Problem 3409: Longest Subsequence With Decreasing Adjacent Difference

Problem Information

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

Problem Description

You are given an array of integers

nums

.

Your task is to find the length of the

longest

subsequence

seq

of

nums

, such that the

absolute differences

between

consecutive

elements form a

non-increasing sequence

of integers. In other words, for a subsequence

seq

0

,

seq

1

,

seq

2

, ...,

seq

m

of

nums

,

|seq

1

- seq

0

| >= |seq

2

- seq

1

| >= ... >= |seq

m

- seq

m - 1

|

.

Return the length of such a subsequence.

Example 1:

Input:

nums = [16,6,3]

Output:

3

Explanation:

The longest subsequence is

[16, 6, 3]

with the absolute adjacent differences

[10, 3]

.

Example 2:

Input:

nums = [6,5,3,4,2,1]

Output:

4

Explanation:

The longest subsequence is

[6, 4, 2, 1]

with the absolute adjacent differences

[2, 2, 1]

.

Example 3:

Input:

nums = [10,20,10,19,10,20]

Output:

5

Explanation:

The longest subsequence is

[10, 20, 10, 19, 10]

with the absolute adjacent differences

[10, 10, 9, 9]

.

Constraints:

2 <= nums.length <= 10

4

1 <= nums[i] <= 300

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Subsequence With Decreasing Adjacent Difference
* 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 longestSubsequence(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Subsequence With Decreasing Adjacent Difference
* 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 longestSubsequence(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Longest Subsequence With Decreasing Adjacent Difference
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 longestSubsequence(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Subsequence With Decreasing Adjacent Difference
* 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 longestSubsequence = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Subsequence With Decreasing Adjacent Difference
* 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 longestSubsequence(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Subsequence With Decreasing Adjacent Difference
* 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 LongestSubsequence(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Longest Subsequence With Decreasing Adjacent Difference
* 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 longestSubsequence(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Longest Subsequence With Decreasing Adjacent Difference
// 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 longestSubsequence(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Longest Subsequence With Decreasing Adjacent Difference
// 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_subsequence(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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