J. Symbolic Computation (1990) **10**, 593-610

On the Worst Case of Three Algorithms for Computing the Jacobi Symbol

JEFFREY SHALLIT[†]

Department of Mathematics and Computer Science
Dartmouth College
Hanover, NH 03755
USA
shallit@dartmouth.edu

(Received 7 February 1989)

We study the worst-case behavior of three iterative algorithms for computing the Jacobi symbol $\left(\frac{v}{u}\right)$. Each algorithm is similar in format to the Euclidean algorithm for computing $\gcd(u, v)$.

Eisenstein's algorithm chooses an even quotient at each step. It is shown that the worst case occurs when $u = 2n + 1$, $v = 2n - 1$.

Lebesgue's algorithm is essentially the least-remainder Euclidean algorithm with powers of 2 removed at each step. Its worst case occurs when $u = 2L_n - L_{n-1}$, $v = L_n$, where $L_0 = 1$, $L_1 = 1$, and $L_n = 2L_{n-1} + L_{n-2}$ for $n \geq 2$.

The "ordinary" Jacobi symbol algorithm is essentially the ordinary Euclidean algorithm with powers of 2 removed at each step. It is the most interesting mathematically of the three. We prove that if the ordinary algorithm on input (u, v) performs n division steps, with $u > v > 0$ and $u + v$ as small as possible, then $u = A_n$ and $v = A_{n-1}$, where $A_0 = 1$, $A_1 = 3$, $A_{2n} = A_{2n-1} + 2A_{2n-2}$ for $n \geq 1$, and $A_{2n+1} = 2A_{2n} + A_{2n-1}$ for $n \geq 1$.

We also discuss the worst-case inputs to the ordinary algorithm under the lexicographic and reverse lexicographic orderings.

1. Introduction.

Interest in efficient computation of the Jacobi symbol was reawakened in 1977 with the publication of a randomized primality test (Solovay & Strassen, 1977). Despite this, there seems to have been little systematic investigation of the worst-case behavior of the classical algorithms for computation of the Jacobi symbol.

In this paper, we discuss the worst-case behavior of three well-known algorithms, each similar to the Euclidean algorithm, for computing the Jacobi symbol $\left(\frac{v}{u}\right)$.

Recall that the ordinary Euclidean algorithm computes $\gcd(u_0, u_1)$ by doing a series of divisions with remainder:

[†] Research supported by NSF grant CCR-8817400.

$$\begin{aligned} u_0 &= a_0 u_1 + u_2 \\ u_1 &= a_1 u_2 + u_3 \\ &\vdots \\ u_{n-1} &= a_{n-1} u_n. \end{aligned}$$

We say that n is the number of *division steps* in the algorithm. Lamé (1844) proved that the worst case of this algorithm occurs when the inputs are consecutive Fibonacci numbers. More precisely, letting $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$, we have (Knuth, 1981, p. 343):

Theorem 1.1. *Let $u > v > 0$ be such that the Euclidean algorithm on inputs (u, v) performs n division steps, and u is as small as possible. Then $u = F_{n+2}$ and $v = F_{n+1}$.*

Corollary. *On inputs (u, v), $u > v > 0$, the Euclidean algorithm performs no more than*

$$2.08 \log u - .32 + .93u^{-1}$$

division steps.

Another method of computing the greatest common divisor is the *least-remainder algorithm*. Again, we do a series of divisions with remainder

$$\begin{aligned} u_0 &= a_0 u_1 + \epsilon_1 u_2 \\ u_1 &= a_1 u_2 + \epsilon_2 u_3 \\ &\vdots \\ u_{n-1} &= a_{n-1} u_n, \end{aligned}$$

but now we choose $\epsilon_i u_{i+1}$ to be the *absolutely least* residue and $u_{i+1} > 0$, $\epsilon_i = \pm 1$. This algorithm was analyzed by Dupré (1846). Letting $D_0 = 0$, $D_1 = 1$, and $D_n = 2D_{n-1} + D_{n-2}$ for $n \geq 2$, we have (Knuth, 1981, exercise 4.5.30)

Theorem 1.2. *Let $u \geq v > 0$ be such that the least-remainder algorithm on inputs (u, v) performs n division steps, and u is as small as possible. Then $u = D_n + D_{n-1}$ and $v = D_n$.*

Corollary. *On inputs (u, v), $u \geq v > 0$, the least-remainder algorithm performs no more than*

$$1.14 \log u + .79 + .41u^{-1}$$

division steps.

The Jacobi symbol $\left(\frac{v}{u}\right)$ is defined for integers v and positive odd integers u. It can be computed using the following identities (Jacobi, 1846):

$$\left(\frac{v}{u}\right) = (-1)^{(u-1)(v-1)/4} \left(\frac{u \bmod v}{v}\right), \quad u, v \text{ odd and positive;} \tag{1}$$

$$\left(\frac{2}{u}\right) = (-1)^{(u^2-1)/8}; \tag{2}$$

$$\left(\frac{a}{u}\right) = a^{(u-1)/2} \text{ for } |a| \leq 1; \tag{3}$$

$$\left(\frac{vw}{u}\right) = \left(\frac{v}{u}\right)\left(\frac{w}{u}\right). \tag{4}$$

Equation (1) shows that we can use a division with remainder to compute the Jacobi symbol, while equations (2)–(4) show how to remove powers of 2 or -1, if necessary, to keep the upper entry of the symbol odd and positive.

In the descriptions that follow, we assume $v > 0$. If $v \leq 0$, we can use equation (3) above.

Eisenstein's algorithm

Eisenstein (1844) proposed the following algorithm for computing $(\frac{v}{u})$: let $u = u_0$ and $v = u_1$ be odd and write

$$\begin{aligned} u_0 &= a_0 u_1 + \epsilon_2 u_2 \\ u_1 &= a_1 u_2 + \epsilon_3 u_3 \\ &\vdots \\ u_{n-1} &= a_{n-1} u_n. \end{aligned}$$

Here $\epsilon_i = \pm 1$ are chosen so that $u_i > 0$, and each a_i, except possibly a_{n-1}, is *even*. More formally, if $q = u_i/u_{i+1}$ is an integer, then $a_i = q$; otherwise $a_i = \lfloor q \rfloor$ or $\lceil q \rceil$, whichever is even.

