Problem 2348: Number of Zero-Filled Subarrays

Problem Information

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

Problem Description

Given an integer array

nums

, return

the number of

subarrays

filled with

0

.

A

subarray

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

Example 1:

Input:

nums = [1,3,0,0,2,0,0,4]

Output:

6

Explanation:

There are 4 occurrences of [0] as a subarray. There are 2 occurrences of [0,0] as a subarray. There is no occurrence of a subarray with a size more than 2 filled with 0. Therefore, we return 6.

Example 2:

Input:

nums = [0,0,0,2,0,0]

Output:

9

Explanation:

There are 5 occurrences of [0] as a subarray. There are 3 occurrences of [0,0] as a subarray. There is 1 occurrence of [0,0,0] as a subarray. There is no occurrence of a subarray with a size more than 3 filled with 0. Therefore, we return 9.

Example 3:

Input:

nums = [2,10,2019]

Output:

0

Explanation:

There is no subarray filled with 0. Therefore, we return 0.

Constraints:

1 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
```

```
function zeroFilledSubarray($nums) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (zero-filled-subarray nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Zero-Filled 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 zeroFilledSubarray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Zero-Filled 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 zeroFilledSubarray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Zero-Filled 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 zeroFilledSubarray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Zero-Filled 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 zeroFilledSubarray = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Number of Zero-Filled 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 zeroFilledSubarray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Zero-Filled 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 ZeroFilledSubarray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Zero-Filled 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 zeroFilledSubarray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Number of Zero-Filled 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 zeroFilledSubarray(nums []int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Zero-Filled 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 zero_filled_subarray(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (zero-filled-subarray nums)
(-> (listof exact-integer?) exact-integer?)
)
```