RESEARCH ANNOUNCEMENT: FASTER FACTORIZATION OF TWO NUMBERS INTO COPRIMES

DANIEL J. BERNSTEIN

Abstract. This paper presents an algorithm that, given positive integers a, b, computes the natural coprime base for $\{a, b\}$ in time $n(\lg n)^{2+o(1)}$, where n is the number of input bits.

My previous paper [1] introduced an algorithm that, given a set S of positive integers, computes the natural coprime base $\operatorname{cb} S$ in time $n(\lg n)^{O(1)}$, where n is the number of input bits. I made no attempt in [1] to optimize the exponent of $\lg n$.

This paper presents an algorithm that computes $\operatorname{cb}\{a, b\}$ in time $n(\lg n)^{2+o(1)}$. It is reasonable to conjecture that the limiting exponent 2 is optimal (for, e.g., a multitape Turing machine): one has $\operatorname{cb}\{a, b\} = \{a, b\} - \{1\}$ if and only if a, b are coprime; the well-known problem of checking coprimality has been stuck at $n(\lg n)^{2+o(1)}$ for thirty years.

Step 1. Swap a, b if necessary so that $a \geq b$. The algorithm will later reduce the input length by at least one third of the length of a.

If $a = 1$, stop.

Step 2. Compute $a_0 = a$, $g_0 = \gcd\{a_0, b\}$, $a_1 = a_0/g_0$, $g_1 = \gcd\{a_1, g_0^2\}$, $a_2 = a_1/g_1$, $g_2 = \gcd\{a_2, g_1^2\}$, and so on, until $g_k = 1$.

For example, if $a = 2^{100}3^{100}$ and $b = 2^{137}3^{13}$, compute $a_0 = 2^{100}3^{100}$, $g_0 = 2^{100}3^{13}$, $a_1 = 3^{87}$, $g_1 = 3^{26}$, $a_2 = 3^{61}$, $g_2 = 3^{52}$, $a_3 = 3^9$, $g_3 = 3^9$, $a_4 = 1$, $g_4 = 1$.

Lower level: The gcd inputs a_i, g_{i-1}^2 are often highly unbalanced. To compute $\gcd\{a_i, g_{i-1}^2\}$, first divide a_i by g_{i-1}^2, and then use any standard fast gcd algorithm to compute $\gcd\{g_{i-1}^2, a_i \bmod g_{i-1}^2\}$. The division takes time at most $n(\lg n)^{1+o(1)}$; the gcd takes time at most $m(\lg m)^{2+o(1)}$ where m is the length of g_{i-1}^2.

All the g's together have length $O(n)$, and k is at most about $\lg n$, so the total time here is at most $n(\lg n)^{2+o(1)}$.

Step 3. Compute $x_0 = g_0/\gcd\{g_0, g_1^\infty\}$, $x_1 = g_1/\gcd\{g_1, g_2^\infty\}$, and so on.

For example, if $a = 2^{100}3^{100}$ and $b = 2^{137}3^{13}$, compute $x_0 = 2^{100}$, $x_1 = 1$, $x_2 = 1$, $x_3 = 3^9$.

Date: 2004.10.09. Permanent ID of this document: `53a2e278e21bcbb7287b81c563995925`. This is a preliminary version meant to announce ideas; it will be replaced by a final version meant to record the ideas for posterity. There may be big changes before the final version. Future readers should not be forced to look at preliminary versions, unless they want to check historical credits; if you cite a preliminary version, please repeat all ideas that you are using from it, so that the reader can skip it.

2000 *Mathematics Subject Classification.* Primary 11Y16.

The author was supported by the National Science Foundation under grant DMS–0140542, and by the Alfred P. Sloan Foundation.

Lower level: Compute each $\gcd\{g_{i-1}, g_i^\infty\}$ as $\gcd\{g_{i-1}, g_i^{2^{e_i}} \bmod g_{i-1}\}$ where e_i is the smallest nonnegative integer satisfying $2^{2^{e_i}} \ge g_{i-1}$. The repeated squarings and gcd take time at most $m(\lg m)^{2+o(1)}$ where m is the total length of g_{i-1}, g_i. The total time here is at most $n(\lg n)^{2+o(1)}$.

