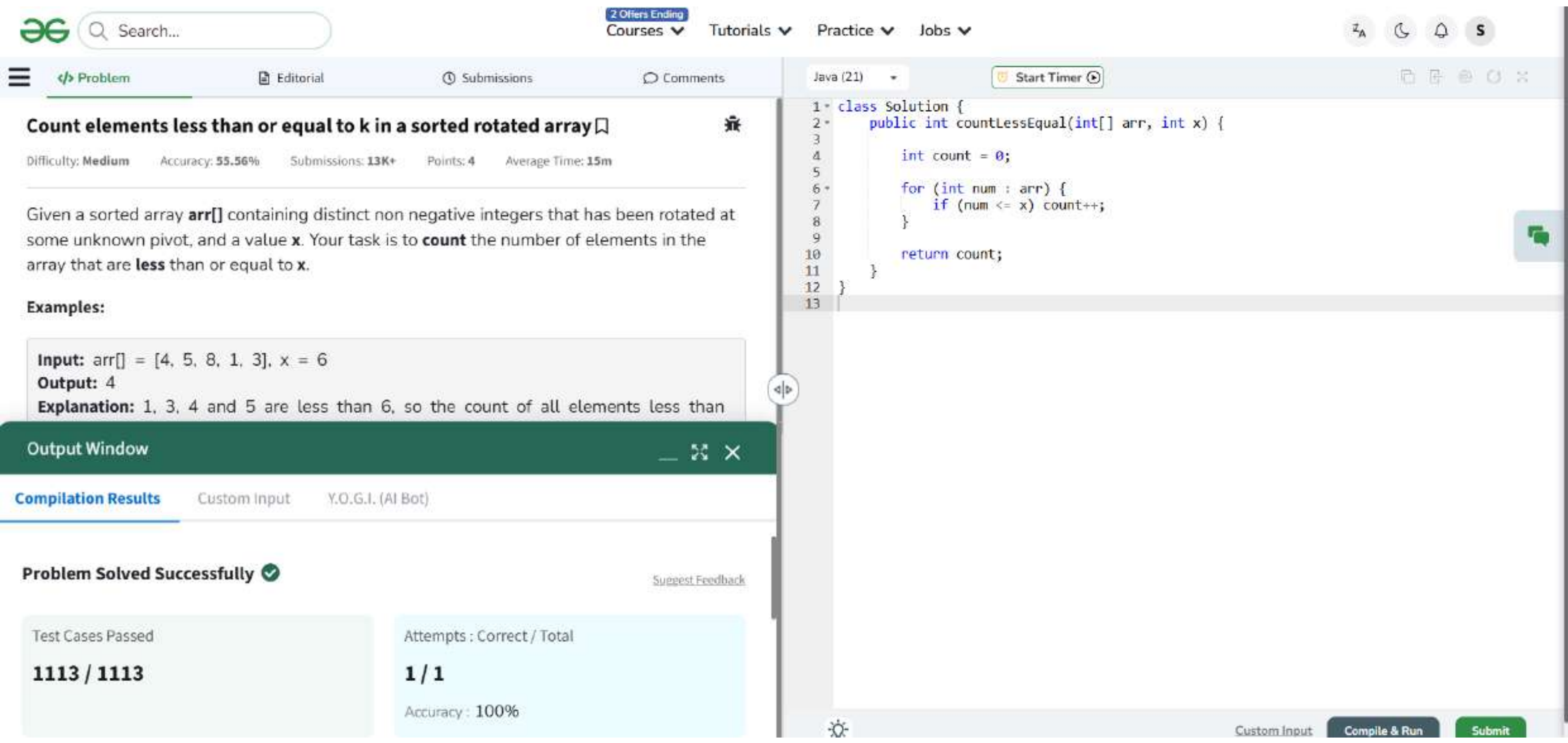


Count elements less than or equal to k in a sorted rotated array

Difficulty: **Medium** Accuracy: **55.56%** Submissions: **13K+** Points: **4** Average Time: **15m**

Given a sorted array **arr[]** containing distinct non negative integers that has been rotated at some unknown pivot, and a value **x**. Your task is to **count** the number of elements in the array that are **less** than or equal to **x**.

Examples:

Input: arr[] = [4, 5, 8, 1, 3], x = 6
Output: 4
Explanation: 1, 3, 4 and 5 are less than 6, so the count of all elements less than

Output Window

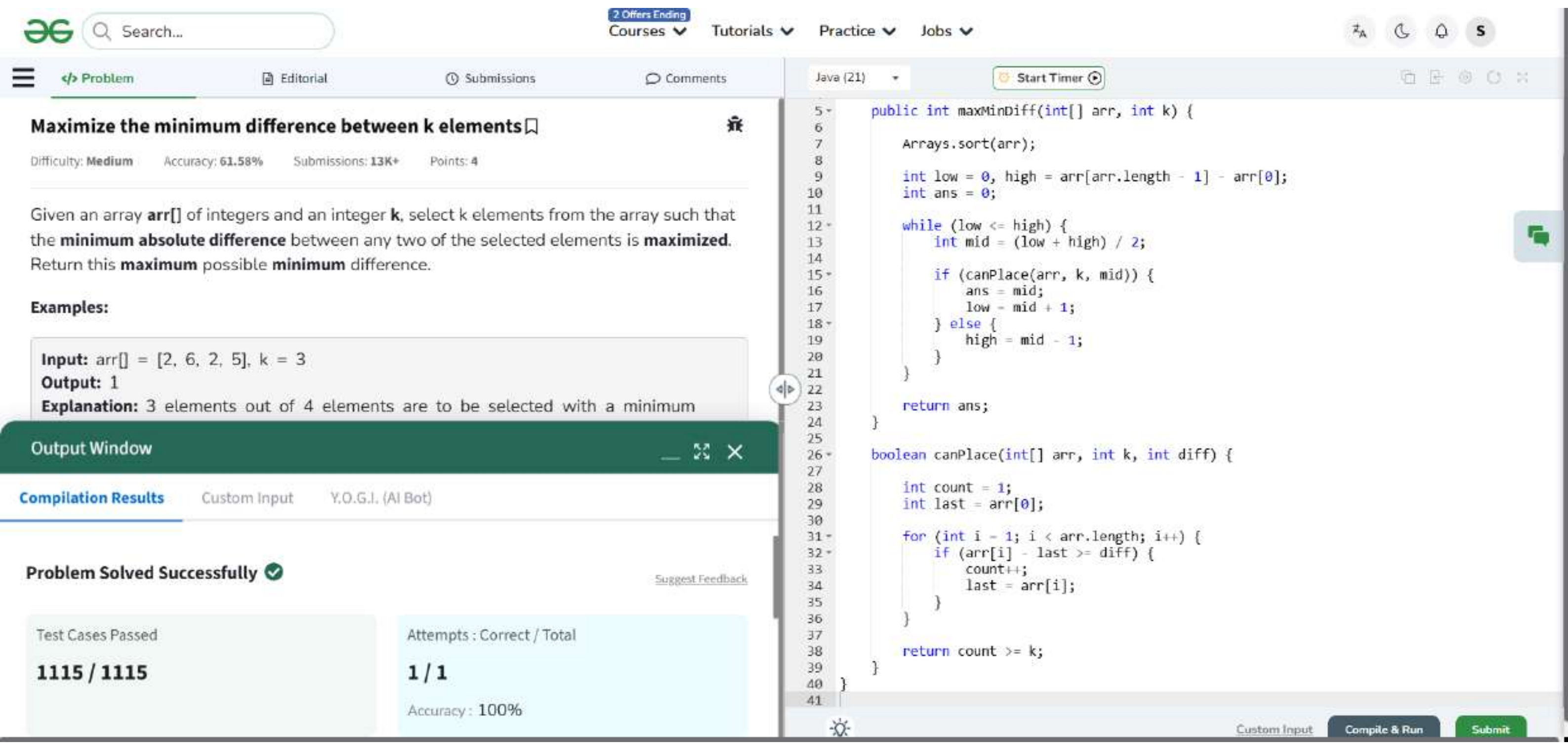
Compilation Results | Custom Input | Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed	Attempts : Correct / Total
1113 / 1113	**1 / 1**
	Accuracy : 100%

```java
class Solution {
    public int countLessEqual(int[] arr, int x) {

        int count = 0;

        for (int num : arr) {
            if (num <= x) count++;
        }

        return count;
    }
}
```



