A Binary Recursive Gcd Algorithm

Damien STEHLÉ

Burlington, June 17th, 2004

Joint work with Paul ZIMMERMANN

http://www.loria.fr/~stehle/

damien.stehle@loria.fr

What is the Point in Studying Gcd Algorithms?

- Computing over rational numbers.
- Inverting modulo an integer (RSA, ...).
- Computing continued fractions expansions.
- Rational reconstruction, $p - 1$ method, ECM, ...

- Central question in algorithmics.
- Plenty of problems resemble this one.

Quick History

- $\approx$ 300 BC, Euclid: Euclid's algorithm, quadratic complexity.
- 1938, Lehmer: practical speed-up of Euclid's algorithm (the most significant bits give some quotients).

(1970, Schönhage and Strassen: fast multiplication in time

$$M(n) = O(n \log n \log \log n).$$

)

- 1970, Knuth: recursive version of Lehmer's algorithm, using fast multiplication. Quasi-linear complexity: $O(M(n) \log^4 n)$.
- 1971, Schönhage: precise analysis of Knuth's algorithm. Quasi-linear complexity: $O(M(n) \log n)$.
  $\Rightarrow$ Knuth-Schönhage (KS) algorithm.

How are Gcds Computed in Practice?

- KS algorithm efficient only for very large numbers (15,000 bits).
- Subquadratic algorithms rarely coded (Mathematica, Magma).
- Two usual techniques to speed-up Euclid's algorithm:

  1) Lehmer: "Whenever possible, use only most significant bits".
  2) Binary: "Shifting and adding are much faster than dividing".

- In GNU MP: Sorenson's algorithm.

Our Results

- Incompatible ideas: how to make them work together?

  high bits for Lehmer $\leftrightarrow$ low bits for binary

- Results:
  - Definition of a generalized binary (GB) division with Lehmer's property for low bits.
  - Transposition of the KS algorithm for the GB division.
  - Same complexity with algorithm, proof and code as simple as for polynomials.
  - No tedious "fix-up" procedure.

The Standard Division

- Given $a, b > 0$, there exists a unique pair (q, r) s.t.:
$$a = bq + r, \;\; 0 \leq r < b, \;\; q \geq 0.$$
- E.g.: $a = 157 = 10011101_2$, $b = 59 = 111011_2$, $(q, r) = (2, 39) = (10_2, 100111_2)$.
- Remark: $10011_2 = 19$, $111_2 = 7$ give the same quotient.
- Computing q, r in time $O(M(n))$ by Newton's iteration (theory).
- Computing q, r in time $O(n)$ because $q = O(1)$ (practice).

The Standard Euclidean Algorithm

- $\begin{pmatrix} r_0 = a \\ r_1 = b \end{pmatrix} \to^{q_1} \begin{pmatrix} r_1 \\ r_2 \end{pmatrix} \to^{q_2} \ldots \to^{q_t} \begin{pmatrix} r_t = \gcd(a,b) \\ r_{t+1} = 0 \end{pmatrix}.$
- $\begin{pmatrix} r_0 \\ r_1 \end{pmatrix} = \begin{pmatrix} q_1 & 1 \\ 1 & 0 \end{pmatrix} \ldots \begin{pmatrix} q_t & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_t \\ 0 \end{pmatrix}.$
- Complexity: $t = O(n)$, remainders sizes $O(n)$, and “in general” quotients sizes $O(1)$.
  $\Rightarrow O(n)$ multiplications $(1, n)$, which costs $O(n^2)$.
- To do better, multiplications should be balanced.

The GB Division (1/2)

- "Mirror" of the standard division.
- If $a \neq 0$, $\nu(a)$ and $\mathrm{o}(a)$ are s.t. $a = \mathrm{o}(a) \cdot 2^{\nu(a)}$ and $\mathrm{o}(a)$ is odd.
- Given a, b with $\nu(a) < \nu(b)$, there exists a unique pair (q, r) s.t.:

$$a = \frac{b}{2^{\nu(b)-\nu(a)}} q + r, \;\; \nu(r) > \nu(b), \;\; |q| < 2^{\nu(b)-\nu(a)}.$$

- E.g.: $a = 157 = 10011101_2, \;\; b = 4 \cdot 59 = 11101100_2$.

