

# Problem 2750: Ways to Split Array Into Good Subarrays

## Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given a binary array

nums

.

A subarray of an array is

good

if it contains

exactly

one

element with the value

1

.

Return

an integer denoting the number of ways to split the array

nums

into

good

subarrays

. As the number may be too large, return it

modulo

10

9

+ 7

.

A subarray is a contiguous

non-empty

sequence of elements within an array.

Example 1:

Input:

nums = [0,1,0,0,1]

Output:

3

Explanation:

There are 3 ways to split nums into good subarrays: - [0,1] [0,0,1] - [0,1,0] [0,1] - [0,1,0,0] [1]

Example 2:

Input:

nums = [0,1,0]

Output:

1

Explanation:

There is 1 way to split nums into good subarrays: - [0,1,0]

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 1

## Code Snippets

**C++:**

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

**Java:**

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

```
}  
}
```

### Python3:

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

### Python:

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

### JavaScript:

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

### TypeScript:

```
function numberOfGoodSubarraySplits(nums: number[]): number {  
  
};
```

### C#:

```
public class Solution {  
    public int NumberOfGoodSubarraySplits(int[] nums) {  
  
    }  
}
```

**C:**

```
int numberOfGoodSubarraySplits(int* nums, int numsSize) {  
  
}
```

**Go:**

```
func numberOfGoodSubarraySplits(nums []int) int {  
  
}
```

**Kotlin:**

```
class Solution {  
    fun numberOfGoodSubarraySplits(nums: IntArray): Int {  
  
    }  
}
```

**Swift:**

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

**Rust:**

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

**Ruby:**

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

## PHP:

```
class Solution {

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

    }

}
```

## Dart:

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

  }
}
```

## Scala:

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

  }
}
```

## Elixir:

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

  end

end
```

## Erlang:

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

## Racket:

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

## Solutions

### C++ Solution:

```
/*
 * Problem: Ways to Split Array Into Good Subarrays
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

    }
};
```

### Java Solution:

```
/**
 * Problem: Ways to Split Array Into Good Subarrays
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

```
}  
}
```

### Python3 Solution:

```
"""  
Problem: Ways to Split Array Into Good Subarrays  
Difficulty: Medium  
Tags: array, dp, math  
  
Approach: Use two pointers or sliding window technique  
Time Complexity: O(n) or O(n log n)  
Space Complexity: O(n) or O(n * m) for DP table  
"""  
  
class Solution:  
    def numberOfGoodSubarraySplits(self, nums: List[int]) -> int:  
        # TODO: Implement optimized solution  
        pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**  
 * Problem: Ways to Split Array Into Good Subarrays  
 * Difficulty: Medium  
 * Tags: array, dp, math  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) or O(n * m) for DP table  
 */
```

```

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

};

```

### TypeScript Solution:

```

/**
 * Problem: Ways to Split Array Into Good Subarrays
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

};

```

### C# Solution:

```

/*
 * Problem: Ways to Split Array Into Good Subarrays
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

    }
}

```

```
}
```

### C Solution:

```
/*
 * Problem: Ways to Split Array Into Good Subarrays
 * Difficulty: Medium
 * Tags: array, dp, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

int numberOfGoodSubarraySplits(int* nums, int numsSize) {

}
```

### Go Solution:

```
// Problem: Ways to Split Array Into Good Subarrays
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numberOfGoodSubarraySplits(nums []int) int {

}
```

### Kotlin Solution:

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

    }
}
```

### Swift Solution:

```

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

    }
}

```

### Rust Solution:

```

// Problem: Ways to Split Array Into Good Subarrays
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

    }
}

```

### Ruby Solution:

```

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

end

```

### PHP Solution:

```

class Solution {

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

    }
}

```

### Dart Solution:

```
class Solution {  
  int numberOfGoodSubarraySplits(List<int> nums) {  
  
  }  
}
```

### Scala Solution:

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

### Elixir Solution:

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

### Erlang Solution:

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

### Racket Solution:

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