Jebelean-Weber's Algorithm without Spurious Factors

Sidi Mohamed Sedjelmaci

LIPN, CNRS UPRES-A 7030
Université Paris-Nord,
Av. J.B.-Clément, 93430 Villetaneuse, France.
sms@lipn.univ-paris13.fr

ABSTRACT

Tudor Jebelean and Ken Weber introduced an algorithm for finding (a, b)-pairs satisfying $au + bv \equiv 0 \pmod{k}$, with $0 < |a|,\ |b| < \sqrt{k}$. It is based on Sorenson's "k-ary reduction" This algorithm does not preserve the GCD and its related GCD algorithm has an $O(n^2)$ time bit complexity in the worst case. We present a modified version which avoids this problem. We show that a slightly modified GCD algorithm has an $O(n^2/\log n)$ running time in the worst case, where n is the number of bits of the larger input.

Keywords: Integer greatest common divisor (GCD); Parallel GCD algorithm; Extended GCD algorithm; Complexity analysis; Number theory.

1 INTRODUCTION

Given two integers a and b, the greatest common divisor of a and b, denoted $GCD(a, b)$, is the largest integer which divides both a and b. Applications for GCD algorithms include computer arithmetic, integer factoring, cryptology and symbolic computation [7, 15, 5] . In [10], Sorenson proposed the "right-shift k-ary algorithm". It is based on the following reduction. Given two positive integers $u > v$ relatively prime to k (i.e., (u, k) and (v, k) are coprime), pairs of integers (a, b) can be found that satisfy

$$\begin{aligned} au + bv &\equiv 0 \pmod{k}, \\ \text{with} \quad & 0 < |a|,\ |b| < \sqrt{k}. \end{aligned} \tag{1}$$

If we perform the transformation (also called "k-ary reduction"):

$$(u, v) \longmapsto (u', v') = (|au+bv|/k, \min(u, v)),$$

which replaces u with $u' = |au + bv|/k$, the size of u is reduced by roughly $\frac{1}{2}\log_2(k)$ bits since

$$|au + bv|/k \leq 2\max(|a|, |b|)\frac{u}{k} < \frac{2u}{\sqrt{k}}. \tag{2}$$

Sorensen suggests table lookup to find sufficiently small a and b satisfying (1). By contrast, Jebelean [4] and Weber [16] both propose a simple algorithm, which finds such small a and b that satisfy (1) with time complexity $O(n^2)$. This latter algorithm we call the "Jebelean-Weber algorithm", or *JWA* for short. A GCD algorithm based on this reduction works very well in practice and is included in Gnu MP multiprecision library [2]. However this GCD algorithm does not preserve the GCD, since for some $\alpha|a$

$$GCD(v, |au + bv|/k) = \alpha\ GCD(u, v),$$

whence some spurious factors must be eliminated (see example in Section 4). This drawback affects the efficiency of the GCD algorithm, at least for small integers ($\leq$ 1000 bits). In this present work we show how a slightly modified version of JWA easily avoids this problem. Not only is this modified version desirable for GCD computations but it is also needed in many other applications, such as Jacobi symbol computation or modular inverse, to mention only a few [7].

The paper is organized as follows. Notations and definitions are given in Section 2. In Section 3, we recall the Jebelean-Weber algorithm and propose a modified version. Section 4 deals with the correctness. In Section 5 we describe our algorithm, study its time complexity, and report on preliminary experiments. We conclude with some remarks in Section 6.

2 NOTATION

Throughout this paper, we restrict ourselves to the set of non-negative integers. Let u and v be two such (non-negative) integers; u and v are respectively n-bits and p-bits numbers with $u \geq v \geq 1$. Let k be an integer parameter s.t. $k \geq 4$.

Given a non-negative integer $x \in N$, $\ell_2(x)$ represents the number of significant bits of an non-negative integer x, not counting leading zeros: $\ell_2(x) = \lfloor \log_2(x) \rfloor + 1$, if $x \geq 1$ and $\ell_2(0) = 0$. So $n = \ell_2(u)$, $p = \ell_2(v)$ and p satisfies $2^{p-1} \leq v < 2^p$. We let $\rho = \rho(u, v) = \ell_2(u) - \ell_2(v) + 1$. Thus, we obtain $2^{\rho-2} < u/v < 2^\rho$.

