Factoring into coprimes in essentially linear time

Daniel J. Bernstein

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

Abstract

Let S be a finite set of positive integers. A "coprime base for S" means a set P of positive integers such that (1) each element of P is coprime to every other element of P and (2) each element of S is a product of powers of elements of P. There is a natural coprime base for S. This paper introduces an algorithm that computes the natural coprime base for S in essentially linear time. The best previous result was a quadratic-time algorithm of Bach, Driscoll, and Shallit. This paper also shows how to factor S into elements of P in essentially linear time. The algorithms use solely multiplication, exact division, gcd, and equality testing, so they apply to any free commutative monoid with fast algorithms for those four operations; for example, given a finite set S of monic polynomials over a finite field, the algorithms factor S into coprimes in essentially linear time. These algorithms can be used as a substitute for prime factorization in many applications.

Keywords: Factoring into coprimes; coprime bases; factor refinement

1. Introduction

It appears to be difficult to factor most integers into primes. The point of this paper is that it is easy to factor integers into *coprimes*.

Given a finite set S of positive integers, I can construct a set P, with any two distinct elements of P coprime, and factor every element of S into elements of P, in essentially linear time. The best previous result was a quadratic-time algorithm by Bach, Driscoll, and Shallit in [2].

Similarly, given a finite set S of monic polynomials in one variable over a finite field, I can factor S into coprimes in essentially linear time. For comparison, an algorithm of Kaltofen and Shoup in [21] factors polynomials into primes in subquadratic random time.

This paper presents my algorithms and proves that they run in essentially linear time. Priority dates: I announced these results without proof in November 1995, and posted a complete draft of this paper in October 1997.

Warning: The algorithms in this paper are *not* designed to be as fast as possible. They are designed to be as *simple* as possible, under the constraints of (1) running in essentially linear time and (2) using a limited set of arithmetic operations. There are many ways to save more time; these speedups are important but are beyond the scope of this paper.

Applications. Factoring into coprimes is often an adequate substitute for factoring into primes. See, e.g., [13]; [4]; [20, Section 3]; [14] and [33, Section 3.1]; [16, Remark 6.8] and [2, page 201]; [27]; [28, Section 4.6] and [9]; [29]; [17] and [8]; [5]; [31, Theorem 5]; and [22, page 427].

Notation and terminology. General-purpose notation: $\{\}$ means the empty set. When S is a set, $\#S$ means the cardinality of S. When S and T are sets, $S - T$ means $\{x \in S : x \notin T\}$. When S is a finite set, $\operatorname{prod} S$ means $\prod_{x \in S} x$. When a proposition appears inside brackets, it means 1 if the proposition is true, 0 otherwise; for example, $[2 < 3] = 1$.

Notation and terminology specific to this paper: bit is defined in Section 17; cb is defined in Section 7; coprime bases are defined in Section 4; λ is defined in Section 9; lg is defined in Section 8; ls is defined in Section 12; M-time is defined in Section 8; μ is defined in Section 8; ord is defined in Section 6; $\operatorname{ppi}, \operatorname{ppo}, \operatorname{ppg}, \operatorname{pple}$ are defined in Section 11; reduce is defined in Section 19; split is defined in Section 15; τ is defined in Section 9.

"Essentially linear time" means time $b^{1+o(1)}$ where b is the number of input bits.

Acknowledgments. The author was supported by the National Science Foundation under grant DMS–9600083.

Thanks to Igor Shparlinski and two anonymous referees for their comments.

2. Outline of the paper

This paper is organized into four parts.

Part I. Existence and uniqueness. Section 4 proves that every finite set S has a finite "coprime base." The proof uses greatest common divisors and exact division (i.e., division where the remainder is known to be 0) to construct the coprime base.

Sections 6 and 7 show that there is only one coprime base that can be obtained from S via multiplication, exact division, and greatest common divisors: the "natural coprime base" for S, written $\operatorname{cb} S$.

Bach, Driscoll, and Shallit stated their results twice, once for integers and once for polynomials. I have instead abstracted the algebraic properties of integers and polynomials that make the constructions work. The setting for the Bach-Driscoll-Shallit algorithm is a "Noetherian coid with cancellation," defined in Section 3. The setting for natural coprime bases, and for my algorithms, is a "free coid," defined in Section 5. Readers not interested in maximum generality can skip Sections 3 and 5, and instead remember the following facts: the set of positive integers is a free coid; the set of monic polynomials in one variable over a field is a free coid; every free coid is a Noetherian coid with cancellation.

Part II. Two-element sets. Sections 10 through 13 give several constructions culminating in Algorithm 13.2, which finds the natural coprime base for any two-element set. The most important idea is explained in Section 9. Algorithm 13.2 takes essentially linear time, given essentially-linear-time subroutines for multiplication, exact division, and gcd, as discussed in Section 8.

Part III. Finite sets. Sections 14 through 18 give several further constructions culminating in Algorithm 18.1, which finds the natural coprime base for any finite set in essentially linear time.

Part IV. Factorization. To factor a set S into coprimes, first construct a coprime base P for S, and then factor S into elements of P. Sections 19 through 21 show how to carry out the factorization, given S and P, in essentially linear time.

