PORTER'S CONSTANT IN EUCLIDEAN ALGORITHM COMPLEXITY

TRISHA SABADRA

Abstract. The Euclidean algorithm, one of the oldest and most efficient algorithms for finding the greatest common divisor (GCD) of two integers, has numerous applications in modern computational problems, particularly in the realm of integer factorization. This paper delves into the intricacies of the Euclidean algorithm, exploring its worst-case complexity characterized by consecutive Fibonacci numbers and its average-case complexity, highlighting the significance of Porter's constant.

1. Introduction to the Euclidean Algorithm

The Euclidean algorithm, a method for finding the greatest common divisor (GCD) of two integers, has stood the test of time as one of the oldest and most efficient algorithms still in use today. Named after the ancient Greek mathematician Euclid, who first described it around 300 BC, this algorithm has been foundational in the field of number theory and has numerous applications in modern computational problems.

In essence, the Euclidean algorithm capitalizes on a simple yet powerful principle: the GCD of two numbers does not change if the larger number is replaced by its difference with the smaller number. This principle allows the algorithm to iteratively reduce the size of the numbers involved, simplifying the problem until it becomes trivially easy to solve.

Example. Consider the numbers 252 and 105. We know $\gcd(252, 105) = 21$, as $252 = 21 \times 12$ and $105 = 21 \times 5$. According to the Euclidean algorithm,

$$\gcd(252, 105) = \gcd(252 - 105, 105) = \gcd(147, 105) = 21.$$

The GCD remains 21 because $147 = 21 \times 7$ and $105 = 21 \times 5$.

Theorem 1. *Let n and m be two integers with $n \geq m$. Then, $gcd(n, m) = gcd(n - m, m)$.*

Proof. Let $d = \gcd(n, m)$. By definition, d is the greatest positive integer that divides both n and m. Thus, $d \mid n$ and $d \mid m$ and for all common divisors $x|m, n$ then $x|d$. This means there exist integers a and b such that $n = ad$ and $m = bd$, so for the difference $n - m$:

$$n - m = ad - bd = d(a - b).$$

Therefore, $d \mid (n - m)$. Let $d_1 = \gcd(n - m, m)$. Since $d \mid (n - m)$ and $d \mid m$, by definition of the GCD, $d \mid d_1$. Since $d_1 \mid (n - m)$ and $d_1 \mid m$, it follows that $d_1 \mid (n - m + m) = n$. Thus, $d_1 \mid n$ and $d_1 \mid m$, so by definition of GCD, $d_1 \mid d$. Since $d_1 \mid d$ and $d \mid d_1$, we have $d = d_1$. Therefore,

$$\gcd(n, m) = \gcd(n - m, m).$$

□

By repeatedly applying this process, we obtain smaller and smaller pairs of numbers until the two numbers become equal. When that occurs, they represent the GCD of the original two numbers. While this method is straightforward, it can be inefficient and take many subtraction steps when one number is much larger than the other. The remainder method is a more efficient version of the algorithm as it shortcuts these steps, instead replacing the larger of the two numbers by its remainder when divided by the smaller of the two. This algorithm can be formalized in the following pseudocode:

```
function gcd(a, b)
    while b != 0
        temp := b
        b := a mod b
        a := temp
    return a
```

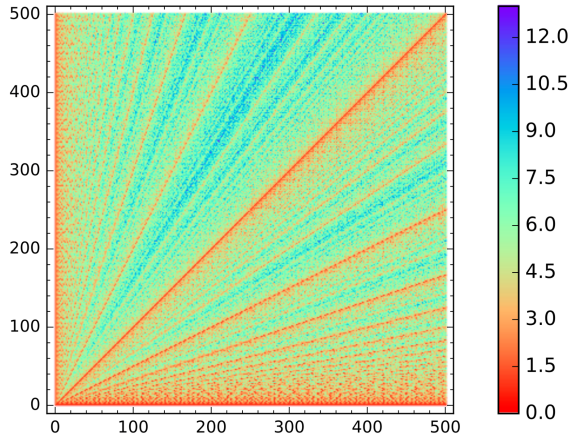

FIGURE 1. Number of steps in the Euclidean algorithm for gcd(x,y). Lighter (red and yellow) points indicate relatively few steps, whereas darker (violet and blue) points indicate more steps. The largest dark area follows the line $y = x\varphi$, where φ is the golden ratio.

