Problem 1712: Ways to Split Array Into Three Subarrays

Problem Information

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

Problem Description

A split of an integer array is

good

if:

The array is split into three

non-empty

contiguous subarrays - named

left

,

mid

,

right

respectively from left to right.

The sum of the elements in

left

is less than or equal to the sum of the elements in

mid

, and the sum of the elements in

mid

is less than or equal to the sum of the elements in

right

.

Given

nums

, an array of

non-negative

integers, return

the number of

good

ways to split

nums

. As the number may be too large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

nums = [1,1,1]

Output:

1

Explanation:

The only good way to split nums is [1] [1] [1].

Example 2:

Input:

nums = [1,2,2,2,5,0]

Output:

3

Explanation:

There are three good ways of splitting nums: [1] [2] [2,2,5,0] [1] [2,2] [2,5,0] [1,2] [2,2] [5,0]

Example 3:

Input:

nums = [3,2,1]

Output:

0

Explanation:

There is no good way to split nums.

Constraints:

3 <= nums.length <= 10

5

0 <= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (ways-to-split nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Ways to Split Array Into Three Subarrays
* Difficulty: Medium
* Tags: array, search
*
* 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 waysToSplit(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Ways to Split Array Into Three Subarrays
* Difficulty: Medium
* Tags: array, search
*
* 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 waysToSplit(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Ways to Split Array Into Three Subarrays
Difficulty: Medium
Tags: array, search

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 waysToSplit(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Ways to Split Array Into Three Subarrays
* Difficulty: Medium
* Tags: array, search
*
* 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 waysToSplit = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Ways to Split Array Into Three Subarrays
* Difficulty: Medium
* Tags: array, search
*
* 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 waysToSplit(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Ways to Split Array Into Three Subarrays
* Difficulty: Medium
* Tags: array, search
*
* 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 WaysToSplit(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Ways to Split Array Into Three Subarrays
* Difficulty: Medium
* Tags: array, search
*
* 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 waysToSplit(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Ways to Split Array Into Three Subarrays
// Difficulty: Medium
// Tags: array, search
//
// 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 waysToSplit(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Ways to Split Array Into Three Subarrays
// Difficulty: Medium
// Tags: array, search
//
// 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 ways_to_split(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (ways-to-split nums)
(-> (listof exact-integer?) exact-integer?)
)
```