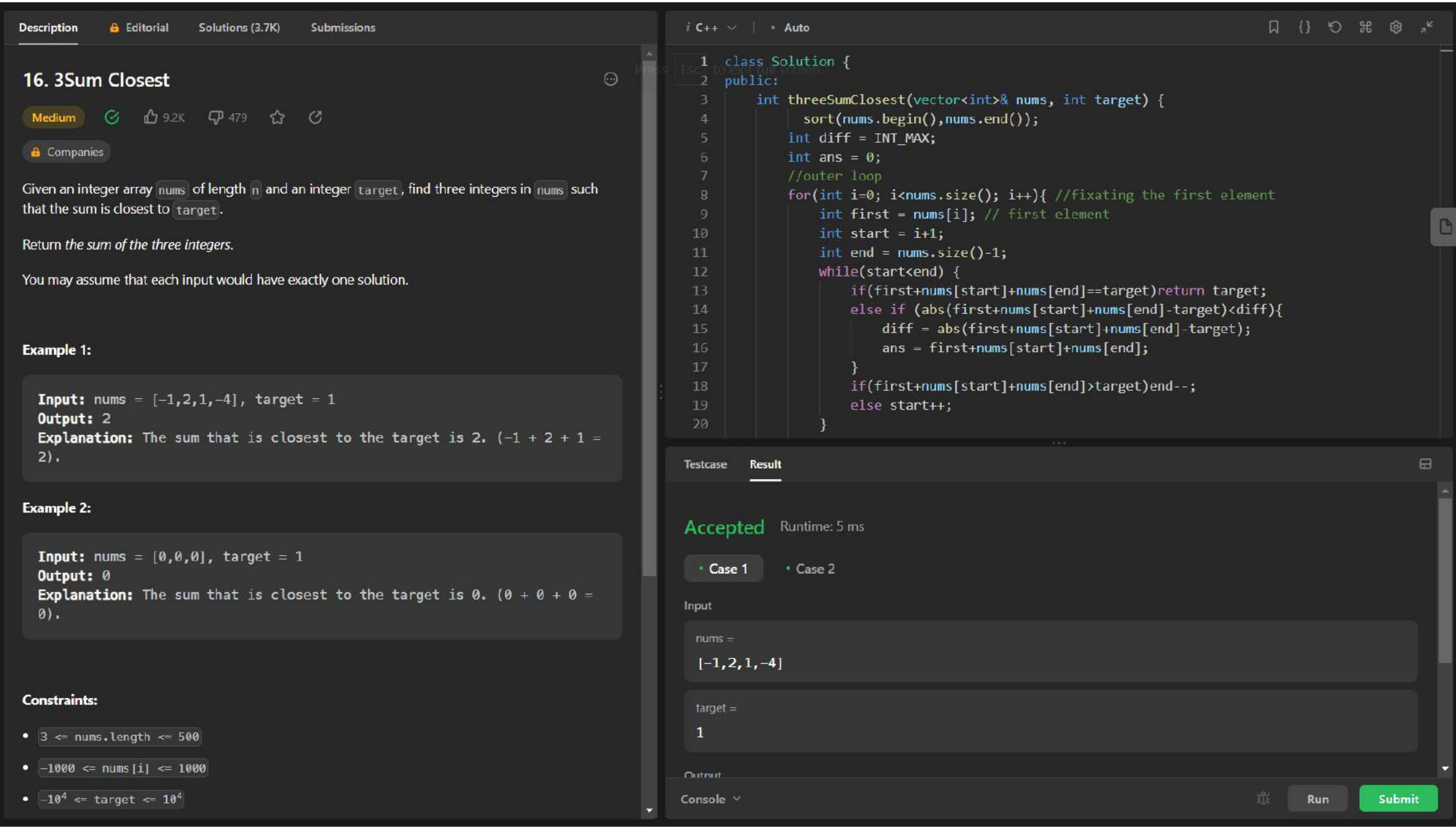


16. 3Sum Closest

Given an integer array `nums` of length `n` and an integer `target`, find three integers in `nums` such that the sum is closest to `target`.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

