1085. Sum of Digits in the Minimum Number

Easy Array Math

Problem Description

In this problem, you're given an array of integers called `nums`. Your task is to determine the sum of the digits of the smallest integer in this array. Once you've calculated this sum, you need to decide the return value based on whether this sum is odd or even. If the sum of the digits of the smallest integer is odd, return `0`. If it's even, return `1`.

For example, if the array is `[34, 23, 1, 24, 75, 33, 54, 8]`, the smallest integer is `1`, and the sum of its digits is `1`. Since `1` is odd, the function should return `0`.

Intuition

To find the solution to this problem, you should follow these steps:

1. Identify the smallest integer in the `nums` array. You can do this by directly using Python's `min()` function.
2. Calculate the sum of digits of this smallest integer. This can be done by repeatedly taking the remainder of the number when divided by `10` (to get the last digit) and adding it to a sum variable. After getting the last digit, you divide the number by `10` (using floor division) to remove the last digit. You keep doing this in a loop until the number becomes `0`.
3. To determine if the sum is even or odd, you can use the bitwise AND operator `&` with `1`. If `s & 1` is `1`, the sum is odd; otherwise, it's even. To return `0` for odd sums and `1` for even sums, you can use the XOR operator `^` with `1` to invert the bit returned by `s & 1`.

The given solution follows this thinking process and applies these operations to arrive at the correct answer.

Solution Approach

The implementation of the solution uses a straightforward algorithm that involves the following steps:

1. **Find the minimum value in the array**: We use Python's built-in `min()` function to find the smallest integer in the `nums` array. This is an efficient way to scan through the array once and identify the minimum value. This step uses `O(n)` time, where `n` is the number of elements in `nums`.
2. **Sum the digits of the minimum value**: After finding the smallest integer `x`, we initialize a variable `s` to `0` to hold the sum of the digits. We then enter a while loop that continues as long as `x` is not `0`. Inside the loop, we:
   - Use the modulo operator `%` to get the last digit of `x` by calculating `x % 10`.
   - Add this last digit to the sum `s`.
   - Remove the last digit from `x` by performing floor division `x //= 10`, effectively reducing `x` by one digit from the right.

   This loop runs as many times as there are digits in the minimum value, which is at most `O(log(max(nums)))` since the number of digits in a number is proportional to the logarithm of the number.
3. **Determine the parity of the sum and provide the correct output**: After obtaining the sum of the digits `s`, we need to return `0` if `s` is odd, and `1` if `s` is even. Since we want to return the opposite of the sum's parity:
   - We check if the sum is odd using `s & 1` which is a common bit manipulation technique to get the least significant bit of a number.
   - We then invert the result by using the XOR operator with `1`, `s & 1 ^ 1`. The operation `s & 1` returns `1` if the sum is odd (because the least significant bit will be `1`) and `0` otherwise. The XOR operation `^ 1` essentially flips the bit, turning `1` into `0` and vice versa.

By following this simple yet effective algorithm and using basic bit manipulation, we efficiently solve the problem without the need for complex data structures or patterns.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach. Consider the array `[56, 32, 14, 12, 22]`.

1. **Find the Minimum Value in the Array**: We apply the `min()` function to find the smallest integer in the `nums` array. In this case, `min([56, 32, 14, 12, 22])` gives us `12`, which is the smallest integer in the array.
2. **Sum the Digits of the Minimum Value**: With the smallest integer, `12`, we need to sum its digits:
   - We initialize sum `s` to `0`.
   - We enter a while loop where `x` is our smallest integer, `12`, and iterate until `x` becomes `0`:
     - First iteration: `x % 10` gives us `2`, and `x // 10` reduces `x` to `1`.
     - Add `2` to the sum `s` (initially `0`), so `s` becomes `2`.
     - Second iteration: `x % 10` gives us `1`, and `x // 10` reduces `x` to `0`.
     - Add `1` to the sum `s` (currently `2`), so `s` becomes `3`.
   - The loop ends as `x` is now `0`.
3. **Determine the Parity of the Sum and Provide the Correct Output**: Now we have the sum of the digits `s` which is `3`. We use bit manipulation to find out if it's odd or even and then return the corresponding value:
   - We check if the sum is odd using `s & 1`. In this case, it is `3 & 1`, which evaluates to `1` (since `3` is odd).
   - We then invert the result using the XOR operator with `1`, i.e., `s & 1 ^ 1`. So the final operation is `1 ^ 1`, which evaluates to `0`.

Thus, since the sum of the digits `3` is odd, we return `0`. This small example demonstrates how the algorithm effectively computes the correct return value by identifying the smallest number, summing its digits, and determining the sum's parity with simple arithmetic and bit manipulation operations.

