Time Complexity of Euclid's Algorithm

Last modified: September 30, 2020

by Emily Marshall

1. Overview

In this short tutorial, we'll look at two common interpretations of Euclid's algorithm and analyze their time complexity.

2. Greatest Common Divisor

Euclid's algorithm is a method for calculating the greatest common divisor of two integers.

Let's start by recalling that the greatest common divisor of two integers is the largest number which divides both numbers with a remainder of zero. We'll use $gcd(a, b)$ to denote the greatest common divisor of integers a and b. So, for example:

- $gcd(10, 5) = 5$
- $gcd(96, 72) = 12$
- $gcd(23, 89) = 1$
- $gcd(x, 1) = 1$, where x is a positive integer

This task might seem trivial for such small numbers, however, it becomes increasingly difficult as our numbers grow. So, if you need some convincing, try calculating $gcd(31487, 21933)$!

Euclid's algorithm can make this process a whole lot easier. Whilst this algorithm has a number of variations, each is ultimately derived from the propositions put forth by Euclid of Alexander in his works *Elements*.

Today we'll explore two common ones.

3. Euclid's Algorithm by Subtraction

The original version of Euclid's algorithm, presented in Proposition 2 in Euclid's Elements, employs subtraction.

For some positive integers a and b, it works by repeatedly subtracting the smaller number from the larger one until they become equal. At this point, the value of either term is the greatest common divisor of our inputs.

3.1. The Algorithm

We can define this algorithm in just a few steps:

Step 1: If $a = b$, then return the value of a
Step 2: Otherwise, if $a > b$ then let $a = a - b$ and return to Step 1
Step 3: Otherwise, if $a < b$, then let $b = b - a$ and return to Step 1

Now, let's step through this algorithm for the example $gcd(25, 10)$:

$a = 25, b = 10$
$a = 25 - 10 = 15, b = 10$
$a = 15 - 10 = 5, b = 10$
$a = 5, b = 10 - 5 = 5$

We have reached $a = b = 5$, which means that $gcd(25, 10) = 5$.

3.2. Pseudocode

We can also present this algorithm in pseudocode:

Algorithm 1: gcd(a, b)

Input: Two integers a and b
Result: The greatest common divisor of a and b
if $a = b$ **then**
  return a
else if $a > b$ **then**
  return $gcd(a - b, b)$
else
  return $gcd(a, b - a)$
end

3.3. Time Complexity

We can estimate the worst-case time complexity of our algorithm by thinking about the sum of $a + b$. If we exclude our base case, this sum will decrease at each recursive step. **Since the smallest possible deduction for each step is 1, and since all positive integers are guaranteed to have a common divisor of 1, our time complexity will have an upper bound of $O(a + b)$.**

4. Euclid's Algorithm by Division

A modern adaption of Euclid's algorithm uses division to calculate the greatest common factor of two integers a and b, where $a < b \leq 0$. It is based upon a few key observations:

- $gcd(x, 0)$ is x, for any positive integer x
- $gcd(a, b) = gcd(b, a \mod b)$

This first observation is quite intuitive, however, the second is less obvious – if you want to examine its proof check out these slides.

With these observations in mind, our algorithm simply repeats the following assignment:

$$gcd(a, b) = gcd(b, a \mod b)$$

until b becomes zero. At this point, we can simply pluck our result from the final value of a.

4.1. The Algorithm

Euclid's algorithm by division has three steps:

Step 1: If $b = 0$, then return the value of a
Step 2: Otherwise, divide a by b and store the remainder in some variable r
Step 3: Let $b = r$, and $a = b$, and return to Step 1

Let's step through the algorithm for the inputs $a = 25$ and $b = 10$:

$a = 25, b = 10$
$a = 25, b = 10, r = 25 \mod 10 = 5$
$a = 10, b = 5$
$a = 10, b = 5, r = 10 \mod 5 = 0$
$a = 5, b = 0$

Now that we have reached $b = 0$, we know that $gcd(25, 10) = a = 5$.

4.2. Pseudocode

We can implement this algorithm as a recursive function $gcd(a, b)$:

Algorithm 2: gcd(a, b)

Input: Two integers a and b
Result: The greatest common divisor of a and b
if $b = 0$ **then**
  return a
else
  $r \leftarrow a \mod b$
  return $gcd(b, r)$
end

Alternatively, we can use a while loop to capture the same behavior:

Algorithm 3: gcd(a, b)

Input: Two integers a and b

Input: Two integers a and b
Result: The greatest common divisor of a and b
while $b \neq 0$ **do**
    $r \leftarrow a \bmod b$
    $a \leftarrow b$
    $b \leftarrow r$
end
return a

4.3. Time Complexity

It turns out that the number of steps our algorithm will take is maximized when the two inputs are consecutive Fibonacci numbers.

More specifically, if $gcd(a, b)$ requires n steps, and $a < b \leq 0$, then the smallest possible values of a and b are F_{n+2} and F_{n+1}, respectively. This can be proven by induction.

So, when n steps are required:

$$b \geq F_{n+1}$$

Next, recall that the $F_i = \frac{\phi^i}{\sqrt{5}}$, where ϕ is the 'golden ratio' and equals $\frac{1+\sqrt{5}}{2}$. We can then deduce that:

$$b \geq \phi^{n-1}$$

Finally, we can solve for n:

$$n - 1 \leq \log_\phi b$$

$$n \leq \log_\phi b + 1$$

So, the number of steps will always be less than $O(\log b)$, where b is the smaller of our two input values. Therefore, $O(\log b)$ is an upper bound on the worst-case cost of our algorithm.

It's worth noting that b doesn't need to be fed as the second argument to our algorithm. That is, we can execute $gcd(u, v)$ where $u < v$. In this case, our algorithm just performs one additional step at the beginning that effectively swaps the values. Think about why this might be the case ... what is the result of $u \mod v$ when $v > u \geq 0$?. Since this additional step adds constant time, so our worst-case bound is unaffected.

5. Conclusion

In this article, we implemented two variations of Euclid's Algorithm to find the greatest common divisor of two positive integers. We then analyzed the complexity of each solution.