Problem 3265: Count Almost Equal Pairs I

Problem Information

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

Problem Description

You are given an array

nums

consisting of positive integers.

We call two integers

x

and

y

in this problem

almost equal

if both integers can become equal after performing the following operation

at most once

:

Choose

either

x

or

y

and swap any two digits within the chosen number.

Return the number of indices

i

and

j

in

nums

where

i < j

such that

nums[i]

and

nums[j]

are

almost equal

.

Note

that it is allowed for an integer to have leading zeros after performing an operation.

Example 1:

Input:

nums = [3,12,30,17,21]

Output:

2

Explanation:

The almost equal pairs of elements are:

3 and 30. By swapping 3 and 0 in 30, you get 3.

12 and 21. By swapping 1 and 2 in 12, you get 21.

Example 2:

Input:

nums = [1,1,1,1,1]

Output:

10

Explanation:

Every two elements in the array are almost equal.

Example 3:

Input:

nums = [123,231]

Output:

0

Explanation:

We cannot swap any two digits of 123 or 231 to reach the other.

Constraints:

2 <= nums.length <= 100

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Almost Equal Pairs I
* Difficulty: Medium
* 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 countPairs(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Almost Equal Pairs I
* Difficulty: Medium
* 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 countPairs(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Almost Equal Pairs I
Difficulty: Medium
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 countPairs(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Almost Equal Pairs I
* Difficulty: Medium
* 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 countPairs = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Almost Equal Pairs I
* Difficulty: Medium
* 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 countPairs(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Almost Equal Pairs I
* Difficulty: Medium
* 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 CountPairs(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Almost Equal Pairs I
* Difficulty: Medium
* 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 countPairs(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Almost Equal Pairs I
// Difficulty: Medium
// 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 countPairs(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Count Almost Equal Pairs I
// Difficulty: Medium
// 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 count_pairs(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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