Problem 2863: Maximum Length of Semi-Decreasing Subarrays

Problem Information

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

Problem Description

You are given an integer array

nums

.

Return

the length of the

longest semi-decreasing

subarray of

nums

, and

0

if there are no such subarrays.

A

subarray

is a contiguous non-empty sequence of elements within an array.

A non-empty array is

semi-decreasing

if its first element is

strictly greater

than its last element.

Example 1:

Input:

nums = [7,6,5,4,3,2,1,6,10,11]

Output:

8

Explanation:

Take the subarray [7,6,5,4,3,2,1,6]. The first element is 7 and the last one is 6 so the condition is met. Hence, the answer would be the length of the subarray or 8. It can be shown that there aren't any subarrays with the given condition with a length greater than 8.

Example 2:

Input:

nums = [57,55,50,60,61,58,63,59,64,60,63]

Output:

6

Explanation:

Take the subarray [61,58,63,59,64,60]. The first element is 61 and the last one is 60 so the condition is met. Hence, the answer would be the length of the subarray or 6. It can be shown that there aren't any subarrays with the given condition with a length greater than 6.

Example 3:

Input:

nums = [1,2,3,4]

Output:

0

Explanation:

Since there are no semi-decreasing subarrays in the given array, the answer is 0.

Constraints:

1 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
```

```
int maxSubarrayLength(vector<int>& nums) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Length of Semi-Decreasing Subarrays
* Difficulty: Medium
* Tags: array, sort, 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 maxSubarrayLength(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Length of Semi-Decreasing Subarrays
```

```
* Difficulty: Medium
* Tags: array, sort, 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 maxSubarrayLength(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Length of Semi-Decreasing Subarrays
Difficulty: Medium
Tags: array, sort, 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 maxSubarrayLength(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Length of Semi-Decreasing Subarrays
* Difficulty: Medium
* Tags: array, sort, 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 maxSubarrayLength = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Length of Semi-Decreasing Subarrays
* Difficulty: Medium
* Tags: array, sort, 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 maxSubarrayLength(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Length of Semi-Decreasing Subarrays
* Difficulty: Medium
* Tags: array, sort, 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 MaxSubarrayLength(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Length of Semi-Decreasing Subarrays
* Difficulty: Medium
* Tags: array, sort, 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 maxSubarrayLength(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Length of Semi-Decreasing Subarrays
// Difficulty: Medium
// Tags: array, sort, 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 maxSubarrayLength(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Length of Semi-Decreasing Subarrays
// Difficulty: Medium
// Tags: array, sort, 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 max_subarray_length(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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