Problem 978: Longest Turbulent Subarray

Problem Information

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

Problem Description

Given an integer array

arr

, return

the length of a maximum size turbulent subarray of

arr

.

A subarray is

turbulent

if the comparison sign flips between each adjacent pair of elements in the subarray.

More formally, a subarray

[arr[i], arr[i + 1], ..., arr[j]]

of

arr

is said to be turbulent if and only if:

For

$i <= k < j$

:

$arr[k] > arr[k + 1]$

when

k

is odd, and

$arr[k] < arr[k + 1]$

when

k

is even.

Or, for

$i <= k < j$

:

$arr[k] > arr[k + 1]$

when

k

is even, and

$arr[k] < arr[k + 1]$

when

k

is odd.

Example 1:

Input:

arr = [9,4,2,10,7,8,8,1,9]

Output:

5

Explanation:

arr[1] > arr[2] < arr[3] > arr[4] < arr[5]

Example 2:

Input:

arr = [4,8,12,16]

Output:

2

Example 3:

Input:

arr = [100]

Output:

1

Constraints:

1 <= arr.length <= 4 * 10

4

0 <= arr[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxTurbulenceSize(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int maxTurbulenceSize(int[] arr) {

}
}
```

Python3:

```
class Solution:
def maxTurbulenceSize(self, arr: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxTurbulenceSize(self, arr):
```

```
"""
:type arr: List[int]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maxTurbulenceSize(arr: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxTurbulenceSize(int[] arr) {

}
}
```

C:

```
int maxTurbulenceSize(int* arr, int arrSize) {

}
```

Go:

```
func maxTurbulenceSize(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun maxTurbulenceSize(arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxTurbulenceSize(_ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_turbulence_size(arr: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer}
def max_turbulence_size(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function maxTurbulenceSize($arr) {

}
}
```

Dart:

```
class Solution {
int maxTurbulenceSize(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def maxTurbulenceSize(arr: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_turbulence_size(arr :: [integer]) :: integer
def max_turbulence_size(arr) do

end
end
```

Erlang:

```
-spec max_turbulence_size(Arr :: [integer()]) -> integer().
max_turbulence_size(Arr) ->
.
```

Racket:

```
(define/contract (max-turbulence-size arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Turbulent Subarray
* 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 maxTurbulenceSize(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Turbulent Subarray
* 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 maxTurbulenceSize(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Turbulent Subarray
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 maxTurbulenceSize(self, arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Turbulent Subarray
* 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[]} arr
* @return {number}
*/
var maxTurbulenceSize = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Turbulent Subarray
* 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 maxTurbulenceSize(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Turbulent Subarray
* 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 MaxTurbulenceSize(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Longest Turbulent Subarray
* 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 maxTurbulenceSize(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Longest Turbulent Subarray
// 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 maxTurbulenceSize(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxTurbulenceSize(arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxTurbulenceSize(_ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Turbulent Subarray
// 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 max_turbulence_size(arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
def max_turbulence_size(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function maxTurbulenceSize($arr) {

}
}
```

Dart Solution:

```
class Solution {
int maxTurbulenceSize(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def maxTurbulenceSize(arr: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_turbulence_size(arr :: [integer]) :: integer
def max_turbulence_size(arr) do

end
end
```

Erlang Solution:

```
-spec max_turbulence_size(Arr :: [integer()]) -> integer().
max_turbulence_size(Arr) ->
.
```

Racket Solution:

```
(define/contract (max-turbulence-size arr)
(-> (listof exact-integer?) exact-integer?)
)
```