Problem 2148: Count Elements With Strictly Smaller and Greater Elements

Problem Information

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

Problem Description

Given an integer array

nums

, return

the number of elements that have

both

a strictly smaller and a strictly greater element appear in

nums

.

Example 1:

Input:

nums = [11,7,2,15]

Output:

2

Explanation:

The element 7 has the element 2 strictly smaller than it and the element 11 strictly greater than it. Element 11 has element 7 strictly smaller than it and element 15 strictly greater than it. In total there are 2 elements having both a strictly smaller and a strictly greater element appear in

nums

.

Example 2:

Input:

nums = [-3,3,3,90]

Output:

2

Explanation:

The element 3 has the element -3 strictly smaller than it and the element 90 strictly greater than it. Since there are two elements with the value 3, in total there are 2 elements having both a strictly smaller and a strictly greater element appear in

nums

.

Constraints:

1 <= nums.length <= 100

-10

5

<= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
```

```
 * @return {number}
 */
var countElements = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Elements With Strictly Smaller and Greater Elements
* Difficulty: Easy
* Tags: array, sort
*
* 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 countElements(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Elements With Strictly Smaller and Greater Elements
* Difficulty: Easy
* Tags: array, sort
*
* 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 countElements(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Elements With Strictly Smaller and Greater Elements
Difficulty: Easy
Tags: array, sort

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 countElements(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Elements With Strictly Smaller and Greater Elements
* Difficulty: Easy
* Tags: array, sort
*
* 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
* @return {number}
*/
var countElements = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Elements With Strictly Smaller and Greater Elements
* Difficulty: Easy
* Tags: array, sort
*
* 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 countElements(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Elements With Strictly Smaller and Greater Elements
* Difficulty: Easy
* Tags: array, sort
*
* 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 CountElements(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Elements With Strictly Smaller and Greater Elements
* Difficulty: Easy
* Tags: array, sort
*
* 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 countElements(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Elements With Strictly Smaller and Greater Elements
// Difficulty: Easy
// Tags: array, sort
//
// 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 countElements(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Elements With Strictly Smaller and Greater Elements
// Difficulty: Easy
// Tags: array, sort
//
// 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_elements(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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