Problem 2464: Minimum Subarrays in a Valid Split

Problem Information

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

Problem Description

You are given an integer array

nums

.

Splitting of an integer array

nums

into

subarrays

is

valid

if:

the

greatest common divisor

of the first and last elements of each subarray is

greater

than

1

, and

each element of

nums

belongs to exactly one subarray.

Return

the

minimum

number of subarrays in a

valid

subarray splitting of

nums

. If a valid subarray splitting is not possible, return

-1

.

Note

that:

The

greatest common divisor

of two numbers is the largest positive integer that evenly divides both numbers.

A

subarray

is a contiguous non-empty part of an array.

Example 1:

Input:

nums = [2,6,3,4,3]

Output:

2

Explanation:

We can create a valid split in the following way: [2,6] | [3,4,3]. - The starting element of the 1

st

subarray is 2 and the ending is 6. Their greatest common divisor is 2, which is greater than 1.
- The starting element of the 2

nd

subarray is 3 and the ending is 3. Their greatest common divisor is 3, which is greater than 1.
It can be proved that 2 is the minimum number of subarrays that we can obtain in a valid split.

Example 2:

Input:

nums = [3,5]

Output:

2

Explanation:

We can create a valid split in the following way: [3] | [5]. - The starting element of the 1

st

subarray is 3 and the ending is 3. Their greatest common divisor is 3, which is greater than 1.
- The starting element of the 2

nd

subarray is 5 and the ending is 5. Their greatest common divisor is 5, which is greater than 1.
It can be proved that 2 is the minimum number of subarrays that we can obtain in a valid split.

Example 3:

Input:

nums = [1,2,1]

Output:

-1

Explanation:

It is impossible to create valid split.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
```

```
var validSubarraySplit = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (valid-subarray-split nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Subarrays in a Valid Split
* 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 validSubarraySplit(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Subarrays in a Valid Split
* 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 validSubarraySplit(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Subarrays in a Valid Split
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 validSubarraySplit(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Subarrays in a Valid Split
* 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 validSubarraySplit = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Subarrays in a Valid Split
* 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 validSubarraySplit(nums: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Subarrays in a Valid Split
* 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 ValidSubarraySplit(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Subarrays in a Valid Split
* 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 validSubarraySplit(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Subarrays in a Valid Split
// 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 validSubarraySplit(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Subarrays in a Valid Split
// 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 valid_subarray_split(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (valid-subarray-split nums)
(-> (listof exact-integer?) exact-integer?)
)
```