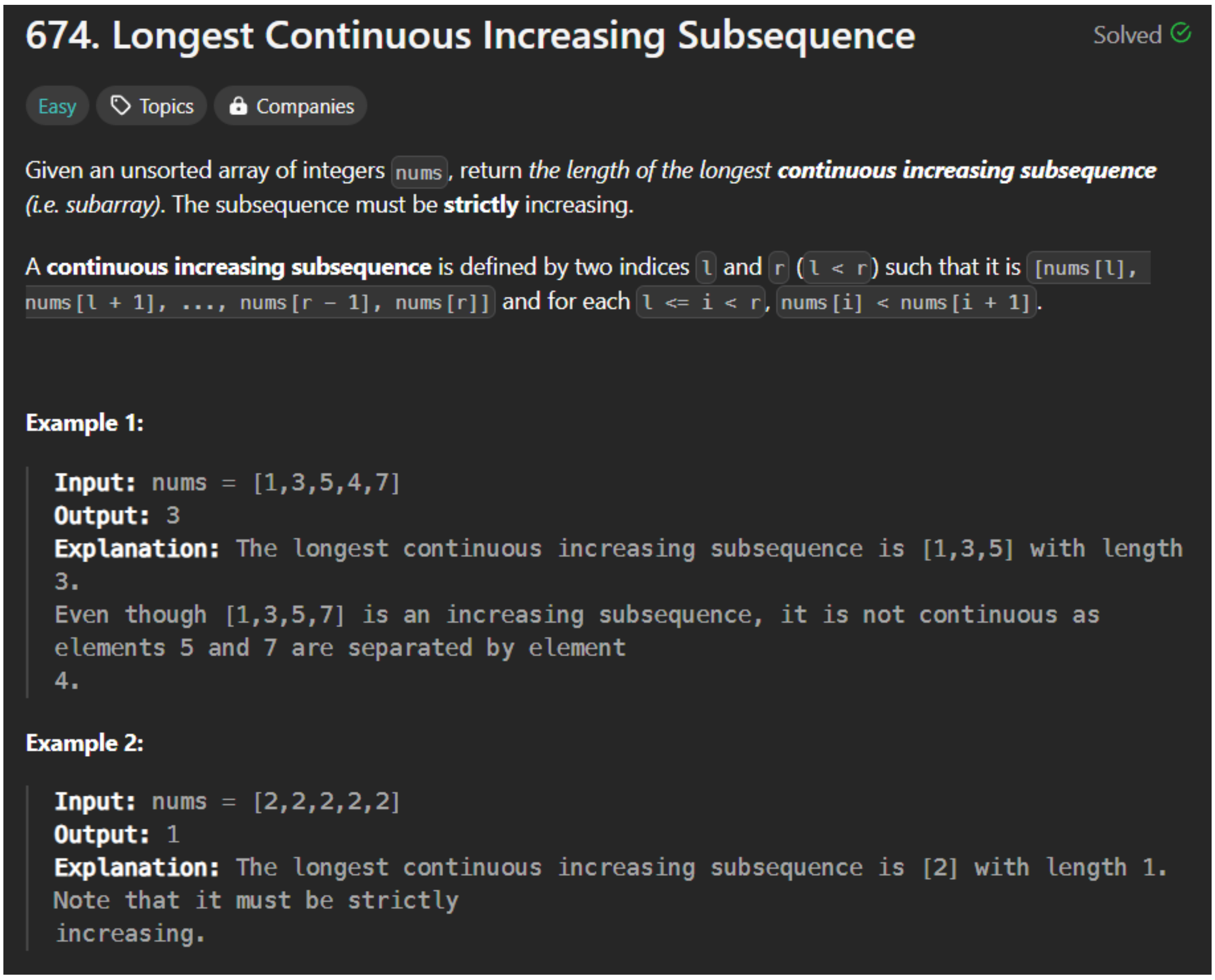


674. Longest Continuous Increasing Subsequence

Given an unsorted array of integers `nums`, return *the length of the longest* ***continuous increasing subsequence*** *(i.e. subarray)*. The subsequence must be **strictly** increasing.

