Problem 2364: Count Number of Bad Pairs

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. A pair of indices

(i, j)

is a

bad pair

if

i < j

and

j - i != nums[j] - nums[i]

.

Return

the total number of

bad pairs

in

nums

.

Example 1:

Input:

nums = [4,1,3,3]

Output:

5

Explanation:

The pair (0, 1) is a bad pair since 1 - 0 != 1 - 4. The pair (0, 2) is a bad pair since 2 - 0 != 3 - 4, 2 != -1. The pair (0, 3) is a bad pair since 3 - 0 != 3 - 4, 3 != -1. The pair (1, 2) is a bad pair since 2 - 1 != 3 - 1, 1 != 2. The pair (2, 3) is a bad pair since 3 - 2 != 3 - 3, 1 != 0. There are a total of 5 bad pairs, so we return 5.

Example 2:

Input:

nums = [1,2,3,4,5]

Output:

0

Explanation:

There are no bad pairs.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-bad-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Number of Bad Pairs
* Difficulty: Medium
* Tags: array, math, hash
*
* 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:
long long countBadPairs(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Number of Bad Pairs
* Difficulty: Medium
* Tags: array, math, hash
*
* 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 long countBadPairs(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Number of Bad Pairs
Difficulty: Medium
Tags: array, math, hash
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Number of Bad Pairs
* Difficulty: Medium
* Tags: array, math, hash
*
* 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 countBadPairs = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Number of Bad Pairs
* Difficulty: Medium
* Tags: array, math, hash
*
* 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 countBadPairs(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Number of Bad Pairs
* Difficulty: Medium
* Tags: array, math, hash
*
* 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 long CountBadPairs(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Number of Bad Pairs
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

long long countBadPairs(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Number of Bad Pairs
// Difficulty: Medium
// Tags: array, math, hash
//
// 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 countBadPairs(nums []int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Number of Bad Pairs
// Difficulty: Medium
// Tags: array, math, hash
```

```
//
// 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 count_bad_pairs(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-bad-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```