Problem 300: Longest Increasing Subsequence

Problem Information

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

Problem Description

Given an integer array

nums

, return

the length of the longest

strictly increasing

subsequence

.

Example 1:

Input:

nums = [10,9,2,5,3,7,101,18]

Output:

4

Explanation:

The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

Example 2:

Input:

nums = [0,1,0,3,2,3]

Output:

4

Example 3:

Input:

nums = [7,7,7,7,7,7,7]

Output:

1

Constraints:

1 <= nums.length <= 2500

-10

4

<= nums[i] <= 10

4

Follow up:

Can you come up with an algorithm that runs in

O(n log(n))

time complexity?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
```

```
var lengthOfLIS = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Longest Increasing Subsequence
Difficulty: Medium
Tags: array, dp, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

```
};
```

C# Solution:

```
/*
* Problem: Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int lengthOfLIS(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Longest Increasing Subsequence
// Difficulty: Medium
```

```
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func lengthOfLIS(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Increasing Subsequence
// Difficulty: Medium
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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