PART I. EXISTENCE AND UNIQUENESS

3. Coids and maximal common divisors

A **coid** is a set with a commutative associative binary operation, written $(a,b) \mapsto ab$, and a neutral element, written 1. Commutativity means $ab = ba$. Associativity means $(ab)c = a(bc)$. Neutrality means $1a = a = a1$.

The word "coid" is nonstandard. It is an abbreviation of "commutative monoid," and an abbreviation of "commutative semigroup with identity"; here **semigroup** means a set with an associative binary operation, and **monoid** means a semigroup with a neutral element. "Coid," like "monoid," should be pronounced to rhyme with "overjoyed."

Divisibility. When $a = bq$ for some q, a is a **multiple of** b; a is **divisible by** b; b **divides** a; b is a **divisor of** a. If a divides b and b divides c then a divides c.

Cancellation. A coid **has cancellation** if $q = r$ whenever $bq = br$. In other words, when c is a multiple of b, there is a unique q such that $c = bq$; this q is denoted c/b.

Noetherian coids. Let S be a subset of a coid. An element of S is **minimal** if it is not divisible by any other elements of S. An element of S is **maximal** if it does not divide any other elements of S. An element of S is **greatest** if it is divisible by all elements of S.

A coid H is **Noetherian** if every nonempty subset of H has a minimal element.

Combinatorial coids. Elements a, b of a coid are **associates** if a divides b and b divides a. A coid is **combinatorial** if $a = b$ whenever a and b are associates. Observe that every Noetherian coid is combinatorial: if a and b are associates with $a \neq b$ then $\{a, b\}$ has no minimal element.

One can systematically replace "equal" by "associate," replace "minimal" by "minimal up to associates," etc., obtaining definitions and results that apply to non-combinatorial coids. I prefer to avoid unnecessary complexity: given a general coid, one can simply consider the combinatorial coid of classes of associates.

Examples. The set of positive integers, with integer multiplication, is a Noetherian coid with cancellation. Equivalently: The set of nonzero ideals of $\mathbf{Z}$, with ideal multiplication, is a Noetherian coid with cancellation.

Let k be a field. The set of monic polynomials in the univariate polynomial ring $k[x]$, with polynomial multiplication, is a Noetherian coid with cancellation. Equivalently: The set of nonzero ideals of $k[x]$ is a Noetherian coid with cancellation.

Generalizing both examples: The set of nonzero ideals of a Dedekind domain is a Noetherian coid with cancellation. Even more generally, the set of cancellable ideals of a Noetherian domain is a Noetherian coid with cancellation. See [8].

Theorem 3.1. *Let B be a nonempty subset of a Noetherian coid with cancellation. Then there exists a maximal common divisor of B.*

Here a **common divisor of** B means a divisor of every element of B.

Proof. Select $b \in B$. Define $S = \{b/d : d \text{ is a common divisor of } B\}$. Observe that $b \in S$. Thus S has some minimal element, say b/g, where g is a common divisor of B. If g divides another common divisor d of B, then b/d divides b/g; but $b/d \in S$, so $b/d = b/g$ by minimality of b/g, so $d = g$. Hence g is a maximal common divisor of B. $\square$

Theorem 3.2. *Let a, b, g be elements of a Noetherian coid with cancellation. If g is a maximal common divisor of $\{a, b\}$ then a/g is coprime to b/g.*

Here c is **coprime to** d if the only common divisor of $\{c, d\}$ is 1. Some authors say "relatively prime to d" or simply "prime to d."

Proof. If d divides a/g and b/g then dg divides a and b. Certainly g divides dg, so by maximality $g = dg$, so $d = 1$ by cancellation. $\square$

Theorem 3.3. *Every element of a Noetherian coid with cancellation can be written as a product $q_1 q_2 \cdots q_n$ for some $n \geq 0$ and some irreducibles $q_1, q_2, \ldots, q_n$.*

Here p is an **irreducible** if $p \neq 1$ and p cannot be written in the form ab with $a \neq 1$ and $b \neq 1$.

Proof. Suppose that the coid has elements that are not products of irreducibles. Let z be a minimal such element.

Case 1: $z = 1$. Then z is the product of 0 irreducibles. Contradiction.

Case 2: $z = ab$ for some a, b with $a \neq 1$ and $b \neq 1$. Then a divides z and $a \neq z$. The minimality of z implies that a is a product of irreducibles. Similarly, b is a product of irreducibles. Thus z is a product of irreducibles. Contradiction.

Case 3: z is irreducible. Then z is the product of 1 irreducible. Contradiction. $\square$

Theorem 3.4. *Let p, a be elements of a Noetherian coid with cancellation. Assume that $p \neq 1$. Then there is a unique integer $m \geq 0$ such that p^m divides a and p^{m+1} does not divide a.*

Proof. Suppose that p^{m+1} divides a for every $m \geq 0$. Then the set $\{a/p, a/p^2, \ldots\}$ does not have a minimal element: a/p^m is divisible by a/p^{m+1}, and $a/p^m \neq a/p^{m+1}$ since $p \neq 1$. Contradiction.

