Problem 2963: Count the Number of Good Partitions

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

consisting of

positive

integers.

A partition of an array into one or more

contiguous

subarrays is called

good

if no two subarrays contain the same number.

Return

the

total number

of good partitions of

nums

.

Since the answer may be large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

nums = [1,2,3,4]

Output:

8

Explanation:

The 8 possible good partitions are: ([1], [2], [3], [4]), ([1], [2], [3,4]), ([1], [2,3], [4]), ([1], [2,3,4]), ([1,2], [3], [4]), ([1,2], [3,4]), ([1,2,3], [4]), and ([1,2,3,4]).

Example 2:

Input:

nums = [1,1,1,1]

Output:

1

Explanation:

The only possible good partition is: ([1,1,1,1]).

Example 3:

Input:

nums = [1,2,1,3]

Output:

2

Explanation:

The 2 possible good partitions are: ([1,2,1], [3]) and ([1,2,1,3]).

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (number-of-good-partitions nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count the Number of Good Partitions
* Difficulty: Hard
* Tags: array, math, 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 numberOfGoodPartitions(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count the Number of Good Partitions
* Difficulty: Hard
* Tags: array, math, 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 numberOfGoodPartitions(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Good Partitions
Difficulty: Hard
Tags: array, math, 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 numberOfGoodPartitions(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Count the Number of Good Partitions
* Difficulty: Hard
* Tags: array, math, 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 numberOfGoodPartitions = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Number of Good Partitions
* Difficulty: Hard
* Tags: array, math, 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 numberOfGoodPartitions(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count the Number of Good Partitions
* Difficulty: Hard
* Tags: array, math, 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 NumberOfGoodPartitions(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count the Number of Good Partitions
* Difficulty: Hard
* Tags: array, math, 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 numberOfGoodPartitions(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count the Number of Good Partitions
// Difficulty: Hard
// Tags: array, math, 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 numberOfGoodPartitions(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count the Number of Good Partitions
// Difficulty: Hard
// Tags: array, math, 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 number_of_good_partitions(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (number-of-good-partitions nums)
(-> (listof exact-integer?) exact-integer?)
)
```