Problem 2824: Count Pairs Whose Sum is Less than Target

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

of length

n

and an integer

target

, return

the number of pairs

(i, j)

where

0 <= i < j < n

and

nums[i] + nums[j] < target

.

Example 1:

Input:

nums = [-1,1,2,3,1], target = 2

Output:

3

Explanation:

There are 3 pairs of indices that satisfy the conditions in the statement: - (0, 1) since 0 < 1 and nums[0] + nums[1] = 0 < target - (0, 2) since 0 < 2 and nums[0] + nums[2] = 1 < target - (0, 4) since 0 < 4 and nums[0] + nums[4] = 0 < target Note that (0, 3) is not counted since nums[0] + nums[3] is not strictly less than the target.

Example 2:

Input:

nums = [-6,2,5,-2,-7,-1,3], target = -2

Output:

10

Explanation:

There are 10 pairs of indices that satisfy the conditions in the statement: - (0, 1) since 0 < 1 and nums[0] + nums[1] = -4 < target - (0, 3) since 0 < 3 and nums[0] + nums[3] = -8 < target - (0, 4) since 0 < 4 and nums[0] + nums[4] = -13 < target - (0, 5) since 0 < 5 and nums[0] + nums[5] = -7 < target - (0, 6) since 0 < 6 and nums[0] + nums[6] = -3 < target - (1, 4) since 1 <

4 and nums[1] + nums[4] = -5 < target - (3, 4) since 3 < 4 and nums[3] + nums[4] = -9 < target - (3, 5) since 3 < 5 and nums[3] + nums[5] = -3 < target - (4, 5) since 4 < 5 and nums[4] + nums[5] = -8 < target - (4, 6) since 4 < 6 and nums[4] + nums[6] = -4 < target

Constraints:

1 <= nums.length == n <= 50

-50 <= nums[i], target <= 50

Code Snippets

C++:

```
class Solution {
public:
int countPairs(vector<int>& nums, int target) {

}
};
```

Java:

```
class Solution {
public int countPairs(List<Integer> nums, int target) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def countPairs(self, nums, target):
"""
:type nums: List[int]
:type target: int
```

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function countPairs(nums: number[], target: number): number {

};
```

C#:

```
public class Solution {
public int CountPairs(IList<int> nums, int target) {

}
}
```

C:

```
int countPairs(int* nums, int numsSize, int target) {

}
```

Go:

```
func countPairs(nums []int, target int) int {

}
```

Kotlin:

```
class Solution {
fun countPairs(nums: List<Int>, target: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def count_pairs(nums, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function countPairs($nums, $target) {

}
```

```
}
```

Dart:

```
class Solution {
int countPairs(List<int> nums, int target) {

}
}
```

Scala:

```
object Solution {
def countPairs(nums: List[Int], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_pairs(nums :: [integer], target :: integer) :: integer
def count_pairs(nums, target) do

end
end
```

Erlang:

```
-spec count_pairs(Nums :: [integer()], Target :: integer()) -> integer().
count_pairs(Nums, Target) ->
.
```

Racket:

```
(define/contract (count-pairs nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Pairs Whose Sum is Less than Target
* Difficulty: Easy
* 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:
int countPairs(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Count Pairs Whose Sum is Less than Target
* Difficulty: Easy
* 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 int countPairs(List<Integer> nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Pairs Whose Sum is Less than Target
Difficulty: Easy
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 countPairs(self, nums: List[int], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Pairs Whose Sum is Less than Target
* Difficulty: Easy
* 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} target
* @return {number}
*/
var countPairs = function(nums, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Pairs Whose Sum is Less than Target
* Difficulty: Easy
* 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 countPairs(nums: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Pairs Whose Sum is Less than Target
* Difficulty: Easy
* 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 int CountPairs(IList<int> nums, int target) {

}
}
```

C Solution:

```
/*
* Problem: Count Pairs Whose Sum is Less than Target
* Difficulty: Easy
* 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
*/

int countPairs(int* nums, int numsSize, int target) {

}
```

Go Solution:

```
// Problem: Count Pairs Whose Sum is Less than Target
// Difficulty: Easy
// 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 countPairs(nums []int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countPairs(nums: List<Int>, target: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Pairs Whose Sum is Less than Target
// Difficulty: Easy
// 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_pairs(nums: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def count_pairs(nums, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function countPairs($nums, $target) {

}
}
```

Dart Solution:

```
class Solution {
int countPairs(List<int> nums, int target) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def countPairs(nums: List[Int], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_pairs(nums :: [integer], target :: integer) :: integer
def count_pairs(nums, target) do

end
end
```

Erlang Solution:

```
-spec count_pairs(Nums :: [integer()], Target :: integer()) -> integer().
count_pairs(Nums, Target) ->
.
```

Racket Solution:

```
(define/contract (count-pairs nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```