Step 4. Compute $y_0 = \gcd\{b, x_0^\infty\}$, $y_1 = \gcd\{g_0, x_1^\infty\}$, $y_2 = \gcd\{b, g_1, x_2^\infty\}$, $y_3 = \gcd\{b, g_2, x_3^\infty\}$, and so on.

For example, if $a = 2^{100}3^{100}$ and $b = 2^{137}3^{13}$, the algorithm computes $y_0 = 2^{137}$, $y_1 = 1$, $y_2 = 1$, $y_3 = 3^{13}$.

Lower level: Compute $b \bmod g_1, b \bmod g_2, \ldots$ with a scaled remainder tree; this takes time $n(\lg n)^{2+o(1)}$ since $b, g_1, g_2, \ldots$ together have length $O(n)$. Then compute $\gcd\{b, g_1\}$ as $\gcd\{b \bmod g_1, g_1\}$; compute $\gcd\{b, g_2\}$ as $\gcd\{b \bmod g_2, g_2\}$; and so on.

Step 5. Recursively print $\mathrm{cb}\{x_0, y_0/x_0\}$; $\mathrm{cb}\{x_1, y_1\}$; $\mathrm{cb}\{x_2, y_2\}$; and so on. Also print $\mathrm{cb}\{a'\} = \{a'\} - \{1\}$ and $\mathrm{cb}\{b'\} = \{b'\} - \{1\}$ where $a' = a/\gcd\{a, b^\infty\}$ and $b' = b/\gcd\{b, a^\infty\}$. Note that a' has already been computed; it equals a_k.

For example, if $a = 2^{100}3^{100}$ and $b = 2^{137}3^{13}$, recursively print $\mathrm{cb}\{2^{100}, 2^{37}\} = \{2\}$ and $\mathrm{cb}\{3^9, 3^{13}\} = \{3\}$. Also print $\mathrm{cb}\{1\} = \{\}$ and $\mathrm{cb}\{1\} = \{\}$. The complete output is $\{2, 3\}$.

I claim that $x_0y_0, x_1y_1, \ldots, a', b'$ are coprime; that $a = a'x_0x_1y_1x_2y_2^3x_3y_3^7\cdots$; that $b = b'y_0y_1y_2y_3\cdots$; and that $y_0x_1y_1x_2y_2\cdots$, the product of inputs to the recursive calls, is at most $ab/a^{1/3} \le (ab)^{5/6}$. Each of these facts can be checked from the following table of ord_p values, expressed in terms of $e = \mathrm{ord}_p\, a$ and $f = \mathrm{ord}_p\, b$:

g_0	g_1	g_2	g_3	$\ldots$	x_0	y_0	x_1	y_1	x_2	y_2	$\ldots$	a'	b'	
0	0	0	0	$\ldots$	0	0	0	0	0	0	$\ldots$	0	f	if $e = 0$
e	0	0	0	$\ldots$	e	f	0	0	0	0	$\ldots$	0	0	if $0 < e \le f$
f	$e-f$	0	0	$\ldots$	0	0	$e-f$	f	0	0	$\ldots$	0	0	if $f < e \le 3f$
f	$2f$	$e-3f$	0	$\ldots$	0	0	0	0	$e-3f$	f	$\ldots$	0	0	if $3f < e \le 7f$
														$\vdots$
0	0	0	0	$\ldots$	0	0	0	0	0	0	$\ldots$	e	0	if $f = 0 < e$

Consequently the outputs of the algorithm are coprime; a and b are products of powers of the outputs; and the recursion multiplies the total time by a bounded factor.

Note that one can easily factor a, b over $\mathrm{cb}\{a, b\}$ by tracing the factorizations $a = a'x_0x_1y_1x_2y_2^3x_3y_3^7\cdots$ and $b = b'y_0y_1y_2y_3\cdots$ through the recursion.

References

[1] Daniel J. Bernstein, *Factoring into coprimes in essentially linear time*, to appear, Journal of Algorithms. ISSN 0196–6774. URL: `http://cr.yp.to/papers.html`. ID `f32943f0bb67a9317d4021513f9eee5a`.

Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, Chicago, IL 60607–7045

E-mail address: `djb@cr.yp.to`