

An Analysis of Lehmer's Euclidean GCD Algorithm

Jonathan Sorenson*

Department of Mathematics and Computer Science

Butler University

4600 Sunset Ave.

Indianapolis, Indiana 46208

sorenson@butler.edu

Abstract

Let u and v be positive integers. We show that a slightly modified version of D. H. Lehmer's greatest common divisor algorithm will compute $\gcd(u, v)$ (with $u > v$) using at most $O\{(\log u \log v)/k + k \log v + \log u + k^2\}$ bit operations and $O(\log u + k2^{2k})$ space, where k is the number of bits in the multiprecision base of the algorithm. This is faster than Euclid's algorithm by a factor that is roughly proportional to k .

Letting n be the number of bits in u and v , and setting $k = \lfloor (\log n)/4 \rfloor$, we obtain a subquadratic running time of $O(n^2/\log n)$ in linear space.

1 Introduction

Let u and v be positive integers. The *greatest common divisor* (GCD) of u and v is the largest integer d such that d divides both u and v . The most well-known algorithm for computing GCDs is the Euclidean Algorithm. Much is known about this algorithm: the number of iterations required is $\Theta(\log v)$, and the worst-case running time is $\Theta(\log u \log v)$, where time is measured in bit operations. For further information, see [3, 4, 13] and also [10].

However, for multiprecision inputs, the Euclidean algorithm is not the best choice in practice. This is primarily because a multiprecision division operation is relatively expensive. D. H. Lehmer observed that many of these division steps can be avoided [11, 10]. His idea was to extract the leading digits \hat{u} and \hat{v} of u and v , and run the Euclidean algorithm on these single precision approximations of the inputs. A prefix of the resulting quotient sequence will match that of the actual inputs u and v , since $v/u \approx \hat{v}/\hat{u}$. In essence, this quotient sequence prefix can be applied to u and v in one multiprecision arithmetic step, thereby effectively "skipping" several multiprecision divisions. The resulting algorithm appears to be much more efficient than Euclid's algorithm in practice on large inputs. However, no rigorous theoretical support for this behavior is known.

*Supported in part by NSF grant CCR-9204414

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantages, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission. ISSAC'95 - 7/95 Montreal, Canada
©1995 ACM 0-89791-699-9/95/0007 \$3.50

In this paper, we remedy this situation by analyzing a slight modification of Lehmer's Euclidean GCD algorithm. This modified version, which we denote Algorithm **ML**, makes use of Jebelean's exact condition for determining the matching quotient sequence mentioned above [9]. In addition to the inputs u and v , Algorithm **ML** has a parameter k which indicates how many leading bits are used to compute the approximations \hat{u} and \hat{v} . We present our modified algorithm, along with some notation and background, in Section 2.

We then prove the following:

- On inputs u, v , with $u > v$, and parameter k , Algorithm **ML** computes $\gcd(u, v)$ using at most

$$O\left(\frac{\log v}{k} + k\right)$$

iterations of its main loop.

Our proof relies on a simple, but important lemma on continued fractions. We present this result in Section 3.

- On inputs u, v , with $u > v$, and parameter k , Algorithm **ML** computes $\gcd(u, v)$ using at most

$$O\left(\frac{\log u \log v}{k} + k \log v + \log u + k^2\right)$$

bit operations and $O(\log u + k2^{2k})$ space.

If u and v are at most n bits in length, then by setting $k = \lfloor (\log n)/4 \rfloor$ we obtain an algorithm with a subquadratic running time of $O(n^2/\log n)$ using $O(n)$ space.

Our proof relies not only on a bound for the number of iterations, but also on some simple results on arithmetic involving small numbers. We present this result in Section 4.

Thus, for small values of k , we have shown that our modified version of Lehmer's algorithm is faster than Euclid's algorithm by a factor proportional to k .

Although our results do not directly apply to most other implementations of Lehmer's methods, we expect that on typical inputs, the performance of Algorithm **ML** will be similar, if not inferior.

