1979. Find Greatest Common Divisor of Array

Easy Array Math Number Theory

Leetcode Link

Problem Description

The LeetCode problem at hand requires finding the greatest common divisor (GCD) of the smallest and largest numbers in a given array of integers, `nums`. The GCD is the highest positive integer that can divide both numbers without leaving a remainder.

To break down the steps we need to follow:

1. Identify and extract the smallest and the largest numbers from the array.
2. Calculate the GCD of these two numbers.

Intuition

To solve this problem efficiently, our solution employs the built-in `gcd` function from Python's math library and two functions `max()` and `min()`.

Here's the step-by-step thought process:

1. Utilize the `max()` function to find the largest number in the array `nums`.
2. Use the `min()` function to find the smallest number in the array `nums`.
3. Apply the `gcd` function, which computes the greatest common divisor of two numbers.

By using these built-in functions, we abstract away the complexity of writing our own algorithms for finding maximum and minimum values and computing the GCD. This allows our solution to be concise and maintain good performance.

Solution Approach

In this problem, the implementation of the solution is fairly straightforward due to Python's powerful built-in functions. Here is how the algorithm and data structures involved work:

1. **Finding the Largest and Smallest Values**: The `max()` function iterates through all the elements in the `nums` list to return the largest value, while the `min()` function does the same to return the smallest value. Under the hood, each of these functions performs a linear scan through the list, which is a simple but effective algorithm.

2. **Computing the Greatest Common Divisor**: The `gcd` function implemented in Python's math library uses the Euclidean algorithm. This age-old algorithm is an efficient way to compute the greatest common divisor of two numbers a and b (where a ≥ b). The process is as follows:

```
while b ≠ 0:
    temp = b
    b = a % b  # '%' is the modulo operator
    a = temp
# At the end of the loop, 'a' holds the GCD of the original a and b
```

In terms of data structures, we only deal with the array structure (Python list) containing the input numbers. There is no need for additional data structures as Python's built-in functions handle the necessary operations internally.

The reference solution code effectively leverages these functions, resulting in a clean and efficient solution:

```
class Solution:
    def findGCD(self, nums: List[int]) -> int:
        return gcd(max(nums), min(nums))
```

Here the `max(nums)` call finds the largest number in `nums`, `min(nums)` finds the smallest, and `gcd()` calculates their greatest common divisor. The key advantage of this approach is its simplicity and the fact that it relies on well-optimized library functions that likely outperform any custom implementation for the same task.

Example Walkthrough

Let us take an array `nums` with the following integers: `[24, 36, 12, 18, 48]`.

1. **Finding the Largest and Smallest Values**:
   - Using the `max()` function on our array, we get the largest value `48`. This is done by comparing each element and keeping track of the highest one encountered.
   - Similarly, applying `min()` function, the smallest value `12` is found, through a similar process of comparison and keeping track of the lowest one.
2. **Computing the Greatest Common Divisor**:
   - Now that we have the smallest number, `12`, and the largest number, `48`, we use the `gcd` function from Python's math library to compute the greatest common divisor of these two numbers.
   - The `gcd` function performs the Euclidean algorithm behind the scenes. For our numbers `48` (a) and `12` (b), the algorithm works as follows:

     ```
     while b ≠ 0:
         temp = b
         b = a % b  # b becomes 48 % 12, which is 0
         a = temp   # a becomes 12
     ```

     As soon as `b` becomes `0`, the loop ends and `a`, which is `12` in our case, is the GCD of `48` and `12`.
3. **Result**: The `gcd` of `48` and `12` returns `12` as the GCD, which is the highest positive integer that divides both `48` and `12` without any remainder.

Following these steps with `max(nums)`, `min(nums)`, and `gcd()`, we find that the GCD of the smallest and largest numbers in the given array `[24, 36, 12, 18, 48]` is `12`.

