Problem 2176: Count Equal and Divisible Pairs in an Array

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

of length

n

and an integer

k

, return

the

number of pairs

(i, j)

where

0 <= i < j < n

,

such that

nums[i] == nums[j]

and

(i * j)

is divisible by

k

.

Example 1:

Input:

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

Output:

4

Explanation:

There are 4 pairs that meet all the requirements: - nums[0] == nums[6], and 0 * 6 == 0, which is divisible by 2. - nums[2] == nums[3], and 2 * 3 == 6, which is divisible by 2. - nums[2] == nums[4], and 2 * 4 == 8, which is divisible by 2. - nums[3] == nums[4], and 3 * 4 == 12, which is divisible by 2.

Example 2:

Input:

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

Output:

0

Explanation:

Since no value in nums is repeated, there are no pairs (i,j) that meet all the requirements.

Constraints:

1 <= nums.length <= 100

1 <= nums[i], k <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Equal and Divisible Pairs in an Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Count Equal and Divisible Pairs in an Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int countPairs(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Equal and Divisible Pairs in an Array
Difficulty: Easy
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Equal and Divisible Pairs in an Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Equal and Divisible Pairs in an Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function countPairs(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Equal and Divisible Pairs in an Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int CountPairs(int[] nums, int k) {

}

}
```

C Solution:

```
/*
* Problem: Count Equal and Divisible Pairs in an Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
```

Go Solution:

```
// Problem: Count Equal and Divisible Pairs in an Array
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func countPairs(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Equal and Divisible Pairs in an Array
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function countPairs($nums, $k) {
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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