Problem 3523: Make Array Non-decreasing

Problem Information

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

Problem Description

You are given an integer array

nums

. In one operation, you can select a

subarray

and replace it with a single element equal to its

maximum

value.

Return the

maximum possible size

of the array after performing zero or more operations such that the resulting array is

non-decreasing

.

Example 1:

Input:

nums = [4,2,5,3,5]

Output:

3

Explanation:

One way to achieve the maximum size is:

Replace subarray

nums[1..2] = [2, 5]

with

5

$\rightarrow$

[4, 5, 3, 5]

.

Replace subarray

nums[2..3] = [3, 5]

with

5

$\rightarrow$

[4, 5, 5]

.

The final array

[4, 5, 5]

is non-decreasing with size

3.

Example 2:

Input:

nums = [1,2,3]

Output:

3

Explanation:

No operation is needed as the array

[1,2,3]

is already non-decreasing.

Constraints:

1 <= nums.length <= 2 * 10

5

1 <= nums[i] <= 2 * 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-possible-size nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, greedy, 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 maximumPossibleSize(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, greedy, 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 maximumPossibleSize(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Make Array Non-decreasing
Difficulty: Medium
Tags: array, greedy, 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 maximumPossibleSize(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, greedy, 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 maximumPossibleSize = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, greedy, 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 maximumPossibleSize(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, greedy, 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 MaximumPossibleSize(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Make Array Non-decreasing
* Difficulty: Medium
* Tags: array, greedy, 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 maximumPossibleSize(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Make Array Non-decreasing
// Difficulty: Medium
// Tags: array, greedy, 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 maximumPossibleSize(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Make Array Non-decreasing
// Difficulty: Medium
// Tags: array, greedy, 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 maximum_possible_size(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-possible-size nums)
(-> (listof exact-integer?) exact-integer?)
)
```