On Schönhage's algorithm and subquadratic integer gcd computation

Niels Möller

April 12, 2005

Abstract

We describe a hitherto unpublished subquadratic left-to-right gcd algorithm, discovered by Schönhage back in 1987, and compare it to the first subquadratic gcd algorithm discovered by Knuth and Schönhage, and to the binary recursive gcd algorithm of Stehlé and Zimmermann. When reorganized into half-gcd form, we get a novel gcd algorithm which runs slightly faster than earlier algorithms, and is much simpler to implement. The subquadratic gcd algorithms all have the same asymptotic running time, $n(\log n)^2 \log\log n$.

1 Introduction

In this paper, we describe four subquadratic gcd algorithms: Schönhage's algorithm from 1971, Stehlé's and Zimmermann's binary recursive gcd, an unpublished algorithm discovered by Schönhage in 1987, and a new variant of the latter. The algorithms are compared with respect to running time and implementation complexity. The new algorithm is slightly faster than earlier algorithms, and much simpler to implement.

The paper is organized as follows. First we review the development of integer gcd algorithms in recent years. Section 2 describes the general structure and flavor of the subquadratic gcd algorithms, the idea of using a half-gcd function, and the resulting asymptotic running time. In Section 3, we briefly describe one variant of Schönhage's 1971 algorithm, and in Section 4, we describe the binary recursive gcd algorithm. The objective of these two sections is to provide sufficient details so that the new algorithm can be compared to earlier algorithms; we define the corresponding half-gcd functions, but we don't provide correctness proofs or detailed analysis.

Section 5 describes a gcd algorithm modeled on Schönhage's algorithm for reduction of binary quadratic forms [7], and in Section 6, this algorithm is reorganized into half-gcd form, resulting in a novel gcd algorithm. Section 7 reports about the implementation of the different gcd algorithms, their running times and code complexity.

History

Euclid's algorithm for computation of the greatest common divisor is one of the oldest algorithms known. For inputs of size n, this algorithm runs in time $O(n^2)$, and since the total size of the remainders is $O(n^2)$, no algorithm which computes all the remainders can run asymptotically faster than this.

Lehmer's algorithm from 1938 cuts the running time of Euclid's algorithm by a constant factor [4]. Lehmer's algorithm computes the initial part of the quotient sequence using single or double precision arithmetic on the most significant words of the input, and applies several quotients at once to the multi-precision numbers. A precise condition for how many correct quotients can be computed in this way was given by Tudor Jebelean in 1995 [2].

Euclid's and Lehmer's algorithms essentially work from the most significant end of the input numbers, and compute a sequence of quotients. A different method, binary gcd, was discovered by Stein in 1961 [10]. This algorithm starts from the *least* significant end. It divides out powers of two, and repeatedly uses subtraction to cancel the least significant bits. In 1994, Jonathan Sorenson described a k-ary reduction which gains speed over the basic binary algorithm by doing more work on just the least significant words [8] (c.f. to Lehmer's algorithm which does more of the work on the most significant one or two words), and this method was improved further in Ken Weber's accelerated gcd algorithm from 1995 [12].

Up to the 20'th century, gcd algorithms with quadratic running time have been dominating, and it has been the asymptotically slowest of the algorithms for basic arithmetic. E.g., subquadratic multiplication using the Karatsuba method has been known since the 1960s, and multiplication based on FFT was discovered by several people (Strassen, Schönhage and Pollard) in the early 1970s, and these methods are also in wide use.

The first subquadratic algorithm for gcd computation was described by Donald Knuth in 1970 [3], and it was improved by Schönhage in the same year [6]. These algorithms are essentially algorithms for computing the continued fraction representation of a rational (or even real) number. One crucial observation is that even if the total length of the remainders produced by Euclid's algorithm is quadratic, the total length of the *quotients* is only $O(n)$. Schönhage's algorithm uses divide-and-conquer to recursively compute the quotient sequence. A clear description of the algorithm details, for both the integer and polynomial case, can be found in a paper by Klaus Thull and Chee K. Yap [11]. A similar algorithm is described tersely in [5].

Schönhage's 1971 algorithm is straightforward to apply to polynomial gcd, but, to quote [11]: "The integer HGCD algorithm turns out to be rather intricate". Both analysis and actual implementation is quite difficult and error prone. For the same reasons, the algorithm is seldom spelled out in detail in textbooks, and when it is, it has been plagued by errors.

In 1987, Schönhage worked out a related algorithm that doesn't suffer from the same intricacies, but this algorithm was not published at that time. For the next decade, the new algorithm was known to Schönhage and a few students and correspondents, but not generally known to other researchers or to authors of bignum packages.

In a paper from 1991, on the reduction of binary quadratic forms [7], Schönhage applies the same ideas to a different problem. This algorithm can be translated, step by step, into the unpublished gcd algorithm from 1987. Another application of the same ideas to a different algebraic context, is Weilert's algorithm for gcd computations on Gaussian integers [13]. In this paper, however, we will consider only the classic case of *integer* gcd.

Since the recursive left-to-right algorithm from 1971 seemed so difficult in practice, Stehlé and Zimmermann investigated if one could use a similar divide-and-conquer technique in a right-to-left fashion. In 2004 they published a binary recursive gcd algorithm [9], which is significantly simpler. It is related to the binary gcd algorithm in roughly the same way that Schönhage's algorithm is related to Euclid's algorithm.

As far as the author is aware, there's still no published reference for Schönhage's algorithm from 1987. This paper intends to fill that gap.

Notation

Sizes of numbers will be important, so we introduce the notation $\#x$ for the bit size of x. Define $\#x = \lceil \log_2(1 + |x|) \rceil$; then for $x \neq 0$, $\#x = s$ means that $2^{s-1} \leq |x| < 2^s$. When $\#$ is applied to a vector or matrix, it denotes the *maximum* bit size of the elements. Occasionally, we also need the minimum bit size, for which we use the notation $\underline{\#}(x, y) = \min(\#x, \#y)$.

For the binary algorithm, we use the notation $v(x)$ for the number of zeros at the least significant end of x. More precisely, $v(0) = \infty$, and $v(x) = k$ if 2^k divides x and $2^{-k}x$ is odd.

When comparing matrices, we use the partial order given by elementwise inequality. In particular, $M \geq 0$ means that the elements of M are non-negative. I denotes the identity matrix. For compactness, column vectors are sometimes written as $(x_1; x_2)$ and 2×2-matrices as $(a_{11}, a_{12}; a_{21}, a_{22})$. All matrices have integer elements.

2 General structure of subquadratic gcd algorithms

In general, subquadratic gcd is a divide-and-conquer strategy, based on the observation that we can do a significant fraction of the work by examining only half of the input. The differences between algorithms is in the precise definition of which fraction of the work is done, and which half of the input is used.

