1685. Sum of Absolute Differences in a Sorted Array Medium <u>Array</u> <u>Math</u> <u>Prefix Sum</u>

```
Problem Description
```

In this problem, we have an array nums that is sorted in non-decreasing order. Our goal is to construct a new array result of the same length as nums. Each element result[i] in this new array should be the sum of the absolute differences between nums[i] and every other element in the array nums. Essentially, for a given element nums[i], we calculate the absolute value of the difference between nums[i] and each element nums[j] where j goes from 0 to the length of the array but is not equal to i. Then we add up all these absolute differences to get result[i].

The direct approach would be to calculate the sum of absolute differences for each element in the array by iterating over the

array for each element, which would be very inefficient with a time complexity of O(n^2). Instead, we can utilize the fact that the

array is sorted and leverage a prefix sum technique that will allow us to calculate the sum of absolute differences in a more

Intuition

efficient manner. Intuition behind the solution: Since the array is sorted, all numbers to the left of nums[i] are less than or equal to nums[i], and all numbers to the right are greater than or equal to it. So, for nums[i], to get the sum of absolute differences, we can sum the differences between nums[i] and all numbers to its left, and then sum the differences between nums[i] and all numbers to its

right. The key insight here is to realize that we can compute the sum of absolute differences for nums[i] using Cumulative Sum (also known as Prefix Sum). A Cumulative Sum is an array cumsum where cumsum[i] represents the sum of the elements nums[0] up to nums[i]. With this, the sum of differences to the left of nums[i] can be found by nums[i] * i - cumsum[i-1] (since nums[i] is

larger than each of these numbers) and the sum of the differences to the right of nums[i] can be found by (cumsum[n-1] cumsum[i]) - nums[i] * (n-1-i) (since nums[i] is smaller than each of these numbers, subtracting it from the cumulative sum gives us the sum of the differences). By adding these two quantities together, we get the sum of absolute differences for each nums[i].

In the provided solution, instead of keeping a separate array for the cumulative sum, the sum of all elements s and running total

of sums t are used to avoid extra space. We iterate through the array, updating t and calculating the sum of absolute

Solution Approach Understanding the provided Python code requires us to focus on how it implements the prefix sum technique described in the Intuition.

3. Initialize a variable t to keep track of the cumulative sum (or running total) of the elements as we iterate through the list. This will be used to calculate the sum of all elements to the left of the current element.

differences on the fly.

4. Loop through the array nums using enumerate to get both the index i and the value x at each iteration. 5. For current element x, calculate the sum of absolute differences using the following steps: • Calculate the sum of differences between x and all the elements on its left (the smaller elements). We can get this by multiplying x with its index i, which gives us the sum of x added i times, and subtracting the running total t which represents the sum of all elements up to x.

o Calculate the sum of differences between x and all the elements on its right (the larger elements). This is done by subtracting the running

total t and x itself from the total sum s to get the sum of all elements to the right of x, and then subtracting x multiplied by the number of elements to the right of it (len(nums) - i).

Add both sums to get the total sum of absolute differences for x and append this value to ans.

1. Initialize an empty list ans, which will be our result array to store the summation of absolute differences for each element.

2. Compute the total sum s of the array nums. This will be used to calculate the sum of all elements to the right of the current element.

7. After the loop is completed, return ans as the result array. The step where the absolute difference is calculated, the mathematical formulation is as follows:

Putting the pieces together, this approach efficiently calculates the desired values with a time complexity of O(n), significantly

reducing the computational load compared to a naive O(n^2) method of calculating absolute differences for each element

t for the running total and s as the total sum. By doing this, it's possible to compute the answer in a single pass after calculating

- This line of the code can be broken down into parts:
- + s t: Total sum minus the cumulative sum to get sum of all elements to the right. • $- \times * (len(nums) - i)$: Subtract x times the count of elements to the right.

• - t: Subtract the cumulative sum of elements to the left.

the sum of the array just once at the beginning.

6. Update the running total t by adding the current element value x to it.

v = x * i - t + s - t - x * (len(nums) - i)

• x * i: The sum of x added i times.

individually.

Example Walkthrough

First, let's go through the steps involved in the algorithm:

The algorithm cleverly handles the calculation without using additional space for a cumulative sum array by updating and utilizing

The total sum s of nums is calculated as 1 + 3 + 6 + 10 = 20.

We set t to 0 which will hold the cumulative sum as we iterate.

Now we iterate through nums. Here are the detailed steps:

• We update t by adding x to it: t = 0 + 1 = 1.

