Problem 3392: Count Subarrays of Length Three With a Condition

Problem Information

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

Problem Description

Given an integer array

nums

, return the number of

subarrays

of length 3 such that the sum of the first and third numbers equals

exactly

half of the second number.

Example 1:

Input:

nums = [1,2,1,4,1]

Output:

1

Explanation:

Only the subarray

[1,4,1]

contains exactly 3 elements where the sum of the first and third numbers equals half the middle number.

Example 2:

Input:

nums = [1,1,1]

Output:

0

Explanation:

[1,1,1]

is the only subarray of length 3. However, its first and third numbers do not add to half the middle number.

Constraints:

3 <= nums.length <= 100

-100 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int countSubarrays(vector<int>& nums) {
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Subarrays of Length Three With a Condition
* 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 countSubarrays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Subarrays of Length Three With a Condition
* 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 countSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Subarrays of Length Three With a Condition
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 countSubarrays(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Subarrays of Length Three With a Condition
* 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
* @return {number}
*/
var countSubarrays = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Subarrays of Length Three With a Condition
* 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 countSubarrays(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Subarrays of Length Three With a Condition
* 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 CountSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Subarrays of Length Three With a Condition
* 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 countSubarrays(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Subarrays of Length Three With a Condition
// 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 countSubarrays(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Subarrays of Length Three With a Condition
// 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 count_subarrays(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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