J. Symbolic Computation (1994) 17, 473–486

Analysis of a Left-Shift Binary GCD Algorithm

JEFFREY SHALLIT[†] AND JONATHAN SORENSON[‡]

[†] *Department of Computer Science, University of Waterloo, Canada*
[‡] *Department of Mathematics and Computer Science, Butler University, Indianapolis, USA.*

(Received 3 October 1994)

We introduce a new left-shift binary algorithm, LSBGCD, for computing the greatest common divisor of two integers, and we provide an analysis of the worst-case behavior of this algorithm. The analysis depends on a theorem of Ramharter about the extremal behavior of certain continuants.

1. Introduction

There are many polynomial-time algorithms known for computing the greatest common divisor of two positive integers: for example, Euclid's algorithm, Euclid's algorithm with least remainder, Stein's "right-shift" binary algorithm (Stein, 1967), and Brent's "left-shift" binary algorithm (Brent, 1976). The analysis of these algorithms has received much attention; for example, see (Lamé, 1844; Collins, 1974; Knuth, 1981) for Euclid's algorithm; (Dupré, 1846) for Euclid's algorithm with least remainder; and (Brent, 1976; Norton, 1989, Shallit and Sorenson, 1993) for the left- and right-shift binary algorithms.

In particular, it is desirable to determine the *worst-case* behavior of such algorithms: for each n, find the minimal input that requires the algorithm to perform n "steps". Of course, it is not immediately clear what is meant by "minimal input", since these algorithms typically have two inputs. (Nor is the definition of "step" immediately obvious. However, it is frequently taken to be an instance of the most expensive operation, such as a division.) One reasonable definition of "minimal input" is the *lexicographically least* input: we say (x, y) is lexicographically less than or equal to (x', y') if $x < x'$, or $x = x'$ and $y \leq y'$.

In this paper, we present and analyze a new kind of left-shift binary algorithm, LSBGCD, for computing the greatest common divisor. The LSBGCD algorithm is, in some sense, a "least-remainder" variant of the usual left-shift algorithm introduced by Brent (1976). It is also efficient in practice (see Section 7). Our worst-case analysis is of independent interest, since it shows that the worst-case inputs do not arise from the truncation of

[†] Research of the first author supported in part by a grant from NSERC Canada. E-mail address: shallit@graceland.uwaterloo.ca.

[‡] Research of the second author supported in part by a Butler University Fellowship and NSF grant CCR-9204414. E-mail address: sorenson@butler.edu.

the continued fraction expansion of an irrational number, as is the case with the ordinary and least-remainder Euclidean algorithms.

Below is a Pascal-like pseudocode description of the LSBGCD algorithm.

```
LSBGCD

    INPUT: Positive integers $u$ and $v$, $u \geq v$
    OUTPUT: $\gcd(u, v)$

    while $v \neq 0$ do
        find $e \geq 0$ such that $2^e v \leq u < 2^{e+1} v$;
        $t := \min\{u - 2^e v, 2^{e+1} v - u\}$;
        $u := v$; $v := t$;
        if $u < v$ then interchange$(u, v)$;
    return$(u)$;
```

Correctness of the algorithm is left to the reader.

As a consequence of our main result, the number of iterations used by this algorithm is at most $O(\log(uv))$. Assuming that u and v are represented in binary, finding e essentially reduces to comparing the lengths of u and v in binary, and multiplying v by 2^e is simply a shift. Therefore, each iteration can be computed using at most $O(\log(uv))$ bit operations. Thus LSBGCD uses at most $O(\log^2(uv))$ bit operations to compute $\gcd(u, v)$. This algorithm has a k-ary extension (see Sorenson, 1994), making LSBGCD the special case of $k = 2$.

Our main result is the following.

THEOREM 1.1. *Let $x_n \geq y_n \geq 0$, and let (x_n, y_n) be the lexicographically least pair of integers that requires the LSBGCD algorithm to perform n iterations. Then both of the sequences (x_n) and (y_n) satisfy linear recurrences of order 16 with characteristic polynomial $(z-1)(z+1)(z^2+1)(z^4-4z^2+1)^2(z^4+4z^2+1)$. Furthermore, letting $\alpha = 2 + \sqrt{3}$ and $\beta = 2 - \sqrt{3}$, we have*

$$\begin{aligned}
x_{2n+1} &= \frac{\sqrt{3}}{6}(\alpha^{n+1} - \beta^{n+1}); \\
y_{2n+1} &= \frac{\sqrt{3}}{6}((1+\sqrt{3})\alpha^n - (1-\sqrt{3})\beta^n); \\
x_{4n} &= \frac{2 + (4-\sqrt{3})\alpha^{2n+1} + (4+\sqrt{3})\beta^{2n+1}}{12}; \\
y_{4n} &= \frac{-2 + (1+3\sqrt{3})\alpha^{2n} + (1-3\sqrt{3})\beta^{2n}}{12}; \\
x_{4n+2} &= \frac{4 + (4-\sqrt{3})\alpha^{2n+2} + (4+\sqrt{3})\beta^{2n+2}}{12}; \\
y_{4n+2} &= \frac{-10 + (1+3\sqrt{3})\alpha^{2n+1} + (1-3\sqrt{3})\beta^{2n+1}}{12}.
\end{aligned}$$

Table 1 gives the first few values of the sequences (x_n) and (y_n).

