1-1-1989

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

Jeffrey Shallit
Dartmouth College

ON THE WORST CASE OF THREE ALGORITHMS FOR COMPUTING THE JACOBI SYMBOL

Jeffrey Shallit

Technical Report PCS-TR89-140

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

Abstract.

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 $A_n = 5A_{n-2} - 2A_{n-4}$.

We also show the existence of inputs to the ordinary algorithm (u_n, v_n) and (u'_n, v'_n) requiring n division steps, such that (i) $u_n < A_n$ and $v_n > A_{n-1}$; (ii) $v'_n < A_{n-1}$ and $u'_n > A_n$. This explains why a characterization of the worst-case inputs (u, v) for the ordinary algorithm that is based on u or v alone is not satisfactory.

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

1. Introduction.

In this paper, I 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:

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

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

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

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 [D]. Letting $D_0 = 0$, $D_1 = 1$, and $D_n = 2D_{n-1} + D_{n-2}$ for $n \geq 2$, we have [K, exercise 4.5.3.30]

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*

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

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 [J]:

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

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

$$\left(\frac{-1}{u}\right) = (-1)^{(u-1)/2}; \tag{3}$$

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

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.

Eisenstein's algorithm

Eisenstein [E] proposed the following algorithm for computing $\left(\frac{v}{u}\right)$: 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 $\epsilon_i = \pm 1$ and each a_i, except 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

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

Lebesgue's algorithm

Lebesgue [Leb] 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

$$\begin{aligned} u_0 &= a_0u_1 + \epsilon_2 2^{e_2}u_2 \\ u_1 &= a_1u_2 + \epsilon_3 2^{e_3}u_3 \\ &\vdots \\ u_{n-1} &= a_{n-1}u_n. \end{aligned}$$

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

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

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. [LeV, p. 112], [Ro, pp. 319-320]), the earliest *explicit* mention I have been able to find is Williams [W]. Collins and Loos [CL] 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. [A], [Ri]). Gaston Gonnet informs me (personal communication) that the ordinary algorithm is the one currently used for computing $\left(\frac{v}{u}\right)$ by the computer algebra system Maple.

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

$$\begin{aligned} u_0 &= a_0u_1 + 2^{e_2}u_2 \\ u_1 &= a_1u_2 + 2^{e_3}u_3 \\ &\vdots \\ u_{n-1} &= a_{n-1}u_n. \end{aligned}$$

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 $e_{i+2} = \nu_2(u_i - a_iu_{i+1})$. Then

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

where

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

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 4.1 and Lemma 4.10.

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 on input (u, v). Then $u \ge 2n + 1$ and $v \ge 2n - 1$.*

Proof.

By induction on n. Clearly the lemma is true for $n = 1$. Now assume it is true for $m < n$, and we wish to prove it for $m = n$. Write $u_0 = a_0 u_1 + \epsilon_2 u_2$, where a_0 is even. Then $u_0 \ge 2u_1 - u_2$, where $u_2 < u_1$. Now Eisenstein's algorithm on input (u_1, u_2) takes $n - 1$ steps, so by induction we have $u_1 \ge 2n - 1$ and $u_2 \ge 2n - 3$. Then $u_0 \ge 2n + 1$, and the proof is complete. ■

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

Proof.

By the Lemma, $u \ge 2n + 1$ and $v \ge 2n - 1$. 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 $n \ge 2$. It is easy to prove by induction that

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

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

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

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 $u_1 \leq u_0 < 2u_1$, 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 $e_2 \geq 1$. Hence $u_0 \geq u_1 + 2u_2$. But $2^{e_2} u_2 \leq u_1/2$ because the least remainder is chosen at each step; hence $2u_2 \leq u_1/2$ and $u_0 \geq 6u_2$. But $u_1/u_2 \geq 2$, so Lemma 3.1 applies and we have $u_1 \geq L_n$ and $u_2 \geq L_{n-1}$. Then $u_0 \geq 6L_{n-1} \geq 3L_{n-1} + 2L_{n-2} = 2L_n - L_{n-1}$, 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 $u_3 \leq u_2/2$, so Lemma 3.1 applies to (u_2, u_3) and we find $u_2 \geq L_{n-1}$, $u_3 \geq L_{n-2}$.

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

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

Theorem 3.3.

Let $u > v > 0$, v odd, be such that Lebesgue's algorithm performs n division steps on input (u, v), and u is as small as possible. Then $u = 2L_n - L_{n-1}$ and $v = L_n$.

Proof.

By Lemma 3.2 we have $u \geq 2L_n - L_{n-1}$ and $v \geq L_n$. 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. Some Lemmas.

Definition. *Let* $A_{-2} = 0$, $A_{-1} = 1$, $A_0 = 1$, $A_1 = 3$, *and* $A_n = 5A_{n-2} - 2A_{n-4}$ *for* $n \geq 2$.

