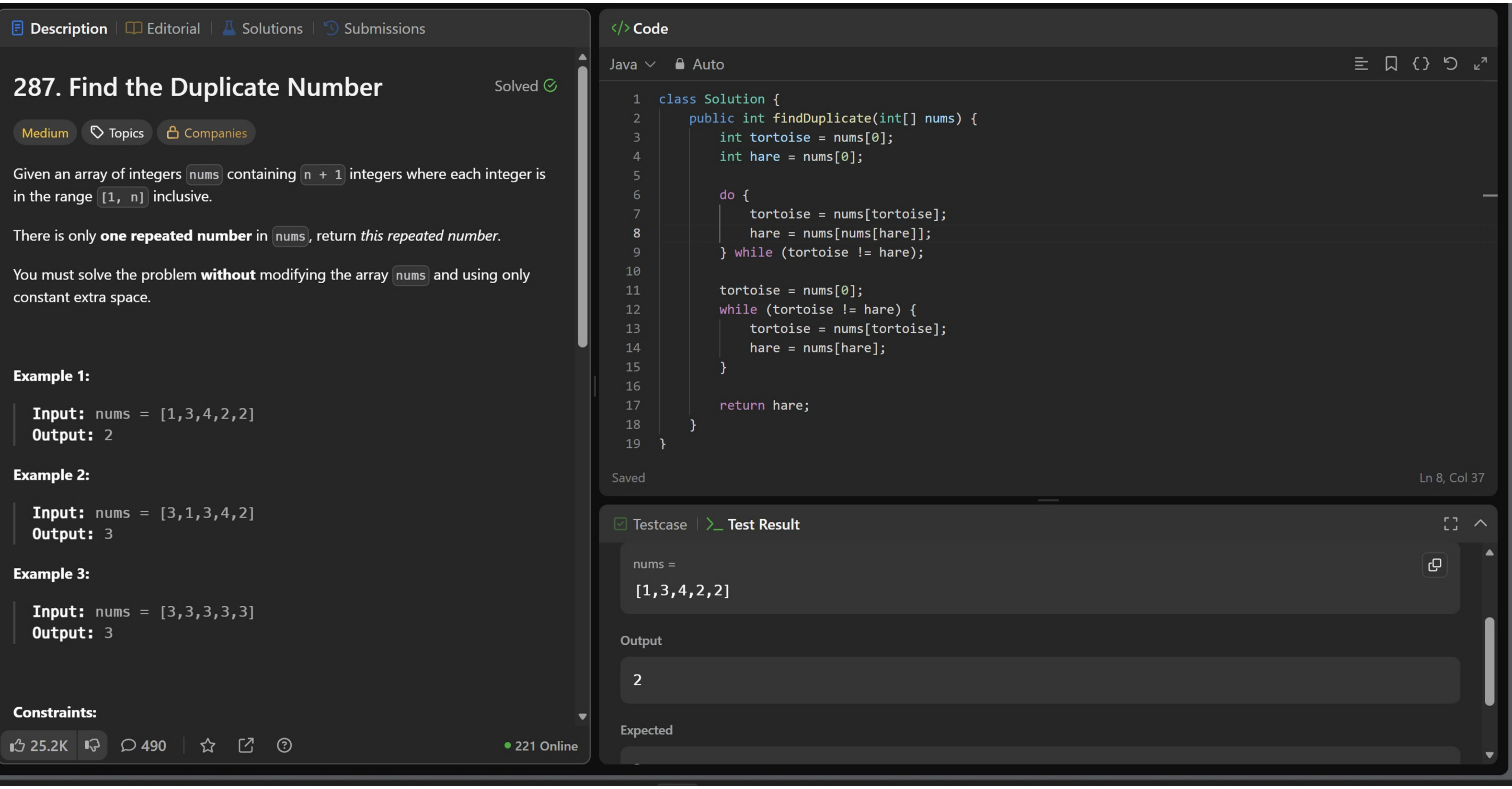


287. Find the Duplicate Number

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and using only constant extra space.

