1390. Four Divisors

Medium Array Math

Problem Description

The given problem presents us with an integer array `nums`. Our task is to find the sum of all divisors of each integer in `nums` that exactly have four divisors. To clarify, for an integer to qualify, it must have exactly four distinct divisors - including '1' and itself. If an integer in the array doesn't meet this criteria, it's not included in the sum. If there's no such integer in the array meeting the criteria, then the function should return `0`.

Intuition

To approach this problem, consider that an integer with exactly four divisors can only qualify if it's either a product of two distinct prime numbers or a cube of a prime number (since the cube will have divisors 1, the prime, the square of the prime, and the cube of the prime). Given this premise, our strategy is as follows:

- Iterate through every element of the input array `nums`.
- For each element, try to find all its divisors.
- Keep a count of how many divisors we've found and simultaneously calculate their sum.
- If by the end, the count of divisors is exactly 4, add the sum of these divisors to our overall total.
- If it's not 4 divisors, disregard this integer and move to the next one.
- Repeat this process until we've checked all the integers in the array, and return the total sum.

The provided solution code defines a nested function `f(x)` that calculates the sum of divisors of an integer `x` if it has exactly four divisors. This function uses a while loop to find divisors of `x`. For each divisor `i` found, the count is incremented and the sum is updated accordingly. Once all potential divisors are checked, the function returns either the sum of divisors, if there are exactly four, or `0` otherwise.

This nested function is then applied to each element in `nums` with a generator expression that's passed to the `sum` function, rolling up the total sum of qualifying divisors. The concept of generators is efficient here since it avoids the need for an intermediate list to hold values for divisors' sums, thus saving memory.

Solution Approach

The solution implements a brute-force algorithm to find integers with exactly four divisors. Here's the step-by-step approach broken down:

1. A helper function `f(x)` is declared inside the `Solution` class, which takes an integer `x` as input and returns the sum of its divisors if it has exactly four divisors, otherwise returns `0`.
2. Inside the function `f(x)`, we initialize two variables: `cnt` to count the number of divisors (starting with 2, to account for '1' and `x` itself), and `s` to keep the sum of divisors (initialized to `x + 1` for the same reason).
3. A while loop is used to iterate over possible divisors `i`, starting from 2 up to the square root of `x`. We use the square root as an optimization because if `x` is divisible by a number greater than its square root, the corresponding divisor (`x // i`) will already have been counted.
4. For each `i` within the loop:
   - We check if `i` divides `x` entirely (`x % i == 0`). If it does, it means `i` is a divisor.
   - The count of divisors `cnt` is incremented, and `i` is added to the sum `s`.
   - If `i` is not a perfect square of `x` (to avoid counting the same divisor twice), we increment `cnt` again and add the corresponding divisor (`x // i`) to the sum `s`.
5. After the loop, we check if the count of divisors `cnt` is exactly 4. If this condition holds true, the function returns the sum `s`, otherwise, it returns `0`.
6. In the main function `sumFourDivisors`, we aggregate the results by using a generator expression `f(x) for x in nums` inside the `sum()` function. This expression calls function `f(x)` for every element `x` of the array `nums`.
7. The `sum()` function calculates the cumulative sum of returned values by `f(x)` (sum of divisors for qualified integers) and returns it.

To reiterate, data structures and patterns used in this approach include:

- A helper function to encapsulate the logic for evaluating individual numbers.
- Standard mathematical operations (modulo `%`, integer division `//`, and square root) for divisor evaluation.
- A generator expression to handle the accumulation of sums in a memory-efficient manner.
- Optimization by checking divisibility only up to the square root of the number to reduce redundant calculations.

