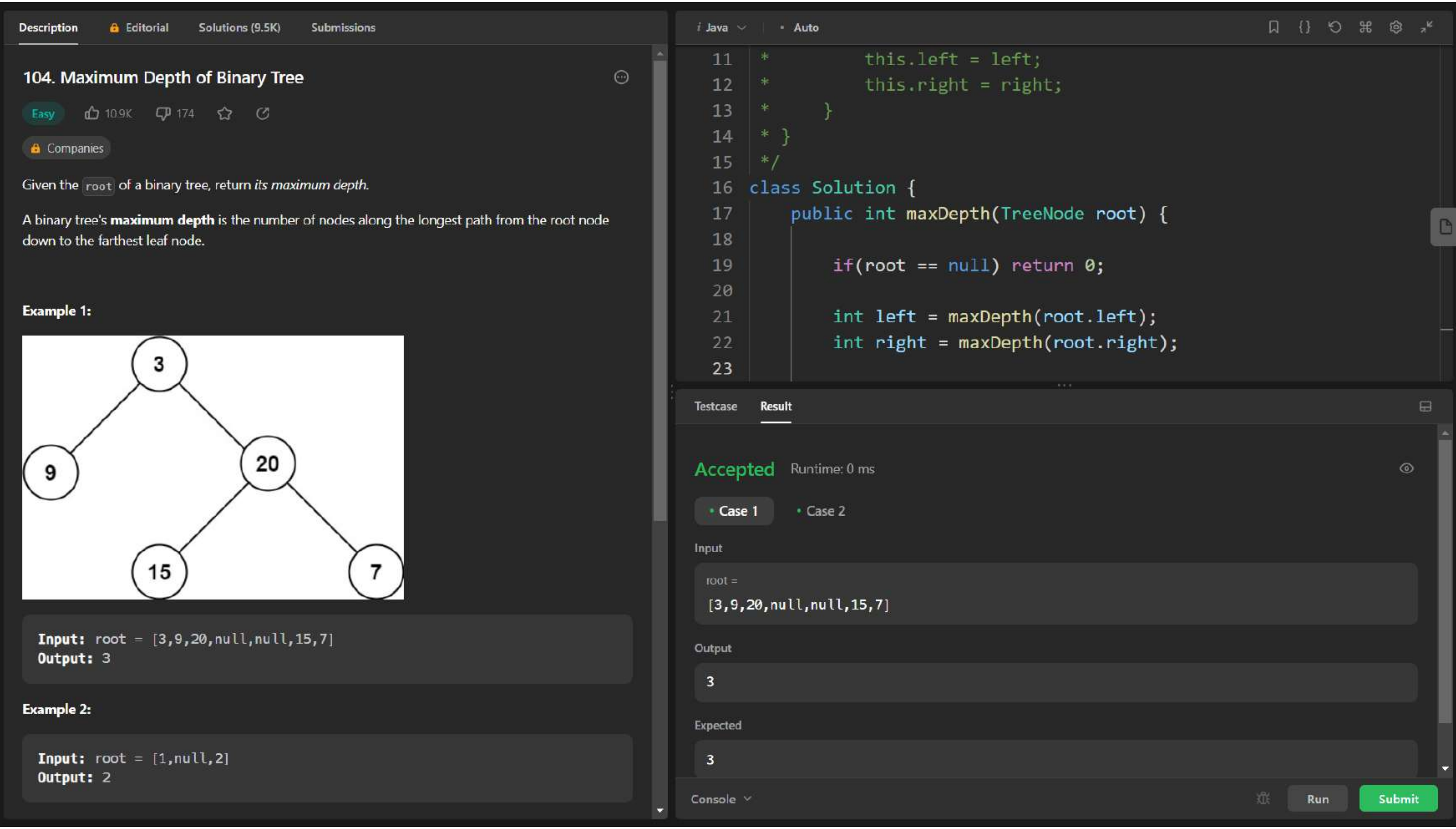


Description | Editorial | Solutions (9.5K) | Submissions

104. Maximum Depth of Binary Tree

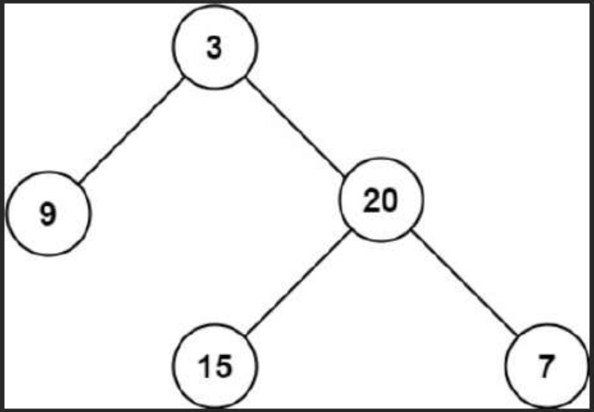
Easy 10.9K 174

Companies

Given the `root` of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:



```
Input: root = [3,9,20,null,null,15,7]
Output: 3
```