There are a number of algorithms, in addition to the Euclidean algorithm, for computing GCDs: the least-remainder Euclidean algorithm [5, 2], the binary algorithm [1, 10, 12,

19], the left-shift binary algorithms [1, 17], the k -ary or generalized binary algorithms [18, 8, 20], and others [14, 15]. The asymptotically fastest GCD algorithm is based on FFT methods and is due to Schönhage [16]. It has a running time of $O(n \log^2 n \log \log n)$ bit operations on inputs of n bits in length, but this algorithm is not considered practical. The k -ary algorithms mentioned above have $O(n^2 / \log n)$ running times, and are quite practical. The rest of the algorithms listed above have $O(n^2)$ running times. For a performance comparison, see [7] and [18].

Note that our Algorithm ML has a running time that is competitive with the k -ary algorithms, currently the fastest practical GCD algorithms. However, using Lehmer's methods provides two advantages: the running time is sensitive to the size of the smaller of the two inputs, and the exact quotient sequence of the Euclidean algorithm is generated, which may be needed in applications involving continued fractions.

In Section 5, we present some performance results comparing Algorithm ML with several other GCD algorithms.

For references on parallel GCD algorithms, see [18].

2 The Algorithm

In this section, we present our modified version of Lehmer's Euclidean GCD algorithm. We begin by briefly describing the main loop. We then review some standard notation regarding the sequences computed by the Euclidean algorithm. Finally, we present procedure **Lehmer**, the heart of the algorithm.

2.1 The Main Loop of Algorithm ML

Below, we give a Pascal-like pseudocode description of our main loop. As a convention, we use uppercase variable names to denote multiprecision integers, and lowercase variable names for single precision integers. For most computers today, $0 < k \leq 64$.

Let $\text{len}(x)$ denote the number of binary digits in the positive integer x , so that $\text{len}(x) := \lfloor \log_2 x \rfloor + 1$ when $x > 0$, and $\text{len}(0) = 1$.

Algorithm ML

```

INPUT: Integers  $U, V, k$ , with  $U \geq V > 0$  and  $k > 0$ ;
OUTPUT:  $\text{gcd}(U, V)$ ;
while  $V \neq 0$  do
  if  $\text{len}(U) - \text{len}(V) \leq k/2$  then
    Lehmer( $U, V, k$ );
  end if;
   $R := U \bmod V$ ;
   $U := V$ ;  $V := R$ ;
end while;
output( $U$ );

```

This is Euclid's algorithm with the insertion of a conditional call to the **Lehmer** procedure, which we will discuss below. Note that although we perform a division, or Euclidean step, every iteration, the number of iterations is significantly smaller than that of the Euclidean algorithm, due to the **Lehmer** procedure.

In practice, pointers may be used for U , V , and R to save time in the copy steps $U := V$; and $V := R$;

2.2 Sequences Based on Euclid's Algorithm

We now require the following definitions and notation.

Let u and v , with $0 < v < u$, denote the inputs to the algorithm. Then let the *remainder sequence* $\{u_i\}$ be defined as follows:

$$\begin{aligned} u_0 &= u; \\ u_1 &= v; \\ u_{i+2} &= u_i \bmod u_{i+1} \quad (i \geq 0). \end{aligned}$$

The *quotient sequence* $\{q_i\}$ is given by

$$q_{i+1} = \lfloor u_i / u_{i+1} \rfloor \quad (i \geq 0).$$

The *cosequences* $\{x_i\}$, $\{y_i\}$ are defined by

$$\begin{aligned} (x_0, y_0) &= (1, 0), \\ (x_1, y_1) &= (0, 1), \\ (x_{i+2}, y_{i+2}) &= (x_i, y_i) - q_{i+1}(x_{i+1}, y_{i+1}) \quad (i \geq 0), \end{aligned}$$

and satisfy the identity

$$u_i = x_i \cdot u + y_i \cdot v \quad (i \geq 0).$$

2.3 The Lehmer Subroutine

The **Lehmer** routine performs the single precision calculations that allow us to combine several Euclidean steps into a single multiprecision calculation. The idea is quite simple. First, we construct approximations \hat{u} and \hat{v} to the multiprecision integers U and V . We then perform Euclid's algorithm on \hat{u} and \hat{v} , while keeping track of the quotient sequence and the cosequences. We make use of Jebelean's exact condition to determine when the quotient sequence differs from that of V/U [9]. Finally, we use the cosequences to compute the new U and V values.