Example 1:

```
Input: nums = [-1,2,1,-4], target = 1
Output: 2
Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 =
2).
```

Example 2:

```
Input: nums = [0,0,0], target = 1
Output: 0
Explanation: The sum that is closest to the target is 0. (0 + 0 + 0 =
0).
```

Constraints:

- `3 <= nums.length <= 500`
- `-1000 <= nums[i] <= 1000`
- `-10^4 <= target <= 10^4`

```cpp
class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        sort(nums.begin(),nums.end());
        int diff = INT_MAX;
        int ans = 0;
        //outer loop
        for(int i=0; i<nums.size(); i++){ //fixating the first element
            int first = nums[i]; // first element
            int start = i+1;
            int end = nums.size()-1;
            while(start<end) {
                if(first+nums[start]+nums[end]==target)return target;
                else if (abs(first+nums[start]+nums[end]-target)<diff){
                    diff = abs(first+nums[start]+nums[end]-target);
                    ans = first+nums[start]+nums[end];
                }
                if(first+nums[start]+nums[end]>target)end--;
                else start++;
            }
```

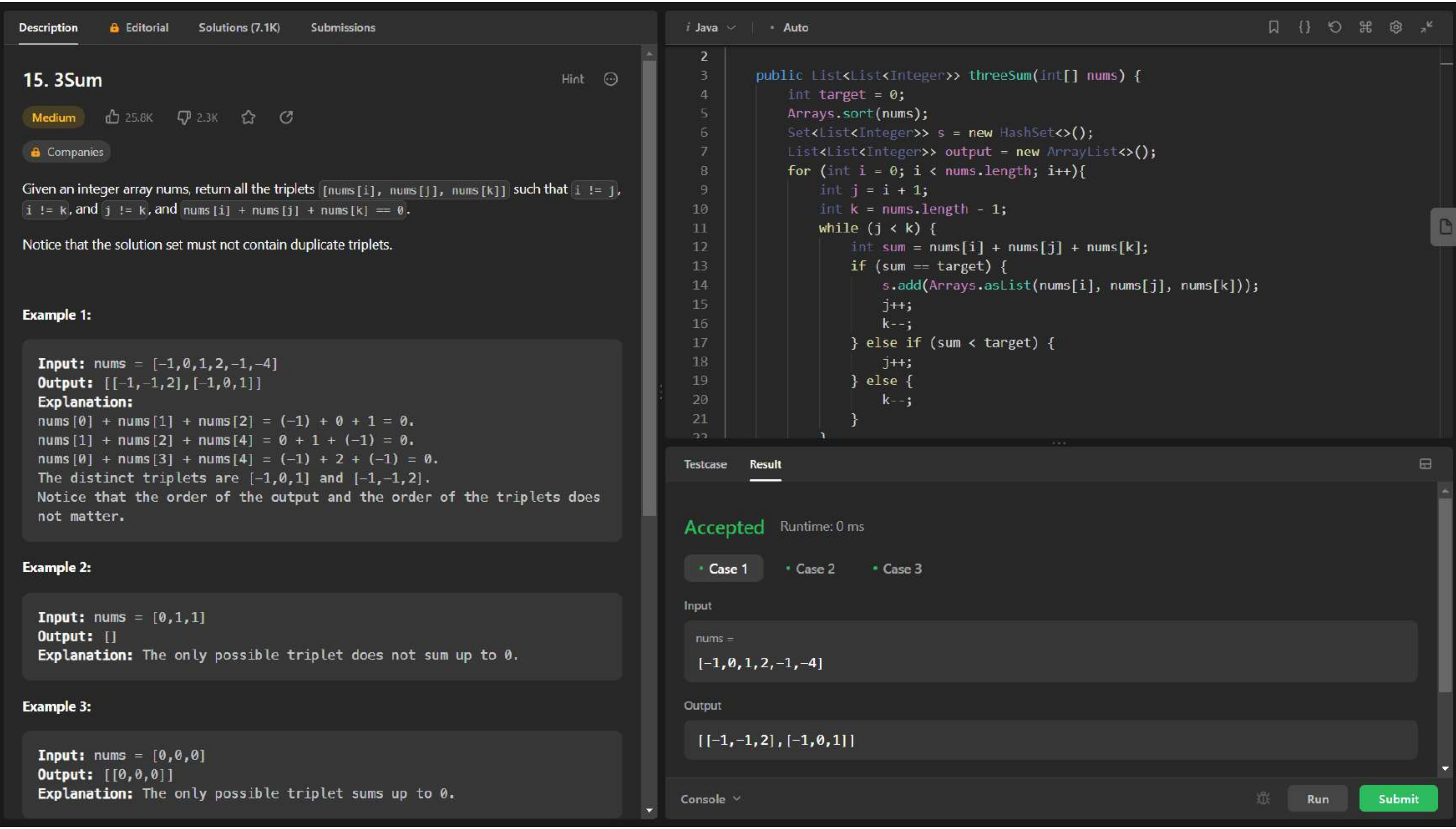
Accepted Runtime: 5 ms

Input

nums =
[-1,2,1,-4]

target =
1

Output



15. 3Sum

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does
not matter.
```

Example 2:

```
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.
```

Example 3:

```
Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.
```

```java

    public List<List<Integer>> threeSum(int[] nums) {
        int target = 0;
        Arrays.sort(nums);
        Set<List<Integer>> s = new HashSet<>();
        List<List<Integer>> output = new ArrayList<>();
        for (int i = 0; i < nums.length; i++){
            int j = i + 1;
            int k = nums.length - 1;
            while (j < k) {
                int sum = nums[i] + nums[j] + nums[k];
                if (sum == target) {
                    s.add(Arrays.asList(nums[i], nums[j], nums[k]));
                    j++;
                    k--;
                } else if (sum < target) {
                    j++;
                } else {
                    k--;
                }
            }
```

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

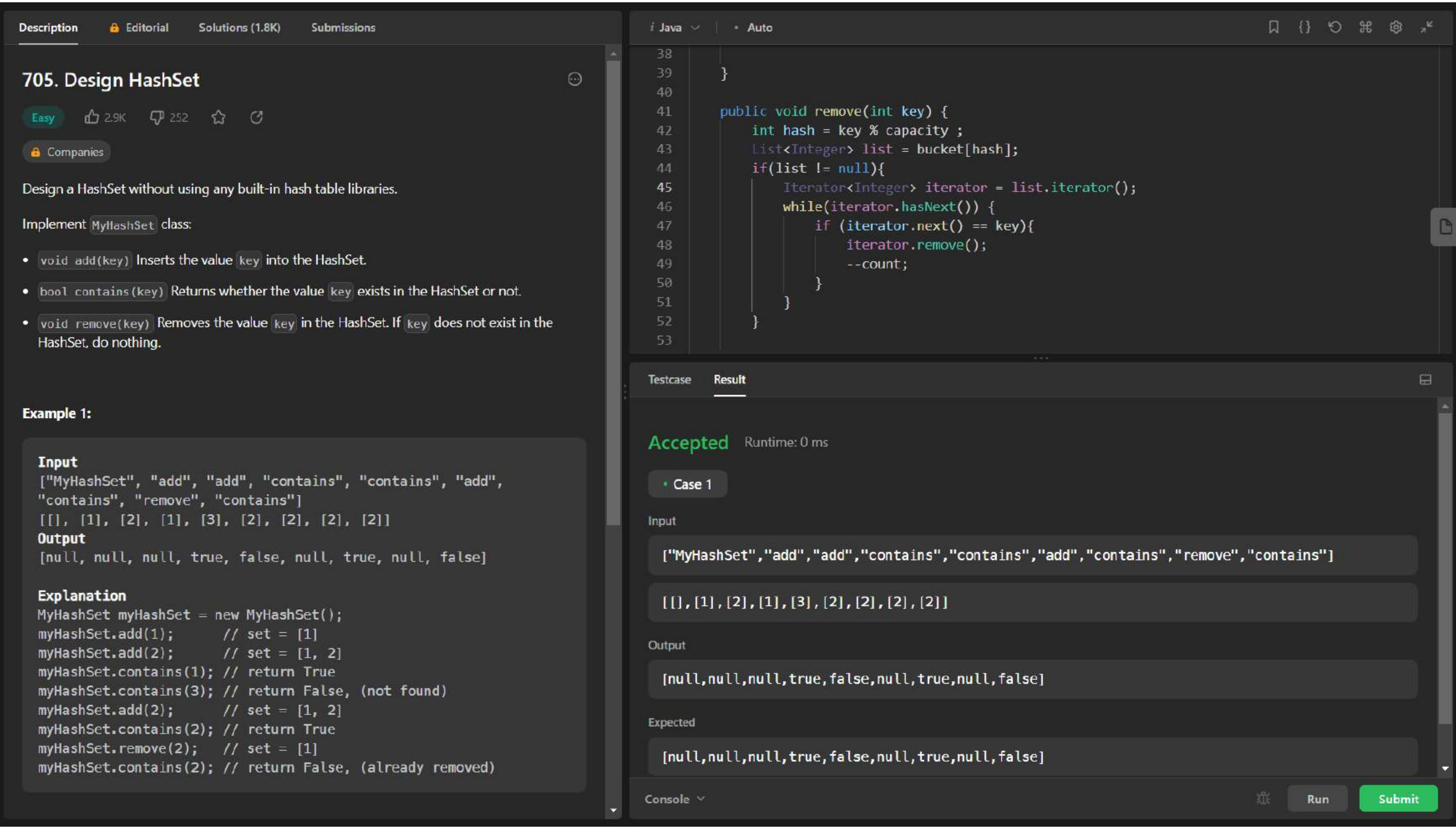
Input

nums =

