03.01.2025

Problem: Number of Ways to Split Array (2270)

Problem Statement:
You are given a 0-indexed integer array `nums` of length `n`.
A valid split at index `i` occurs when:

1. The sum of the first `i + 1` elements is greater than or equal to the sum of the remaining elements.
2. There must be at least one element to the right of `i` (`0 <= i < n - 1`).

Goal: Return the number of valid splits in the array.

Example 1:
`nums = [10, 4, -8, 7]`

- Split at index 0 → `[10]` and `[4, -8, 7]` → `10 >= 3` → Valid
- Split at index 1 → `[10, 4]` and `[-8, 7]` → `14 >= -1` → Valid
- Split at index 2 → `[10, 4, -8]` and `[7]` → `6 < 7` → Invalid
  Output: 2

Example 2:
`nums = [2, 3, 1, 0]`

- Split at index 1 → `[2, 3]` and `[1, 0]` → `5 >= 1` → Valid
- Split at index 2 → `[2, 3, 1]` and `[0]` → `6 >= 0` → Valid
  Output: 2

Approach:

Key Idea:

- Use **prefix sum** to keep track of cumulative sums.
- Calculate the total sum of the array once and use the prefix sum to compare both parts for every split.

Code Explanation:

```cpp
class Solution {
public:
    int waysToSplitArray(vector<int>& nums) {
        // Initialize the prefix sum array and count for valid splits
        vector<long long> prefixSum(nums.size());
        prefixSum[0] = nums[0];  // First element is the same as the prefix sum for index 0
        int count = 0;           // Count to store the number of valid splits
        long long sum = nums[0]; // To keep track of cumulative sum

        // Step 1: Build the prefix sum array
        for(int i = 1; i < nums.size(); i++) {
            sum += nums[i];             // Update the cumulative sum
            prefixSum[i] = sum;         // Store it in the prefix sum array
        }

        // Step 2: Calculate the total sum of the entire array
        long long totalSum = prefixSum[nums.size() - 1];

        // Step 3: Check valid splits using the prefix sum array
        for(int i = 0; i < prefixSum.size() - 1; i++) {
            // If the sum of the left part is greater than or equal to the sum of the right part
            if(prefixSum[i] >= totalSum - prefixSum[i]) {
                count++;  // Increment valid split count
            }
        }

        // Return the number of valid splits
        return count;
    }
};
```

Step-by-Step Explanation:

Step 1: Build the prefix sum array.

- `prefixSum[i]` = Sum of elements from index `0` to `i`.

Step 2: Compute the total sum of the array.

Step 3: Check the condition for each possible split.

- If the sum of the left part (`prefixSum[i]`) is greater than or equal to the sum of the right part (`totalSum - prefixSum[i]`), increment the count.

Step 4: Return the count of valid splits.

Example Walkthrough:

Input: `nums = [10, 4, -8, 7]`
Prefix Sum Construction:

- `prefixSum = [10, 14, 6, 13]`
  Total Sum: `13`

Checking Splits:

- Split at index `0`: `10 >= 3` → Valid
- Split at index `1`: `14 >= -1` → Valid
- Split at index `2`: `6 < 7` → Invalid

Output: `2`

Complexity Analysis:

- **Time Complexity:** `O(n)` (Building the prefix sum array and checking all splits takes linear time)
- **Space Complexity:** `O(n)` (Storing the prefix sum array)

Edge Cases Handled:

- If `nums` has only 2 elements → Always returns `1` (Only one valid split is possible).
- Negative numbers and zero are correctly handled.

Key Takeaways:

- The prefix sum technique efficiently reduces the complexity from `O(n^2)` to `O(n)`.
- This problem is a great example of how **cumulative sums** can simplify range queries.