We use a function HGCD(a, b) which takes two n-bit numbers as input, and returns two smaller numbers α, β, of size roughly $n/2$, and a transformation matrix M whose elements also are roughly of size $n/2$. The idea is that the smaller numbers α, β should have the same gcd as a, b, and that the transformation M should be relevant not only for a, b, but for some larger numbers that a, b were extracted from.

The HGCD function is computed recursively, using an algorithm of the following form.

HGCD(A, B)

1 $n \leftarrow \#(A, B)$
2 $(a, A') \leftarrow$ SPLIT$(A, n/2)$
3 $(b, B') \leftarrow$ SPLIT$(B, n/2)$
4 $(\alpha, \beta, M) \leftarrow$ HGCD(a, b)
5 $(A, B) =$ ADJUST$(\alpha, \beta, M, A', B')$
 $\triangleright$ A, B are now of size $\approx 3n/4$
6 $(a, A') \leftarrow$ SPLIT$(A, n/2)$
7 $(b, B') \leftarrow$ SPLIT$(B, n/2)$
8 $(\alpha, \beta, M') \leftarrow$ HGCD(a, b)
9 $(A, B) =$ ADJUST$(\alpha, \beta, M', A', B')$
 $\triangleright$ A, B are now of size $\approx n/2$
10 $M \leftarrow M' \cdot M$
11 Return A, B, M

The function SPLIT(A, p) splits A into one piece a consisting of p bits, and a remaining piece A'. The function ADJUST$(\alpha, \beta, M', A', B')$ applies M to the parts A' and B' that were not involved in the previous HGCD call, and combines with the values α and β that were returned from that call.

With a fast implementation of HGCD, a fast GCD algorithm can be written as

GCD(A, B)

while $\#(A, B) >$ GCD-THRESHOLD
 do
  $n \leftarrow \#(A, B)$
  $(a, A') \leftarrow$ SPLIT$(A, n/2)$
  $(b, B') \leftarrow$ SPLIT$(B, n/2)$
  $(\alpha, \beta, M) \leftarrow$ HGCD(a, b)
  $(A, B) \leftarrow$ ADJUST$(\alpha, \beta, M, A', B')$
return GCD-BASE(A, B)

Each round through the loop reduces the size of A and B by a factor $3/4$, and GCD-BASE should be some good quadratic gcd-algorithm, which is used once the numbers are small enough. We will focus on the HGCD algorithm, since GCD is comparatively trivial.

2.1 Asymptotic running time

Let $T(n)$ denote the maximum running time of HGCD, with inputs of size n, and let $\mu(n)$ denote the maximum time for multiplying or dividing two n-bit numbers. The SPLIT function takes $O(n)$ time. The matrix multiplication at line 10 of HGCD corresponds to small number of scalar multiplications. We haven't yet specified what ADJUST does, but let's assume that it performs a bounded number of multiplications and divisions of n-bit numbers. Then it follows that $T(n) \leq 2T(n/2) + c\mu(n)$ for some constant c, and this inequality implies

$$T(n) \leq c\mu(n) \log_2 n$$

When FFT methods are used for multiplication, and Newton's method is used for division[1], $\mu(n) = O(n \log n \log \log n)$, and then we get the asymptotic running time $T(n) = O(n (\log n)^2 \log \log n)$.

The additional loop in the GCD algorithm contributes only a constant factor. One round through the loop takes time $T(n/2) + c'\mu(n)$, and reduces the size by a factor $3/4$. Then the total time of GCD is bounded by

$$2T(n) + 4c'\mu(n)$$

Hence, GCD needs asymptotically twice as much time as HGCD with inputs of the same size.

3 Classical Schönhage gcd

Given two numbers a and b, $a > b$, define the quotient sequence q_i and the remainders r_i as

$$r_0 = a \qquad r_1 = b$$
$$q_i = \lfloor r_{i-1}/r_i \rfloor \qquad r_{i+1} = r_{i-1} - q_i r_i$$

Also define two co-sequences u_i and v_i, such that $r_i = u_i a + v_i b$:

$$u_0 = 1 \qquad v_0 = 0$$
$$u_1 = 0 \qquad v_1 = 1$$
$$u_{i+1} = u_{i-1} - q_i u_i \qquad v_{i+1} = v_{i-1} - q_i v_i$$

[1] In practice, division of large numbers will not contribute much to the running time, since large quotients are very unlikely.

The co-factors are bounded by $|u_k| \leq b/r_{k-1}, |v_k| \leq a/r_{k-1}$.

Classical Schönhage gcd uses a HGCD-Q function that computes the first half of the quotient sequence. The HGCD-Q function is used with inputs a and b that are the most significant parts of some larger numbers A and B, and we want to ensure that the quotients computed for a and b are correct also for A and B. This is similar to Lehmer's algorithm, but we use most significant parts a and b that are larger than just one or two words.

The following criterion guarantees that quotients computed from just the most significant parts of two numbers are in fact valid for the full numbers.

Lemma 1 (Jebelean's criterion) *Let $a > b > 0$, and let r_i, q_i be the quotient and remainder sequences as defined above. Assume that the following relations are satisfied for $0 \leq i \leq k$:*

$$r_{i+1} \geq \max(-u_{i+1}, -v_{i+1})$$
$$r_i - r_{i+1} \geq \max(u_{i+1} - u_i, v_{i+1} - v_i)$$

Then for any $p > 0$ and any A' and B', with $0 \leq A', B' < 2^p$, the sequence $q_1, \ldots, q_k$ is the initial quotient sequence for the numbers $A = a2^p + A', B = a2^p + B'$.

We refer to [2] for the proof. There is also a simplified version of the criterion, using only the size of the remainders.

Lemma 2 (Jebelean's simplified criterion) *Let $a > b > 0$, and let r_i, q_i be the remainder sequence. Let $n = \#a$ and $m = \lceil n/2 \rceil$. If k is such that $\#r_{k+2} > m$, then Jebelean's criterion is satisfied for $0 \leq < \leq k$.*

Note that $\#r_k > m + 1$ is necessary but *not* sufficient.

We can now define precisely what the HGCD-Q function is expected to return. The input is the two numbers a and b, with $a \geq b > 0$ and $n = \#a$. Let $m = \lceil n/2 \rceil$. If $\#b \leq m$ or $\#(a \bmod b) \leq m$, the function fails. Otherwise, it computes the quotient sequence until it encounters the first remainder r_{k+3} that fits in m bits. It then returns r_k, r_{k+1} and a matrix M made up from the corresponding co-factors. The matrix elements are bounded by $\#M \leq \lfloor n/2 \rfloor - 1$.

It's possible to save some work by returning also r_{k+2} and r_{k+3}, and use Jebelean's full criterion to check if the corresponding quotients are correct. But for simplicity we omit the details of this variation.

