Problem 315: Count of Smaller Numbers After Self

Problem Information

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

Problem Description

Given an integer array

nums

, return

an integer array

counts

where

counts[i]

is the number of smaller elements to the right of

nums[i]

.

Example 1:

Input:

nums = [5,2,6,1]

Output:

[2,1,1,0]

Explanation:

To the right of 5 there are

2

smaller elements (2 and 1). To the right of 2 there is only

1

smaller element (1). To the right of 6 there is

1

smaller element (1). To the right of 1 there is

0

smaller element.

Example 2:

Input:

nums = [-1]

Output:

[0]

Example 3:

Input:

nums = [-1,-1]

Output:

[0,0]

Constraints:

1 <= nums.length <= 10

5

-10

4

<= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> countSmaller(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<int> CountSmaller(int[] nums) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* countSmaller(int* nums, int numsSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun countSmaller(nums: IntArray): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count of Smaller Numbers After Self
* 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:
vector<int> countSmaller(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count of Smaller Numbers After Self
* 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 List<Integer> countSmaller(int[] nums) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Count of Smaller Numbers After Self
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 countSmaller(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count of Smaller Numbers After Self
* 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[]} nums
* @return {number[]}
*/
var countSmaller = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count of Smaller Numbers After Self
* 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 countSmaller(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Count of Smaller Numbers After Self
* 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 IList<int> CountSmaller(int[] nums) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Count of Smaller Numbers After Self
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* countSmaller(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Count of Smaller Numbers After Self
// 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 countSmaller(nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun countSmaller(nums: IntArray): List<Int> {

}
```

```
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count of Smaller Numbers After Self
// 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 count_smaller(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
```

```
*/
function countSmaller($nums) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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

```
)
```