Example 2:

```
Input: root = [1,null,2]
Output: 2
```

Java | Auto

```java
 *          this.left = left;
 *          this.right = right;
 *      }
 * }
 */
class Solution {
    public int maxDepth(TreeNode root) {

        if(root == null) return 0;

        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

```
root =
[3,9,20,null,null,15,7]
```

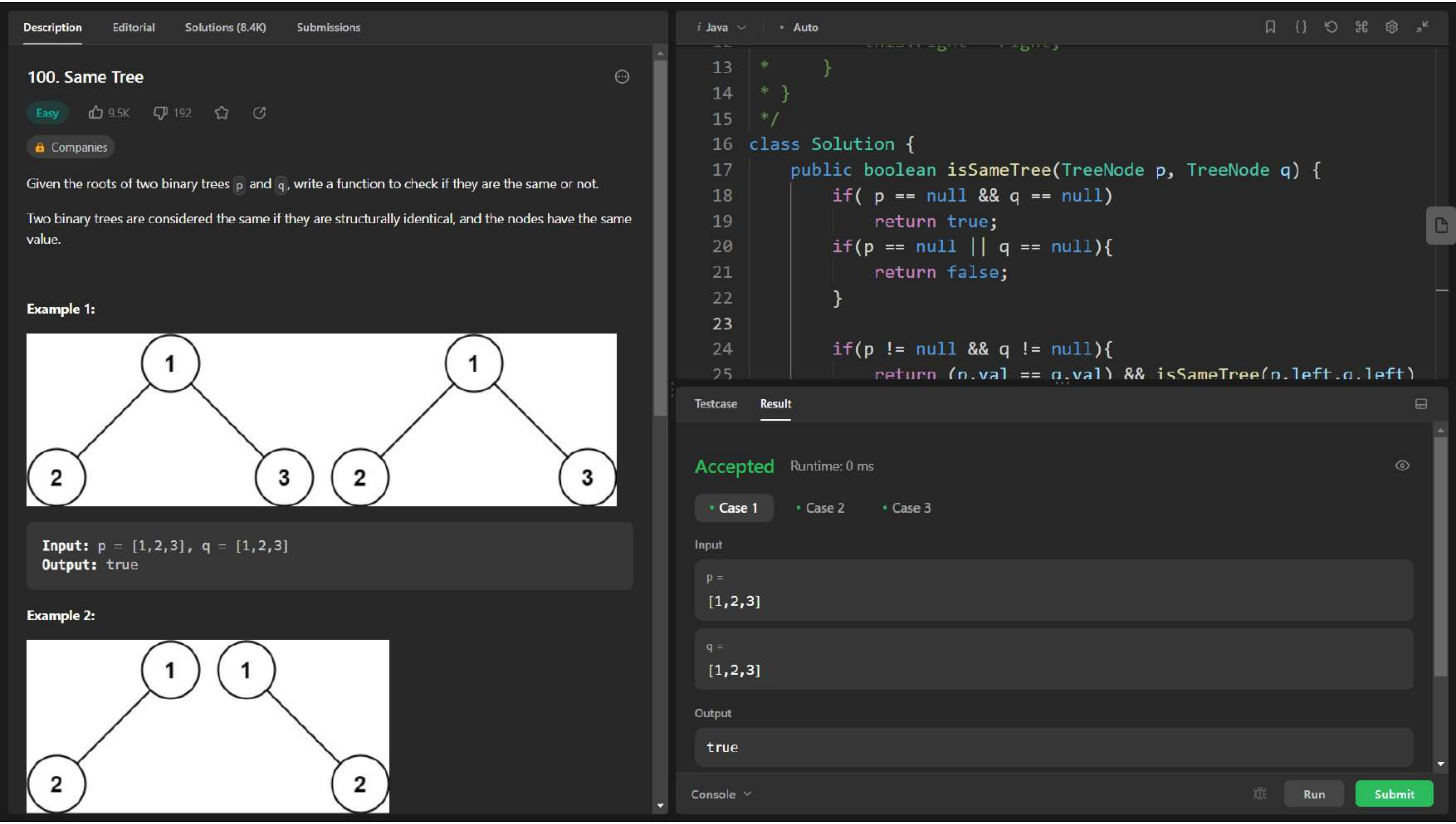
Output

```
3
```

Expected

```
3
```

Console | Run | Submit



100. Same Tree

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:

```
Input: p = [1,2,3], q = [1,2,3]
Output: true
```

Example 2:

