Problem 2183: Count Array Pairs Divisible by K

Problem Information

Difficulty: Hard
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)

such that:

0 <= i < j <= n - 1

and

nums[i] * nums[j]

is divisible by

k

.

Example 1:

Input:

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

Output:

7

Explanation:

The 7 pairs of indices whose corresponding products are divisible by 2 are (0, 1), (0, 3), (1, 2), (1, 3), (1, 4), (2, 3), and (3, 4). Their products are 2, 4, 6, 8, 10, 12, and 20 respectively. Other pairs such as (0, 2) and (2, 4) have products 3 and 15 respectively, which are not divisible by 2.

Example 2:

Input:

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

Output:

0

Explanation:

There does not exist any pair of indices whose corresponding product is divisible by 5.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i], k <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

C:

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

}
```

Go:

```
func countPairs(nums []int, k int) int64 {
```

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

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): Long = {

}
}
```

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 Array Pairs Divisible by K
* Difficulty: Hard
* Tags: array, math
*
* 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:
long long countPairs(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Count Array Pairs Divisible by K
* Difficulty: Hard
* Tags: array, math
*
* 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 long countPairs(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Array Pairs Divisible by K
Difficulty: Hard
Tags: array, math

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 Array Pairs Divisible by K
* Difficulty: Hard
* Tags: array, math
*
* 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 Array Pairs Divisible by K
* Difficulty: Hard
* Tags: array, math
*
* 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 Array Pairs Divisible by K
* Difficulty: Hard
* Tags: array, math
*
* 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 long CountPairs(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Count Array Pairs Divisible by K
* Difficulty: Hard
* Tags: array, math
*
* 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
*/

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

}
```

Go Solution:

```
// Problem: Count Array Pairs Divisible by K
// Difficulty: Hard
// Tags: array, math
//
// 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) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
```

```
}
```

Rust Solution:

```
// Problem: Count Array Pairs Divisible by K
// Difficulty: Hard
// Tags: array, math
//
// 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) -> i64 {

}
}
```

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): Long = {

}
}
```

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?)
)
```