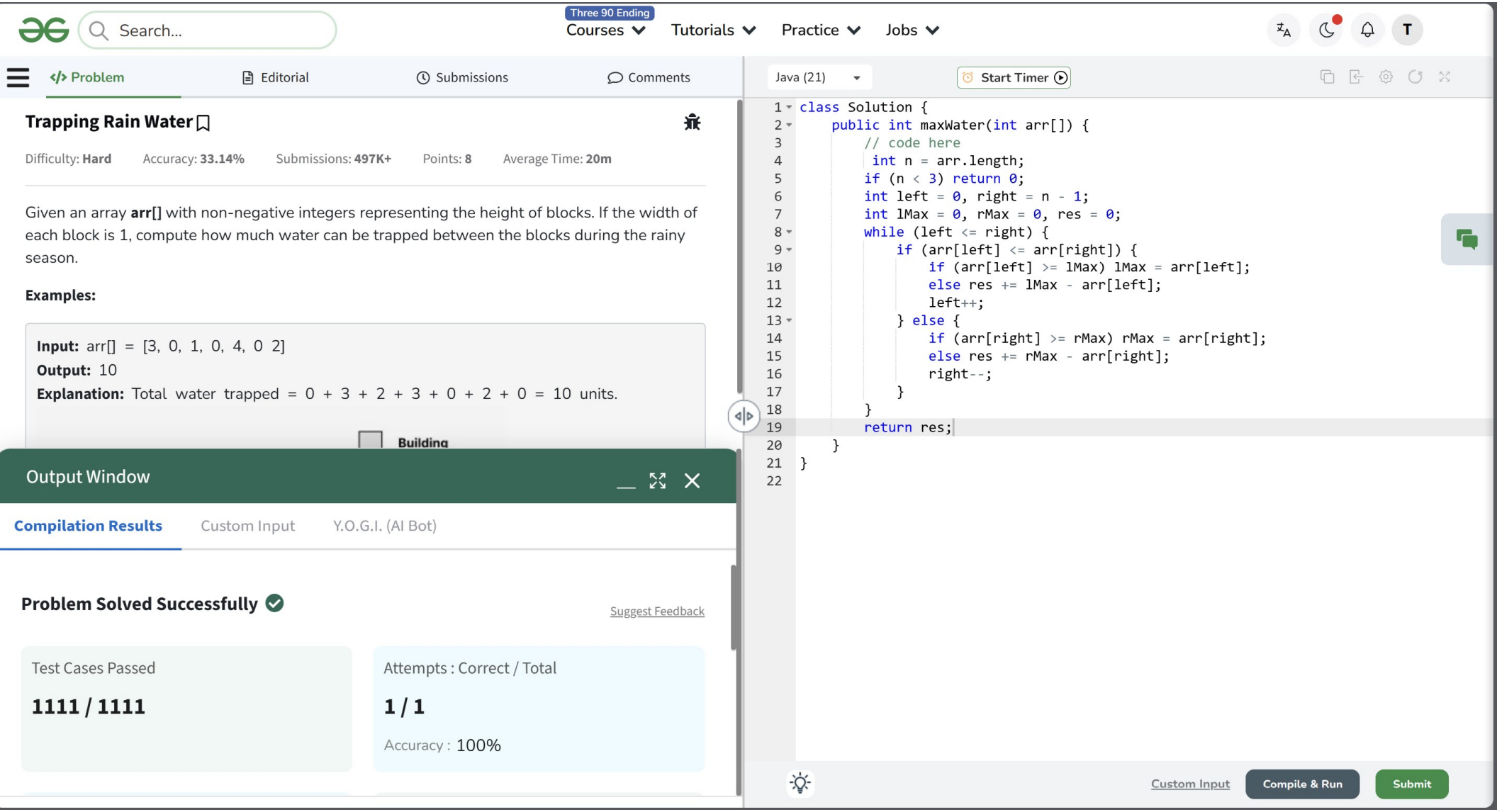

Search...
Three 90 Ending
Courses
Tutorials
Practice
Jobs
T
Problem
Editorial
Submissions
Comments
Trapping Rain Water
Difficulty: Hard
Accuracy: 33.14%
Submissions: 497K+
Points: 8
Average Time: 20m
Given an array arr[] with non-negative integers representing the height of blocks. If the width of each block is 1, compute how much water can be trapped between the blocks during the rainy season.
Examples:
Input: arr[] = [3, 0, 1, 0, 4, 0 2]
Output: 10
Explanation: Total water trapped = 0 + 3 + 2 + 3 + 0 + 2 + 0 = 10 units.
Building
Output Window
Compilation Results
Custom Input
Y.O.G.I. (AI Bot)
Problem Solved Successfully
Suggest Feedback
Test Cases Passed
1111 / 1111
Attempts : Correct / Total
1 / 1
Accuracy : 100%
Java (21)
Start Timer
class Solution {
public int maxWater(int arr[]) {
// code here
int n = arr.length;
if (n < 3) return 0;
int left = 0, right = n - 1;
int lMax = 0, rMax = 0, res = 0;
while (left <= right) {
if (arr[left] <= arr[right]) {
if (arr[left] >= lMax) lMax = arr[left];
else res += lMax - arr[left];
left++;
} else {
if (arr[right] >= rMax) rMax = arr[right];
else res += rMax - arr[right];
right--;
}
}
return res;
}
}
Custom Input
Compile & Run
Submit

Description | Editorial | Solutions | Subm

Given an array of `intervals` where `intervals[i] = [start_i, end_i]`, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input.*

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

Example 3:

```
Input: intervals = [[4,7],[1,4]]
Output: [[1,7]]
Explanation: Intervals [1,4] and [4,7] are considered overlapping.
```

Constraints:

24.3K | 263 | 421 Online

Code

Java | Auto

```java
class Solution {
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

        LinkedList<int[]> merged = new LinkedList<>();

        for (int[] interval : intervals) {
            if (merged.isEmpty() || merged.getLast()[1] < interval[0]) {
                merged.add(interval);
            }
            else {
                merged.getLast()[1] = Math.max(merged.getLast()[1], interval[1]);
            }
        }

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

    }
}
```