Table 1. Values of (x_n) and (y_n) for $1 \leq n \leq 20$.

n	x_n	y_n	n	x_n	y_n
1	1	1	11	780	571
2	3	1	12	1906	1395
3	4	3	13	2911	2131
4	10	7	14	7113	5206
5	15	11	15	10864	7953
6	37	26	16	26545	19432
7	56	41	17	40545	29681
8	137	100	18	99067	72521
9	209	153	19	151316	110771
10	511	373	20	369722	270655

The proof of Theorem 1.1 has two main steps: First, we show that, for worst-case inputs, LSBGCD always assigns either $t := u - v$ or $t := u - 2v$ in its main loop, and never performs an interchange. After introducing necessary notation in Section 2, this is proven in Sections 3 and 4. Second, we observe that, for worst-case inputs (x_n, y_n), the steps performed by the algorithm correspond to the continued fraction expansion of x_n/y_n, and therefore must have a special form. Applying a theorem of Ramharter (1983) then allows us to complete the proof. With this strategy, we complete the proof of our main result in Sections 5 and 6. We conclude in Section 7 with some remarks on the implementation of the LSBGCD algorithm.

2. Notation, Definitions, and an Example

In order to determine the smallest pair (u, v), we will model computations performed by this algorithm in terms of linear transformations on $\mathbf{N} \times \mathbf{N}$.

One loop iteration of the algorithm must perform precisely one of the following two transformations:

$$\begin{array}{ll} A_e(u,v) := (v, u - 2^e v) & \text{occurs when } u - 2^e v \leq 2^{e+1}v - u; \\ B_e(u,v) := (v, 2^{e+1}v - u) & \text{occurs when } u - 2^e v > 2^{e+1}v - u. \end{array}$$

In addition, either of these is followed by the *interchange* transformation $I(u,v) := (v,u)$ if the new pair (u,v) satisfies $u < v$.

Thus, a computation of the algorithm can be expressed as the composition of A_e, B_e, and I transformations. If f and g are transformations, we write $fg(u,v)$ for $f(g(u,v))$. Thus one iteration of the algorithm performs one of A_e, B_e, IA_e, or IB_e. We refer to one of these four options as a *step*, and we refer to any composition of steps as a *partial computation*. The *length* of a partial computation is the number of steps it contains, and corresponds to the number of loop iterations performed by the algorithm.

Example

Consider the computation of the algorithm on the inputs $(47, 40)$. The following table summarizes the steps performed by the algorithm:

u	v	t	Step
47	40	7	A_0
40	7	12	IA_2
12	7	2	B_0
7	2	1	B_1
2	1	0	A_1
1	0		

We write this computation as

$$A_1 B_1 B_0 I A_2 A_0.$$

This partial computation has length 5.

Loosely speaking, our goal is to construct a long partial computation where the corresponding inputs are as small as possible. We do this by starting with the pair $(1, 0)$ and applying the inverses of steps to obtain the corresponding input. The inverses of our three transformations are:

$$\begin{aligned} A_e^{-1}(x,y) &:= (2^e x + y, x) \\ B_e^{-1}(x,y) &:= (2^{e+1} x - y, x) \\ I^{-1}(x,y) &:= (y,x) = I(x,y) \end{aligned}$$

Let f be the partial computation $A_1 B_1 A_0$. Observe that $f^{-1}(1,0) = (9,7)$. If we trace the algorithm on $(9,7)$, we in fact obtain the computation f, of length 3. However, it is possible to construct partial computations which would never occur as actual computations of the algorithm. The shortest such example is $A_0 A_0$. Inverting this computation on $(1,0)$, we obtain the pair $(2,1)$, and the algorithm actually performs step A_1 on this input.

Let f be a partial computation of length n. Define $(u,v) = f^{-1}(x,y)$, and let

$$(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$$

be the successive values taken by (u, v) after the 1st, 2nd, ..., nth iterations of the algorithm. We say f is *valid* on the pair (x,y) if $u \geq v$, $(u_n, v_n) = (x,y)$, and the n iterations performed by the algorithm correspond to the partial computation f. Thus, $A_1 B_1 A_0$ is valid on $(1,0)$, and $A_0 A_0$ is not. Also notice that $A_1 A_1$ is valid on $(10,5)$, and in fact on any pair (x,y) where $x \geq y$ except the pair $(0,0)$.

In the following section, we give necessary and sufficient conditions for constructing valid partial computations. We assume throughout that $x, y \geq 0$.

3. Valid Computations

We begin by examining the properties of the A_e and B_e transformations.

LEMMA 3.1. *The algorithm will perform the transformation A_e resulting in the pair (x,y) if and only if $y \le 2^{e-1}x$.*

PROOF. Let $(u,v) = A_e^{-1}(x,y)$ so that $u = 2^e x + y$ and $v = x$. The algorithm performs A_e on (u,v) if and only if $u \ge 2^e v$ and $u - 2^e v \le 2^{e+1}v - u$. But these two inequalities are true if and only if $2^e v \le u \le (3/2)\cdot 2^e v$, which in turn is true if and only if $2^e x \le 2^e x + y \le (3/2)\cdot 2^e x$. But this is equivalent to $y \ge 0$ and $y \le 2^{e-1}x$. □

LEMMA 3.2. *The algorithm will perform the transformation B_e resulting in the pair (x,y) if and only if $y < 2^{e-1}x$ and $y > 0$.*

PROOF. Let $(u,v) = B_e^{-1}(x,y)$ so that $u = 2^{e+1}x - y$ and $v = x$. The algorithm performs B_e on (u,v) if and only if $u < 2^{e+1}v$ and $u - 2^e v > 2^{e+1}v - u$. But these two inequalities are true if and only if $(3/2)\cdot 2^e v < u < 2^{e+1}v$, which in turn is true if and only if $(3/2)\cdot 2^e x < 2^{e+1}x - y < 2^{e+1}x$. But this is equivalent to $y > 0$ and $y < 2^{e-1}x$. □

THEOREM 3.3. *A partial computation consisting of a single step is valid on the pair (x,y) if and only if $x \ge y$ and the condition indicated below holds:*

Step	Condition
A_e	$y \le 2^{e-1}x$
B_e	$y < 2^{e-1}x$ *and* $y > 0$
IA_e	$x \le 2^{e-1}y$ *and* $e > 1$
IB_e	$x < 2^{e-1}y$, $x > 0$, *and* $e > 1$

PROOF. The proof for steps A_e and B_e follows from a direct application of Lemmas 3.1 and 3.2.

