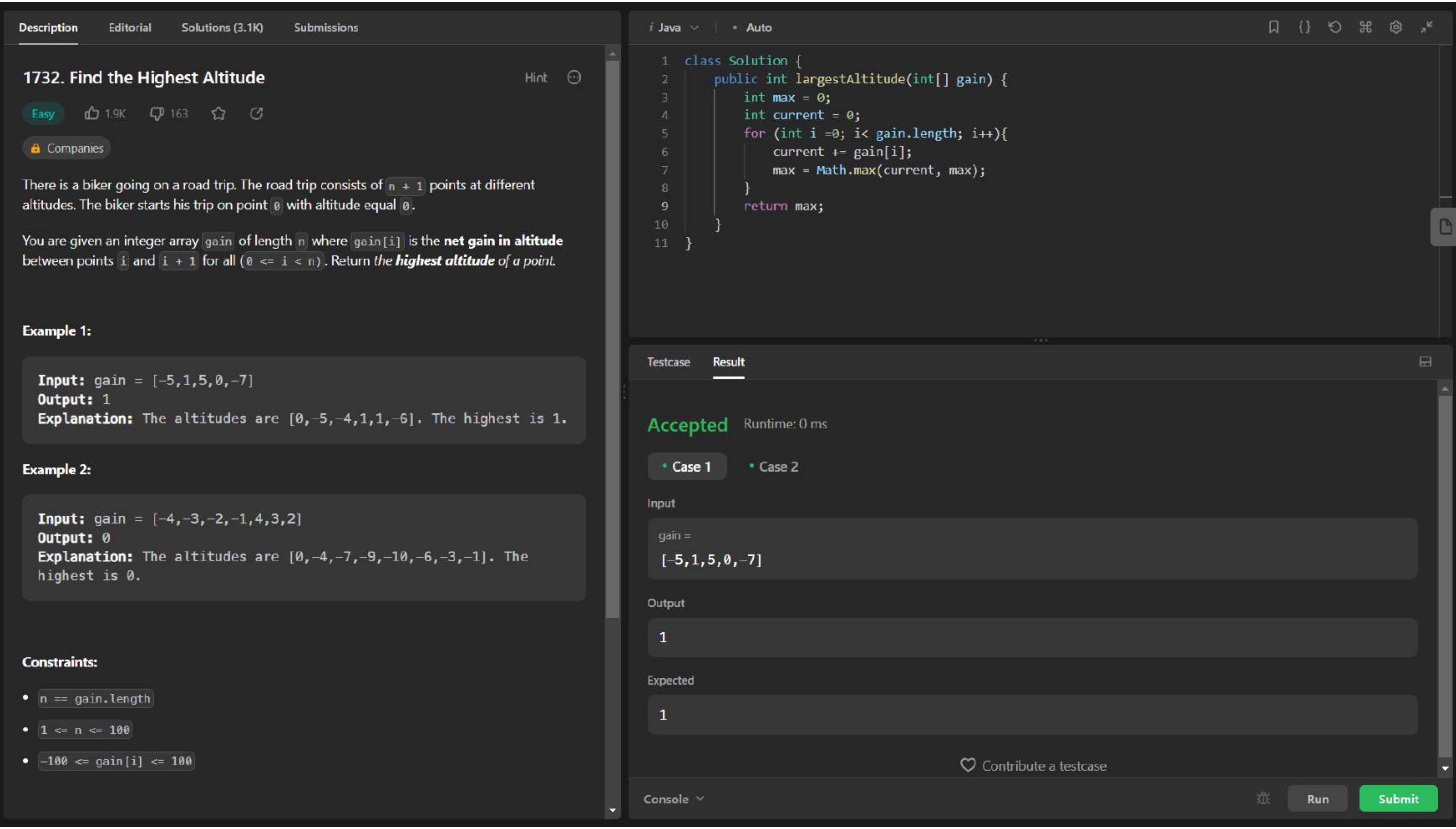


1732. Find the Highest Altitude

There is a biker going on a road trip. The road trip consists of `n + 1` points at different altitudes. The biker starts his trip on point `0` with altitude equal `0`.

You are given an integer array `gain` of length `n` where `gain[i]` is the **net gain in altitude** between points `i` and `i + 1` for all (`0 <= i < n)`. Return *the **highest altitude** of a point.*

Example 1:

```
Input: gain = [-5,1,5,0,-7]
Output: 1
Explanation: The altitudes are [0,-5,-4,1,1,-6]. The highest is 1.
```

Example 2:

```
Input: gain = [-4,-3,-2,-1,4,3,2]
Output: 0
Explanation: The altitudes are [0,-4,-7,-9,-10,-6,-3,-1]. The
highest is 0.
```

Constraints:

- `n == gain.length`
- `1 <= n <= 100`
- `-100 <= gain[i] <= 100`

```java
class Solution {
    public int largestAltitude(int[] gain) {
        int max = 0;
        int current = 0;
        for (int i =0; i< gain.length; i++){
            current += gain[i];
            max = Math.max(current, max);
        }
        return max;
    }
}
```

