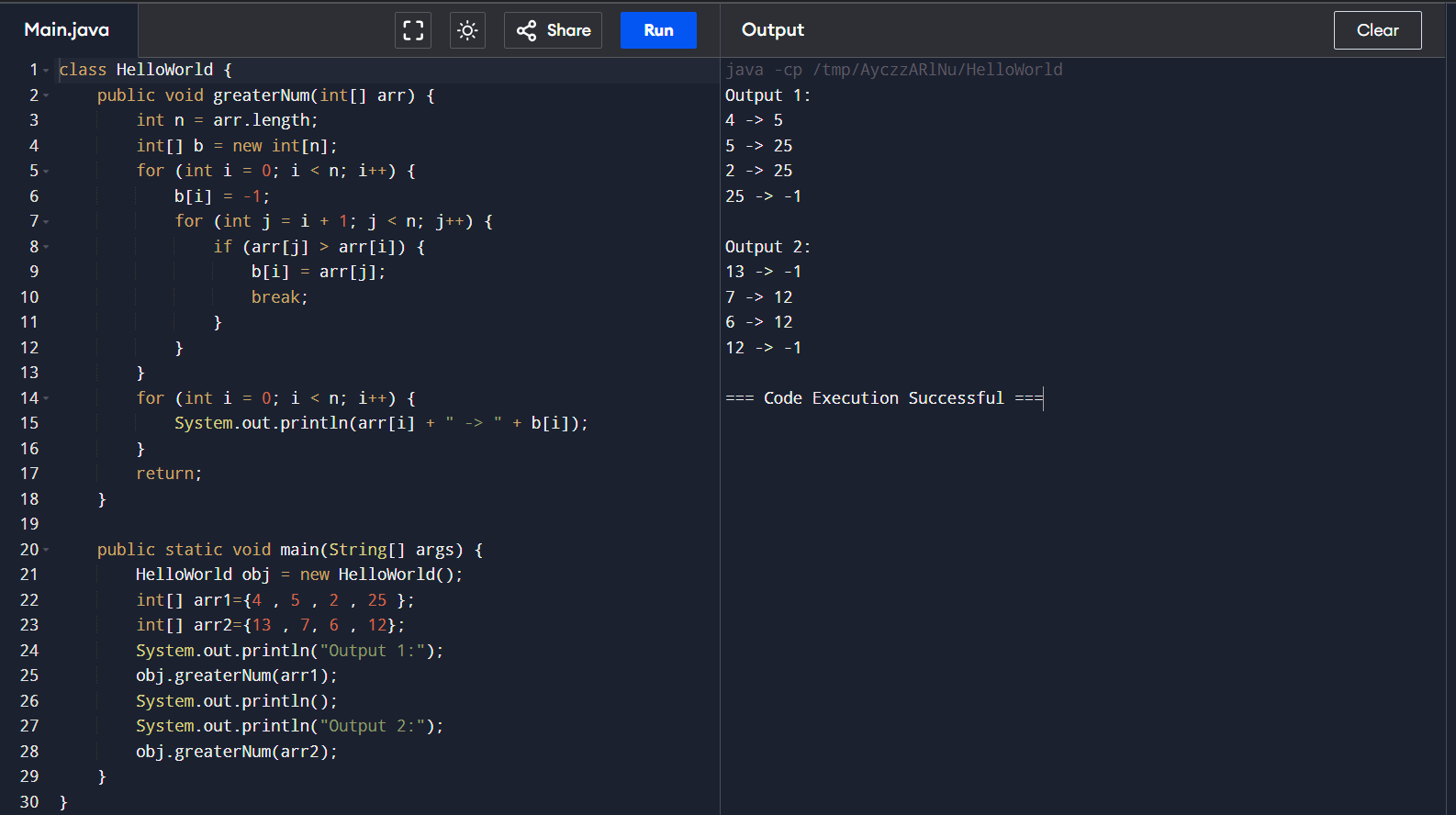
**System.out.println();**

**System.out.println("Output 2:");**

**obj.greaterNum(arr2);**

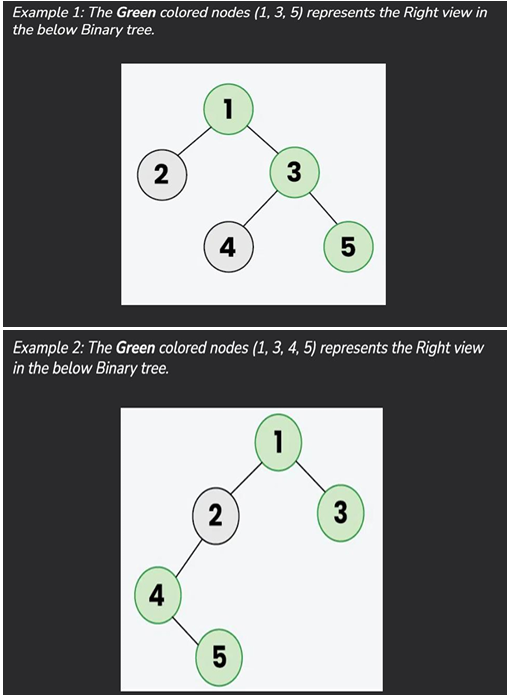
**}**

**}**



**Time Complexity: O(N\*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 TreeNode {**

**int val;**

**TreeNode left, right;**

**TreeNode(int x) {**

**val = x;**

**left = right = null;**

**}**

**}**

**class RightViewBinaryTree {**

**private List<Integer> computeRightView(TreeNode node, List<Integer> result, int depth) {**

**if (node == null) {**

**return result;**

**}**

**if (depth == result.size()) {**

**result.add(node.val);**

**}**

**computeRightView(node.right, result, depth + 1);**

**computeRightView(node.left, result, depth + 1);**

**return result;**

**}**

**public static void main(String[] args) {**

**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);**

**root.right.right = new TreeNode(6);**

**root.left.left.left = new TreeNode(7);**

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

**RightViewBinaryTree solution = new RightViewBinaryTree();**

**List<Integer> rightViewList = solution.computeRightView(root, result, 0);**

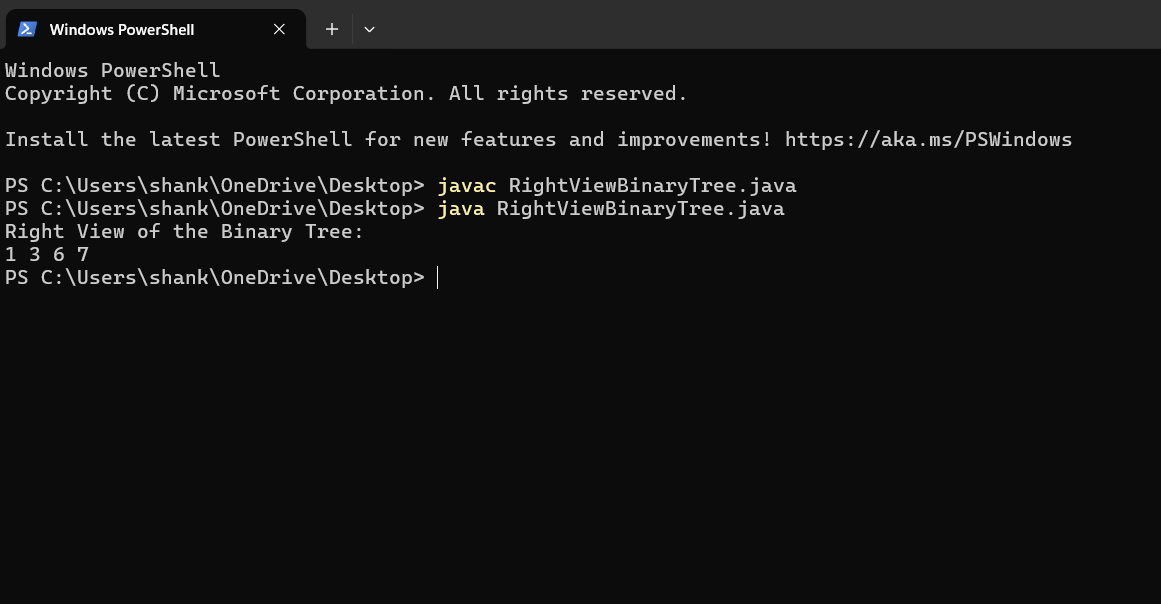
