Problem 2964: Number of Divisible Triplet Sums

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

and an integer

d

, return

the number of triplets

(i, j, k)

such that

i < j < k

and

(nums[i] + nums[j] + nums[k]) % d == 0

.

Example 1:

Input:

nums = [3,3,4,7,8], d = 5

Output:

3

Explanation:

The triplets which are divisible by 5 are: (0, 1, 2), (0, 2, 4), (1, 2, 4). It can be shown that no other triplet is divisible by 5. Hence, the answer is 3.

Example 2:

Input:

nums = [3,3,3,3], d = 3

Output:

4

Explanation:

Any triplet chosen here has a sum of 9, which is divisible by 3. Hence, the answer is the total number of triplets which is 4.

Example 3:

Input:

nums = [3,3,3,3], d = 6

Output:

0

Explanation:

Any triplet chosen here has a sum of 9, which is not divisible by 6. Hence, the answer is 0.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 10

9

1 <= d <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int divisibleTripletCount(vector<int>& nums, int d) {

}
};
```

Java:

```
class Solution {
public int divisibleTripletCount(int[] nums, int d) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function divisibleTripletCount(nums: number[], d: number): number {

};
```

C#:

```
public class Solution {
public int DivisibleTripletCount(int[] nums, int d) {

}
}
```

C:

```
int divisibleTripletCount(int* nums, int numsSize, int d) {

}
```

Go:

```
func divisibleTripletCount(nums []int, d int) int {

}
```

Kotlin:

```
class Solution {
fun divisibleTripletCount(nums: IntArray, d: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} d
# @return {Integer}
def divisible_triplet_count(nums, d)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @param Integer $d
* @return Integer
*/
function divisibleTripletCount($nums, $d) {

}
}
```

Dart:

```
class Solution {
int divisibleTripletCount(List<int> nums, int d) {

}
}
```

Scala:

```
object Solution {
def divisibleTripletCount(nums: Array[Int], d: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec divisible_triplet_count(nums :: [integer], d :: integer) :: integer
def divisible_triplet_count(nums, d) do

end
end
```

Erlang:

```
-spec divisible_triplet_count(Nums :: [integer()], D :: integer()) ->
integer().
divisible_triplet_count(Nums, D) ->
.
```

Racket:

```
(define/contract (divisible-triplet-count nums d)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Divisible Triplet Sums
* 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:
int divisibleTripletCount(vector<int>& nums, int d) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Divisible Triplet Sums
* 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 int divisibleTripletCount(int[] nums, int d) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Number of Divisible Triplet Sums
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 divisibleTripletCount(self, nums: List[int], d: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Divisible Triplet Sums
* 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[]} nums
* @param {number} d
* @return {number}
*/
var divisibleTripletCount = function(nums, d) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Divisible Triplet Sums
* 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 divisibleTripletCount(nums: number[], d: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Divisible Triplet Sums
* 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 int DivisibleTripletCount(int[] nums, int d) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Number of Divisible Triplet Sums
* 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
*/

int divisibleTripletCount(int* nums, int numsSize, int d) {

}
```

Go Solution:

```
// Problem: Number of Divisible Triplet Sums
// 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 divisibleTripletCount(nums []int, d int) int {

}
```

Kotlin Solution:

```
class Solution {
fun divisibleTripletCount(nums: IntArray, d: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Divisible Triplet Sums
// 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 divisible_triplet_count(nums: Vec<i32>, d: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} d
# @return {Integer}
def divisible_triplet_count(nums, d)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $d
* @return Integer
*/
```

```
function divisibleTripletCount($nums, $d) {

}
}
```

Dart Solution:

```
class Solution {
int divisibleTripletCount(List<int> nums, int d) {

}
}
```

Scala Solution:

```
object Solution {
def divisibleTripletCount(nums: Array[Int], d: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec divisible_triplet_count(nums :: [integer], d :: integer) :: integer
def divisible_triplet_count(nums, d) do

end
end
```

Erlang Solution:

```
-spec divisible_triplet_count(Nums :: [integer()], D :: integer()) ->
integer().
divisible_triplet_count(Nums, D) ->
.
```

Racket Solution:

```
(define/contract (divisible-triplet-count nums d)
(-> (listof exact-integer?) exact-integer? exact-integer?)
```

```
)
```