$$\nu(b) - \nu(a) = 2$$
$$q = -1, \;\; r = a + \tfrac{b}{4} = 11011000_2 \; (= 216)$$
$$\nu(r) = 3 > \nu(b).$$

The GB Division (2/2)

$$q = \mathrm{o}(a) \cdot \mathrm{o}(b)^{-1} \text{ cmod } 2^{\nu(b)-\nu(a)+1}.$$

- Cost of computing q, r by Hensel's lifting: $O(M(n))$ (theory).
- Cost by the naive alg.: $O(n)$ because $q = O(1)$ (practice).
- q only depends of the $\nu(b) - \nu(a) + 1$ last nonzero bits of a and b.

The GB Euclidean Algorithm

		$a+3b/4$ →		$a+3b/8$ →		$a+b/4$ →		$a-23b/32$ →	
a	1100010010011		1001110111100		10011101100000		10001010000000		11000000000000
b	1001110111100		10011101100000		10001010000000		11000000000000		0

gcd=3

$$\begin{bmatrix} 3.2^{12} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -23/32 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1/4 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 3/8 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 3/4 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}$$

The GB Division is not so Weird

- Growth of the remainders, but fewer additional high bits than canceled low bits.
  $\Rightarrow$ It converges in $O(n)$ divisions.
- It does not give the gcd g, but returns $2^k \cdot \mathrm{o}(g)$ for some k.

The KS Algorithm

- With the standard division.
- Two remarks:

  1) Storing the r_j's: $O(n^2)$, storing the q_j's: $O(n)$.

  2) The k MSBs suffice to compute $\approx k/2$ quotients bits.
- KS alg.: recursive use of 2) to get all the q_j's but very few r_j's.
- DIFFICULTY: 2) is only approximately true.

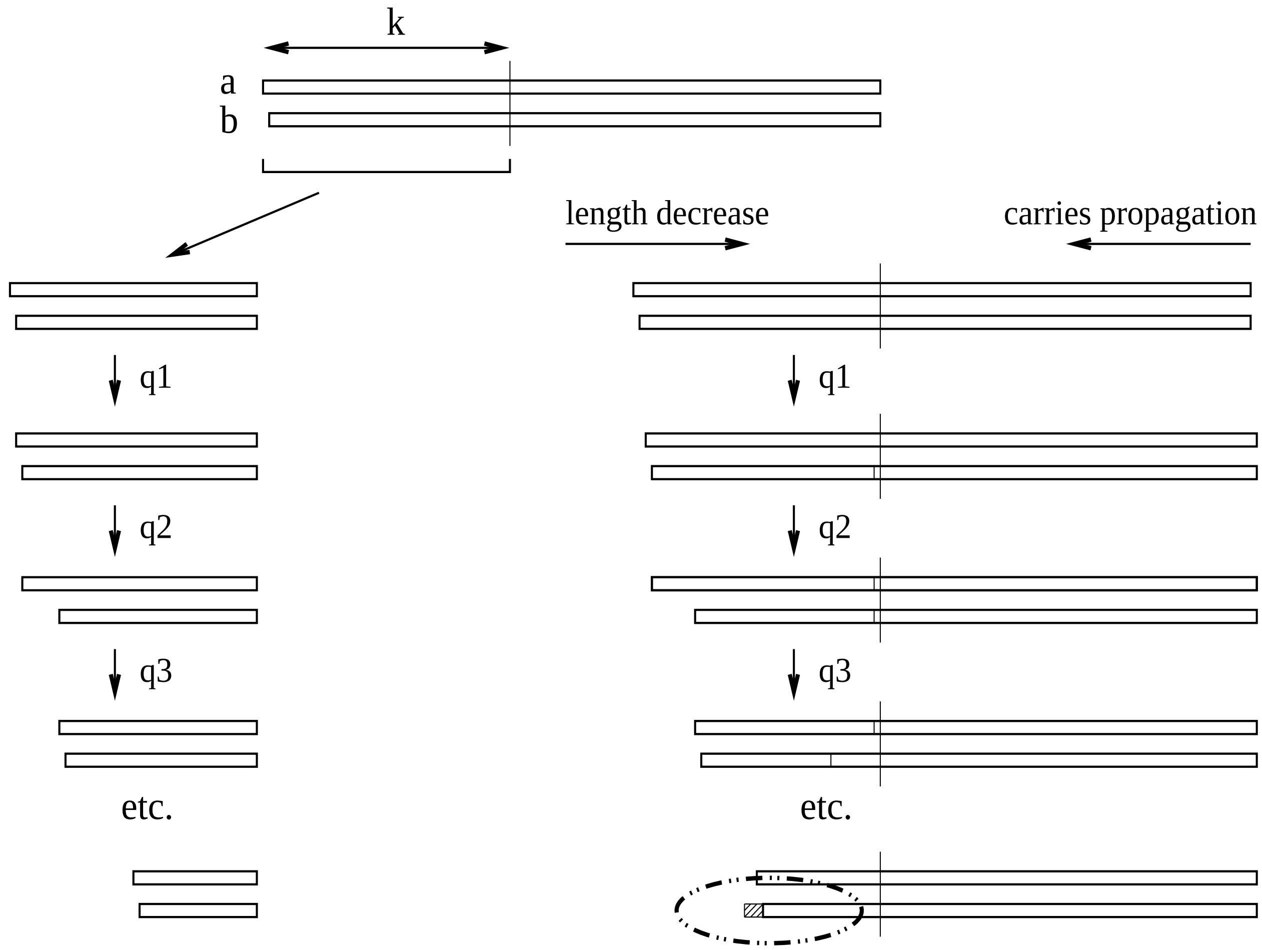
The Need to Correct the Quotients (1/2)

