Problem 673: Number of Longest Increasing Subsequence

Problem Information

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

Problem Description

Given an integer array

nums

, return

the number of longest increasing subsequences.

Notice

that the sequence has to be

strictly

increasing.

Example 1:

Input:

nums = [1,3,5,4,7]

Output:

2

Explanation:

The two longest increasing subsequences are [1, 3, 4, 7] and [1, 3, 5, 7].

Example 2:

Input:

nums = [2,2,2,2,2]

Output:

5

Explanation:

The length of the longest increasing subsequence is 1, and there are 5 increasing subsequences of length 1, so output 5.

Constraints:

1 <= nums.length <= 2000

-10

6

<= nums[i] <= 10

6

The answer is guaranteed to fit inside a 32-bit integer.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Number of Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, tree, dp
*
* 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 findNumberOfLIS(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, tree, dp
*
* 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 findNumberOfLIS(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Longest Increasing Subsequence
Difficulty: Medium
Tags: array, tree, dp

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 findNumberOfLIS(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, tree, dp
*
* 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 findNumberOfLIS = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, tree, dp
*
* 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 findNumberOfLIS(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, tree, dp
*
```

```
* 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 FindNumberOfLIS(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Longest Increasing Subsequence
* Difficulty: Medium
* Tags: array, tree, dp
*
* 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 findNumberOfLIS(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Number of Longest Increasing Subsequence
// Difficulty: Medium
// Tags: array, tree, dp
//
// 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 findNumberOfLIS(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Longest Increasing Subsequence
// Difficulty: Medium
// Tags: array, tree, dp
//
// 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 find_number_of_lis(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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