Let $(u,v) = f^{-1}(x,y)$, where f is either IA_e or IB_e. For f to be valid on (x,y), we must have $x > y$ because if $x = y$ no interchange takes place. Thus $e > 1$. The rest of the proof is easily completed with the use of Lemmas 3.1 and 3.2. □

COROLLARY 3.4. *The partial computations A_0A_0, A_0B_0, B_0A_0, and B_0B_0 are not valid for any pairs (x,y).*

PROOF. We prove the corollary for A_0B_0. The other three cases are similar.

Let $(w,z) = A_0^{-1}(x,y) = (x+y, x)$. For the A_0 step to be valid, we must have $y \le x/2$. But then $z \le w \le (3/2)z$, so $z \ge (2/3)w$. This implies the B_0 step is not valid by Theorem 3.3 (since $z > w/2$), and hence the partial computation is invalid. □

LEMMA 3.5. *Let $f = s_1 s_2 \cdots s_n$ be a partial computation of length n, where each s_i is an A_e step, and no two consecutive steps are both A_0. Then f is valid on (x,y) if s_1 is valid on (x,y).*

PROOF. Fix some pair (x,y). Define $(x_0,y_0) = (x,y)$ and $(x_i,y_i) = s_i^{-1}(x_{i-1},y_{i-1})$ for all i, $1 \le i \le n$. Then f is valid on (x,y) if s_i is valid on (x_{i-1},y_{i-1}) for all i.

If $s_{i-1} = A_e$ with $e > 0$, then we have $y_{i-1} \le x_{i-1}/2$, so s_i is valid on (x_{i-1},y_{i-1}) by Theorem 3.3. If $s_{i-1} = A_0$, we have $y_{i-1} \le x_{i-1}$, but since we know $s_i = A_e$ with $e > 0$, s_i is valid on (x_{i-1},y_{i-1}) by Theorem 3.3.

Thus, if s_1 is valid on (x, y), so is f. $\square$

4. Steps Performed on Worst-Case Inputs

The main result of this section implies that, on worst-case inputs, the LSBGCD algorithm performs only steps A_0 and A_1. Before we proceed, we need some technical lemmas.

We say that (a, b) is *smaller* than (c, d) if both $a \le c$ and $a + b \le c + d$ are true. If this is the case, we write $(a, b) \le (c, d)$. Note that this relation is transitive, and implies lexicographic order.

LEMMA 4.1. *For any pair (x, y) where $x > 0$ and $x \ge y$, the following are true:*

(1) If $y \le 2^{e-1}x$ then $A_e^{-1}(x, y) \le B_e^{-1}(x, y)$.
(2) If $e \le e'$ then $A_e^{-1}(x, y) \le A_{e'}^{-1}(x, y)$.
(3) If $e > 1$ and $x \le 2^{e-1}y$ then $A_1^{-1}(x, y) \le (IA_e)^{-1}(x, y)$.

PROOF. Let (x, y) be as stated above.

(1) We need to show that $(2^e x + y, x) \le (2^{e+1}x - y, x)$. But this is true since $(2^{e+1}x - y) - (2^e x + y) = 2^e x - 2y \ge 0$.
(2) We need to show that $(2^e x + y, x) \le (2^{e'}x + y, x)$, which is obvious.
(3) We must show that
$$A_1^{-1}(x, y) = (2x + y, x) \le (IA_e)^{-1}(x, y) = (2^e y + x, y).$$
We have
$$2^e y + x = 2(2^{e-1}y) + x \ge 2^{e-1}y + 2x \ge 2x + y$$
and similarly
$$(2^e y + x) + (y) = 2(2^{e-1}y) + x + y \ge 3x + y = (2x + y) + (x).$$

$\square$

LEMMA 4.2. *Let g be a partial computation consisting entirely of A_e steps, with no interchange transformations (I's). If $(x_1, y_1) \le (x_2, y_2)$ then $g^{-1}(x_1, y_1) \le g^{-1}(x_2, y_2)$.*

PROOF. Let $(x_1, y_1) \le (x_2, y_2)$ and let n be the length of g. We use induction on n. If $n = 0$ there is nothing to prove, so assume $n > 0$. Then $g = fA_e$ where f is a partial computation of length $n - 1$. Let $(w_i, z_i) = f^{-1}(x_i, y_i)$ for $i = 1, 2$. By the inductive hypothesis, we have $f^{-1}(x_1, y_1) \le f^{-1}(x_2, y_2)$ and so $(w_1, z_1) \le (w_2, z_2)$. It remains to show that $A_e^{-1}(w_1, z_1) \le A_e^{-1}(w_2, z_2)$. But this is true because we have $A_e^{-1}(w_1, z_1) = (2^e w_1 + z_1, w_1)$ and $A_e^{-1}(w_2, z_2) = (2^e w_2 + z_2, w_2)$, which imply $2^e w_1 + z_1 = (2^e - 1)w_1 + (w_1 + z_1) \le (2^e - 1)w_2 + (w_2 + z_2) = 2^e w_2 + z_2$. $\square$