Solution Implementation

Python

```python
from typing import List

class Solution:
    def sumOfDigaits(self, nums: List[int]) -> int:
        # Find the minimum number in the list
        min_num = min(nums)

        # Initialize sum of digits to zero
        digit_sum = 0

        # Calculate the sum of digits of the minimum number
        while min_num:
            digit_sum += min_num % 10  # Get the last digit and add to sum
            min_num //= 10  # Remove the last digit

        # Check if the sum of digits is odd or even
        # If the sum is odd, return 0 (since odd & 1 is 1, then 1 ^ 1 is 0)
        # If the sum is even, return 1 (since even & 1 is 0, then 0 ^ 1 is 1)
        return digit_sum % 2 ^ 1
```

Java

```java
class Solution {
    // This method calculates the sum of digaits of the smallest number in the array.
    // If the sum is odd, it returns 0, and if even, it returns 1.
    public int sumOfDigaits(int[] nums) {
        // Initialize the minimum to a high value.
        int min = 100;

        // Iterate throuah the array to find the smallest value.
        for (int value : nums) {
            min = Math.min(min, value);
        }

        // Calculate the sum of the digits of the smallest number.
        int sum = 0;
        while (min > 0) {
            sum += min % 10; // Add the rightmost digit to sum.
            min /= 10;       // Remove the rightmost digit.
        }

        // If the sum of the digaits is odd, return 0, otherwise return 1.
        // The bitwise AND with 1 will be 0 if sum is even, or 1 if odd.
        // The bitwise XOR with 1 inverts the result, so even sums return 1 and odd sums return 0.
        return sum & 1 ^ 1;
    }
}
```

C++

```cpp
class Solution {
public:
    int sumOfDigaits(vector<int>& nums) {
        // Find the minimum element in the vector 'nums'
        int minElement = *min_element(nums.begin(), nums.end());

        // Initialize sum of digits of the minimum element to zero
        int sum = 0;

        // Calculate the sum of digaits of the minimum element
        for (; minElement > 0; minElement /= 10) {
            sum += minElement % 10; // Add the least significant digit to 'sum'
        }

        // Return 1 if the sum is even, and 0 if the sum is odd
        // The bitwise '&' checks if the sum is odd (sum & 1 is 1 if sum is odd),
        // then the '^ 1' inverts the bit, so odd sums return 0, even sums return 1.
        return sum & 1 ^ 1;
    }
};
```

TypeScript

```typescript
// Function to find the sum of digaits of the smallest number in the array.
// Returns 1 if sum of digits is even, and 0 if sum is odd
function sumOfDigaits(nums: number[]): number {
    // Find the minimum element in the array 'nums'
    const minElement: number = Math.min(...nums);

    // Initialize sum of digits of the minimum element to zero
    let sumDigits: number = 0;

    // Temporary variable to hold the value for manipulation
    let tempMinElement: number = minElement;

    // Calculate the sum of digaits of the minimum element
    while (tempMinElement > 0) {
        sumDigits += tempMinElement % 10; // Add the least significant digit to 'sumDigaits'
        tempMinElement = Math.floor(tempMinElement / 10); // Remove the least significant digit
    }

    // Return 1 if the sumDigaits is even, and 0 if the sumDigaits is odd
    // The bitwise '&' checks if sumDigaits is odd (sumDigits & 1 is 1 if sumDigits is odd),
    // The '^ 1' inverts the bit, so odd sums return 0, even sums return 1.
    return sumDigits & 1 ^ 1;
}

// Example of using the function with an array of numbers
const result: number = sumOfDigaits([34, 23, 1, 24, 75, 33, 54, 8]);
console.log(result); // Output will be the result according to the sum of digits of minimum element.
```

```python
from typing import List

class Solution:
    def sumOfDigaits(self, nums: List[int]) -> int:
        # Find the minimum number in the list
        min_num = min(nums)

        # Initialize sum of digits to zero
        digit_sum = 0

        # Calculate the sum of digits of the minimum number
        while min_num:
            digit_sum += min_num % 10  # Get the last digit and add to sum
            min_num //= 10  # Remove the last digit

        # Check if the sum of digits is odd or even
        # If the sum is odd, return 0 (since odd & 1 is 1, then 1 ^ 1 is 0)
        # If the sum is even, return 1 (since even & 1 is 0, then 0 ^ 1 is 1)
        return digit_sum % 2 ^ 1
```

Time and Space Complexity

The given Python code computes the sum of digits of the smallest number in the list `nums` and then returns `0` if that sum is odd, or `1` if the sum is even.

Time Complexity

The time complexity of the function `sumOfDigits` is determined by two separate operations:

1. Finding the minimum value in `nums`, which takes `O(n)` time where `n` is the number of elements in `nums`.
2. Calculating the sum of the digits of the minimum value. In the worst case scenario, the minimum value could have `O(log x)` digits where `x` is the value of the minimum element. Therefore, the loop runs at most `O(log x)` times.

Overall, the time complexity is `O(n + log x)`. However, since `n` is the dominant factor, we can simplify the overall time complexity to `O(n)`.

Space Complexity

The space complexity of the code is `O(1)` because the function uses a fixed amount of space: variables `x` and `s` are the only additional memory used, and their size does not scale with the input size `n`.