Problem 3164: Find the Number of Good Pairs II

Problem Information

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

Problem Description

You are given 2 integer arrays

nums1

and

nums2

of lengths

n

and

m

respectively. You are also given a

positive

integer

k

.

A pair

(i, j)

is called

good

if

nums1[i]

is divisible by

nums2[j] * k

(

0 <= i <= n - 1

,

0 <= j <= m - 1

).

Return the total number of

good

pairs.

Example 1:

Input:

nums1 = [1,3,4], nums2 = [1,3,4], k = 1

Output:

5

Explanation:

The 5 good pairs are

(0, 0)

,

(1, 0)

,

(1, 1)

,

(2, 0)

, and

(2, 2)

.

Example 2:

Input:

nums1 = [1,2,4,12], nums2 = [2,4], k = 3

Output:

2

Explanation:

The 2 good pairs are

(3, 0)

and

(3, 1)

.

Constraints:

1 <= n, m <= 10

5

1 <= nums1[i], nums2[j] <= 10

6

1 <= k <= 10

3

Code Snippets

C++:

```
class Solution {
public:
long long numberOfPairs(vector<int>& nums1, vector<int>& nums2, int k) {

}
};
```

Java:

```
class Solution {
public long numberOfPairs(int[] nums1, int[] nums2, int k) {
}
}
```

Python3:

```
class Solution:
def numberOfPairs(self, nums1: List[int], nums2: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def numberOfPairs(self, nums1, nums2, k):
"""
:type nums1: List[int]
:type nums2: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @param {number} k
* @return {number}
*/
var numberOfPairs = function(nums1, nums2, k) {

};
```

TypeScript:

```
function numberOfPairs(nums1: number[], nums2: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long NumberOfPairs(int[] nums1, int[] nums2, int k) {
}
}
```

C:

```
long long numberOfPairs(int* nums1, int nums1Size, int* nums2, int nums2Size,
int k) {

}
```

Go:

```
func numberOfPairs(nums1 []int, nums2 []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun numberOfPairs(nums1: IntArray, nums2: IntArray, k: Int): Long {

}
}
```

Swift:

```
class Solution {
func numberOfPairs(_ nums1: [Int], _ nums2: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_pairs(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer} k
# @return {Integer}
def number_of_pairs(nums1, nums2, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @param Integer $k
* @return Integer
*/
function numberOfPairs($nums1, $nums2, $k) {

}
}
```

Dart:

```
class Solution {
int numberOfPairs(List<int> nums1, List<int> nums2, int k) {

}
}
```

Scala:

```
object Solution {
def numberOfPairs(nums1: Array[Int], nums2: Array[Int], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_pairs(nums1 :: [integer], nums2 :: [integer], k :: integer)
```

```
:: integer
def number_of_pairs(nums1, nums2, k) do

end
end
```

Erlang:

```
-spec number_of_pairs(Nums1 :: [integer()], Nums2 :: [integer()], K ::
integer()) -> integer().
number_of_pairs(Nums1, Nums2, K) ->
.
```

Racket:

```
(define/contract (number-of-pairs nums1 nums2 k)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Number of Good Pairs II
* Difficulty: Medium
* Tags: array, 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 numberOfPairs(vector<int>& nums1, vector<int>& nums2, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Number of Good Pairs II
* Difficulty: Medium
* Tags: array, 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 numberOfPairs(int[] nums1, int[] nums2, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Number of Good Pairs II
Difficulty: Medium
Tags: array, 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 numberOfPairs(self, nums1: List[int], nums2: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberOfPairs(self, nums1, nums2, k):
"""
:type nums1: List[int]
:type nums2: List[int]
:type k: int
```

```
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Number of Good Pairs II
* Difficulty: Medium
* Tags: array, 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[]} nums1
* @param {number[]} nums2
* @param {number} k
* @return {number}
*/
var numberOfPairs = function(nums1, nums2, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Number of Good Pairs II
* Difficulty: Medium
* Tags: array, 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 numberOfPairs(nums1: number[], nums2: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Number of Good Pairs II
* Difficulty: Medium
* Tags: array, 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 NumberOfPairs(int[] nums1, int[] nums2, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the Number of Good Pairs II
* Difficulty: Medium
* Tags: array, 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 numberOfPairs(int* nums1, int nums1Size, int* nums2, int nums2Size,
int k) {

}
```

Go Solution:

```
// Problem: Find the Number of Good Pairs II
// Difficulty: Medium
// Tags: array, 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 numberOfPairs(nums1 []int, nums2 []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfPairs(nums1: IntArray, nums2: IntArray, k: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func numberOfPairs(_ nums1: [Int], _ nums2: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Number of Good Pairs II
// Difficulty: Medium
// Tags: array, 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 number_of_pairs(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer} k
# @return {Integer}
def number_of_pairs(nums1, nums2, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @param Integer $k
* @return Integer
*/
function numberOfPairs($nums1, $nums2, $k) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfPairs(List<int> nums1, List<int> nums2, int k) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfPairs(nums1: Array[Int], nums2: Array[Int], k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_pairs(nums1 :: [integer], nums2 :: [integer], k :: integer)
:: integer
def number_of_pairs(nums1, nums2, k) do

end
end
```

Erlang Solution:

```
-spec number_of_pairs(Nums1 :: [integer()], Nums2 :: [integer()], K ::
integer()) -> integer().
number_of_pairs(Nums1, Nums2, K) ->
.
```

Racket Solution:

```
(define/contract (number-of-pairs nums1 nums2 k)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?)
)
```