THEOREM 4.3. *Let g be a partial computation of length n, valid on the pair (x, y), with $x \ge y$. There is another partial computation f of length n, also valid on (x, y), where f consists entirely of A_0 and A_1 computations, and $f^{-1}(x, y) \le g^{-1}(x, y)$.*

PROOF. Let g and (x, y) be as stated above. Let m denote the number of steps in g that are not A_0 or A_1. We prove the theorem using induction on m. There is nothing to prove if $m = 0$, so assume $m > 0$. Then we can write

$$g = rst$$

where t is a partial computation containing only A_0 and A_1 steps, s is one step that is not A_0 or A_1, and r is a partial computation having at most $m - 1$ steps that are not A_0 or A_1.

If $s = B_0$, let $h = A_0t$, and otherwise let $h = A_1t$. If we can show rh is valid on (x, y) and that $(rh)^{-1}(x, y) \leq g^{-1}(x, y)$, then by the inductive hypothesis we are done.

Let $(w, z) = r^{-1}(x, y)$. We know that s is valid on (w, z).

If $s = B_0$, by Theorem 3.3 we have $z \leq w/2$. Similarly, A_0 is valid on (w, z) too. In addition, since g is valid on (x, y), we know the first step of t (if t has positive length) must be A_1. Thus t is valid on $A_0^{-1}(w, z)$ by Lemma 3.5. Therefore, rh is valid on (x, y).

If $s \neq B_0$, we can only assume $z \leq w$. But by Theorem 3.3 we know A_1 is valid on (w, z). Again, by Lemma 3.5 t is valid on $A_1^{-1}(w, z)$. So rh is valid on (x, y).

All that remains to show is that $h^{-1}(w, z) \leq (st)^{-1}(w, z)$, but this follows from Lemma 4.2 (applied to t), the fact that $A_0^{-1}(w, z) \leq B_0^{-1}(w, z)$, and if $s \neq B_0$, $A_1^{-1}(w, z) \leq s^{-1}(w, z)$ by Lemma 4.1.

That completes the proof. □

5. A Theorem on Continuants

In this section we exhibit a relationship between the worst case of the LSBGCD algorithm and extremal values of certain continuants. We will assume that the reader is familiar with the elementary properties of continued fractions and continuants, as explained, for example, in Hardy and Wright (1985) and Knuth (1981). In particular, we use the standard abbreviation $[a_1, a_2, \ldots, a_n]$ to denote the continued fraction

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots + \cfrac{1}{a_n}}}. \tag{5.1}$$

As we have seen in Theorem 4.3, to determine the lexicographically smallest pair (x, y) that requires n while-loop iterations in the LSBGCD algorithm, it suffices to consider inputs that result in the algorithm performing only steps of the form A_0 and A_1. But it is easy to see that the transformations $A_0(u, v) = (v, u - v)$ and $A_1(u, v) = (v, u - 2v)$ are precisely the same transformations performed by the ordinary Euclidean algorithm when the partial quotients are 1 or 2, respectively. Furthermore, we have seen that the sequence of steps A_0A_0 is not valid for any pair (x, y); this corresponds to a restriction that the corresponding ordinary Euclidean algorithm cannot use two consecutive steps of partial quotient 1.

It follows that to determine the lexicographically least pair (x, y) with $x \geq y$, that causes the LSBGCD algorithm to perform n iterations, it suffices to determine

$$a_1, a_2, \ldots, a_n \in \{1, 2\}$$

such that $(a_i, a_{i+1}) \neq (1, 1)$ for $1 \leq i \leq n$, and (x, y) is lexicographically minimized, where

$$[a_1, a_2, \ldots, a_n] = \frac{x}{y}.$$

Now it is well-known (see, e.g., (Knuth, 1981, p. 340)) that the numerator and denominator of a continued fraction can be expressed as continuant polynomials in the partial quotients a_i. We have

$$[a_1, a_2, \ldots, a_n] = \frac{Q_n(a_1, a_2, \ldots, a_n)}{Q_{n-1}(a_2, a_3, \ldots, a_n)}.$$

Here Q_n represents a polynomial in n variables, defined as follows:

$$\begin{aligned} Q_0() &= 1 \\ Q_1(x_1) &= x_1 \\ Q_n(x_1, x_2, \ldots, x_n) &= x_n Q_{n-1}(x_1, x_2, \ldots, x_{n-1}) + Q_{n-2}(x_1, x_2, \ldots, x_{n-2}) \end{aligned} \tag{5.2}$$

for $n \geq 2$.

