2455. Average Value of Even Numbers That Are Divisible by Three

Easy Array Math

Leetcode Link

Problem Description

The problem provides an array of positive integers `nums` and requires us to calculate the average value of all even integers in the array that are also divisible by 3. To satisfy both conditions, an integer must be divisible by 2 and 3, which ultimately means it must be divisible by 6 (since 2 * 3 = 6). The average is calculated by taking the sum of these integers and dividing by their quantity, with the result being rounded down to the nearest integer.

Intuition

To solve this problem, we need to iterate over each integer in the `nums` array and check if it meets the criteria of being divisible by 6. This check is performed using the modulus operator `%` - if `x % 6 == 0`, then the integer `x` is divisible by 6. For every integer that complies, we add its value to a running sum `s` and increment a count `n`. Afterwards, we calculate the average by dividing `s` by `n` with integer division to ensure the result is rounded down. If no integers satisfy the condition (which means `n` is 0), we return 0 as there's nothing to average.

Solution Approach

The implementation of the solution follows a straightforward approach, utilizing a simple loop to iterate over the elements of the array `nums` and conditional statements to filter out the required integers. Here are the main components of the solution:

1. **Initialization**: Two variables are used to store the cumulative sum `s` of integers satisfying the condition and the count `n` of such integers.
   - `s = 0` initializes the sum to zero.
   - `n = 0` initializes the count to zero.
2. **Looping through the array**: A `for` loop iterates over each integer `x` in the array `nums`.
3. **Condition check**: Inside the loop, an `if` statement checks whether the current integer `x` is divisible by 6 using the modulus operation `x % 6 == 0`.
4. **Updating sum and count**: If an integer `x` satisfies the condition (is divisible by 6), its value is added to the sum `s` (`s += x`), and the count `n` is incremented by 1 (`n += 1`).
5. **Calculating the average**: After the loop, the average is calculated by dividing the sum `s` by the count `n` using integer division `s // n`, which automatically rounds down the result to the nearest integer.
6. **Edge case handling**: If no integer met the condition (`n == 0`), the function returns `0` to avoid division by zero.

The algorithm does not use additional data structures; it only requires a fixed amount of extra space for the sum and count variables, regardless of the input size, making it an `O(1)` space complexity solution. In terms of time complexity, since it requires a single pass through the array, it operates at `O(n)`, where `n` is the number of elements in `nums`.

No particular patterns or advanced algorithms are used; the solution leverages basic arithmetic operations, conditional logic, and a single loop—common constructs for most straightforward array processing tasks in algorithmic problems.

Example Walkthrough

Let's consider an example `nums` array: `[12, 18, 5, 6, 7]`.

1. **Initialization**: Begin by initializing the sum `s` and count `n` to 0. At this point, `s = 0`, `n = 0`.
2. **Looping through the array**:
   - First integer: `12`
   - Second integer: `18`
   - Third integer: `5`
   - Fourth integer: `6`
   - Fifth integer: `7`

   We will now iterate over each one and perform the necessary checks and operations.
3. **Condition check and updating sum and count**:
   - First integer `12`: it is divisible by 6, so we add it to the sum and increment the count. Now `s = 12` and `n = 1`.
   - Second integer `18`: this is also divisible by 6, hence it is added to the sum and count is incremented. Now `s = 30` (`12 + 18`) and `n = 2`.
   - Third integer `5`: this is not divisible by 6, so no changes are made. The values remain `s = 30` and `n = 2`.
   - Fourth integer `6`: it is divisible by 6, so we include it in our sum and increment the count. The updated values become `s = 36` (`30 + 6`) and `n = 3`.
   - Fifth integer `7`: it is not divisible by 6, so no changes are made. We end with `s = 36` and `n = 3`.
4. **Calculating the average**: With the final sum being 36 and the count being 3, we divide `s` by `n` using integer division to calculate the average: `average = s // n = 36 // 3 = 12`.
5. **Edge case handling**: There is no need for edge case handling in this example, as we have integers that meet the condition.

