A Binary Recursive Gcd Algorithm

Damien Stehlé and Paul Zimmermann

LORIA/INRIA Lorraine, 615 rue du jardin botanique, BP 101, F-54602
Villers-lès-Nancy, France, {stehle,zimmerma}@loria.fr

Abstract. The binary algorithm is a variant of the Euclidean algorithm that performs well in practice. We present a quasi-linear time recursive algorithm that computes the greatest common divisor of two integers by simulating a slightly modified version of the binary algorithm. The structure of our algorithm is very close to the one of the well-known Knuth-Schönhage fast gcd algorithm; although it does not improve on its $O(M(n) \log n)$ complexity, the description and the proof of correctness are significantly simpler in our case. This leads to a simplification of the implementation and to better running times.

1 Introduction

Gcd computation is a central task in computer algebra, in particular when computing over rational numbers or over modular integers. The well-known Euclidean algorithm solves this problem in time quadratic in the size n of the inputs. This algorithm has been extensively studied and analyzed over the past decades. We refer to the very complete average complexity analysis of Vallée for a large family of gcd algorithms, see [10]. The first quasi-linear algorithm for the integer gcd was proposed by Knuth in 1970, see [4]: he showed how to calculate the gcd of two n-bit integers in time $O(n \log^5 n \log \log n)$. The complexity of this algorithm was improved by Schönhage [6] to $O(n \log^2 n \log \log n)$. A comprehensive description of the Knuth-Schönhage algorithm can be found in [12]. The correctness of this algorithm is quite hard to establish, essentially because of the technical details around the so-called "fix-up procedure", and a formal proof is by far out of reach. As an example, several mistakes can be noticed in the proof of [12] and can be found at `http://www.cs.nyu.edu/cs/faculty/yap/book/errata.html`. This "fix-up procedure" is a tedious case analysis and is quite difficult to implement. This usually makes the implementations of this algorithm uninteresting except for very large numbers (of length significantly higher than 10^5 bits).

In this paper, we present a variant of the Knuth-Schönhage algorithm that does not have the "fix-up procedure" drawback. To achieve this, we introduce a new division (GB for Generalized Binary), which can be seen as a natural generalization of the binary division and which has some natural meaning over the 2-adic integers. It does not seem possible to use the binary division itself in a Knuth-Schönhage-like algorithm, because its definition is asymmetric: it eliminates least significant bits but it also considers most significant bits to perform comparisons. There is no such asymmetry in the GB division. The recursive GB Euclidean algorithm is much simpler to describe and to prove than the Knuth-Schönhage algorithm, while admitting the same asymptotic complexity.

This simplification of the description turns out to be an important advantage in practice: we implemented the algorithm in GNU MP, and it ran between three and four times faster than the implementations of the Knuth-Schönhage algorithm in Magma and Mathematica.

The rest of the paper is organized as follows. In §2 we introduce the GB division and give some of its basic properties. In §3, we describe precisely the new recursive algorithm. We prove its correctness in §4 and analyze its complexity in §5. Some implementation issues are discussed in §6.

Notations: The standard notation $O(.)$ is used. The complexity is measured in elementary operations on bits. Unless specified explicitly, all the logarithms are taken in base 2. If a is a non-zero integer, $\ell(a)$ denotes the length of the binary representation of a, i.e. $\ell(a) = \lfloor \log |a| \rfloor + 1$; $\nu_2(a)$ denotes the 2-adic valuation of a, i.e. the number of consecutive zeroes in the least significant bits of the binary representation of a; by definition, $\nu_2(0) = \infty$. $r := a \text{ cmod } b$ denotes the centered remainder of a modulo b, i.e. $a = r \bmod b$ and $-\frac{b}{2} < r \leq \frac{b}{2}$. We recall that $M(n) = \Theta(n \log n \log \log n)$ is the asymptotic time required to multiply two n-bit integers with Schönhage-Strassen multiplication [7]. We assume the reader is familiar with basic arithmetic operations such as fast multiplication and fast division based on Newton's iteration. We refer to [11] for a complete description of these algorithms.

2 The Generalized Binary Division

In this section we first recall the binary algorithm. Then we define the generalized binary division — GB division for short — and give some basic properties about it. Subsection 2.4 explains how to compute modular inverses from the output of the Euclidean algorithm based on the GB division.

2.1 The Binary Euclidean Algorithm

The binary division is based on the following properties: $\gcd(2a, 2b) = 2\gcd(a, b)$, $\gcd(2a+1, 2b) = \gcd(2a+1, b)$, and $\gcd(2a+1, 2b+1) = \gcd(2b+1, a-b)$. It consists of eliminating the least significant bit at each loop iteration. Fig. 1 is a description of the binary algorithm. The behavior of this algorithm is very well understood (see [1] and the references there). Although it is still quadratic in the size of the inputs, there is a significant gain over the usual Euclidean algorithm, in particular because there is no need to compute any quotient.

2.2 The Generalized Binary Division

In the case of the standard Euclidean division of a by b with $|a| > |b|$, one computes a quotient q such that when qb is added to a, the obtained remainder

```
Algorithm Binary-Gcd.
Input: $a, b \in \mathbb{Z}$.
Output: $\gcd(a, b)$.

 1. If $|b| > |a|$, return Binary-Gcd$(b, a)$.
 2. If $b = 0$, return $a$.
 3. If $a$ and $b$ are both even then return $2 \cdot$ Binary-Gcd$(a/2, b/2)$.
 4. If $a$ is even and $b$ is odd then return Binary-Gcd$(a/2, b)$.
 5. If $a$ is odd and $b$ is even then return Binary-Gcd$(a, b/2)$.
 6. Otherwise return Binary-Gcd$((|a| - |b|)/2, b)$.
```

Fig. 1. The binary Euclidean algorithm.

is smaller than b. Roughly speaking, left shifts of b are subtracted from a as long as possible, that is to say until a has lost its $\ell(a) - \ell(b)$ most significant bits (approximately). The GB division is the dual: in order to GB-divide a by b with $|a|_2 > |b|_2$, where $|a|_2 := 2^{-\nu_2(a)}$ is the 2-adic norm of a, one computes a quotient $\frac{q}{2^k}$ such that when $\frac{q}{2^k}b$ is added to a, the obtained remainder is smaller than b for the 2-adic norm. Roughly speaking, right shifts of b are subtracted from a as long as possible, that is to say until a has lost its $\nu_2(b) - \nu_2(a)$ least significant bits.

