GCD of Two Numbers

The code aims to find the Greatest Common Divisor (GCD) of two given numbers, 'a' and 'b', using two different approaches. The first approach is the Brute Force Approach, which iterates through potential divisors to find the GCD. The second approach is the Euclidean Algorithm, a more efficient method to calculate the GCD.

Approach 1: Function to find GCD using the Brute Force Approach

- This function takes two integers, 'a' and 'b', as input.
- It initializes a variable 'gcd' to the minimum of 'a' and 'b'.
- The function then uses a while loop to iteratively decrement 'gcd' until it finds the largest common divisor of 'a' and 'b'.
- Inside the loop, it checks if 'gcd' is a common divisor for both 'a' and 'b' by using the modulo operator.
- When a common divisor is found, the loop breaks, and the GCD is returned.
- **Time Complexity: The time complexity of this approach is O(min(a, b)). In the worst case, it will take 'min(a, b)' iterations to find the GCD.**
- **Space Complexity: space complexity of O(1) since the approach uses only a constant amount of additional memory, regardless of the input size.**

Approach 2: Function to find GCD using the Euclidean Algorithm

- This function takes two integers, 'a' and 'b', as input.
- It uses recursion to find the GCD.
- The base case is when 'b' becomes 0, at which point 'a' is the GCD, and it is returned.
- Otherwise, the function makes a recursive call, passing 'b' and the remainder of 'a' divided by 'b' as arguments.
- The function continues to recurse until the base case is reached.
- **Time Complexity: The time complexity of the Euclidean Algorithm for finding the GCD is O(log(min(a, b))). It performs significantly better than the Brute Force Approach.**
- **Space Complexity: space complexity of O(1) since the approach uses only a constant amount of additional memory, regardless of the input size.**