Problem 674: Longest Continuous Increasing Subsequence

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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.

Constraints:

1 <= nums.length <= 10

4

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int findLengthOfLCIS(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int findLengthOfLCIS(int[] nums) {

}
}
```

Python3:

```
class Solution:
def findLengthOfLCIS(self, nums: List[int]) -> int:
```

Python:

```
class Solution(object):
def findLengthOfLCIS(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
var findLengthOfLCIS = function(nums) {

};
```

TypeScript:

```
function findLengthOfLCIS(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int FindLengthOfLCIS(int[] nums) {
```

```
}
}
```

C:

```
int findLengthOfLCIS(int* nums, int numsSize) {

}
```

Go:

```
func findLengthOfLCIS(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun findLengthOfLCIS(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func findLengthOfLCIS(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_length_of_lcis(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def find_length_of_lcis(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function findLengthOfLCIS($nums) {

}
}
```

Dart:

```
class Solution {
int findLengthOfLCIS(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def findLengthOfLCIS(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_length_of_lcis(nums :: [integer]) :: integer
def find_length_of_lcis(nums) do

end
end
```

Erlang:

```
-spec find_length_of_lcis(Nums :: [integer()]) -> integer().
find_length_of_lcis(Nums) ->
.
```

Racket:

```
(define/contract (find-length-of-lcis nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Continuous Increasing Subsequence
* Difficulty: Easy
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int findLengthOfLCIS(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Continuous Increasing Subsequence
* Difficulty: Easy
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int findLengthOfLCIS(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Continuous Increasing Subsequence
Difficulty: Easy
Tags: array, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def findLengthOfLCIS(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findLengthOfLCIS(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Continuous Increasing Subsequence
* Difficulty: Easy
```

```
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @return {number}
*/
var findLengthOfLCIS = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Continuous Increasing Subsequence
* Difficulty: Easy
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function findLengthOfLCIS(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Continuous Increasing Subsequence
* Difficulty: Easy
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

public class Solution {
public int FindLengthOfLCIS(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Longest Continuous Increasing Subsequence
* Difficulty: Easy
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int findLengthOfLCIS(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Longest Continuous Increasing Subsequence
// Difficulty: Easy
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func findLengthOfLCIS(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findLengthOfLCIS(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func findLengthOfLCIS(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Continuous Increasing Subsequence
// Difficulty: Easy
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn find_length_of_lcis(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def find_length_of_lcis(nums)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @return Integer
*/
function findLengthOfLCIS($nums) {

}
}
```

Dart Solution:

```
class Solution {
int findLengthOfLCIS(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def findLengthOfLCIS(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_length_of_lcis(nums :: [integer]) :: integer
def find_length_of_lcis(nums) do

end
end
```

Erlang Solution:

```
-spec find_length_of_lcis(Nums :: [integer()]) -> integer().
find_length_of_lcis(Nums) ->
.
```

Racket Solution:

```
(define/contract (find-length-of-lcis nums)
(-> (listof exact-integer?) exact-integer?)
)
```