Fast inverse for big numbers: Picarte's iteration

Claudio Gutierrez and Mauricio Monsalve

Computer Science Department, Universidad de Chile
cgutierr,mnmonsal@dcc.uchile.cl

Abstract. This paper presents an algorithm to compute the inverse of an n-bit integer with $k \geq n$ bits of precision running in time $O(M(k,n))$, where $M(k,n)$ is the time needed to multiply two numbers of k and n bits. This bound improves asymptotically Newton's iteration in this range which gives $O(M(k,k))$, and hence improves asymptotically the best known upper bound for computing the inverse with high precision. Additionally we show that a simple implementation of the algorithm behaves very well in practice for precision $k >> n$, improving largely Newton's iteration and also outperforming the standard multiple precision arithmetic library package GMP for big n.

Introduction

It is well known that the representation of the inverse of an integer consists of a fixed part plus a period, whose lengths depend on the representation base. For example, for decimal representation, given an integer b odd and not divisible by 5, $1/b$ is pure recurring and the length of the period is the order of 10 modulo b [3]. This means that the period could be exponentially long in the length of the decimal representation. For example 23 is represented by 2 decimal digits, while $1/23 = 0.\overline{0434782608695652173913}$ has a period of 22 decimals. Similar analysis holds for any base, in particular for binary representation (cf. Thm. 136, [3]). Thus, for an integer of n digits, computing its inverse with k digits of precision for k in the range $n \leq k \leq 2^n$ is relevant and not trivial.

Given an integer b (assume its binary representation has n bits), the standard routine to compute efficiently its inverse $1/b$ is Newton's iteration, which gives n significative bits of the inverse of b in time $O(M(n,n))$, where $M(x,y)$ is the time taken to multiply an x-bit number by an y-bit number. In fact, $O(M(n,n))$ is currently the best theoretical asymptotical bound for computing the inverse of an n-bit number [4, 1], and also is the method suggested to be used in practice for large numbers ([4], 4.3.3, Division). Specialized packages for multiprecision arithmetic also consider it (e.g. GMP[1] [8]). Note that this bound depends on the multi-

[1] GNU Multiple Precision Arithmetic Library, "the fastest bignum library on the planet!"

plication procedure used. The current best theoretical bound for multiplication $M(n,n)$, which uses Schönhage and Strassen procedure [6], gives $O(n \lg n \lg\lg n)$.

When more bits of precision for $1/b$ are needed, say $k > n$, there seems to be no way to avoid the $O(M(k,k))$ time bound obtained by running Newton's iteration. This bound is got essentially by the cost of the squaring in Newton's iteration formula $x_{i+1} = 2x_i - bx_i^2$.

In this paper we present an algorithm to compute the inverse $1/b$ of an n-bit integer b with $k \geq n$ bits of approximation in time $O(M(k,n))$. Hence, using the current best asymptotical bounds for multiplication plus Shönhage's observation ([4], 4.3.3, Ex. 13) on computing k times n for $k > n$, it gives $O(k \lg n \lg\lg n)$ time, which improves asymptotically the current best bound $O(k \lg k \lg\lg k)$ (which uses Newton's iteration). This is particularly useful in the case we discussed before when k is of the order 2^n.

Additionally, we show experimentally that this algorithm behaves very well in practice for $k >> n$. In fact, with a naive implementation we are able to: 1) illustrate that the behaviour of Picarte's iteration linear time-complexity with parameter k when the size b is fixed; 2) show that in practice Picarte's iteration is much faster than Newton's iteration than what appears in the asymptotical bound difference; and 3) show that Picarte's iteration is faster than current best widely used implementation GMP for big n and large k.

1 Picarte's iteration

In 1861 Ramón Picarte published a table of inverses [5] with remarkable precision for the time. In order to calculate his table of inverses, Picarte starts from the following identity:

$$\frac{a^2}{b} = (2a - b) + \frac{(a-b)^2}{b}.$$

The key observation he made is that the integer part of $\frac{a^2}{b}$ is $2a - b$ plus the integer part of $\frac{(a-b)^2}{b}$, thus:

$$\lfloor \frac{a^2}{b} \rfloor = (2a - b) + \lfloor \frac{(a-b)^2}{b} \rfloor.$$

