An $O(M(n) \log n)$ algorithm for the Jacobi symbol

Paul Zimmermann
(joint work with Richard P. Brent)

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE | INRIA
centre de recherche NANCY - GRAND-EST

-IX

20 July 2010

Motivation

```
From: Galbraith Steven
Date: Fri, Apr 17, 2009 at 4:26 PM
To: Paul Zimmermann, Pierrick Gaudry

Hi Paul and Pierrick,

Sorry to bother you.

The usual algorithm to compute the Legendre (or Jacobi) symbol
is closely related to Euclid's algorithm.  There are variants of
Euclid for n-bit integers which run in O( M(n) \log(n)) bit operations.
Hence it is natural to expect a O( M(n) \log(n)) algorithm for
Legendre symbols.

I don't see this statement anywhere in the literature. Is this:

 (a) in the literature somewhere
 (b) so obvious no-one ever wrote it down
 (c) false due to some subtle reason.

Thanks for your help.

Regards
Steven
```

(b) so obvious no-one ever wrote it down

This is what we first thought.

However we soon realized it was not so easy...

Magma V2.16-10 on 2.83Ghz Core 2:

```
> a:=3^209590; b:=5^143067;
> time c := Gcd(a,b);
Time: 0.080
> time d := JacobiSymbol(a,b);
Time: 2.390
```

Sage 4.4.4 on 2.83Ghz Core 2:

```
sage: a=3^209590; b=5^143067
sage: a.ndigits(), b.ndigits()
(100000, 100000)
sage: %timeit a.gcd(b)
5 loops, best of 3: 49.9 ms per loop
sage: %timeit a.jacobi(b)
5 loops, best of 3: 2.04 s per loop
```

GMP 5.0.1 on 2.83Ghz Core 2:

```
patate% ./speed -s 5190 mpn_gcd mpz_jacobi
              mpn_gcd    mpz_jacobi
5190     #0.040993000   1.577760000
```

GP/PARI 2.4.3:

```
? a=3^209590; b=5^143067;
? gcd(a,b);
time = 41 ms.
? kronecker(a,b)
  ***   at top-level: kronecker(a,b)
  ***                 ^--------------
  *** kronecker: the PARI stack overflows !
  current stack size: 8000000 (7.629 Mbytes)
break> allocatemem()
  ***  new stack size = 4096000000 (3906.250 Mbytes).
? kronecker(a,b)
time = 4,893 ms.
```

(a) in the literature somewhere

Two MSB (Most Signific Bits first) algorithms:

- “Algorithmic Number Theory” from Bach and Shallit, solution of Exercise 5.52 (Gauss, Bachmann) [sketch];
- a different algorithm mentioned by Schönhage in his “TP book”, but without details.

As far as we know, no subquadratic implementation exists, except that of Schönhage in the TP language.

```
From Arnold Schoenhage <schoe@cs.uni-bonn.de> 20091126 via email:

Excerpt from my old file IGCDOC (1987):

(7.1) x_{j-1}= q_j*x_j + x_{j+1},  where x_0=x, x_1=y, x_k=x_{k+1}=1.

(7.2) r_j == x_j mod 4    with    0 \le r_j < 4

Theorem 7.1.  With regard to the quantities in (7.1) and (7.2),
------------  the Jacobi symbol satisfies the following recurrence
relations, valid for odd values of  x_{j-1} > 1. If x_j is odd, then

(7.3)   (x_j|x_{j-1}) = (x_{j+1}|x_j)*s(r_{j-1},r_j),

       where    s(1,1) = s(1,3) = s(3,1)= 1,   s(3,3)= -1.

If x_j is even, then x_{j+1} must be odd, and in this case one has

                     / (x_{j+2}|x_{j+1}),                    if r_j= 0,
(7.4)   (x_j|x_j-1)= <
                     \ (x_{j+2}|x_{j+1})*t(r_{j+1}, q_j),  if r_j= 2,

       where   t(r,q) = / -1  for  q == 2  or  q == r mod 4,
                        \ +1  otherwise.
```