Lemma 1 (GB Division). *Let a, b be non-zero integers with $\nu_2(a) < \nu_2(b)$. Then there exists a unique pair of integers (q, r) such that:*

$$r = a + q\frac{b}{2^{\nu_2(b)-\nu_2(a)}}, \qquad (1)$$

$$|q| < 2^{\nu_2(b)-\nu_2(a)}, \qquad (2)$$

$$\nu_2(r) > \nu_2(b). \qquad (3)$$

The integers q and r are called respectively the GB quotient and the GB remainder of (a, b). We define $GB(a, b)$ as the pair (q, r).

Proof. From (1), $q = -\frac{a}{2^{\nu_2(a)}} \cdot (\frac{b}{2^{\nu_2(b)}})^{-1} \bmod 2^{\nu_2(b)-\nu_2(a)+1}$. Since q is odd, the second condition is fulfilled and gives the uniqueness of q. As a consequence, r is uniquely defined by (1). Moreover, since $r = a + q\frac{b}{2^{\nu_2(b)-\nu_2(a)}}$, we have $r = 0 \bmod 2^{\nu_2(b)+1}$, which gives condition (3).

The GB division resembles Hensel's odd division, which was introduced by Hensel around 1900. A description can be found in [8]. For two integers a and b with b odd, it computes q and r in $\mathbb{Z}$, such that: $a = -bq + 2^p r$ and $r < 2b$, where $p = \ell(a) - \ell(b)$ is the difference between the bit lengths of a and b. In other words, Hensel's odd division computes $r := (2^{-p}a) \bmod b$, which may be found efficiently as shown in [5]. Besides, there is also some similarity with the PM algorithm of Brent and Kung [2]. When $\nu_2(a) = 0$ and $\nu_2(b) = 1$, i.e. a is odd and $b = 2b'$ with b' odd, the GB division finds $q = \pm 1$ such that $(a + qb')/2$ is even, which is exactly the PM algorithm. Unlike the binary, Hensel and PM algorithms, the GB division considers only low order bits of a and b: there is no need comparing a and b nor computing their bit lengths. It can be seen as a natural operation when considering a and b as 2-adic integers.

We now give two algorithms to perform the GB division. The first one is the equivalent of the naive division algorithm, and is asymptotically slower than the second one, which is the equivalent of the fast division algorithm. For most of the input pairs (a, b), the Euclidean algorithm based on the GB division performs almost all its divisions on pairs (c, d) for which $\nu_2(d) - \nu_2(c)$ is small. For this reason the first algorithm suffices in practice.

```
Algorithm Elementary-GB.
Input: Two integers $a, b$ satisfying $\nu_2(a) < \nu_2(b) < \infty$.
Output: $(q, r) = GB(a, b)$.

1. $q := 0$, $r := a$.
2. While $\nu_2(r) \leq \nu_2(b)$ do
3.     $q := q - 2^{\nu_2(r)-\nu_2(a)}$,
4.     $r := r - 2^{\nu_2(r)-\nu_2(b)} b$.
5. $q := q \text{ cmod } 2^{\nu_2(b)-\nu_2(a)+1}$, $r := q\frac{b}{2^{\nu_2(b)-\nu_2(a)}} + a$.
6. Return $(q, r)$.
```

Fig. 2. Algorithm **Elementary-GB**.

Lemma 2. *The algorithm* **Elementary-GB** *of Fig. 2 is correct and if the input (a, b) satisfies $\ell(a), \ell(b) \leq n$, then it finishes in time $O(n \cdot [\nu_2(b) - \nu_2(a)])$.*

It is also possible to compute $GB(a, b)$ in quasi-linear time, in the case of Schönhage-Strassen multiplication, by using Hensel's lifting (which is the p-adic dual of Newton's iteration).

Lemma 3. *The algorithm* **Fast-GB** *of Fig. 3 is correct and with Schönhage-Strassen multiplication, if a and b satisfy the conditions $\ell(a), \ell(b) \leq 2n$ and $\nu_2(b) - \nu_2(a) \leq n$, then it finishes in time $O(M(n))$.*

2.3 The GB Euclidean Algorithm

A GB Euclidean algorithm can be derived very naturally from the definition of the GB division, see Fig. 4.

Lemma 4. *The GB Euclidean algorithm of Fig. 4 is correct, and if we use the algorithm* **Elementary-GB** *of Fig. 2, then for any input (a, b) satisfying $\ell(a), \ell(b) \leq n$, it finishes in time $O(n^2)$.*

```
Algorithm Fast-GB.
Input: Two integers $a, b$ satisfying $\nu_2(a) < \nu_2(b) < \infty$.
Output: $(q, r) = GB(a, b)$.

1. $A := -\frac{a}{2^{\nu_2(a)}}$, $B := \frac{b}{2^{\nu_2(b)}}$, $n := \nu_2(b) - \nu_2(a) + 1$.
2. $q := 1$.
3. For $i$ from 1 to $\lceil \log n \rceil$ do
4.     $q := q + q(1 - Bq) \bmod 2^{2^i}$.
5. $q := Aq$ cmod $2^n$.
6. $r := a + \frac{q}{2^{n-1}} b$.
7. Return $(q, r)$.
```

Fig. 3. Algorithm **Fast-GB**.

```
Algorithm GB-gcd.
Input: Two integers $a, b$ satisfying $\nu_2(a) < \nu_2(b)$.
Output: The odd part $\frac{g}{2^{\nu_2(g)}}$ of the greatest common divisor $g$ of $a$ and $b$.

1. If $b = 0$, return $\frac{a}{2^{\nu_2(a)}}$.
2. $(q, r) := GB(a, b)$.
3. Return GB-gcd $(b, r)$.
```

Fig. 4. The GB Euclidean algorithm.

Proof. Let $r_0 = a, r_1 = b, r_2, \ldots$ be the sequence of remainders that appear in the execution of the algorithm. We first show that this sequence is finite and thus that the algorithm terminates.

