Problem 2760: Longest Even Odd Subarray With Threshold

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and an integer

threshold

.

Find the length of the

longest subarray

of

nums

starting at index

l

and ending at index

r

(0 <= l <= r < nums.length)

that satisfies the following conditions:

nums[l] % 2 == 0

For all indices

i

in the range

[l, r - 1]

,

nums[i] % 2 != nums[i + 1] % 2

For all indices

i

in the range

[l, r]

,

nums[i] <= threshold

Return

an integer denoting the length of the longest such subarray.

Note:

A

subarray

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

Example 1:

Input:

nums = [3,2,5,4], threshold = 5

Output:

3

Explanation:

In this example, we can select the subarray that starts at l = 1 and ends at r = 3 => [2,5,4]. This subarray satisfies the conditions. Hence, the answer is the length of the subarray, 3. We can show that 3 is the maximum possible achievable length.

Example 2:

Input:

nums = [1,2], threshold = 2

Output:

1

Explanation:

In this example, we can select the subarray that starts at l = 1 and ends at r = 1 => [2]. It satisfies all the conditions and we can show that 1 is the maximum possible achievable length.

Example 3:

Input:

nums = [2,3,4,5], threshold = 4

Output:

3

Explanation:

In this example, we can select the subarray that starts at l = 0 and ends at r = 2 => [2,3,4]. It satisfies all the conditions. Hence, the answer is the length of the subarray, 3. We can show that 3 is the maximum possible achievable length.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

1 <= threshold <= 100

Code Snippets

C++:

```
class Solution {
public:
int longestAlternatingSubarray(vector<int>& nums, int threshold) {

}
};
```

Java:

```
class Solution {
public int longestAlternatingSubarray(int[] nums, int threshold) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function longestAlternatingSubarray(nums: number[], threshold: number):
number {

};
```

C#:

```
public class Solution {
public int LongestAlternatingSubarray(int[] nums, int threshold) {
```

```
}
}
```

C:

```
int longestAlternatingSubarray(int* nums, int numsSize, int threshold) {

}
```

Go:

```
func longestAlternatingSubarray(nums []int, threshold int) int {

}
```

Kotlin:

```
class Solution {
fun longestAlternatingSubarray(nums: IntArray, threshold: Int): Int {

}
}
```

Swift:

```
class Solution {
func longestAlternatingSubarray(_ nums: [Int], _ threshold: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn longest_alternating_subarray(nums: Vec<i32>, threshold: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} threshold
# @return {Integer}
def longest_alternating_subarray(nums, threshold)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $threshold
* @return Integer
*/
function longestAlternatingSubarray($nums, $threshold) {

}
}
```

Dart:

```
class Solution {
int longestAlternatingSubarray(List<int> nums, int threshold) {

}
}
```

Scala:

```
object Solution {
def longestAlternatingSubarray(nums: Array[Int], threshold: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_alternating_subarray(nums :: [integer], threshold :: integer)
:: integer
def longest_alternating_subarray(nums, threshold) do
```

```
end
end
```

Erlang:

```
-spec longest_alternating_subarray(Nums :: [integer()], Threshold ::
integer()) -> integer().
longest_alternating_subarray(Nums, Threshold) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Even Odd Subarray With Threshold
* 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 longestAlternatingSubarray(vector<int>& nums, int threshold) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Even Odd Subarray With Threshold
* 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 longestAlternatingSubarray(int[] nums, int threshold) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Even Odd Subarray With Threshold
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 longestAlternatingSubarray(self, nums: List[int], threshold: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Even Odd Subarray With Threshold
* 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
* @param {number} threshold
* @return {number}
*/
var longestAlternatingSubarray = function(nums, threshold) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Even Odd Subarray With Threshold
* 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 longestAlternatingSubarray(nums: number[], threshold: number):
number {

};
```

C# Solution:

```
/*
* Problem: Longest Even Odd Subarray With Threshold
```

```
* 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 LongestAlternatingSubarray(int[] nums, int threshold) {

}
}
```

C Solution:

```
/*
* Problem: Longest Even Odd Subarray With Threshold
* 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 longestAlternatingSubarray(int* nums, int numsSize, int threshold) {

}
```

Go Solution:

```
// Problem: Longest Even Odd Subarray With Threshold
// 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 longestAlternatingSubarray(nums []int, threshold int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun longestAlternatingSubarray(nums: IntArray, threshold: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestAlternatingSubarray(_ nums: [Int], _ threshold: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Even Odd Subarray With Threshold
// 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_alternating_subarray(nums: Vec<i32>, threshold: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} threshold
# @return {Integer}
def longest_alternating_subarray(nums, threshold)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $threshold
* @return Integer
*/
function longestAlternatingSubarray($nums, $threshold) {

}
}
```

Dart Solution:

```
class Solution {
int longestAlternatingSubarray(List<int> nums, int threshold) {

}
}
```

Scala Solution:

```
object Solution {
def longestAlternatingSubarray(nums: Array[Int], threshold: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_alternating_subarray(nums :: [integer], threshold :: integer)
:: integer
def longest_alternating_subarray(nums, threshold) do

end
```

```
end
```

Erlang Solution:

```
-spec longest_alternating_subarray(Nums :: [integer()], Threshold ::
integer()) -> integer().
longest_alternating_subarray(Nums, Threshold) ->
.
```

Racket Solution:

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