The algorithm in Figure 1 computes HGCD-Q recursively. Besides the current remainders and cofactors, the algorithm also needs to maintain a quotient stack, with the quotients leading up to the current remainders. This book-keeping costs $O(n)$ in both space and time, and is omitted in the algorithm description. (According to [11], the previous quotient can also be recovered from M, without using any explicit quotient stack).

The source of most of the complexity of the algorithm is that we don't have precise control over the size of the remainders after the second recursive call. At line 24 we have $\#r_0 > m$, but r_1 can be smaller. We may have to undo one or two of the divisions performed by the second HGCD-Q call, in order to return two remainders that satisfy the specification, see Section 6.3 for an example where this happens. For more details, we refer to [11].

4 Binary recursive gcd

Binary gcd works from the least significant end. Instead of the division used in Euclid's algorithm, we use binary division, denoted $\operatorname{bdiv}(a, b)$ and defined as follows. Let a, b be two integers, a odd and b even and non-zero. Let $k = v(b)$, so that $b' = 2^{-k}b$ is odd. The quotient $q = -a(b')^{-1} \pmod{2^{k+1}}$ satisfies

$$r = a + qb' = 0 \pmod{2^{k+1}}$$

and q is uniquely determined modulo 2^{k+1}. Since a is odd, q is odd too, and can be chosen in the interval $|q| < 2^k$. Then $2^{-k}|q| < 1$, so that

$$|r| = |a + q2^{-k}b| \leq |a| + 2^{-k}|q||b| < |a| + |b| \quad (1)$$

It's also clear that q depends only on the $k+1$ least significant bits of a and b', or on the $2k + 1$ least significant bits of a and b.

To apply this to gcd computation, first note that $\gcd(b, r) = 2^k \gcd(a, b)$. Consider the algorithm in Figure 2. Why does this algorithm terminate? After n rounds through the loop, $j \geq n$, so that 2^n divides x and y. This implies that as long as $y \neq 0$, we must have $|y| \geq 2^n$. From this and Equation 1, we get

$$2^n \leq |y| \leq F_{n+2} \max(|a|, |b|)$$

HGCD-Q(A, B)

$n \leftarrow \#A, m \leftarrow \lceil n/2 \rceil$
if $\#B \leq m$
  then return FAILURE
$n_1 \leftarrow n - m$
if $\#B > m + \lceil n_1/2 \rceil + 1$
  then
    Split: $A = 2^m a + A'$, $B = 2^m b + B'$
    $(\alpha, \beta, M) \leftarrow$ HGCD-Q(a, b)
    $(r_0; r_1) \leftarrow 2^m(\alpha; \beta) + M(A'; B')$
  else $\triangleright$ Or if the HGCD-Q call failed
    $(r_0, r_1) \leftarrow (A, B), M \leftarrow I$
while $\#r_0 > m + \lceil n_1/2 \rceil + 1$ and $\#r_1 > m$
  do
    $q \leftarrow \lfloor r_0/r_1 \rfloor$
    $(r_0, r_1) \leftarrow (r_1, r_0 - qr_1)$
    Update M accordingly.
$p \leftarrow 2m - \#r_0, n_2 \leftarrow \#r_0 - p$
if $\#r_1 > p + \lceil n_2/2 \rceil + 1$
  then
    Split: $r_0 = 2^p a + A'$, $r_1 = 2^p b + B'$
    $(\alpha, \beta, M') \leftarrow$ HGCD-Q(a, b)
    $(r_0; r_1) \leftarrow 2^p(\alpha; \beta) + M'(A'; B')$
    $M \leftarrow M' \cdot M$
if $\#r_1 < m$
  then
    if $v_0 = 0$
      then return FAILURE
    Pop last quotient q; update M
    $(r_0, r_1, r_2) \leftarrow (r_1 + qr_0, r_0, r_1)$
  else
    $q \leftarrow \lfloor r_0/r_1 \rfloor, r_2 \leftarrow r_0 - qr_1$
if $\#r_2 < m$
  then
    if $v_0 = 0$
      then return FAILURE
    Pop last quotient q; update M
    return $r_1 + qr_0, r_0, M$
$q \leftarrow \lfloor r_1/r_2 \rfloor, r_3 \leftarrow r_1 - qr_2$
while $\#r_3 > m$
  do
    Update M
    $q \leftarrow \lfloor r_2/r_3 \rfloor$
    $(r_0, r_1, r_2, r_3) \leftarrow (r_1, r_2, r_3, r_2 - qr_3)$
return r_0, r_1, M

Figure 1: Classical Schönhage gcd

BINARY-GCD(a, b)

$j \leftarrow 0$
$(x, y) \leftarrow (a, b)$
while $y \neq 0$
  do
    $k \leftarrow v(y) - j$
    $q \leftarrow \mathrm{bdiv}(2^{-j}x, 2^{-j}y)$
    $(x, y) \leftarrow (y, x + q2^{-k}y)$
    $j \leftarrow j + k$
return $2^{-j}x$

Figure 2: Binary gcd

where F_n is the n'th Fibonacci number. These inequalities can hold for only finitely many n, hence the algorithm must terminate with $y = 0$ after a finite number of steps. The use of quotients in the *symmetric* interval $-2^k < q < 2^k$ is essential; if one tries to use positive quotients $0 < q < 2^{k+1}$, the algorithm no longer terminates.

The binary recursive algorithm by Stehlé and Zimmermann uses a function HGCD-B defined as follows. The input is a bit size ℓ and two numbers $0 \leq a, b < 2^\ell$, with a even and b odd. Let $k = \lfloor (\ell - 1)/2 \rfloor$, and consider the remainder sequence r_i formed by repeated use of bdiv. For some i, we have $v(r_i) < k \leq v(r_{i+1})$. Then HGCD-B returns $j = v(r_i)$, $\alpha = 2^{-j} r_i$, $\beta = 2^{-j} r_{i+1}$, and a matrix M such that

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = 2^{-2j} M \begin{pmatrix} a \\ b \end{pmatrix}$$

We have $j < k$ and $j + v(\beta) \geq k$, and the size of the output is bounded by

$$\#M \leq \left\lfloor \frac{11(j+1)}{8} \right\rfloor \leq \left\lfloor \frac{11k}{8} \right\rfloor$$

$$\#(\alpha, \beta) \leq \begin{cases} \ell - \lfloor \frac{5j-12}{8} \rfloor & j > 2 \\ \ell & j = 1, 2 \end{cases}$$

These bounds are related to powers of the matrix $(0, 2; 2, 1)$, with eigenvalues $(1 \pm \sqrt{17})/2$, with $\log_2[(1 + \sqrt{17})/2] < 11/8$. The HGCD-B function doesn't quite reduce the input numbers a and b to numbers α and β of half the size, but the reduction is sufficient for GCD to terminate in subquadratic time.

