Problem 2763: Sum of Imbalance Numbers of All Subarrays

Problem Information

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

Problem Description

The

imbalance number

of a

0-indexed

integer array

arr

of length

n

is defined as the number of indices in

sarr = sorted(arr)

such that:

0 <= i < n - 1

, and

sarr[i+1] - sarr[i] > 1

Here,

sorted(arr)

is the function that returns the sorted version of

arr

.

Given a

0-indexed

integer array

nums

, return

the

sum of imbalance numbers

of all its

subarrays

.

A

subarray

is a contiguous

non-empty

sequence of elements within an array.

Example 1:

Input:

nums = [2,3,1,4]

Output:

3

Explanation:

There are 3 subarrays with non-zero

imbalance numbers: - Subarray [3, 1] with an imbalance number of 1. - Subarray [3, 1, 4] with an imbalance number of 1. - Subarray [1, 4] with an imbalance number of 1. The imbalance number of all other subarrays is 0. Hence, the sum of imbalance numbers of all the subarrays of nums is 3.

Example 2:

Input:

nums = [1,3,3,3,5]

Output:

8

Explanation:

There are 7 subarrays with non-zero imbalance numbers: - Subarray [1, 3] with an imbalance number of 1. - Subarray [1, 3, 3] with an imbalance number of 1. - Subarray [1, 3, 3, 3] with an imbalance number of 1. - Subarray [1, 3, 3, 3, 5] with an imbalance number of 2. - Subarray [3, 3, 3, 5] with an imbalance number of 1. - Subarray [3, 3, 5] with an imbalance number of 1.

- Subarray [3, 5] with an imbalance number of 1. The imbalance number of all other subarrays is 0. Hence, the sum of imbalance numbers of all the subarrays of nums is 8.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sum-imbalance-numbers nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Imbalance Numbers of All Subarrays
* Difficulty: Hard
* Tags: array, hash, sort
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Sum of Imbalance Numbers of All Subarrays
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Imbalance Numbers of All Subarrays
Difficulty: Hard
Tags: array, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""
```

```
class Solution:
def sumImbalanceNumbers(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Imbalance Numbers of All Subarrays
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Imbalance Numbers of All Subarrays
* Difficulty: Hard
* Tags: array, hash, sort
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

C# Solution:

```
/*
* Problem: Sum of Imbalance Numbers of All Subarrays
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
}
```

C Solution:

```
/*
* Problem: Sum of Imbalance Numbers of All Subarrays
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

```
}
```

Go Solution:

```
// Problem: Sum of Imbalance Numbers of All Subarrays
// Difficulty: Hard
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func sumImbalanceNumbers(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Imbalance Numbers of All Subarrays
// Difficulty: Hard
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sum-imbalance-numbers nums)
(-> (listof exact-integer?) exact-integer?)
)
```