Problem 1995: Count Special Quadruplets

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

, return

the number of

distinct

quadruplets

(a, b, c, d)

such that:

nums[a] + nums[b] + nums[c] == nums[d]

, and

a < b < c < d

Example 1:

Input:

nums = [1,2,3,6]

Output:

1

Explanation:

The only quadruplet that satisfies the requirement is (0, 1, 2, 3) because 1 + 2 + 3 == 6.

Example 2:

Input:

nums = [3,3,6,4,5]

Output:

0

Explanation:

There are no such quadruplets in [3,3,6,4,5].

Example 3:

Input:

nums = [1,1,1,3,5]

Output:

4

Explanation:

The 4 quadruplets that satisfy the requirement are: - (0, 1, 2, 3): 1 + 1 + 1 == 3 - (0, 1, 3, 4): 1 + 1 + 3 == 5 - (0, 2, 3, 4): 1 + 1 + 3 == 5 - (1, 2, 3, 4): 1 + 1 + 3 == 5

Constraints:

4 <= nums.length <= 50

1 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int countQuadruplets(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int countQuadruplets(int[] nums) {

}
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

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

};
```

TypeScript:

```
function countQuadruplets(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int CountQuadruplets(int[] nums) {

}
}
```

C:

```
int countQuadruplets(int* nums, int numsSize) {

}
```

Go:

```
func countQuadruplets(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun countQuadruplets(nums: IntArray): Int {
```

```
}
}
```

Swift:

```
class Solution {
func countQuadruplets(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_quadruplets(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def count_quadruplets(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function countQuadruplets($nums) {

}
}
```

Dart:

```
class Solution {
int countQuadruplets(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def countQuadruplets(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_quadruplets(nums :: [integer]) :: integer
def count_quadruplets(nums) do

end
end
```

Erlang:

```
-spec count_quadruplets(Nums :: [integer()]) -> integer().
count_quadruplets(Nums) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Special Quadruplets
```

```
* Difficulty: Easy
* 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 countQuadruplets(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Special Quadruplets
* Difficulty: Easy
* 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 countQuadruplets(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Special Quadruplets
Difficulty: Easy
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 countQuadruplets(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Special Quadruplets
* Difficulty: Easy
* 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
* @return {number}
*/
var countQuadruplets = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Special Quadruplets
* Difficulty: Easy
* 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 countQuadruplets(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Special Quadruplets
* Difficulty: Easy
* 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 CountQuadruplets(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Special Quadruplets
* Difficulty: Easy
* 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 countQuadruplets(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Special Quadruplets
// Difficulty: Easy
// 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 countQuadruplets(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countQuadruplets(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func countQuadruplets(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Special Quadruplets
// Difficulty: Easy
// 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 count_quadruplets(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def count_quadruplets(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function countQuadruplets($nums) {

}
}
```

Dart Solution:

```
class Solution {
int countQuadruplets(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def countQuadruplets(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_quadruplets(nums :: [integer]) :: integer
def count_quadruplets(nums) do

end
end
```

Erlang Solution:

```
-spec count_quadruplets(Nums :: [integer()]) -> integer().
count_quadruplets(Nums) ->
.
```

Racket Solution:

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