Then

$$\left(\frac{v}{u}\right) = \begin{cases} 0, & \text{if } u_n > 1; \\ (-1)^r, & \text{if } u_n = 1, \end{cases}$$

where

$$r = \sum_{0 \leq i \leq n-2} \left(\frac{(u_i - 1)(u_{i+1} - 1)}{4} + \left(\frac{u_{i+1} - 1}{2}\right)\left(\frac{1 - \epsilon_{i+2}}{2}\right) \right).$$

Eisenstein's work is summarized by Smith (1965, § 23).

Lebesgue's algorithm

Lebesgue (1847) proposed a different algorithm, similar to the least-remainder Euclidean algorithm, except that powers of 2 are removed at each step to ensure that the next u_i is always odd. Let $u = u_0$ and $v = 2^{e_1} u_1$. Then write

$$\begin{aligned} u_0 &= a_0 u_1 + \epsilon_2 2^{e_2} u_2 \\ u_1 &= a_1 u_2 + \epsilon_3 2^{e_3} u_3 \\ &\vdots \\ u_{n-1} &= a_{n-1} u_n. \end{aligned}$$

Here $u_i > 0$ is always odd and $\epsilon_i = \pm 1$. The quotient a_i is chosen so that $2^{e_{i+2}} u_{i+2} < u_{i+1}/2$.

Then

$$\left(\frac{v}{u}\right) = \begin{cases} 0, & \text{if } u_n > 1; \\ (-1)^r, & \text{if } u_n = 1, \end{cases}$$

where

$$r = \left(\sum_{0 \le i \le n-1} e_{i+1} \frac{u_i^2 - 1}{8} \right) + \sum_{0 \le i \le n-2} \left(\frac{(u_i - 1)(u_{i+1} - 1)}{4} + \left(\frac{u_{i+1} - 1}{2}\right)\left(\frac{1 - \epsilon_{i+2}}{2}\right) \right).$$

The ordinary algorithm

Finally, there is a third algorithm for computing the Jacobi symbol which is similar to the ordinary Euclidean algorithm. Positive remainders are chosen at each step, but again powers of 2 are removed to ensure that each succeeding u_i is odd. We call this algorithm the "ordinary" Jacobi symbol algorithm.

While this algorithm is implicit in many elementary texts on number theory (e. g. (LeVeque, 1977, p. 112), (Rosen, 1984, pp. 319-320)), the earliest *explicit* mention we have been able to find is Williams (1980). Collins and Loos (1982) analyzed the ordinary algorithm and produced a bad case (but not the *worst* case!). The focus of their paper was slightly different, however: while they counted the number of bit operations, we count the number of division steps.

The ordinary algorithm also deserves attention as one that seems to be used frequently in practice (e. g. (Angluin, 1982), (Riesel, 1985)). Gaston Gonnet informs me (personal communication) that the ordinary algorithm is the one currently used for computing $(\frac{v}{u})$ by the computer algebra system Maple. Dan Grayson informs me (personal communication) that the ordinary algorithm is also used in the Mathematica system.

On input (u, v), we let $u_0 = u$ and $2^{e_1} u_1 = v$ and then write

$$\begin{aligned} u_0 &= a_0 u_1 + 2^{e_2} u_2 \\ u_1 &= a_1 u_2 + 2^{e_3} u_3 \\ &\vdots \\ u_{n-1} &= a_{n-1} u_n. \end{aligned}$$

The e_i are chosen such that the u_i are all odd. Formally, we have $a_i = \lfloor u_i / u_{i+1} \rfloor$ and $e_{i+2} = \nu_2(u_i - a_i u_{i+1})$, where $\nu_2(n)$ is the exponent of the highest power of 2 that divides n. Then

$$\left(\frac{v}{u}\right) = \begin{cases} 0, & \text{if } u_n > 1; \\ (-1)^r, & \text{if } u_n = 1, \end{cases}$$

where

$$r = \sum_{0 \le i \le n-1} \left(e_{i+1} \frac{u_i^2 - 1}{8} + \frac{(u_i - 1)(u_{i+1} - 1)}{4} \right).$$

In this paper, we analyze the worst-case complexity of Eisenstein's algorithm, Lebesgue's algorithm, and the ordinary algorithm. Eisenstein's algorithm and Lebesgue's algorithm are both easy to analyze, and the results appear in sections 2 and 3. The behavior of the ordinary algorithm is much more complicated, and it is discussed in sections 4, 5, and 6. The main results of the paper are contained in these sections; in particular, see Theorem 5.1 and Lemma 4.6.

2. Eisenstein's algorithm.

In this section, we show that the worst-case behavior of Eisenstein's algorithm is actually quite bad. Theorem 2.2 below seems to be a "folk theorem" and was first shown to the author by Eric Bach.

Lemma 2.1. *Let $u > v > 0$, u, v odd, be such that Eisenstein's algorithm performs n division steps in computing $\left(\frac{v}{u}\right)$. Then $u \geq 2n+1$ and $v \geq 2n-1$.*

Proof. Let u_i denote the sequence of terms in Eisenstein's algorithm, where $u_0 = u$ and $u_1 = v$. Then clearly $u_0 > u_1 > \ldots > u_n \geq 1$ and all the u_i are odd.

Theorem 2.2. *Let $u > v > 0$, u, v odd, be such that Eisenstein's algorithm performs n division steps $\left(\frac{v}{u}\right)$ and u is as small as possible. Then $u = 2n+1$, and $v = 2n-1$.*

Proof. By the Lemma, $u \geq 2n+1$ and $v \geq 2n-1$. To complete the proof it suffices to show that on input $(u, v) = (2n+1, 2n-1)$, Eisenstein's algorithm takes exactly n steps. This is left to the reader.

