Problem 3101: Count Alternating Subarrays

Problem Information

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

Problem Description

You are given a

binary array

nums

.

We call a

subarray

alternating

if

no

two

adjacent

elements in the subarray have the

same

value.

Return

the number of alternating subarrays in

nums

.

Example 1:

Input:

nums = [0,1,1,1]

Output:

5

Explanation:

The following subarrays are alternating:

[0]

,

[1]

,

[1]

,

[1]

, and

[0,1]

.

Example 2:

Input:

nums = [1,0,1,0]

Output:

10

Explanation:

Every subarray of the array is alternating. There are 10 possible subarrays that we can choose.

Constraints:

1 <= nums.length <= 10

5

nums[i]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long countAlternatingSubarrays(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long CountAlternatingSubarrays(int[] nums) {

}
}
```

C:

```
long long countAlternatingSubarrays(int* nums, int numsSize) {

}
```

Go:

```
func countAlternatingSubarrays(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun countAlternatingSubarrays(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn count_alternating_subarrays(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-alternating-subarrays nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Alternating Subarrays
* Difficulty: Medium
* Tags: array, math
*
* 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:
long long countAlternatingSubarrays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Alternating Subarrays
* Difficulty: Medium
* Tags: array, math
*
* 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 long countAlternatingSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Alternating Subarrays
Difficulty: Medium
Tags: array, math

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Count Alternating Subarrays
* Difficulty: Medium
* Tags: array, math
*
* 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 countAlternatingSubarrays = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Alternating Subarrays
* Difficulty: Medium
* Tags: array, math
*
* 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 countAlternatingSubarrays(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Alternating Subarrays
* Difficulty: Medium
* Tags: array, math
*
* 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 long CountAlternatingSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Alternating Subarrays
* Difficulty: Medium
* Tags: array, math
*
* 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
*/

long long countAlternatingSubarrays(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Alternating Subarrays
// Difficulty: Medium
// Tags: array, math
//
// 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 countAlternatingSubarrays(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countAlternatingSubarrays(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Alternating Subarrays
// Difficulty: Medium
// Tags: array, math
//
// 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 count_alternating_subarrays(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-alternating-subarrays nums)
(-> (listof exact-integer?) exact-integer?)
)
```