■ The total is 20 + 0 = 20. Append 20 to ans.

a much more efficient calculation with a linear time complexity.

def getSumAbsoluteDifferences(self, nums: List[int]) -> List[int]:

Iterate through the list, with both the index and the value.

elements to the left of 'value' (including itself).

Update the prefix sum with the current value.

// Initialize the total sum of the array elements to 0

Compute the value for the current index.

Calculate the sum of the entire array to use later in calculations.

'value * index' is the sum of 'value' being subtracted from all

'total sum - prefix_sum' is the sum of all the elements to the

// Loop through each element to calculate the sum of absolute differences

// Calculate the sum of absolute difference for the current element

// The sum of differences of all elements to the left of the current

// element can be found by nums[i] * i (since there are 'i' elements to the left)

// Similarly, the sum of differences of all elements to the right of the current

prefixSum += nums[i]; // Update the prefix sum with the current element's value

// element is the total sum of all elements minus the sum of all elements till the

// current index (including itself), minus the sum total of the elements to the left

result[i] = sumAbsoluteDifferences; // Assign the computed value to the result array

// minus the sum of all elements to the left which is given by prefixSum (t in your code).

// times the number of elements to the right: (totalSum - prefixSum - nums[i]) * (n - i).

int sumAbsoluteDifferences = nums[i] * i - prefixSum + totalSum - prefixSum - nums[i] * (n - i);

Initialize a list to store the answers.

```
Following the solution steps:
    We initialize our result array ans as an empty list.
```

Let's take a small example to illustrate the solution approach. Suppose our array nums is [1, 3, 6, 10].

First iteration (i=0, x=1): ■ Calculate the left sum: x*i - t which is 1*0 - 0 = 0 because there are no elements to the left of the first element. ■ Calculate the right sum: (s - t) - x*(len(nums)-i) which is (20 - 0) - 1*(4-0) = 20 - 1*4 = 16. ■ The total sum of absolute differences for x is 0 + 16 = 16. So we append 16 to ans.

■ The total sum of differences is 2 + 10 = 12. Append 12 to ans. Now our ans is [16, 12].

Python

from typing import List

answers = []

prefix_sum = 0

total_sum = sum(nums)

Initialize a prefix sum to 0.

right of 'value'.

prefix_sum += value

Return the list of computed answers.

public int[] getSumAbsoluteDifferences(int[] nums) {

// Calculate the total sum of the array elements

for index, value in enumerate(nums):

class Solution:

Now ans is [16].

Second iteration (i=1, x=3):

• Update t: t = 1 + 3 = 4. Third iteration (i=2, x=6):

■ Left sum: x*i - t which is 3*1 - 1 = 2 since the sum of all elements to the left of 3 is 1.

■ Right sum: (s - t) - x*(len(nums)-i) which is (20 - 1) - 3*(4-1) = 19 - 3*3 = 10.

- Right sum: (s t) x*(len(nums)-i) which is (20 4) 6*(4-2) = 16 6*2 = 4. ■ The total is 8 + 4 = 12. Append 12 to ans.
- Our ans is now [16, 12, 12]. • Update t: t = 4 + 6 = 10.

Right sum: There are no elements to the right of the last element, hence the value is 0.

We've completed the loop and now ans holds our final result, which is [16, 12, 12, 20].

Fourth iteration (i=3, x=10):

■ Left sum: x*i - t which is 10*3 - 10 = 20 since the sum of 1 + 3 + 6 is 10.

■ Left sum: x*i - t which is 6*2 - 4 = 8 since the sum of 1 + 3 is 4.

- Now ans is [16, 12, 12, 20]. There's no need to update t as we're at the end of the array.
- Solution Implementation

Subtract 'value * (len(nums) - index)', which is 'value' being # added to all elements to the right of 'value'. current_value = value * index - prefix_sum + total_sum - prefix_sum - value * (len(nums) - index) # Append the computed value to the answer list. answers.append(current_value)

int prefixSum = 0; // Initialize the prefix sum to keep track of the sum of elements till the current index