Let a, b be positive integers, the integer $x = a \bmod b$ is the unique non-negative integer x such that $0 \leq x \leq b-1$ and $x - a$ is divisible by b. Note that this notation still holds when $a < 0$. If b is relatively prime to k, then $r = a/b \bmod k$ is the unique non-negative integer r such that $0 \leq r \leq k-1$ and $r\, b \equiv a \pmod k$.

As noticed by many authors the main difficulty in GCD algorithms happens when the input data u and v are roughly of the same size [10, 4, 16]. So we shall assume that when Sorenson's reduction is applied : $u/v < \sqrt{k}$. Otherwise, we usually apply a more efficient reduction: an Euclidean step or the *bmod*, defined as: $bmod(u, v) = |u - (u/v \mod 2^\rho)v|/2^\rho$.

The extended version of Euclid GCD algorithm is noted *EEA* [5]. It is tightly linked with continued fractions [3, 5] and is important for its multiple applications in cryptology and computer algebra.

3 THE ALGORITHMS

3.1 `JWA`: The Jebelean-Weber Algorithm

First we recall the JWA as stated in [16].

Input: $x,\ y > 0,\ k \geq 4$, and $\gcd(k, x) = \gcd(k, y) = 1$.
Output: (n, d) such that $0 < n,\ |d| < \sqrt{k}$, and $ny \equiv dx \pmod k$.

$r := x/y \bmod k$;
$f_1 = (n_1, d_1) := (k, 0)$;
$f_2 = (n_2, d_2) := (r, 1)$;
while $n_2 \geq \sqrt{k}$ **do**
  $f_1 := f_1 - \lfloor n_1/n_2 \rfloor f_2$;
  swap (f_1, f_2) ;
endwhile
return f_2

Fig. 1. *The Jebelean-Weber Algorithm*

When (n, d) is the output result of JWA, the pair $(a, b) = (d, -n)$ (or $(-d, n)$) satisfies the property $au + bv = 0 \bmod k$. The algorithm `JWA` is nothing but the extended version of Euclid `EEA` applied to the pair $(k, u/v \bmod k)$, where only one column is added instead of two for `EEA` (see [5]), and they only differ on their exit test.

3.2 The Modified Jebelean-Weber Algorithm: `M-JWA`

We give below a modified version that avoids spurious factors introduced in `JWA`.

Input: $x,\ y > 0,\ k \geq 4$ such that $\gcd(k, x) = \gcd(k, y) = 1$
Output: A 2×2 integer matrix $M = M(x, y, k) = \begin{pmatrix} n_1 & d_1 \\ n_2 & d_2 \end{pmatrix}$ such that $0 < n_2,\ |d_2| < \sqrt{k}$, $n_2 y \equiv d_2 x \pmod k$ and $n_1 y \equiv d_1 x \pmod k$.

$r := x/y \bmod k$;
$f_1 = (n_1, d_1) := (k, 0)$;
$f_2 = (n_2, d_2) := (r, 1)$;

while $n_2 \geq \sqrt{k}$ **do**
$f_1 := f_1 - \lfloor n_1/n_2 \rfloor f_2$;
swap (f_1, f_2) ;
endwhile
return $M = \begin{pmatrix} n_1 & d_1 \\ n_2 & d_2 \end{pmatrix}$

Fig. 2. *The Modified Jebelean-Weber Algorithm:* `M-JWA`

The new transformation associated with the output matrix of `M-JWA` is defined by $(u, v) \leftarrow (R_1, R_2)$ with:

$$R_1 = |n_1 v - d_1 u|/k \text{ and} \quad (3)$$
$$R_2 = |n_2 v - d_2 u|/k. \quad (4)$$

We will prove in the next section that the transformation $(u, v) \leftarrow (R_1, R_2)$ preserves the GCD, i.e.: $GCD(R_1, R_2) = GCD(u, v)$ and avoids the spurious factors of algorithm `JWA`.

4 CORRECTNESS

Before proving that indeed, `M-JWA` preserves the GCD, we first recall below some well known properties [3, 5] of `EEA` that are also valid for `JWA` as well as for `M-JWA`. Let $(n_s, d_s)_{s \geq 1}$ be the pair of sequences corresponding to the successive results of f_2 in `JWA` or `M-JWA` and $(n_0, d_0) = (k, 0)$; then $\forall s \geq 1$ we have

- $n_s > 0$ and $d_s d_{s+1} < 0$
- $n_s/d_s \equiv x/y \pmod{k}$
- $n_s d_{s+1} - n_{s+1} d_s = (-1)^s k$
- $(n_s)_s$ is decreasing and $(|d_s|)_s$ is increasing.