3. Lebesgue's algorithm.

Define $L_0 = 1$, $L_1 = 1$, and $L_n = 2L_{n-1} + L_{n-2}$ for $n \geq 2$. It is easy to prove by induction that

$$L_n = \frac{(1+\sqrt{2})^n + (1-\sqrt{2})^n}{2}.$$

Lemma 3.1. *Suppose $u \geq 2v > 0$, u odd, and Lebesgue's algorithm performs n division steps in computing $\left(\frac{v}{u}\right)$. Then $u \geq L_{n+1}$ and $v \geq L_n$.*

Proof. By induction on n. It is easily verified for $n = 1$. Now assume it is true for all $m < n$; we wish to prove it for $m = n$.

Without loss of generality we may assume v is odd. Then the first division step writes $u_0 = a_0 u_1 + \epsilon_2 2^{e_2} u_2$. Since $u_0/u_1 \geq 2$, we have $a_0 \geq 2$. If $a_0 = 2$, then parity considerations show $e_2 = 0$ and $\epsilon_2 = +1$. If $a_0 = 3$, then since $2^{e_2} u_2 \leq u_1/2$, we have $u_0 \geq 2u_1 + u_2$. If $a_0 \geq 4$, then the same inequality holds. Thus in all cases we have $u_0 \geq 2u_1 + u_2$. Now $u_2 \leq u_1/2$, so the induction hypothesis applies and we have $u_1 \geq L_n$ and $u_2 \geq L_{n-1}$. Hence $u_0 \geq 2L_n + L_{n-1} = L_{n+1}$ and the proof is complete.

Lemma 3.2. *Suppose $v \le u < 2v$, u odd, and Lebesgue's algorithm performs n division steps in computing $\left(\frac{v}{u}\right)$. Then $u \ge 2L_n - L_{n-1}$ and $v \ge L_n$.*

Proof. The lemma is easily verified for $n = 1, 2$. Suppose the first two steps of Lebesgue's algorithm are

$$u_0 = a_0 u_1 + \epsilon_2 2^{e_2} u_2;$$

$$u_1 = a_1 u_2 + \epsilon_3 2^{e_3} u_3.$$

Since $u_1 \le u_0 < 2u_1$, either $a_0 = 1$ or $a_0 = 2$.

Case (a): $a_0 = 1$. In this case we have $u_0 = u_1 + 2^{e_2}u_2$. Parity considerations show that $e_2 \ge 1$. Hence $u_0 \ge u_1 + 2u_2$. But $2^{e_2}u_2 \le u_1/2$ because the least remainder is chosen at each step; hence $2u_2 \le u_1/2$ and $u_0 \ge 6u_2$. But $u_1/u_2 \ge 2$, so Lemma 3.1 applies and we have $u_1 \ge L_n$ and $u_2 \ge L_{n-1}$. Then $u_0 \ge 6L_{n-1} \ge 3L_{n-1} + 2L_{n-2} = 2L_n - L_{n-1}$, and we are done.

Case (b): $a_0 = 2$. In this case we have $u_0 = 2u_1 - u_2$, $u_1 = a_1 u_2 + \epsilon_3 2^{e_3} u_3$. Now $u_3 \le u_2/2$, so Lemma 3.1 applies to (u_2, u_3) and we find $u_2 \ge L_{n-1}$, $u_3 \ge L_{n-2}$.

If $a_1 = 2$, then $u_1 = 2u_2 + u_3 \ge 2L_{n-1} + L_{n-2} = L_n$, and $u_0 = 2u_1 - u_2 = 3u_2 + 2u_3 \ge 3L_{n-1} + 2L_{n-2} = 2L_n - L_{n-1}$, as was to be shown.

If $a_1 \ge 3$, then $u_1 \ge (5/2)u_2 \ge L_{n-1}$ and therefore $u_0 \ge 4u_2 \ge 4L_{n-1} \ge 3L_{n-1} + 2L_{n-2} = 2L_n - L_{n-1}$. This completes the proof.

Theorem 3.3. *Let $u > v > 0$, u odd, be such that Lebesgue's algorithm performs n division steps in computing $\left(\frac{v}{u}\right)$, and u is as small as possible. Then $u = 2L_n - L_{n-1}$ and $v = L_n$.*

Proof. We may assume $u < 2v$. For if $u \ge 2v$, we write $u = av + \epsilon 2^e r$ as the first step of the algorithm, and $a \ge 2$. There are two cases: $a = 2$ and $a \ge 3$.

If $a = 2$, then the first step in the algorithm for (u, v) must be $u = 2v + r$; the algorithm continues with (v, r). Then the algorithm on input $(v + 2r, v)$ takes the same number of steps as on input (u, v), and $v + 2r < u$, a contradiction.

If $a \ge 3$, then set $u' = (a-2)v + 2^e r$. A similar argument shows that the algorithm on input (u', v) takes the same number of steps as on input (u, v). But $u' < u$, a contradiction.

Hence we may assume $u < 2v$. By Lemma 3.2 we have $u \ge 2L_n - L_{n-1}$ and $v \ge L_n$. To complete the proof it suffices to show that Lebesgue's algorithm actually performs n division steps on input $(u, v) = (2L_n - L_{n-1}, L_n)$. This is left to the reader.

Corollary. *On inputs (u, v), $u > v > 0$, Lebesgue's algorithm performs no more than*

$$1.14 \log u + .27 + 2.27u^{-1}$$

division steps.

4. The ordinary algorithm: preliminary analysis.

In this section we will explore the worst-case complexity of the ordinary Jacobi symbol algorithm. As we will see, the behavior of the ordinary algorithm is fundamentally more complicated than that of the algorithms of Eisenstein and Lebesgue.

For each n, we are interested in the "smallest" pair (u, v) such that the ordinary algorithm performs n division steps. For the other algorithms in this paper, it was not really necessary to discuss what was meant by "smallest", since for $u > v > 0$, the input (u, v) that minimized u *also* minimized v, and such an input was *unique*.

Unfortunately, the state of affairs is more complicated for the ordinary algorithm. Indeed, if we search for inputs (u, v) that require 7 steps and minimize v, we find $v = 105$, which occurs in the pair $(269, 105)$. However, the corresponding input that minimizes u is $u = 259$, which occurs in the pair $(259, 141)$. Furthermore, the pair $(259, 145)$ also requires 7 steps! Thus we see that for the ordinary algorithm, the "worst case" depends strongly on our choice of ordering of the inputs, which was not the case for the other algorithms.