The choice of k implies $2k+1 \le \ell$, which guarantees that the quotients computed by HGCD-B are valid for any numbers for which a and b are the ℓ least significant bits. We can now describe Stehlé's and Zimmermann's binary recursive algorithm.

HGCD-B(A, B, ℓ)

$k \leftarrow \lfloor (\ell - 1)/2 \rfloor$
if $v(B) \ge k$
  then return $0, A, B, I$
$\ell_1 = k + 1$
Split: $A = 2^{\ell_1} A' + a,\ B = 2^{\ell_1} B' + b$
$(j_1, \alpha, \beta, M) \leftarrow$ HGCD-B(a, b, ℓ_1)
$(A; B) \leftarrow (\alpha, \beta) + 2^{\ell_1 - 2j_1} M(A'; B')$
$v_1 \leftarrow v(B)$
if $j_1 + v_1 \ge k$
  then return j_1, A, B, M
$q \leftarrow \mathrm{bdiv}(A, B)$
$(A, B) \leftarrow (B, A + qB)$
$M \leftarrow (0, 2^{v_1}; 2^{v_1}, q) \cdot M$
if $j_1 + v_1 + v(B) \ge k$
  then return j_1, A, B, M
$\ell_2 \leftarrow 2(k - j_1 - v_1) + 1$
Split: $A = 2^{\ell_2} A' + a,\ B = 2^{\ell_2} B' + b$
$(j_2, \alpha, \beta, M') \leftarrow$ HGCD-B(a, b, ℓ_2)
$(A; B) \leftarrow (\alpha, \beta) + 2^{\ell_2 - 2j_2} M'(A'; B')$
$M \leftarrow M' \cdot M$
return $j_1 + v_1 + j_2, A, B, M$

For analysis and further details, see [9].

5 Schönhage's 1987 algorithm

Let a and b be two positive integers. Euclid's algorithm repeatedly replaces the larger number by the remainder when dividing the larger number by the smaller. We can decompose these steps further: Repeatedly replace the larger number by the difference between the numbers.

The Euclidean step corresponds to multiplying the vector $(a, b)^T$ by the one of the matrices

$$\begin{pmatrix} 1 & -q \\ 0 & 1 \end{pmatrix} \text{ or } \begin{pmatrix} 1 & 0 \\ -q & 1 \end{pmatrix}$$

depending on which of a and b is largest. The decomposition of the division step as q subtraction steps corresponds to the q'th power $(1, -q; 0, 1) = (1, -1; 0, 1)^q$.

Note that these matrices all have determinant 1, hence they have inverses with all-integer elements. The correctness proof for Euclid's algorithm can be generalized to the following important observation:

Lemma 3 *If a, b, α, and β are integers, M is a 2×2 matrix, $\det M = 1$, and*

$$\begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{2}$$

then $\gcd(a, b) = \gcd(\alpha, \beta)$.

Proof: Equation 2 directly implies that $\gcd(\alpha, \beta)$ divides both a and b, and hence also $\gcd(a, b)$. By multiplying with M^{-1} we get the other direction; $\gcd(a, b)$ divides $\gcd(\alpha, \beta)$. □

We also need a bound for the matrix elements.

Lemma 4 *If $a, b, \alpha, \beta > 0$, $\#(a, b) \le n$, $\underline{\#}(\alpha, \beta) > s$, and Equation 2 holds with some $M \ge 0$, then $\#M \le n - s$. In fact, if $M = (u, u'; v, v')$, we also have $\#(u + u', v + v') \le n - s$.*

Proof: From the first row of Equation 2 we get

$$2^n > a = u\alpha + u'\beta \ge u2^s + u'2^s = 2^s(u + u')$$

and the result follows by dividing both sides by 2^s. The same argument applies to the second row. □

During the algorithm, we need a division operation $\mathrm{sdiv}(a, b, s)$ that never returns a too small "remainder". Input is a bit size s and two positive integers a and b with $\underline{\#}(a, b) > s$. Then sdiv produces the largest q such that $\#(a - qb) > s$. It can be implemented using standard division as follows: First compute $q' = \lfloor a/b \rfloor$. If $\#(a - q'b) > s$, set $q = q'$, otherwise, set $q = q' - 1$.

After these preliminaries, we describe a generalized gcd function.

5.1 The SGCD function

Lemma 5 *Given a bit size $s \ge 0$ and two positive integers a and b with $\underline{\#}(a, b) > s$, there exists integers α, β, and a matrix M such that*

$$\begin{aligned} &M \ge 0 && \alpha, \beta > 0 \\ &\begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} \alpha \\ \beta \end{pmatrix} && \underline{\#}(\alpha, \beta) > s \\ &\det M = 1 && \#(\alpha - \beta) \le s \end{aligned} \tag{3}$$

Existence follows from the following algorithm, using repeated division:

SGCD-SLOW(a, b, s)

$M \leftarrow I$
while $\#(a - b) > s$
  do
    if $a > b$
      then
        $(q, a) \leftarrow \text{sdiv}(a, b, s)$
        $M \leftarrow M \cdot (1, q; 0, 1)$
      else
        $(q, b) \leftarrow \text{sdiv}(b, a, s)$
        $M \leftarrow M \cdot (1, 0; q, 1)$
return a, b, M

This is essentially Euclid's algorithm, but with a different stop condition. In fact, α, β, and M are determined uniquely by a, b, s, and the conditions of Equation 3; this is not needed for the algorithm to work, but it is nevertheless proved in Appendix A.

By Lemma 3, $\gcd(a, b) = \gcd(\alpha, \beta)$, and by Lemma 4, $\#M \leq \#(a, b) - s$. The SGCD function is a generalization of extended gcd, since for $s = 0$, SGCD terminates with $\alpha = \beta = \gcd(a, b)$ and a matrix $M = (u, u'; v, v')$ such that $\gcd(a, b) = v'a - u'b = -va + ub$.

5.2 Subquadratic SGCD

The SGCD algorithm in Figure 3 is a step-by-step translation of Schönhage's algorithm for reduction of quadratic forms [7].

The correctness argument in [7] carries over directly, and will not be repeated here, except for explaining Step 8 which is the most subtle one.