Lemma 4.1 *The output of JWA satisfies* $n_2 y - d_2 x \equiv 0 \pmod{k}$ *and*
$$0 < n_2, |d_2| < \sqrt{k} \leq n_1.$$

Proof : ([16]) In the last iteration i of `JWA` or `M-JWA` n_i must meet the condition $n_i < \sqrt{k} < n_{i-1}$. Hence, since $n_{i-1}|d_i| + n_i|d_{i-1}| = k$, $n_{i-1}|d_i| \leq k$ and $|d_i| < k/n_{i-1} \leq \sqrt{k}$. Moreover, we have at the end of the while loop $n_2 < \sqrt{k} \leq n_1$. *QED*

We prove in the following that the output integer matrix of `M-JWA` enjoys more interesting properties.

Lemma 4.2 *Let* $u \geq v \geq 1$ *and* $k \geq 4$ *be three positive integers such that* $\gcd(u, k) = \gcd(v, k) = 1$ *and* $u/v < \sqrt{k}$. *Let* $\begin{pmatrix} c & d \\ a & b \end{pmatrix}$ *be the output integer matrix of* `M-JWA` *with* (u, v) *as inputs, then* $G = (|du - cv|)/k$ *is a positive integer such that* $0 \leq G \leq v$.

Proof : First, G is an integer since $c/d \equiv a/b \equiv u/v \bmod k = r$. Moreover, if $r < \sqrt{k}$ then $c = k$, $d = 0$ and $G = v$. Otherwise $k > r \geq \sqrt{k}$ and since $|d| \leq |b|$, we proceed in two cases:
Case 1: If $|b| = |d|$, then this case only happens only when $b = -1$, $d = 1$, $c = r$ and $\lfloor k/r \rfloor = 1$. Since $k > c > \sqrt{k} > u/v$, we obtain
$G = |u - cv|/k = |u/v - c|(v/k) = (c - u/v)(v/k) < (c/k)v < v$.
Case 2: If $|b| > |d|$. We have $G \leq (|d|u + cv)/k = (\frac{|d|u}{kv} + c/k)v$. Let us prove that $\frac{|d|u}{kv} + c/k < 1$, i.e.: $u/v < \frac{k-c}{|d|}$. From $|b| > |d|$ we obtain $\frac{|b|-1}{|d|} \geq 1$. From Lemma 4.1 we have $c \geq \sqrt{k}$ and using the relation $k = c|b| + a|d|$, we obtain the result

$$\frac{k - c}{|d|} = \frac{c|b| + a|d| - c}{|d|}$$

$$= c(\frac{|b| - 1}{|d|}) + a \geq \sqrt{k} > u/v.$$

QED

Lemma 4.3 *Let* $u \geq v \geq 1$ *and* $k \geq 1$ *be three integers such that* $GCD(u, k) = GCD(v, k) = 1$. *Let* $M = \begin{pmatrix} c & d \\ a & b \end{pmatrix}$ *be an integer matrix with* $|det\ M| = |cb - ad| = k$. *If there exist two integers* R_1, R_2 *satisfying*
$$k \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} = M \begin{pmatrix} u \\ v \end{pmatrix}, \quad \textit{then}$$
$GCD(R_1, R_2) = GCD(u, v)$.

Proof : Let $\alpha = GCD(u, v)$ and $\beta = GCD(R_1, R_2)$ then $k\ R_1 = (cu + dv)$ so $\alpha | kR_1$ but $GCD(u, k) = 1$ then

$GCD(\alpha, k) = 1$ and $\alpha | R_1$. Similarly $k\ R_2 = (au + bv)$ and $\alpha | R_2$. Hence $\alpha | \beta$. Moreover, since $|cb - ad| = k \neq 0$, M^{-1} exists and $\begin{pmatrix} u \\ v \end{pmatrix} = k \times M^{-1} \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} = k \times \epsilon / k \times \begin{pmatrix} bR_1 - dR_2 \\ -aR_1 + cR_2 \end{pmatrix}$, with $\epsilon = \pm 1$, hence $\beta | \alpha$ and the result $\alpha = \beta$. *QED*

Example: If $k = 2^\rho$, then the transformation $(u, v) := (v, \texttt{bmod}(u, v))$ preserves the GCD since the associated matrix is $M = \begin{pmatrix} 0 & k \\ 1 & -r \end{pmatrix}$, with $r = u/v \bmod k$.