10 digits

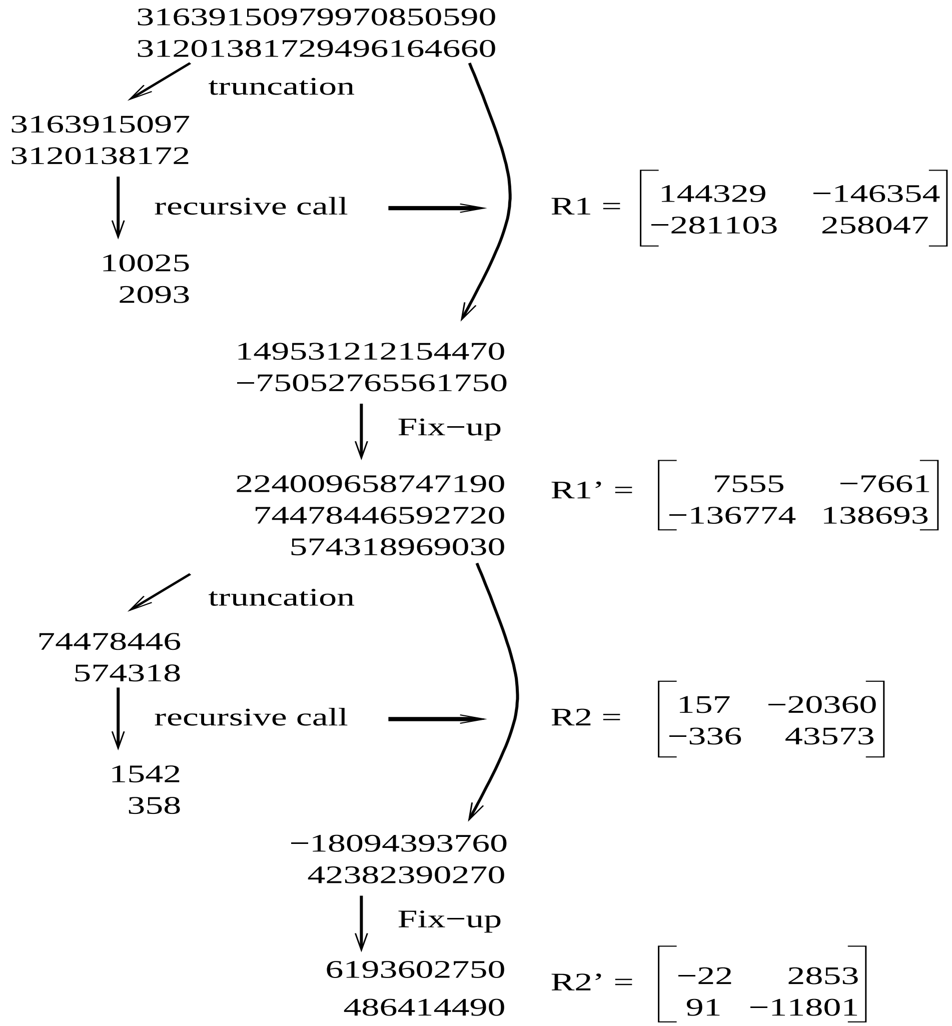
use these quotients

3163915097		31639150979970850590
3120138172	1	31201381729496164660
43776925	71	437769250474685930
11976497	3	119764945793463630
7847434	1	78474413094295040
4129063	1	41290532699168590