**System.out.println("Right View of the Binary Tree:");**

**for (int val : rightViewList) {**

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

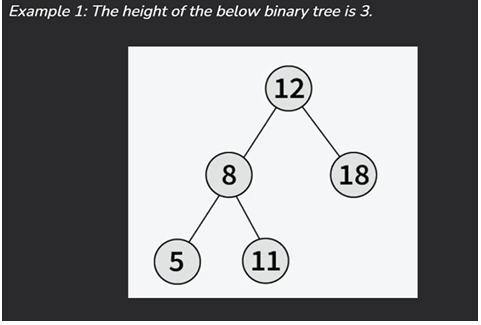
**}**

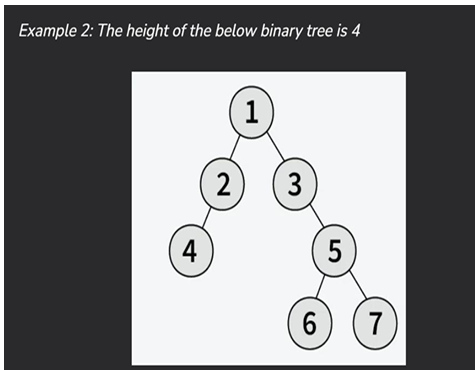
**}**

**}** 

**Time Complexity: O(N)**

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.





**class TreeNode {**

**int val;**

**TreeNode left;**

**TreeNode right;**

**TreeNode(int x) {**

**val = x;**

**left = null;**

**right = null;**

**}**

**}**

**class depth\_BT{**

**public int maxDepth(TreeNode root) {**

**return root == null ? 0 : 1 + Math.max(maxDepth(root.left), maxDepth(root.right));**

**}**

**public static void main(String[] args) {**

**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);**

**root.right.right = new TreeNode(6);**

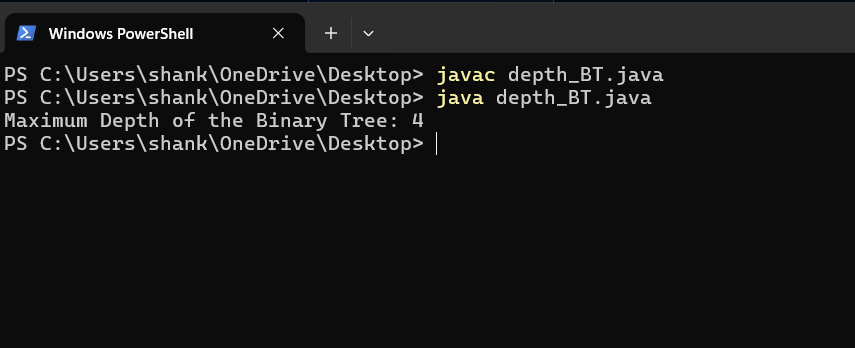
**root.left.left.left = new TreeNode(7);**

**depth\_BT solution = new depth\_BT();**

**System.out.println("Maximum Depth of the Binary Tree: " + solution.maxDepth(root));**

**}**

**}**

****

**Time Complexity: O(n)**