This problem suggests searching for a more "natural" ordering of the inputs—one for which the ordinary algorithm is well-behaved. It turns out that $u + v$ is such an ordering; more precisely, we show that if $u > v > 0$ are such that the ordinary algorithm requires n steps and $u + v$ is as small as possible, then $u = A_n$ and $v = A_{n-1}$, for a certain linear recurrence A_n.

Later, in section 6, we will discuss the worst-case inputs under lexicographic and reverse lexicographic orderings.

Definition. *Let* $A_{-2} = 0$, $A_{-1} = 1$, $A_{2n} = A_{2n-1} + 2A_{2n-2}$ *for* $n \geq 0$, *and* $A_{2n+1} = 2A_{2n} + A_{2n-1}$ *for* $n \geq 0$.

Here is a brief table of the A_i:

$n =$	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	...
$A_n =$	0	1	1	3	5	13	23	59	105	269	479	1227	2185	5597	9967	...

In section 5, we will prove the following

Theorem 5.1. *Let* $u > v > 0$, *u odd, be such that the ordinary Jacobi symbol algorithm to compute* $\left(\frac{v}{u}\right)$ *performs n division steps, and* $u + v$ *is as small as possible. Then* $u = A_n$ *and* $v = A_{n-1}$.

(The intuition behind the choice of the A_i is as follows, arguing informally: we try to choose A_n minimal such that one step of the algorithm leads from (A_{n+1}, A_n) to (A_n, A_{n-1}). This suggests choosing $A_n = A_{n-1} + 2A_{n-2}$. Once this choice is made, however, we cannot choose $A_{n+1} = A_n + 2A_{n-1}$, since then $2A_{n-1} < A_n$, which would imply that the next step is $A_n = 2A_{n-1} + A_{n-2}$. Hence instead, we choose $A_{n+1} = 2A_n + A_{n-1}$. This gives the recurrence for the A_i.)

To prove Theorem 5.1, we need some simple lemmas on the properties of the sequence A_n:

Lemma 4.1. *Let $\alpha = (5+\sqrt{17})/2$, $\beta = (5-\sqrt{17})/2$, $c_1 = (\sqrt{17}+1)/2$, $c_2 = (\sqrt{17}-1)/2$. Then*

$$A_{2n} = (\alpha^{n+1} - \beta^{n+1})/\sqrt{17}, \quad n \geq -1;$$
$$A_{2n-1} = (c_1\alpha^n + c_2\beta^n)/\sqrt{17}, \quad n \geq 0.$$

Proof. Easily derived by the method of constant coefficients or proved by induction.

Lemma 4.2. *For $n \geq 2$ we have $A_n = 5A_{n-2} - 2A_{n-4}$. For $n \geq 1$ we have $A_{2n} = 4A_{2n-2} + A_{2n-3}$. For $n \geq 0$ we have $A_{2n+1} = 3A_{2n-1} + 4A_{2n-2}$.*

Proof. Left to the reader.

Lemma 4.3. *For $n \geq -1$ we have $A_n \equiv 1 \pmod 2$.*

Proof. By induction on n.

Lemma 4.4. *For $n \geq 0$ we have*

$$1 \leq \frac{A_{2n}}{A_{2n-1}} < \frac{3+\sqrt{17}}{4}.$$

For $n \geq 1$ we have

$$\frac{1+\sqrt{17}}{2} < \frac{A_{2n+1}}{A_{2n}} < 3.$$

Proof. By Lemma 4.1 we have

$$\frac{A_{2n}}{A_{2n-1}} < \frac{\alpha}{c_1} = \frac{3+\sqrt{17}}{4},$$

and clearly $A_{2n} \geq A_{2n-1}$. This proves the first inequality. By Lemma 4.1 we have

$$\frac{A_{2n+1}}{A_{2n}} > c_1 = \frac{1+\sqrt{17}}{2};$$

also,

$$\frac{A_{2n+1}}{A_{2n}} = 2 + \frac{A_{2n-1}}{A_{2n}} < 3.$$

Lemma 4.5. *Let $\gamma = (1+\sqrt{17})/8 \doteq .6404$ and $\delta = (3+\sqrt{17})/2 \doteq 3.562$. Let $f(x) = (3x+1)/(4x+2)$, $g(x) = (4x+2)/(x+1)$, and $h(x) = (3x+1)/(x+1)$. Then if $x \geq \gamma$, we have $f(x), g(x) \geq \gamma$ and $h(x) \geq (3+\sqrt{17})/4$. If $0 \leq x \leq \delta$, we have $f(x), g(x) \leq \delta$ and $h(x) \leq (1+\sqrt{17})/2$.*

Proof. Left to the reader.

We now come to the main lemma of this paper:

Lemma 4.6. *Let u, v be integers, u odd, $u > v > 0$. Let γ and δ be as in Lemma 4.5.*

(a) If the ordinary algorithm to compute $\left(\frac{v}{u}\right)$ performs n division steps, and n is even, then

$$xu + v \geq xA_n + A_{n-1}$$

for $x \geq \gamma$.

(b) If the ordinary algorithm to compute $\left(\frac{v}{u}\right)$ performs n division steps, and n is odd, then

$$xu + v \geq xA_n + A_{n-1}$$

for $0 \leq x \leq \delta$.

Proof. We prove both statements simultaneously by induction on n.

For $n = 1$, it is easily verified that $u \geq 3$, $v \geq 1$, and so we have $xu + v \geq xA_1 + A_0$ for all $x \geq 0$.

Similarly, for $n = 2$, we have $u \geq 5$, $v \geq 3$, and so $xu + v \geq xA_2 + A_1$ for all $x \geq 0$.

Now assume that the lemma is true for all $m < n$; we prove it for $m = n$.

We may assume without loss of generality that v is odd. Put $u_0 = u$ and $u_1 = v$. The first step of the algorithm sets

$$u_0 = a_0 u_1 + 2^{e_2} u_2.$$

Either a_0 is odd or a_0 is even.