3718371	1	37183880395126450
410692	9	4106652304042140
22143	18	224009658747190
12118	1	74478446592720
10025	1	149531212154470
2093		−75052765561750

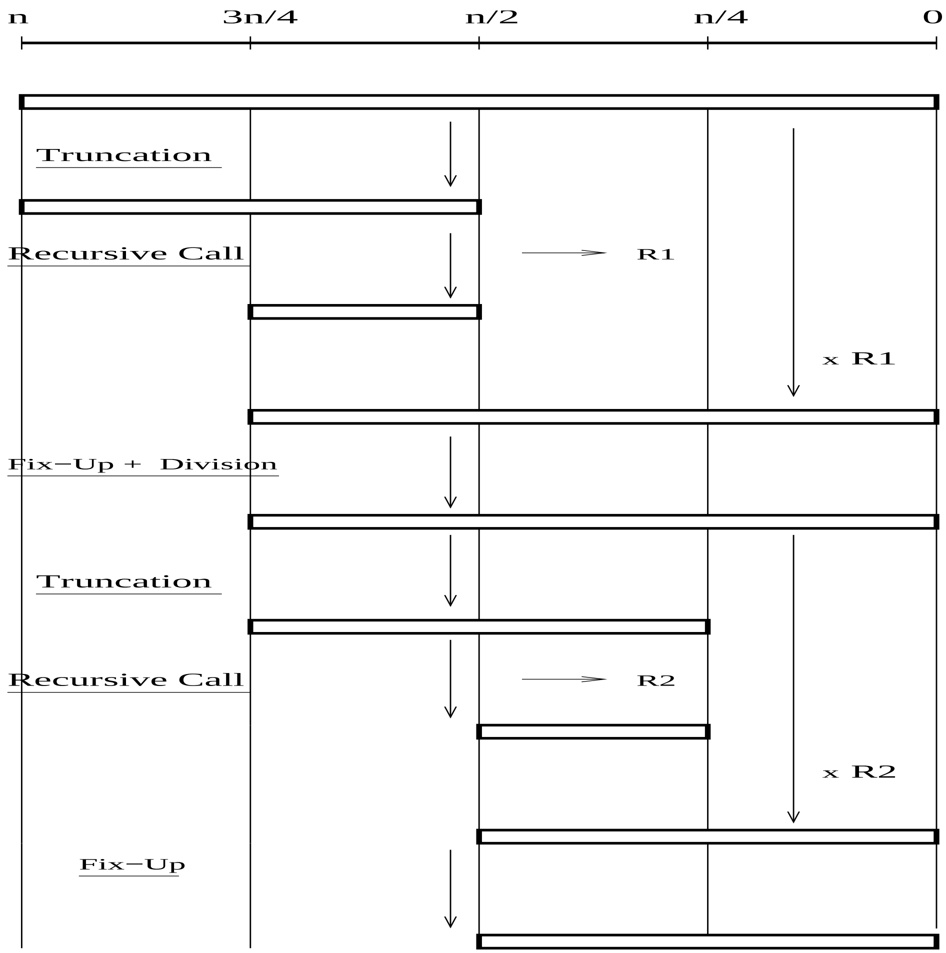
The Need to Correct the Quotients (2/2)



