Problem 1534: Count Good Triplets

Problem Information

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

Problem Description

Given an array of integers

arr

, and three integers

a

,

b

and

c

. You need to find the number of good triplets.

A triplet

(arr[i], arr[j], arr[k])

is

good

if the following conditions are true:

0 <= i < j < k < arr.length

|arr[i] - arr[j]| <= a

|arr[j] - arr[k]| <= b

|arr[i] - arr[k]| <= c

Where

|x|

denotes the absolute value of

x

.

Return

the number of good triplets

.

Example 1:

Input:

arr = [3,0,1,1,9,7], a = 7, b = 2, c = 3

Output:

4

Explanation:

There are 4 good triplets: [(3,0,1), (3,0,1), (3,1,1), (0,1,1)].

Example 2:

Input:

arr = [1,1,2,2,3], a = 0, b = 0, c = 1

Output:

0

Explanation:

No triplet satisfies all conditions.

Constraints:

3 <= arr.length <= 100

0 <= arr[i] <= 1000

0 <= a, b, c <= 1000

Code Snippets

C++:

```
class Solution {
public:
int countGoodTriplets(vector<int>& arr, int a, int b, int c) {

}
};
```

Java:

```
class Solution {
public int countGoodTriplets(int[] arr, int a, int b, int c) {

}
}
```

Python3:

```
class Solution:
def countGoodTriplets(self, arr: List[int], a: int, b: int, c: int) -> int:
```

Python:

```
class Solution(object):
def countGoodTriplets(self, arr, a, b, c):
"""
:type arr: List[int]
:type a: int
:type b: int
:type c: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} arr
* @param {number} a
* @param {number} b
* @param {number} c
* @return {number}
*/
var countGoodTriplets = function(arr, a, b, c) {

};
```

TypeScript:

```
function countGoodTriplets(arr: number[], a: number, b: number, c: number):
number {

};
```

C#:

```
public class Solution {
public int CountGoodTriplets(int[] arr, int a, int b, int c) {

}
}
```

C:

```



int countGoodTriplets(int* arr, int arrSize, int a, int b, int c){

}
```

Go:

```
func countGoodTriplets(arr []int, a int, b int, c int) int {

}
```

Kotlin:

```
class Solution {
fun countGoodTriplets(arr: IntArray, a: Int, b: Int, c: Int): Int {

}
}
```

Swift:

```
class Solution {
func countGoodTriplets(_ arr: [Int], _ a: Int, _ b: Int, _ c: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_good_triplets(arr: Vec<i32>, a: i32, b: i32, c: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @return {Integer}
def count_good_triplets(arr, a, b, c)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @return Integer
*/
function countGoodTriplets($arr, $a, $b, $c) {

}
}
```

Scala:

```
object Solution {
def countGoodTriplets(arr: Array[Int], a: Int, b: Int, c: Int): Int = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Count Good Triplets
* 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 countGoodTriplets(vector<int>& arr, int a, int b, int c) {

}
};
```

Java Solution:

```
/**
* Problem: Count Good Triplets
* 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 countGoodTriplets(int[] arr, int a, int b, int c) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Good Triplets
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 countGoodTriplets(self, arr: List[int], a: int, b: int, c: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countGoodTriplets(self, arr, a, b, c):
"""
:type arr: List[int]
:type a: int
:type b: int
:type c: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Good Triplets
* 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[]} arr
* @param {number} a
* @param {number} b
* @param {number} c
* @return {number}
*/
var countGoodTriplets = function(arr, a, b, c) {
};
```

TypeScript Solution:

```
/**
* Problem: Count Good Triplets
* 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 countGoodTriplets(arr: number[], a: number, b: number, c: number):
number {

};
```

C# Solution:

```
/*
* Problem: Count Good Triplets
* 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 CountGoodTriplets(int[] arr, int a, int b, int c) {

}
}
```

C Solution:

```
/*
* Problem: Count Good Triplets
* 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 countGoodTriplets(int* arr, int arrSize, int a, int b, int c){

}
```

Go Solution:

```go
// Problem: Count Good Triplets
// 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 countGoodTriplets(arr []int, a int, b int, c int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun countGoodTriplets(arr: IntArray, a: Int, b: Int, c: Int): Int {

}
}
```

Swift Solution:

```swift
class Solution {
func countGoodTriplets(_ arr: [Int], _ a: Int, _ b: Int, _ c: Int) -> Int {
}
}
```

Rust Solution:

```
// Problem: Count Good Triplets
// 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_good_triplets(arr: Vec<i32>, a: i32, b: i32, c: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @return {Integer}
def count_good_triplets(arr, a, b, c)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @return Integer
*/
function countGoodTriplets($arr, $a, $b, $c) {

}
}
```

Scala Solution:

```
object Solution {
def countGoodTriplets(arr: Array[Int], a: Int, b: Int, c: Int): Int = {

}
}
```