Example Walkthrough

Let's consider an example array `nums = [8, 10, 20]` to illustrate the solution approach:

1. We start by applying the function `f(x)` to each element in `nums`.
2. For `x = 8`, the function `f(8)` tries to find divisors. We know that `8` is a cube of the prime number `2`, so its divisors are `1`, `2`, `4`, and `8`. Here the count of divisors is indeed 4. The sum of the divisors is `1 + 2 + 4 + 8 = 15`. Since it has exactly four divisors, the function `f(8)` will return `15`.
3. For `x = 10`, the divisors are `1`, `2`, `5`, and `10`. Again, we have exactly four divisors. The sum is `1 + 2 + 5 + 10 = 18`, and the function `f(10)` returns `18`.
4. For `x = 20`, the divisors are `1`, `2`, `4`, `5`, `10`, and `20`. There are more than four divisors, so the function `f(20)` does not meet the criteria and returns `0`.
5. Finally, the sum of all sums returned by `f(x)` for each element in the array is calculated. This is `15 + 18 + 0 = 33`.
6. Therefore, the `sumFourDivisors` function will return `33`, as it is the sum of all the divisors of integers within the array `nums` that have exactly four divisors.

By using this step-by-step approach for each number in any given array `nums`, we can efficiently find the sum of the divisors for numbers with exactly four divisors.

Solution Implementation

Python

```python
class Solution:
    def sumFourDivisors(self, nums: List[int]) -> int:
        # Define a helper function to find if a number has exactly four divisors
        def sum_if_four_divisors(num: int) -> int:
            divisor = 2
            count, sum_of_divisors = 2, num + 1  # start with 1 and 'num' as divisors
            # Iterate through potential divisors starting from 2 up to the square root of 'num'
            while divisor <= num // divisor:
                if num % divisor == 0:
                    count += 1
                    sum_of_divisors += divisor
                    # Check if the divisor and its counterpart are different
                    if divisor * divisor != num:
                        count += 1
                        sum_of_divisors += num // divisor
                divisor += 1
            # If exactly 4 divisors have been found, return their sum, otherwise return 0
            return sum_of_divisors if count == 4 else 0

        # Sum the results of the helper function for each number in the input list 'nums'
        return sum(sum_if_four_divisors(num) for num in nums)
```

Java

```java
class Solution {

    // Method to calculate the sum of all four divisors of the elements in the array.
    public int sumFourDivisors(int[] nums) {
        int sumTotal = 0; // Initialize the sum of the four divisors
        for (int number : nums) {
            sumTotal += sumOfFourDivisors(number); // Add the sum of the four divisors of the current number
        }
        return sumTotal; // Return the total sum
    }

    // Helper method to calculate the sum of four divisors of a single number.
    private int sumOfFourDivisors(int number) {
        int divisorCount = 2; // Start with 2 divisors: 1 and the number itself
        int divisorSum = 1 + number; // Sum of divisors starts with 1 and the number

        // Iterate to find other divisors
        for (int i = 2; i <= number / i; ++i) {
            // Check if 'i' is a divisor of 'number'
            if (number % i == 0) {
                divisorCount++; // Increase count of divisors
                divisorSum += i; // Add 'i' to the sum of divisors
                // Check if 'i' and 'number/i' are not the same divisor
                if (i * i != number) {
                    divisorCount++; // If not, we have another divisor
                    divisorSum += number / i; // Add 'number/i' to the sum of divisors
                }
            }
        }

        // Return the sum of divisors only if exactly four divisors are found
        return divisorCount == 4 ? divisorSum : 0;
    }
}
```

C++

```cpp
class Solution {
public:
    // Function to sum the divisors of each number in the given vector
    // that have exactly four distinct divisors.
    int sumFourDivisors(vector<int>& nums) {
        int totalSum = 0; // This will hold the sum of the divisors
        // Iterate through all numbers in the given vector
        for (int number : nums) {
            totalSum += sumDivisorsIfFour(number);
        }
        return totalSum;
    }

    // Helper function to calculate the sum of divisors of a number
    // if and only if it has exactly four distinct divisors.
    int sumDivisorsIfFour(int number) {
        int divisorCount = 2; // Start with 2 (1 and the number itself)
        int sumOfDivisors = number + 1; // Include 1 and the number in the sum

        // Iterate over possible divisors starting from 2 up to the square root of number
        for (int i = 2; i <= number / i; ++i) {
            if (number % i == 0) { // Check if i is a divisor
                // Increment the divisor count and add it to the sum
                ++divisorCount;
                sumOfDivisors += i;
                // Check if the divisor is not the square root of the number
                if (i * i != number) {
                    // This means the quotient is also a divisor
                    ++divisorCount;
                    sumOfDivisors += number / i;
                }
            }
        }

        // If the number has exactly four divisors, return the sum
        // Otherwise, return 0
        return divisorCount == 4 ? sumOfDivisors : 0;
    }
};
```