Procedure Lehmer(U, V, k)

```

 $h := \text{len}(U) - k$ ;
 $\hat{u} := \lfloor U/2^h \rfloor$ ;  $\hat{v} := \lfloor V/2^h \rfloor$ ;
 $(x_0, y_0) := (1, 0)$ ;  $(x_1, y_1) := (0, 1)$ ;
 $i := 0$ ;
 $done := false$ ;
repeat
   $q_{i+1} := \lfloor \hat{u}/\hat{v} \rfloor$ ;
   $(x_{i+2}, y_{i+2}) := (x_i, y_i) - q_{i+1}(x_{i+1}, y_{i+1})$ ;
   $(\hat{u}, \hat{v}) := (\hat{v}, \hat{u} - q_{i+1}\hat{v})$ ;
   $i := i + 1$ ;
  { Determine if  $q_i$  is correct (Jebelean's condition). }
  if  $i$  is even then
     $done := \hat{v} < -x_{i+1}$  or  $\hat{u} - \hat{v} < y_{i+1} - y_i$ ;
  else
     $done := \hat{v} < -y_{i+1}$  or  $\hat{u} - \hat{v} < x_{i+1} - x_i$ ;
  endif;
until  $done$ ;
{ We know  $q_1, \dots, q_{i-1}$  were correct,  $q_i$  is incorrect. }
 $(U, V) := (x_{i-1}U + y_{i-1}V, x_iU + y_iV)$ ;
return;

```

Observe that \hat{u} and \hat{v} can be found via bit extraction; 2^h is not needed, nor is the multiprecision division. Also, since $\hat{u}, \hat{v} \leq 2^k$, all integers represented by lowercase variables are at most 2^k in absolute value. Finally, arrays for the quotient sequence and the cosequences are not required; only the last 3 terms of each sequence are needed.

3 Counting Iterations

In this section we prove an upper bound on the number of iterations of the main loop of Algorithm ML needed to compute the $\gcd(u, v)$. We begin with an important but simple lemma on continued fractions.

3.1 A Lemma on Continued Fractions

Let $\alpha = v/u$ so that $0 < \alpha < 1$. Then the continued fraction expansion of α is

$$\alpha = [0, q_1, q_2, \dots]$$

where $\{q_i\}$ is the quotient sequence for u and v as defined previously. The convergents a_n/b_n to α are rational numbers in reduced form given by

$$\frac{a_n}{b_n} = [0, q_1, q_2, \dots, q_n] \quad (n \geq 0)$$

and are related to the cosequences by

$$(a_i, b_i) = (|x_{i+1}|, |y_{i+1}|) \quad (i \geq 0).$$

Note that $x_i/y_i < 0$ for $i > 1$.

Consider the continued fraction expansions of two real numbers α and β . If α and β are approximately equal, then for some $n > 0$, the first n quotients in their quotient sequences will match. We would expect that, the smaller the value of $|\alpha - \beta|$, the closer the n th convergents a_n/b_n come to approximating both α and β .

In the following lemma, we prove this conjecture. Note that α and β may be either rational or irrational.

Lemma 1 *Let $0 < \alpha < \beta < 1$, with $|\alpha - \beta| < \delta$. Let n be the unique non-negative integer such that, if the continued fraction expansions of α and β are*

$$\begin{aligned} \alpha &= [0, q_1, \dots, q_n, q_{n+1}, \dots], \\ \beta &= [0, q'_1, \dots, q'_n, q'_{n+1}, \dots], \end{aligned}$$

then $q_i = q'_i$ for all $i \leq n$, and $q_{n+1} \neq q'_{n+1}$. Let a_n/b_n be the n th convergents to α (and β). Then

$$\left| \frac{a_n}{b_n} - \alpha \right|, \left| \frac{a_n}{b_n} - \beta \right| \leq \delta + \frac{\sqrt{2\delta}}{b_n}.$$

Proof Let a'_i/b'_i be the i th convergents to β , so that $(a_i, b_i) = (a'_i, b'_i)$ for $i \leq n$.

