Problem 2179: Count Good Triplets in an Array

Problem Information

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

Problem Description

You are given two

0-indexed

arrays

nums1

and

nums2

of length

n

, both of which are

permutations

of

[0, 1, ..., n - 1]

.

A

good triplet

is a set of

3

distinct

values which are present in

increasing order

by position both in

nums1

and

nums2

. In other words, if we consider

pos1

v

as the index of the value

v

in

nums1

and

pos2

v

as the index of the value

v

in

nums2

, then a good triplet will be a set

(x, y, z)

where

0 <= x, y, z <= n - 1

, such that

pos1

x

< pos1

y

< pos1

z

and

pos2

x

< pos2

y

< pos2

z

.

Return

the

total number

of good triplets

.

Example 1:

Input:

nums1 = [2,0,1,3], nums2 = [0,1,2,3]

Output:

1

Explanation:

There are 4 triplets (x,y,z) such that pos1

x

< pos1

y

< pos1

z

. They are (2,0,1), (2,0,3), (2,1,3), and (0,1,3). Out of those triplets, only the triplet (0,1,3) satisfies pos2

x

< pos2

y

< pos2

z

. Hence, there is only 1 good triplet.

Example 2:

Input:

nums1 = [4,0,1,3,2], nums2 = [4,1,0,2,3]

Output:

4

Explanation:

The 4 good triplets are (4,0,3), (4,0,2), (4,1,3), and (4,1,2).

Constraints:

n == nums1.length == nums2.length

3 <= n <= 10

5

0 <= nums1[i], nums2[i] <= n - 1

nums1

and

nums2

are permutations of

[0, 1, ..., n - 1]

.

Code Snippets

C++:

```
class Solution {
public:
long long goodTriplets(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

```
class Solution {
public long goodTriplets(int[] nums1, int[] nums2) {

}
}
```

Python3:

```
class Solution:
def goodTriplets(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):
def goodTriplets(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var goodTriplets = function(nums1, nums2) {

};
```

TypeScript:

```
function goodTriplets(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
public long GoodTriplets(int[] nums1, int[] nums2) {

}
}
```

C:

```
long long goodTriplets(int* nums1, int nums1Size, int* nums2, int nums2Size)
{

}
```

Go:

```
func goodTriplets(nums1 []int, nums2 []int) int64 {

}
```

Kotlin:

```
class Solution {
fun goodTriplets(nums1: IntArray, nums2: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func goodTriplets(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn good_triplets(nums1: Vec<i32>, nums2: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def good_triplets(nums1, nums2)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function goodTriplets($nums1, $nums2) {

}
}
```

Dart:

```
class Solution {
int goodTriplets(List<int> nums1, List<int> nums2) {

}
}
```

Scala:

```
object Solution {
def goodTriplets(nums1: Array[Int], nums2: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec good_triplets(nums1 :: [integer], nums2 :: [integer]) :: integer
def good_triplets(nums1, nums2) do

end
end
```

Erlang:

```
-spec good_triplets(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
good_triplets(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (good-triplets nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Good Triplets in an Array
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
long long goodTriplets(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Count Good Triplets in an Array
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public long goodTriplets(int[] nums1, int[] nums2) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Count Good Triplets in an Array
Difficulty: Hard
Tags: array, tree, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def goodTriplets(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def goodTriplets(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Good Triplets in an Array
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var goodTriplets = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Good Triplets in an Array
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function goodTriplets(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Good Triplets in an Array
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public long GoodTriplets(int[] nums1, int[] nums2) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Count Good Triplets in an Array
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

long long goodTriplets(int* nums1, int nums1Size, int* nums2, int nums2Size)
{

}
```

Go Solution:

```
// Problem: Count Good Triplets in an Array
// Difficulty: Hard
// Tags: array, tree, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func goodTriplets(nums1 []int, nums2 []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun goodTriplets(nums1: IntArray, nums2: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func goodTriplets(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Good Triplets in an Array
// Difficulty: Hard
// Tags: array, tree, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn good_triplets(nums1: Vec<i32>, nums2: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def good_triplets(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
```

```
function goodTriplets($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
int goodTriplets(List<int> nums1, List<int> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def goodTriplets(nums1: Array[Int], nums2: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec good_triplets(nums1 :: [integer], nums2 :: [integer]) :: integer
def good_triplets(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec good_triplets(Nums1 :: [integer()], Nums2 :: [integer()]) -> integer().
good_triplets(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (good-triplets nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```