Example 1:

```
Input: nums = [1,3,4,2,2]
Output: 2
```

Example 2:

```
Input: nums = [3,1,3,4,2]
Output: 3
```

Example 3:

```
Input: nums = [3,3,3,3,3]
Output: 3
```

Constraints:

```java
class Solution {
    public int findDuplicate(int[] nums) {
        int tortoise = nums[0];
        int hare = nums[0];

        do {
            tortoise = nums[tortoise];
            hare = nums[nums[hare]];
        } while (tortoise != hare);

        tortoise = nums[0];
        while (tortoise != hare) {
            tortoise = nums[tortoise];
            hare = nums[hare];
        }

        return hare;
    }
}
```

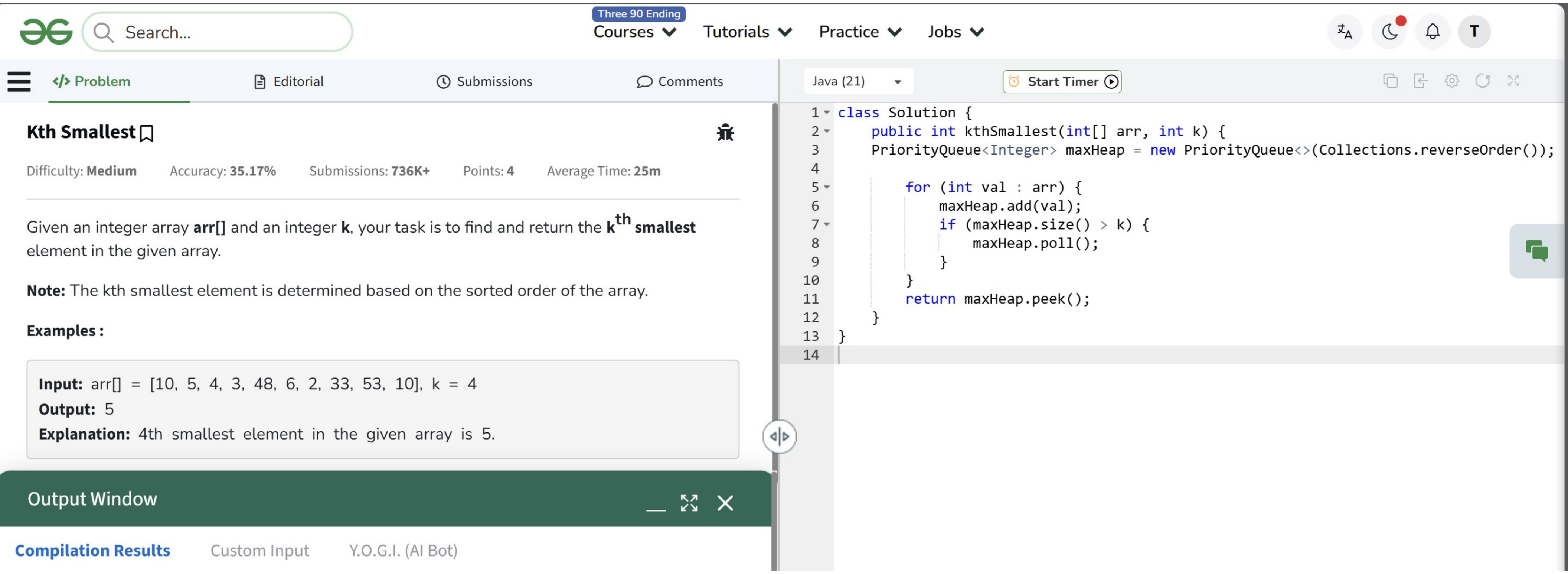
nums =

```
[1,3,4,2,2]
```

Output

```
2
```

Expected



Kth Smallest

Difficulty: **Medium** Accuracy: **35.17%** Submissions: **736K+** Points: **4** Average Time: **25m**

