2219. Maximum Sum Score of Array

Medium Array Prefix Sum Leetcode Link

Problem Description

In this problem, you're provided with an array of integers named `nums` indexed from 0 to `n-1`, where `n` is the length of the array. You need to calculate what is called the **sum score** for each index of the array. The **sum score** at a particular index `i` is defined as the maximum between two sums:

- The sum of elements from the start of the array up through index `i`.
- The sum of elements from index `i` through the end of the array.

Your task is to determine the maximum **sum score** that can be obtained at any index `i` in the provided array.

Intuition

The key to solving this problem lies in understanding prefix and suffix sums, which are cumulative sums of the elements of the array from the beginning up to a certain element, and from a certain element to the end, respectively.

1. Calculate the prefix sum for the array, which gives you the sum of elements from the start of the array up to each index `i`. This is stored in an array `s` where `s[i]` would represent the sum of elements from `nums[0]` to `nums[i-1]`.

2. Iterate through the array to calculate the sum score at each index. This can be done in a single pass after the prefix sum array has been built:

   - For each index `i`, determine the sum from the start of the array to index `i`. This is given by the prefix sum at index `i + 1` (since we initialized prefix sum with `0` at the beginning).
   - To determine the sum from index `i` to the end of the array, subtract the prefix sum up to index `i` from the total sum of the array, which is the last element of the prefix sum array.
   - The sum score at index `i` is the maximum of these two sums.
3. After calculating the sum scores for each index, the maximum sum score is the answer, which represents the maximum value obtained from any index `i`.

By using a prefix sum array and iterating through the array only once, we can solve this problem efficiently with a time complexity of O(n), where n is the length of the input array.

Solution Approach

The given solution implements the intuition behind the problem in Python. It utilizes the `accumulate` function from the `itertools` module to generate prefix sums efficiently and a simple loop to compare sums at each index.

Here is a step-by-step explanation of the provided code:

Step 1: Calculate Prefix Sums

The line `s = [0] + list(accumulate(nums))` is crucial. It creates a new list `s` that stores the prefix sums of the `nums` array. The `accumulate` function takes each element and adds it to the sum of all the previous elements. The `[0]` at the start of this list is to simplify calculations for the prefix sum at index 0. After this line, `s[i]` will contain the sum of `nums` from `nums[0]` up to `nums[i-1]`.

Step 2: Iterate to Find Maximum Sum Score

The expression `(max(s[i + 1], s[-1] - s[i]) for i in range(len(nums)))` uses a generator to calculate sum scores without storing them. For each index `i` in `nums`, it calculates two sums:

- `s[i + 1]` gives us the sum of elements from the beginning of the array to index `i` (prefix sum).
- `s[-1] - s[i]` gives us the sum of elements from index `i` to the end of the array (this works because `s[-1]` represents the total sum of the array, which is the last element in the prefix sums list).

Step 3: Find the Maximum Value

The `max` function is used again to find the maximum sum score from the generator. This is the maximum value that can be obtained from any of the sum scores calculated in the previous step.

The result of the `max` function call is returned as the final answer, representing the overall maximum sum score at any index `i` in the input `nums` array.

Algorithm Pattern

The algorithm follows a pattern often used for solving cumulative sum problems: it first preprocesses the input array to build a data structure (in this case, a prefix sum array), enabling efficient calculations of subarray sums. It then iterates through the array a single time to find the desired maximum value.

Data Structure

An array (or list in Python) is the primary data structure used here for storing the prefix sums.

Time Complexity

Since each of the steps above runs in O(n) time, where `n` is the length of the `nums` array, and there are no nested loops, the overall time complexity of the function is O(n).

Example Walkthrough

Let's walk through an example to illustrate the solution approach.

Consider the array `nums = [3, 1, -2, 5, 2]`. Our goal is to find the maximum sum score at any index `i` after performing the calculations as outlined in the problem description.

Step 1: Calculate Prefix Sums

First, we calculate the prefix sums for the array `nums`.

1. Initialize an array `s` with an extra `0` at the beginning to simplify the prefix sum calculations.
2. Using the `accumulate` function from the `itertools` module, we generate the prefix sums of `nums`. This results in the following array:

   ```
   s = [0] + list(accumulate([3, 1, -2, 5, 2])) = [0, 3, 4, 2, 7, 9]
   ```

   In this array, `s[i]` represents the sum of elements from `nums[0]` to `nums[i-1]`. For example, `s[3] = 2` corresponds to the sum of elements `3 + 1 - 2`.

Step 2: Iterate to Find Maximum Sum Score

Next, we iterate through the array to calculate the sum score at each index `i`.

- For index `i = 0`, the sum from the start to index `0` is `s[1] = 3`, and from index `0` to the end is `s[-1] - s[0] = 9 - 0 = 9`. The sum score for index `0` is the maximum of these two, which is `9`.

- For index `i = 1`, the sum from the start to index `1` is `s[2] = 4`, and from index `1` to the end is `s[-1] - s[1] = 9 - 3 = 6`. The sum score for index `1` is the maximum, which is `6`.