For any $k \geq 0$, Eqs. (1) and (2) give $|r_{k+2}| \leq |r_{k+1}| + |r_k|$, so that $|r_k| \leq 2^{n+1} \left(\frac{1+\sqrt{5}}{2}\right)^k$. Moreover, 2^k divides $|r_k|$, which gives $2^k \leq |r_k| \leq 2^{n+1} \left(\frac{1+\sqrt{5}}{2}\right)^k$ and $(1 - \log \frac{1+\sqrt{5}}{2})k \leq n + 1$. Therefore there are $O(n)$ remainders in the remainder sequence. Let $t = O(n)$ be the length of the remainder sequence. Suppose that r_t is the last non-zero remainder. From Lemma 2, we know that each of the calls to a GB-division involves a number of bit operations bounded by $O(\log |r_k| \cdot [\nu_2(r_{k+1}) - \nu_2(r_k)]) = O(n \cdot [\nu_2(r_{k+1}) - \nu_2(r_k)])$, so that the overall complexity is bounded by $O(n \cdot \nu_2(r_t))$.

For the correctness, remark that the GB remainder r from a and b satisfies $r = a \bmod b'$ where $b' = \frac{b}{2^{\nu_2(b)}}$ is the odd part of b, thus $\gcd(a, b') = \gcd(r, b')$.

Remark. For a practical implementation, one should remove factors of two in Algorithm GB-gcd. If one replaces the return value in Step 1 by a, and in Step 3 by $2^{\nu_2(a)}$GB-gcd$(\frac{b}{2^{\nu_2(b)}}, \frac{r}{2^{\nu_2(b)}})$, the algorithm directly computes $g = \gcd(a, b)$.

A better bound on $|r_k|$ and thus on t is proved in §5.1. Nonetheless the present bound is sufficient to guarantee the quasi-linear time complexity of the recursive algorithm. The improved bound only decreases the multiplying constant of the asymptotic complexity.

For $n \geq 1$ and $q \in [-2^n+1, 2^n-1]$ we define the matrix $[q]_n = \begin{pmatrix} 0 & 2^n \\ 2^n & q \end{pmatrix}$. Let r_0, r_1 be two non-zero integers with $0 = \nu_2(r_0) < \nu_2(r_1)$, and $r_0, r_1, r_2, \ldots$ be their GB remainder sequence, and $q_1, q_2, \ldots$ be the corresponding quotients: $r_{i+1} = r_{i-1} + q_i \frac{r_i}{2^{\nu_2(r_i)-\nu_2(r_{i-1})}}$ for any $i \geq 1$. Then the following relation holds for any $i \geq 1$:

$$\begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix} = \frac{1}{2^{\nu_2(r_i)}} [q_i]_{n_i} \ldots [q_1]_{n_1} \cdot \begin{pmatrix} r_0 \\ r_1 \end{pmatrix},$$

where $n_j = \nu_2(r_j) - \nu_2(r_{j-1}) \geq 1$ for any $j \geq 1$.

In what follows, we use implicitly the following simple fact several times.

Lemma 5. *Let r_0, r_1 be two non-zero integers with $0 = \nu_2(r_0) < \nu_2(r_1)$, and $r_0, r_1, r_2, \ldots$ be their GB remainder sequence. Let $d \geq 0$. Then there exists a unique $i \geq 0$ such that $\nu_2(r_i) \leq d < \nu_2(r_{i+1})$.*

2.4 Computing Modular Inverses

This subsection is independent of the remainder of the paper but is justified by the fact that computing modular inverses is a standard application of the Euclidean algorithm.

Let a, b be two non-zero integers with $0 = \nu_2(a) < \nu_2(b)$ and $\ell(a), \ell(b) \leq n$. Suppose that we want to compute the inverse of b modulo a, by using an extended version of the Euclidean algorithm based on the GB division. The execution of the extended GB Euclidean algorithm gives two integers A and B such that $Aa + Bb = 2^\alpha g$, where $\alpha = O(n)$ and $g = \gcd(a, b)$. From such a relation, it is easy to check that $g = 1$. Suppose now that the inverse B' of b modulo a does exist. From the relation $Aa + Bb = 2^\alpha$, we know that:

$$B' = \frac{B}{2^\alpha} \bmod a.$$

Therefore, in order to obtain B', it is sufficient to compute the inverse of 2^α modulo a. By using Hensel's lifting (like in the algorithm **Fast-GB** of Fig. 3), we obtain the inverse of a modulo 2^α. This gives x and y satisfying: $xa + y2^\alpha = 1$. Clearly y is the inverse of 2^α modulo a.

Since multiplication, Hensel's lifting and division on numbers of size $O(n)$ can be performed in time $O(M(n))$ (see [11]), given two n-bit integers a and b, the additional cost to compute the inverse of b modulo a given the output of an extended Euclidean algorithm based on the GB division is $O(M(n))$.

3 The Recursive Algorithm

We now describe the recursive algorithm based on the GB division. This description closely resembles the one of [12]. It uses two routines: the algorithm

Half-GB-gcd and the algorithm **Fast-GB-gcd**. Given two non-zero integers r_0 and r_1 with $0 = \nu_2(r_0) < \nu_2(r_1)$, the algorithm **Half-GB-gcd** outputs the GB remainders r_i and r_{i+1} of the GB remainder sequence of (r_0, r_1) that satisfy $\nu_2(r_i) \leq \ell(r_0)/2 < \nu_2(r_{i+1})$. It also outputs the corresponding matrix $2^{-(n_1+\ldots+n_i)}[q_i]_{n_i} \ldots [q_1]_{n_1}$. Then we describe the algorithm **Fast-GB-gcd**, which, given two integers a and b, outputs the gcd of a and b by making successive calls to the algorithm **Half-GB-gcd**.

The algorithm **Half-GB-gcd** works as follows: a quarter of the least significant bits of a and b are eliminated by doing a recursive call on the low $\ell(a)/2$ of the bits of a and b. The crucial point is that the GB quotients computed for the truncated numbers are exactly the same as the first GB quotients of a and b. Therefore, by multiplying a and b by the matrix obtained recursively one gets two remainders (a', b') of the GB remainder sequence of (a, b). A single step of the GB Euclidean algorithm is performed on (a', b'), which gives a new remainder pair (b', r). Then there is a second recursive call on approximately $\ell(a)/2$ of the least significant bits of (b', r). The size of the inputs of this second recursive call is similar to the one of the first recursive call. Finally, the corresponding remainders (c, d) of the GB remainder sequence of (a, b) are computed using the returned matrix R_2, and the output matrix R is calculated from R_1, R_2 and the GB quotient of (a', b'). Fig. 5 illustrates the execution of this algorithm.