If a_0 is odd, then parity considerations show $e_2 \geq 1$. Hence

$$u_0 = a_0 u_1 + 2^{e_2} u_2 \geq u_1 + 2u_2.$$

Also note that $2u_2 < u_1$, so that if

$$u_1 = a_1 u_2 + 2^{e_3} u_3,$$

then $a_1 \geq 2$. Hence $u_1 \geq 2u_2 + u_3$.

If a_0 is even, then $u_0 \geq 2u_1 + u_2$. Then, depending on whether u_1/u_2 is less than or greater than 2, we have $u_1 \geq u_2 + 2u_3$ or $u_1 \geq 2u_2 + u_3$.

To summarize, we have three cases to consider: (i) $u_0 \geq u_1 + 2u_2$, $u_1 \geq 2u_2 + u_3$; (ii) $u_0 \geq 2u_1 + u_2$, $u_1 \geq u_2 + 2u_3$; and (iii) $u_0 \geq 2u_1 + u_2$, $u_1 \geq 2u_2 + u_3$.

Case (i): Here we have

$$u_0 \geq 4u_2 + u_3;$$
$$u_1 \geq 2u_2 + u_3.$$

Hence it follows that

$$xu_0 + u_1 \geq (4x+2)u_2 + (x+1)u_3. \tag{5}$$

Now the algorithm on (u_2, u_3) performs $n - 2$ division steps. By Lemma 4.5, we see that if $x \geq \gamma$ then $g(x) = (4x+2)/(x+1) \geq \gamma$; and if $0 \leq x \leq \delta$, then $0 \leq g(x) \leq \delta$. Hence the induction hypothesis applies to $g(x)$ and we find

$$\frac{4x+2}{x+1}u_2 + u_3 \geq \frac{4x+2}{x+1}A_{n-2} + A_{n-3}.$$

Hence

$$(4x+2)u_2 + (x+1)u_3 \geq (4x+2)A_{n-2} + (x+1)A_{n-3}. \tag{6}$$

Now suppose n is even. Then by Lemma 4.2 we have

$$(4x+2)A_{n-2} + (x+1)A_{n-3} = xA_n + A_{n-1}, \tag{7}$$

and so by combining (5)–(7) we find

$$xu_0 + u_1 \geq xA_n + A_{n-1},$$

as desired.

Now suppose n is odd. Then using Lemma 4.2, we find

$$(4x+2)A_{n-2} + (x+1)A_{n-3} = xA_n + A_{n-1} + (x+1)A_{n-2} - (3x+1)A_{n-3}. \tag{8}$$

On the other hand, Lemmas 4.4 and 4.5 tell us that

$$A_{n-2} > \frac{1+\sqrt{17}}{2}A_{n-3} \geq \frac{3x+1}{x+1}A_{n-3}$$

for $0 \leq x \leq \delta$. Hence

$$(x+1)A_{n-2} - (3x+1)A_{n-3} \geq 0, \tag{9}$$

and combining (5), (6), (8), and (9) yields the result.

Case (ii): Here we have

$$u_0 \geq 3u_2 + 4u_3;$$
$$u_1 \geq u_2 + 2u_3.$$

Hence it follows that

$$xu_0 + u_1 \geq (3x+1)u_2 + (4x+2)u_3. \tag{10}$$

Now the algorithm on (u_2, u_3) performs $n-2$ division steps. By Lemma 4.5, we see that if $x \geq \gamma$ then $f(x) = (3x+1)/(4x+2) \geq \gamma$; and if $0 \leq x \leq \delta$, then $0 \leq f(x) \leq \delta$. Hence the induction hypothesis applies to $f(x)$ and we find

$$\frac{3x+1}{4x+2}u_2 + u_3 \geq \frac{3x+1}{4x+2}A_{n-2} + A_{n-3}.$$

Hence

$$(3x+1)u_2 + (4x+2)u_3 \geq (3x+1)A_{n-2} + (4x+2)A_{n-3}. \tag{11}$$

Now suppose n is odd. Then by Lemma 4.2 we have

$$(3x+1)A_{n-2} + (4x+2)A_{n-3} = xA_n + A_{n-1}, \tag{12}$$

and so by combining (10)–(12) we find

$$xu_0 + u_1 \geq xA_n + A_{n-1},$$

as desired.

Now suppose n is even. Then using Lemma 4.2, we find

$$(3x+1)A_{n-2} + (4x+2)A_{n-3} = xA_n + A_{n-1} - (x+1)A_{n-2} + (3x+1)A_{n-3}. \qquad (13)$$

On the other hand, Lemmas 4.4 and 4.5 tell us that

$$A_{n-2} < \frac{3+\sqrt{17}}{4}A_{n-3} < \frac{3x+1}{x+1}A_{n-3}$$

for $x \geq \gamma$. Hence

$$(3x+1)A_{n-3} - (x+1)A_{n-2} > 0, \qquad (14)$$

and combining (10), (11), (13), and (14) yields the result.

Case (iii): Here we have

$$u_0 \geq 5u_2 + 2u_3;$$
$$u_1 \geq 2u_2 + u_3.$$

In this case, both u_0 and u_1 are at least as large as the u_0 and u_1 covered in case (i), so the inequality also holds here.

This completes the proof of Lemma 4.6.

5. Proof of Theorem 5.1.

We can now prove Theorem 5.1, which was stated in the last section.

Proof of Theorem 5.1.

First, we show that on input $u_0 = A_n$, $u_1 = A_{n-1}$, the algorithm actually performs exactly n steps. Clearly this is true for $n = 1, 2$. Assume true for $m < n$; we wish to prove it for $m = n$.

If n is odd, then by Lemma 4.4 we know $\lfloor A_n/A_{n-1} \rfloor = 2$, so $u_0 - 2u_1 = A_n - 2A_{n-1} = A_{n-2}$ by the definition of A_n. And A_{n-2} is odd by Lemma 4.3, so $e_2 = 0$. Thus the algorithm continues with $(u_1, u_2) = (A_{n-1}, A_{n-2})$, which by induction requires $n-1$ division steps. Hence the result follows.