- Continuing in this way for each index, we compute the sum score:

  ```
  Index 0: max(3, 9) = 9
  Index 1: max(4, 6) = 6
  Index 2: max(4, 7) = 7
  Index 3: max(2, 7) = 7
  Index 4: max(7, 2) = 7
  ```

Step 3: Find the Maximum Value

Finally, we find the maximum sum score from all the calculated sum scores, which is `9` (obtained at index `0`).

Using this approach, we can efficiently determine that the maximum sum score for the array `[3, 1, -2, 5, 2]` is `9`. The solution only iterates through the array once (after computing the prefix sums), thus ensuring a time complexity of O(n).

Python Solution

```python
from typing import List
from itertools import accumulate

class Solution:
    def maximum_sum_score(self, nums: List[int]) -> int:
        # Prefix sum array initialization with an extra zero at the beginning
        # to handle the case where we accumulate from the start of the array.
        prefix_sums = [0] + list(accumulate(nums))

        # Calculate the maximum sum score by iterating through each element
        # in the nums array while taking into account both the sum of elements
        # before the current element (prefix sum) and the sum of elements
        # after the current element (suffix sum).
        max_score = max(
            max(prefix_sums[i + 1], prefix_sums[-1] - prefix_sums[i])
            for i in range(len(nums))
        )

        # Return the maximum sum score found.
        return max_score

```

Java Solution

```java
class Solution {
    public long maximumSumScore(int[] nums) {
        int length = nums.length;
        // Create an array to store the prefix sums
        long[] prefixSums = new long[length + 1];

        // Calculate the prefix sums
        for (int i = 0; i < length; ++i) {
            prefixSums[i + 1] = prefixSums[i] + nums[i];
        }

        // Initialize the maximum sum as the smallest possible value
        long maxSum = Long.MIN_VALUE;

        // Find the maximum sum score by choosing the larger between
        // the sum from the start to the current element or
        // the sum from the current element to the end
        for (int i = 0; i < length; ++i) {
            long sumFromStart = prefixSums[i + 1];
            long sumFromEnd = prefixSums[length] - prefixSums[i];
            maxSum = Math.max(maxSum, Math.max(sumFromStart, sumFromEnd));
        }

        // Return the maximum score found
        return maxSum;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <algorithm> // for std::max
#include <climits> // For INT_MIN constant

class Solution {
public:
    long long maximumSumScore(vector<int>& nums) {
        int n = nums.size(); // Get the size of the input vector
        vector<long long> prefixSums(n + 1, 0); // Initialize prefix sums vector with an extra element

        // Calculate prefix sums for the entire array
        for (int i = 0; i < n; ++i) {
            prefixSums[i + 1] = prefixSums[i] + nums[i];
        }

        long long maxScore = LLONG_MIN; // Initialize the maxScore with the smallest possible value for long long

        // Iterate through the array to find the maximum sum score
        for (int i = 0; i < n; ++i) {
            // For each element, we consider two cases:
            // 1. The sum of elements from start up to i (prefixSums[i + 1])
            // 2. The sum of elements from i to the end of the array (prefixSums[n] - prefixSums[i])
            // The maximum of these two values is compared with the maxScore to update it
            maxScore = max(maxScore, max(prefixSums[i + 1], prefixSums[n] - prefixSums[i]));
        }

        // Return the maximum sum score
        return maxScore;
    }
};

```

Typescript Solution

```typescript
function maximumSumScore(nums: number[]): number {
    const numElements = nums.length; // Total number of elements in the array

    // Create and initialize a prefix sum array with an additional
    // first element set to 0 for ease of calculation
    let prefixSums = new Array(numElements + 1).fill(0);

    // Populate the prefix sum array with cumulative sums,
    // where prefixSums[i] will hold the sum of nums[0] to nums[i - 1]
    for (let i = 0; i < numElements; ++i) {
        prefixSums[i + 1] = prefixSums[i] + nums[i];
    }

    // Initialize the answer variable to the minimum possible value
    // since we are looking for the maximum
    let maxSumScore = -Infinity;

    // Iterate through each element and calculate the sum score
    // by taking the higher value between:
    // 1. The sum of elements from the start to the current index (inclusive)
    // 2. The sum of elements from the current index (excluding) to the end
    for (let i = 0; i < numElements; ++i) {
        // Compare the sum score at each index with the maxSumScore found so far
        maxSumScore = Math.max(
            maxSumScore,
            Math.max(prefixSums[i + 1], prefixSums[numElements] - prefixSums[i])
        );
    }

    // Return the maximum sum score found
    return maxSumScore;
}

```

Time and Space Complexity

The time complexity of the given code is `O(n)`, where `n` is the number of elements in the input list `nums`. This is due to the following reasons:

1. We first precompute the prefix sums with `accumulate(nums)`, which takes `O(n)` time.
2. Then, we perform a single pass over the `nums` list to calculate the maximum sum score. During each iteration, we calculate the maximum between `s[i + 1]` and `s[-1] - s[i]`. There are `n` such calculations, each taking constant time.

The space complexity of the code is `O(n)`. This is because we are creating a new list called `s` that contains the prefix sums of the `nums` array, which is of size `n + 1`.