An important property of continuants is the following (Knuth, 1981, p. 340):

$$Q_n(x_1, x_2, \ldots, x_n) = Q_n(x_n, x_{n-1}, \ldots, x_1). \tag{5.3}$$

Let P_n denote the set of all permutations $(a_1, a_2, \ldots, a_n)$ such that $a_i \in \{1, 2\}$ for $1 \leq i \leq n$, and $(a_i, a_{i+1}) \neq (1, 1)$ for $1 \leq i \leq n-1$. Then it follows that we seek to minimize the continuant $Q_n(a_1, a_2, \ldots, a_n)$ over all elements $(a_1, a_2, \ldots, a_n)$ of P_n.†

To do this, we employ the following theorem of Ramharter (1983), which determines all the minimal values of $Q(c_1, c_2, \ldots, c_n)$, where the c_i form a permutation of a given list of (not necessarily distinct) integers:

THEOREM 5.1. (RAMHARTER, 1983) *Let $B = (b_1, b_2, \ldots, b_n)$ denote a list of positive integers such that $b_1 \leq b_2 \leq b_3 \leq \ldots \leq b_n$. Let P denote the set of all permutations of B. Then (up to reversal of the order of the arguments of Q_n) the continuant polynomial Q_n uniquely attains its minimum on P at $C = (c_1, c_2, \ldots, c_n)$, where*

$$C = \begin{cases} (b_1, b_{2i}, b_3, b_{2i-2}, \ldots, b_{2i-3}, b_4, b_{2i-1}, b_2), \\ \qquad \text{if } n = 2i; \\ (b_1, b_{4i-1}, b_3, b_{4i-3}, \ldots, b_{2i-1}, b_{2i+1}; b_{2i}, b_{2i+2}, \ldots, b_{4i-4}, b_4, b_{4i-2}, b_2), \\ \qquad \text{if } n = 4i-1; \\ (b_1, b_{4i+1}, b_3, b_{4i-1}, \ldots, b_{2i+3}, b_{2i+1}; b_{2i+2}, b_{2i}, \ldots, b_{4i-2}, b_4, b_{4i}, b_2), \\ \qquad \text{if } n = 4i+1. \end{cases}$$

(The semicolon above plays the same semantic role as the comma, namely to denote concatenation. We have used the semicolon, following Ramharter, to indicate a logical break in the ordering of the coefficients.)

Unfortunately, we cannot apply Ramharter's theorem immediately to our situation, since we do not yet know how many of the a_i are equal to 1 in the continuant that is minimal over all members of P_n. We now prove the following theorem:

† In fact, because of the identity (5.3), there will potentially be *two* minimal n-tuples, one of which is the reversal of the other.

THEOREM 5.2. *The continuant polynomial $Q_n(a_1, a_2, \ldots, a_n)$ attains its minimum over all elements $(a_1, a_2, \ldots, a_n) \in P_n$ precisely as described below:*

$$(a_1, a_2, \ldots, a_n) = \begin{cases} (\overbrace{1,2,1,2,\ldots,1,2,1}^{n}), & \textit{if } n \textit{ odd;} \\ (\overbrace{1,2,1,2,\ldots,1,2,}^{n/2}\ \overbrace{2,1,2,1,\ldots,2,1}^{n/2}), & \textit{if } n \equiv 0 \pmod 4; \\ (\overbrace{1,2,1,2,\ldots,1,2,}^{n/2\pm1}\ \overbrace{2,1,2,1,\ldots,2,1}^{n/2\mp1}), & \textit{if } n \equiv 2 \pmod 4. \end{cases}$$

We remark that the minimum is attained uniquely in the case $n \not\equiv 2 \pmod 4$. When $n \equiv 2 \pmod 4$, the minimum is attained at precisely two distinct elements of P_n, one of which is the reversal of the other.

PROOF. Let $A = (a_1, a_2, \ldots, a_n)$ be a permutation chosen from P_n that minimizes $Q_n(a_1, a_2, \ldots, a_n)$. Let j denote the number of a_i in A that are equal to 1. Define $R_{n,j}$ to be the set of all permutations of j 1's and $n - j$ 2's (with no restrictions on adjacency). Using the notation of Ramharter's theorem, define $b_1, b_2, \ldots, b_j = 1$ and $b_{j+1}, b_{j+2}, \ldots, b_n = 2$.

The proof of the theorem has three parts, corresponding to the cases (i) $j > \lfloor (n+1)/2 \rfloor$; (ii) $j < \lfloor (n+1)/2 \rfloor$; and (iii) $j = \lfloor (n+1)/2 \rfloor$.

Case (i): $j > \lfloor (n+1)/2 \rfloor$. In this case, it is easy to see that $P_n \cap R_{n,j} = \emptyset$, since any permutation P of n 1's and 2's containing more than $\lfloor (n+1)/2 \rfloor$ 1's must contain two consecutive 1's, a contradiction.

Case (ii): $j < \lfloor (n+1)/2 \rfloor$. Then it is easy to see that, by Ramharter's theorem, the n-tuples which minimize Q_n over all members of $R_{n,j}$ are in fact members of P_n. If $1 \le n \le 3$, then a short computation shows that no permutation in P_n containing fewer than $\lfloor (n+1)/2 \rfloor$ 1's can minimize Q_n. Hence we may assume $n > 3$.

An easy application of Ramharter's theorem shows the element $A = (a_1, a_2, \ldots, a_n)$ that minimizes Q_n over all members of $R_{n,j}$ (up to reversal of A) satisfies

$$\begin{cases} (a_k, a_{k+1}, a_{k+2}) = (b_{k+2}, b_{k+1}, b_k) = (2,2,2), & \text{if } n \equiv 0 \pmod 4; \\ (a_k, a_{k+1}, a_{k+2}) = (b_{k+3}, b_{k+1}, b_{k+2}) = (2,2,2), & \text{if } n \equiv 1 \pmod 4; \\ (a_k, a_{k+1}, a_{k+2}) = (b_k, b_{k+1}, b_{k+2}) = (2,2,2), & \text{if } n \equiv 2 \pmod 4; \\ (a_{k+1}, a_{k+2}, a_{k+3}) = (b_{k+2}, b_{k+1}, b_{k+3}) = (2,2,2), & \text{if } n \equiv 3 \pmod 4, \end{cases}$$

where $k = \lfloor n/2 \rfloor$.