TypeScript

```typescript
/**
 * Function that returns the sum of four divisors for a single number
 * @param x The number to find the sum of its four divisors
 * @return The sum of the four divisors if there are exactly four, or 0 otherwise
 */
function sumOfDivisors(x: number): number {
    // Start with 2 divisors (1 and the number itself) and their sum
    let divisorCount = 2;
    let sumOfDivisors = x + 1;

    // Start from 2 to check for factors other than 1 and the number itself
    for (let i = 2; i * i <= x; ++i) {
        if (x % i === 0) {
            // If i is a divisor, increment count and add it to the sum
            ++divisorCount;
            sumOfDivisors += i;

            // If i is not a square root of x, account for the quotient as well
            if (i * i !== x) {
                ++divisorCount;
                sumOfDivisors += Math.floor(x / i);
            }
        }
    }

    // If there are exactly 4 divisors, return the sum, otherwise, return 0
    return divisorCount === 4 ? sumOfDivisors : 0;
}

/**
 * Function that returns the sum of all numbers in the input array that have exactly four divisors
 * @param nums Array of numbers
 * @return The sum of numbers with only four divisors from the input array
 */
function sumFourDivisors(nums: number[]): number {
    let totalSum = 0;

    for (const num of nums) {
        // For each number, calculate the sum of its four divisors if any
        totalSum += sumOfDivisors(num);
    }

    // Return the total sum for the array
    return totalSum;
}
```


```python
class Solution:
    def sumFourDivisors(self, nums: List[int]) -> int:
        # Define a helper function to find if a number has exactly four divisors
        def sum_if_four_divisors(num: int) -> int:
            divisor = 2
            count, sum_of_divisors = 2, num + 1  # start with 1 and 'num' as divisors
            # Iterate through potential divisors starting from 2 up to the square root of 'num'
            while divisor <= num // divisor:
                if num % divisor == 0:
                    count += 1
                    sum_of_divisors += divisor
                    # Check if the divisor and its counterpart are different
                    if divisor * divisor != num:
                        count += 1
                        sum_of_divisors += num // divisor
                divisor += 1
            # If exactly 4 divisors have been found, return their sum, otherwise return 0
            return sum_of_divisors if count == 4 else 0

        # Sum the results of the helper function for each number in the input list 'nums'
        return sum(sum_if_four_divisors(num) for num in nums)
```


Time and Space Complexity

The time complexity of the solution is determined by two parts: the iteration through the list of numbers `nums`, and the computation of the sum of four divisors for each number within the list.

The outer part of the code is a simple iteration through each number `x` in nums, which has a complexity of `O(n)`, where `n` is the size of `nums`.

The more complex part is the function `f(x)`, which calculates the sum of the divisors for each number. The while loop within this function iterates over potential divisors from `2` to `sqrt(x)`. In the worst-case scenario, this runs in `O(sqrt(x))` time, because the number of divisors up to the square root of `x` determines how many times the loop executes.

The total time complexity of this algorithm is a combination of the iteration through `nums` and the divisor function applied to each element. Therefore, the time complexity is `O(n * sqrt(x))`, where `x` is the value of the largest number in `nums`.

The space complexity of the provided code is `O(1)`. Other than a few variables for intermediate calculations, the space used does not depend on the input size. The `cnt`, `s`, `i`, and `x` variables are reused for each function call, and no extra space that scales with the input size is allocated.