Maximize the minimum difference between k elements

Difficulty: **Medium** Accuracy: **61.58%** Submissions: **13K+** Points: **4**

Given an array **arr[]** of integers and an integer **k**, select k elements from the array such that the **minimum absolute difference** between any two of the selected elements is **maximized**. Return this **maximum** possible **minimum** difference.

Examples:

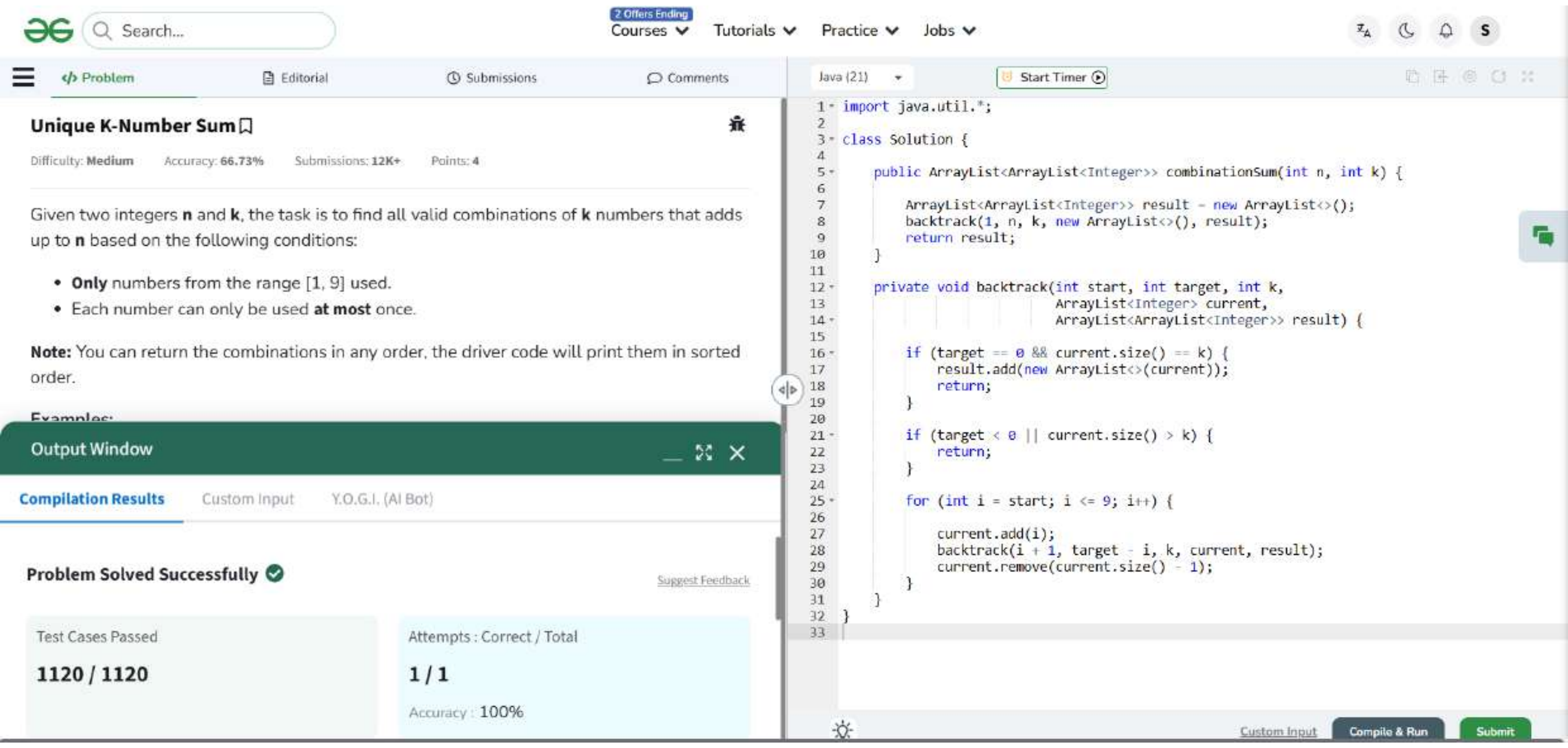
Input: arr[] = [2, 6, 2, 5], k = 3
Output: 1
Explanation: 3 elements out of 4 elements are to be selected with a minimum

Problem Solved Successfully

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
1 / 1
Accuracy : 100%

```java
    public int maxMinDiff(int[] arr, int k) {

        Arrays.sort(arr);

        int low = 0, high = arr[arr.length - 1] - arr[0];
        int ans = 0;

        while (low <= high) {
            int mid = (low + high) / 2;

            if (canPlace(arr, k, mid)) {
                ans = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        return ans;
    }

    boolean canPlace(int[] arr, int k, int diff) {

        int count = 1;
        int last = arr[0];

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] - last >= diff) {
                count++;
                last = arr[i];
            }
        }

        return count >= k;
    }
}
```



Unique K-Number Sum

Difficulty: **Medium** Accuracy: **66.73%** Submissions: **12K+** Points: **4**

Given two integers **n** and **k**, the task is to find all valid combinations of **k** numbers that adds up to **n** based on the following conditions:

- **Only** numbers from the range [1, 9] used.
- Each number can only be used **at most** once.

Note: You can return the combinations in any order, the driver code will print them in sorted order.

```java
import java.util.*;

class Solution {

    public ArrayList<ArrayList<Integer>> combinationSum(int n, int k) {

        ArrayList<ArrayList<Integer>> result = new ArrayList<>();
        backtrack(1, n, k, new ArrayList<>(), result);
        return result;
    }

    private void backtrack(int start, int target, int k,
                           ArrayList<Integer> current,
                           ArrayList<ArrayList<Integer>> result) {

        if (target == 0 && current.size() == k) {
            result.add(new ArrayList<>(current));
            return;
        }

        if (target < 0 || current.size() > k) {
            return;
        }

        for (int i = start; i <= 9; i++) {

            current.add(i);
            backtrack(i + 1, target - i, k, current, result);
            current.remove(current.size() - 1);
        }
    }
}
```