But then we can replace the middle 2 in each of these cases by 1, obtaining a smaller continuant, which contradicts our assumption that A minimized Q_n over P_n.

Case (iii): $j = \lfloor (n+1)/2 \rfloor$. Now we can use Ramharter's theorem directly, and we conclude that the member(s) of $R_{n,\lfloor (n+1)/2 \rfloor}$ which minimize(s) Q_n is (are)

$$A = \begin{cases} (\overbrace{1,2,1,2,\ldots,1,2,1}^{n}), & \text{if } n \equiv 1,3 \pmod 4; \\ (\overbrace{1,2,1,2,\ldots,1,2}^{n/2},\ \overbrace{2,1,2,1,\ldots,2,1}^{n/2}), & \text{if } n \equiv 0 \pmod 4; \\ (\overbrace{1,2,1,2,\ldots,1,2}^{n/2\pm1},\ \overbrace{2,1,2,1,\ldots,2,1}^{n/2\mp1}), & \text{if } n \equiv 2 \pmod 4). \end{cases}$$

Note that $A \in P_n$ in each case. This completes the proof of Theorem 5.2. □

6. Proof of the Main Result

PROOF. Let (x_n, y_n) denote the lexicographically least pair that requires n iterations of the LSBGCD algorithm. We obtain a linear recurrence relation for the sequences $(x_n)_{n\geq0}$ and $(y_n)_{n\geq0}$. The proof consists of three cases, depending on $n \pmod 4$.

Case (i) $n \equiv 1 \pmod 2$: It follows immediately from Theorem 5.2 that if n is odd, then

$$x_n/y_n = [\,\overbrace{1,2,1,2,\ldots,1,2,1}^{n}\,].$$

Using standard techniques, we now deduce that $(x_1, y_1) = (1,1)$; $(x_3, y_3) = (4,3)$, and $(x_n, y_n) = (4x_{n-2} - x_{n-4}, 4y_{n-2} - y_{n-4})$ for n odd, $n \geq 5$.

Case (ii) $n \equiv 0 \pmod 4$: Then $x_n/y_n = [\,\overbrace{1,2,1,2,\ldots,1,2}^{n/2},\ \overbrace{2,1,2,1,\ldots,2,1}^{n/2}\,]$. In this case, we can obtain a recurrence relation for (x_n) and (y_n) using a trick from Shallit [1990], as follows: first, we observe that

$$\begin{bmatrix} x_1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x_2 & 1 \\ 1 & 0 \end{bmatrix}\cdots\begin{bmatrix} x_n & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} Q_n(x_1,\ldots,x_n) & Q_{n-1}(x_1,\ldots,x_{n-1}) \\ Q_{n-1}(x_2,\ldots,x_n) & Q_{n-2}(x_2,\ldots,x_{n-1}) \end{bmatrix}, \tag{6.1}$$

a fact which is easily proved by induction (Frame, 1949; Kolden, 1949).

Now define $M_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, and $M_2 = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}$, and $M(k) = (M_1M_2)^k(M_2M_1)^k$. Also define

$$M(k) = \begin{bmatrix} C_k & D_k \\ E_k & F_k \end{bmatrix}.$$

Then we find

$$\begin{aligned} M(k+1) &= M_1M_2M(k)M_2M_1 \\ &= \begin{bmatrix} 9C_k + 3D_k + 3E_k + F_k & 6C_k + 3D_k + 2E_k + F_k \\ 6C_k + 2D_k + 3E_k + F_k & 4C_k + 2D_k + 2E_k + F_k \end{bmatrix}; \\ M(k+2) &= M_1M_2M(k+1)M_2M_1 \\ &= \begin{bmatrix} 121C_k + 44D_k + 44E_k + 16F_k & 88C_k + 33D_k + 32E_k + 12F_k \\ 88C_k + 32D_k + 33E_k + 12F_k & 64C_k + 24D_k + 24E_k + 9F_k \end{bmatrix}; \\ M(k+3) &= M_1M_2M(k+2)M_2M_1 \\ &= \begin{bmatrix} 1681C_k + 615D_k + 615E_k + 225F_k & 1230C_k + 451D_k + 450E_k + 165F_k \\ 1230C_k + 450D_k + 451E_k + 165F_k & 900C_k + 330D_k + 330E_k + 121F_k \end{bmatrix}. \end{aligned}$$

It is now easy to verify that $M(k+3) = 15M(k+2) - 15M(k+1) + M(k)$; it follows that each of the sequences $(C_k), (D_k), (E_k), (F_k)$ are linear recurrences with characteristic polynomial $z^3 - 15z^2 + 15z - 1 = (z-1)(z^2 - 14z + 1)$. It follows that $(x_4, y_4) = (10, 7)$; $(x_8, y_8) = (137, 100)$; $(x_{12}, y_{12}) = (1906, 1395)$; and $(x_n, y_n) = (15x_{n-4} - 15x_{n-8} + x_{n-12}, 15y_{n-4} - 15y_{n-8} + y_{n-12})$ for $n \equiv 0 \pmod 4$, $n \geq 16$.

