Problem 2552: Count Increasing Quadruplets

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

of size

n

containing all numbers from

1

to

n

, return

the number of increasing quadruplets

.

A quadruplet

(i, j, k, l)

is increasing if:

0 <= i < j < k < l < n

, and

nums[i] < nums[k] < nums[j] < nums[l]

.

Example 1:

Input:

nums = [1,3,2,4,5]

Output:

2

Explanation:

- When i = 0, j = 1, k = 2, and l = 3, nums[i] < nums[k] < nums[j] < nums[l]. - When i = 0, j = 1, k = 2, and l = 4, nums[i] < nums[k] < nums[j] < nums[l]. There are no other quadruplets, so we return 2.

Example 2:

Input:

nums = [1,2,3,4]

Output:

0

Explanation:

There exists only one quadruplet with i = 0, j = 1, k = 2, l = 3, but since nums[j] < nums[k], we return 0.

Constraints:

4 <= nums.length <= 4000

1 <= nums[i] <= nums.length

All the integers of

nums

are

unique

.

nums

is a permutation.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

C:

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

}
```

Go:

```
func countQuadruplets(nums []int) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn count_quadruplets(nums: Vec<i32>) -> i64 {

}
}
```

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]): Long = {

}
}
```

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 Increasing Quadruplets
* Difficulty: Hard
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Count Increasing Quadruplets
* Difficulty: Hard
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

Python3 Solution:

```
"""
Problem: Count Increasing Quadruplets
Difficulty: Hard
Tags: array, tree, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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 Increasing Quadruplets
* Difficulty: Hard
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Increasing Quadruplets
* Difficulty: Hard
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Count Increasing Quadruplets
* Difficulty: Hard
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Count Increasing Quadruplets
* Difficulty: Hard
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Count Increasing Quadruplets
// Difficulty: Hard
// Tags: array, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func countQuadruplets(nums []int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Increasing Quadruplets
// Difficulty: Hard
// Tags: array, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn count_quadruplets(nums: Vec<i32>) -> i64 {

}
}
```

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]): Long = {

}
}
```

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?)
)
```