Now suppose $m = qn + r$, with $0 \le r < n$. Then $m/n = q + r/n$; $2m/n = 2q + 2r/n$; ..., $n'm/n = n'q + n'r/n$. Adding these up, and applying the result above, we get

$$\psi(m, n) = \binom{n'+1}{2} q + n'r' - \psi(n, r). \tag{A.6}$$

Now assume $u, v > 0$, with v odd. Expand u/v as a continued fraction, obtaining $u_0 = a_0 u_1 + u_2$; $u_1 = a_1 u_2 + u_3$; ..., $u_{n-1} = a_{n-1} u_n + u_{n+1}$, where $u_0 = u$, $u_1 = v$, and $u_{n+1} = 0$. If $u_n \ne 1$, then $\gcd(u, v) > 1$, and so $\left(\frac{u}{v}\right) = 0$. Otherwise, use Eq. (A.6) and compute

$$\psi = \sum_{0 \le i \le n-1} a_i \binom{u'_{i+1} + 1}{2} + \sum_{0 \le i \le n-1} (u'_{i+1} u'_{i+2} \bmod 2).$$

Then $\left(\frac{u}{v}\right) = (-1)^{\psi}$ if u is odd or $v \equiv \pm 1 \pmod 8$, and $\left(\frac{u}{v}\right) = (-1)^{\psi+1}$ otherwise.

Using Schönhage's rapid method for computing continued fractions, it follows that $\left(\frac{u}{v}\right)$ can be computed in the stated time bound.

This complexity bound is part of the "folklore" and apparently has never appeared in print. The basic idea can be found in Gauss [1876]. Our presentation is based on that in Bachmann [1902]. H. W. Lenstra, Jr. also informed us of this idea; he attributes it to A. Schönhage.

Bach & Shallit, ANT

(c) false due to some subtle reason

We'll try to show this is not so!

Plan of the talk

- The Binary (Generalized) Division
- A Cubic LSB Algorithm
- A Quadratic LSB Algorithm
- A Subquadratic LSB Algorithm
- Implementation and Timings

The Jacobi symbol

$\left(\frac{b}{a}\right)$ or $(b|a)$ is defined for integers a, b, with a *odd positive*.

$$(b|a) = (b \bmod a|a)$$

$$(b|a) = (-1)^{(a-1)(b-1)/4}(a|b) \qquad \text{for } b \text{ odd positive}$$

$$(bc|a) = (b|a)(c|a)$$

$$(2|a) = (-1)^{(a^2-1)/8}$$

$$(-1|a) = (-1)^{(a-1)/2}$$

$$(b|a) = 0 \qquad \text{if } (a, b) \neq 1$$

In this talk we propose a LSB (Least Signific Bit) algorithm, that can be easily implemented in $O(M(n) \log n)$ by modifying a LSB gcd.

We assume a is odd positive, b is even positive.

- if b is negative, use $(b|a) = (-1)^{(a-1)/2}(-b|a)$.
- if b is odd, use $(b|a) = (b+a|a)$.

The Binary Division

The Binary Division

A binary recursive gcd algorithm, Stehlé and Z., ANTS VI, 2004.

Classical (MSB) division forces 0's in the MSBs:

decimal	*binary*
935	1110100111
714	1011001010
221	0011011101
51	0000110011
17	0000010001
0	0000000000

GCD = $(10001)_2 = 17$

Binary Division

$$a = 935 = (1110100111)_2$$

$$b = 714 = (1011001010)_2$$

- divide b by the largest possible power of two:

$$b/2 = 357 = (101100101)_2$$

- now choose between $a + b/2$ and $a - b/2$ the one with most trailing zeroes:

$$a + b/2 = 1292 = (10100001100)_2$$

$$a - b/2 = 578 = (1001000010)_2$$

Binary Division: Another Example

$$a = 935 = (1110100111)_2$$

$$b = 716 = (1011001100)_2$$

$$a + b/4 = 1114 = (10001011010)_2$$

$$a - b/4 = 756 = (1011110100)_2$$

$$a + 3b/4 = 1472 = (10111000000)_2$$

$$a - 3b/4 = 398 = (110001110)_2$$

Here we choose $a + 3b/4$ as next term.