```java
 *     }
 * }
 */
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if( p == null && q == null)
            return true;
        if(p == null || q == null){
            return false;
        }

        if(p != null && q != null){
            return (p.val == q.val) && isSameTree(p.left,q.left)
```

Accepted Runtime: 0 ms

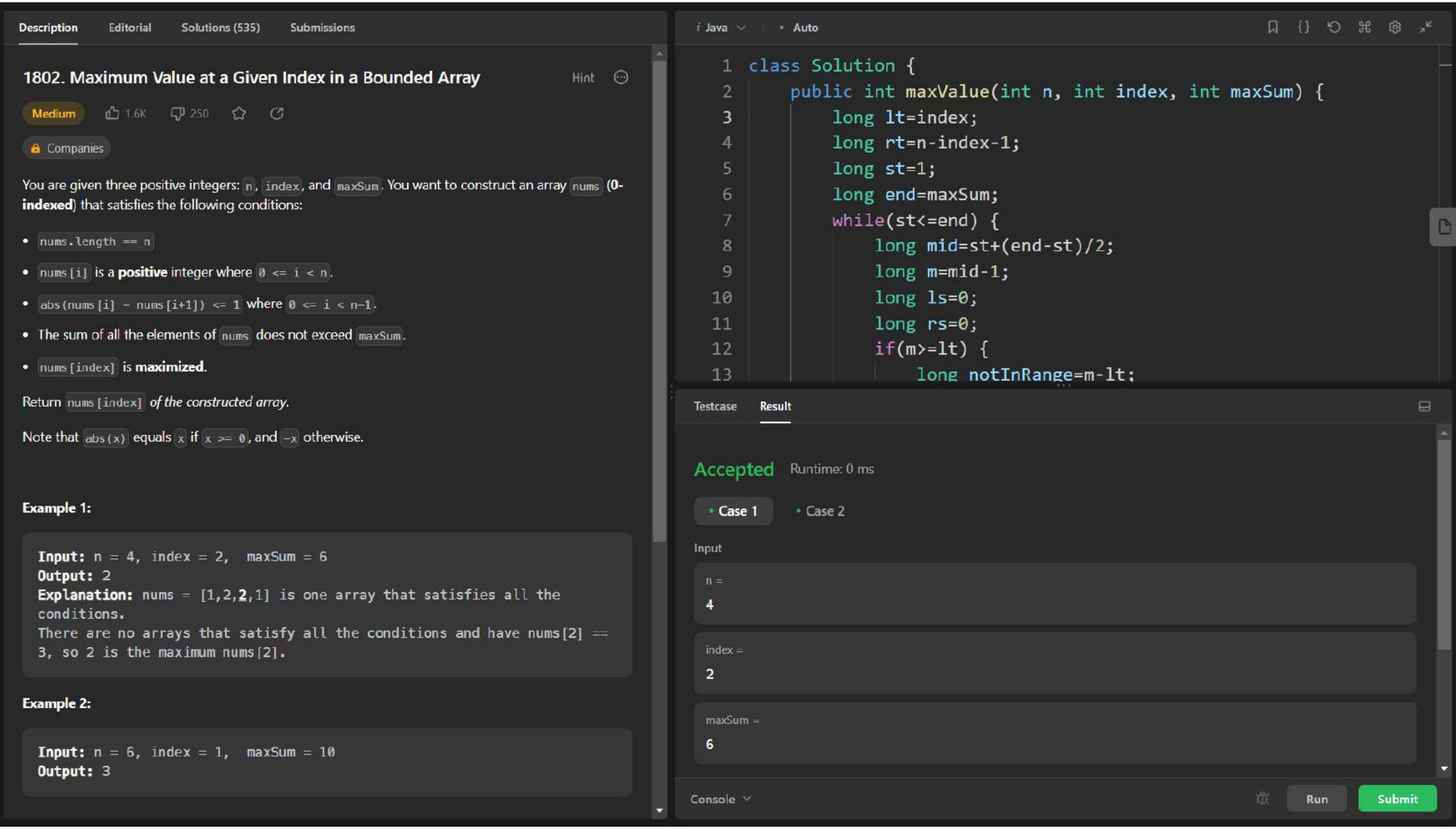
Input

```
p =
[1,2,3]
```

```
q =
[1,2,3]
```

Output

```
true
```



1802. Maximum Value at a Given Index in a Bounded Array

You are given three positive integers: `n`, `index`, and `maxSum`. You want to construct an array `nums` (**0-indexed**) that satisfies the following conditions:

- `nums.length == n`
- `nums[i]` is a **positive** integer where `0 <= i < n`.
- `abs(nums[i] - nums[i+1]) <= 1` where `0 <= i < n-1`.
- The sum of all the elements of `nums` does not exceed `maxSum`.
- `nums[index]` is **maximized**.

Return `nums[index]` *of the constructed array.*

Note that `abs(x)` equals `x` if `x >= 0`, and `-x` otherwise.

Example 1:

```
Input: n = 4, index = 2,  maxSum = 6
Output: 2
Explanation: nums = [1,2,2,1] is one array that satisfies all the
conditions.
There are no arrays that satisfy all the conditions and have nums[2] ==
3, so 2 is the maximum nums[2].
```

Example 2:

```
Input: n = 6, index = 1,  maxSum = 10
Output: 3
```

```java
class Solution {
    public int maxValue(int n, int index, int maxSum) {
        long lt=index;
        long rt=n-index-1;
        long st=1;
        long end=maxSum;
        while(st<=end) {
            long mid=st+(end-st)/2;
            long m=mid-1;
            long ls=0;
            long rs=0;
            if(m>=lt) {
                long notInRange=m-lt;
```

Testcase Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
n =
4
index =
2
maxSum =
6
```