RESEARCH ANNOUNCEMENT: FASTER FACTORIZATION 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. This paper also presents an algorithm that, given a set S of positive integers, computes the natural coprime base for S in time $n(\lg n)^{4+o(1)}$.

1. Introduction

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 $\lg n$.

Section 2 of 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.

Section 4 of this paper presents an algorithm that computes $\operatorname{cb}(\{a\} \cup Q)$, where Q is any coprime set, in time $n(\lg n)^{2+o(1)}$. Substitute this algorithm into Algorithms 17.3 and 18.1 of [1] to obtain $\operatorname{cb}(P \cup Q)$ in time $n(\lg n)^{3+o(1)}$ and $\operatorname{cb} S$ in time $n(\lg n)^{4+o(1)}$. I'm not willing to conjecture that the 3 and 4 are optimal.

This is a very early draft. I'm confident in the basic structure of the algorithms, but there could be some silly omissions, and of course the proofs need vastly more detail.

2. Computing a coprime base for two positive integers

The following algorithm computes $\operatorname{cb}\{a, b\}$, given positive integers a and b, in time $n(\lg n)^{2+o(1)}$.

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

Date: 2004.11.03. 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.

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 $n(\lg n)^{1+o(1)}$; the gcd takes time $m(\lg m)^{2+o(1)}$ 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 $n(\lg n)^{2+o(1)}$.

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 $x_0 = 2^{100}$, $x_1 = 1$, $x_2 = 1$, $x_3 = 3^9$.

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 $m(\lg m)^{2+o(1)}$ where m is the total length of g_{i-1}, g_i. The total time here is $n(\lg n)^{2+o(1)}$.

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: Use a scaled remainder tree to compute $b \bmod g_1, b \bmod g_2, \ldots$; 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 $\operatorname{cb}\{a, b\}$ by tracing the factorizations $a = a' x_0 x_1 y_1 x_2 y_2^3 x_3 y_3^7 \cdots$ and $b = b' y_0 y_1 y_2 y_3 \cdots$ through the recursion.

3. An algorithm without a catchy name

The following algorithm computes $\gcd\{s, p^\infty\}$ for each s in a multiset S and for each p in a nonempty coprime set P. It takes time $(k+1)n(\lg n)^{2+o(1)}$ if $\#P \le 2^k$.

See [2] and [3] for similar algorithms.

Step 1. If $\#P = 1$: Find $p \in P$. Use a scaled remainder tree to compute $p \bmod s$ for each $s \in S$. Compute $\gcd\{s, p^\infty\}$ as $\gcd\{s, (p \bmod s)^\infty\}$. This takes time $n(\lg n)^{2+o(1)}$.

Assume from now on that $\#P \ge 2$.

Step 2. Select $Q \subseteq P$ with $\#Q = \lfloor \#P/2 \rfloor$. Use a product tree to compute $y = \prod_{p \in Q} p$. This takes time $n(\lg n)^{2+o(1)}$.

Step 3. Use a scaled remainder tree to compute $y \bmod s$ for each $s \in S$. This takes time $n(\lg n)^{2+o(1)}$.

Step 4. Compute $\gcd\{s, (y \bmod s)^\infty\} = \gcd\{s, y^\infty\}$ for each $s \in S$. This takes time $n(\lg n)^{2+o(1)}$.

Step 5. Apply the algorithm recursively to $\{\gcd\{s, y^\infty\} : s \in S\}$ and Q; separately handle each s for which $\gcd\{s, y^\infty\} = 1$. This produces $\gcd\{\gcd\{s, y^\infty\}, p^\infty\} = \gcd\{s, p^\infty\}$ for each $p \in Q$.

Apply the algorithm recursively to $\{s/\gcd\{s, y^\infty\} : s \in S\}$ and $P - Q$; separately handle each s for which $s/\gcd\{s, y^\infty\} = 1$. This produces $\gcd\{s/\gcd\{s, y^\infty\}, p^\infty\} = \gcd\{s, p^\infty\}$ for each $p \in P - Q$.

The product of inputs at each level of recursion is exactly the original product of inputs, so the total input size at each level of recursion is $O(n)$.