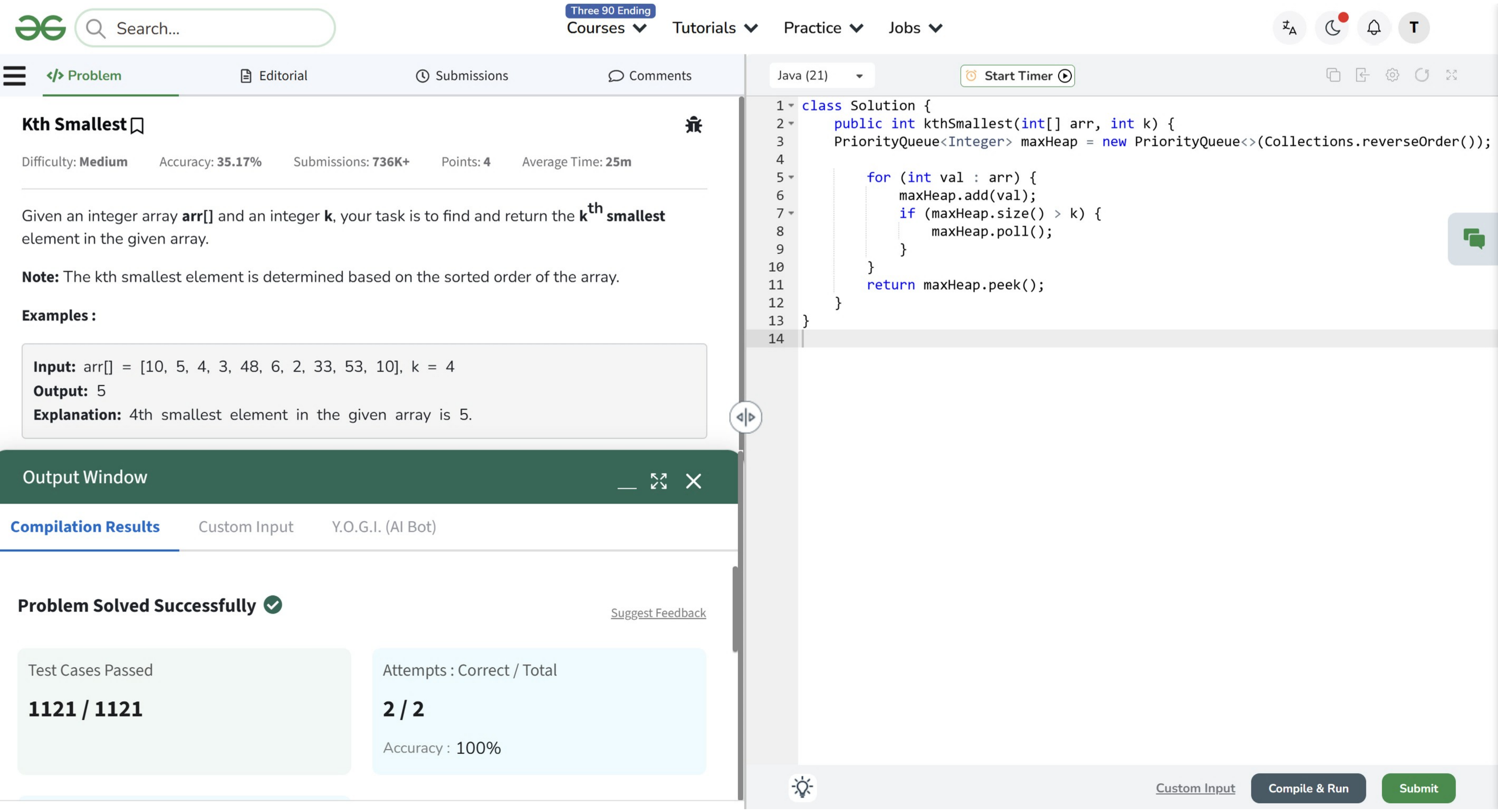
Given an integer array **arr[]** and an integer **k**, your task is to find and return the **k**th **smallest** element in the given array.