Note that in the description of the algorithm **Half-GB-gcd** in Fig. 6, a routine GB' is used. This is a simple modification of the GB division: given a and b as input with $0 = \nu_2(a) < \nu_2(b)$, it outputs their GB quotient q, and $\frac{r}{2^{\nu_2(b)}}$ if r is their GB remainder. The algorithm **Fast-GB-gcd** uses several times the algorithm **Half-GB-gcd** to decrease the lengths of the remainders quickly.

The main advantage over the other quasi-linear time algorithms for the integer gcd is that if a matrix R is returned by a recursive call of the algorithm **Half-GB-gcd**, then it contains only "correct quotients". There is no need to go back in the GB remainder sequence in order to make the quotients correct, and thus no need to store the sequence of quotients. The underlying reason is that the remainders are shortened by the least significant bits, and since the carries go in the direction of the most significant bits, these two phenomena do not interfere. For that reason, the algorithm is as simple as the Knuth-Schönhage algorithm in the case of polynomials.

4 Correctness of the Recursive Algorithm

In this section, we show that the algorithm **Fast-GB-gcd** of Fig. 7 is correct. We first give some results about the GB division, and then we show the correctness of the algorithm **Half-GB-gcd** which clearly implies the correctness of the algorithm **Fast-GB-gcd**.

4.1 Some Properties of the GB Division

The properties described below are very similar to the ones of the standard Euclidean division that make the Knuth-Schönhage algorithm possible. The first

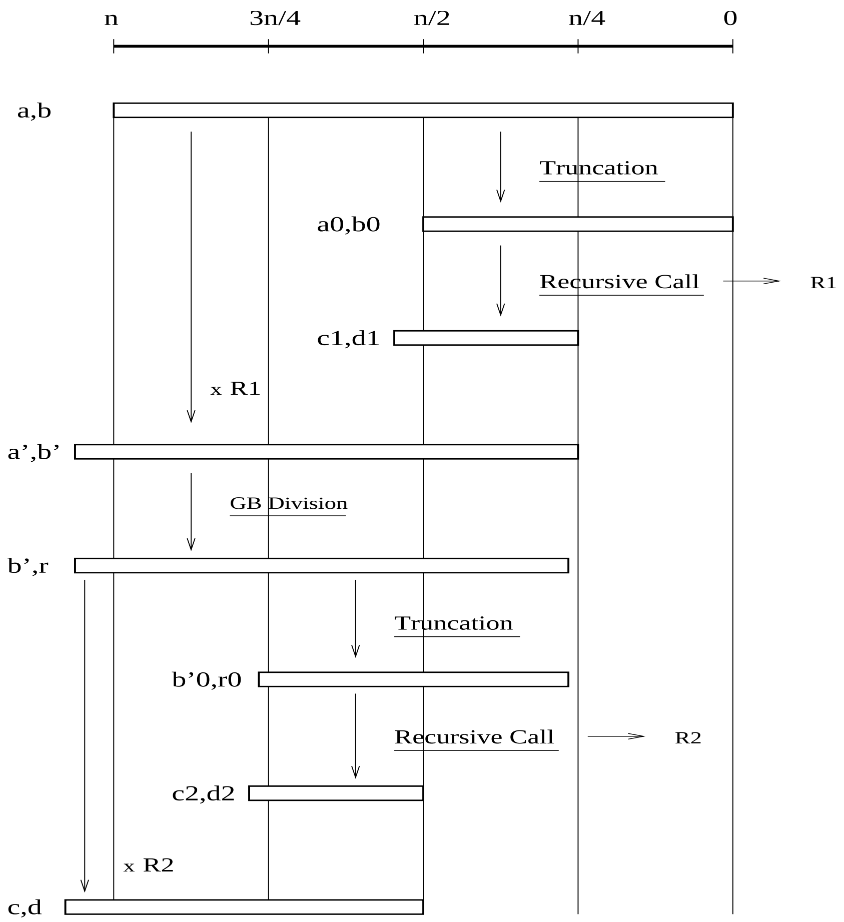

Fig. 5. The recursive structure of the algorithm **Half-GB-gcd**.

result states that the $\nu_2(b) - \nu_2(a) + 1$ last non-zero bits of a and of b suffice to compute the GB quotient of two integers a and b.

Lemma 6. *Let a, b, a' and b' be such that $a' = a \bmod 2^l$ and $b' = b \bmod 2^l$ with $l \geq 2\nu_2(b) + 1$. Assume that $0 = \nu_2(a) < \nu_2(b)$. Let $(q, r) = GB(a, b)$ and $(q', r') = GB(a', b')$. Then $q = q'$ and $r = r' \bmod 2^{l-\nu_2(b)}$.*

Proof. By definition, $q := -a \left(\frac{b}{2^{\nu_2(b)}}\right)^{-1} \operatorname{cmod} 2^{\nu_2(b)+1}$. Therefore, since $l \geq 2\nu_2(b)+1$, $\frac{b}{2^{\nu_2(b)}} = \frac{b'}{2^{\nu_2(b')}} \bmod 2^{\nu_2(b)+1}$, we have $a = a' \bmod 2^{\nu_2(b)+1}$ and $q = q'$. Moreover, $r = a + q\frac{b}{2^{\nu_2(b)}}$ and $r' = a' + q\frac{b'}{2^{\nu_2(b')}}$. Consequently $r = r' \bmod 2^{l-\nu_2(b)}$.