We can now use the same formula for $\lfloor \frac{(a-b)^2}{b} \rfloor$. Iterating this process q times, where q is such that $a = bq + r$, $0 \leq r < b$, we obtain:

$$\lfloor \frac{a^2}{b} \rfloor = 2aq - bq^2 + \lfloor \frac{r^2}{b} \rfloor. \tag{1}$$

Note that we got a discrete version of Newton's iteration. In fact, if we consider binary representation, and put $a = 2^i$, then $x_i = \lfloor \frac{2^i}{b} \rfloor$ is the binary representation of $1/b$ with i bits of precision. Also, if $2^i = bq + r$, $0 \leq r < b$ is the Euclidean decomposition, we get from (1) the identity

$$x_{2i} = 2^{i+1}x_i - bx_i^2 + \lfloor \frac{r^2}{b} \rfloor, \tag{2}$$

which is essentially a discrete version of the iteration recommended by Knuth for implementing Newton's iteration for high-precision reciprocal (Algorithm R, [4], 4.3.3).

Improved version based on Picarte. The advantage of Picarte's formula (1) for inverse over the classical Newton's one $x_{2i} = 2x_i - bx_i^2$, is not only that Picarte's is discrete and exact, but that it allows to improve the efficiency of the evaluation of it by replacing the expression $2qa - bq^2$ for the equivalent, but simpler to evaluate expression $qa + rq$ (recall that $a = bq + r$, hence $qa = bq^2 + rq$), getting:

$$\lfloor \frac{a^2}{b} \rfloor = qa + rq + \lfloor \frac{r^2}{b} \rfloor. \tag{3}$$

Now, using the same arguments for getting a binary representation used in the derivation of (2), we obtain the following formula, which we will call *Picarte's iteration*:

$$x_{2i} = 2^i x_i + r_i x_i + \lfloor \frac{r_i^2}{b} \rfloor. \tag{4}$$

Note that compared to Newton's iteration (formula (2) above), the evaluation of this expression requires only multiplication of x_i by a bounded number ($r_i < b$) plus a bounded division and a shift. From this iteration formula we obtain:

Theorem 1. *For a n-bit number b, the time complexity of computing $1/b$ with $k \geq n$ binary digits of precision is $O(M(k,n))$, where $M(k,n)$ is the time complexity of multiplying a k-bit number by an n-bit number.*

Proof. The deduction above proves that the iteration (4) computes what is claimed. To calculate the time complexity $T(x)$ of computing x_{2i}, observe that on the right hand side of (4) we have a shifting by i, a multiplication of an i-bit number by an n-bit number $M(i,n)$, a squaring of an n-bit number $M(n,n)$, a division of a $2n$-bit number by an n-bit number $D(2n)$, plus two sums of at most $2i$-bit numbers $S(2i)$. That is:

$$T(2i) = T(i) + M(i,n) + 2S(2i) + M(n,n) + D(2n) + c(\text{shift}, i). \quad (5)$$

By standard techniques (cf. Theorem 4.1 [2]), and the fact that for $i \geq n$, $M(i,n)$ asymptotically dominates over the terms on its right, it follows that for $k \geq n$, $T(k) = O(M(k,n))$.

It is interesting to note that using the above method to compute $1/b$ with $k \geq n$ bits of precision and Schönhage's observation on how to multiply an n-bit number by a k-bit number for $k > n$ ([4], 4.3.3, Ex. 13), we get the following result:

Corollary 1. *For n-bit integers a, b, the time complexity of computing a/b with $k > n$ binary digits of precision is $O(M(k,n))$.*

2 Illustrations of theoretical results and comparison with practical implementations

The aim of the experiments shown in this section is three-fold:

1. To illustrate the behaviour of Picarte's iteration linear time-complexity (of parameter k) when the size b is fixed.

2. To show that in practice Picarte's iteration is much faster than Newton's iteration. We already know from theory that Picarte's is asymptotically faster than Newton's. But the role of constants is hidden in the asymptotic comparison. In fact, the experiments show this difference.

3. To show that Picarte's iteration is faster than current best widely used implementations. As point of illustration, we chose the GMP package, and compared our (naive) implementation of Picarte's iteration with the GMP's division algorithm for precision k. The results show that Picarte's iteration is more efficient for large k.

Picarte's iteration was implemented in C language using the GMP library for big numbers. We also implemented Newton's iteration in a similar way (it differs from Picarte's essentially in two lines). See Algorithms 1 and 2 (code available at `http://www.dcc.uchile.cl/~mnmonsal/picarte`).

