Problem 2762: Continuous Subarrays

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. A subarray of

nums

is called

continuous

if:

Let

i

,

i + 1

, ...,

j

be the indices in the subarray. Then, for each pair of indices

i <= i

1

, i

2

<= j

,

0 <=

|nums[i

1

] - nums[i

2

]| <= 2

.

Return

the total number of

continuous

subarrays.

A subarray is a contiguous

non-empty

sequence of elements within an array.

Example 1:

Input:

nums = [5,4,2,4]

Output:

8

Explanation:

Continuous subarray of size 1: [5], [4], [2], [4]. Continuous subarray of size 2: [5,4], [4,2], [2,4]. Continuous subarray of size 3: [4,2,4]. There are no subarrys of size 4. Total continuous subarrays = 4 + 3 + 1 = 8. It can be shown that there are no more continuous subarrays.

Example 2:

Input:

nums = [1,2,3]

Output:

6

Explanation:

Continuous subarray of size 1: [1], [2], [3]. Continuous subarray of size 2: [1,2], [2,3]. Continuous subarray of size 3: [1,2,3]. Total continuous subarrays = 3 + 2 + 1 = 6.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long continuousSubarrays(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long ContinuousSubarrays(int[] nums) {

}
}
```

C:

```
long long continuousSubarrays(int* nums, int numsSize){

}
```

Go:

```
func continuousSubarrays(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun continuousSubarrays(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn continuous_subarrays(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (continuous-subarrays nums)
(-> (listof exact-integer?) exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Continuous Subarrays
* Difficulty: Medium
* Tags: array, queue, heap
```

```
*
* 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:
long long continuousSubarrays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Continuous Subarrays
* Difficulty: Medium
* Tags: array, queue, heap
*
* 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 long continuousSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Continuous Subarrays
Difficulty: Medium
Tags: array, queue, heap

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Continuous Subarrays
* Difficulty: Medium
* Tags: array, queue, heap
*
* 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 continuousSubarrays = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Continuous Subarrays
* Difficulty: Medium
```

```
* Tags: array, queue, heap
*
* 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 continuousSubarrays(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Continuous Subarrays
* Difficulty: Medium
* Tags: array, queue, heap
*
* 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 long ContinuousSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Continuous Subarrays
* Difficulty: Medium
* Tags: array, queue, heap
*
* 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
*/
```

```
long long continuousSubarrays(int* nums, int numsSize){

}
```

Go Solution:

```
// Problem: Continuous Subarrays
// Difficulty: Medium
// Tags: array, queue, heap
//
// 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 continuousSubarrays(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun continuousSubarrays(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Continuous Subarrays
// Difficulty: Medium
// Tags: array, queue, heap
//
// 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 continuous_subarrays(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (continuous-subarrays nums)
(-> (listof exact-integer?) exact-integer?)

)
```