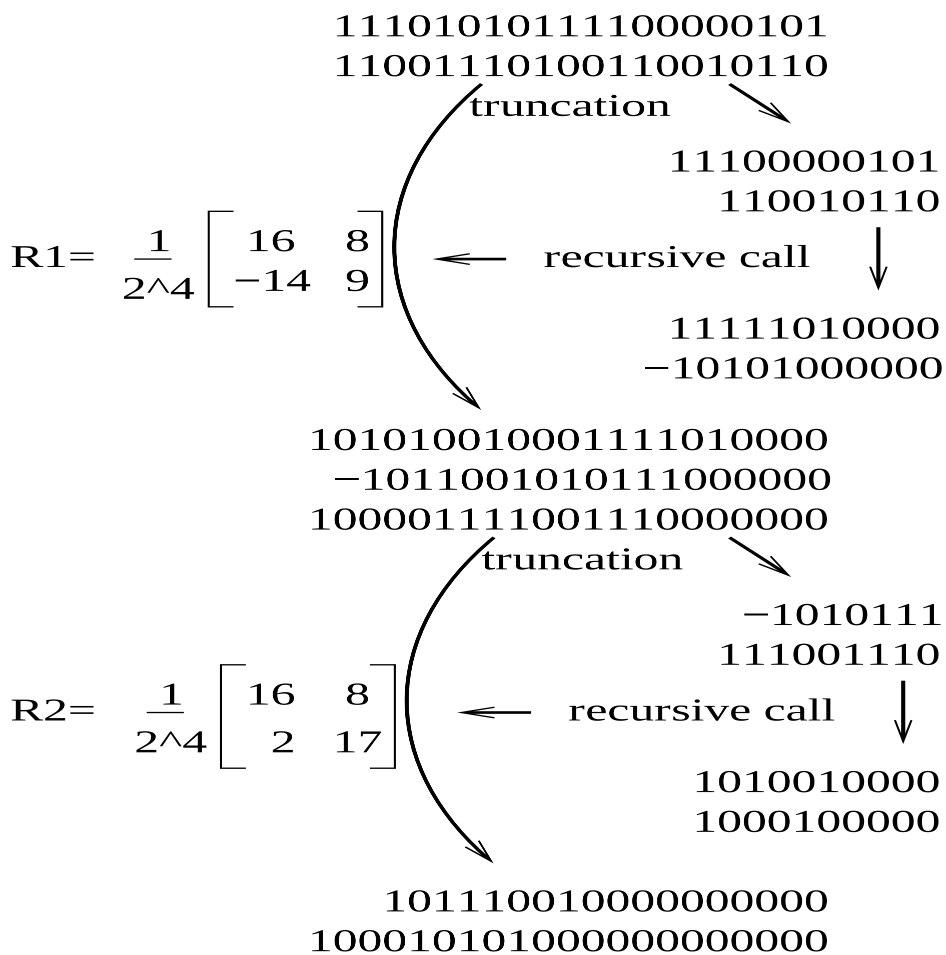
The KS Algorithm, Graphically (1/2)