All tests were performed on a processor Intel Core 2 Duo with 2 Ghz and 1 Gb RAM running the Fedora Core 6 linux distribution.

Algorithm 1 Picarte's iteration

Input j the number of iterations,
  b the number to invert
Output x the inverse,
  k the precision reached
procedure $Picarte(j, b)$
  $r \leftarrow 2$, $x \leftarrow 0$, $k \leftarrow 1$
  while $j > 0$
    $y_1 \leftarrow x << k$
    $y_2 \leftarrow r \times x$
    $x \leftarrow y_1 + y_2$
    $rr \leftarrow r \times r$
    // Division with remainder
    $y_1, r \leftarrow \lfloor \frac{rr}{b} \rfloor, rest(\frac{rr}{b})$
    $y_2 \leftarrow x + y_1$
    $x \leftarrow y_2$
    $k \leftarrow k + k$
    $j \leftarrow j - 1$
  end while
return x, k

Algorithm 2 Newton's iteration

Input j the number of iterations,
  b the number to invert
Output x the inverse,
  k the precision reached
procedure $Newton(j, b)$
  $r \leftarrow 2$, $x \leftarrow 0$, $k \leftarrow 1$
  while $j > 0$
    $y_1 \leftarrow x << k$
    $xx \leftarrow x \times x$
    $y_2 \leftarrow b \times xx$
    $x \leftarrow y_1 - y_2$
    $rr \leftarrow r \times r$
    $y_1, r \leftarrow \lfloor \frac{rr}{b} \rfloor, rest(\frac{rr}{b})$
    $y_2 \leftarrow x + y_1$
    $x \leftarrow y_2$
    $k \leftarrow k + k$
    $j \leftarrow j - 1$
  end while
return x, k

2.1 Picarte's iteration asymptotic behaviour

GMP uses the current faster multiplication algorithms, so Picarte's asymptotic bound should be linear in k and logarithmic in n. The experiments do not deviate from this prediction. Fig. 1 shows Picarte's iteration when $n = 2^5$, $n = 2^{10}$ and $n = 2^{15}$, where n is the size of the binary representation of b.

2.2 Picarte's iteration versus Newton's iteration

Picarte's iteration improves the performance of Newton's iteration by essentially replacing $-bx_i^2$ for $r_i x_i$ (recall equations (2) and (4)). Hence the cost of each iteration in Picarte's algorithm is one multiplication by a bounded number $M(i, n)$ as compared to two multiplications $M(2i, n) + M(i, i)$ (one unbounded) in Newton's. Also note that $r_i < b$ for all i. Thus, in practice, Picarte's iteration greatly improves the performance of Newton's as Figure 2 shows.

Figure 2 shows Picarte's iteration versus Newton's iteration. Clearly, Picarte's iteration outperforms Newton's iteration. When k is big, greater

