Problem 2289: Steps to Make Array Non-decreasing

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. In one step,

remove

all elements

nums[i]

where

nums[i - 1] > nums[i]

for all

0 < i < nums.length

.

Return

the number of steps performed until

nums

becomes a

non-decreasing

array

.

Example 1:

Input:

nums = [5,3,4,4,7,3,6,11,8,5,11]

Output:

3

Explanation:

The following are the steps performed: - Step 1: [5,

3

,4,4,7,

3

,6,11,

8

,

5

,11] becomes [5,4,4,7,6,11,11] - Step 2: [5,

4

,4,7,

6

,11,11] becomes [5,4,7,11,11] - Step 3: [5,

4

,7,11,11] becomes [5,7,11,11] [5,7,11,11] is a non-decreasing array. Therefore, we return 3.

Example 2:

Input:

nums = [4,5,7,7,13]

Output:

0

Explanation:

nums is already a non-decreasing array. Therefore, we return 0.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var totalSteps = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (total-steps nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Steps to Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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 totalSteps(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Steps to Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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 totalSteps(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Steps to Make Array Non-decreasing
Difficulty: Medium
Tags: array, linked_list, stack

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Steps to Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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 totalSteps = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Steps to Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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 totalSteps(nums: number[]): number {
};
```

C# Solution:

```
/*
* Problem: Steps to Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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 TotalSteps(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Steps to Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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 totalSteps(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Steps to Make Array Non-decreasing
// Difficulty: Medium
// Tags: array, linked_list, stack
//
// 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 totalSteps(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Steps to Make Array Non-decreasing
// Difficulty: Medium
// Tags: array, linked_list, stack
//
// 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 total_steps(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (total-steps nums)
(-> (listof exact-integer?) exact-integer?)
)
```