Problem 376: Wiggle Subsequence

Problem Information

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

Problem Description

A

wiggle sequence

is a sequence where the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with one element and a sequence with two non-equal elements are trivially wiggle sequences.

For example,

[1, 7, 4, 9, 2, 5]

is a

wiggle sequence

because the differences

(6, -3, 5, -7, 3)

alternate between positive and negative.

In contrast,

[1, 4, 7, 2, 5]

and

[1, 7, 4, 5, 5]

are not wiggle sequences. The first is not because its first two differences are positive, and the second is not because its last difference is zero.

A

subsequence

is obtained by deleting some elements (possibly zero) from the original sequence, leaving the remaining elements in their original order.

Given an integer array

nums

, return

the length of the longest

wiggle subsequence

of

nums

.

Example 1:

Input:

nums = [1,7,4,9,2,5]

Output:

6

Explanation:

The entire sequence is a wiggle sequence with differences (6, -3, 5, -7, 3).

Example 2:

Input:

nums = [1,17,5,10,13,15,10,5,16,8]

Output:

7

Explanation:

There are several subsequences that achieve this length. One is [1, 17, 10, 13, 10, 16, 8] with differences (16, -7, 3, -3, 6, -8).

Example 3:

Input:

nums = [1,2,3,4,5,6,7,8,9]

Output:

2

Constraints:

1 <= nums.length <= 1000

0 <= nums[i] <= 1000

Follow up:

Could you solve this in

O(n)

time?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Wiggle Subsequence
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 wiggleMaxLength(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Wiggle Subsequence
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 wiggleMaxLength(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Wiggle Subsequence
Difficulty: Medium
Tags: array, dp, greedy

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Wiggle Subsequence
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 wiggleMaxLength = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Wiggle Subsequence
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 wiggleMaxLength(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Wiggle Subsequence
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 WiggleMaxLength(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Wiggle Subsequence
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 wiggleMaxLength(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Wiggle Subsequence
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 wiggleMaxLength(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Wiggle Subsequence
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 wiggle_max_length(nums: Vec<i32>) -> i32 {

}

}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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