Problem 2563: Count the Number of Fair Pairs

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

of size

n

and two integers

lower

and

upper

, return

the number of fair pairs

.

A pair

(i, j)

is

fair

if:

0 <= i < j < n

, and

lower <= nums[i] + nums[j] <= upper

Example 1:

Input:

nums = [0,1,7,4,4,5], lower = 3, upper = 6

Output:

6

Explanation:

There are 6 fair pairs: (0,3), (0,4), (0,5), (1,3), (1,4), and (1,5).

Example 2:

Input:

nums = [1,7,9,2,5], lower = 11, upper = 11

Output:

1

Explanation:

There is a single fair pair: (2,3).

Constraints:

1 <= nums.length <= 10

5

nums.length == n

-10

9

<= nums[i] <= 10

9

-10

9

<= lower <= upper <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long countFairPairs(vector<int>& nums, int lower, int upper) {

}
};
```

Java:

```
class Solution {
public long countFairPairs(int[] nums, int lower, int upper) {

}
}
```

Python3:

```
class Solution:
def countFairPairs(self, nums: List[int], lower: int, upper: int) -> int:
```

Python:

```
class Solution(object):
def countFairPairs(self, nums, lower, upper):
"""
:type nums: List[int]
:type lower: int
:type upper: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} lower
* @param {number} upper
* @return {number}
*/
var countFairPairs = function(nums, lower, upper) {

};
```

TypeScript:

```
function countFairPairs(nums: number[], lower: number, upper: number): number
{

};
```

C#:

```
public class Solution {
public long CountFairPairs(int[] nums, int lower, int upper) {

}
}
```

C:

```
long long countFairPairs(int* nums, int numsSize, int lower, int upper) {

}
```

Go:

```
func countFairPairs(nums []int, lower int, upper int) int64 {

}
```

Kotlin:

```
class Solution {
fun countFairPairs(nums: IntArray, lower: Int, upper: Int): Long {

}
}
```

Swift:

```
class Solution {
func countFairPairs(_ nums: [Int], _ lower: Int, _ upper: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_fair_pairs(nums: Vec<i32>, lower: i32, upper: i32) -> i64 {

}
}
```

Ruby:

```ruby
# @param {Integer[]} nums
# @param {Integer} lower
# @param {Integer} upper
# @return {Integer}
def count_fair_pairs(nums, lower, upper)

end
```

PHP:

```php
class Solution {

/**
* @param Integer[] $nums
* @param Integer $lower
* @param Integer $upper
* @return Integer
*/
function countFairPairs($nums, $lower, $upper) {

}
}
```

Dart:

```dart
class Solution {
int countFairPairs(List<int> nums, int lower, int upper) {

}
}
```

Scala:

```scala
object Solution {
def countFairPairs(nums: Array[Int], lower: Int, upper: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_fair_pairs(nums :: [integer], lower :: integer, upper :: integer)
:: integer
def count_fair_pairs(nums, lower, upper) do

end
end
```

Erlang:

```
-spec count_fair_pairs(Nums :: [integer()], Lower :: integer(), Upper ::
integer()) -> integer().
count_fair_pairs(Nums, Lower, Upper) ->
.
```

Racket:

```
(define/contract (count-fair-pairs nums lower upper)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count the Number of Fair Pairs
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 countFairPairs(vector<int>& nums, int lower, int upper) {

}
};
```

Java Solution:

```
/**
* Problem: Count the Number of Fair Pairs
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 countFairPairs(int[] nums, int lower, int upper) {

}
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Fair Pairs
Difficulty: Medium
Tags: array, sort, search

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 countFairPairs(self, nums: List[int], lower: int, upper: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countFairPairs(self, nums, lower, upper):
"""
:type nums: List[int]
:type lower: int
```

```
:type upper: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count the Number of Fair Pairs
* Difficulty: Medium
* Tags: array, sort, search
*
* 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} lower
* @param {number} upper
* @return {number}
*/
var countFairPairs = function(nums, lower, upper) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Number of Fair Pairs
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 countFairPairs(nums: number[], lower: number, upper: number): number
{
};
```

C# Solution:

```
/*
* Problem: Count the Number of Fair Pairs
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 CountFairPairs(int[] nums, int lower, int upper) {

}
}
```

C Solution:

```
/*
* Problem: Count the Number of Fair Pairs
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 countFairPairs(int* nums, int numsSize, int lower, int upper) {

}
```

Go Solution:

```
// Problem: Count the Number of Fair Pairs
// Difficulty: Medium
// Tags: array, sort, search
```

```
//
// 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 countFairPairs(nums []int, lower int, upper int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countFairPairs(nums: IntArray, lower: Int, upper: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func countFairPairs(_ nums: [Int], _ lower: Int, _ upper: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count the Number of Fair Pairs
// Difficulty: Medium
// Tags: array, sort, search
//
// 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_fair_pairs(nums: Vec<i32>, lower: i32, upper: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} lower
# @param {Integer} upper
# @return {Integer}
def count_fair_pairs(nums, lower, upper)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $lower
* @param Integer $upper
* @return Integer
*/
function countFairPairs($nums, $lower, $upper) {

}
}
```

Dart Solution:

```
class Solution {
int countFairPairs(List<int> nums, int lower, int upper) {

}
}
```

Scala Solution:

```
object Solution {
def countFairPairs(nums: Array[Int], lower: Int, upper: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_fair_pairs(nums :: [integer], lower :: integer, upper :: integer)
:: integer
def count_fair_pairs(nums, lower, upper) do

end
end
```

Erlang Solution:

```
-spec count_fair_pairs(Nums :: [integer()], Lower :: integer(), Upper ::
integer()) -> integer().
count_fair_pairs(Nums, Lower, Upper) ->
.
```

Racket Solution:

```
(define/contract (count-fair-pairs nums lower upper)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```