Note: The kth smallest element is determined based on the sorted order of the array.

Examples :

Input: arr[] = [10, 5, 4, 3, 48, 6, 2, 33, 53, 10], k = 4
Output: 5
Explanation: 4th smallest element in the given array is 5.

```java
class Solution {
    public int kthSmallest(int[] arr, int k) {
    PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());

        for (int val : arr) {
            maxHeap.add(val);
            if (maxHeap.size() > k) {
                maxHeap.poll();
            }
        }
        return maxHeap.peek();
    }
}
```



Three 90 Ending
Courses Tutorials Practice Jobs

Problem Editorial Submissions Comments

Kth Smallest

Difficulty: **Medium** Accuracy: **35.17%** Submissions: **736K+** Points: **4** Average Time: **25m**

Given an integer array **arr[]** and an integer **k**, your task is to find and return the **k**th **smallest** element in the given array.

Note: The kth smallest element is determined based on the sorted order of the array.

Examples :

Input: arr[] = [10, 5, 4, 3, 48, 6, 2, 33, 53, 10], k = 4
Output: 5
Explanation: 4th smallest element in the given array is 5.

Output Window

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

Problem Solved Successfully

Suggest Feedback

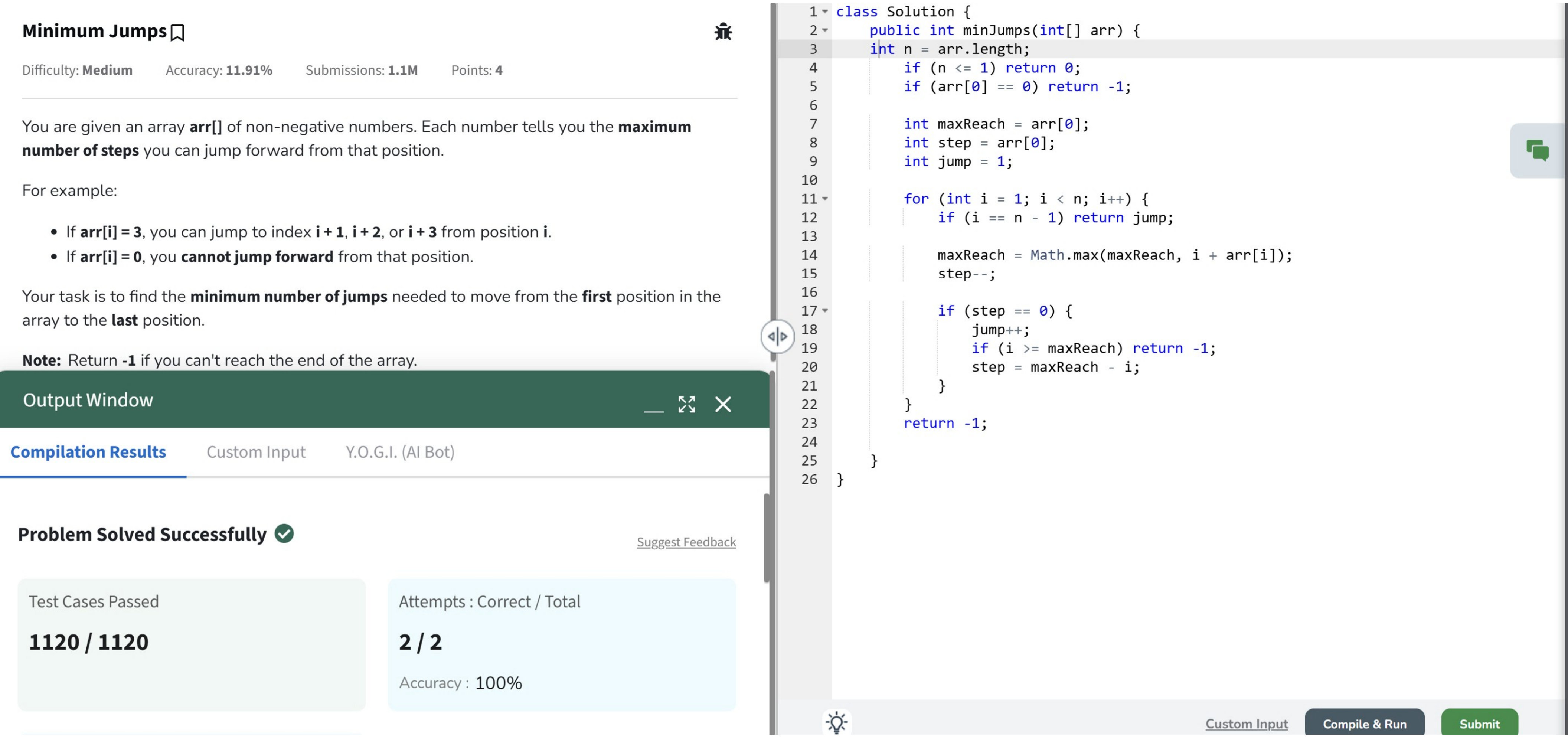
Test Cases Passed
1121 / 1121

Attempts : Correct / Total
2 / 2
Accuracy : 100%

Java (21) Start Timer

```java
class Solution {
    public int kthSmallest(int[] arr, int k) {
    PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());

        for (int val : arr) {
            maxHeap.add(val);
            if (maxHeap.size() > k) {
                maxHeap.poll();
            }
        }
        return maxHeap.peek();
    }
}
```

Custom Input Compile & Run Submit


Minimum Jumps

Difficulty: **Medium** Accuracy: **11.91%** Submissions: **1.1M** Points: **4**

You are given an array **arr[]** of non-negative numbers. Each number tells you the **maximum number of steps** you can jump forward from that position.

For example:

- If **arr[i] = 3**, you can jump to index **i + 1**, **i + 2**, or **i + 3** from position **i**.
- If **arr[i] = 0**, you **cannot jump forward** from that position.

Your task is to find the **minimum number of jumps** needed to move from the **first** position in the array to the **last** position.

Note: Return **-1** if you can't reach the end of the array.