Example. Let's compute the GCD of 252 and 105 using the above code.

(1) Calculate 252 mod 105:

$$252 = 2 \times 105 + 42 \quad \text{(remainder 42)}$$

Replace 252 with 42. Now we have the pair (105, 42).

(2) Calculate 105 mod 42:

$$105 = 2 \times 42 + 21 \quad \text{(remainder 21)}$$

Replace 105 with 21. Now we have the pair (42, 21).

(3) Calculate 42 mod 21:

$$42 = 2 \times 21 + 0 \quad \text{(remainder 0)}$$

The algorithm terminates here as the remainder is 0. The last non-zero remainder is 21, so we found $\gcd(252, 105) = 21$ in just three steps.

2. Worst Case Complexity

The computational efficiency of Euclid's algorithm has been thoroughly studied, primarily focusing on the number of division steps required and the computational expense of each step. The first known analysis was conducted by A. A. L. Reynaud in 1811, who initially showed that the number of division steps is bounded by the smaller number, v, and later refined this to $v/2 + 2$. In 1841, P. J. E. Finck demonstrated that the number of division steps is at most $2\log_2 v + 1$, indicating that Euclid's algorithm runs in polynomial time relative to the number of bits, or digits in the smaller number. Émile Léger studied the worst-case scenario in 1837, identifying it as the case when inputs are consecutive Fibonacci numbers [Sha94].

Definition 1. *The **Fibonacci Sequence** is defined by $F_0 = 0, F_1 = 1$, and the recursive function for $n > 1$:*

$$F_n = F_{n-1} + F_{n-2}$$

Theorem 2. *For $n \geq 1$, let u and v be integers with $u > v > 0$ such that Euclid's algorithm applied to u and v requires exactly n division steps, and such that u is as small as possible satisfying these conditions. Then $u = F_{n+2}$ and $v = F_{n+1}$, where F_n denotes the n-th Fibonacci number.*

Proof. The worst case is characterized by the slowest reduction in the size of the numbers, which occurs when the quotients in each division step are minimal, i.e., 1. For consecutive Fibonacci numbers, each quotient is 1, leading to the maximum number of steps before the algorithm terminates.

Consider the Euclidean algorithm applied to $u = F_{n+2}$ and $v = F_{n+1}$. Each step reduces the pair to the next pair of consecutive Fibonacci numbers:

$$(F_{n+2}, F_{n+1}) \to (F_{n+1}, F_n) \to (F_n, F_{n-1}) \to \cdots \to (F_3, F_2) \to (F_2, F_1).$$

This sequence shows that n division steps are required.

To generalize this, we use a polynomial representation. Let $u = K_n(A_1, A_2, \ldots, A_n)d$, where K_n is a polynomial, $A_1, A_2, \ldots, A_n$ are positive integers, and $A_n \geq 2$. The minimum value for u is achieved when:

$$A_1 = 1, \ldots, A_{n-1} = 1, A_n = 2, d = 1.$$

This configuration corresponds to the Fibonacci sequence. Thus, for $n \geq 1$, the worst-case scenario for Euclid's algorithm is when $u = F_{n+2}$ and $v = F_{n+1}$, requiring exactly n division steps.

Therefore, we have shown that the sequence $u = F_{n+2}$ and $v = F_{n+1}$ will require exactly n division steps in Euclid's algorithm, confirming the theorem [Knu81]. □

Gabriel Lamé further refined the analysis in 1844, showing that the number of steps required is never more than five times the number of base-10 digits of the smaller number. Known as Lamé's theorem, this represents the beginning of computational complexity theory and also the first practical application of the Fibonacci numbers [Sha94].