First we assume $b_{n+1} \geq 1/\sqrt{2\delta}$. Then, by Theorem 171 of [6], we have

$$\left| \frac{a_n}{b_n} - \alpha \right| \leq \frac{1}{b_n b_{n+1}} \leq \frac{\sqrt{2\delta}}{b_n}$$

and

$$\left| \frac{a_n}{b_n} - \beta \right| \leq \left| \frac{a_n}{b_n} - \alpha \right| + |\alpha - \beta| \leq \delta + \frac{\sqrt{2\delta}}{b_n}.$$

The case when $b'_{n+1} \geq 1/\sqrt{2\delta}$ is similar.

So we may assume that $b_{n+1}, b'_{n+1} < 1/\sqrt{2\delta}$. We will show this case leads to a contradiction. By Theorem 184 of [6], we have

$$\begin{aligned} \left| \frac{a'_{n+1}}{b'_{n+1}} - \alpha \right| &> \frac{1}{2b'^2_{n+1}} > \delta; \\ \left| \frac{a_{n+1}}{b_{n+1}} - \beta \right| &> \frac{1}{2b^2_{n+1}} > \delta. \end{aligned}$$

Since a_{n+1}/b_{n+1} is closer to α than to β , and a'_{n+1}/b'_{n+1} is closer to β than to α , neither of the convergents lie in the interval $[\alpha, \beta]$; we must have $a_{n+1}/b_{n+1} < \alpha$ and $a'_{n+1}/b'_{n+1} > \beta$. This contradicts the fact that convergents are alternately smaller and larger than what they converge to (see [6, Theorem 164]). \square

3.2 The Main Result

The following theorem establishes an upper bound on the number of iterations of the main loop of Algorithm ML.

We write $f(n) \ll g(n)$ to denote $f(n) = O(g(n))$.

Theorem 2 *Let $U > V \geq 0$ and $k > 0$ be integers. Algorithm ML requires at most*

$$O\left(k + \frac{\log V}{k}\right)$$

iterations of its main loop to compute $\gcd(U, V)$.

Proof For each iteration of the main loop we have either

1. $\text{len}(U) - \text{len}(V) \leq k/2$ and $\text{len}(V) \geq k$ or
2. $\text{len}(U) - \text{len}(V) > k/2$ and $\text{len}(V) \geq k$ or
3. $\text{len}(V) < k$.

We will show that there are at most $O(1 + (\log V)/k)$ iterations of types (1) and (2), and $O(k)$ iterations of type (3).

Case (1): $\text{len}(U) - \text{len}(V) \leq k/2$ and $\text{len}(V) \geq k$.

Our goal is to establish that

$$x_i U + y_i V \ll U 2^{-k/2},$$

where i, x_i , and y_i are as computed by the Lehmer routine. Assuming this is true, then it is easy to complete our theorem, as we now explain.

Observe that $x_i U + y_i V$ is the final value of U after one loop iteration; after Lehmer finishes, the Euclidean step copies this value from V to U . As a result, each case (1) iteration reduces U by a factor $\gg 2^{k/2}$. Hence the number of iterations is at most $O(1 + (\log V)/k)$: 1 for the first iteration, and $(\log V)/k$ for the rest, since V is an upper bound for U during the second and subsequent iterations.

We now proceed to prove our claim.

The Lehmer routine computes the value i and constructs the quotient sequence

$$\hat{v}/\hat{u} = [0, q_1, \dots, q_{i-1}, q_i, \dots]$$

such that, if $\{Q_j\}$ is the quotient sequence for V/U , then $q_j = Q_j$ for $j < i$.