Output Window

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

Problem Solved Successfully

Suggest Feedback

Test Cases Passed

1120 / 1120

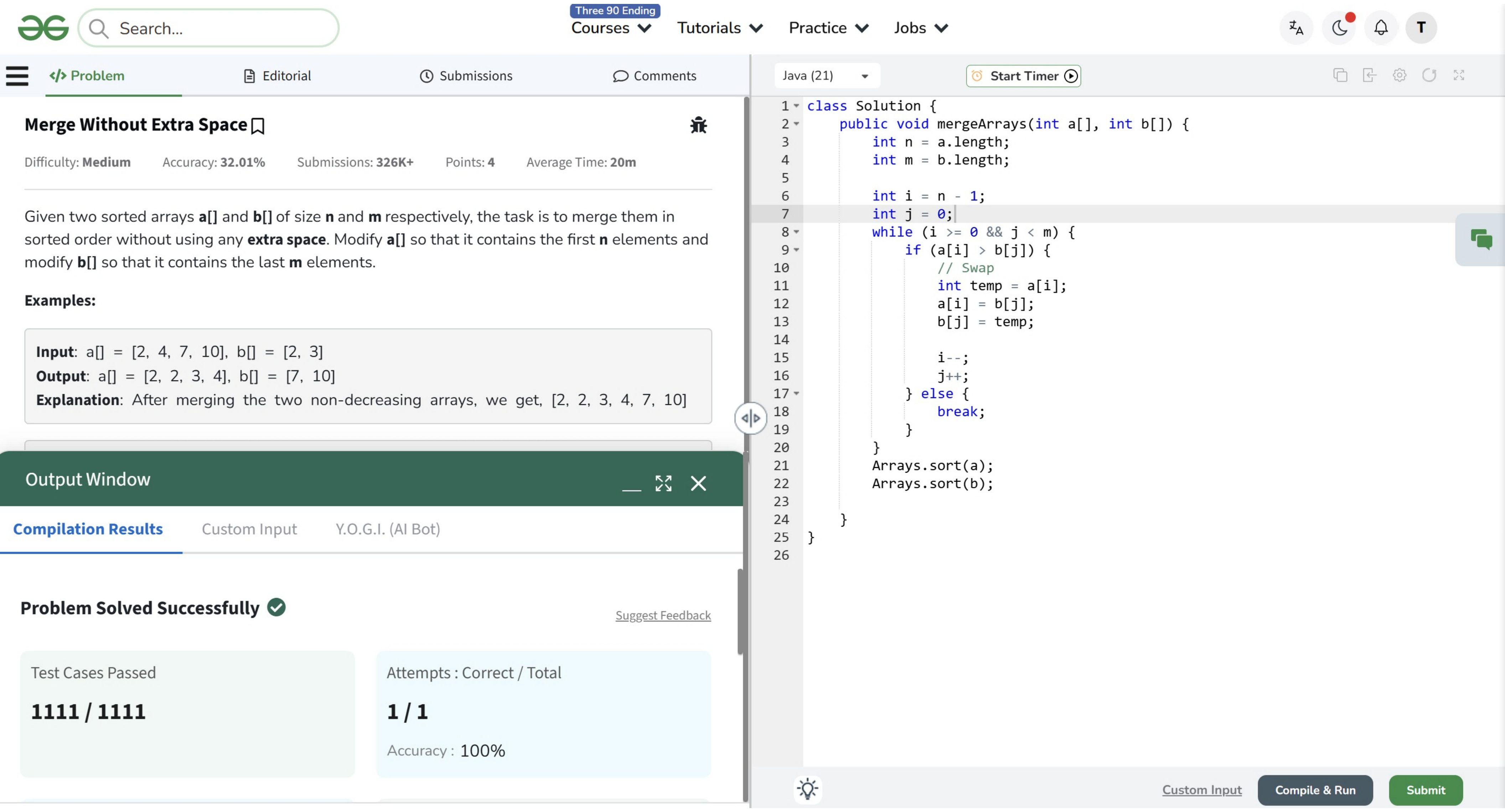
Attempts : Correct / Total

2 / 2

Accuracy : 100%

```java
class Solution {
    public int minJumps(int[] arr) {
    int n = arr.length;
        if (n <= 1) return 0;
        if (arr[0] == 0) return -1;

        int maxReach = arr[0];
        int step = arr[0];
        int jump = 1;

        for (int i = 1; i < n; i++) {
            if (i == n - 1) return jump;

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

            if (step == 0) {
                jump++;
                if (i >= maxReach) return -1;
                step = maxReach - i;
            }
        }
        return -1;

    }
}
```

Custom Input   Compile & Run   Submit



Merge Without Extra Space

Difficulty: **Medium** Accuracy: **32.01%** Submissions: **326K+** Points: **4** Average Time: **20m**

Given two sorted arrays **a[]** and **b[]** of size **n** and **m** respectively, the task is to merge them in sorted order without using any **extra space**. Modify **a[]** so that it contains the first **n** elements and modify **b[]** so that it contains the last **m** elements.

Examples:

Input: a[] = [2, 4, 7, 10], b[] = [2, 3]
Output: a[] = [2, 2, 3, 4], b[] = [7, 10]
Explanation: After merging the two non-decreasing arrays, we get, [2, 2, 3, 4, 7, 10]

```java
class Solution {
    public void mergeArrays(int a[], int b[]) {
        int n = a.length;
        int m = b.length;

        int i = n - 1;
        int j = 0;
        while (i >= 0 && j < m) {
            if (a[i] > b[j]) {
                // Swap
                int temp = a[i];
                a[i] = b[j];
                b[j] = temp;

                i--;
                j++;
            } else {
                break;
            }
        }
        Arrays.sort(a);
        Arrays.sort(b);

    }
}

```

Problem Solved Successfully

Test Cases Passed

1111 / 1111

Attempts : Correct / Total

1 / 1

Accuracy : 100%