Remark: It is worth to note that this lemma generalizes a well known result in the case $k = \pm 1$, i.e.: if $\begin{pmatrix} R_1 \\ R_2 \end{pmatrix} = M \begin{pmatrix} u \\ v \end{pmatrix}$, and $\det M = \pm 1$ then $GCD(R_1, R_2) = GCD(u, v)$. This situation occurs in `EEA`.

A similar method can be applied to eliminate spurious factors for the left-shift k-ary GCD of Sorenson [10]. Following the same approach, a pair of integers (c, d) can be found such that $\det \begin{pmatrix} c & d \\ a & b \end{pmatrix} = \pm 1$.

Proposition 4.1
Let $M(u, v, k) = \begin{pmatrix} n_1 & d_1 \\ n_2 & d_2 \end{pmatrix}$ be the output integer matrix of `M-JWA`*, given input u, v and k such that $u/v < \sqrt{k}$. If $\begin{pmatrix} R_1 \\ R_2 \end{pmatrix} = \begin{pmatrix} |n_1 v - d_1 u|/k \\ |n_2 v - d_2 u|/k \end{pmatrix}$, then R_1 and R_2 are two integers satisfying $0 \leq R_1 \leq v$, $0 \leq R_2 \leq 2u/\sqrt{k}$ and $GCD(R_1, R_2) = GCD(u, v)$.*

Proof : We have $k \begin{pmatrix} R_1 \\ R_2 \end{pmatrix} = N \begin{pmatrix} u \\ v \end{pmatrix}$, where N is one of the four following matrices $N_1 = \begin{pmatrix} -d_1 & n_1 \\ -d_2 & n_2 \end{pmatrix}$, $N_2 = \begin{pmatrix} d_1 & -n_1 \\ d_2 & -n_2 \end{pmatrix}$, $N_3 = \begin{pmatrix} -d_1 & n_1 \\ d_2 & -n_2 \end{pmatrix}$ or $N_4 = \begin{pmatrix} d_1 & -n_1 \\ -d_2 & n_2 \end{pmatrix}$. Then the result derives straightforwards from Lemma 4.2, Lemma 4.3 and relation (2). *QED*

Example: Let $(u, v) = (28865, 19203)$ and $k = 2^6 = 64$. Note that $GCD(u, v) = 1$. We obtain in turn $u/v \bmod k = 1/3 \bmod 64 = 43$, and $M = \begin{pmatrix} 21 & -1 \\ 1 & 3 \end{pmatrix}$. Hence $R_1 = |u + 21v|/64 = 6752$, and $R_2 = |3u - v|/64 = 1053$. The `JWA` algorithm uses the transformation $(u, v) \leftarrow (v, R_2)$. However $GCD(v, R_2) = GCD(19203, 1053) = 3 \neq GCD(u, v)$, while, with `M-JWA` algorithm, we obtain $GCD(R_1, R_2) = GCD(6752, 1053) = GCD(u, v) = 1$. The spurious factor 3 has been eliminated. Table 1 gives some examples of spurious factors with Fibonacci pair inputs.

5 THE `M-JWA` `GCD` ALGORITHM

An easy GCD algorithm can be designed by simply alternating `M-JWA` reductions and `Euclidean` reductions to achieve an $O(n^2/\log n)$ running time in the worst case. This algorithm is similar to `ModGenBin` algorithm of Sorenson [12], however the difference is that there are no spurious factors at all. We first recall some results on basic arithmetic (see [10, 11] for more details).

Lemma 5.1 *Let x, y and $k = 2^m$, $m \geq 2$ be three positive integers with $x > y$. If $y \leq k$, then xy, $\lfloor x/y \rfloor$ and $x \bmod y$ can be computed in $O(\log x)$ bit operations. If $y > k$, then xy can be computed in $O(\log x + (\log x \log y)/\log k)$ bits operations. Moreover, $\lfloor x/y \rfloor$ and $x \bmod y$ can be computed in $O(\log x + (\log \lceil x/y \rceil \log y)/\log k)$ bits operations. These results require precomputed tables of size $O(k^2 \log k)$ bits. It requires at most $O(k^2 \log^2 k)$ bit operations to compute these tables.*

Proof : See [12]. *QED*

If the length n of the input u is such that $\log n > W/2$, where $W = 32$ or 64, then we allow the parameter k to grow with the length n: $\log k = \Theta(\log n)$. For example, as in [12], we can choose the parameter k such that $k = n^{0.4}$.

