Problem 3105: Longest Strictly Increasing or Strictly Decreasing Subarray

Problem Information

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

Problem Description

You are given an array of integers

nums

. Return

the length of the

longest

subarray

of

nums

which is either

strictly increasing

or

strictly decreasing

.

Example 1:

Input:

nums = [1,4,3,3,2]

Output:

2

Explanation:

The strictly increasing subarrays of

nums

are

[1]

,

[2]

,

[3]

,

[3]

,

[4]

, and

[1,4]

.

The strictly decreasing subarrays of

nums

are

[1]

,

[2]

,

[3]

,

[3]

,

[4]

,

[3,2]

, and

[4,3]

.

Hence, we return

2

.

Example 2:

Input:

nums = [3,3,3,3]

Output:

1

Explanation:

The strictly increasing subarrays of

nums

are

[3]

,

[3]

,

[3]

, and

[3]

.

The strictly decreasing subarrays of

nums

are

[3]

,

[3]

,

[3]

, and

[3]

.

Hence, we return

1

.

Example 3:

Input:

nums = [3,2,1]

Output:

3

Explanation:

The strictly increasing subarrays of

nums

are

[3]

,

[2]

, and

[1]

.

The strictly decreasing subarrays of

nums

are

[3]

,

[2]

,

[1]

,

[3,2]

,

[2,1]

, and

[3,2,1]

.

Hence, we return

3

.

Constraints:

1 <= nums.length <= 50

1 <= nums[i] <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Strictly Increasing or Strictly Decreasing Subarray
* Difficulty: Easy
* Tags: array
*
* 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 longestMonotonicSubarray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Strictly Increasing or Strictly Decreasing Subarray
* Difficulty: Easy
* Tags: array
*
* 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 longestMonotonicSubarray(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Longest Strictly Increasing or Strictly Decreasing Subarray
Difficulty: Easy
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Strictly Increasing or Strictly Decreasing Subarray
* Difficulty: Easy
* Tags: array
*
* 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 longestMonotonicSubarray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Strictly Increasing or Strictly Decreasing Subarray
* Difficulty: Easy
* Tags: array
*
* 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 longestMonotonicSubarray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Strictly Increasing or Strictly Decreasing Subarray
* Difficulty: Easy
* Tags: array
*
* 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 LongestMonotonicSubarray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Longest Strictly Increasing or Strictly Decreasing Subarray
* Difficulty: Easy
* Tags: array
*
* 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 longestMonotonicSubarray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Longest Strictly Increasing or Strictly Decreasing Subarray
// Difficulty: Easy
// Tags: array
//
// 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 longestMonotonicSubarray(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Longest Strictly Increasing or Strictly Decreasing Subarray
// Difficulty: Easy
// Tags: array
//
// 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 longest_monotonic_subarray(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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