Problem 2799: Count Complete Subarrays in an Array

Problem Information

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

Problem Description

You are given an array

nums

consisting of

positive

integers.

We call a subarray of an array

complete

if the following condition is satisfied:

The number of

distinct

elements in the subarray is equal to the number of distinct elements in the whole array.

Return

the number of

complete

subarrays

.

A

subarray

is a contiguous non-empty part of an array.

Example 1:

Input:

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

Output:

4

Explanation:

The complete subarrays are the following: [1,3,1,2], [1,3,1,2,2], [3,1,2] and [3,1,2,2].

Example 2:

Input:

nums = [5,5,5,5]

Output:

10

Explanation:

The array consists only of the integer 5, so any subarray is complete. The number of subarrays that we can choose is 10.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 2000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Complete Subarrays in an Array
* Difficulty: Medium
* Tags: array, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int countCompleteSubarrays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Complete Subarrays in an Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int countCompleteSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Complete Subarrays in an Array
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""
```

```
class Solution:
def countCompleteSubarrays(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Complete Subarrays in an Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Complete Subarrays in an Array
* Difficulty: Medium
* Tags: array, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function countCompleteSubarrays(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Complete Subarrays in an Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int CountCompleteSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Complete Subarrays in an Array
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

```
}
```

Go Solution:

```
// Problem: Count Complete Subarrays in an Array
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func countCompleteSubarrays(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Complete Subarrays in an Array
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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