On the other hand, if n is even, then by Lemma 4.4 we know $\lfloor A_n/A_{n-1} \rfloor = 1$, so $u_0 - u_1 = A_n - A_{n-1} = 2A_{n-2}$ by the definition of A_n. Again A_{n-2} is odd by Lemma 4.3, so $e_2 = 1$. Thus the algorithm continues with $(u_1, u_2) = (A_{n-1}, A_{n-2})$, which by induction requires $n-1$ division steps. Hence the result follows.

Now let (u', v') be any input on which the ordinary algorithm performs n steps. By setting $x = 1$ in Lemma 4.6, we see that $u' + v' \geq A_n + A_{n-1}$. If we could now show that $u = A_n$, $v = A_{n-1}$ is actually the *only* pair requiring n division steps with $u + v = A_n + A_{n-1}$, our result would follow.

To do this, suppose (u', v') is another pair with

$$u' + v' = A_n + A_{n-1}. \tag{15}$$

Then by Lemma 4.6 we have $2u' + v' \geq 2A_n + A_{n-1}$. Subtracting (15), we see $u' \geq A_n$. On the other hand, by Lemma 4.6 we also have $\frac{2}{3}u' + v' \geq \frac{2}{3}A_n + A_{n-1}$. Subtracting (15), we see $-u'/3 \geq -A_n/3$, or $u' \leq A_n$. Hence $u' = A_n$, $v' = A_{n-1}$, and the result follows.

Corollary. *Let the inputs to the ordinary algorithm be $u > v > 0$. Then the ordinary algorithm performs no more than $1.32 \log(u + v) - .72$ division steps.*

Thus we see that, in terms of the number of division steps for the worst case, Lebesgue's algorithm is superior to both Eisenstein's algorithm and the ordinary algorithm, as might be expected.

6. Worst-case inputs of the ordinary algorithm under lexicographic orderings.

In this section, we again consider the ordinary Jacobi symbol algorithm, and seek the "smallest" inputs requiring n division steps, where the implied orderings are the lexicographic ordering and the reverse lexicographic ordering.

We say that a pair (u, v) is *lexicographically less* than (u', v') if $u < u'$, or if $u = u'$ and $v < v'$. We write $(u, v) < (u', v')$.

Similarly, (u, v) is *reverse lexicographically less* than (u', v') if (v, u) is lexicographically less than (v', u'). We write $(u, v) <_R (u', v')$.

Theorem 6.1. *Let $u > v > 0$ and u odd.*

(a) If the ordinary algorithm to compute $\left(\frac{v}{u}\right)$ requires $2n$ division steps and (u, v) is lexicographically least among all pairs with this property, then $u = A_{2n}, v = A_{2n-1}$.

(b) If the ordinary algorithm to compute $\left(\frac{v}{u}\right)$ requires $2n + 1$ division steps and (u, v) is reverse lexicographically least among all pairs with this property, then $u = A_{2n+1}, v = A_{2n}$.

Proof.

(a) Let (u, v) take $2n$ division steps, and suppose $(u, v) < (A_{2n}, A_{2n-1})$. By Lemma 4.6 (a) we have $xu + v \geq xA_{2n} + A_{2n-1}$ for $x \geq \gamma$. Hence

$$u - A_{2n} \geq \frac{A_{2n-1} - v}{x},$$

and by choosing x sufficiently large we see $u \geq A_{2n}$. Hence $u = A_{2n}$ and $v < A_{2n-1}$. But by setting $x = 1$ in Lemma 4.6 (a), we have $v \geq A_{2n-1}$, a contradiction. Thus (A_{2n}, A_{2n-1}) must be lexicographically least.

(b) Let (u, v) take $2n + 1$ division steps, and suppose $(u, v) <_R (A_{2n+1}, A_{2n})$. Then by setting $x = 0$ in Lemma 4.6 (b) we see $v \geq A_{2n}$. Hence we must have $u < A_{2n+1}$. But by setting $x = 1$ in Lemma 4.6 (b) we see $u \geq A_{2n+1}$, a contradiction.

The reader will note the theorem above says nothing about the two missing symmetric cases: $2n$ division steps under the reverse lexicographic ordering, and $2n+1$ division steps under the lexicographic ordering. For these cases, we have Conjecture 6.2 below.

Define $R_0 = 0$, $R_1 = 1$, $R_2 = 7$, $R_3 = 31$, and $R_n = 5R_{n-1} - 10R_{n-3} + 4R_{n-4}$. Define $S_0 = 1$, $S_1 = 5$, $S_2 = 31$, $S_3 = 141$, and $S_n = 5S_{n-1} - 10S_{n-3} + 4S_{n-4}$. Finally, define $T_0 = 1$, $T_1 = 3$, $T_2 = 13$, $T_3 = 57$, and $T_n = 5T_{n-1} - 10T_{n-3} + 4T_{n-4}$.

Here is a brief table of the sequences R_n, S_n, and T_n:

$n =$	0	1	2	3	4	5	6	7	8	9	...
$R_n =$	0	1	7	31	145	659	3013	13739	62685	285931	...
$S_n =$	1	5	31	141	659	3005	13739	62669	285931	1304285	...
$T_n =$	1	3	13	57	259	1177	5367	24473	111631	509193	...

Numerical evidence supports the following conjecture:

Conjecture 6.2. *Let $u > v > 0$ and u odd.*

(a) If the ordinary algorithm to compute $\left(\frac{v}{u}\right)$ takes $2n+1$ steps and (u,v) is lexicographically least among all pairs having this property, then $(u,v) = (T_{n+1}, S_n)$.

(b) If the ordinary algorithm to compute $\left(\frac{v}{u}\right)$ takes $2n$ steps and (u,v) is reverse lexicographically least among all pairs having this property, then $(u,v) = (R_{n+1}, T_n)$.

If true, this conjecture would be remarkable, because these worst cases do not correspond to an ultimately periodic sequence of quotients, as is the case with every other known Euclidean-type algorithm. (See the description of the matrices $M(n)$ below.)

While the author is unable to prove Conjecture 6.2, it is possible to prove the following:

Theorem 6.3.

The ordinary algorithm on input (T_{n+1}, S_n) performs $2n+1$ division steps. Further, $(T_{n+1}, S_n) < (A_{2n+1}, A_{2n})$ for $n \geq 2$.