Input: $u \geq v \geq 3$, two odd integers.
Output: $\gcd(u, v)$.

$k = 2^{32}$ or $k = 2^{64}$;
$n := \lfloor \log_2(u) \rfloor + 1$;
if $(n^{0.4} > k)$ **then**
  $m := \lfloor 0.4 \log_2 n \rfloor + 1$;
  $m := m + (m \bmod 2)$; $k := 2^m$;
  Precompute tables
  (see Lemma 5.1) ;
endif
while $uv \neq 0$ **do**
  if $u < v$ **then** $(u, v) := (v, u)$;
  if $u/v < \sqrt{k}$ **then**
    $\begin{pmatrix} c & d \\ a & b \end{pmatrix} :=$ `M-JWA`(u, v, k);
    $u := |du - cv|/k$;
    $v := |bu - av|/k$;
  else $(u, v) := (v, u \bmod v)$;
  `makeodd`(u); `makeodd`(v);
endwhile
return $u + v$;

Fig. 3. *The Modified Jebelean-Weber GCD Algorithm:* `M-JWA-GCD`

The `makeodd`(x) function removes all the powers of 2 from the integer x. `M-JWA`(u, v, k) is the output matrix of `M-JWA` algorithm described in Fig. 2.

5.1 COMPLEXITY ANALYSIS

Theorem 5.1 *If u and v are two positive integers of at most n bits in length, then* `M-JWA-GCD` *computes* $\gcd(u, v)$ *with a worst case running time of* $O(n^2/\log n)$.

Proof : The proof is similar to those described in [10, 11, 12]. First we assume that only `M-JWA` reductions occur. The computation of the matrix `M-JWA`(u, v, k) costs $O(\log^2 k)$. The computation of $|du - cv|$ and $|bu - av|$ can be computed in $O(n)$ time. Since there are, at most, $O(n/\log k)$ iterations, then we obtain $O(\ (n/\log k)\ \times (n + \log^2 k)\ ) = O(n^2/\log n)$ running time.

Now, we assume that only Euclidean steps occur. Let v_i and q_i, $i = 1, 2, \ldots, s$, be respectively the remainders and quotients sequences obtained in Euclid algorithm, with $s = O(n/\log k)$. The running time is bounded by (up to a constant)

$$\sum_{i=1}^{s} \frac{\log v_i \log q_i}{\log k} + O(\log v_i) + O(\log q_i)$$

$$\leq \frac{n+1}{\log k} \sum_{i=1}^{s} \log q_i + sn = O(n^2/\log n),$$

since $\log k = \Theta(\log n)$ and $\prod_{i=1}^{s} q_i \leq 2^n$.

Finally, the precomputed tables require a memory space of $O(k^2 \log k) = O(n^{0.8} \log n)$ bits, which is no more than the length of the inputs u and v. It also needs at most $O(k^2 \log^2 k) = O(n^{0.8} \log^2 n)$ bit operations in time to compute these tables. *QED*

5.2 EXPERIMENTS

The implementation is written in GNU C Compiler *gcc*, version 2.7 (Stallman, 1991 [2]) with the 3.1.1 Gnu MP library on a Pentium IV, 3.1 GHz Dell PC, running Linux system. The average times are in microseconds (μs). We used the same parameters for both `M-JWA-GCD` and `JWA-GCD` and the code is not optimized. The experiments were done on $N = 10^4$ random numbers u and v of size SIZE words of 32 bits, with $10 \leq SIZE \leq 70$. We used $k = 2^{30}$ with `M-JWA` reduction for $\ell_2(u) - \ell_2(v) \leq 4$, and `Euclidean` step otherwise.

The results described in Table 2 show that `M-JWA-GCD` has a slightly better running time for integers of size less than 30 words of 32 bits. For larger inputs, the parameter $k = 2^{30}$ was not suitable and we suggest to experiment it with more `M-JWA` reductions, i.e.: for $\ell_2(u) - \ell_2(v) \leq C$, with $C > 4$ and larger parameter k, i.e.: $k = 2^{64}$.