This result can be seen as a continuity statement: two pairs of 2-adic integers (a, b) and (a', b') which are sufficiently close for the 2-adic norm (i.e. some least

```
Algorithm Half-GB-gcd.
Input: $a, b$ satisfying $0 = \nu_2(a) < \nu_2(b)$.
Output: An integer $j$, an integer matrix $R$ and two integers $c$ and $d$ with
$0 = \nu_2(c) < \nu_2(d)$, such that $\begin{pmatrix} c \\ d \end{pmatrix} = 2^{-2j} R \cdot \begin{pmatrix} a \\ b \end{pmatrix}$, and $c^* = 2^j c$, $d^* = 2^j d$
are the two consecutive remainders of the GB remainder sequence of $(a, b)$
that satisfy $\nu_2(c^*) \le \ell(a)/2 < \nu_2(d^*)$.

 1. $k := \lfloor \ell(a)/2 \rfloor$.
 2. If $\nu_2(b) > k$, then return $0, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, a, b$.
 3. $k_1 := \lfloor k/2 \rfloor$.
 4. $a := a_1 2^{2k_1+1} + a_0,\ b := b_1 2^{2k_1+1} + b_0$ with $0 \le a_0, b_0 < 2^{2k_1+1}$.
 5. $j_1, R_1, c_1, d_1 :=$ Half-GB-gcd$(a_0, b_0)$.
 6. $\begin{pmatrix} a' \\ b' \end{pmatrix} := 2^{2k_1+1-2j_1} R_1 \cdot \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} c_1 \\ d_1 \end{pmatrix}$,   $j_0 := \nu_2(b')$.
 7. If $j_0 + j_1 > k$, then return $j_1, R_1, a', b'$.
 8. $(q, r) := GB'(a', b')$.
 9. $k_2 := k - (j_0 + j_1)$.
10. $\frac{b'}{2^{j_0}} := b'_1 2^{2k_2+1} + b'_0,\ r := r_1 2^{2k_2+1} + r_0$ with $0 \le b'_0, r_0 < 2^{2k_2+1}$.
11. $j_2, R_2, c_2, d_2 :=$ Half-GB-gcd$(b'_0, r_0)$.
12. $\begin{pmatrix} c \\ d \end{pmatrix} := 2^{2k_2+1-2j_2} R_2 \cdot \begin{pmatrix} b'_1 \\ r_1 \end{pmatrix} + \begin{pmatrix} c_2 \\ d_2 \end{pmatrix}$.
13. Return $j_1 + j_0 + j_2,\ R_2 \cdot [q]_{j_0} \cdot R_1,\ c,\ d$.
```

Fig. 6. Algorithm **Half-GB-gcd**.

significant bits of a and a', and some of b and b' are equal) have the same quotient and similar remainders (the closer the pairs, the closer the remainders). The second lemma extends this result to the GB continued fraction expansions: if (a, b) and (a', b') are sufficiently close, their first GB quotients are identical. We obtain this result by applying the first one several times.

Lemma 7. *Let a, b, a' and b' such that $a' = a$ mod 2^{2k+1} and $b' = b$ mod 2^{2k+1}, with $k \ge 0$. Suppose that $0 = \nu_2(a) < \nu_2(b)$. Let $r_0 = a, r_1 = b, r_2, \ldots$ be the GB remainder sequence of (a, b), and let $q_1, q_2, \ldots$ be the corresponding GB quotients: $r_{j+1} = r_{j-1} + q_j \frac{r_j}{2^{n_j}}$, with $n_j = \nu_2(r_j) - \nu_2(r_{j-1})$. Let $r'_0 = a', r'_1 = b', r'_2, \ldots$ be the GB remainder sequence of (a', b'), and let $q'_1, q'_2, \ldots$ be the corresponding GB quotients: $r'_{j+1} = r'_{j-1} + q'_j \frac{r'_j}{2^{n'_j}}$, with $n'_j = \nu_2(r'_j) - \nu_2(r'_{j-1})$.*
Then, if r_{i+1} is the first remainder such that $\nu_2(r_{i+1}) > k$, we have $q_j = q'_j$ and $r_{j+1} = r'_{j+1}$ mod $2^{2k+1-\nu_2(r_j)}$, for any $j \le i$.

```
Algorithm Fast-GB-gcd.
Input: $a, b$ satisfying $0 = \nu_2(a) < \nu_2(b)$.
Output: $g = \gcd(a, b)$.

 1. $j, R, a', b' :=$ Half-GB-gcd$(a, b)$.
 2. If $b' = 0$, return $a'$.
 3. $(q, r) := GB'(a', b')$.
 4. Return Fast-GB-gcd$(b', r')$.
```

Fig. 7. Algorithm **Fast-GB-gcd**.

Proof. We prove this result by induction on $j \geq 0$. This is true for $j = 0$, because $a' = a \bmod 2^{2k+1}$ and $b' = b \bmod 2^{2k+1}$. Suppose now that $1 \leq j \leq i$. We use Lemma 6 with $\frac{r_{j-1}}{2^{\nu_2(r_{j-1})}}, \frac{r_j}{2^{\nu_2(r_{j-1})}}, \frac{r'_{j-1}}{2^{\nu_2(r_{j-1})}}, \frac{r'_j}{2^{\nu_2(r_{j-1})}}$ and $l = 2k+1-2\nu_2(r_{j-1})$. By induction, modulo $2^{2k+1-\nu_2(r_{j-1})}$, $r_{j-1} = r'_{j-1}$ and $r_j = r'_j$. Since $j \leq i$, we have, by definition of i, $2k+1-2\nu_2(r_{j-1}) \geq 2(\nu_2(r_j) - \nu_2(r_{j-1})) + 1$, and consequently we can apply Lemma 6. Thus $q_j = q'_j$ and $r_{j+1} = r'_{j+1} \bmod 2^{2k+1-\nu_2(r_j)}$.

Practically, this lemma says that k bits can be gained as regard to the initial pair (a, b) by using only $2k$ bits of a and $2k$ bits of b. This is the advantage of using the GB division instead of the standard division: in the case of the standard Euclidean division, this lemma is only "almost true", because some of the last quotients before gaining k bits can differ, and have to be repaired.

4.2 Correctness of the Half-GB-gcd Algorithm

To show the correctness of the algorithm **Fast-GB-gcd**, it suffices to show the correctness of the algorithm **Half-GB-gcd**, which is established in the following theorem. (Since each call to **Fast-GB-gcd** which does not return in Step 2 performs at least one GB division, the termination is ensured.)

Theorem 1. *The algorithm* **Half-GB-gcd** *of Fig. 6 is correct.*

Proof. We prove the correctness of the algorithm by induction on the size of the inputs. If $\ell(a) = 1$, then the algorithm finishes at Step 2 because $\nu_2(b) \geq 1$. Suppose now that $k \geq 2$ and that $\nu_2(b) \leq k$.