Accepted Runtime: 0 ms

Input

gain =

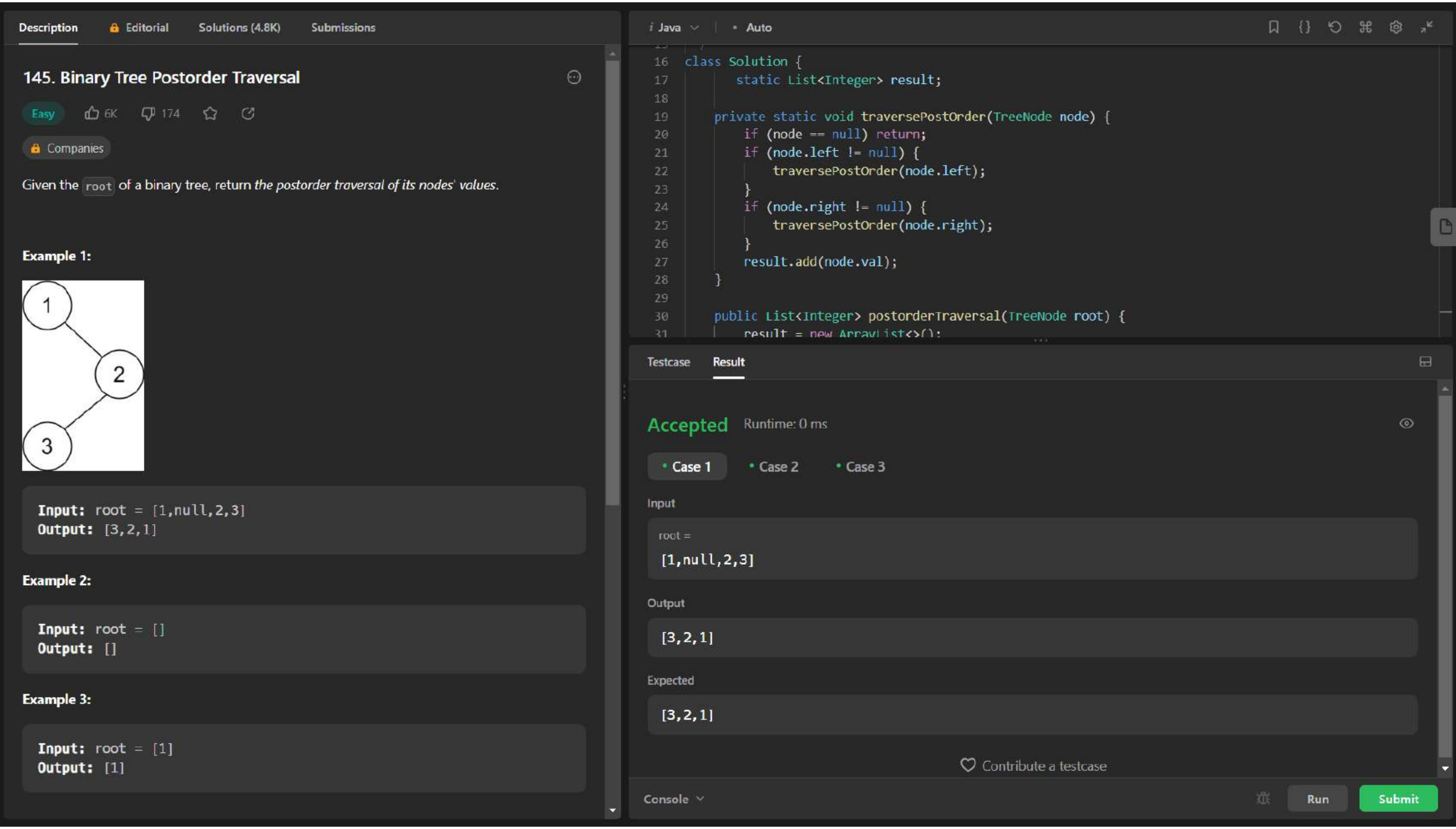
[-5,1,5,0,-7]

Output

1

Expected

1



Description | Editorial | Solutions (4.8K) | Submissions

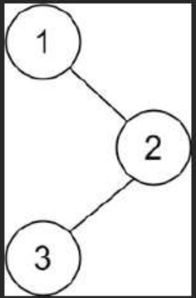
145. Binary Tree Postorder Traversal

Easy 6K 174

Companies

Given the `root` of a binary tree, return *the postorder traversal of its nodes' values.*

Example 1:



```
Input: root = [1,null,2,3]
Output: [3,2,1]
```

Example 2:

```
Input: root = []
Output: []
```

Example 3:

```
Input: root = [1]
Output: [1]
```

Java | Auto

```java
class Solution {
        static List<Integer> result;

    private static void traversePostOrder(TreeNode node) {
        if (node == null) return;
        if (node.left != null) {
            traversePostOrder(node.left);
        }
        if (node.right != null) {
            traversePostOrder(node.right);
        }
        result.add(node.val);
    }

    public List<Integer> postorderTraversal(TreeNode root) {
        result = new ArrayList<>();
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

root =

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

Output

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

Expected

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

Contribute a testcase

Console | Run | Submit