For this array `[12, 18, 5, 6, 7]`, the average value of all even integers that are also divisible by 3 is `12`.

Python Solution

```python
class Solution:
    def averageValue(self, nums: List[int]) -> int:
        """
        This method calculates the average of the numbers in the given list
        that are divisible by 6. It returns an integer value.

        :param nums: List[int] - A list of integers
        :return: The average value as an integer or 0 if no element is divisible by 6
        """

        # Initialize the sum and count of numbers divisible by 6
        total_sum = total_count = 0

        # Iterate over each number in the list
        for number in nums:
            # If the number is divisible by 6
            if number % 6 == 0:
                total_sum += number
                total_count += 1

        # Calculate average if at least one number is divisible by 6; otherwise, return 0
        return 0 if total_count == 0 else total_sum // total_count
```

Java Solution

```java
class Solution {
    public int averageValue(int[] nums) {
        int sum = 0; // Initialize sum to store the sum of numbers divisible by 6
        int count = 0; // Initialize count to store the number of numbers divisible by 6

        // Iterate over each number in the array
        for (int number : nums) {
            // Check if the number is divisible by 6
            if (number % 6 == 0) {
                sum += number; // Add the number to the sum
                ++count; // Increment the count of numbers divisible by 6
            }
        }
        // If count is zero, return 0 as we cannot divide by zero
        // Otherwise, return the average of the numbers divisible by 6
        return count == 0 ? 0 : sum / count;
    }
}
```

C++ Solution

```cpp
#include <vector> // Include the vector header for using std::vector

// Solution class that contains the method to calculate the average value.
class Solution {
public:
    // Method to calculate the average of numbers that are divisible by 6 in the vector.
    int averageValue(std::vector<int>& nums) {
        int sum = 0; // Variable to hold the sum of numbers divisible by 6.
        int count = 0; // Variable to count numbers divisible by 6.

        // Loop through all numbers in the input vector.
        for (int number : nums) {
            // Check if the number is divisible by 6.
            if (number % 6 == 0) {
                sum += number; // Add the number to the sum.
                ++count; // Increment the count of numbers divisible by 6.
            }
        }

        // If no number is divisible by 6, return 0 as the average.
        if (count ==0) {
            return 0;
        }

        // Calculate and return the average of numbers divisible by 6.
        return sum / count;
    }
};
```

Typescript Solution

```typescript
/**
 * Calculates the average of all numbers divisible by 6 in an array.
 * Returns 0 if there are no such numbers.
 *
 * @param {number[]} numbers - An array of numbers to calculate the average from.
 * @returns {number}
 */
function averageValue(numbers: number[]): number {
    // Initialize the sum of numbers divisible by 6
    let sum = 0;
    // Initialize the count of numbers divisible by 6
    let count = 0;

    // Iterate over all numbers in the input array
    for (const num of numbers) {
        // Check if number is divisible by 6
        if (num % 6 === 0) {
            // Add to the sum
            sum += num;
            // Increment the count
            ++count;
        }
    }

    // If there are no numbers divisible by 6, return 0
    if (count === 0) return 0;

    // Calculate the floor of the average value to return an integer result
    // `~~` is a double bitwise NOT operator which is a quicker substitute for `Math.floor`
    return ~~(sum / count);
}
```

Time and Space Complexity

The given Python code iterates through the list `nums` exactly once. During this iteration, it performs a constant-time operation to check if an element is divisible by 6, and if so, it adds its value to `s` and increments `n` by 1. Therefore, the time complexity is linear with respect to the number of elements in the list. In big-O notation, this is `O(n)` where `n` is the length of `nums`.

The space complexity is the amount of extra space or temporary storage that the algorithm uses beyond what is necessary for the input. In this case, the function maintains a fixed number of variables (`s` and `n`) regardless of the input size. As a result, the space complexity is constant, denoted as `O(1)`.