Theory

$a, b \in \mathbb{Z}$ with $j := \nu_2(b) - \nu_2(a) > 0$

There is a unique $|q| < 2^j$ such that $\nu_2(b) < \nu_2(r)$ and:

$$r = a + q2^{-j}b$$

q is the *binary quotient* of a by b
r is the *binary remainder* of a by b

Rationale: if a, b have both n bits, $b' = 2^{-j}b$ has $n - j$ bits, and qb' has about n bits, thus r has about the same bit-size as a, but at least $j + 1$ more zeros in the LSB.

Computation

$$j = \nu_2(b) - \nu_2(a) > 0$$

$$q \equiv -a/(b/2^j) \bmod 2^{j+1} \quad \text{(centered)}$$

Binary remainder sequence $\nu_2(a) < \nu_2(b) < \nu_2(r) < \cdots$

Binary Division (and GCD)

Binary (LSB) division forces 0's in the LSBs:

935	1110100111
714	1011001010
1292	10100001100
1360	10101010000
1632	11001100000
2176	100010000000
0	000000000000

GCD $= (10001)_2 = 17$

Advages of the Binary Division:

- $\oplus$ simpler to compute (division mod 2^{j+1} instead of MSB division);
- $\oplus$ no “repair step” in the subquadratic GCD (see however Möller, Math. Comp., 2008);
- $\oplus$ an average reduction of two LSB bits per iteration;
- $\ominus$ an average increase of 0.05 MSB bit per iteration (analyzed precisely by Daireaux, Maume-Deschamps and Vallée, DMTCS, 2005).

Using the Binary Division for the Jacobi Symbol

It seems easy to adapt, using $b' = b/2^j$ odd:

$$(b|a) = (-1)^{j(a^2-1)/8}(b'|a)$$

$$(b'|a) = (-1)^{(a-1)(b'-1)/4}(a|b')$$

$$(a|b') = (a + qb'|b') = (r|b')$$

$$(r|b') = (-1)^{j(b'^2-1)/8}(r/2^j|b')$$

However r can be negative!
Example: 935, 738, 1304, -240, 1184, -832, 768, -1024, 0.

Incompatible with definition of Jacobi symbol, which requires a odd positive.

A Cubic LSB Algorithm

Binary Division with Positive Quotient

Instead of taking $q = a/(b/2^j)$ in $[-2^j, 2^j]$, take it in $[0, 2^{j+1}]$.

Since $q > 0$, if $a, b > 0$, all terms are non-negative:

$$r = a + q2^{-j}b$$

Stopping GCD criterion: $a/2^{\nu_2(a)} = b/2^{\nu_2(b)}$.

Example: 935, $714 = 357 \cdot 2$, $1292 = 323 \cdot 2^2$, $1360 = 85 \cdot 2^4$, $1632 = 51 \cdot 2^5$, $2176 = 17 \cdot 2^7$, $4352 = 17 \cdot 2^8$.

A Cubic LSB Algorithm

Algorithm CubicBinaryJacobi.

Input: $a, b \in \mathbb{N}$ with $\nu(a) = 0 < \nu(b)$

Output: Jacobi symbol $(b|a)$

$s \leftarrow 0$

$j \leftarrow \nu(b)$

while $2^j a \neq b$ **do**

$\quad b' \leftarrow b/2^j$

$\quad (q, r) \leftarrow \mathrm{BinaryDividePos}(a, b)$

$\quad s \leftarrow \left(s + \frac{j(a^2-1)}{8} + \frac{(a-1)(b'-1)}{4} + \frac{j(b'^2-1)}{8}\right) \bmod 2$

$\quad (a, b) \leftarrow (b', r/2^j)$

$\quad j \leftarrow \nu(b)$

if $a = 1$ **then** return $(-1)^s$ **else** return 0

(lines in red are added to the GCD LSB-algorithm)

Cost of the Cubic Algorithm

Let n be the bit-size of the inputs a, b.

Each iteration costs $O(n)$ (unless j is large, but this is unlikely, and in this case (a, b) decrease even more).