Output Window

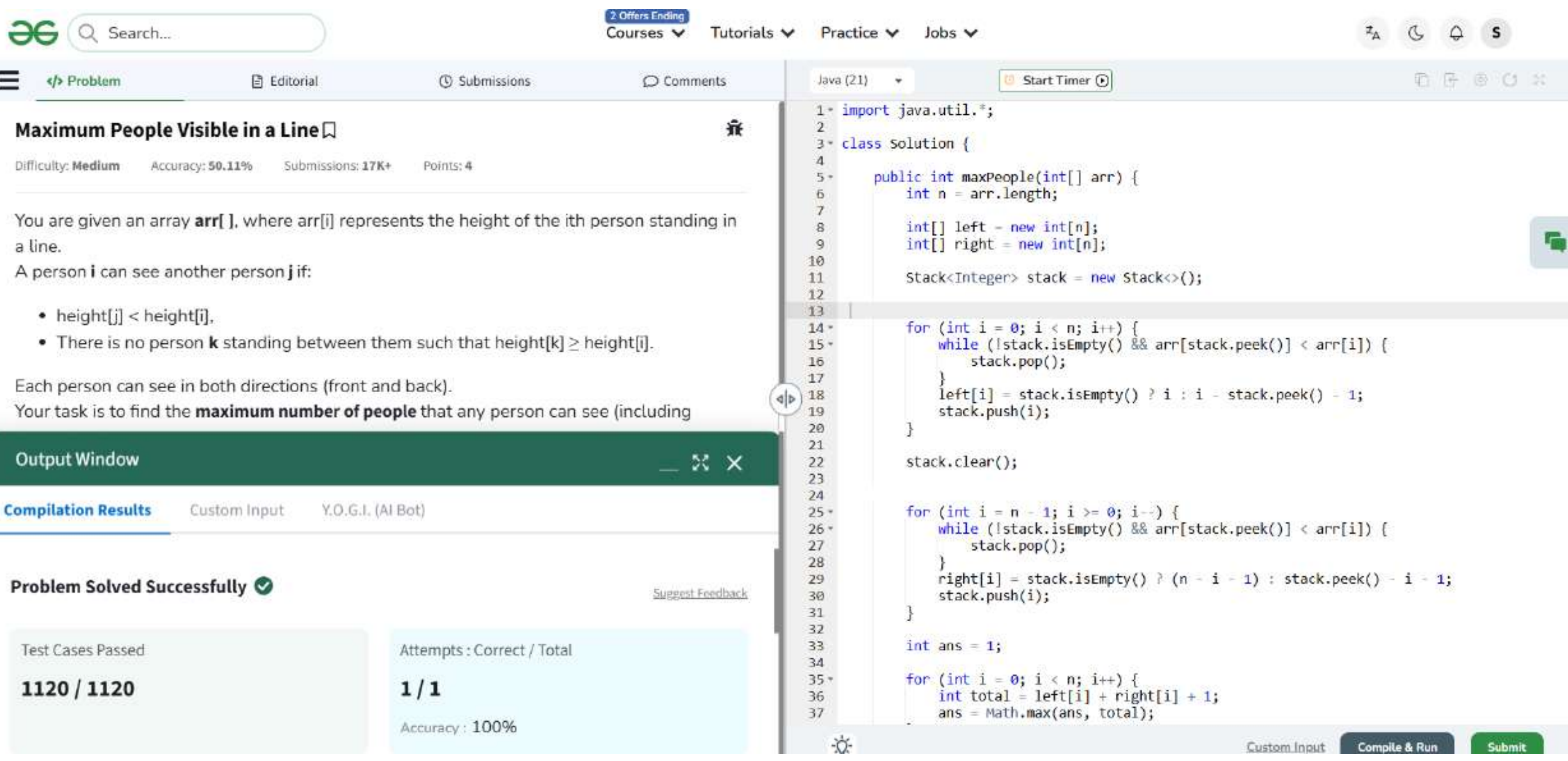
Compilation Results | Custom Input | Y.O.G.I. (AI Bot)

Problem Solved Successfully ✓

Suggest Feedback

Test Cases Passed
1120 / 1120

Attempts : Correct / Total
1 / 1
Accuracy : 100%



Maximum People Visible in a Line

Difficulty: **Medium** Accuracy: **50.11%** Submissions: **17K+** Points: **4**

You are given an array **arr[ ]**, where arr[i] represents the height of the ith person standing in a line.
A person **i** can see another person **j** if:

- height[j] < height[i],
- There is no person **k** standing between them such that height[k] ≥ height[i].

Each person can see in both directions (front and back).
Your task is to find the **maximum number of people** that any person can see (including

Output Window

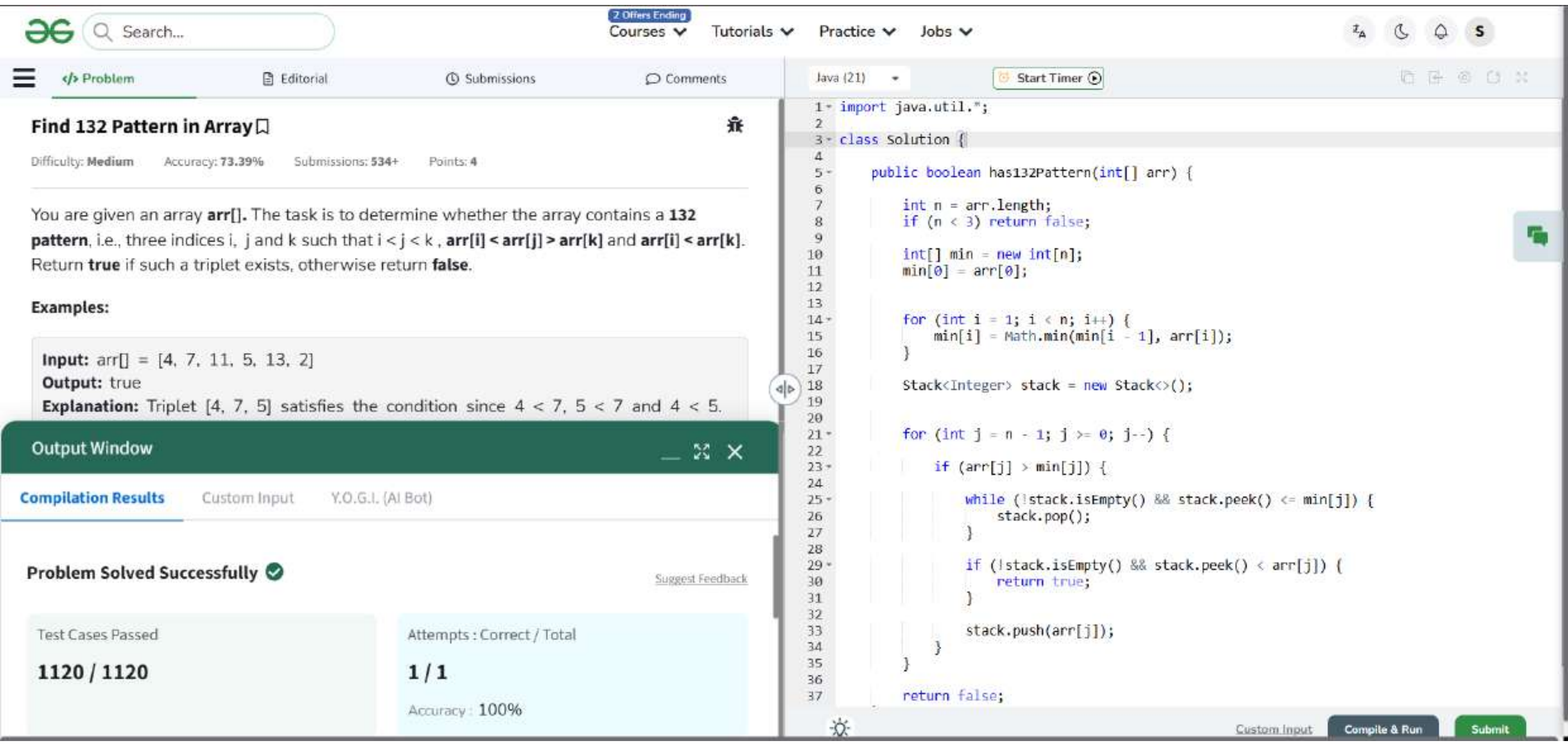
Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed	Attempts : Correct / Total
1120 / 1120	**1 / 1**
	Accuracy : 100%

```java
import java.util.*;

class Solution {

    public int maxPeople(int[] arr) {
        int n = arr.length;

        int[] left = new int[n];
        int[] right = new int[n];

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


        for (int i = 0; i < n; i++) {
            while (!stack.isEmpty() && arr[stack.peek()] < arr[i]) {
                stack.pop();
            }
            left[i] = stack.isEmpty() ? i : i - stack.peek() - 1;
            stack.push(i);
        }

        stack.clear();


        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && arr[stack.peek()] < arr[i]) {
                stack.pop();
            }
            right[i] = stack.isEmpty() ? (n - i - 1) : stack.peek() - i - 1;
            stack.push(i);
        }

        int ans = 1;

        for (int i = 0; i < n; i++) {
            int total = left[i] + right[i] + 1;
            ans = Math.max(ans, total);
```



