One (more) line on the most Ancient Algorithm in History

Bruno Grenet, Ilya Volkovich

One (more) line on the most Ancient Algorithm in History

Bruno Grenet* Ilya Volkovich†

Abstract

We give a new simple and short ("one-line") analysis for the runtime of the well-known Euclidean Algorithm. While very short simple, the obtained upper bound in near-optimal.

1 Introduction

Perhaps the most ancient algorithm recorded in history is the Euclidean Algorithm. Published in 300 BC, the algorithm computes the greatest common divisor (gcd) of two integer numbers. (See Algorithm 1 for the description of the algorithm.) The following is a quote from [Knu98]: "[The Euclidean algorithm] is the granddaddy of all algorithms, because it is the oldest nontrivial algorithm that has survived to the present day."

The Euclidean Algorithm serves as a subroutine for many tasks such as: finding Bézout's coefficients, computing multiplicative inverses and the RSA algorithm, Chinese Remainder Theorem and others. For more details on these and other applications, see [Knu97, Knu98]. Yet, despite its simplicity, the actual runtime complexity cannot be easily inferred from its description due to the recursive nature of the algorithm.

As the operations inside each iteration are "elementary", the actual runtime complexity is dominated by the number of (recursive) iterations of the algorithm. Indeed, various upper bounds for this number were given [Lam44, Sed83, Knu97, Knu98, CLRS09]. The first one, presented in [Lam44], ties the number of iteration with the so-called *Fibonacci* numbers. Another kind of analysis shows that the larger argument shrinks by a factor of at least 2 every *two* iterations (see e.g. [Knu97, CLRS09]). Yet, the proof of this claim requires a somewhat non-trivial case-analysis. Note that in the case of univariate polynomials, the analysis is much simpler: the degree of the highest-degree polynomial decreases at each step. A similar easy argument was lacking in the integer case since the same phenomenon does not occur: it does **not** hold that the bit-size of the largest integer decreases at each step.

Our new analysis provides a clean, "one-line" upper bound which turns out to be optimal. The main idea is based on the so-called *Potential Method* (see e.g. [CLRS09]). More specifically, we define appropriate *potential* functions and show that these functions lose a constant fraction of their mass with every recursive iteration of the algorithm. At the same time, the functions are bounded away from zero. Formally, we give a simple proof to the following theorems:

Theorem 1. *For any $x, y \in \mathbb{N}$, the Euclidean Algorithm performs at most $\log_{1.5}(x+y)+1$ iterations.*

Using a slightly more sophisticated potential function yields an even tighter result.

Theorem 2. *For any $x, y \in \mathbb{N}$, the Euclidean Algorithm performs at most $\log_\phi(\phi x + y)$ iterations.* Here $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61803398875$ denotes the *golden ratio*.

Our analysis should be contrasted with the analysis of [Lam44] which states that if the Euclidean algorithm requires m iterations, then it must be the case that $x \geq F_{m+2}$ and $y \geq F_{m+1}$, where F_i-s are the Fibonacci numbers: $1, 1, 2, 3, 5, \ldots$. And, indeed, selecting x and y as consecutive Fibonacci numbers demonstrates the tightness of **this** analysis. Now for a general pair x and y we have:

$$\phi x + y \geq \phi F_{m+2} + F_{m+1} \geq F_{m+3} \gtrapprox \phi^{m+1}.$$

Therefore $m \lessapprox \log_\phi(\phi x + y)$, implying the optimality of **our** analysis.

*LIRMM, Université de Montpellier, CNRS, Montpellier, France. Email: bruno.grenet@lirmm.fr.

†Department of EECS, CSE Division, University of Michigan, Ann Arbor, MI. Email: ilyavol@umich.edu.

2 The Algorithm and the Runtime Analysis

In this section we review the algorithm and give the new, simplified runtime analysis. The description of the algorithm is given below in Algorithm 1. The algorithm is given non-negative integers, x and y as an input. We assume w.l.o.g that $x > y$.

Input: Two non-negative integers, $x > y \in \mathbb{N}$
Output: $\gcd(x, y)$

1 **if** $y = 0$ **then return** x;
2 **if** $y = 1$ **then return** 1;
3 **return** $\gcd(y, x \bmod y)$

Algorithm 1: Euclidean Algorithm

For the sake of analysis, let us fix $x > y$ and let m denote the number of the (recursive) iterations of algorithm.

Definition 2.1. *For $1 \leq i \leq m$, let x_i and y_i denote the values of x and y in the i-th iteration of the algorithm, respectively. In particular, $x_1 = x, y_1 = y$.*

Observation 2.2. *For $1 \leq i \leq m$: $x_i > y_i \geq 0$.*

Proof. Follows from the fact that $y_i = x_{i-1} \bmod y_{i-1} = x_{i-1} \bmod x_i$, which attains values between 0 and $x_i - 1$. □

Definition 2.3. *For $1 \leq i \leq m$, we define $s_i \stackrel{\Delta}{=} x_i + y_i$.*

Next, we will show that s_i is a *potential* function for this algorithm. In particular, the function loses a constant fraction of its mass in every step, yet it is bounded away from zero. The following is immediate from the definition, given the above observation.

