Problem 3388: Count Beautiful Splits in an Array

Problem Information

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

Problem Description

You are given an array

nums

.

A split of an array

nums

is

beautiful

if:

The array

nums

is split into three

subarrays

:

nums1

,

nums2

, and

nums3

, such that

nums

can be formed by concatenating

nums1

,

nums2

, and

nums3

in that order.

The subarray

nums1

is a

prefix

of

nums2

OR

nums2

is a

prefix

of

nums3

.

Return the

number of ways

you can make this split.

Example 1:

Input:

nums = [1,1,2,1]

Output:

2

Explanation:

The beautiful splits are:

A split with

nums1 = [1]

,

nums2 = [1,2]

,

nums3 = [1]

.

A split with

nums1 = [1]

,

nums2 = [1]

,

nums3 = [2,1]

.

Example 2:

Input:

nums = [1,2,3,4]

Output:

0

Explanation:

There are 0 beautiful splits.

Constraints:

1 <= nums.length <= 5000

0 <= nums[i] <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (beautiful-splits nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Beautiful Splits in an Array
* Difficulty: Medium
* Tags: array, dp
*
* 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 beautifulSplits(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Beautiful Splits in an Array
* Difficulty: Medium
* Tags: array, dp
*
* 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 beautifulSplits(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Beautiful Splits in an Array
Difficulty: Medium
Tags: array, dp

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Beautiful Splits in an Array
* Difficulty: Medium
* Tags: array, dp
*
* 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 beautifulSplits = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Beautiful Splits in an Array
* Difficulty: Medium
* Tags: array, dp
*
* 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 beautifulSplits(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Beautiful Splits in an Array
* Difficulty: Medium
* Tags: array, dp
*
* 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 BeautifulSplits(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Beautiful Splits in an Array
* Difficulty: Medium
* Tags: array, dp
*
* 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 beautifulSplits(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Beautiful Splits in an Array
// Difficulty: Medium
// Tags: array, dp
//
// 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 beautifulSplits(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Beautiful Splits in an Array
// Difficulty: Medium
// Tags: array, dp
//
// 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 beautiful_splits(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (beautiful-splits nums)
(-> (listof exact-integer?) exact-integer?)
)
```