Consider the case $p > 0$. Before the adjustment, we have $\underline{\#}(\alpha, \beta) > s', \#(\alpha - \beta) \leq s'$. We also have $p + s' = s + 1$ and a bound for the matrix elements, $\#M \leq (n + s - p) - s' = n - 1$. Let $M = (u, u'; v, v')$, then the new adjusted values for α and β are given by

$$2^p \begin{pmatrix} \alpha \\ \beta \end{pmatrix} + \begin{pmatrix} v' & -u' \\ -v & u \end{pmatrix} \begin{pmatrix} A' \\ B' \end{pmatrix}$$

Consider the first row. For the first term, we have $\#(2^p \alpha) > p + s' = s + 1$. For the second term, we have $\#(v'A' - u'B') \leq n - 1 + p = n - 1 + s - n + 1 = s$. Then

$$2^p \alpha + (v'A' - u'B') \geq 2^{s+1} - 2^s = 2^s$$

SGCD(a, b, s)

1 **if** $\underline{\#}(a, b) \leq s + 2$
  then $(\alpha, \beta) \leftarrow (a, b), M = I$; **goto** Step 9
2 $n \leftarrow \#(a, b) - s$ ▷ Number of bits to reduce
  if $s \leq n$
  then
    $p \leftarrow 0, s' \leftarrow s, \alpha \leftarrow a, \beta \leftarrow b$
  else
    $s' \leftarrow n, p \leftarrow s - n + 1$
    Split: $a = 2^p \alpha + A'$, $b = 2^p \beta + B'$
3 $h \leftarrow s' + \lfloor n/2 \rfloor$
  if $\underline{\#}(\alpha, \beta) \leq h$
  then $M \leftarrow I$; **goto** Step 5
4 $(\alpha, \beta, M) \leftarrow$ SGCD(α, β, h)
5 **while** $\#(\alpha, \beta) > h$
  do
    if $\#(\alpha - \beta) \leq s'$
      then goto Step 8.
    One sdiv step on (α, β); update M
6 $(\alpha, \beta, M') \leftarrow$ SGCD(α, β, s')
7 $M \leftarrow M \cdot M'$
8 **if** $p > 0$
  then $(\alpha; \beta) \leftarrow 2^p(\alpha; \beta) + M^{-1}(A'; B')$
9 **while** $\#(\alpha - \beta) > s$
  do
    One sdiv step on (α, β); update M
10 **return** α, β, M

Figure 3: The SGCD algorithm

A similar inequality holds for β.

We also need a upper bound for $\#(\alpha - \beta)$ after adjustment. Let $d, \#d \leq s'$, denote the difference $\alpha - \beta$ before the adjustment step. After the adjustment, we get

$$\begin{aligned} |\alpha - \beta| &\leq 2^p d + \left|(v + v')a_0 - (u + u')b_0\right| \\ &< 2^{p+s'} + 2^{n-1+p} = 2^{s+1} + 2^s < 2^{s+2} \end{aligned}$$

To summarize, after Step 8, we have $\underline{\#}(\alpha, \beta) \geq s, \#(\alpha - \beta) \leq s + 2$.

The analysis for the running time in [7] also applies directly also to SGCD. Let $T(n)$ denote the maximal running time for inputs with $s \leq 3n$ and $\#(a, b) \leq n + s$. Then the argument in [7] shows that $T(n) \leq c\mu(n) \log n$.

When using SGCD to compute plain gcd, without co-factors, one disadvantage of the SGCD algorithm as described above is that it computes the matrix M even though it is not needed. One important optimization is to note that if the caller doesn't need M, and $p = 0$, then it's not necessary to compute neither the product $M \cdot M'$ in Step 7, nor the two factors. Avoiding these unnecessary computations is essential for making the algorithm competitive, since it eliminates the final and largest matrix multiplication.

6 A new half-gcd function

In this section, we reorganize Schönhage's 1987 algorithm to put it into half-gcd form. There are two main changes: The responsibility for the adjustment, Step 8, is moved to the caller of the function, and the s parameter is not passed freely, but is determined by input size. This makes the algorithm more streamlined and easier to compare to the other half-gcd algorithms, and the elimination of s also makes the analysis of the running time more straight-forward.

6.1 The HGCD-D function

In short:

$$\text{HGCD-D}(a, b) = \text{SGCD}(a, b, \lfloor \#(a, b)/2 \rfloor + 1)$$

The input to the function is two integers $a, b > 0$. Put $n = \#(a, b)$ and $s = \lfloor n/2 \rfloor + 1$, and assume $\underline{\#}(a, b) > s$. Then HGCD returns two numbers $\alpha, \beta > 0$, with $\underline{\#}(\alpha, \beta) > s$ but $\#(\alpha - \beta) \le s$, and a matrix $M \ge 0$ such that

$$\begin{pmatrix} a \\ b \end{pmatrix} = M \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \qquad \det M = 1$$

The bound in Lemma 4 implies that $\#M \le n - s < n/2$. This choice of s is just sufficient to make it useful to apply HGCD-D to the most significant part of two larger numbers. The following lemma plays the role of the adjustment Step 8 in Section 5.2.

Lemma 6 *Let A and B be two given numbers of size $N = \#(A, B)$, assume $0 < p < N$. Partition A and B into the least significant p bits A' and B' and the $n = N - p$ most significant bits a and b, so that $A = 2^p a + A'$ and $B = 2^p b + B'$. Also assume that $\underline{\#}(a, b) > \lfloor n/2 \rfloor + 1$, so that HGCD$(a, b)$ is well defined. Let $(c, d, M) =$ HGCD(a, b), and form*

$$\begin{pmatrix} C \\ D \end{pmatrix} = M^{-1} \begin{pmatrix} A \\ B \end{pmatrix} = 2^p \begin{pmatrix} a \\ b \end{pmatrix} + M^{-1} \begin{pmatrix} A' \\ B' \end{pmatrix} \qquad (4)$$

Then $\underline{\#}(C, D) > p + \lfloor n/2 \rfloor$ and $\#(C - D) \le p + \lfloor n/2 \rfloor + 2$.

Proof: From the definition of HGCD, we have $\#M \le n - s$, $\underline{\#}(c, d) > s$, and $\#(c - d) \le s$, where $s = \lfloor n/2 \rfloor + 1$. Note that $s > n/2 > n - s$. Let $M^{-1} = (v', -u'; -v, u)$. In Equation 4 we have

$$\begin{aligned} C &= 2^p c + v'A' - u'B' \\ &> 2^p 2^s - 2^{n-s} 2^p \ge 2^p 2^{s-1} = 2^{p + \lfloor n/2 \rfloor} \end{aligned}$$

Then $\#C > p + \lfloor n/2 \rfloor$ as claimed, and the same holds for D. Next,

$$\begin{aligned} |C - D| &= |2^p(c - d) + (v + v')A' - (u + u')B')| \\ &< 2^p 2^s + 2^{n-s} 2^p \le 2^{p+s+1} = 2^{p + \lfloor n/2 \rfloor + 2} \end{aligned}$$

□

6.2 Subquadratic HGCD-D

The recursive algorithm in Figure 4 computes HGCD-D in subquadratic time. The following lemma explains the most essential details:

Lemma 7 *In the two recursive calls in Step 6 and 17 of the HGCD-D algorithm, we have*

$$\#(a, b) \le \lceil N/2 \rceil$$

The loops at Step 9 and Step 20 are executed at most 4 times each.