The number of iterations is $O(n^2)$ (see below).

Thus the total cost is $O(n^3)$ (probably less, see below).

A Quadratic LSB Algorithm

Lemma

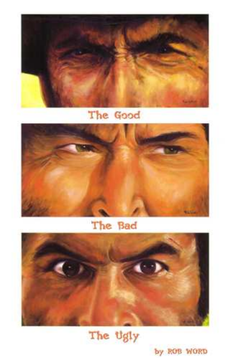
The quantity $a + 2b$ is non-increasing in CubicBinaryJacobi.

Proof.

At each iteration, $a + 2b$ becomes:

$$\frac{2a}{2^j} + (1 + \frac{2q}{2^j})\frac{b}{2^j}.$$

If $j \geq 2$, $a + 2b$ is multiplied by a factor at most $9/16$: *good* iteration.
If $j = 1$ and $q = 1$, $a + 2b$ decreases, but with a factor that can be arbitrarily close to 1: *bad* iteration.
If $j = 1$ and $q = 3$, $a + 2b$ remains unchanged: *ugly* iteration.



$\square$

Examples

Good iteration: $a = 9, b = 4$ gives $j = 2, q = 7$, $b' = 1, r/2^j = 4$, $a + 2b = 17$ becomes 9.

Bad iteration: $a = 9, b = 6$ gives $b' = 3, r/2^j = 6$, $a + 2b = 21$ becomes 15.

Ugly iteration: $a = 9, b = 10$ gives $b' = 5, r/2^j = 12$, $a + 2b = 29$ remains 29.

Lemma

If $\mu = \nu(a - b/2)$, there are exactly $\lfloor \mu/2 \rfloor$ ugly iterations starting from (a, b), followed by a good iteration if μ is even, otherwise by a bad iteration.

Example 1: $a - b/2 = 64 = 2^6$

$$(85, 42) \underbrace{\rightarrow}_{\textit{ugly}} (21, 74) \underbrace{\rightarrow}_{\textit{ugly}} (37, 66) \underbrace{\rightarrow}_{\textit{ugly}} (33, 68) \underbrace{\rightarrow}_{\textit{good}} (34, 38) \cdots$$

Example 2: $a - b/2 = 128 = 2^7$

$$(149, 42) \underbrace{\rightarrow}_{\textit{ugly}} (21, 106) \underbrace{\rightarrow}_{\textit{ugly}} (53, 90) \underbrace{\rightarrow}_{\textit{ugly}} (45, 94) \underbrace{\rightarrow}_{\textit{bad}} (47, 46) \cdots$$

A Quadratic LSB Algorithm

Main idea: from the 2-valuation of $a - b/2$, compute the number $m > 0$ of consecutive ugly iterations, and apply them all at once: *harmless* iteration.

The Jacobi symbol can also be easily updated for m consecutive ugly iterations (see the proceedings).

Now we have only good (G), bad (B), or harmless (H) iterations, where HH is forbidden.

Algorithm QuadraticBinaryJacobi

$s \leftarrow 0, \quad j \leftarrow \nu(b), \quad b' \leftarrow b/2^j$
while $a \neq b'$ **do**
  $s \leftarrow (s + j(a^2 - 1)/8) \bmod 2$
  $(q, r) \leftarrow \mathrm{BinaryDividePos}(a, b)$
  if $(j, q) = (1, 3)$ **then** ▷ harmless iteration
    $d \leftarrow a - b'$
    $m \leftarrow \nu(d) \operatorname{div} 2$
    $c \leftarrow (d - (-1)^m d/4^m)/5$
    $s \leftarrow (s + m(a - 1)/2) \bmod 2$
    $(a, b) \leftarrow (a - 4c, b + 2c)$
  else ▷ good or bad iteration
    $s \leftarrow (s + (a - 1)(b' - 1)/4) \bmod 2$
    $(a, b) \leftarrow (b', r/2^j)$
  $s \leftarrow (s + j(a^2 - 1)/8) \bmod 2, \quad j \leftarrow \nu(b), \quad b' \leftarrow b/2^j$
if $a = 1$ **then** return $(-1)^s$ **else** return 0

Analysis of the Quadratic Algorithm

Lemma

Algorithm QuadraticBinaryJacobi needs $O(n)$ iterations.