Saved

Ln 10, Col 15

Testcase | Test Result

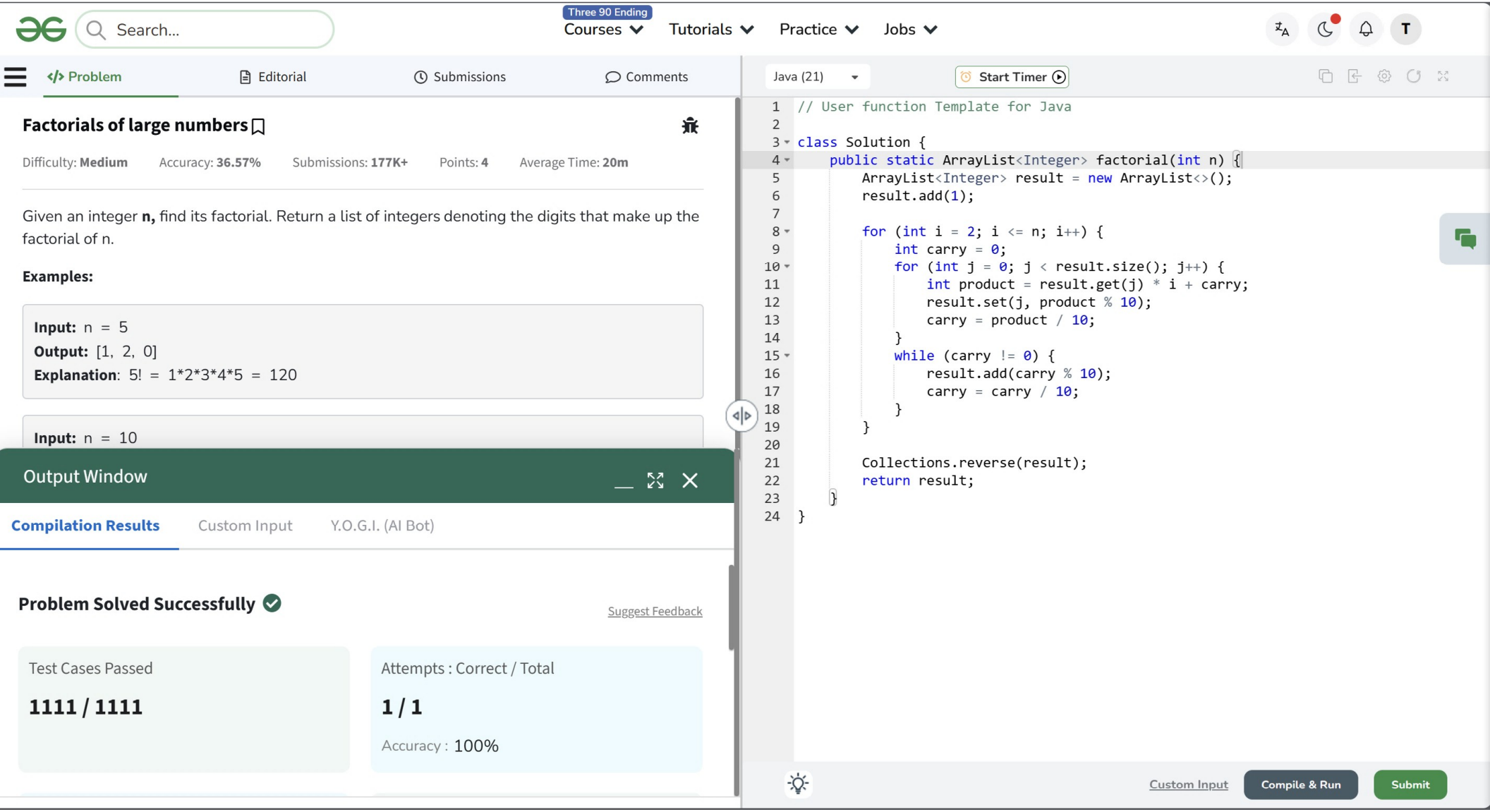
```
[[1,3],[2,6],[8,10],[15,18]]
```

Output

```
[[1,6],[8,10],[15,18]]
```

Expected

```
[[1,6],[8,10],[15,18]]
```

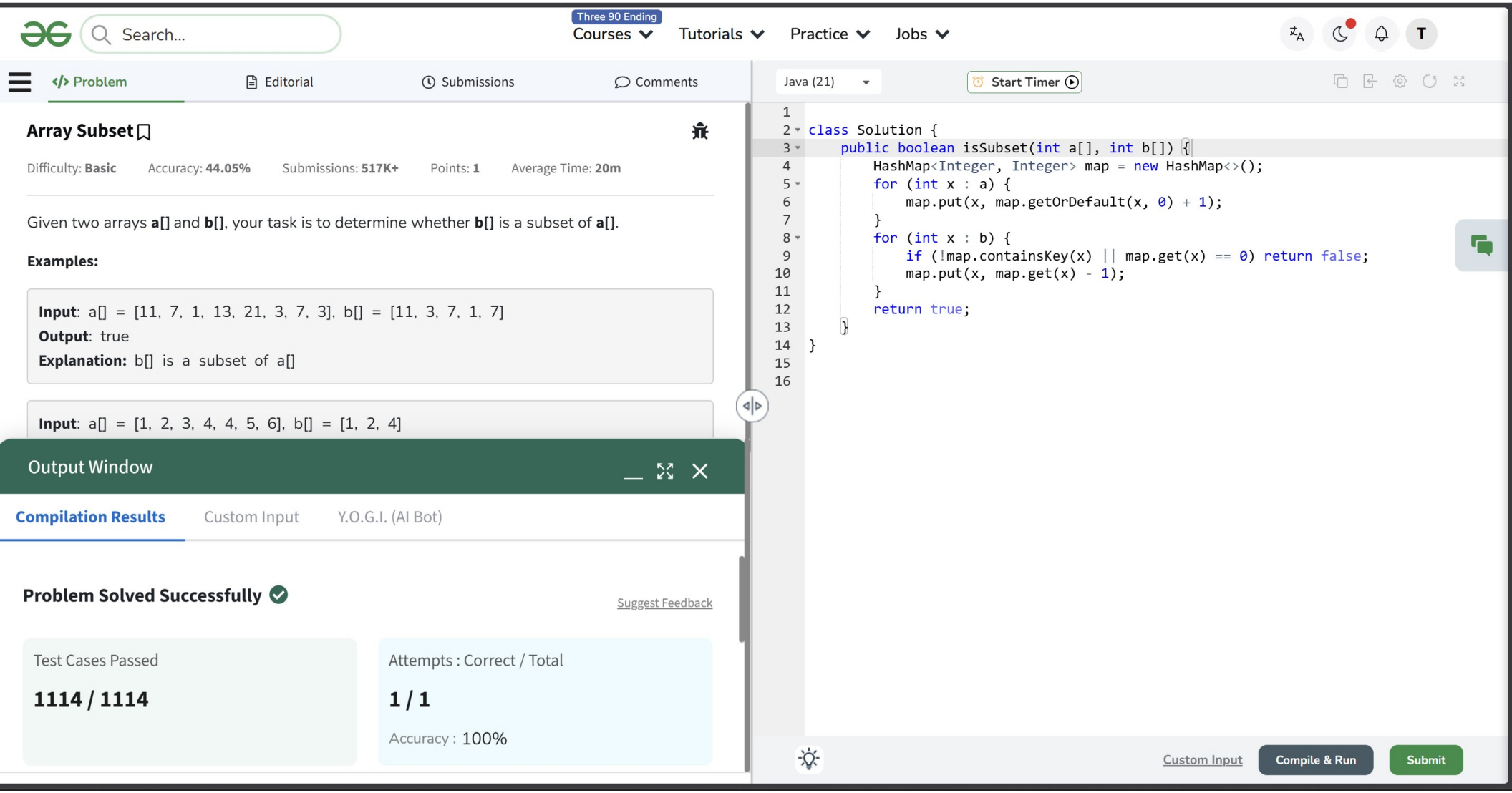


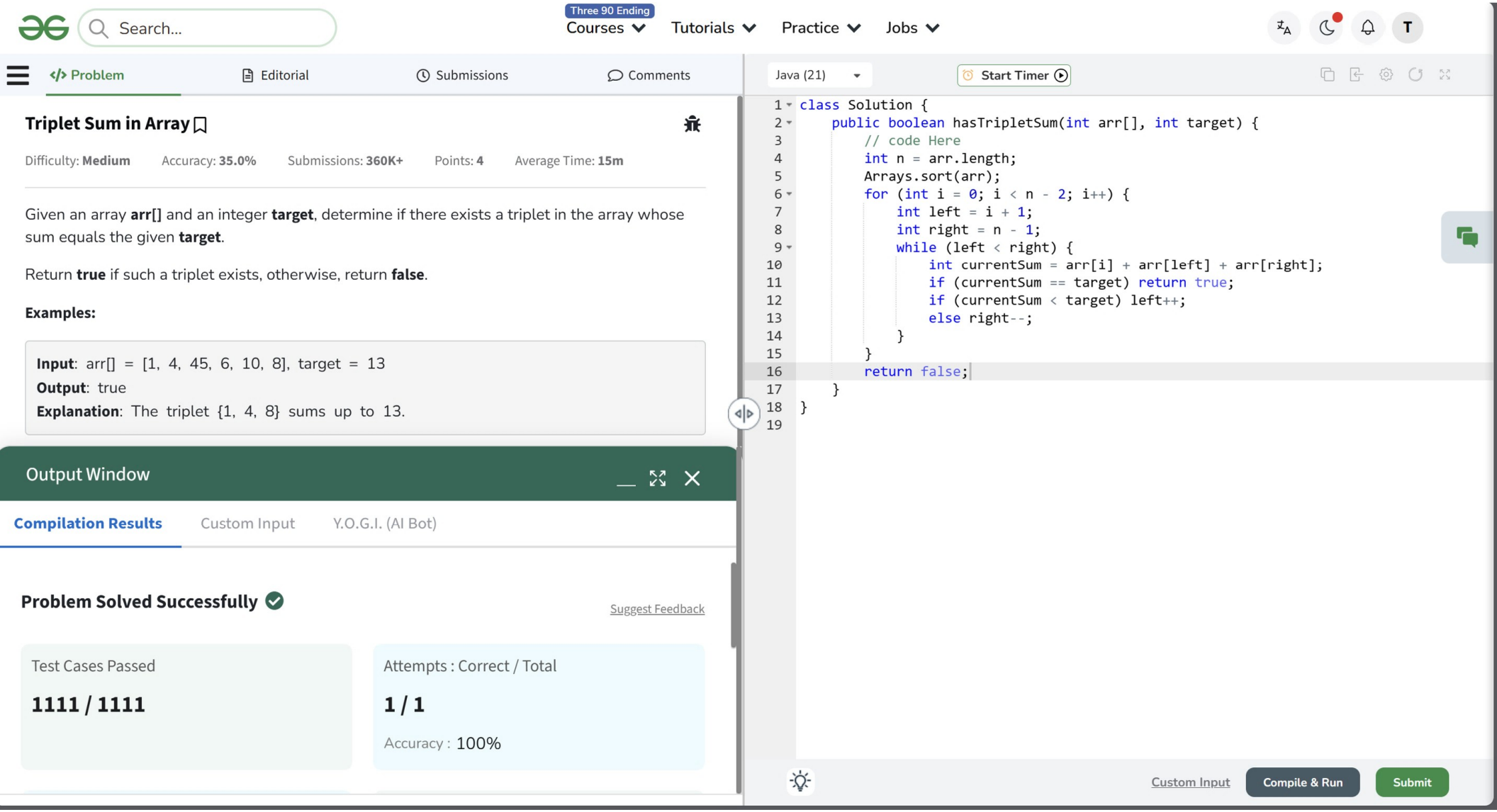

```java
// User function Template for Java

class Solution {
    public static ArrayList<Integer> factorial(int n) {
        ArrayList<Integer> result = new ArrayList<>();
        result.add(1);

        for (int i = 2; i <= n; i++) {
            int carry = 0;
            for (int j = 0; j < result.size(); j++) {
                int product = result.get(j) * i + carry;
                result.set(j, product % 10);
                carry = product / 10;
            }
            while (carry != 0) {
                result.add(carry % 10);
                carry = carry / 10;
            }
        }

        Collections.reverse(result);
        return result;
    }
}
```