Proof: Consider the two recursive calls in turn. For the first call in Step 6, it is clear that $n_1 = \lceil N/2 \rceil$. We have

$$p_1 + \lfloor n_1/2 \rfloor = \lfloor N/2 \rfloor + \lfloor (N + 1)/4 \rfloor = \lfloor 3N/4 \rfloor$$

and $\underline{\#}(a, b) = \underline{\#}(A, B) - p_1 > \lfloor n_1/2 \rfloor + 2$, so that HGCD-D$(a, b)$ is well defined. By Lemma 6, the new A and B after the call satisfy $\#(A - B) \le \lfloor 3N/4 \rfloor + 2$, and after at most two sdiv steps, $\#(A, B) \le \lfloor 3N/4 \rfloor + 2$. Then at most two additional steps are needed to reduce the size one more bit (two Euclid steps using standard division reduces the size by

HGCD-D(A, B)

1 $N \leftarrow \#(A,B), S \leftarrow \lfloor N/2 \rfloor + 1.$
2 **if** $\underline{\#}(A,B) > \lfloor 3N/4 \rfloor + 2$
3   **then**
4         $p_1 \leftarrow \lfloor N/2 \rfloor, n_1 \leftarrow N - p_1 = \lceil N/2 \rceil$
5   Split: $A = 2^{p_1} a + A'$, $B = 2^{p_1} b + B'$
6         $(\alpha, \beta, M) \leftarrow$ HGCD-D(a, b)
7         $(A; B) \leftarrow 2^{p_1}(\alpha; \beta) + M^{-1}(A'; B').$
8   **else** $M \leftarrow I$
9 **while** $\#(A,B) > \lfloor 3N/4 \rfloor + 1$
    and $\#(A - B) > S$
10   **do**
11   One sdiv step on (α, β); update M
12 **if** $\underline{\#}(A,B) > S + 2$
13   **then**
14         $N_2 \leftarrow \#(A,B)$
15         $p_2 \leftarrow 2S - N_2 + 1, n_2 \leftarrow N_2 - p_2$
16   Split: $A = 2^{p_2} a + A'$, $B = 2^{p_2} b + B'$
17         $(\alpha, \beta, M') \leftarrow$ HGCD-D(a, b)
18         $(A; B) \leftarrow 2^{p_2}(\alpha; \beta) + M'^{-1}(A'; B')$
19         $M \leftarrow M \cdot M'.$
20 **while** $\#(A - B) > S$
21   **do**
22   One sdiv step on (α, β); update M
23 **return** A, B, M

Figure 4: The HGCD-D algorithm.

at least one bit. The remainders produced by sdiv differ from the Euclid remainders only when one of the Euclid remainders is of size S or smaller, and in this case, the HGCD-D algorithm terminates with no further steps of any kind).

For the second recursive call, Step 17, first note that $n_2 = 2(N_2 - S) - 1$. The bound $N_2 \le \lfloor 3N/4 \rfloor + 1$ implies $n_2 = 2(N_2 - S) - 1 \le 2(\lfloor 3N/4 \rfloor - \lfloor N/2 \rfloor) - 1 \le 2\lfloor (N+1)/4 \rfloor - 1 \le \lceil N/2 \rceil - 1$.

Next, $\lfloor n_2/2 \rfloor = N_2 - S - 1$, and since $\underline{\#}(a,b) > S + 2 - p_2 = N_2 - S + 1 = \lfloor n_2/2 \rfloor + 2$, HGCD-D$(a,b)$ is well defined. We also get

$$p_2 + \lfloor n_2/2 \rfloor = (2S - N_2 + 1) + (N_2 - S - 1) = S$$

Then Lemma 6 implies that after the call, the new A and B satisfy $\#(A - B) \le S + 2$. It then takes at most four sdiv steps until the algorithm terminates with $\#(A - B) \le S$. □

Lemma 8 *The above algorithm runs in time $T(n) = c\mu(n) \log n$, where $\mu(n)$ is the time for one multiplication or division of n-bit numbers. With* FFT*-based multiplication, $T(n) = O(n(\log n)^2 \log\log n)$.*

Proof: Let $T(n)$ denote the maximum running time of the HGCD-D algorithm, with input a and b of size n. Then $T(n) \le 2T(\lceil n/2 \rceil) + c\mu(n)$, which implies $T(n) \le c\mu(n) \log n$. □

So all the considered subquadratic gcd algorithms have the same asymptotic running time, but they may differ by constant factors.

6.3 About the need for backup steps

The difference between HGCD-Q and HGCD-D, i.e. between Schönhage's 1971 algorithm, and the new algorithm, is quite subtle. The difference is in the definition of a correct reduction, and in the correctness and stop conditions that follow from that.

The correctness of HGCD-Q depends on the quotient sequence being correct. In fact, the HGCD-Q algorithm is expected to produce a quotient sequence that is the prefix of the continued fraction expansion for *any* x in the interval $A/(B+1) < x < (A+1)/B$. It seems that to achieve this, the backup steps are essential.

Let's consider an example where HGCD-Q without backup logic returns an incorrect result: $A = 858824$ and $B = 528747$. Then $n = 20, m = 10, \gcd(A, B) = 1$ and the quotient sequence is $(1, 1, 1, 1, 1, 1, 20, 1, 1, 3, 3, 5, 8, 3)$. The HGCD-Q algorithm reduces these numbers as follows:

Input	Output
$858824, 528747$	
First rec. call	$128, 66$
$838, 516$	$(2, -3; -3, 5)$
Loop	$64144, 3119$
$131407, 67263$	$(5, -1; -8, 13)$
Second rec. call	$129, 65$
$4009, 194$	$(1, -20; -1, 21)$
Without backup	$1764, 1355$
	$(165, -268; -173, 281)$

The final remainders, $r_0 = 1764$ and $r_1 = 1355$, violates both versions of Jebelean's criterion, since $r_0 - r_1 = 409 < 2^m$, but $v_2 - v_1 = 549$. For any $p \ge 3$, let $A' = 1$ and $B' = 2^p - 1$; then the numbers $2^p A + A'$ and $2^p B + B'$ have a quotient sequence that starts with $(1,1,1,1,1,1,20,2)$, for example, $(8A+1, 8B+7) = (6870593, 4229983)$ has the quotient sequence $(1,1,1,1,1,1,20,2,53,4,2,12,29)$. This shows that the backup steps in the HGCD-Q algorithm are essential for correctness; the final quotient must be discarded.

The stop and correctness conditions for HGCD-Q are related to Jebelean's criterion. We can only be sure that a quotient q_k is correct if the remainder after the remainder after the *next* division step, $r_{k+2} = r_k - q_{k+1} r_{k+1}$, is large. And the stop condition checks if the remainder after yet another division, $r_{k+3} = r_{k+1} - q_{k+2} r_{k+2}$, is small.