Case (iii) $n \equiv 2 \pmod 4$: This case is slightly more difficult than the preceding cases, for now Theorem 5.2 tells us that there are *two* distinct minimal continuants. These two continuants correspond to two distinct candidates for the lexicographically least pair (x_n, y_n) for each n with $n \equiv 2 \pmod 4$. Define

$$x'_n/y'_n = [\,\overbrace{1,2,1,2,\ldots,1,2}^{n/2-1},\ \overbrace{2,1,2,1,\ldots,2,1}^{n/2+1}\,]$$

and

$$x''_n/y''_n = [\,\overbrace{1,2,1,2,\ldots,1,2}^{n/2+1},\ \overbrace{2,1,2,1,\ldots,2,1}^{n/2-1}\,].$$

We know that $x_n = x'_n = x''_n$; to determine which is in fact the lexicographically least pair (x_n, y_n), we must determine which of the inequalities $y'_n < y''_n$ or $y'_n > y''_n$ is true.

As in the previous case, define $M_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$; $M_2 = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}$; and

$$N(k) = (M_1M_2)^k(M_2M_1)^{k+1} = \begin{bmatrix} R_k & S_k \\ T_k & U_k \end{bmatrix}. \tag{6.2}$$

It follows that $x'_{4k+2} = x''_{4k+2} = R_k$ and $y'_{4k+2} = T_k$. By taking the transpose of equation (6.2), we see that

$$N(k)^T = (M_1M_2)^{k+1}(M_2M_1)^k = \begin{bmatrix} R_k & T_k \\ S_k & U_k \end{bmatrix},$$

and hence $y''_{4k+2} = S_k$.

Now we prove by induction that $S_k - T_k = 1$ for all $k \geq 0$. Clearly this is true for $k = 0$, for then

$$N(0) = M_2M_1 = \begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix},$$

and hence $S_0 = 2$ and $T_0 = 1$. Now assume that $S_k - T_k = 1$ for all $k' \leq k$; we prove it for $k+1$. We find

$$N(k+1) = M_1M_2N(k)M_2M_1 = \begin{bmatrix} 9R_k + 3S_k + 3T_k + U_k & 6R_k + 3S_k + 2T_k + U_k \\ 6R_k + 2S_k + 3T_k + U_k & 4R_k + 2S_k + 2T_k + U_k \end{bmatrix}.$$

It follows that $S_{k+1} - T_{k+1} = (6R_k + 3S_k + 2T_k + U_k) - (6R_k + 2S_k + 3T_k + U_k) = S_k - T_k = 1$, by induction. This completes the proof that $S_k - T_k = 1$ for all $k \geq 0$.

It now follows that $y'_{4k+2} < y''_{4k+2}$. Thus we have established that, for $n \equiv 2 \pmod 4$,

$$x_n/y_n = [\,\overbrace{1,2,1,2,\ldots,1,2}^{n/2-1},\ \overbrace{2,1,2,1,\ldots,2,1}^{n/2+1}\,].$$

We can obtain a recurrence relation for (x_n, y_n) as before. Omitting the calculations, it follows that $(x_2, y_2) = (3, 1)$; $(x_6, y_6) = (37, 26)$; $(x_{10}, y_{10}) = (511, 373)$; and

$$(x_n, y_n) = (15x_{n-4} - 15x_{n-8} + x_{n-12}, 15y_{n-4} - 15y_{n-8} + y_{n-12})$$

for $n \equiv 2 \pmod 4$, $n \geq 14$.

Let S be the shift operator. It follows that $(S^{12} - 15S^8 + 15S^4 - 1)(S^4 - 4S^2 + 1)$ annhilates both the sequences $(x_n)_{n \geq 0}$ and $(y_n)_{n \geq 0}$. We can now use standard techniques to express $(x_n)_{n \geq 0}$ and $(y_n)_{n \geq 0}$ in terms of the roots of the associated characteristic polynomials. This provides the description given in Theorem 1.1. □

In this paper, we determined the exact worst-case behavior of the LSBGCD algorithm under lexicographic ordering. It may also be of interest to examine the algorithm's performance under other orderings. For example, numerical evidence suggests (and it should not be terribly difficult to prove) that the pair (u, v) with $u \geq v > 0$ that minimizes $u^2 + v^2$ and forces LSBGCD to perform n iterations is (p_{n-1}, p_{n-2}), where p_n denotes the numerator of the n'th convergent to $\sqrt{3}$. (The sequence (p_n) satisifies the linear recurrence $p_0 = 1$, $p_1 = 3$, and $p_{2k} = p_{2k-1} + p_{2k-2}$, $p_{2k+1} = 2p_{2k} + p_{2k-1}$ for $k \geq 1$.)

It would be of interest to obtain similar results for other algorithms; for example, the algorithm of Purdy (1983).

7. Remarks on Implementation

The algorithms we have mentioned in this paper (Euclid's algorithm; Euclid's algorithm with least remainder; the left- and right- shift binary algorithms, and LSBGCD) all use at most $O(\log uv)$ iterations on input (u, v), and hence all are quite efficient. Deciding which is most efficient in practice is fraught with difficulty, since the answer depends fundamentally on the relative speed of the underlying extended-precision arithmetic routines. The reader may find it useful to compare the conclusions in the work of Sorenson (1991, 1994) and Jebelean (1993a, 1993b).

We can compare the worst-case behaviour of LSBGCD to more traditional algorithms as follows. It follows from well-known results (Lamé, 1844) that if $u \geq v > 0$ and (u, v) is the lexicographically least input that causes Euclid's algorithm to perform n division steps, then $u = F_{n+2}$, the $(n + 2)$nd Fibonacci number. Thus $u \sim \tau^{n+2}/\sqrt{5}$, where $\tau = (1 + \sqrt{5})/2$. Since $1/(\log_2 \tau) \doteq 1.441$, it follows that Euclid's algorithm performs at most $1.441 \log_2 u + O(1)$ division steps on input (u, v) with $u \geq v > 0$.