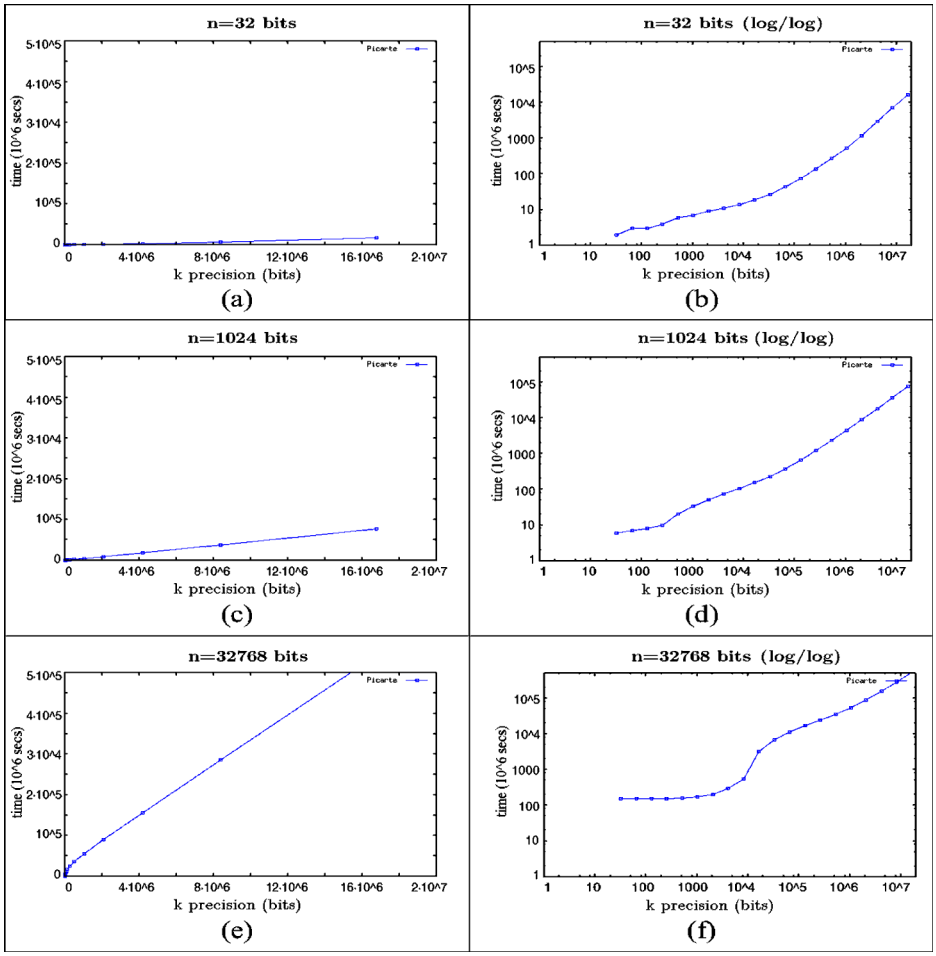

Fig. 1. *Picarte's iteration.* Graphs on the left-hand side, (a), (c) and (e), show the time used by Picarte's iteration to compute $1/b$ as the precision k increases, for, respectively, $n = 2^5$, $n = 2^{10}$ and $n = 2^{15}$ respectively, where n is the size of the binary representation of b. In each graph, the X-axis shows the precision k, and the Y-axis shows the time used to compute $1/b$ with k bits of precision in microseconds. Graphs on the right-hand side, (b), (d) and (f), show the same data as (a), (c) and (e), but in log/log scale.

than $2^{20}(\approx 10^6)$, the difference in time becomes of the order of milliseconds for Picarte's versus seconds and even minutes for Newton's. Also, Picarte's iteration uses less memory, so it can compute the inverse for greater numbers and precisions than Newton's iteration (up to $k = 2^{29}$ using GMP, unlike Newton).