4. Extending a coprime base

The following algorithm computes $\operatorname{cb}(\{a\} \cup Q)$, where Q is coprime, in time $n(\lg n)^{2+o(1)}$.

Step 1. Use a product tree to compute $b = \prod_{q \in Q} q$. This takes time $n(\lg n)^{2+o(1)}$.

Step 2. Define, and compute, $a_0, g_0, a_1, g_1, \ldots, x_0, x_1, \ldots, y_0, y_1, \ldots, a'$ exactly as in Section 2. This takes time $n(\lg n)^{2+o(1)}$.

Step 3. Write $y_i(q)$ for $\gcd\{q, y_i^\infty\}$. Compute $y_0(q), y_1(q), y_2(q), \ldots$ for each $q \in Q$ as explained in Section 3. This takes time $n(\lg n)^{2+o(1)} \lg\lg n = n(\lg n)^{2+o(1)}$. Check for and discard 1's so that they do not slow down subsequent computations.

Step 4. Use product trees to compute $z_0 = \prod_q y_0(q)$, $z_1 = \prod_q y_1(q)$, etc. This takes time $n(\lg n)^{2+o(1)}$.

Notice that $z_1 z_2^3 z_3^7 \cdots$ divides a. Indeed, take any prime p dividing $z_1 z_2^3 z_3^7 \cdots$. Recall that $y_1, y_2, \ldots$ are coprime, so p divides z_i for a unique i. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$; then $(2^i - 1)f < e \le (2^{i+1} - 1)f$. Furthermore, p divides a unique $q \in Q$, and $f = \operatorname{ord}_p q = \operatorname{ord}_p z_i$ by definition of b and z_i, so $(2^i - 1)\operatorname{ord}_p z_i < \operatorname{ord}_p a$.

Step 5. Use a scaled remainder tree to compute $a \bmod z_0, a \bmod z_1^3, a \bmod z_2^7, \ldots$. This takes time $n(\lg n)^{2+o(1)}$, since the product $z_1^3 z_2^7 \cdots$ divides a^3.

Step 6. Use scaled remainder trees to compute $(a \bmod z_0) \bmod y_0(q) = a \bmod y_0(q)$ for each q; $(a \bmod z_1^3) \bmod y_1(q)^3 = a \bmod y_1(q)^3$ for each q; $(a \bmod z_2^7) \bmod y_2(q)^7 = a \bmod y_2(q)^7$ for each q; etc. This takes time $n(\lg n)^{2+o(1)}$.

Step 7. Compute $\gcd\{a, y_0(q)\}$, $\gcd\{a, y_1(q)^3\}$, $\gcd\{a, y_2(q)^7\}$, etc. This takes time $n(\lg n)^{2+o(1)}$.

Observe that $\gcd\{a, y_0(q)\}$, $\gcd\{a, y_1(q)^3\}$, $\gcd\{a, y_2(q)^7\}$, etc. are the same as $\gcd\{a, y_0(q)^\infty\}$, $\gcd\{a, y_1(q)^\infty\}$, $\gcd\{a, y_2(q)^\infty\}$, etc. Consider, for example, a prime p dividing $\gcd\{a, y_2(q)^\infty\}$. Recall that $3f < e \le 7f$ where $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b = \operatorname{ord}_p y_2(q)$; thus $\operatorname{ord}_p \gcd\{a, y_2(q)^\infty\} = e = \operatorname{ord}_p \gcd\{a, y_2(q)^7\}$.

Step 8. Print $\operatorname{cb}\{y_i(q), \gcd\{a, y_i(q)^\infty\}\}$ for each i and q. Also print $q/y_0(q)y_1(q)\cdots$ for each q, and print a'. This takes time $n(\lg n)^{2+o(1)}$.

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`.

[2] Daniel J. Bernstein, *How to find small factors of integers*, accepted to Mathematics of Computation; now being revamped. URL: `http://cr.yp.to/papers.html`.

[3] Daniel J. Bernstein, *How to find smooth parts of integers*, draft. URL: `http://cr.yp.to/papers.html#smoothparts`. ID `201a045d5bb24f43f0bd0d97fcf5355a`.

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`