Problem 3738: Longest Non-Decreasing Subarray After Replacing at Most One Element

Problem Information

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

Problem Description

You are given an integer array

nums

.

You are allowed to replace

at most

one element in the array with any other integer value of your choice.

Return the length of the

longest non-decreasing

subarray

that can be obtained after performing at most one replacement.

An array is said to be

non-decreasing

if each element is greater than or equal to its previous one (if it exists).

Example 1:

Input:

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

Output:

4

Explanation:

Replacing

nums[3] = 1

with 3 gives the array [1, 2, 3, 3, 2].

The longest non-decreasing subarray is [1, 2, 3, 3], which has a length of 4.

Example 2:

Input:

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

Output:

5

Explanation:

All elements in

nums

are equal, so it is already non-decreasing and the entire

nums

forms a subarray of length 5.

Constraints:

1 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
```

```
*/
function longestSubarray($nums) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (longest-subarray nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Non-Decreasing Subarray After Replacing at Most One
Element
* Difficulty: Medium
* Tags: array, 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 longestSubarray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Non-Decreasing Subarray After Replacing at Most One
Element
* Difficulty: Medium
* Tags: array, 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 longestSubarray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Non-Decreasing Subarray After Replacing at Most One Element
Difficulty: Medium
Tags: array, 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 longestSubarray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Non-Decreasing Subarray After Replacing at Most One
Element
* Difficulty: Medium
* Tags: array, 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 longestSubarray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Non-Decreasing Subarray After Replacing at Most One
Element
* Difficulty: Medium
* Tags: array, 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 longestSubarray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Non-Decreasing Subarray After Replacing at Most One
Element
* Difficulty: Medium
* Tags: array, 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 LongestSubarray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Longest Non-Decreasing Subarray After Replacing at Most One
Element
* Difficulty: Medium
* Tags: array, 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 longestSubarray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Longest Non-Decreasing Subarray After Replacing at Most One
Element
// Difficulty: Medium
// Tags: array, 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 longestSubarray(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Non-Decreasing Subarray After Replacing at Most One
Element
// Difficulty: Medium
// Tags: array, 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 longest_subarray(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
```

```
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (longest-subarray nums)
(-> (listof exact-integer?) exact-integer?)
)
```