The ordinary algorithm on input (R_{n+1}, T_n) performs $2n$ division steps. Further, $(R_{n+1}, T_n) <_R (A_{2n}, A_{2n-1})$ for $n \geq 3$.

Proof.

We prove only the first result, as the proof of the second is almost identical.

First we define some matrices that describe the transformations taking place in the ordinary algorithm:

Let $M_1 = \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix}$, $M_2 = \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}$, $M_3 = \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix}$. If $e_i \in \{1,2,3\}$, we define

$$M_{e_1 e_2 \cdots e_k} = M_{e_1} M_{e_2} \cdots M_{e_k}.$$

Lemma 6.4. *Let $e_i \in \{1,2\}$ for $1 \leq i \leq k-1$ and set*

$$\begin{bmatrix} a_k & b_k \\ c_k & d_k \end{bmatrix} = M_{e_1 e_2 \cdots e_{k-1}} M_3.$$

If e_i and e_{i+1} are never both equal to 1 for $1 \le i \le k-2$, then the Jacobi symbol algorithm performs k division steps on input (a_k, c_k).

Proof. Left to the reader.

Now define

$$M(n) = M_{12}^n M_{21}^n = \begin{bmatrix} w_n & x_n \\ y_n & z_n \end{bmatrix}.$$

We wish to find a recursion for the sequences $\{w_n\}$, $\{x_n\}$, $\{y_n\}$, $\{z_n\}$. We find

$$M(n+1) = M_{12}M(n)M_{21} = \begin{bmatrix} 12w_n + 4x_n + 3y_n + z_n & 16w_n + 8x_n + 4y_n + 2z_n \\ 6w_n + 2x_n + 3y_n + z_n & 8w_n + 4x_n + 4y_n + 2z_n \end{bmatrix},$$

$$\begin{aligned} M(n+2) &= M_{12}M(n+1)M_{21} \\ &= \begin{bmatrix} 234w_n + 90x_n + 65y_n + 25z_n & 360w_n + 144x_n + 100y_n + 40z_n \\ 130w_n + 50x_n + 39y_n + 15z_n & 200w_n + 80x_n + 60y_n + 24z_n \end{bmatrix}, \end{aligned}$$

and

$$\begin{aligned} M(n+3) &= M_{12}M(n+2)M_{21} \\ &= \begin{bmatrix} 4838w_n + 1886x_n + 1357y_n + 529z_n & 7544w_n + 2952x_n + 2116y_n + 828z_n \\ 2714w_n + 1058x_n + 767y_n + 299z_n & 4232w_n + 1656x_n + 1196y_n + 468z_n \end{bmatrix} \end{aligned}$$

It is easy to find that

$$M(n+3) = 23M(n+2) - 46M(n+1) + 8M(n) \tag{16}$$

by solving a system of linear equations; hence each of the sequences $\{w_n\}$, $\{x_n\}$, $\{y_n\}$, $\{z_n\}$ satisfy this linear recurrence.

Now put

$$M(n)M_3 = \begin{bmatrix} w'_n & x'_n \\ y'_n & z'_n \end{bmatrix}$$

and

$$M(n)M_{213} = \begin{bmatrix} w''_n & x''_n \\ y''_n & z''_n \end{bmatrix}.$$

Each of the eight sequences defined as the entries of the above matrices must satisfy the same recurrence (16), as each entry is a linear combination of terms which satisfy (16).

Hence if we now define $S_{2k} = y'_k$ and $S_{2k+1} = y''_k$ for $k \ge 0$, then we deduce $S_0 = 1$, $S_1 = 5$, $S_2 = 31$, $S_3 = 141$, $S_4 = 659$, $S_5 = 3005$, and $S_n = 23S_{n-2} - 46S_{n-4} + 8S_{n-6}$ for $n \ge 6$.

Similarly, if we define $T_{2k+1} = w'_k$ and $T_{2k+2} = w''_k$ for $k \ge 0$, then we deduce $T_1 = 3$, $T_2 = 13$, $T_3 = 57$, $T_4 = 259$, $T_5 = 1177$, $T_6 = 5367$, and $T_n = 23T_{n-2} - 46T_{n-4} + 8T_{n-6}$ for $n \ge 7$.

We can now prove Theorem 6.3:

Proof.

It follows from Lemma 6.4 that on input (T_{n+1}, S_n), the ordinary algorithm performs $2n+1$ division steps.

It remains to show that the sequences $\{S_n\}$ and $\{T_n\}$ actually satisfy the recursion stated in Theorem 6.3, and that $T_{n+1} < A_{2n+1}$ for $n \geq 2$.

For this, it is necessary to find a closed form for T_n and S_n. We observe that the associated characteristic polynomial for the recurrence is $x^6 - 23x^4 + 46x^2 - 8$. It factors as follows:

$$x^6 - 23x^4 + 46x^2 - 8 = (x^2-2)(x^2-5x+2)(x^2+5x+2).$$

This, together with the help of a computer algebra system, allows us to find a closed form for the recurrences. Let α and β be as in Lemma 4.1. Then

$$S_n = \left(\frac{23+7\sqrt{17}}{34}\right)\alpha^n + \left(\frac{23-7\sqrt{17}}{34}\right)\beta^n + \left(\frac{8\sqrt{2}-3}{17}\right)(-\sqrt{2})^n + \left(\frac{-8\sqrt{2}-3}{17}\right)(\sqrt{2})^n,$$

$$T_n = \left(\frac{6+\sqrt{17}}{17}\right)\alpha^n + \left(\frac{6-\sqrt{17}}{17}\right)\beta^n + \left(\frac{5-2\sqrt{2}}{34}\right)(-\sqrt{2})^n + \left(\frac{5+2\sqrt{2}}{34}\right)(\sqrt{2})^n.$$

These formulas are easily verified by induction. From these formulas, it is easy to see that $S_n = 5S_{n-1} - 10S_{n-3} + 4S_{n-4}$, as asserted, and that T_n also satisfies the same recurrence.