Find 132 Pattern in Array

Difficulty: **Medium** Accuracy: **73.39%** Submissions: **534+** Points: **4**

You are given an array **arr[].** The task is to determine whether the array contains a **132 pattern**, i.e., three indices i, j and k such that i < j < k , **arr[i] < arr[j] > arr[k]** and **arr[i] < arr[k]**. Return **true** if such a triplet exists, otherwise return **false**.

Examples:

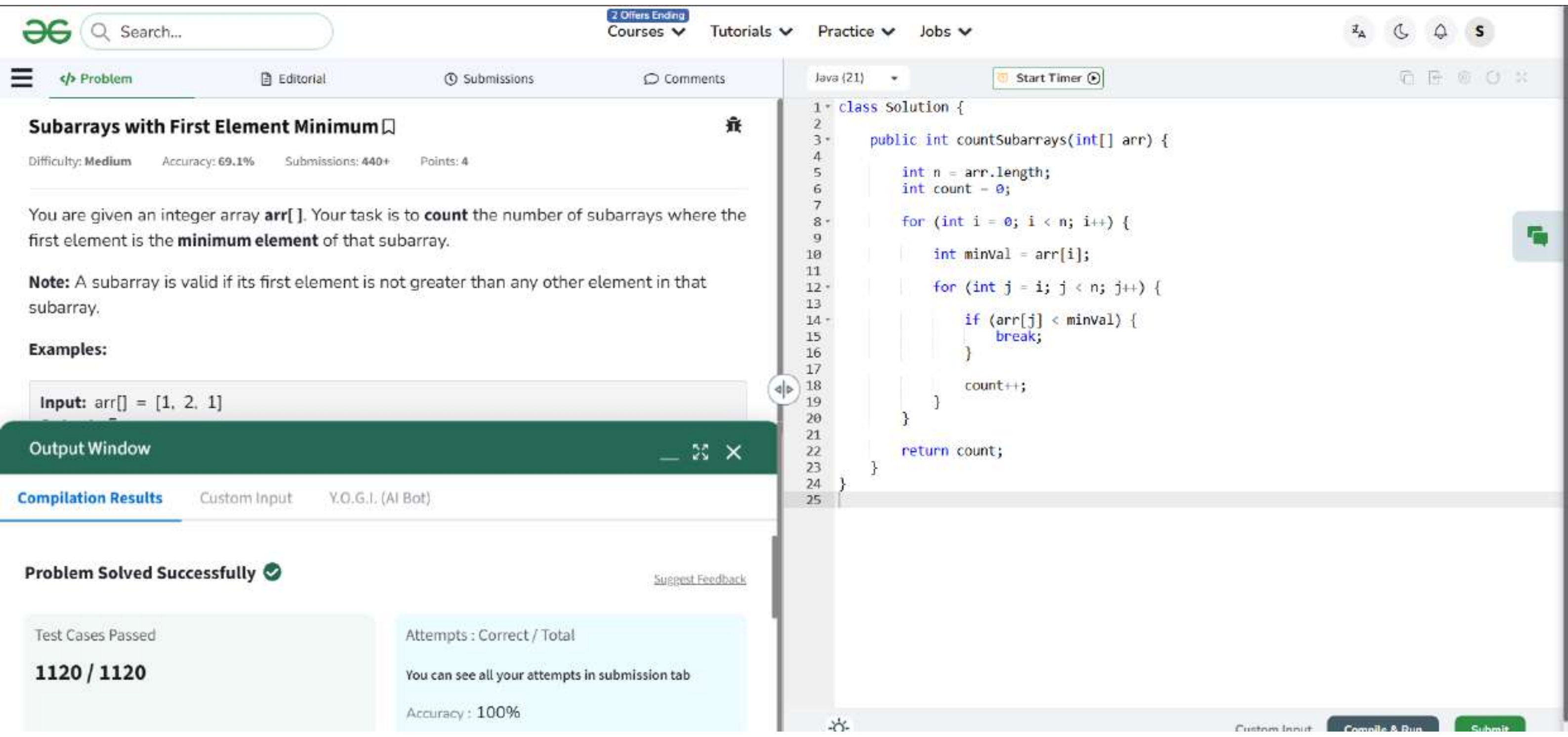
Input: arr[] = [4, 7, 11, 5, 13, 2]
Output: true
Explanation: Triplet [4, 7, 5] satisfies the condition since 4 < 7, 5 < 7 and 4 < 5.

Problem Solved Successfully

Test Cases Passed
1120 / 1120

Attempts : Correct / Total
1 / 1
Accuracy : 100%

```java
import java.util.*;

class Solution {

    public boolean has132Pattern(int[] arr) {

        int n = arr.length;
        if (n < 3) return false;

        int[] min = new int[n];
        min[0] = arr[0];


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

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


        for (int j = n - 1; j >= 0; j--) {

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

                while (!stack.isEmpty() && stack.peek() <= min[j]) {
                    stack.pop();
                }

                if (!stack.isEmpty() && stack.peek() < arr[j]) {
                    return true;
                }

                stack.push(arr[j]);
            }
        }

        return false;
```



Subarrays with First Element Minimum

Difficulty: **Medium** Accuracy: **69.1%** Submissions: **440+** Points: **4**

You are given an integer array **arr[ ]**. Your task is to **count** the number of subarrays where the first element is the **minimum element** of that subarray.

Note: A subarray is valid if its first element is not greater than any other element in that subarray.

Examples:

Input: arr[] = [1, 2, 1]

```java
class Solution {

    public int countSubarrays(int[] arr) {

        int n = arr.length;
        int count = 0;

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

            int minVal = arr[i];

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

                if (arr[j] < minVal) {
                    break;
                }

                count++;
            }
        }

        return count;
    }
}
```

