Problem 1512: Number of Good Pairs

Problem Information

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

Problem Description

Given an array of integers

nums

, return

the number of

good pairs

.

A pair

(i, j)

is called

good

if

nums[i] == nums[j]

and

i

<

j

.

Example 1:

Input:

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

Output:

4

Explanation:

There are 4 good pairs (0,3), (0,4), (3,4), (2,5) 0-indexed.

Example 2:

Input:

nums = [1,1,1,1]

Output:

6

Explanation:

Each pair in the array are

good

.

Example 3:

Input:

nums = [1,2,3]

Output:

0

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
```

```
function numIdenticalPairs($nums) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-identical-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Good Pairs
* Difficulty: Easy
* 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:
int numIdenticalPairs(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Good Pairs
* Difficulty: Easy
* 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 int numIdenticalPairs(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Good Pairs
Difficulty: Easy
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 numIdenticalPairs(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Good Pairs
* Difficulty: Easy
* 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 numIdenticalPairs = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Number of Good Pairs
* Difficulty: Easy
* 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 numIdenticalPairs(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Good Pairs
* Difficulty: Easy
* 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 int NumIdenticalPairs(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Good Pairs
* Difficulty: Easy
```

```
* 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
*/

int numIdenticalPairs(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Number of Good Pairs
// Difficulty: Easy
// 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 numIdenticalPairs(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Good Pairs
// Difficulty: Easy
// 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 num_identical_pairs(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-identical-pairs nums)
(-> (listof exact-integer?) exact-integer?)
)
```