Problem 2426: Number of Pairs Satisfying Inequality

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays

nums1

and

nums2

, each of size

n

, and an integer

diff

. Find the number of

pairs

(i, j)

such that:

0 <= i < j <= n - 1

and

nums1[i] - nums1[j] <= nums2[i] - nums2[j] + diff

.

Return

the

number of pairs

that satisfy the conditions.

Example 1:

Input:

nums1 = [3,2,5], nums2 = [2,2,1], diff = 1

Output:

3

Explanation:

There are 3 pairs that satisfy the conditions: 1. i = 0, j = 1: 3 - 2 <= 2 - 2 + 1. Since i < j and 1 <= 1, this pair satisfies the conditions. 2. i = 0, j = 2: 3 - 5 <= 2 - 1 + 1. Since i < j and -2 <= 2, this pair satisfies the conditions. 3. i = 1, j = 2: 2 - 5 <= 2 - 1 + 1. Since i < j and -3 <= 2, this pair satisfies the conditions. Therefore, we return 3.

Example 2:

Input:

nums1 = [3,-1], nums2 = [-2,2], diff = -1

Output:

0

Explanation:

Since there does not exist any pair that satisfies the conditions, we return 0.

Constraints:

n == nums1.length == nums2.length

2 <= n <= 10

5

-10

4

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

4

-10

4

<= diff <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Number of Pairs Satisfying Inequality
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Number of Pairs Satisfying Inequality
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public long numberOfPairs(int[] nums1, int[] nums2, int diff) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Pairs Satisfying Inequality
Difficulty: Hard
Tags: array, tree, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
```

```
def numberOfPairs(self, nums1: List[int], nums2: List[int], diff: int) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Pairs Satisfying Inequality
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Number of Pairs Satisfying Inequality
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

C# Solution:

```
/*
* Problem: Number of Pairs Satisfying Inequality
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public long NumberOfPairs(int[] nums1, int[] nums2, int diff) {

}
}
```

C Solution:

```
/*
* Problem: Number of Pairs Satisfying Inequality
* Difficulty: Hard
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
```

Go Solution:

```
// Problem: Number of Pairs Satisfying Inequality
// Difficulty: Hard
// Tags: array, tree, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Pairs Satisfying Inequality
// Difficulty: Hard
// Tags: array, tree, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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