Problem Solved Successfully ✓

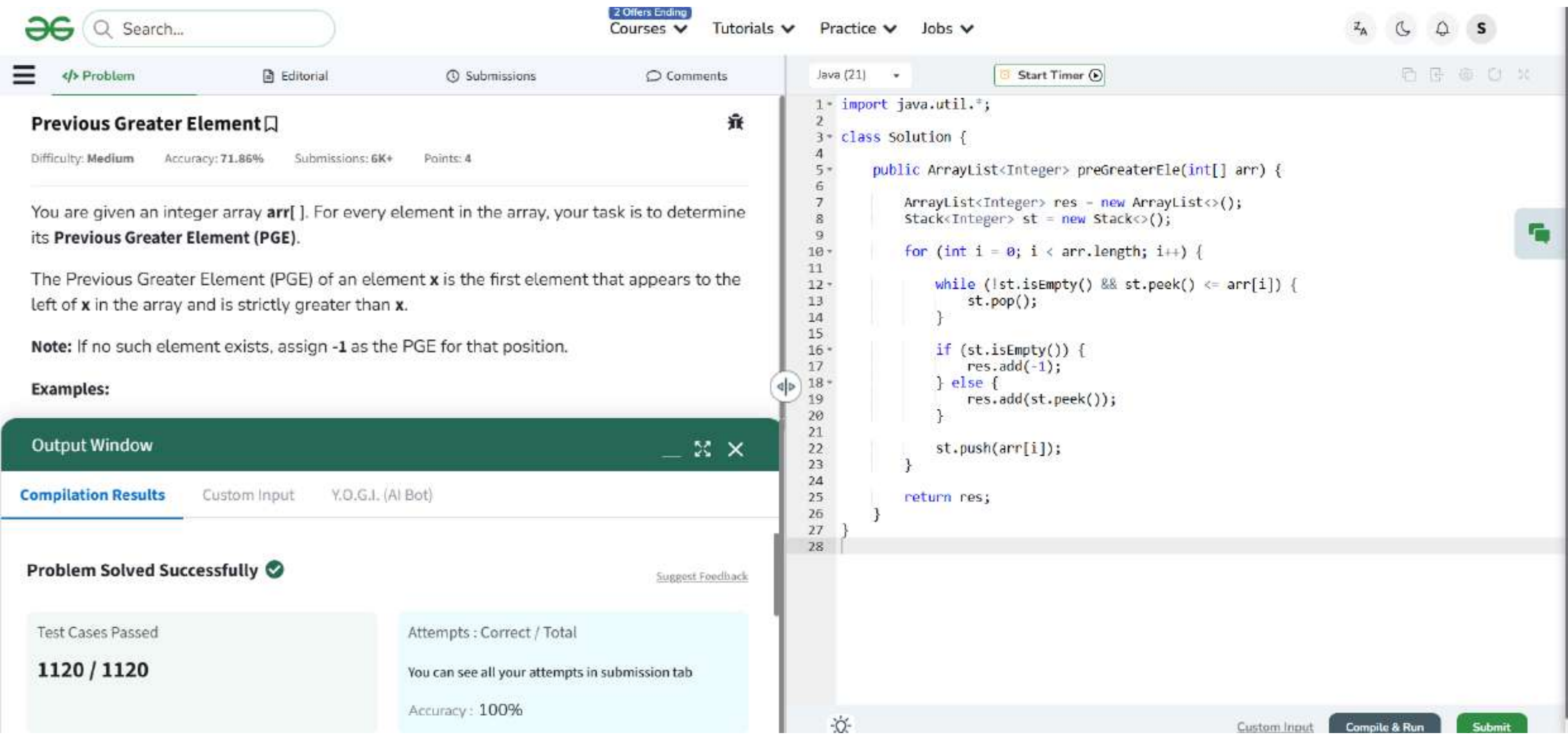
Test Cases Passed

1120 / 1120

Attempts : Correct / Total

You can see all your attempts in submission tab

Accuracy : 100%



Previous Greater Element

Difficulty: **Medium** Accuracy: **71.86%** Submissions: **6K+** Points: **4**

You are given an integer array **arr**[ ]. For every element in the array, your task is to determine its **Previous Greater Element (PGE)**.

The Previous Greater Element (PGE) of an element **x** is the first element that appears to the left of **x** in the array and is strictly greater than **x**.

Note: If no such element exists, assign **-1** as the PGE for that position.

Examples:

Problem Solved Successfully

Test Cases Passed

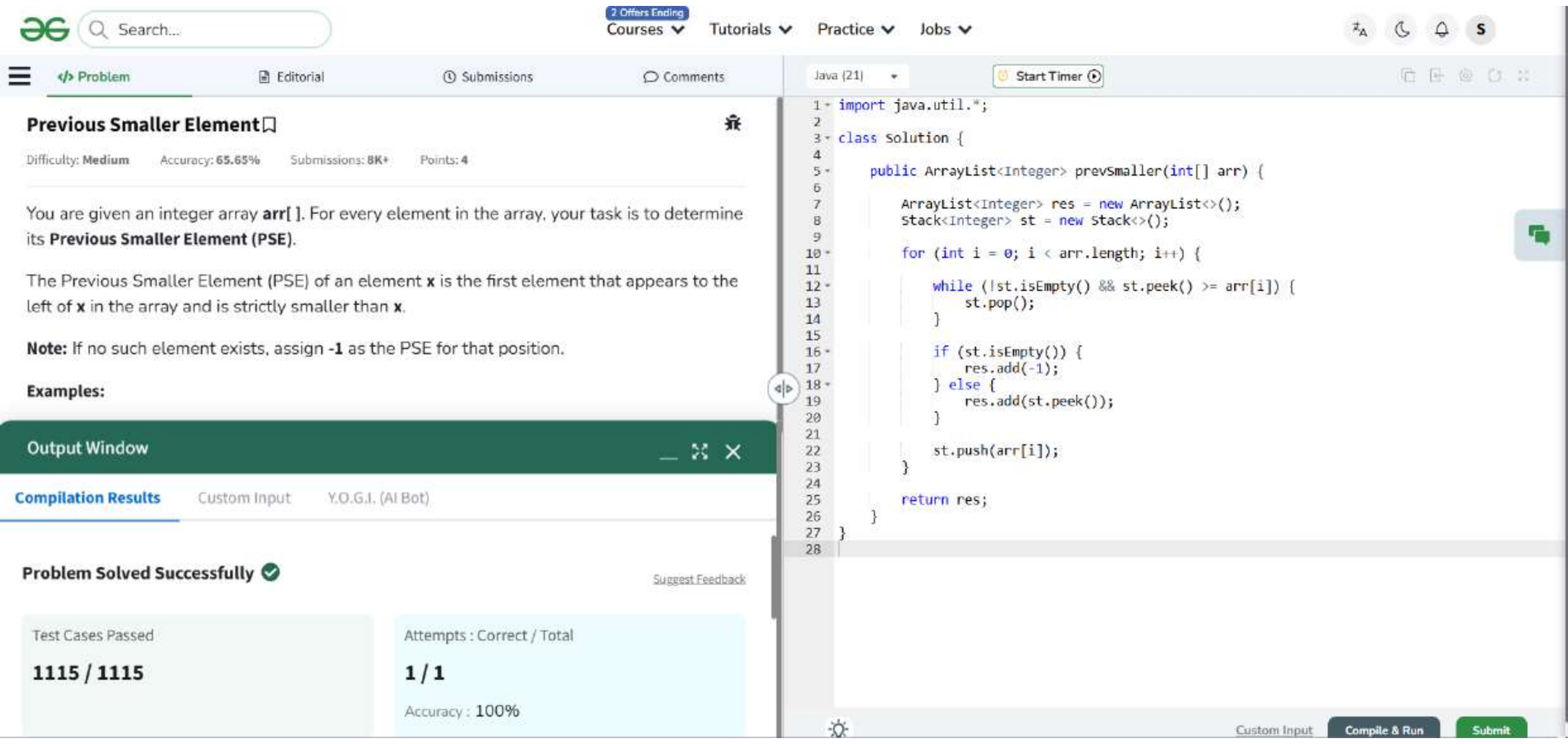
1120 / 1120

Attempts : Correct / Total

You can see all your attempts in submission tab

Accuracy : 100%

```java
import java.util.*;

class Solution {

    public ArrayList<Integer> preGreaterEle(int[] arr) {

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

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

            while (!st.isEmpty() && st.peek() <= arr[i]) {
                st.pop();
            }

            if (st.isEmpty()) {
                res.add(-1);
            } else {
                res.add(st.peek());
            }

            st.push(arr[i]);
        }

        return res;
    }
}
```



Previous Smaller Element

Difficulty: **Medium** Accuracy: **65.65%** Submissions: **8K+** Points: **4**

You are given an integer array **arr[ ]**. For every element in the array, your task is to determine its **Previous Smaller Element (PSE)**.