Since $2\lfloor \frac{k}{2} \rfloor + 1 < \ell(a)$, Step 5 is a recursive call (its inputs satisfy the input conditions). By induction j_1, R_1, c_1 and d_1 satisfy $\begin{pmatrix} c_1 \\ d_1 \end{pmatrix} = 2^{-2j_1} R_1 \cdot \begin{pmatrix} a_0 \\ b_0 \end{pmatrix}$, and $2^{j_1} c_1$ and $2^{j_1} d_1$ are the consecutive remainders r'_{i_1} and r'_{i_1+1} of the GB remainder sequence of $r'_0 = a_0$ and $r'_1 = b_0$ that satisfy $\nu_2(r'_{i_1}) \leq k_1 < \nu_2(r'_{i_1+1})$. From Lemma 7, we know that $2^{-j_1} R_1 \cdot \begin{pmatrix} a \\ b \end{pmatrix}$ are two consecutive remainders $2^{j_1} a' = r_{i_1}$ and $2^{j_1} b' = r_{i_1+1}$ of the GB remainder sequence of $r_0 = a$ and $r_1 = b$, and they satisfy $r_{i_1} = r'_{i_1}$ and $r_{i_1+1} = r'_{i_1+1}$ modulo 2^{k_1+1}. From these last equalities, we have that $\nu_2(r_{i_1}) = \nu_2(r'_{i_1}) \le k_1 < \nu_2(r_{i_1+1}) \le \nu_2(r'_{i_1+1})$. Thus, if the execution of the algorithm stops at Step 7, the output is correct.

Otherwise, r_{i_1+2} is computed at Step 8. At Step 9 we compute $k_2 = k - \nu_2(r_{i_1+1})$. Step 7 ensures that $k_2 \ge 0$. Since $\nu_2(r_{i_1+1}) > \lfloor k/2 \rfloor$, we have $k_2 \le \lceil k/2 \rceil - 1$. Therefore Step 11 is a recursive call (and the inputs satisfy the input conditions). By induction, j_2, S_2, c_2 and d_2 satisfy: $\begin{pmatrix} c_2 \\ d_2 \end{pmatrix} = 2^{-2j_2} R_2 \cdot \begin{pmatrix} b'_0 \\ r_0 \end{pmatrix}$, and $2^{j_2} c_2$ and $2^{j_2} d_2$ are the consecutive remainders r'_{i_2} and r'_{i_2+1} of the GB remainder sequence of (b_0, r'_0). Moreover, $\nu_2(r'_{i_2}) \le k_2 < \nu_2(r'_{i_2+1})$. From Lemma 7, we know that $2^{-j_2} S_2 \cdot \begin{pmatrix} 2^{j_1} b' \\ 2^{j_1} r' \end{pmatrix}$ are two consecutive remainders r_i and r_{i+1} (with $i = i_1 + i_2 + 1$) of the GB remainder sequence of (a, b), that $\frac{r_i}{2^{j_1 + \nu_2(b')}} = 2^{j_2} c_2 \bmod 2^{k_2+2}$ and that $\frac{r_{i+1}}{2^{j_1 + \nu_2(b')}} = 2^{j_2} d_2 \bmod 2^{k_2+1}$. Therefore the following sequence of inequalities is valid: $\nu_2(r_i) = j_1 + j_2 + \nu_2(b') \le k < \nu_2(r_{i+1})$. This ends the proof of the theorem.

5 Analysis of the Algorithms

In this section, we first study the GB Euclidean algorithm. In particular we give a worst-case bound regarding the length of the GB remainder sequence. Then we bound the complexity of the recursive algorithm in the case of the use of Schönhage-Strassen multiplication, and we give some intuition about the average complexity.

5.1 Length of the GB Remainder Sequence

In this subsection, we bound the size of the matrix $\frac{1}{2^{\nu_2(r_i)}} [q_i]_{n_i} \cdots [q_1]_{n_1}$, where the q_j's are the GB quotients of a truncated GB remainder sequence $r_0, \ldots, r_{i+1}$ with $\nu_2(r_i) \le \nu_2(r_0) + d < \nu_2(r_{i+1})$ for some $d \ge 0$. This will make possible the analysis of the lengths and the number of the GB remainders. As mentioned in §2, this subsection is not necessary to prove the quasi-linear time complexity of the algorithm.

Theorem 2. *Let $d \ge 1$. Let r_0, r_1 with $0 = \nu_2(r_0) < \nu_2(r_1)$, and $r_0, r_1, \ldots, r_{i+1}$ their first GB remainders, where i is such that $\nu_2(r_i) \le d < \nu_2(r_{i+1})$. We consider the matrix $\frac{1}{2^{\nu_2(r_i)}} [q_i]_{n_i} \cdots [q_1]_{n_1}$, where the q_j's are the GB quotients and $n_j = \nu_2(r_j) - \nu_2(r_{j-1})$ for any $1 \le j \le i$. Let M be the maximum of the absolute values of the four entries of this matrix. Then we have:*

- *If $d = 0$ or 1, $M = 1$,*
- *If $d = 3$, $M \le 11/8$,*
- *If $d = 5$, $M \le 67/32$,*
- *If $d = 2, 4$ or $d \ge 6$, $M \le \frac{2}{\sqrt{17}} \left(\left(\frac{1+\sqrt{17}}{4} \right)^{d+1} - \left(\frac{1-\sqrt{17}}{4} \right)^{d+1} \right)$.*

Moreover, all these bounds are reached, in particular, the last one is reached when $n_j = 1$ and $q_j = 1$ for any $1 \leq j \leq i$.

The proof resembles the worst case analysis of the Gaussian algorithm in [9].

Proof. We introduce a partial order on the 2×2 matrices: $A < B$ if and only if for any coordinate $[i, j]$, $A[i, j] \leq B[i, j]$. First, the proof can be restricted to the case where the q_j's are non-negative integers, because we have the inequality $|[q_i]_{n_i} \dots [q_1]_{n_1}| \leq [|q_i|]_{n_i} \dots [|q_1|]_{n_1}$. This can be easily showed by induction on i by using the properties: $|A \cdot B| \leq |A| \cdot |B|$, and if the entries of A, A', B, B' are non-negative, $A \leq A'$ and $B \leq B'$ implies $A \cdot B \leq A' \cdot B'$.