Here is a brief table of the A_i:

$$\begin{array}{rl} n &= \text{-2 -1 0 1 2 3 4 5 6 7 8 9 10 11 12 } \ldots \\ A_n &= \ 0\ \ 1\ \ 1\ \ 3\ \ 5\ 13\ 23\ 59\ 105\ 269\ 479\ 1227\ 2185\ 5597\ 9967\ \ldots \end{array}$$

In section 5, we will prove the following

Theorem 4.1. *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* $v = A_{n-1}$.

To prove Theorem 4.1, we need some simple lemmas on the properties of the sequence A_n:

Lemma 4.2. *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 4.3. *For* $n \geq 1$ *we have* $A_{2n} = A_{2n-1} + 2A_{2n-2} = 4A_{2n-2} + A_{2n-3}$. *For* $n \geq 0$ *we have* $A_{2n+1} = 2A_{2n} + A_{2n-1} = 3A_{2n-1} + 4A_{2n-2}$.

Proof. Easily proved using Lemma 4.2. ■

Lemma 4.4. *For* $n \geq -1$ *we have* $A_n \equiv 1 \pmod 2$.

Proof. By induction on n. ■

Lemma 4.5. *For* $n \geq 0$ *we have*

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

Proof. By Lemma 4.2 we have

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

and $A_{2n} \geq A_{2n-1}$ by Lemma 4.3. ■

Lemma 4.6. *For $n \geq 1$ we have*

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

Proof. By Lemma 4.2 we have

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

and by Lemma 4.3,

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

Let $\gamma = (1+\sqrt{17})/8 \doteq .6404$ and $\delta = (3+\sqrt{17})/2 \doteq 3.562$. We need three technical lemmas about properties of the closed interval $[\gamma, \delta]$:

Lemma 4.7. *Let $y \in [\gamma, \delta]$, and $f : x \to \frac{4x+2}{x+1}$. Then $f(y) \in [\gamma, \delta]$.*

Proof. In fact,

$$f(y) \in [f(\gamma), f(\delta)] = [(7+\sqrt{17})/4, \delta] \subset [\gamma, \delta]. \quad \blacksquare$$

Lemma 4.8. *Let $y \in [\gamma, \delta]$, and $g : x \to \frac{3x+1}{4x+2}$. Then $g(y) \in [\gamma, \delta]$.*

Proof. In fact,

$$g(y) \in [g(\gamma), g(\delta)] = [\gamma, (7-\sqrt{17})/4] \subset [\gamma, \delta]. \quad \blacksquare$$

Lemma 4.9. *Let $h : x \to \frac{3x+1}{x+1}$. Then*

$$\min_{y \in [\gamma, \delta]} h(y) = h(\gamma) = \frac{3+\sqrt{17}}{4},$$

and

$$\max_{y \in [\gamma, \delta]} h(y) = h(\delta) = \frac{1+\sqrt{17}}{2}.$$

Proof. Left to the reader. $\blacksquare$

We now come to the main lemma of this paper:

Lemma 4.10. *Let u, v be integers, u odd, $u > v > 0$, such that the ordinary algorithm to compute $\left(\frac{v}{u}\right)$ performs n division steps. Then*

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

for $x \in [\gamma, \delta]$.

Proof. By induction on n. The lemma is easily verified for $n = 1, 2$. Now we will assume it is true for all $m < n$ and 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: (a) $u_0 \geq u_1 + 2u_2$, $u_1 \geq 2u_2 + u_3$; (b) $u_0 \geq 2u_1 + u_2$, $u_1 \geq u_2 + 2u_3$; and (c) $u_0 \geq 2u_1 + u_2$, $u_1 \geq 2u_2 + u_3$.

Case (a): 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. Since $x \in [\gamma, \delta]$, we have $\frac{4x+2}{x+1} \in [\gamma, \delta]$ by Lemma 4.7, so the induction hypothesis applies and

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

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 4.3 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 4.3, 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 4.6 and 4.9 tell us that

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

for $x \in [\gamma, \delta]$. Hence it follows that

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

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

Case (b): 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. Since $x \in [\gamma, \delta]$, we have $\frac{3x+1}{4x+2} \in [\gamma, \delta]$ by Lemma 4.8, so the induction hypothesis applies and

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

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 4.3 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 4.3, we find

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

On the other hand, Lemmas 4.5 and 4.9 tell us that

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

for $x \in [\gamma, \delta]$. Hence it follows that

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

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

Case (c): 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 (a), so the inequality also holds here.

This completes the proof of Lemma 4.10. ■

5. Proof of Theorem 4.1.

We can now prove Theorem 4.1.

Proof. Let $u > v > 0$ be such that the ordinary algorithm for computing $\left(\frac{v}{u}\right)$ performs n division steps. Then from Lemma 4.10 we have $u + v \geq A_n + A_{n-1}$.