6 CONCLUSION

We have shown that a slight modification easily avoids the spurious factors introduced by `JWA`. Although in our experiments `M-JWA` is faster only for integers less then 30 words of 32 bits, it makes the complexity analysis much easier and helps to design other Jebelean like GCD algorithms. Sorenson also proposed in [12] a small modification of the

N	Spurious factor
300	5
1000	151875
2000	122542875
3000	$\sim 1.02\ 10^{15}$
4000	$\sim 2.37\ 10^{15}$
5000	$\sim 1.15\ 10^{19}$
6000	$\sim 2.74\ 10^{25}$
9000	$\sim 9.67\ 10^{43}$

Table 1: Spurious factors in `JWA` with Fibonacci pair of inputs (F_N, F_{N-1}) .

$SIZE$	M-JWA	JWA
10	39.4	42.3
20	117.7	118.4
30	150.1	153.4
40	246.0	240.8
50	353.9	339.5
70	588.7	563.0

Table 2: CPU times in microseconds for `M-JWA` and `JWA` gcd algorithms with 10^4 random integers of $SIZE$ words of 32 bits.

`JWA` algorithm but its GCD algorithm has an $O(n^2/\log n)$ running time on average, and $O(n^2)$ running time in the worst case. We improve this result, since our algorithm has an $O(n^2/\log n)$ running time in the worst case. On the other hand, for very large integers, there are many half-gcd like algorithms [1, 6, 8, 14, 15, 9] that computes the GCD in $O(n \log^2 n \log\log n)$ time, but all these fast algorithms fall down to more basic algorithms at some point of their recursion. Moreover, we observe that Sorenson's reduction (1) is basically a half-gcd like procedure (consider $k = 2^n$) and the cofactors a and b in relation (1) depend only on the least significant bits of u and v. Therefor, one may consider to built a half-gcd like algorithm based on a recursive Sorenson's reduction. This is the direction we intend to next take our research.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison Wesley, 1974

[2] Gnu MP 4.1.2, online reference http://swox.com/gmp/manual, /index.html, 2002.

[3] G.H. Hardy and E.V. Wright, An Introduction To The Theory of Number, Oxford University Press., London, 1979

[4] T. Jebelean, A Generalization of the Binary GCD Algorithm, in Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'93), 1993, 111-116

[5] D.E. Knuth, The Art of Computer Programming, Vol. 2, 3rd ed., Addison Wesley, 1981

[6] D. Lichtblau, Half-GCD and Fast rational recovery, in Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'2005), 2005, 254-258

[7] A.J. Manezes, P.C.van Oorschot, S.,A., Vanstone, Handbook of Applied Cryptography, Vol. 1-2, 2nd ed., CRC Press, 1997

[8] A. Schönhage, Schnelle Berechnung von Kettenbruchentwicklugen, Acta Informatica, 1, 1971, 139-144

[9] M.S. Sedjelmaci, The Accelerated Euclidean Algorithm, Poster talk presented at the International Symposium on Symbolic and Algebraic Computation (ISSAC'2004), University of Cantabria, Santander, Spain, July 4-7, 2004

[10] J. Sorenson, Two Fast GCD Algorithms, J. of Algorithms, 16, 1994, 110-144

[11] J. Sorenson, An Analysis of Lehmer 's Euclidean Algorithm, in Proc. of the International Symposium on Symbolic and Algebraic Computation (ISSAC'95), 1995, 254-258

[12] J. Sorenson, An Analysis of the generalized binary GCD algorithm, High Primes and Misdemeanors, Lectures in Honour of Hugh Cowie Williams, Alf van der Poorten and Andreas Stein ed., Banff, Alberta, Canada, AMS Math Review 2005h:11279, Vol. 41, 2004, 254-258, http://euclid.butler.edu/

[13] J. Sorenson, Lehmer's Algorithm for very Large Numbers, Poster talk presented at ANTS VI, University of Vermont, USA, June 13-18, 2004

[14] D. Stehle, P. Zimmermann, A Binary Recursive Gcd Algorithm, in Proc. of ANTS VI,University of Vermont,USA, June 13-18,2004, 411-425

[15] J. von zur Gathen, J. Gerhard, Modern Computer Algebra1st ed., Cambridge University Press, 1999

[16] K. Weber, Parallel implementation of the accelerated integer GCD algorithm, J. of symbolic Computation (Special Issue on Parallel Symbolic Computation), 21, 1996, 457-466