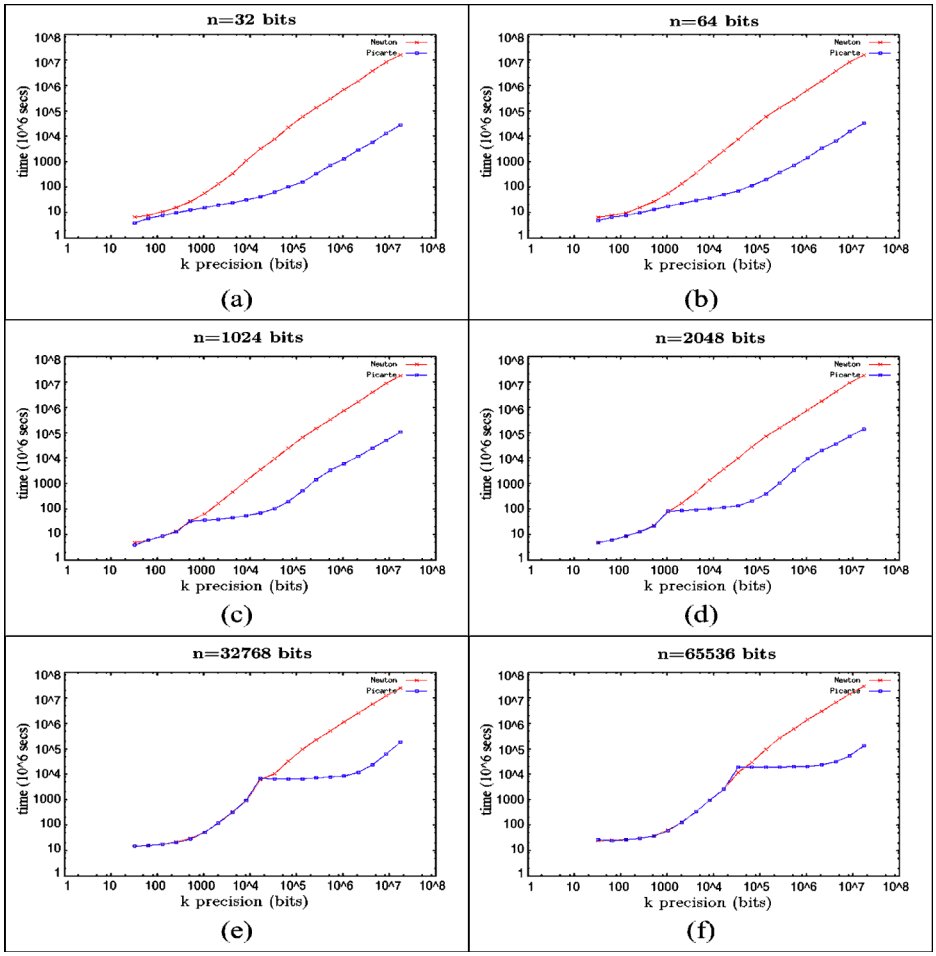

Fig. 2. *Picarte versus Newton.* Graphs show the performance of both algorithms for different sizes of b, ranging from $n = 2^{10}$ (graph (a)) to 2^{16} (graph (f)). Each graph depicts the time in microseconds of computing $1/b$ for different precisions k. All graphs are in log/log scale.

Note that Picarte's iteration and Newton's iteration perform very similarly when the precision required is no more than the size of the binary representation n of b (see Figure 2, graphs (c), (d), (e) and (f)). Picarte's starts working much better than Newton's after that level of precision, that is, when $k \geq n$.

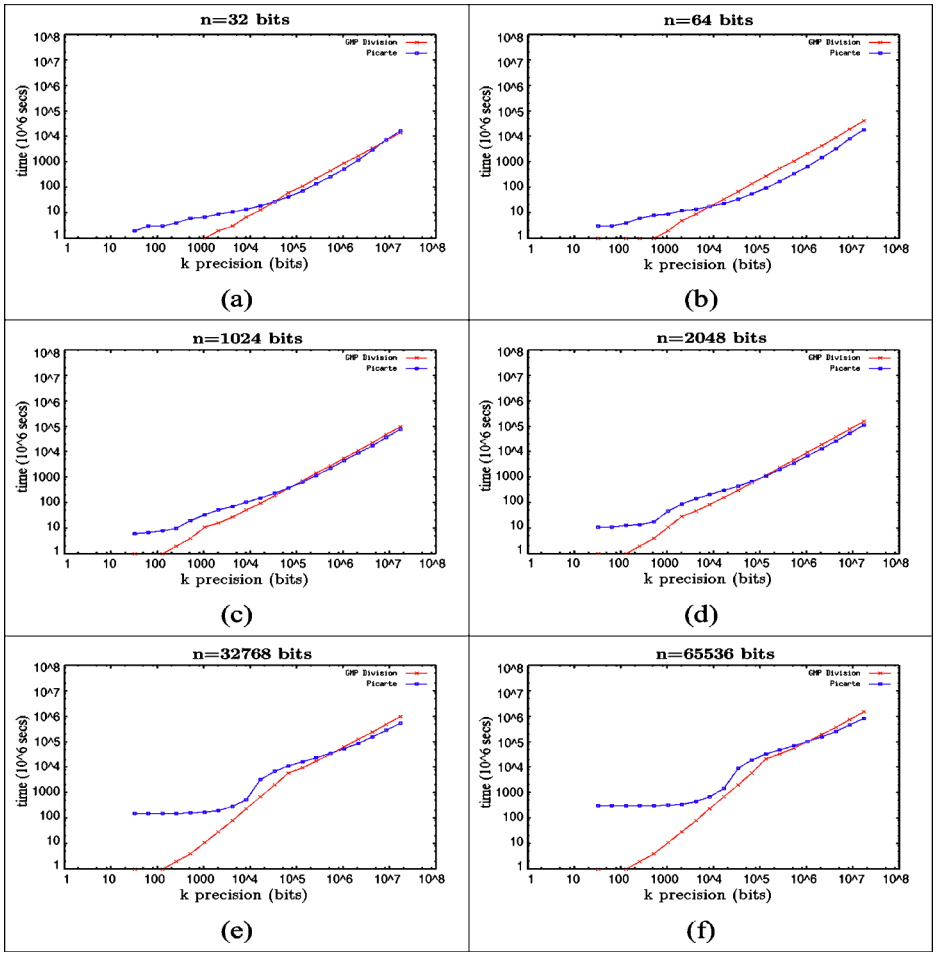

Fig. 3. *Picarte versus GMP.* Graphs show the performance of Picarte versus GMP inverse for different sizes of b, ranging from $n = 2^5$ (graph (a)) to 2^{16} (graph (f)). Each graph depicts the time in microseconds of computing $1/b$ for different precisions k. All graphs are in log/log scale.