We now show that $T_{n+1} < A_{2n+1}$ for all n sufficiently large. For this it suffices to observe that the closed forms for T_n and A_n imply that $T_{n+1} \sim ((6+\sqrt{17})/17)\alpha^{n+1}$ and $A_{2n+1} \sim ((\sqrt{17}+1)/2\sqrt{17})\alpha^{n+1}$. Since $((6+\sqrt{17})/17) < ((\sqrt{17}+1)/2\sqrt{17})$, the result follows for all n sufficiently large. We leave the proof that $T_{n+1} < A_{2n+1}$ for $n \geq 2$ to the reader.

7. Some remarks.

Gauss (1876) showed how to compute the Jacobi symbol $\left(\frac{v}{u}\right)$ using the partial quotients in the continued fraction for u/v. Thus by using the fast methods of Schönhage (1971) for computation of continued fractions, one can compute $\left(\frac{v}{u}\right)$ in $O(n(\log n)^2 \log\log n)$ bit operations, where $u, v < 2^n$. (This was pointed out to the author by H. W. Lenstra, Jr. and E. Bach.) However, this method is unlikely to be competitive in practice, except for extremely large inputs.

For a discussion of other methods to compute Jacobi symbols, see Bachmann (1968, pp. 290-302).

V. C. Harris (1970) found the worst case of a Euclidean algorithm similar to the ones described here. G. J. Rieger (1976, 1980a, 1980b) has analyzed this algorithm.

A "shift-remainder" algorithm for computing the GCD, with some similarity to algorithms mentioned here, was analyzed by G. Norton (1987).

It is also possible to adapt the so-called "binary GCD algorithm" of Stein (1967) to compute the Jacobi symbol. Also see Knuth (1981).

E. Bach points out (personal communication) that the three Jacobi symbol algorithms discussed in this paper could also be used to compute $\gcd(u, v)$, where v is odd. In fact, in the notation of section 1, this gcd is just u_n.

Bach has also suggested that one could investigate the *average number* of division steps in computing $(\frac{v}{u})$, as Heilbronn (1969) and Porter (1975) have done for the ordinary Euclidean algorithm, and Rieger (1978) for the least-remainder algorithm. This analysis is probably feasible to carry out for Eisenstein's algorithm, and it seems likely that the average number of division steps is $O((\log u)^2)$. However, determining the average-case behavior for Lebesgue's algorithm or the ordinary algorithm seems quite hard.

8. Acknowledgments.

A preliminary version of this paper was presented at the AMS Summer Meeting in Boulder, Colorado, on August 9, 1989.

V. C. Harris read an earlier version of this paper and made many helpful suggestions.

Thanks to Kevin McCurley for pointing out the reference to Smith (1965) and Williams (1980).

Thanks also to one of the referees, whose detailed remarks substantially improved the presentation of this paper.

References

Angluin, D. (1982). Lecture notes on the complexity of some problems in number theory, Yale University, Department of Computer Science, Technical Report 243.

Bachmann, P. (1968). "Niedere Zahlentheorie," Chelsea, New York.

Collins, G. E., Loos, R. G. K. (1982). The Jacobi symbol algorithm, *ACM SIGSAM Bulletin* **16** (1), 12-16.

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.

Eisenstein, G. (1844). Einfacher Algorithmus zur Bestimmung des Werthes von $(\frac{a}{b})$, *J. für die Reine und Angew. Math.* **27**, 317-318.

Gauss, C. F. (1876). Theorematis fundamentalis in doctrina de residuis quadraticis demonstrationes et ampliationes novae, in *Werke*, V. II, pp. 49-64.

Harris, V. C. (1970). An algorithm for finding the greatest common divisor, *Fib. Quart.* **8**, 102-103.

Heilbronn, H. (1969). On the average length of a class of finite continued fractions, in *Number Theory & Analysis*, pp. 87-96.

Jacobi, C. G. J. (1846). Über die Kreistheilung und ihre Anwendung auf die Zahlentheorie, *J. für die Reine und Angew. Math.* **30**, 166-182. (= *Werke*, V. 6, pp. 254-274.)

Knuth, D. E. (1981). "The art of computer programming," V. II (Seminumerical Algorithms), 2nd edition, Addison-Wesley, Reading, Mass.

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.

Lebesgue, V.-A. (1847). Sur le symbole $\left(\frac{a}{b}\right)$ et quelques-unes de ses applications, *J. Math. Pures Appl.* **12**, 497-517.

LeVeque, W. J. (1977). "Fundamentals of number theory," Addison-Wesley, Reading, Mass.

Norton, G. (1987). A shift-remainder GCD algorithm, in L. Huguet and A. Poli, eds., *AAECC-5* (Lecture Notes in Computer Science #356), pp. 350-356.

Porter, J. W. (1975). On a theorem of Heilbronn, *Mathematika* **22**, 20-28.

Rieger, G. J. (1976). On the Harris modification of the Euclidean algorithm, *Fib. Quart.* **14**, 196,200.

Rieger, G. J. (1978). Über die mittlere Schrittanzahl bei Divisionsalgorithmen, *Math. Nachr.* **82**, 157-180.

Rieger, G. J. (1980a). Über die Schrittanzahl beim Algorithmus von Harris und dem nach nächsten Ganzen, *Arch. Math.* **34**, 421-427.

Rieger, G. J. (1980b). Continued fractions and related algorithms, in *London Mathematical Society Lecture Note Series #56*, Journées Arithmétiques, J. V. Armitage, editor, pp. 372-378.

Riesel, H. (1985). "Prime numbers and computer methods for factorization," Birkhäuser, Boston.

Rosen, K. H. (1984). "Elementary number theory and its applications," Addison-Wesley, Reading, Mass.

Schönhage, A. (1971). Schnelle Berechnung von Kettenbruchentwicklungen, *Acta Inform.* **1**, 139-144.

Smith, H. J. S. (1965). "Report on the theory of numbers," Chelsea, New York.

Solovay, R. and Strassen, V. (1977). A fast Monte-Carlo test for primality, *SIAM J. Comput.* **6**, 84-85; erratum, **7** (1978), 118.

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

Williams, H. C. (1980). A modification of the RSA public-key encryption procedure, *IEEE Trans. Info. Theory* **IT-26**, 726-729.