Consequently we are looking for the maximal elements for the partial order $>$ in the set: $\{\Pi_{n_1+\ldots+n_i \leq d}\ [q_i]_{n_i} \dots [q_1]_{n_1}\ /\ \forall 1 \leq j \leq i,\ 0 < q_j \leq 2^{n_j} - 1 \text{ and } n_j \geq 1\}$. We can restrict the analysis to the case where $n_1 + \ldots + n_i = d$ and all the q_j's are maximal, which gives the set:

$$\{\Pi_{n_1+\ldots+n_i=d}\ [2^{n_i} - 1]_{n_i} \dots [2^{n_1} - 1]_{n_1}\}.$$

Remark now that $[2^n - 1]_n \leq [1]_1^n$ for any $n \geq 3$. Therefore, it is sufficient to consider the case where the n_j's are in $\{1, 2\}$. Moreover, for any integer $j \geq 0$, $[3]_2 \cdot [1]_1^j \cdot [3]_2 \leq [1]_1^{j+4}$, and we also have the inequalities $[3]_2^2 \leq [1]_1^4$, $[1]_1^5 \cdot [3]_2 \leq [1]^7$, $[3]_2 \cdot [1]_1^5 \leq [1]^7$ and $[1]_1^2 \cdot [3]_2 \cdot [1]_1^2 \leq [1]_1^6$.

From these relations, we easily obtain the maximal elements:

- For $d = 1$, $[1]_1$.
- For $d = 2$, $[1]_1^2$ and $[3]_2$.
- For $d = 3$, $[1]_1^3$, $[3]_2 \cdot [1]_1$ and $[1]_1 \cdot [3]_2$.
- For $d = 4$, $[1]_1^4$, $[3]_2 \cdot [1]_1^2$, $[1]_1 \cdot [3]_2 \cdot [1]_1$ and $[1]_1^2 \cdot [3]_2$.
- For $d = 5$, $[1]_1^5$, $[3]_2 \cdot [1]_1^3$, $[1]_1^2 \cdot [3]_2 \cdot [1]_1$, $[1]_1 \cdot [3]_2 \cdot [1]_1^2$ and $[1]_1^3 \cdot [3]_2$.
- For $d = 6$, $[1]_1^6$, $[3]_2 \cdot [1]_1^4$, $[1]_1^3 \cdot [3]_2 \cdot [1]_1$, $[1]_1 \cdot [3]_2 \cdot [1]_1^3$ and $[1]_1^4 \cdot [3]_2$.
- For $d = 7$, $[1]_1^7$, $[1]_1 \cdot [3]_2 \cdot [1]_1^4$ and $[1]_1^4 \cdot [3]_2 \cdot [1]_1$.
- For $d \geq 8$, $[1]_1^d$.

The end of the proof is obvious once we note that $2^{-d}[1]_1^d = \begin{pmatrix} u_{d-1} & u_d \\ u_d & u_{d+1} \end{pmatrix}$, where $u_0 = 0$, $u_1 = 1$ and $u_i = u_{i-2} + \frac{1}{2} u_{i-1}$.

From this result on the quotient matrices, we can easily deduce the following on the size of the remainders and on the length of the remainder sequence.

Theorem 3. *Let r_0, r_1 be two non-zero integers with $0 = \nu_2(r_0) < \nu_2(r_1)$, and $r_0, r_1, \dots, r_{t+1}$ their complete GB remainder sequence, i.e. with $r_{t+1} = 0$. Assume that $9 \leq j \leq t$. Then:*

$$2^{\nu_2(r_j)} \leq |r_j| \leq \frac{2}{\sqrt{17}} \left[|r_0| \left(\left(\frac{1+\sqrt{17}}{4} \right)^{\nu_2(r_j)-1} - \left(\frac{1-\sqrt{17}}{4} \right)^{\nu_2(r_j)-1} \right) \right.$$
$$\left. + |r_1| \left(\left(\frac{1+\sqrt{17}}{4} \right)^{\nu_2(r_j)} - \left(\frac{1-\sqrt{17}}{4} \right)^{\nu_2(r_j)} \right) \right].$$

The upper bound is reached by $\begin{pmatrix} r_0 \\ r_1 \end{pmatrix} = [1]_1^{-t} \cdot \begin{pmatrix} 2^t \\ 0 \end{pmatrix}$.
Moreover, if $\ell(r_0), \ell(r_1) \leq n$*, then we have:* $t \leq n/\log(\frac{\sqrt{17}-1}{2})$.

Proof. (Sketch) For the inequations concerning $|r_j|$, use the maximal element $[1]_1^d$ from the proof of Theorem 2, with $d = \nu_2(r_i)$, where $i = j-1$. Remark then that the upper bound grows far slower with $\nu_2(r_j)$ than the lower bound: this fact gives an upper bound on $\nu_2(r_t)$ and therefore on t.

As a comparison, we recall that the worst case for the standard Euclidean division corresponds to the Fibonacci sequence, with $d \leq n/\log(\frac{1+\sqrt{5}}{2}) + o(n)$. Remark that $1/\log(\frac{1+\sqrt{5}}{2}) \approx 1.440$ and $1/\log(\frac{\sqrt{17}-1}{2}) \approx 1.555$.

5.2 Complexity Bound for the Recursive Algorithm

In what follows, $H(n)$ and $G(n)$ respectively denote the maximum of the number of bit operations performed by the algorithms **Half-GB-gcd** and **Fast-GB-gcd**, given as inputs two integers of lengths at most n.

Lemma 8. *Let* $c = \frac{1}{2}\log\frac{1+\sqrt{17}}{2} \approx 0.679$*. The following two relations hold:*

- $G(n) = H(n) + G(\lceil cn \rceil) + O(n)$,
- $H(n) = 2H(\lfloor \frac{n}{2} \rfloor + 1) + O(M(n))$.

Proof. The first relation is an obvious consequence of Theorem 2. We now prove the second relation. The costs of Steps 1, 2, 3, 4, 7, 9 and 10 are negligible. Steps 5 and 11 are recursive calls and the cost of each one is bounded by $H(\lfloor \frac{n}{2} \rfloor + 1)$. Steps 6, 12 and 13 consist of multiplications of integers of size $O(n)$. Finally, Step 8 is a single GB division, and we proved in Lemma 3 that it can be performed in time $O(M(n))$.