Theorem 3 (Lamé's Theorem)**.** *The Euclidean Algorithm never requires more than $5k$ steps, where k is the number of digits (base 10) of the smaller integer.*

Proof. Let u and v be two positive integers. Applying the Euclidean algorithm, we generate two sequences: $(q_1, \ldots, q_n)$ and $(v_2, \ldots, v_n)$ where q_i represents the

quotients and v_i represents the remainders. More formally, $v_0 = u$, $v_1 = v$, and $v_{n+1} = 0$, such that:

$$v_{i-1} = q_i v_i + v_{i+1}$$

for $i = 1, 2, \ldots, n$, where n is the *number of steps* in the Euclidean algorithm.

First, we will prove $v \geq F_{n+1}$ using induction on n. For the base case, we know when $n = 1$: $v_1 \geq F_2 = 1$ and when $n = 2$: $v_2 \geq F_3 = 2$. Assume that the claim holds when $n = k$, so we want to prove the claim when $n = k - 1$. By induction,

$$\begin{aligned} v_{k-1} &= q_k v_k + v_{k+1} \\ &\geq v_k + v_{k+1} \\ &\geq F_{k+2} + F_{k+1} = F_{k+3} \end{aligned}$$

by the definition of the Fibonacci sequence.

Next, we want to prove the Fibonacci numbers grow exponentially according to the golden ratio φ :

$$\varphi = \frac{1 + \sqrt{5}}{2}.$$

Specifically, we will prove $F_k \geq \varphi^{k-2}$ for all integers $k \geq 2$ by induction. For the base case, $F_2 = 1 = \varphi^0$ and $F_3 = 2 > \varphi$. Now for $k + 1$:

$$\begin{aligned} F_{k+1} &= F_k + F_{k-1} \\ &\geq \varphi^{k-2} + \varphi^{k-3} \\ &= \varphi^{k-3}(\varphi + 1) \\ &= \varphi^{k-1} \end{aligned}$$

using $\varphi^2 = \varphi + 1$. Thus, we proved that $F_{n+1} \geq \varphi^{n-1}$. Now combining this result with $v \geq F_{n+1}$, we get

$$\begin{aligned} v &\geq \varphi^{n-1} \\ \log_{10}(v) &\geq (n-1)\log_{10}(\varphi) \\ n - 1 &\leq \frac{\log_{10}(v)}{\log_{10}(\varphi)} \end{aligned}$$

Using the approximation:

$$\log_{10}(\varphi) \approx 0.20899$$

$$\frac{1}{\log_{10}(\varphi)} \approx 4.79 < 5$$

Therefore,

$$n - 1 \leq \frac{\log_{10}(v)}{\log_{10}(\varphi)} < 5\log_{10}(v)$$

If k is the number of decimal digits of v, one has $v < 10^k$ and $\log_{10}(v) < k$. So,

$$n - 1 < 5k \rightarrow n \leq 5k$$

which completes the proof. □

Thus, Lamé's analysis implies that the running time is $O(h)$, where h is the number of digits in the smaller number. While this is a good upper bound, it only represents the the worst-case scenario: when the input numbers are consecutive Fibonacci numbers. These cases occur rarely, and don't reflect the typical behavior of the Euclidean algorithm. Thus, understanding the average-case complexity provides insights into the typical efficiency of the algorithm in practice.

3. Average-Case Complexity

The average number of steps $\tau(n)$ taken by the Euclidean algorithm for a fixed n and averaged over all integers m that are coprime to n is:

$$\tau(n) = \frac{1}{\varphi(n)} \sum_{\substack{0 \le m < n \\ \gcd(m,n)=1}} T(n,m),$$

where $T(n,m)$ is the number of steps to compute $\gcd(n,m)$. In 1969, Hans Heilbronn proved the following theorem [Hei34].

Theorem 4. *The average number of iterations for the Euclidean algorithm for fixed n is*

$$\tau_n = \frac{12\ln 2}{\pi^2}\ln n + O((\log\log n)^4).$$

The error estimate was improved to $O(\log\log n^3)$ by Tonkov [Ton71]. Then, Porter showed that the error term in this estimate is a constant, plus a polynomially-small correction, proving the sharper form [Por75]:

$$\tau_n = \frac{12\ln 2}{\pi^2}\ln n + C + O(n^{-1/6+\epsilon}).$$

for all $\epsilon > 0$. The constant C in this term is known as **Porter's Constant**, defined by

