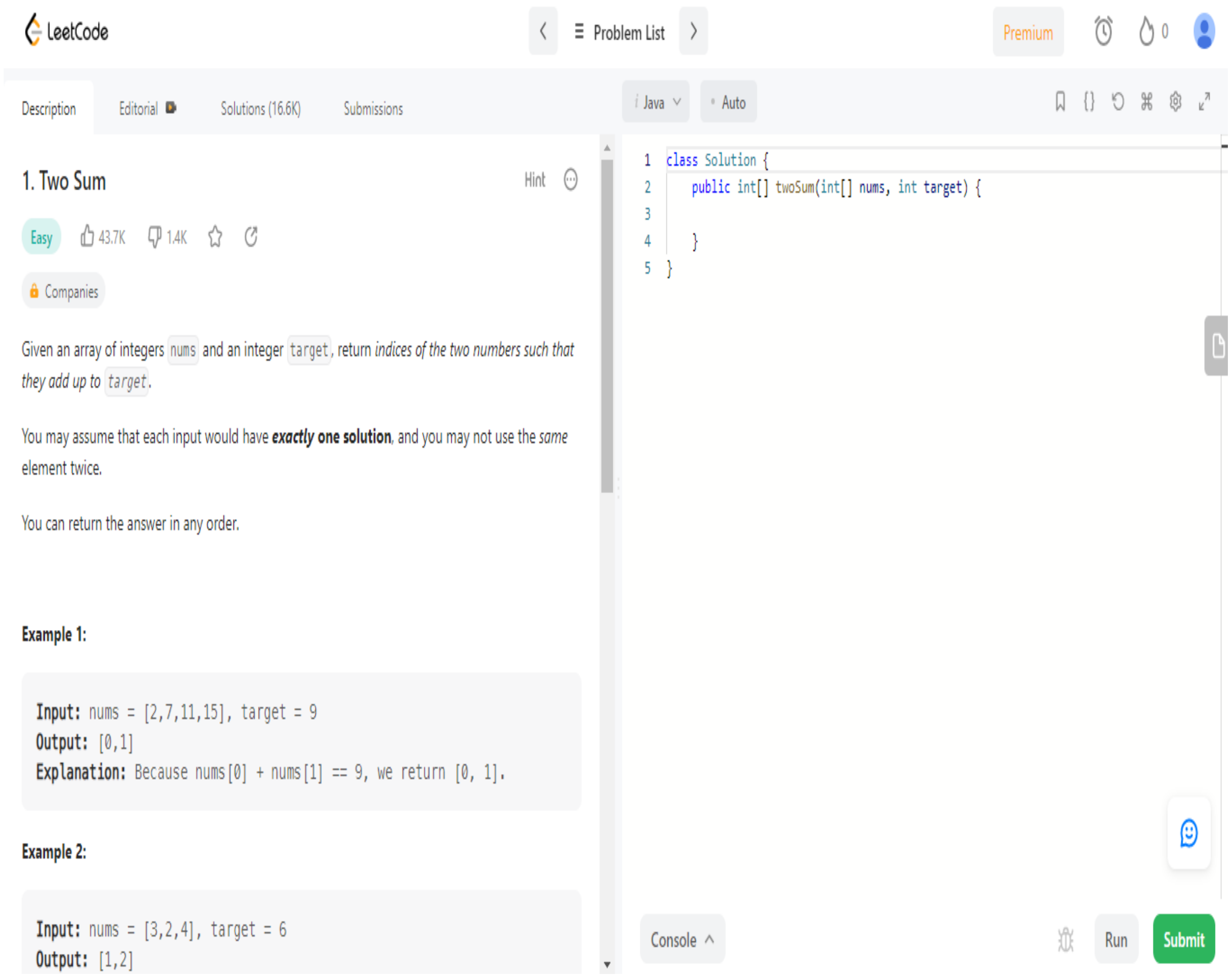
TWO SUM (EASY)



1. Two Sum

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {

    }
}
```

- **Approaches to solve problem**

  1- First over the brute force approach.

  - We will perform a nested ‘for’ loop

  For each of element ‘x’.

- We will want to perform another 'for' loop to find 'y', where 'x + y' is equal to 'target'.
- This approach will cost O(n^2) time complexity.

2- Optimized Approach

- For each of the element 'x'.
- We will want to check if 'target – x' exists.
- We will want a way to save the previous element we have seen before.
  We should note that we need to return the indices of the two numbers.
  This means we need to keep track of element and index.
  We can create a HashMap to map 'element -> index'.
- For each of the element 'x'.
  We can check our HashMap for 'target – x'.
- This approach will allow us to reduce the time complexity to o(n).

- Answer
  - Create a HashMap 'map' to keep track of element and index.
  - Iterate through the indices of the input array -- 'nums', denoted as 'I'.

```
If ‘map’ contains ‘target – nums[i]’
    Return { I, map.get(target – nums[i]) }

Put ‘nums[i]’ and ‘I’ into ‘map’.
```

- Return an empty array.

What is the Time and space Complexity?

- Time Complexity = O(n), where n is the length of the input array.
    - O(n), visit each index once
- Space Complexity = O(n), where n is the length of the input array
    - O(n), map