Similarly, since $1/(\log_2 \sqrt{\alpha}) \doteq 1.053$, it follows from Theorem 1.1 that the LSBGCD algorithm performs at most $1.053 \log_2 u + O(1)$ iterations on input (u, v) with $u \geq v > 0$. Thus, LSBGCD performs about 27% fewer iterations than Euclid's algorithm in the worst-case.

In the average case, we noticed experimentally that LSBGCD performs about 15% more iterations of its main loop than does Euclid's algorithm. However, since each loop iteration can be performed using only shifts and subtractions, on most computers the LSBGCD outperforms the Euclidean algorithm, and is competitive with the binary algorithm.

In Table 2 we give timing results for four GCD algorithms: the binary algorithm, Brent's left-shift binary algorithm, Euclid's algorithm, Euclid's algorithm with least remainder, and LSBGCD. 100 pseudo-random numbers of each of four digit sizes were used as inputs. The times given in the table are averages. A 486/33 CompuAdd PC running MS-DOS version 6 was used, and the algorithms were implemented in Turbo C++. In Table 3 we give the average number of main loop iterations performed by each algorithm using the same inputs as in Table 2. Other results are presented in Sorenson (1994).

The LSBGCD algorithm is uniquely suited for extended GCD computation; it uses

Table 2. Average Running Times in CPU Seconds

Algorithm	Input Size in Decimal Digits			
	100	250	500	1000
Binary	0.049	0.201	0.667	2.39
Brent's	0.050	0.215	0.736	2.70
Euclidean	0.074	0.357	1.28	4.82
LR Euclidean	0.066	0.319	1.146	4.30
LSBGCD	0.050	0.214	0.725	2.64

Table 3. Average Number of Main Loop Iterations

Algorithm	Input Size in Decimal Digits			
	100	250	500	1000
Binary	467	1168	2345	4690
Brent's	290	726	1455	2905
Euclidean	194	486	972	1938
LR Euclidean	135	339	677	1347
LSBGCD	219	551	1101	2201

only left shifts and subtractions, and has a relatively low number of iterations. Nice (1993) implemented five different extended GCD algorithms, including the Euclidean and binary algorithms, and found LSBGCD to be the fastest.

8. Acknowledgments

We are most grateful to Prof. C. P. Schnorr, who ignited our interest in the subject when he told the first author about Brent's left-shift binary GCD algorithm in 1988. Robert Black read an early version of this paper and made several useful suggestions. Heinrich Rolletschek read a draft of this paper very carefully and helped us clarify our arguments. We are also grateful to the referees for their valuable comments.

References

Brent, R. P. (1976). Analysis of the binary Euclidean algorithm. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 321–355. Academic Press, New York.

Collins, G. E. (1974). The computing time of the Euclidean algorithm. *SIAM J. Comput.* **3**, 1–10.

Dupré, A. (1846). Sur le nombre de divisions à effectuer pour obtenir le plus grand commun diviseur entre deux nombres entiers. *J. Math. Pures Appl.* **11**, 41–64.

Frame, J. S. (1949). Continued fractions and matrices. *Amer. Math. Monthly* **56**, 98–103.

Hardy, G. H. and Wright, E. M. (1985). *An Introduction to the Theory of Numbers.* Oxford University Press, 5th edition.

Jebelean, T. (1993a). Comparing several GCD algorithms. In E. Swartzlander, Jr., M. J. Irwin, and G. Jullien, editors, *Proc. 11th Symposium on Computer Arithmetic*, pages 180–185. IEEE Computer Society Press, Los Alamitos, California.

Jebelean, T. (1993b). A generalization of the binary GCD algorithm. In M. Bronstein, editor, *ISSAC '93: Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation*, pages 111–116. ACM Press, New York.

Knuth, D. E. (1981). *The Art of Computer Programming. Volume 2: Seminumerical Algorithms.* Addison-Wesley. 2nd edition.

Kolden, K. (1949). Continued fractions and linear substitutions. *Archiv for Mathematik og Naturvidenskab* **50**, 141–196.

Lamé, G. (1844). Note sur la limite du nombre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers. *C. R. Acad. Sci. Paris* **19**, 867–870.

Nice, B. (1993). Extended greatest common divisors. Manuscript, 1993.

Norton, G. H. (1989). Precise analyses of the right- and left-shift greatest common divisor algorithms for $GF(q)[x]$. *SIAM J. Comput.* **18**, 608–624.

Purdy, G. B. (1983). A carry-free algorithm for finding the greatest common divisor of two integers. *Comput. Math. with Appl.* **9**, 311–316.

Ramharter, G. (1983). Extremal values of continuants. *Proc. Amer. Math. Soc.* **89**, 189–201.

Shallit, J. O. (1990). On the worst case of three algorithms for computing the Jacobi symbol. *J. Symbolic Comput.* **10**, 593–610.

Shallit, J. O., and Sorenson, J. P. (1993). A binary algorithm for the Jacobi symbol. *ACM SIGSAM Bull.* **27** (1), 4–11.

Sorenson, J. P. (1991). *Algorithms in Number Theory*. PhD thesis, University of Wisconsin, Madison, June 1991. Available as Computer Sciences Technical Report No. 1027.

Sorenson, J. P. (1994). Two fast GCD algorithms. *J. Algorithms* **16**, 110–144.

Stein, J. (1967). Computational problems associated with Racah algebra. *J. Comput. Phys.* **1**, 397–405.