Problem 327: Count of Range Sum

Problem Information

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

Problem Description

Given an integer array

nums

and two integers

lower

and

upper

, return

the number of range sums that lie in

[lower, upper]

inclusive

.

Range sum

S(i, j)

is defined as the sum of the elements in

nums

between indices

i

and

j

inclusive, where

i <= j

.

Example 1:

Input:

nums = [-2,5,-1], lower = -2, upper = 2

Output:

3

Explanation:

The three ranges are: [0,0], [2,2], and [0,2] and their respective sums are: -2, -1, 2.

Example 2:

Input:

nums = [0], lower = 0, upper = 0

Output:

1

Constraints:

1 <= nums.length <= 10

5

-2

31

<= nums[i] <= 2

31

- 1

-10

5

<= lower <= upper <= 10

5

The answer is

guaranteed

to fit in a

32-bit

integer.

Code Snippets

C++:

```
class Solution {
public:
int countRangeSum(vector<int>& nums, int lower, int upper) {

}
};
```

Java:

```
class Solution {
public int countRangeSum(int[] nums, int lower, int upper) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} lower
* @param {number} upper
* @return {number}
*/
var countRangeSum = function(nums, lower, upper) {
```

```
};
```

TypeScript:

```
function countRangeSum(nums: number[], lower: number, upper: number): number
{

};
```

C#:

```
public class Solution {
public int CountRangeSum(int[] nums, int lower, int upper) {

}
}
```

C:

```
int countRangeSum(int* nums, int numsSize, int lower, int upper) {

}
```

Go:

```
func countRangeSum(nums []int, lower int, upper int) int {

}
```

Kotlin:

```
class Solution {
fun countRangeSum(nums: IntArray, lower: Int, upper: Int): Int {

}
}
```

Swift:

```
class Solution {
func countRangeSum(_ nums: [Int], _ lower: Int, _ upper: Int) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn count_range_sum(nums: Vec<i32>, lower: i32, upper: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} lower
# @param {Integer} upper
# @return {Integer}
def count_range_sum(nums, lower, upper)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $lower
* @param Integer $upper
* @return Integer
*/
function countRangeSum($nums, $lower, $upper) {

}
}
```

Dart:

```
class Solution {
int countRangeSum(List<int> nums, int lower, int upper) {

}
```

```
}
```

Scala:

```
object Solution {
def countRangeSum(nums: Array[Int], lower: Int, upper: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_range_sum(nums :: [integer], lower :: integer, upper :: integer)
:: integer
def count_range_sum(nums, lower, upper) do

end
end
```

Erlang:

```
-spec count_range_sum(Nums :: [integer()], Lower :: integer(), Upper ::
integer()) -> integer().
count_range_sum(Nums, Lower, Upper) ->
.
```

Racket:

```
(define/contract (count-range-sum nums lower upper)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count of Range Sum
* 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:
int countRangeSum(vector<int>& nums, int lower, int upper) {

}
};
```

Java Solution:

```
/**
* Problem: Count of Range Sum
* 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 int countRangeSum(int[] nums, int lower, int upper) {

}
}
```

Python3 Solution:

```
"""
Problem: Count of Range Sum
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 countRangeSum(self, nums: List[int], lower: int, upper: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count of Range Sum
* 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
* @param {number} lower
* @param {number} upper
* @return {number}
*/
var countRangeSum = function(nums, lower, upper) {

};
```

TypeScript Solution:

```
/**
* Problem: Count of Range Sum
* 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 countRangeSum(nums: number[], lower: number, upper: number): number
{

};
```

C# Solution:

```
/*
* Problem: Count of Range Sum
* 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 int CountRangeSum(int[] nums, int lower, int upper) {

}
}
```

C Solution:

```
/*
* Problem: Count of Range Sum
* 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
*/

int countRangeSum(int* nums, int numsSize, int lower, int upper) {

}
```

Go Solution:

```
// Problem: Count of Range Sum
// 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 countRangeSum(nums []int, lower int, upper int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countRangeSum(nums: IntArray, lower: Int, upper: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countRangeSum(_ nums: [Int], _ lower: Int, _ upper: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count of Range Sum
// 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_range_sum(nums: Vec<i32>, lower: i32, upper: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} lower
# @param {Integer} upper
# @return {Integer}
def count_range_sum(nums, lower, upper)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $lower
* @param Integer $upper
* @return Integer
*/
function countRangeSum($nums, $lower, $upper) {

}
}
```

Dart Solution:

```
class Solution {
int countRangeSum(List<int> nums, int lower, int upper) {

}
}
```

Scala Solution:

```
object Solution {
def countRangeSum(nums: Array[Int], lower: Int, upper: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_range_sum(nums :: [integer], lower :: integer, upper :: integer)
:: integer
def count_range_sum(nums, lower, upper) do

end
end
```

Erlang Solution:

```
-spec count_range_sum(Nums :: [integer()], Lower :: integer(), Upper ::
integer()) -> integer().
count_range_sum(Nums, Lower, Upper) ->
.
```

Racket Solution:

```
(define/contract (count-range-sum nums lower upper)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```