Search...
Three 90 Ending
Courses
Tutorials
Practice
Jobs
T
Problem
Editorial
Submissions
Comments
Java (21)
Start Timer
Array Subset
Difficulty: Basic
Accuracy: 44.05%
Submissions: 517K+
Points: 1
Average Time: 20m
Given two arrays a[] and b[], your task is to determine whether b[] is a subset of a[].
Examples:
Input: a[] = [11, 7, 1, 13, 21, 3, 7, 3], b[] = [11, 3, 7, 1, 7]
Output: true
Explanation: b[] is a subset of a[]
Input: a[] = [1, 2, 3, 4, 4, 5, 6], b[] = [1, 2, 4]
class Solution {
public boolean isSubset(int a[], int b[]) {
HashMap<Integer, Integer> map = new HashMap<>();
for (int x : a) {
map.put(x, map.getOrDefault(x, 0) + 1);
}
for (int x : b) {
if (!map.containsKey(x) || map.get(x) == 0) return false;
map.put(x, map.get(x) - 1);
}
return true;
}
}
Output Window
Compilation Results
Custom Input
Y.O.G.I. (AI Bot)
Problem Solved Successfully
Suggest Feedback
Test Cases Passed
1114 / 1114
Attempts : Correct / Total
1 / 1
Accuracy : 100%
Custom Input
Compile & Run
Submit


Search...
Three 90 Ending
Courses
Tutorials
Practice
Jobs
T
Problem
Editorial
Submissions
Comments
Triplet Sum in Array
Difficulty: Medium
Accuracy: 35.0%
Submissions: 360K+
Points: 4
Average Time: 15m
Given an array arr[] and an integer target, determine if there exists a triplet in the array whose sum equals the given target.
Return true if such a triplet exists, otherwise, return false.
Examples:
Input: arr[] = [1, 4, 45, 6, 10, 8], target = 13
Output: true
Explanation: The triplet {1, 4, 8} sums up to 13.
Output Window
Compilation Results
Custom Input
Y.O.G.I. (AI Bot)
Problem Solved Successfully
Suggest Feedback
Test Cases Passed
1111 / 1111
Attempts : Correct / Total
1 / 1
Accuracy : 100%
Java (21)
Start Timer
class Solution {
public boolean hasTripletSum(int arr[], int target) {
// code Here
int n = arr.length;
Arrays.sort(arr);
for (int i = 0; i < n - 2; i++) {
int left = i + 1;
int right = n - 1;
while (left < right) {
int currentSum = arr[i] + arr[left] + arr[right];
if (currentSum == target) return true;
if (currentSum < target) left++;
else right--;
}
}
return false;
}
}
Custom Input
Compile & Run
Submit