Range Sum of Sorted Subarray Sums

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.

Algorithm

1.) **Initialize Constants**:

- Define mod as 109+710^9 + 7109+7 to handle large sums and prevent overflow.

2.) **Generate All Subarray Sums**:

- Create a list subarraySums to store the sums of all possible subarrays.
- Use two nested loops to calculate the sums of all subarrays:
    - The outer loop iterates over each starting index i of the subarray.
    - The inner loop calculates the sum of subarrays starting at i and ending at j.

3.) **Sort the Subarray Sums**:

- Sort the subarraySums list to get all subarray sums in ascending order.

4.) **Calculate the Sum in the Given Range**:

- Initialize a variable result to 0.
- Iterate over the indices from left-1 to right-1 in the sorted list, summing the elements in this range and applying the modulo operation to handle large numbers.

5.) **Return the Result**:

- Return the final sum as an integer.

Pseudocode

```
function rangeSum(nums: array of int, n: int, left: int, right: int) -> int:
    mod = 1000000007
    subarraySums = []

    # Generate all subarray sums
    for i from 0 to n - 1:
        sum = 0
        for j from i to n - 1:
            sum += nums[j]
            subarraySums.append(sum)

    # Sort the subarray sums
    sort(subarraySums)

    # Calculate the sum from left to right
    result = 0
    for i from left - 1 to right - 1:
        result = (result + subarraySums[i]) % mod

    return int(result)
```

Code

```
class Solution {
    public int rangeSum(int[] nums, int n, int left, int right) {
        int mod = 1000000007;
        List<Integer> subarraySums = new ArrayList<>();

        // All possible subarrays sums
        for(int i = 0; i < n; i++){
            int sum = 0;
            for(int j = i; j < n; j++){
                sum += nums[j];
                subarraySums.add(sum);
            }
        }

        Collections.sort(subarraySums);

        // Calculate the sum from left to right
        long result = 0;
        for(int i = left-1; i<right; i++){
            result = (result + subarraySums.get(i)) % mod;
        }
        return (int) result;
    }
}
```

Conclusion

The rangeSum function efficiently calculates the sum of elements in a specified range of the sorted list of all possible subarray sums. By generating all subarray sums, sorting them, and then summing the elements in the specified range, the function ensures accurate results while handling potential overflow with the modulo operation. This approach guarantees that the function can handle large input sizes and return the correct sum within the given constraints.