The Previous Smaller Element (PSE) of an element **x** is the first element that appears to the left of **x** in the array and is strictly smaller than **x**.

Note: If no such element exists, assign **-1** as the PSE for that position.

Examples:

Output Window

Problem Solved Successfully

Suggest Feedback

Test Cases Passed

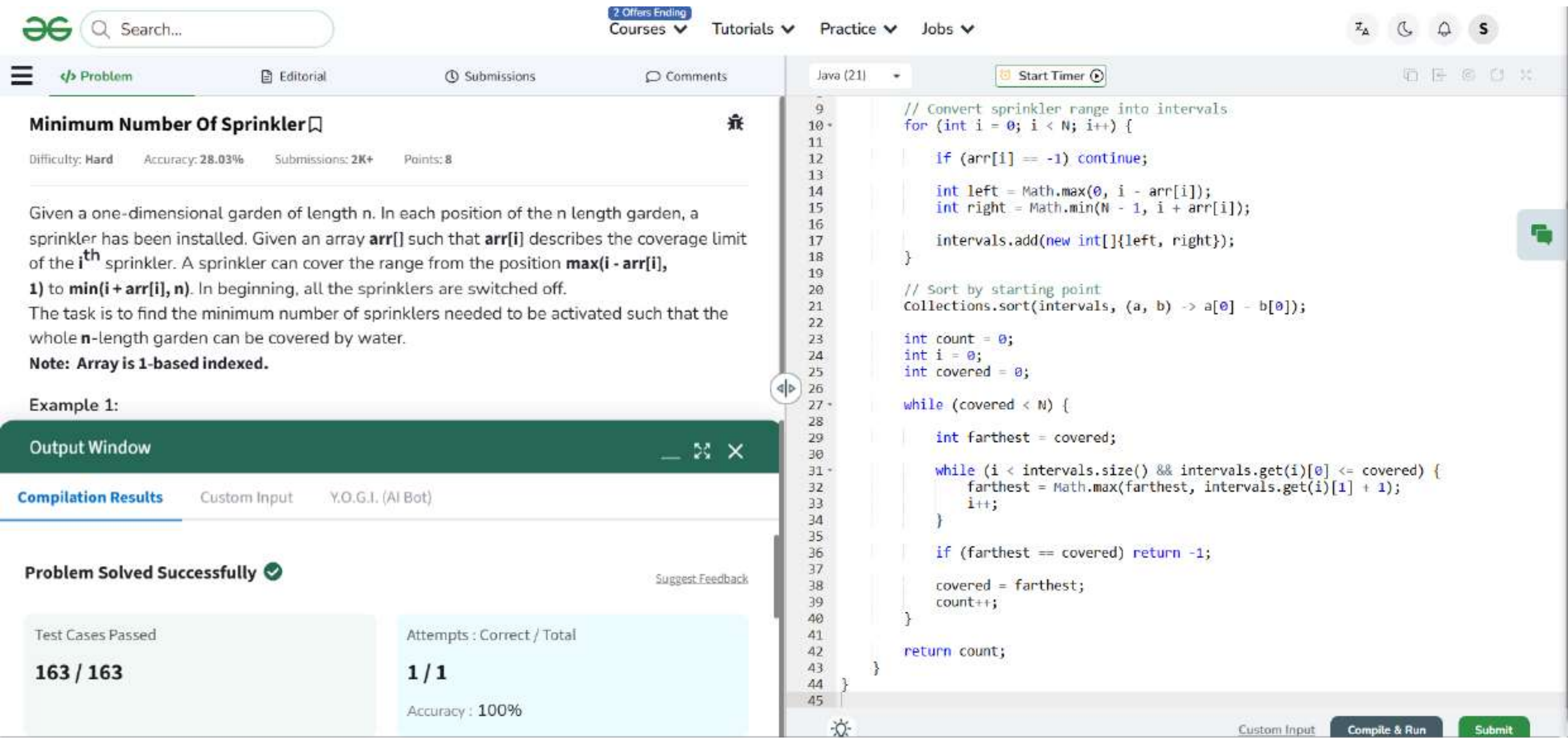
1115 / 1115

Attempts : Correct / Total

1 / 1

Accuracy : 100%

```java
import java.util.*;

class Solution {

    public ArrayList<Integer> prevSmaller(int[] arr) {

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

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

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

            if (st.isEmpty()) {
                res.add(-1);
            } else {
                res.add(st.peek());
            }

            st.push(arr[i]);
        }

        return res;
    }
}
```



Minimum Number Of Sprinkler

Difficulty: Hard Accuracy: 28.03% Submissions: 2K+ Points: 8

Given a one-dimensional garden of length n. In each position of the n length garden, a sprinkler has been installed. Given an array **arr[]** such that **arr[i]** describes the coverage limit of the **i**th sprinkler. A sprinkler can cover the range from the position **max(i - arr[i], 1)** to **min(i + arr[i], n)**. In beginning, all the sprinklers are switched off.
The task is to find the minimum number of sprinklers needed to be activated such that the whole **n**-length garden can be covered by water.
Note: Array is 1-based indexed.

Example 1:

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

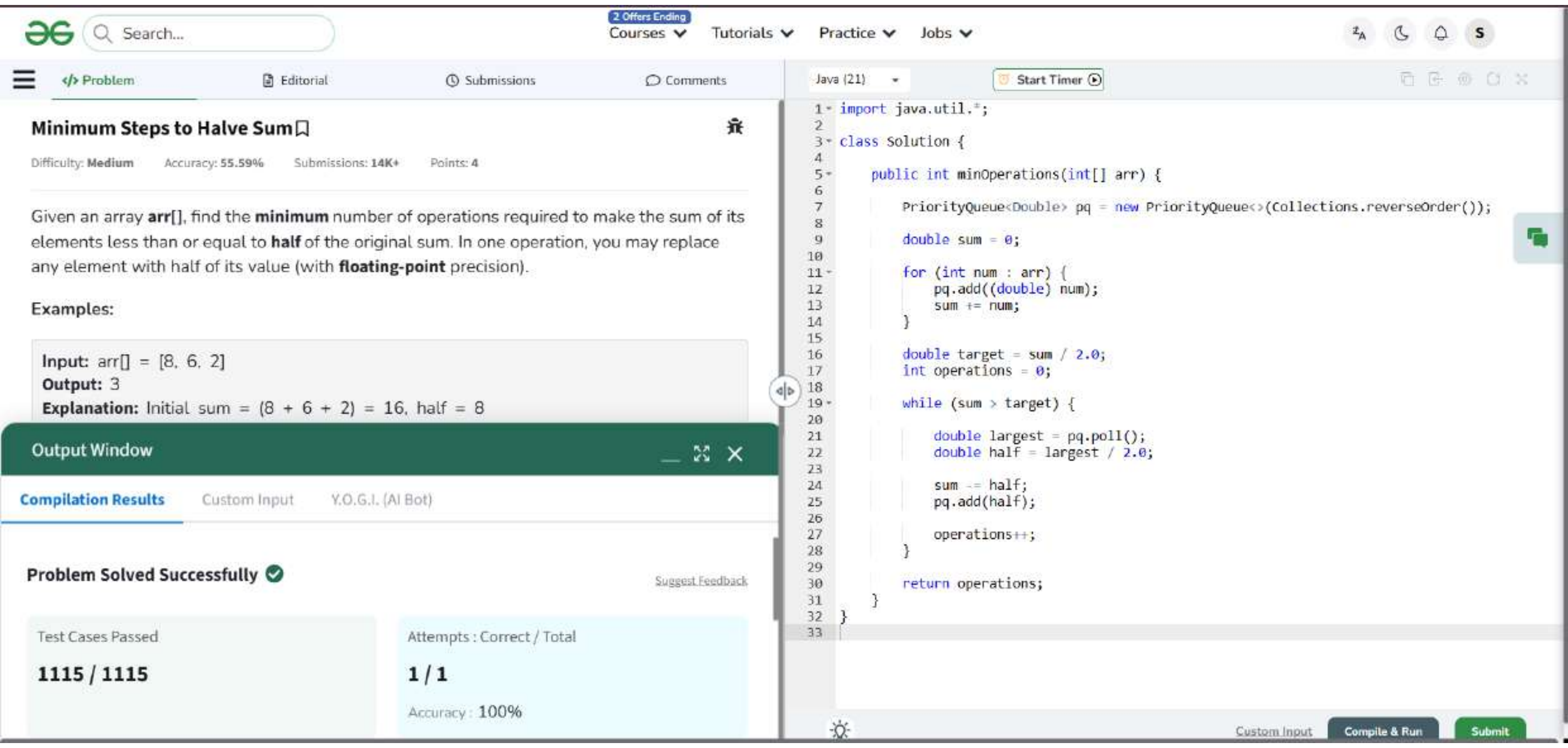
Problem Solved Successfully

Suggest Feedback

Test Cases Passed
163 / 163

Attempts : Correct / Total
1 / 1
Accuracy : 100%

```java
        // Convert sprinkler range into intervals
        for (int i = 0; i < N; i++) {

            if (arr[i] == -1) continue;

            int left = Math.max(0, i - arr[i]);
            int right = Math.min(N - 1, i + arr[i]);

            intervals.add(new int[]{left, right});
        }

        // Sort by starting point
        Collections.sort(intervals, (a, b) -> a[0] - b[0]);

        int count = 0;
        int i = 0;
        int covered = 0;

        while (covered < N) {

            int farthest = covered;

            while (i < intervals.size() && intervals.get(i)[0] <= covered) {
                farthest = Math.max(farthest, intervals.get(i)[1] + 1);
                i++;
            }

            if (farthest == covered) return -1;

            covered = farthest;
            count++;
        }

        return count;
    }
}
```



Minimum Steps to Halve Sum

Difficulty: **Medium** Accuracy: **55.59%** Submissions: **14K+** Points: **4**

Given an array **arr[]**, find the **minimum** number of operations required to make the sum of its elements less than or equal to **half** of the original sum. In one operation, you may replace any element with half of its value (with **floating-point** precision).

Examples:

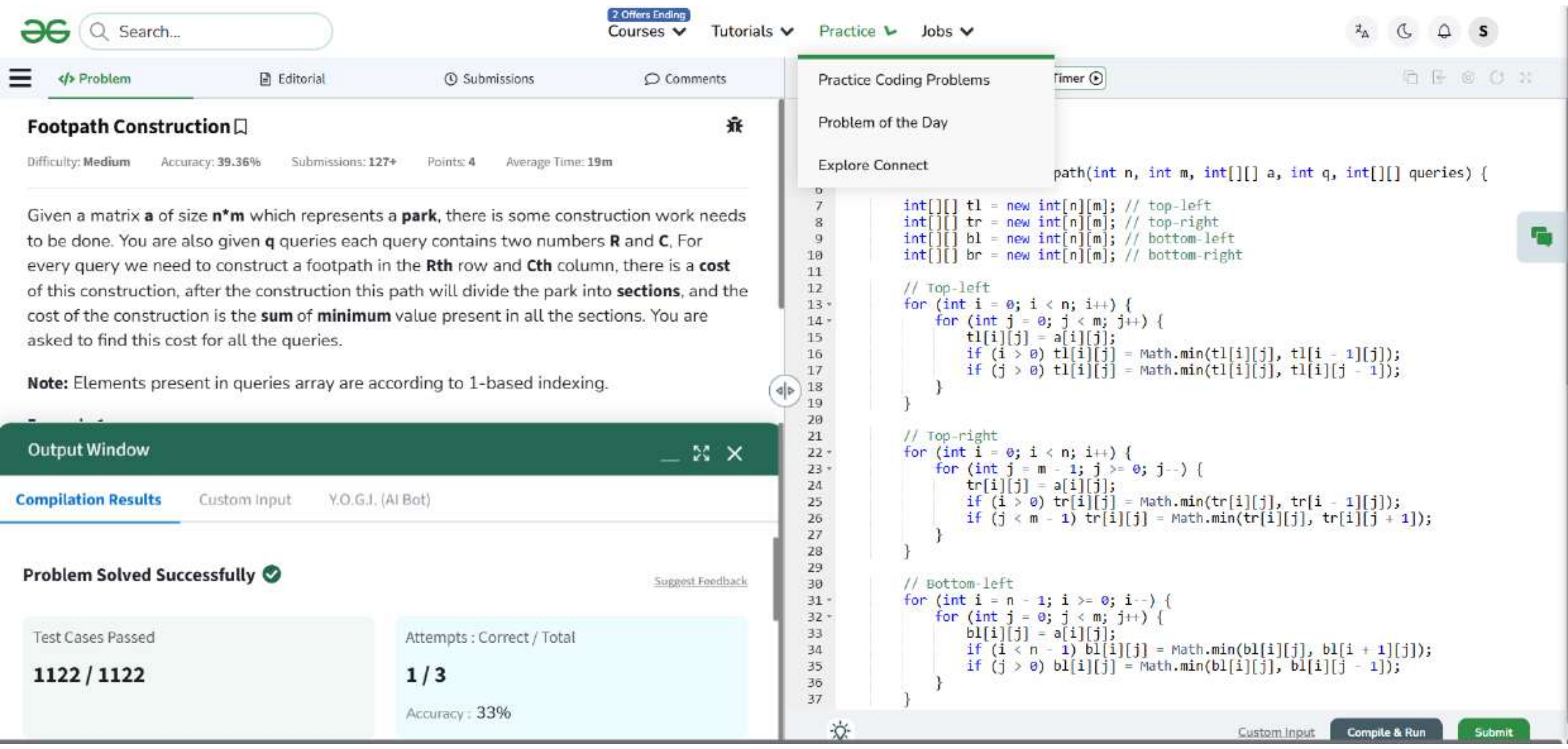
Input: arr[] = [8, 6, 2]
Output: 3
Explanation: Initial sum = (8 + 6 + 2) = 16, half = 8

Problem Solved Successfully

Test Cases Passed
1115 / 1115

Attempts : Correct / Total
1 / 1
Accuracy : 100%

```java
import java.util.*;

class Solution {

    public int minOperations(int[] arr) {

        PriorityQueue<Double> pq = new PriorityQueue<>(Collections.reverseOrder());

        double sum = 0;

        for (int num : arr) {
            pq.add((double) num);
            sum += num;
        }

        double target = sum / 2.0;
        int operations = 0;

        while (sum > target) {

            double largest = pq.poll();
            double half = largest / 2.0;

            sum -= half;
            pq.add(half);

            operations++;
        }

        return operations;
    }
}
```



Footpath Construction

Difficulty: **Medium** Accuracy: **39.36%** Submissions: **127+** Points: **4** Average Time: **19m**

Given a matrix **a** of size **n*m** which represents a **park**, there is some construction work needs to be done. You are also given **q** queries each query contains two numbers **R** and **C**, For every query we need to construct a footpath in the **Rth** row and **Cth** column, there is a **cost** of this construction, after the construction this path will divide the park into **sections**, and the cost of the construction is the **sum** of **minimum** value present in all the sections. You are asked to find this cost for all the queries.

