Problem 1577: Number of Ways Where Square of Number Is Equal to Product of Two Numbers

Problem Information

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

Problem Description

Given two arrays of integers

nums1

and

nums2

, return the number of triplets formed (type 1 and type 2) under the following rules:

Type 1: Triplet (i, j, k) if

nums1[i]

2

== nums2[j] * nums2[k]

where

0 <= i < nums1.length

and

0 <= j < k < nums2.length

.

Type 2: Triplet (i, j, k) if

nums2[i]

2

== nums1[j] * nums1[k]

where

0 <= i < nums2.length

and

0 <= j < k < nums1.length

.

Example 1:

Input:

nums1 = [7,4], nums2 = [5,2,8,9]

Output:

1

Explanation:

Type 1: (1, 1, 2), nums1[1]

2

= nums2[1] * nums2[2]. (4

2

= 2 * 8).

Example 2:

Input:

nums1 = [1,1], nums2 = [1,1,1]

Output:

9

Explanation:

All Triplets are valid, because 1

2

= 1 * 1. Type 1: (0,0,1), (0,0,2), (0,1,2), (1,0,1), (1,0,2), (1,1,2). nums1[i]

2

= nums2[j] * nums2[k]. Type 2: (0,0,1), (1,0,1), (2,0,1). nums2[i]

2

= nums1[j] * nums1[k].

Example 3:

Input:

nums1 = [7,7,8,3], nums2 = [1,2,9,7]

Output:

2

Explanation:

There are 2 valid triplets. Type 1: (3,0,2). nums1[3]

2

= nums2[0] * nums2[2]. Type 2: (3,0,1). nums2[3]

2

= nums1[0] * nums1[1].

Constraints:

1 <= nums1.length, nums2.length <= 1000

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

5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int numTriplets(int[] nums1, int[] nums2) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int NumTriplets(int[] nums1, int[] nums2) {

}
}
```

C:

```
int numTriplets(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go:

```
func numTriplets(nums1 []int, nums2 []int) int {

}
```

Kotlin:

```
class Solution {
fun numTriplets(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

```
.
```

Racket:

```
(define/contract (num-triplets nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways Where Square of Number Is Equal to Product of Two
Numbers
* Difficulty: Medium
* 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 numTriplets(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways Where Square of Number Is Equal to Product of Two
Numbers
* Difficulty: Medium
* 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 numTriplets(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways Where Square of Number Is Equal to Product of Two
Numbers
Difficulty: Medium
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 numTriplets(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Ways Where Square of Number Is Equal to Product of Two
```

```
Numbers
* Difficulty: Medium
* 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[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var numTriplets = function(nums1, nums2) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways Where Square of Number Is Equal to Product of Two
Numbers
* Difficulty: Medium
* 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 numTriplets(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways Where Square of Number Is Equal to Product of Two
Numbers
* Difficulty: Medium
* 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 NumTriplets(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways Where Square of Number Is Equal to Product of Two
Numbers
* Difficulty: Medium
* 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 numTriplets(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Number of Ways Where Square of Number Is Equal to Product of Two
Numbers
// Difficulty: Medium
// 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 numTriplets(nums1 []int, nums2 []int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun numTriplets(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Ways Where Square of Number Is Equal to Product of Two
Numbers
// Difficulty: Medium
// 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_triplets(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
```

```
def num_triplets(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-triplets nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```