We now 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 4.6 we know $\lfloor A_n/A_{n-1} \rfloor = 2$, so $u_0 - 2u_1 = A_n - 2A_{n-1} = A_{n-2}$ by Lemma 4.3. And A_{n-2} is odd by Lemma 4.4, 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 4.5 we know $\lfloor A_n/A_{n-1} \rfloor = 1$, so $u_0 - u_1 = A_n - A_{n-1} = 2A_{n-2}$ by Lemma 4.3. Again A_{n-2} is odd by Lemma 4.4, 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 we must show that $u = A_n$, $v = A_{n-1}$ is actually the *only* pair requiring n division steps with $u + v = A_n + A_{n-1}$. To do this, suppose (u', v') is another pair with

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

Then by Lemma 4.10 we have $2u' + v' \geq 2A_n + A_{n-1}$. Subtracting (15), we see $u' \geq A_n$. On the other hand, by Lemma 4.10 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.

6. "Paradoxical" inputs.

Suppose we examine inputs (u, v) to one of the algorithms mentioned in this article that require n division steps. Then the ordinary Euclidean algorithm, the least-remainder Euclidean algorithm, Eisenstein's Jacobi symbol algorithm, and Lebesgue's algorithm are all sufficiently simple that specifying u *alone* to be as small as possible forces *both* u and v to be the terms of a linear recurrence.

However, no such simple characterization is possible for the ordinary Jacobi symbol algorithm, as there exist inputs with "paradoxical" behavior. For example, the input (u, v) that requires 6 division steps and minimizes $u + v$ is $(105, 59)$, as predicted by Theorem 4.1. On the other hand, a smaller v is possible: namely, the input $(145, 57)$. The input (u, v) that requires 7 division steps and minimizes $u + v$ is $(269, 105)$, but a smaller u is possible, as shown by the input $(259, 141)$. One way to interpret this is to observe that Lemma 4.10 is actually *false* for very large and very small values of x.

In this section, we show the existence of an infinite sequence of such "paradoxical" inputs. Since the development is not crucial to the main result, some details are omitted.

Theorem 6.1. *Define $S_0 = 1$, $S_1 = 5$, $S_2 = 31$, $S_3 = 141$, and $S_n = 5S_{n-1} - 10S_{n-3} + 4S_{n-4}$. Define $T_0 = 1$, $T_1 = 3$, $T_2 = 13$, $T_3 = 57$, and $T_n = 5T_{n-1} - 10T_{n-3} + 4T_{n-4}$. Then the ordinary Jacobi symbol algorithm on input (T_{n+1}, S_n) performs $2n + 1$ division steps. Further, $T_{n+1} < A_{2n+1}$ for $n \geq 2$.*

Theorem 6.2. *Define $R_0 = 0$, $R_1 = 1$, $R_2 = 7$, $R_3 = 31$, and $R_n = 5R_{n-1} - 10R_{n-3} + 4R_{n-4}$. Then the ordinary algorithm on input (R_{n+1}, T_n) performs $2n$ division steps.*

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	3	13	57	259	1177	5367	24473	111631	509193	...
T_n =	1	5	31	141	659	3005	13739	62669	285931	1304285	...

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 6.3. *Let $e_i \in \{1,2\}$ for $1 \le i \le k-1$ 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 n 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, though tedious, to verify that

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

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 $k \geq 0$, 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 $n \geq 7$.

We can now prove Theorem 6.1:

Proof.

It follows from Lemma 6.3 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 6.1, 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 4.2. 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.

For a discussion of other methods to compute Jacobi symbols, see [B, pp. 290-302].

V. C. Harris found the worst case of a Euclidean algorithm similar to the ones described here [Ha]. G. J. Rieger has analyzed this algorithm [R1, R2, R3].

Eric 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 $\left(\frac{v}{u}\right)$, as Heilbronn [He] and Porter [P] have done for the ordinary Euclidean algorithm, and Rieger [R4] 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.

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 Williams [W].

References

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

[B] P. Bachmann, "Niedere Zahlentheorie," Chelsea, New York, 1968.

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

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

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

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

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

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

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

[L] G. Lamé, 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** (1844), 867-870.

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

[LeV] W. J. LeVeque, "Fundamentals of number theory," Addison-Wesley, Reading, Mass., 1977.

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

[R1] G. J. Rieger, On the Harris modification of the Euclidean algorithm, *Fib. Quart.* **14** (1976), 196,200.

[R2] G. J. Rieger, Über die Schrittanzahl beim Algorithmus von Harris und dem nach nächsten Ganzen, *Arch. Math.* **34** (1980), 421-427.

[R3] G. J. Rieger, Continued fractions and related algorithms, in *London Mathematical Society Lecture Note Series #56*, Journées Arithmétiques 1980, J. V. Armitage, editor, pp. 372-378.

[R4] G. J. Rieger, Über die mittlere Schrittanzahl bei Divisionsalgorithmen, *Math. Nachr.* **82** (1978), 157-180.

[Ri] H. Riesel, "Prime numbers and computer methods for factorization," Birkhäuser, Boston, 1985.

[Ro] K. H. Rosen, "Elementary number theory and its applications," Addison-Wesley, Reading, Mass., 1984.

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

Last revision: April 7, 1989.