Note: Elements present in queries array are according to 1-based indexing.

Output Window

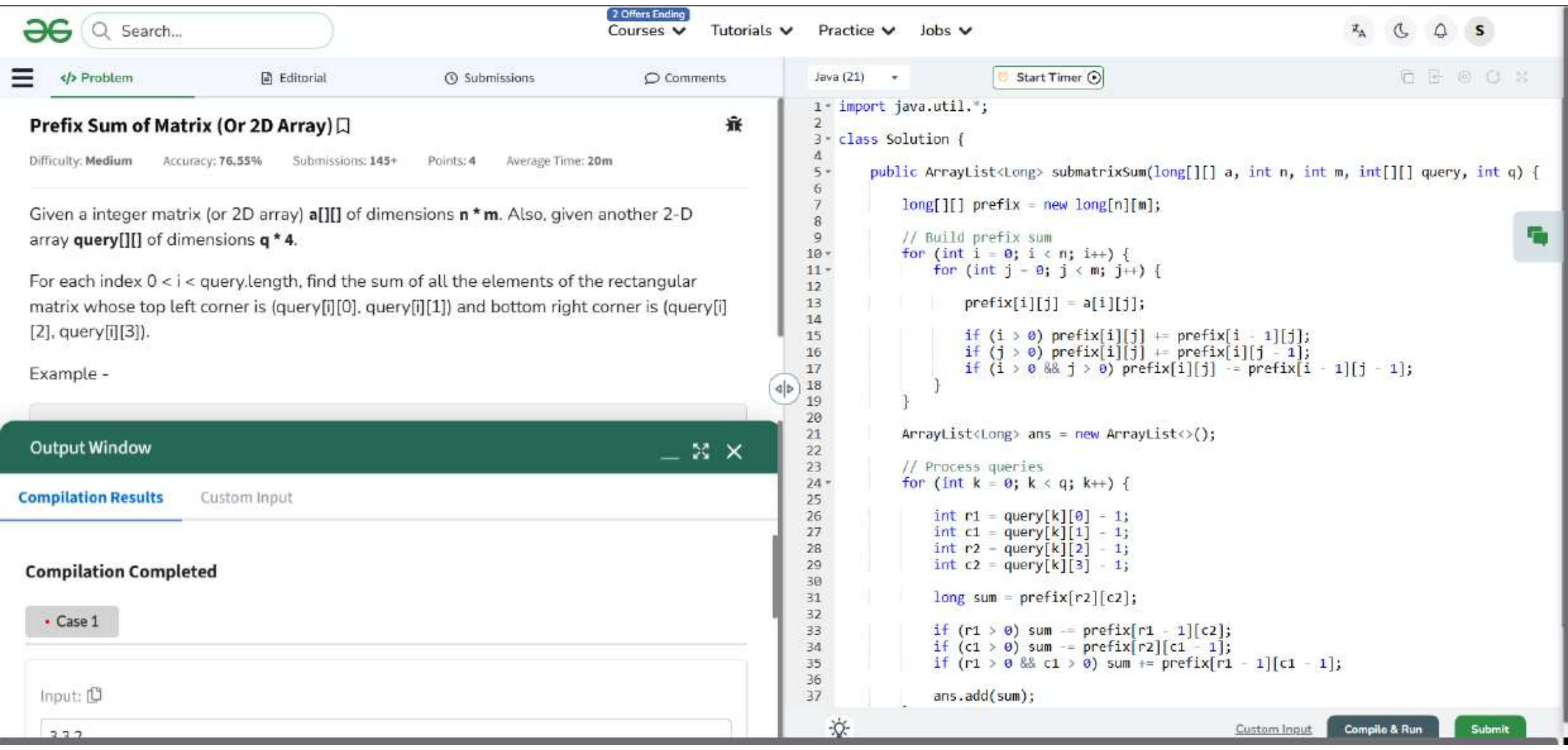
Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully Suggest Feedback

Test Cases Passed
1122 / 1122

Attempts : Correct / Total
1 / 3
Accuracy : 33%

```java
path(int n, int m, int[][] a, int q, int[][] queries) {
        int[][] tl = new int[n][m]; // top-left
        int[][] tr = new int[n][m]; // top-right
        int[][] bl = new int[n][m]; // bottom-left
        int[][] br = new int[n][m]; // bottom-right

        // Top-left
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                tl[i][j] = a[i][j];
                if (i > 0) tl[i][j] = Math.min(tl[i][j], tl[i - 1][j]);
                if (j > 0) tl[i][j] = Math.min(tl[i][j], tl[i][j - 1]);
            }
        }

        // Top-right
        for (int i = 0; i < n; i++) {
            for (int j = m - 1; j >= 0; j--) {
                tr[i][j] = a[i][j];
                if (i > 0) tr[i][j] = Math.min(tr[i][j], tr[i - 1][j]);
                if (j < m - 1) tr[i][j] = Math.min(tr[i][j], tr[i][j + 1]);
            }
        }

        // Bottom-left
        for (int i = n - 1; i >= 0; i--) {
            for (int j = 0; j < m; j++) {
                bl[i][j] = a[i][j];
                if (i < n - 1) bl[i][j] = Math.min(bl[i][j], bl[i + 1][j]);
                if (j > 0) bl[i][j] = Math.min(bl[i][j], bl[i][j - 1]);
            }
        }
```



Prefix Sum of Matrix (Or 2D Array)

Difficulty: **Medium** Accuracy: **76.55%** Submissions: **145+** Points: **4** Average Time: **20m**

Given a integer matrix (or 2D array) **a[][]** of dimensions **n * m**. Also, given another 2-D array **query[][]** of dimensions **q * 4**.

For each index 0 < i < query.length, find the sum of all the elements of the rectangular matrix whose top left corner is (query[i][0], query[i][1]) and bottom right corner is (query[i][2], query[i][3]).

Example -

Output Window

Compilation Results Custom Input

Compilation Completed

Case 1

Input:

3 3 2

```java
import java.util.*;

class Solution {

    public ArrayList<Long> submatrixSum(long[][] a, int n, int m, int[][] query, int q) {

        long[][] prefix = new long[n][m];

        // Build prefix sum
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {

                prefix[i][j] = a[i][j];

                if (i > 0) prefix[i][j] += prefix[i - 1][j];
                if (j > 0) prefix[i][j] += prefix[i][j - 1];
                if (i > 0 && j > 0) prefix[i][j] -= prefix[i - 1][j - 1];
            }
        }

        ArrayList<Long> ans = new ArrayList<>();

        // Process queries
        for (int k = 0; k < q; k++) {

            int r1 = query[k][0] - 1;
            int c1 = query[k][1] - 1;
            int r2 = query[k][2] - 1;
            int c2 = query[k][3] - 1;

            long sum = prefix[r2][c2];

            if (r1 > 0) sum -= prefix[r1 - 1][c2];
            if (c1 > 0) sum -= prefix[r2][c1 - 1];
            if (r1 > 0 && c1 > 0) sum += prefix[r1 - 1][c1 - 1];

            ans.add(sum);
```