Proof.

Consider a block of three iterations (G, B, or H):

- G multiplies $a+2b$ by at most $9/16 < 5/8$;
- HH is forbidden, thus we have either $HB = U^m B$ or BB;
- UB multiplies $a+2b$ by at most $5/8$, and U^{m-1} leaves it unchanged;
- BB multiplies $a+2b$ by at most $1/2 < 5/8$.

Thus each three iterations multiply $a+2b$ by at most $5/8$, thus the number of iterations if $cn + O(1)$, where $c = 3/\log_2(8/5) \approx 4.4243$. □

A Subquadratic LSB Algorithm

Cf Algorithm 3.1 page 90 in the proceedings.

Implementation and Timings

Experimental Results for Large Numbers

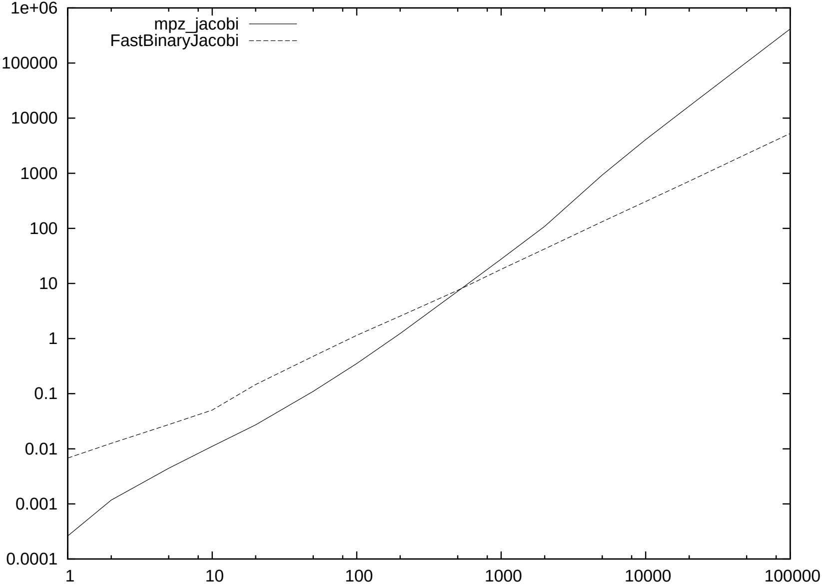
Timings on a 2.83Ghz Core 2 with GMP 4.3.1, with inputs of one million 64-bit words.

GMP's fast gcd takes 45.8s.

An implementation of the (fast) binary gcd takes 48.3s.

Our implementation FastBinaryJacobi takes 83.1s.

Our implementation is faster than GMP's $O(n^2)$ code up from 535 words (about 10,000 decimal digits).



mpz_jacobi
FastBinaryJacobi
1e+06
100000
10000
1000
100
10
1
0.1
0.01
0.001
0.0001
1
10
100
1000
10000
100000

Concluding Remarks

- first complete (description + code) subquadratic Jacobi algorithm
- first LSB algorithm for the problem
- does not need to compute the (MSB) quotient sequence
- we can use the "cubic" algorithm with a centered quotient. Moreover we can choose $q \pm 2^{j+1} \in [-2^{j+1}, 2^{j+1}]$ such that $bq/2^j$ has sign opposite to a. We then gain on average 2.19 bits per iteration, against 1.95 for the centered quotient, 1.35 for the positive quotient, and 1.42 for Stein's "binary gcd".

GMP code available from:

`http://www.loria.fr/~zimmerma/papers/#jacobi`

Thanks to:

- Steven Galbraith for asking the original question;
- Damien Stehlé for suggesting using the LSB algorithm;
- Arnold Schönhage for his comments and pointers to earlier work;
- the anonymous  reviewer who took the time to implement and try our algorithm:

> *The new method is very easy to implement. In fact, I implemented it in Magma myself, and my non-optimised version was already faster than whatever Magma uses as standard algorithm, for reasonable inputs.*