The new algorithm, HGCD-D, uses a relaxed notion of a correct reduction. It doesn't pay attention to the quotient sequence, instead its correctness is based on the more general Lemma 3. This relaxation makes it possible to use a simpler correctness and stop conditions: A co-factor matrix is correct if the two corresponding remainders r_k and r_{k+1} are large, and the stop condition checks if $|r_k - r_{k+1}|$ is small.

7 Evaluation

The gcd algorithms were implemented in the `mpn` layer in GMP [1], and compared with respect to running time and implementation complexity. We compare the following algorithms:

`mpn_gcd`: Ken Weber's accelerated gcd algorithm, shipped in GMP-4.1.4, falling back to Lehmer's algorithm for small inputs.[2]

`mpn_rgcd`: Schönhage's 1971 algorithm, as described in Section 3.

`mpn_bgcd`: Stehlé's and Zimmermann's recursive binary GCD algorithm, as described in 4.

`mpn_sgcd`: Schönhage's 1987 algorithm, as described in Section 5. The implementation avoids computing unneeded matrices.

`mpn_ngcd`: GCD based on the HGCD-D algorithm, as described in Section 6.

[2] GMP-4.1.4 uses the binary algorithm for small inputs, but it turns out that Lehmer's algorithm is faster for numbers up to a few dozen words, so that is what is used in this evaluation.

Algorithm	Thresholds	
	HGCD	GCD
`mpn_rgcd`	191	951
`mpn_bgcd`	133	1015
`mpn_sgcd`	95	1193
`mpn_ngcd`	159	866

Table 1: Threshold values used for the various algorithms, tuned for an AMD Duron.

As far as possible, the implementation avoids splitting numbers at arbitrary bit boundaries, and instead work with word boundaries.

7.1 Threshold values

All the subquadratic algorithms use quadratic base cases in two ways, in the half-gcd and full-gcd functions. The use of GCD-THRESHOLD has already been described in Section 2. For the HGCD-style functions, when the input size is below HGCD-THRESHOLD, the functions stop chopping numbers in half, and instead repeatedly chop off two words at a time, using a HGCD function specialized for input size of two words, and producing a transformation matrix with single word elements. For the left-to-right algorithms, this is analogous to Lehmer's algorithm. The input size and the thresholds are expressed in number of words, where a word is 32 bits on the architecture used for the evaluation.

The optimal threshold values depend on the architecture. Table 1 gives the thresholds used on an AMD Duron processor. These values are tuned automatically to get close to optimal, except the HGCD-THRESHOLD for `mpn_sgcd`. Due to the structure of the code, it is reasonable to set this value to half of the corresponding threshold for `mpn_rgcd`.

7.2 Performance

Figure 5 shows the running time for inputs ranging from 10 words (320 bits) to 110000 words (3400000 bits), evaluated on an 1.4 GHz AMD Duron (a fairly low-end PC). On this logarithmic scale, all the subquadratic algorithms are very close, and they start to pay off for sizes around 1000 words and larger; at 10000 words, the subquadratic algorithms are about twice as fast as Weber's accelerated GCD algorithm, and for 100000 words they are about 30 times faster.

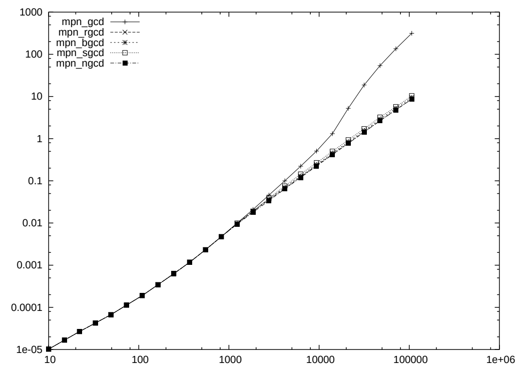

Figure 5: Running for large inputs on AMD Duron. Both sizes (in words) and running times (in second) are shown in logarithmic scale. The mpn_gcd function is much slower than the subquadratic algorithms, which all look very similar.

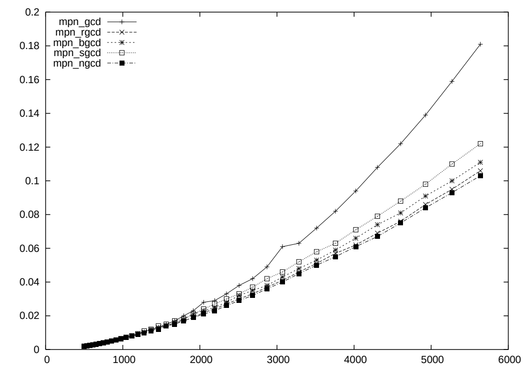

Figure 6: Running time for medium size input, on an AMD Duron. mpn_gcd is slowest, as expected, followed by mpn_sgcd, and mpn_bgcd. The new algorithm, mpn_ngcd, is fastest, but with a very small margin to mpn_rgcd, which is Schönhage's 1971 algorithm.

The running time for the different subquadratic algorithms differ by constant factors, which can be seen if we focus on input sizes of a few thousand words. The running times on the same processor are shown in Figures 6, using linear scale for both input size and running time.

We see the quadratic behavior of the mpn_gcd function, as expected. The binary algorithm is 5–10% slower than Schönhage's 1971 algorithm. It's often hard to explain performance differences between algorithms that have the same asymptotic running time and differ only by a constant factor. The binary algorithm is simpler in structure, but works with slightly larger numbers. Where one application of the HGCD-Q function in Schönhage's algorithm reduces n-bit numbers to size close to $n/2$ bits, one application of of the HGCD-B function in the binary algorithm can produce numbers of size close to $11n/16$. Other factors that need investigation is the average number of quotients, and the average number of quotients that can be represented by a single word matrix M.

Also seen in Figure 6, implementing Schönhage's 1987 algorithm as described in Section 5 results in a subquadratic algorithm that is slower than both the binary recursive algorithm and Schönhage's 1971 algorithm. However, when reorganized into half-gcd form, performance is significantly improved, and the algorithm mpn_ngcd is slightly faster than mpn_rgcd.

Algorithm	# lines	Cyclomatic complexity
mpn_rgcd	1967	292
mpn_bgcd	1348	206
mpn_sgcd	760	144
mpn_ngcd	733	133

Table 2: Number of non-comment lines of code, and cyclomatic complexity, for the subquadratic GCD-algorithms.

7.3 Complexity

Of the compared algorithms, Schönhage's 1971 algorithm (mpn_rgcd) seems to be the most complex. Comparisons using popular code complexity measures, such as number of lines and McCabe's cyclomatic complexity, support this impression. See Table 2 for a detailed comparison. The implementation of Schönhage's 1971 algorithm, mpn_rgcd, is more than twice as large as the new mpn_ngcd algorithm.

When reading the line counts, one should keep in mind that the implementation environment is quite low-level. The code uses library functions for basic arithmetic on large non-negative numbers, but both signs and storage sizes have to be kept track of explicitly.