$$C = \frac{24}{\pi^2}\left(B + \frac{3}{16} + \left(\gamma + \frac{5}{16}\right)\ln 2 - \frac{7}{8}(\ln 2)^2 + \frac{2}{\pi^2}\zeta'(2)\ln 2\right) + 2.5,$$

$$B = \sum_{k=1}^{\infty} \frac{1}{k}(H_{2k-1} - H_k - \ln 2),$$

$$I = \int_0^{1/\sqrt{8}} \frac{t^2 \ln t}{(t^2+1)^{1/2}}\,dt,$$

and H_k denotes the harmonic series sum:

$$H_k = \sum_{m=1}^{k} \frac{1}{m} \approx \ln k + \gamma,$$

where γ is the Euler-Mascheroni constant. In 1976, Donald Knuth evaluated C to high accuracy [Knu76].

The integral I is readily evaluated by expanding into power series,

$$I = \int_0^{1/\sqrt{8}} \sum_{k=0}^{\infty} \left(-\frac{1}{2k}\right) t^{2-2k} \ln t\,dt = \sum_{k=0}^{\infty} (-1)^{k+1} \left(\frac{2}{2k}\right) 2^{-5k-4.5}(2k+3)^{-1}(1+(3k+4.5)\ln 2).$$

To evaluate B, let $B' = B + \frac{1}{2}\zeta(2)$, so that $B' = \sum_{k=1}^{\infty} A_k/k$, where

$$A_n = H_{2n} - H_n - \ln 2 = -\frac{1}{4n} + \sum_{k=1}^{r} \frac{B_{2k}(1-2^{-2k})}{n^k} + O(n^{-2r}),$$

by Euler's summation formula. Letting $\zeta_m(s) = \sum_{k\le m} k^{-s} = O(m^{1-s})$, we have

$$B' = \sum_{1\le k<m} \frac{1}{k}A_k - \frac{1}{4n}\zeta_m(2) + \sum_{1\le k<m} \frac{B_{2k}(1-2^{-2k})}{n^k}\zeta_m(2k+1) + O(m^{-2r}).$$

For all fixed r. The necessary values of $\zeta_m(s)$ can themselves be obtained from Euler's summation formula,

$$\zeta_m(s) = m^{-s} + \frac{1}{2}m^{-s} + \sum_{1\le k<r} \frac{B_{2k}m^{-s-2k}}{s-1} + O(m^{-s-2r-1}).$$

Finally, we need to compute $\zeta'(2)$. Once again, Euler's summation formula gives a satisfactory approach, since it implies that

$$-\zeta'(s) = m^{-s}\ln m + \frac{m^{-s}}{2} + \frac{\ln m}{2m^s} + \sum_{1\le k<r} \frac{B_{2k}}{2km^{s-2k}}(s-1) + O(m^{-s-2r-1}\ln m).$$

Using a computer, we can find these approximations to 40 digits:

$$\frac{12\ln 2}{\pi^2} = 0.84276\quad 59132\quad 72194\quad 51690\quad 72631\quad 93963\quad 96411\quad 55945,$$
$$B = -1.16448\quad 10529\quad 30025\quad 01180\quad 53126\quad 40319\quad 36021\quad 74884,$$
$$I = -0.01958\quad 27168\quad 97011\quad 53218\quad 32291\quad 14034\quad 54827\quad 84625,$$
$$C = 1.46707\quad 80794\quad 33975\quad 47289\quad 77984\quad 84707\quad 22995\quad 34499.$$

References

[Hei34] Hans Heilbronn. On the average length of a class of finite continued fractions. *Journal of the London Mathematical Society*, 1934.

[Knu76] David E. Knuth. Evaluation of porter's constant. *Computers Mathematics with Applications*, 2:137–139, 1976.

[Knu81] David E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1981.

[Por75] J.W. Porter. On a theorem of heilbronn. *Journal of the London Mathematical Society*, 1975.

[Sha94] Jeremy Shallit. Origins of the analysis of the euclidean algorithm. *Historia Mathematica*, 21:401–419, 1994.

[Ton71] V. A. Tonkov. On the average length of euclidean algorithm. *Seriya Matematicheskaya*, 1971.

Euler Circle, Mountain View, CA 94040