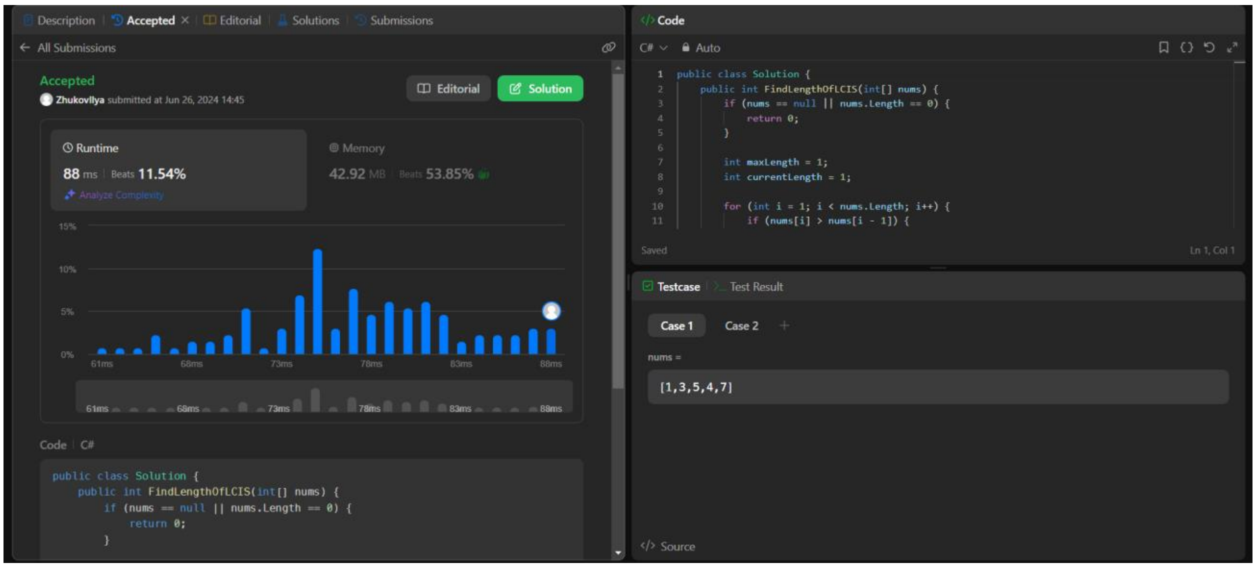
A **continuous increasing subsequence** is defined by two indices `l` and `r` (`l < r`) such that it is `[nums[l], nums[l + 1], ..., nums[r - 1], nums[r]]` and for each `l <= i < r`, `nums[i] < nums[i + 1]`.

Example 1:

```
Input: nums = [1,3,5,4,7]
Output: 3
Explanation: The longest continuous increasing subsequence is [1,3,5] with length
3.
Even though [1,3,5,7] is an increasing subsequence, it is not continuous as
elements 5 and 7 are separated by element
4.
```

Example 2:

```
Input: nums = [2,2,2,2,2]
Output: 1
Explanation: The longest continuous increasing subsequence is [2] with length 1.
Note that it must be strictly
increasing.
```



Accepted
Zhukovllya submitted at Jun 26, 2024 14:45

Runtime
88 ms | Beats 11.54%

Memory
42.92 MB | Beats 53.85%

Code | C#

```
public class Solution {
    public int FindLengthOfLCIS(int[] nums) {
        if (nums == null || nums.Length == 0) {
            return 0;
        }
```

```
public class Solution {
    public int FindLengthOfLCIS(int[] nums) {
        if (nums == null || nums.Length == 0) {
            return 0;
        }

        int maxLength = 1;
        int currentLength = 1;

        for (int i = 1; i < nums.Length; i++) {
            if (nums[i] > nums[i - 1]) {
```

Saved
Ln 1, Col 1

nums =

```
[1,3,5,4,7]
```

Код:

```
public class Solution
{
    public int FindLengthOfLCIS(int[] nums)
    {
        if (nums == null || nums.Length == 0)
        {
            return 0;
        }
        int maxLength = 1;
        int currentLength = 1;

        for (int i = 1; i < nums.Length; i++)
        {
            if (nums[i] > nums[i - 1])
            {
                currentLength++;
                if (currentLength > maxLength)
                {
                    maxLength = currentLength;
                }
            }
            else
            {
                currentLength = 1;
            }
        }

        return maxLength;
    }
}
```