First we assume $|y_i| \geq 2^{k/2}$. Since $|x_i/y_i|$ are convergents to V/U , by [6, Theorem 171] we have

$$\begin{aligned} x_i U + y_i V &= U|y_i| \left| \frac{V}{U} + \frac{x_i}{y_i} \right| = U|y_i| \left| \frac{V}{U} - \frac{x_i}{y_i} \right| \\ &< U|y_i| \leq U 2^{-k/2}. \end{aligned}$$

So now we may assume $|y_i| < 2^{k/2}$. We will apply Lemma 1 with $\alpha, \beta \in \{\hat{v}/\hat{u}, V/U\}$, $n = i - 1$, and $\delta = O(2^{-k})$. Since the convergents $a_n/b_n = |x_i/y_i|$, we have

$$\begin{aligned} x_i U + y_i V &= U|y_i| \left| \frac{V}{U} - \frac{x_i}{y_i} \right| \leq U|y_i| \left(\delta + \frac{\sqrt{2\delta}}{|y_i|} \right) \\ &\ll U \left(\frac{y_i}{2^k} + \frac{1}{2^{k/2}} \right) \ll U 2^{-k/2}. \end{aligned}$$

Case (2): $\text{len}(U) - \text{len}(V) > k/2$ and $\text{len}(V) \geq k$.

Let n denote the number of iterations when case (2) occurs, with $\{Q_1, Q_2, \dots, Q_n\}$ the corresponding sequence of quotients for case 2. It suffices to show that $n = O(1 + (\log V)/k)$. We have $\log Q_i \gg \text{len}(U) - \text{len}(V) \gg k$ for all i . This gives us $(n-1)k \ll \sum_{i=2}^n \log Q_i \ll \log V$, which completes this case of the proof.

Case (3): $\text{len}(V) < k$.

Once $\text{len}(V)$ falls below k , Algorithm **ML** performs exactly as the ordinary Euclidean algorithm does. Thus, the number of iterations for case (3) is $O(k)$, and our proof is complete. \square

Note that this bound is tight in the worst case. Let the inputs U and V be the $n+1$ st and n th Fibonacci numbers (the worst-case inputs for the Euclidean algorithm). Then $n = \Theta(\log V)$, and Algorithm **ML** will perform $\Theta(n/k + k)$ iterations. We leave the verification of this to the reader.

4 Complexity

In this section we present an upper bound on the complexity of Algorithm **ML**. We begin with a discussion of our model of computation and a lemma on arithmetic using a k -bit multiprecision base.

4.1 Model of Computation

Our model of computation is a random access machine (RAM) with potentially infinite memory addressable at the bit level. We measure the complexity of an algorithm in terms of the number of bit operations it performs.

We require the following lemma on the complexity of arithmetic using a k -bit multiprecision base.

Lemma 3 *Let $k > 0$ and $U \geq V > 0$ be integers. Using a precomputed table of $O(k2^{2k})$ bits, it is possible to*

- *compute the product UV using $O(\log U + (\log U \log V)/k)$ bit operations, and*
- *compute the quotient and remainder when dividing U by V using $O(\log U + (\log V \log \lceil U/V \rceil)/k)$ bit operations.*

Precomputing the table takes $O(k^2 2^{2k})$ bit operations.

Proof (Sketch) The basic idea is to precompute the product and quotient/remainder of all pairs of integers $\leq 2^k$. Then, to perform arithmetic with larger numbers, break their binary expansions into k -bit blocks and perform the arithmetic in base 2^k , using the precomputed table to compute products and quotients of numbers bounded by 2^k . For more details, see [18, Lemma 4.1]. \square

As a corollary, we have that an $O(n)$ -bit integer can be multiplied or divided by an $O(k)$ -bit integer in $O(n)$ bit operations, under the same conditions as the lemma above.

Note that, although the construction mentioned in the previous lemma would not be practical, in practice most computer hardware performs the multiplication and division of integers bounded by the word size of the machine in effectively constant time. Hence, the performance of multiprecision arithmetic in practice mirrors the effect of this lemma, but without any additional work on the part of the programmer.

4.2 The Main Result

We are now ready to prove the following theorem.