In this comparison, the binary recursive algorithm, mpn_bgcd, turns out to be more complex than `mpn_ngcd`. This is a little surprising, since the algorithm as described in Section 4 is so straightforward. Some factors that contribute to the measured implementation complexity of the binary recursive algorithm are:

- The bdiv function is not part of the standard arithmetic library.
- It's essential to the binary recursive algorithm to use signed quotients and matrix elements.
- Bit shift logic in the adjustment step, depending on the j returned by the recursive HGCD-B calls.

8 Conclusions

We describe a new subquadratic GCD algorithm based on Schönhage's work in the late 1980's. This is shown to have the same asymptotic complexity as both Schönhage's algorithm from 1971, and the binary recursive algorithm by Stehlé and Zimmermann. The new algorithm is slightly faster and much simpler to implement. Hopefully, this can open the door for wider use of subquadratic gcd in practice.

Acknowledgements

The author wishes to thank Bradley Lucier, for pointing out the importance of Schönhage's 1991 paper, and for a helpful discussions on how to implement and optimize it, Torbjörn Granlund, for writing GMP, and for his zealous testing, and Damien Stehlé for helping me understand the binary recursive algorithm, and for many helpful comments on during the work with this paper.

References

[1] Torbjörn Granlund. GNU multiple precision arithmetic library, version 4.1.4, September 2004. `http://www.swox.se/gmp`.

[2] Tudor Jebelean. A double-digit Lehmer-Euclid algorithm for finding the GCD of long integers. *Journal of Symbolic Computation*, 19(1-3):145–157, 1995.

[3] Donald E. Knuth. The analysis of algorithms. *Actes du Congrés International des Mathématiciens*, pages 269–274, 1970.

[4] D. H. Lehmer. Euclid's algorithm for large numbers. *American Mathematical Monthly*, 45:227–233, 1938.

[5] Victor Y. Pan and Xinmao Wang. Acceleration of euclidean algorithm and extensions. In *ISSAC '02: Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 207–213, New York, NY, USA, 2002. ACM Press.

[6] Arnold Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.

[7] Arnold Schönhage. Fast reduction and composition of binary quadratic forms. In S. M. Watt, editor, *Proc. of Intern. Symp. on Symbolic and Algebraic Computation*, pages 128–133, Bonn, 1991. ACM Press.

[8] Jonathan P. Sorenson. Two fast GCD algorithms. *Journal of Algorithms*, 16(1):110–144, January 1994.

[9] Damien Stehlé and Paul Zimmermann. A binary recursive GCD algorithm. In D. Buell, editor, *ANTS-VI*, volume 3076 of *LCNS*, Burlington, June 2004. Springer-Verlag.

[10] J. Stein. Computational problems associated with Racah algebra. *Journal of Computational Physics*, 1(3):397–405, February 1967.

[11] Klaus Thull and Chee K. Yap. A unified approach to HGCD algorithms for polynomials and integers, 1990. Manuscript. Available from `http://cs.nyu.edu/cs/faculty/yap/papers`.

[12] Ken Weber. The accelerated integer GCD algorithm. *ACM Transactions on Mathematical Software*, 21:111–122, March 1995.

[13] André Weilert. Asymptotically fast GCD computation in $z[i]$. In *ANTS-IV: Proc. of 4th Intern. Symp. Algorithmic Number Theory*, number 1838 in LNCS, pages 595–613, Leiden, July 2000. Springer-Verlag. See page 602, in particular.

A Uniqueness of M

To prove that α, β, and M, the output of $\textsc{sgcd}(a, b, s)$, are uniquely determined by a, b, and s, we first show that the matrix M has a particular factorization.

Lemma 9 *If $M \geq 0$ and $\det M = 1$, then M can be factorized as a product where each factor is either $(1, 1; 0, 1)$ or $(1, 0; 1, 1)$.*

Proof: We use induction over the sum of matrix elements. Let $M = (u, u'; v, v')$ and $n = u+u'+v+v'$. If $n = 2$, then the identity matrix is the only one that has unit determinant and non-negative elements, and it has a trivial factorization.

Make the induction assumption that all matrices with element sum smaller than n have a factorization of the required form.

For $n > 2$, we have a few different cases, depending on the relative order of the matrix elements u, u', v, and v'. If one column is elementwise smaller than the other, e.g. $u \geq u'$ and $v \geq v'$, then

$$M = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u - u' & u' \\ v - v' & v' \end{pmatrix}$$

where the matrix on the right is non-negative, has unit determinant, and a reduced element sum $n - u' - v' < n$. Hence, by induction, it has a factorization of the required form (note that $\det M = 1$ implies that at least one of u' and v' is non-zero). Similarly, if $u \leq u'$ and $v \leq v'$, we can take out the factor $(1, 0; 1, 1)$.

So which cases remain? If neither column is elementwise smaller than the other, the two strictly largest elements must be on one of the diagonals. If $\min(u, v') > \max(u', v)$, then $1 = \det M = uv' - u'v \geq (\min(u, v'))^2 - (\max(u', v))^2$ which implies that $u = v' = 1$ and $u' = v = 0$, i.e. M is the identity matrix, contradicting $n > 2$.

If $\max(u, v') < \min(u', v)$, then $1 = \det M = uv' - u'v \leq (\max(u, v'))^2 - (\min(u', v))^2 < 0$ which also is a contradiction. □

We can now prove that α, β, and M are uniquely determined by a, b, s, and the conditions of Equation 3. This result follows from the following lemma, specialized to $k = 2^s$.

Lemma 10 *If $k > 0$ and $a, b \geq k$ are given, then α, β, and M are uniquely determined by the requirements $(a; b) = M(\alpha; \beta)$, $M \geq 0$, $\det M = 1$, $\alpha \geq k$, $\beta \geq k$, and $|\alpha - \beta| < k$.*

Proof: From the previous lemma, we know that either $M = I$, $M = (1, 1; 0, 1)M'$, or $M = (1, 0; 1, 1)M'$, where $M' \geq 0$ and $\det M' = 1$. When $M \neq I$, let $(a'; b') = M'(\alpha; \beta)$, then $a', b' \geq k$ and either $(a', b') = (a - b, b)$ or $(a', b') = (a, b - a)$.

We use induction on $n = \max(a, b)$.

If $|a - b| < k$, in particular if $a = b$, then we must have $M = I$, since otherwise, either a' or b' equals $|a - b| < k$, which contradicts $a', b' \geq k$.

If $|a - b| \geq k$, then we must have $M \neq I$. If $a > b$, then we must have $(a'; b') = (a - b; b)$, so that $\max(a', b') < a = n$. By induction, α, β, and M' are uniquely determined by a', b', and k, and then $M = (1, 1; 0, 1)M'$. The case $a < b$ is similar. □