```
[-1,0,1,2,-1,-4]
```

Output

```
[[-1,-1,2],[-1,0,1]]
```



705. Design HashSet

Design a HashSet without using any built-in hash table libraries.

Implement `MyHashSet` class:

- `void add(key)` Inserts the value `key` into the HashSet.
- `bool contains(key)` Returns whether the value `key` exists in the HashSet or not.
- `void remove(key)` Removes the value `key` in the HashSet. If `key` does not exist in the HashSet, do nothing.

Example 1:

```
Input
["MyHashSet", "add", "add", "contains", "contains", "add",
"contains", "remove", "contains"]
[[], [1], [2], [1], [3], [2], [2], [2], [2]]
Output
[null, null, null, true, false, null, true, null, false]

Explanation
MyHashSet myHashSet = new MyHashSet();
myHashSet.add(1);      // set = [1]
myHashSet.add(2);      // set = [1, 2]
myHashSet.contains(1); // return True
myHashSet.contains(3); // return False, (not found)
myHashSet.add(2);      // set = [1, 2]
myHashSet.contains(2); // return True
myHashSet.remove(2);   // set = [1]
myHashSet.contains(2); // return False, (already removed)
```

```java

    }

    public void remove(int key) {
        int hash = key % capacity ;
        List<Integer> list = bucket[hash];
        if(list != null){
            Iterator<Integer> iterator = list.iterator();
            while(iterator.hasNext()) {
                if (iterator.next() == key){
                    iterator.remove();
                    --count;
                }
            }
        }

```

Accepted Runtime: 0 ms

Case 1

Input

```
["MyHashSet","add","add","contains","contains","add","contains","remove","contains"]
```

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

Output

```
[null,null,null,true,false,null,true,null,false]
```

Expected

```
[null,null,null,true,false,null,true,null,false]
```