Theorem 4 *Let U, V, k be integers with $U \geq V > 0$ and $k > 0$. Algorithm **ML** computes $\text{gcd}(U, V)$ using at most*

$$O\left(\frac{\log U \log V}{k} + k \log V + \log U + k^2\right)$$

bit operations. $O(k^2 2^{2k})$ precomputation time and $O(\log U + k2^{2k})$ space are also required.

Proof Once $\text{len}(V) < k$, then the time needed by Algorithm **ML** is at most $O(\log U)$ for the first division, followed by $O(k^2)$ operations afterwards. So we may assume $\text{len}(V) \geq k$.

We will now examine the time spent in the **Lehmer** routine, after which we will examine the time spent performing the Euclidean steps.

Let n denote the number of times procedure **Lehmer** is called. The steps in procedure **Lehmer** that precede the repeat loop can be done in $O(k + \log U)$ bit operations. Recall that all integers used in the loop are bounded by 2^k . Since the loop essentially performs the Euclidean algorithm, it takes $O(k^2)$ bit operations. And by Lemma 3, the final multiprecision step takes $O(\log U)$ time. Thus, the total time spent in procedure **Lehmer** is $O(n(k^2 + \log U))$. From the proof of Theorem 2, we have $n = O(1 + (\log V)/k)$, giving a total time of $O((\log U \log V)/k + k \log V + \log U + k^2)$.

Now let n denote the number of times the Euclidean steps $R := U \bmod V$; $U := V$; and $V := R$; are performed with $\text{len}(V) \geq k$. Let $\{Q_j\}$ denote the sequence of quotients computed. Then by Lemma 3, the time spent for the j th Euclidean step is no more than $O(\log U + (\log V \log Q_j)/k)$. Summing over all j , we obtain that the running time is at most

$$O\left(n \log U + \frac{\log V}{k} \sum_{j=1}^n \log Q_j\right).$$

But we have $\sum_{j=1}^n \log Q_j \leq \log U$, and from the proof of Theorem 2 we have $n = O(1 + (\log V)/k)$. This gives us a total of $O(\log U + (\log U \log V)/k)$ bit operations spent performing Euclidean steps.

That completes the proof. \square

5 Timing Results

We conclude with timing results.

In our first table (Table 1) we have the average running times of the Euclidean algorithm, the least-remainder Euclidean algorithm, the k -ary (generalized binary) and left-shift k -ary algorithms, an implementation of Lehmer's algorithm based on Knuth [10] (using 15 and 30 bit sizes for \hat{u} and \hat{v}), and Algorithm **ML** for values of k ranging from 5 to 30 bits. This data was obtained using Turbo C++ on a CompuAdd 486/33 PC running MS-DOS version 6. Each data point is the average time from 100 pseudo-random input pairs.

Table 1: Average Running Times in CPU Seconds

Algorithm	Input Size in Decimal Digits			
	100	250	500	1000
<i>Euclidean</i>	0.049	0.245	0.906	3.45
<i>LR Euclidean</i>	0.044	0.220	0.813	3.10
<i>k-ary</i>	0.049	0.171	0.622	1.81
<i>LS k-ary</i>	0.044	0.177	0.578	1.96
<i>Lehmer (15 bit)</i>	0.041	0.202	0.747	2.85
<i>Lehmer (30 bit)</i>	0.029	0.119	0.388	1.37
ML with $k = 5$	0.072	0.358	1.319	5.04
ML with $k = 10$	0.055	0.274	1.014	3.89
ML with $k = 15$	0.040	0.201	0.741	2.84
ML with $k = 20$	0.034	0.160	0.582	2.19
ML with $k = 25$	0.029	0.134	0.474	1.77
ML with $k = 30$	0.026	0.117	0.403	1.49

This data, though interesting, is highly dependent on the programmer and the particular computer platform used. In our next table (Table 2), however, we present data which is implementation independent: the average number of main loop iterations. This data was generated using the same methods as for the first table. For the algorithms based on Lehmer's methods, only the number of multiprecision loop iterations were counted; counting all iterations would simply give the same count as that of the Euclidean algorithm.