Corollary 2.4. *For $1 \leq i \leq m$: $s_i \geq 1$.*

The following lemma is the heart of the argument.

Lemma 2.5. *For $1 \leq i \leq m$: $s_i \leq 2/3 \cdot s_{i-1}$.*

Proof. Recall that $s_{i-1} = x_{i-1} + y_{i-1}$. Let us divide x_{i-1} by y_{i-1} with a reminder. In particular,

$$x_{i-1} = q_{i-1} \cdot y_{i-1} + r_{i-1}, \text{ where } 0 \leq r_{i-1} \leq y_{i-1} - 1.$$

Observe that $s_i = y_{i-1} + r_{i-1}$ and in addition, $q_{i-1} \geq 1$, as $x_{i-1} > y_{i-1}$. We obtain the following:

$$s_{i-1} = (q_{i-1}+1)y_{i-1}+r_{i-1} \geq 2y_{i-1}+r_{i-1} \geq 2y_{i-1}+r_{i-1}-(y_{i-1}-r_{i-1})/2 = 1.5y_{i-1}+1.5r_{i-1} = 1.5s_i.$$

The second inequality holds since $r_{i-1} < y_{i-1}$. □

By applying the lemma repeatedly, we obtain:

Corollary 2.6. *For $1 \leq i \leq m$: $s_i \leq s_1 \cdot (2/3)^{i-1}$.*

Theorem 1 follows immediately from the above.

Proof of Theorem 1. By Corollaries 2.4 and 2.6:

$$1 \leq s_m \leq s_1 \cdot (2/3)^{m-1} = (x + y) \cdot (2/3)^{m-1}.$$

Therefore, $m \leq \log_{1.5}(x + y) + 1$. □

2.1 Improved Analysis

In this section we show that using a slightly more sophisticated potential function yields an even tighter result. As before, for the sake of analysis, let us fix $x > y$ and let m denote the number of the (recursive) iterations of algorithm. In addition, let $\phi = \frac{1+\sqrt{5}}{2} \approxeq 1.61803398875$ denote the golden ratio.

Definition 2.7. *For $1 \leq i \leq m$, we define $s_i \stackrel{\Delta}{=} x_i + \frac{1}{\phi} \cdot y_i$.*

As before, for $1 \leq i \leq m$: $s_i \geq 1$. The following lemma mirrors Lemma 2.5.

Lemma 2.8. *For $1 \leq i \leq m$: $s_i \leq \frac{1}{\phi} \cdot s_{i-1}$.*

Proof. Recall that $s_{i-1} = x_{i-1} + \frac{1}{\phi} \cdot y_{i-1}$. Let us divide x_{i-1} by y_{i-1} with a reminder. In particular,

$$x_{i-1} = q_{i-1} \cdot y_{i-1} + r_{i-1}, \text{ where } 0 \leq r_{i-1} \leq y_{i-1} - 1.$$

Observe that $s_i = y_{i-1} + \frac{1}{\phi} \cdot r_{i-1}$ and in addition, $q_{i-1} \geq 1$, as $x_{i-1} > y_{i-1}$. We obtain the following:

$$s_{i-1} = \left(q_{i-1} + \frac{1}{\phi}\right) y_{i-1} + r_{i-1} \geq \left(1 + \frac{1}{\phi}\right) y_{i-1} + r_{i-1} = \phi \cdot y_{i-1} + r_{i-1} = \phi \left(y_{i-1} + \frac{1}{\phi} \cdot r_{i-1}\right) = \phi \cdot s_i.$$

Recall that $1 + \frac{1}{\phi} = \phi$. □

The proof of Theorem 2 follows by repeating the previous argument. Actually, a slightly better bound can be obtained: Since the algorithm stops as soon as $y_m \leq 1$, $y_{m-1} \geq 2$, and moreover $x_{m-1} > y_{m-1}$. Thus $s_m \geq 3 + \frac{2}{\phi} = \phi^3$ and since $s_m \leq s_1 \phi^{-m+2}$, $m \leq \log_\phi(x + \frac{1}{\phi} y) - 1$. This bound is tight, attained for consecutive Fibonacci numbers.

3 Conclusion & Open Questions

In this short note we add one (more) line of analysis to the well-known Euclidean Algorithm. The new analysis does not require any background and can be taught even in an introductory-level undergraduate class. It would be nice to see if we could replace other kind of analyses that rely on Fibonacci numbers by a potential argument. One such example is the analysis of the height of an AVL-Tree [AVL62, Sed83].

References

[AVL62] G. Adelson-Velsky and E. Landis. An algorithm for the organization of information. In *Proceedings of the USSR Academy of Sciences (in Russian)*, volume 146, pages 263–266, 1962.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009.

[Knu97] D. E. Knuth. *The art of computer programming, Volume I: Fundamental Algorithms, 3rd Edition.* Addison-Wesley, 1997.

[Knu98] D. E. Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition.* Addison-Wesley, 1998.

[Lam44] G. Lamé. Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers. *Comptes Rendus Acad. Sci.*, (19):867–870, 1844.

[Sed83] R. Sedgewick. *Algorithms*. Addison-Wesley, 1983.