Python Solution

```
from typing import List
from math import gcd

class Solution:
    def findGCD(self, nums: List[int]) -> int:
        """
        Computes the greatest common divisor (GCD) of the maximum and
        minimum numbers in the given list.

        :param nums: List of integers
        :return: The GCD of the max and min values in the list
        """
        # Find the maximum value in the list
        max_num = max(nums)
        # Find the minimum value in the list
        min_num = min(nums)

        # Compute the GCD of the max and min values
        return gcd(max_num, min_num)

```

Java Solution

```
class Solution {

    // Method to find the Greatest Common Divisor (GCD) of the largest and smallest numbers in the array.
    public int findGCD(int[] nums) {
        // Initialize maxNum to the smallest possible integer value and minNum to the largest possible integer value
        int maxNum = Integer.MIN_VALUE;
        int minNum = Integer.MAX_VALUE;

        // Iterate through all numbers in the array
        for (int num : nums) {
            // Update the maxNum with the maximum value found so far
            maxNum = Math.max(maxNum, num);
            // Update the minNum with the minimum value found so far
            minNum = Math.min(minNum, num);
        }

        // Return the GCD of the largest and smallest numbers found in the array
        return gcd(maxNum, minNum);
    }

    // Helper method to calculate GCD of two numbers using the Euclidean algorithm
    private int gcd(int a, int b) {
        // If b is 0, we have found the GCD and return a
        if (b == 0) {
            return a;
        }
        // Recursively call the gcd method with b and the remainder of a divided by b
        return gcd(b, a % b);
    }
}

```

C++ Solution

```
#include <vector>
#include <algorithm> // Include necessary headers

class Solution {
public:
    // Function to find the greatest common divisor (GCD) of the max and min
    // element in a vector of integers.
    int findGCD(std::vector<int>& nums) {
        // Find the maximum element in the vector
        int maxElement = *std::max_element(nums.begin(), nums.end());
        // Find the minimum element in the vector
        int minElement = *std::min_element(nums.begin(), nums.end());

        // Return the GCD of the max and min elements
        return gcd(maxElement, minElement);
    }

private:
    // Helper function to compute the GCD of two numbers using Euclid's algorithm
    int gcd(int a, int b) {
        // Continue until no remainder is left
        while (b != 0) {
            int temp = b;
            b = a % b;  // Replace b with the remainder of a divided by b
            a = temp;   // Replace a with b
        }
        return a; // When b is 0, a contains the GCD
    }
};

```

Typescript Solution

```
/**
 * Finds the Greatest Common Divisor (GCD) of two numbers,
 * utilizing the Euclidean algorithm.
 * @param a The first number.
 * @param b The second number.
 * @return The GCD of a and b.
 */
function gcd(a: number, b: number): number {
    // Base case: if b is 0, a is the GCD
    if (b === 0) {
        return a;
    }
    // Recursive case: call gcd with b and the remainder of a divided by b
    return gcd(b, a % b);
}

/**
 * Finds the GCD of the smallest and largest numbers in the provided array.
 * @param nums The array of non-negative integers.
 * @return The GCD of the smallest and largest integers in nums.
 */
function findGCD(nums: number[]): number {
    // Initialize variables to store the smallest and largest numbers
    // Start with opposite extremes for comparison
    // 'maxValue' will hold the largest number in the array
    // 'minValue' will hold the smallest number in the array
    let maxValue: number = 1;
    let minValue: number = 1000;

    // Iterate through all numbers in the array to find the smallest
    // and largest numbers
    for (const num of nums) {
        // Update the largest number (maxValue) found so far
        maxValue = Math.max(maxValue, num);
        // Update the smallest number (minValue) found so far
        minValue = Math.min(minValue, num);
    }

    // Return the GCD of the largest and smallest number in the array
    return gcd(maxValue, minValue);
}

```

Time and Space Complexity

The time complexity of the provided code is `O(N)` where `N` is the number of elements in the `nums` list. This is because the `max(nums)` and `min(nums)` functions each require a full pass through the list to find the respective maximum and minimum values, and each pass is `O(N)`. Both operations are sequential and do not depend on each other's results, therefore the time complexity does not compound.

The space complexity of the code is `O(1)` since it uses a fixed amount of extra space. The `gcd` calculation and the retrieval of the max and min values are done in-place without allocating additional space proportional to the input size.