Table 2: Average Number of Main Loop Iterations

Algorithm	Input Size in Decimal Digits			
	100	250	500	1000
<i>Euclidean</i>	195	483	972	1935
<i>LR Euclidean</i>	136	336	675	1345
<i>k-ary</i>	86	214	343	616
<i>LS k-ary</i>	70	175	315	574
<i>Lehmer (15 bit)</i>	71	170	337	667
<i>Lehmer (30 bit)</i>	34	74	142	277
ML with $k = 5$	127	316	636	1268
ML with $k = 10$	73	179	359	715
ML with $k = 15$	50	121	240	476
ML with $k = 20$	39	91	180	354
ML with $k = 25$	32	74	144	282
ML with $k = 30$	29	63	121	236

Although modified for purposes of analysis, Algorithm **ML** performs nearly as well as a traditional implementation of Lehmer's methods. We also observe that, at least for this implementation, the methods based on Lehmer's ideas are very competitive with the k -ary (generalized binary) algorithms.

For timing results using a Unix-based implementation and the GMP package, see Jebelean [7] and Weber [20]. Other results also appear in [17, 18].

References

- [1] R. P. Brent. Analysis of the binary Euclidean algorithm. In J. F. Traub, editor, *Algorithms and Complexity*, pages 321–355, Academic Press, 1976.
- [2] J. L. Brown and R. L. Duncan. The least remainder algorithm. *Fibonacci Quarterly*, 9(4):347–350, 1971.
- [3] G. E. Collins. The computing time of the Euclidean algorithm. *SIAM Journal on Computing*, 3(1):1–10, 1974.
- [4] J. Dixon. The number of steps in the Euclidean algorithm. *Journal of Number Theory*, 2:414–422, 1970.
- [5] A. W. Goodman and W. M. Zaring. Euclid's algorithm and the least-remainder algorithm. *American Mathematical Monthly*, 59:156–159, 1952.
- [6] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 5th edition, 1979.
- [7] T. Jebelean. Comparing several GCD algorithms. In *11th IEEE Symposium on Computer Arithmetic*, Windsor, Ontario, 1993.
- [8] T. Jebelean. A generalization of the binary GCD algorithm. In *ISSAC '93*, Kiev, 1993.
- [9] T. Jebelean. Improving the multiprecision Euclidean algorithm. In *DISCO '93: International Symposium on Design and Implementation of Symbolic Computation Systems*, pages 45–58, Springer-Verlag, Gmunden, Austria, 1993. LNCS 722.
- [10] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Volume 2, Addison-Wesley, Reading, Mass., 2nd edition, 1981.
- [11] D. H. Lehmer. Euclid's algorithm for large numbers. *American Mathematical Monthly*, 45:227–233, 1938.
- [12] G. Norton. Extending the binary GCD algorithm. In J. Calmet, editor, *Proceedings of the 3rd International Conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 363–372, Springer-Verlag, 1985. LNCS 229.
- [13] G. Norton. On the asymptotic analysis of the Euclidean algorithm. *Journal of Symbolic Computation*, 10, 1990.
- [14] G. Norton. A shift-remainder GCD algorithm. In L. Huguet and A. Poli, editors, *Proceedings of the 5th International Conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 350–356, Springer-Verlag, 1987. LNCS 356.
- [15] G. B. Purdy. A carry-free algorithm for finding the greatest common divisor of two integers. *Computers & Mathematics with Applications*, 9(2):311–316, 1983.
- [16] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.
- [17] J. Shallit and J. Sorenson. Analysis of a left-shift binary GCD algorithm. *Journal of Symbolic Computation*, 17:473–486, 1994.
- [18] J. Sorenson. Two fast GCD algorithms. *Journal of Algorithms*, 16:110–144, 1994.
- [19] J. Stein. Computational problems associated with Raca algebra. *Journal of Computational Physics*, 1:397–405, 1967.
- [20] K. Weber. *The Accelerated Integer GCD Algorithm*. Technical Report ICM-9307-55, Institute for Computational Mathematics, Kent State University, July 1993.