2.3 Picarte's iteration and GMP division

GMP current implementation uses base-case division or divide and conquer division when the divisor is big enough. Newton's iteration is not implemented because it seems to surpass divide and conquer division only for very large numbers.[2] As we saw, Picarte's iteration greatly improves

[2] GMP's documentation says: "Newton's method used for division is asymptotically O(M(N)) and should therefore be superior to divide and conquer, but it's believed this would only be for large to very large N."

Newton's iteration. In our experiments, Picarte's iteration also surpasses GMP's division when the precision k needed is big enough (Fig. 3). Note that our implementation is not optimized in any way, it is just the theoretical formula of Picarte's iteration naively implemented.

Figure 3 shows Picarte's iteration versus GMP division. We tested two different divisions for GMP: the integer division $\frac{2^k}{b}$ and the floating point division $\frac{1}{b}$ with k bits of precision, both yielding similar results. The graphs show the integer division.

Figure 3-(a) shows that GMP's division performs better than Picarte's iteration for small b. This is due to the fact that GMP uses a different division algorithm in this range (the 'single limb' division) which takes advantage directly from the arithmetic provided by the hardware. But for large b, Picarte's iteration behaves better than GMP's division for big k. See Fig. 4 for a threshold of the compromise between n and k.

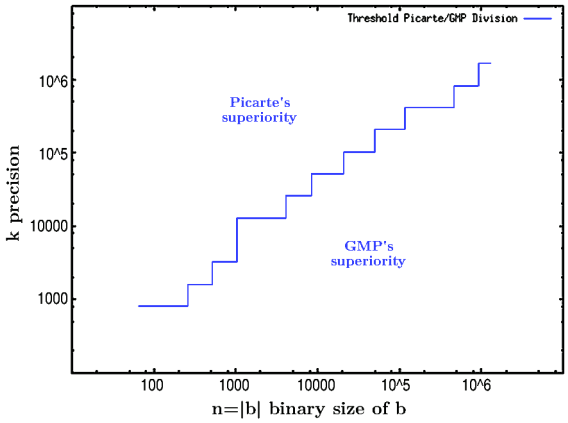

Fig. 4. *Threshold when Picarte surpasses GMP.* Picarte's iteration is faster than GMP's division for each pair (n, k) above the threshold line. If the pair (n, k) is below the line, GMP's division is faster. Built for $n \in \{64, 96, 128, ..., 1572864\}$.

3 Conclusions

We presented an algorithm to find the inverse of an n-bit integer with k bits of precision for $k \geq n$. Our algorithm improves the asymptotical time bound in this range of all previously known algorithms for computing the inverse of integers. In fact the closest one is Algorithm R in [4] as discussed in the introduction. From here it also follows an algorithm for dividing

two n-bit integers with $k \geq n$ bits of precision whose running time is $O(M(k, n))$.

We illustrated these theoretical results with a simple implementation that outperforms current best implemented algorithms for inverse for large n and k.

References

1. C. Burnikel, J. Ziegler. *Fast Recursive Division*, Max-Planck-Institut für Informatik Research Report MPI-I-98-1-022, `http://www.mpi-sb.mpg.de/~ziegler/TechRep.ps.gz`
2. T. H. Cormen, Ch. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, The MIT Press and McGraw-Hill, second edition, 2001.
3. G.H. Hardy, E.M Wright. *An Introduction to the Theory of Numbers*, Third Ed., Oxford, 1953.
4. D. Knuth. *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Third Ed.*, Addison-Wesley, 1999.
5. R. Picarte. *La division reduite a une addition*, Mallet-Bachelier, Paris, 1861.
6. A. Schonhage, V. Strassen. *Schnelle Multiplication grosser Zalen*, Computing 7,1971, 281-292.
7. J. Von Zum Gathen. *Computer Algebra*, Cambridge Univ. Press, 1999.
8. GMP, `www.gmplib.org`