Thus, the array [1, 3, 6, 10] transforms into [16, 12, 12, 20] following our solution approach. Each element in the result

array represents the sum of the absolute differences between that element in nums and all other elements. This process provides

```
totalSum += num;
int n = nums.length; // Store the length of the array
int[] result = new int[n]; // Initialize the result array to store answers
```

int totalSum = 0;

for (int num : nums) {

for (int i = 0; i < n; ++i) {

return answers

Java

class Solution {

```
// Return the final result array
        return result;
C++
#include <vector>
#include <numeric> // For std::accumulate
class Solution {
public:
    // Function to calculate the sum of absolute differences of each element
    vector<int> getSumAbsoluteDifferences(vector<int>& nums) {
        // Calculate the sum of the entire array
        int total sum = std::accumulate(nums.begin(), nums.end(), 0);
        // Initialize variable to keep track of the sum of elements seen so far
        int partial sum = 0;
        // Get number of elements in the array
        int n = nums.size();
        // Initialize result vector with the same size as input
        vector<int> result(n);
        // Iterate over each element in the array
        for (int i = 0; i < n; ++i) {
            // Calculate the sum of absolute differences for the current element
            int absolute difference sum = nums[i] * i - partial sum
                                          + total sum - partial_sum
                                          - \text{nums}[i] * (n - i);
            // Store the result in the corresponding position
            result[i] = absolute difference sum:
            // Update the partial sum with the value of the current element
            partial_sum += nums[i];
        // Return the final result vector
        return result;
};
TypeScript
function getSumAbsoluteDifferences(nums: number[]): number[] {
    // Calculate the sum of all elements in the array.
    const totalSum = nums.reduce((previousValue, currentValue) => previousValue + currentValue, 0);
    let cumulativeSum = 0;
```

```
return result; // Return the final array containing the sum of absolute differences for each element.
```

const length = nums.length;

for (let i = 0; i < length; ++i) {

cumulativeSum += nums[i];

from typing import List

answers = []

total_sum = sum(nums)

Initialize a prefix sum to 0.

class Solution:

const result = new Array(length); // Initialize the result array.

// Calculate the sum of absolute differences for nums[i]:

const absoluteDifferenceSum = nums[i] * i - cumulativeSum

def getSumAbsoluteDifferences(self, nums: List[int]) -> List[int]:

// Update the cumulative sum with the value of the current element.

Calculate the sum of the entire array to use later in calculations.

// This includes the difference between nums[i] and each element to the left of it,

result[i] = absoluteDifferenceSum; // Store the computed value in the result array.

+ totalSum - cumulativeSum - nums[i] * (length - i);

// Iterate through the elements of the array to calculate the

// sum of absolute differences for each element.

// and each element to the right of it.

Initialize a list to store the answers.

```
prefix_sum = 0
       # Iterate through the list, with both the index and the value.
        for index. value in enumerate(nums):
            # Compute the value for the current index.
           # 'value * index' is the sum of 'value' being subtracted from all
           # elements to the left of 'value' (including itself).
           # 'total sum - prefix_sum' is the sum of all the elements to the
           # right of 'value'.
           # Subtract 'value * (len(nums) - index)', which is 'value' being
            # added to all elements to the right of 'value'.
            current_value = value * index - prefix_sum + total_sum - prefix_sum - value * (len(nums) - index)
           # Append the computed value to the answer list.
           answers.append(current_value)
           # Update the prefix sum with the current value.
           prefix_sum += value
       # Return the list of computed answers.
        return answers
Time and Space Complexity
  The given code snippet implements an algorithm to get the sum of absolute differences for each number in the input list nums
  with every other number in the list.
```

To analyze the time complexity, let's consider the number of operations performed by the code: • We first calculate the sum of all numbers in nums (s = sum(nums)). This iteration over the array has a time complexity of O(n), where n is the

• The variables s, t, and v use constant space, i.e., 0(1).

by the algorithm is essentially the space taken by the ans list.

number of elements in nums. • Then, we iterate over the list nums once. For each element x at index i, we calculate the sum of absolute differences using the precomputed total sum s and the temporary sum t. The formula v = x * i - t + s - t - x * (len(nums) - i) does so using a constant number of operations for each element.

• Since all other variables (i and x) are also using constant space and there are no other data structures used, the total additional space used

• The total number of operations inside the loop is independent of the size of the list, meaning it's 0(1) per element. • As we do a constant amount of work for each of the n elements during this iteration, the time complexity for this loop is O(n).

O(n) space.

Time Complexity:

- Combining both steps, the overall time complexity of the algorithm is O(n) + O(n), which simplifies to O(n).
- **Space Complexity:**
- When analyzing the space complexity of the code, consider the following: • A new list ans is created to store the result. In the worst case, it will contain the same number of elements as the input list nums, hence it takes
- Therefore, the overall space complexity of the function is O(n). In summary, the function has a time complexity of O(n) and a space complexity of O(n).