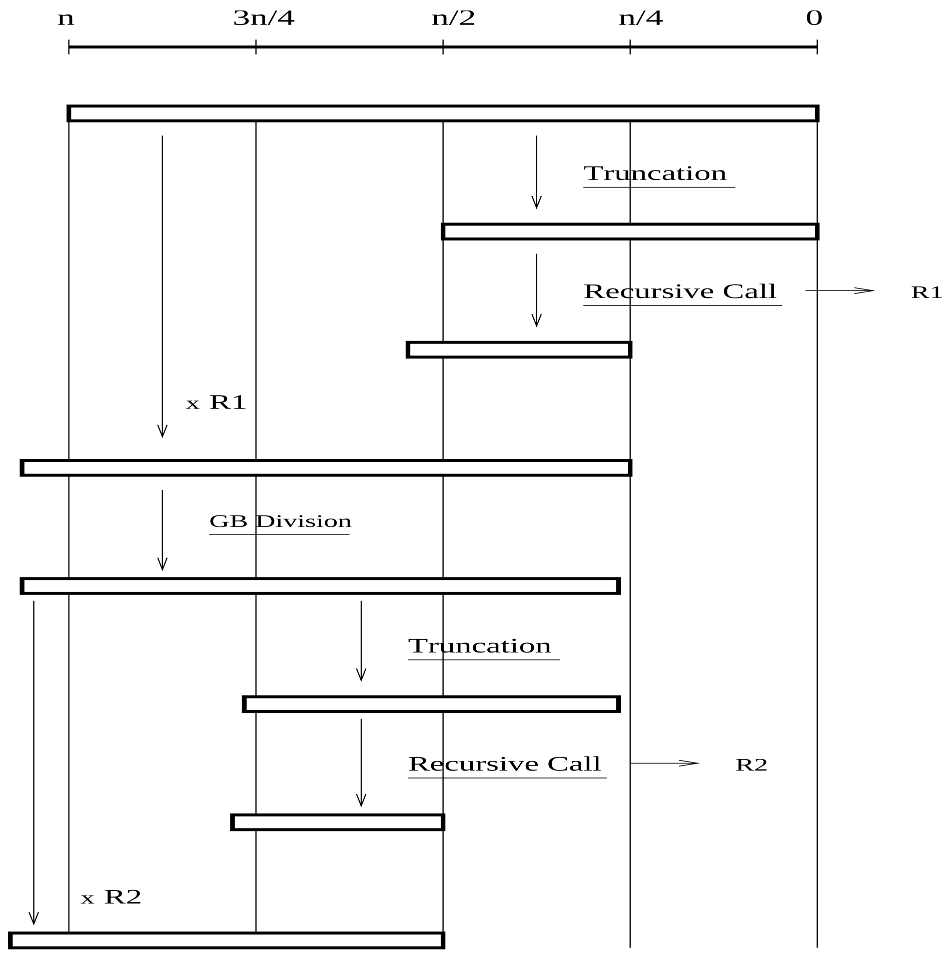
The KS Algorithm, Graphically (2/2)



The Binary Recursive Gcd Algorithm (1/3)



The Binary Recursive Gcd Algorithm (2/3)



The Binary Recursive Gcd Algorithm (3/3)

- Standard division by the left, GB division by the right.
- The algorithm is the same except that:

  1) most significants bits become least significant bits.

  2) the quotients computed recursively are correct.

  $\Rightarrow$ There is nothing to fix.

Why are the Quotients Correct? (1/2)

Let a, b, a', b' with:

- $0 = \nu(a) < \nu(b)$,
- $a' = a \bmod 2^l$ and $b' = b \bmod 2^l$,
- $l \geq 2\nu(b) + 1$.

Let $(q, r) = \mathbf{GB}(a, b)$ and $(q', r') = \mathbf{GB}(a', b')$.

Then: $$q = q' \text{ and } r = r' \bmod 2^{l-\nu(b)}.$$

Why are the Quotients Correct? (2/2)

Let a, b, a', b' with:

- $0 = \nu(a) < \nu(b)$,
- $a' = a \bmod 2^{2k+1}$ and $b' = b \bmod 2^{2k+1}$ for some $k \geq 0$.
- $a, b \longrightarrow r_0, r_1, r_2, \ldots; q_0, q_1, q_2, \ldots$
- $a', b' \longrightarrow r'_0, r'_1, r'_2, \ldots; q'_0, q'_1, q'_2, \ldots$

If r_{i+1} is the first remainder s.t. $\nu(r_{i+1}) > k$, then:

$$\forall j \leq i,\ q_j = q'_j \text{ and } r_{j+1} = r'_{j+1} \bmod 2^{2k+1-\nu(r_j)}.$$

Complexity Analysis

- $H_n = 2H_{\frac{n}{2}+1} + O(M(n))$.
- $F_n = H_n + F_{\alpha n} + O(M(n))$, with $\alpha = \frac{1}{2}(1 + \log \frac{1+\sqrt{17}}{4}) < 1$.
  $\Rightarrow \ F_n, H_n = O(M(n) \log n))$.
- Heuristic calculation of the $O(\cdot)$ constant:
  1) the quotients are small,
  2) the matrix making the lengths decrease by k bits has entries with $\approx k$ bits.

$$\Rightarrow \quad F_n \approx \frac{17}{4} M(n) \log n.$$

Tuning the Algorithm in Practice

- Batching quotients to fill machine words.
- Returning the current remainders along with the transformation matrix in the recursive calls.
- Changing **Half-Gcd** to cancel γn bits instead of $n/2$ (with γ to be optimized), in Karatsuba and Toom-Cook domains.