From this result and the fact that $c < 1$, one easily obtains the following theorem:

Theorem 4. *The algorithm* **Fast-GB-gcd** *of Fig. 7 runs in quasi-linear time. More precisely,* $G(n) = O(M(n)\log n)$.

The constants that one can derive from the previous proofs are rather large, and not very significant in practice. In fact, for randomly chosen n-bit integers, the quotients of the GB remainder sequence are $O(1)$, and therefore Step 8 of the algorithm **Half-GB-gcd** has a negligible cost. Moreover, the worst-case analysis on the size of the coefficients of the returned matrices gives a worst-case bound $O\left(\left(\frac{1+\sqrt{17}}{2}\right)^n\right)$, which happens to be $O(2^n)$ in practice. With these two heuristics, the "practical cost" of the algorithm **Half-GB-gcd** satisfies the same recurrence than the Knuth-Schönhage algorithm:

$$H(n) \approx 2H(\frac{n}{2}) + kM(n),$$

where the constant k depends from the implementation [11].

6 Implementation Issues

We have implemented the algorithms described in this paper in GNU MP [3]. In this section, we first give some "tricks" that we implemented to improve the efficiency of the algorithm, and then we give some benchmarks.

6.1 Some Savings

First of all, note that some multiplications can be saved easily from the fact that when the algorithm **Fast-GB-gcd** calls the algorithm **Half-GB-gcd**, the returned matrix is not used. Therefore, for such "top-level" calls to the algorithm **Half-GB-gcd**, there is no need to compute the product $R_2 \cdot [q]_{j_0} \cdot R_1$.

Note also that for interesting sizes of inputs (our implementation of the recursive algorithm is faster than usual Euclidean algorithms for several thousands of bits), we are in the domains of Karatsuba and Toom-Cook multiplications, and below the domain of FFT-based multiplication. This leads to some improvements. For example, the algorithm **Fast-GB-gcd** should use calls to the algorithm **Half-GB-gcd** in order to gain γn bits instead of $\frac{n}{2}$, with a constant $\gamma \neq \frac{1}{2}$ that has to be optimized.

Below a certain threshold in the size of the inputs (namely several hundreds of bits), a naive quadratic algorithm that has the requirements of algorithm **Half-GB-gcd** is used. Moreover, each time the algorithm has to compute a GB quotient, it computes several of them in order to obtain a 2×2 matrix with entries of length as close to the size of machine words as possible. This is done by considering only the two least significant machine words of the remainders (which gives a correct result, because of Lemma 6).

6.2 Comparison to Other Implementations of Subquadratic Gcd Algorithms

We compared our implementation in GNU MP — using the ordinary integer interface `mpz` — with those of Magma V2.10-12 and Mathematica 5.0, which both provide a subquadratic integer gcd. This comparison was performed on `laurent3.medicis.polytechnique.fr`, an Athlon MP 2200+. Our implementation wins over the quadratic gcd of GNU MP up from about 2500 words of 32 bits, i.e. about 24000 digits. We used as test numbers both the worst case of the classical subquadratic gcd, i.e. consecutive Fibonacci numbers F_n and F_{n-1}, and the worst case of the binary variant, i.e. G_n and $2G_{n-1}$, where $G_0 = 0$, $G_1 = 1$, $G_n = -G_{n-1} + 4G_{n-2}$, which gives all binary quotients equal to 1. Our experiments show that our implementation in GNU MP of the binary recursive gcd is 3 to 4 times faster than the implementations of the classical recursive gcd in Magma or Mathematica. This ratio does not vary much with the inputs. For example, ratios for F_n and G_n are quite similar.

type, n	Magma V2.10-12	Mathematica 5.0	Fast-GB-gcd (GNU MP)
F_n, 10^6	2.89	2.11	0.70
F_n, $2 \cdot 10^6$	7.74	5.46	1.91
F_n, $5 \cdot 10^6$	23.3	17.53	6.74
F_n, 10^7	59.1	43.59	17.34
G_n, $5 \cdot 10^5$	2.78	2.06	0.71
G_n, 10^6	7.99	5.30	1.94

Fig. 8. Timings in seconds of the gcd routines of Magma, Mathematica and our implementation in GNU MP.

Acknowledgements.

The first author thanks Brigitte Vallée for introducing him to fast gcd algorithms and for very helpful discussions on that topic. We thank Jean Vuillemin for the pointer to Hensel's odd division, the Medicis center for allowing the comparison to Magma and Mathematica, and Richard Brent for his comments, which helped to improve both the contents and the form of the paper.

References

1. R. P. Brent. Twenty years' analysis of the binary Euclidean algorithm. In A. W. Roscoe J. Davies and J. Woodcock, editors, *Millenial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in honour of Professor Sir Antony Hoare*, pages 41–53, Palgrave, New York, 2000.
2. R. P. Brent and H. T. Kung. A systolic VLSI array for integer GCD computation. In K. Hwang, editor, *Proceedings of the 7th Symposium on Computer Arithmetic (ARITH-7)*. IEEE CS Press, 1985.
3. T. Granlund. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 4.1.2 edition, 2002. `http://www.swox.se/gmp/#DOC`.
4. D. Knuth. The analysis of algorithms. In *Actes du Congrès International des Mathématiciens de 1970*, volume 3, pages 269–274, Paris, 1971. Gauthiers-Villars.
5. P. L. Montgomery. Modular multiplication without trial division. *Math. Comp.*, 44(170):519–521, 1985.
6. A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.
7. A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.
8. M. Shand and J. E. Vuillemin. Fast implementations of RSA cryptography. In E. E. Swartzlander, M. J. Irwin, and J. Jullien, editors, *Proceedings of the 11th IEEE Symposium on Computer Arithmetic (ARITH-11)*, pages 252–259. IEEE Computer Society Press, Los Alamitos, CA, 1993.
9. B. Vallée. Gauss' algorithm revisited. *Journal of Algorithms*, 12:556–572, 1991.
10. B. Vallée. Dynamical analysis of a class of Euclidean algorithms. *Th. Computer Science*, 297(1-3):447–486, 2003.
11. J. von zur Gathen and J. Gerhard. *Modern Computer Algebra.* Cambridge University Press, 2nd edition, 2003.
12. C. K. Yap. *Fundamental Problems in Algorithmic Algebra.* Oxford University Press, 2000.