Problem 1508: Range Sum of Sorted Subarray Sums

Problem Information

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

Problem Description

You are given the array

nums

consisting of

n

positive integers. You computed the sum of all non-empty continuous subarrays from the array and then sorted them in non-decreasing order, creating a new array of

n * (n + 1) / 2

numbers.

Return the sum of the numbers from index

left

to index

right

(

indexed from 1

)

, inclusive, in the new array.

Since the answer can be a huge number return it modulo

10

9

+ 7

.

Example 1:

Input:

nums = [1,2,3,4], n = 4, left = 1, right = 5

Output:

13

Explanation:

All subarray sums are 1, 3, 6, 10, 2, 5, 9, 3, 7, 4. After sorting them in non-decreasing order we have the new array [1, 2, 3, 3, 4, 5, 6, 7, 9, 10]. The sum of the numbers from index le = 1 to ri = 5 is 1 + 2 + 3 + 3 + 4 = 13.

Example 2:

Input:

nums = [1,2,3,4], n = 4, left = 3, right = 4

Output:

6

Explanation:

The given array is the same as example 1. We have the new array [1, 2, 3, 3, 4, 5, 6, 7, 9, 10]. The sum of the numbers from index le = 3 to ri = 4 is 3 + 3 = 6.

Example 3:

Input:

nums = [1,2,3,4], n = 4, left = 1, right = 10

Output:

50

Constraints:

n == nums.length

1 <= nums.length <= 1000

1 <= nums[i] <= 100

1 <= left <= right <= n * (n + 1) / 2

Code Snippets

C++:

```
class Solution {
public:
int rangeSum(vector<int>& nums, int n, int left, int right) {

}
};
```

Java:

```
class Solution {
public int rangeSum(int[] nums, int n, int left, int right) {

}
}
```

Python3:

```
class Solution:
def rangeSum(self, nums: List[int], n: int, left: int, right: int) -> int:
```

Python:

```
class Solution(object):
def rangeSum(self, nums, n, left, right):
"""
:type nums: List[int]
:type n: int
:type left: int
:type right: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} n
* @param {number} left
* @param {number} right
* @return {number}
*/
var rangeSum = function(nums, n, left, right) {

};
```

TypeScript:

```
function rangeSum(nums: number[], n: number, left: number, right: number):
number {
```

```
};
```

C#:

```
public class Solution {
public int RangeSum(int[] nums, int n, int left, int right) {

}
}
```

C:

```
int rangeSum(int* nums, int numsSize, int n, int left, int right) {

}
```

Go:

```
func rangeSum(nums []int, n int, left int, right int) int {

}
```

Kotlin:

```
class Solution {
fun rangeSum(nums: IntArray, n: Int, left: Int, right: Int): Int {

}
}
```

Swift:

```
class Solution {
func rangeSum(_ nums: [Int], _ n: Int, _ left: Int, _ right: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn range_sum(nums: Vec<i32>, n: i32, left: i32, right: i32) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} n
# @param {Integer} left
# @param {Integer} right
# @return {Integer}
def range_sum(nums, n, left, right)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $n
* @param Integer $left
* @param Integer $right
* @return Integer
*/
function rangeSum($nums, $n, $left, $right) {

}
}
```

Dart:

```
class Solution {
int rangeSum(List<int> nums, int n, int left, int right) {

}
}
```

Scala:

```
object Solution {
def rangeSum(nums: Array[Int], n: Int, left: Int, right: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec range_sum(nums :: [integer], n :: integer, left :: integer, right ::
integer) :: integer
def range_sum(nums, n, left, right) do

end
end
```

Erlang:

```
-spec range_sum(Nums :: [integer()], N :: integer(), Left :: integer(), Right
:: integer()) -> integer().
range_sum(Nums, N, Left, Right) ->
.
```

Racket:

```
(define/contract (range-sum nums n left right)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Range Sum of Sorted Subarray Sums
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 rangeSum(vector<int>& nums, int n, int left, int right) {

}
};
```

Java Solution:

```
/**
* Problem: Range Sum of Sorted Subarray Sums
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 rangeSum(int[] nums, int n, int left, int right) {

}
}
```

Python3 Solution:

```
"""
Problem: Range Sum of Sorted Subarray Sums
Difficulty: Medium
Tags: array, sort, search

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 rangeSum(self, nums: List[int], n: int, left: int, right: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def rangeSum(self, nums, n, left, right):
"""
:type nums: List[int]
:type n: int
:type left: int
:type right: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Range Sum of Sorted Subarray Sums
* Difficulty: Medium
* Tags: array, sort, search
*
* 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
* @param {number} n
* @param {number} left
* @param {number} right
* @return {number}
*/
var rangeSum = function(nums, n, left, right) {

};
```

TypeScript Solution:

```
/**
* Problem: Range Sum of Sorted Subarray Sums
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 rangeSum(nums: number[], n: number, left: number, right: number):
number {

};
```

C# Solution:

```
/*
* Problem: Range Sum of Sorted Subarray Sums
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 RangeSum(int[] nums, int n, int left, int right) {

}
}
```

C Solution:

```
/*
* Problem: Range Sum of Sorted Subarray Sums
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 rangeSum(int* nums, int numsSize, int n, int left, int right) {

}
```

Go Solution:

```
// Problem: Range Sum of Sorted Subarray Sums
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 rangeSum(nums []int, n int, left int, right int) int {

}
```

Kotlin Solution:

```
class Solution {
fun rangeSum(nums: IntArray, n: Int, left: Int, right: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func rangeSum(_ nums: [Int], _ n: Int, _ left: Int, _ right: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Range Sum of Sorted Subarray Sums
// Difficulty: Medium
```

```
// Tags: array, sort, search
//
// 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 range_sum(nums: Vec<i32>, n: i32, left: i32, right: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} n
# @param {Integer} left
# @param {Integer} right
# @return {Integer}
def range_sum(nums, n, left, right)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $n
* @param Integer $left
* @param Integer $right
* @return Integer
*/
function rangeSum($nums, $n, $left, $right) {

}
}
```

Dart Solution:

```
class Solution {
int rangeSum(List<int> nums, int n, int left, int right) {

}
}
```

Scala Solution:

```
object Solution {
def rangeSum(nums: Array[Int], n: Int, left: Int, right: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec range_sum(nums :: [integer], n :: integer, left :: integer, right ::
integer) :: integer
def range_sum(nums, n, left, right) do

end
end
```

Erlang Solution:

```
-spec range_sum(Nums :: [integer()], N :: integer(), Left :: integer(), Right
:: integer()) -> integer().
range_sum(Nums, N, Left, Right) ->
.
```

Racket Solution:

```
(define/contract (range-sum nums n left right)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```