Find the smallest integer $m \geq 0$ such that p^{m+1} does not divide a. If $m = 0$ then $p^m = 1$ so p^m divides a. If $m \geq 1$ then p^m divides a by minimality of m.

Uniqueness: $p^0, p^1, \ldots, p^m$ divide a, while $p^{m+1}, p^{m+2}, \ldots$ do not. $\square$

4. Coprime bases

Let P and S be subsets of a Noetherian coid H with cancellation. The **coid generated by** P is the smallest subset of H that contains $P \cup \{1\}$ and is closed under multiplication; more concretely, it is the set of products of powers of elements of P. P is a **base for** S if S is contained in the coid generated by P; i.e., each element of S is a product of powers of elements of P.

P is **coprime** if each element of P is coprime to every other element of P. (Some authors instead say "P is pairwise coprime," saving "P is coprime" for the less important concept that $\gcd P = 1$. A few authors say "P is gcd-free," as if elements of P were somehow immune to the gcd operation.)

P is a **coprime base for** S if P is coprime and P is a base for S.

Theorem 4.1. *Let H be a Noetherian coid with cancellation. Every finite subset of H has a finite coprime base.*

Proof. Suppose not. Select a finite subset S of H, without a finite coprime base, in such a way that $\operatorname{prod} S$ is minimal.

S cannot be coprime—otherwise it is a finite coprime base for itself. Thus there are two distinct elements $a, b \in S$ with a not coprime to b.

By Theorem 3.1, there is a maximal common divisor g of $\{a, b\}$. If $g = 1$ then a is coprime to b; thus $g \neq 1$. Define $T = (S - \{a, b\}) \cup \{g, a/g, b/g\}$. Then $\operatorname{prod} T$ divides $((\operatorname{prod} S)/ab)g(a/g)(b/g) = (\operatorname{prod} S)/g$; so $\operatorname{prod} T$ is a divisor of $\operatorname{prod} S$ different from $\operatorname{prod} S$. The minimality of $\operatorname{prod} S$ implies that T has a finite coprime base, say P.

Now $a = g(a/g)$ and $b = g(b/g)$ are products of elements of T, and thus are in the coid generated by P. All other elements of S are elements of T, and thus are in the coid generated by P. Hence P is a base for S. Contradiction. $\square$

5. Free coids and greatest common divisors

A **free coid** is a Noetherian coid with cancellation in which a is coprime to bc whenever a is coprime to both b and c.

Bach, Driscoll, and Shallit suggested in [2, page 216] (in different language) that free coids would form a "suitable abstract setting" for factoring into coprimes. The algorithms in [2] actually work for arbitrary Noetherian coids with cancellation, but free coids are the setting for most of this paper.

Examples. The set of nonzero ideals in a Dedekind domain is a free coid. The point is that, in a Dedekind domain, the sum $a+b$ of two ideals a and b is a maximal common divisor of $\{a,b\}$. Therefore, if a is coprime to both b and c, then $a+b=1$ and $a+c=1$, so $a+bc=1a+bc=(1+b)a+bc=a+ba+bc=a+b(a+c)=a+b1=a+b=1$ by elementary ideal arithmetic, so a is coprime to bc.

In particular, the set of positive integers is a free coid, and if k is a field then the set of monic polynomials in $k[x]$ is a free coid.

Theorem 5.1. *Let a,b,c be elements of a free coid. If a divides bc, with a coprime to b, then a divides c.*

Proof. By Theorem 3.1, $\{a,c\}$ has a maximal common divisor g. By Theorem 3.2, a/g is coprime to c/g. By hypothesis, a/g is coprime to b. Thus a/g is coprime to $b(c/g)=bc/g$. But a/g divides bc/g, so a/g must be 1. Thus $a=g$ divides c. $\square$

Theorem 5.2. *Let x,y,z,h be elements of a free coid. If x and y divide z, and h is a maximal common divisor of $\{x,y\}$, then xy/h divides z.*

Proof. By hypothesis, x/h divides $z/h=(z/y)(y/h)$. By Theorem 3.2, x/h and y/h are coprime. Thus x/h divides z/y by Theorem 5.1. Hence $(x/h)y$ divides $(z/y)y=z$. $\square$

Theorem 5.3. *Let B be a nonempty subset of a free coid. Then there exists a greatest common divisor of B.*

The greatest common divisor of B is denoted $\gcd B$.

Proof. By Theorem 3.1, there is a maximal common divisor g of B. I claim that every common divisor d of B is a divisor of g. Indeed, let h be a maximal common divisor of $\{g,d\}$. By Theorem 5.2, gd/h is a common divisor of B. But g is maximal, so $g=gd/h$, so $d=h$, so d divides g as claimed. $\square$

Theorem 5.4. *Let z be an element of a free coid. Let $x_1,x_2,\ldots,x_n$ be divisors of z such that x_i is coprime to x_j whenever $i\neq j$. Then $x_1x_2\cdots x_n$ divides z.*

Proof. Induct on n. For $n=0$: 1 divides z. For $n=1$: By hypothesis x_1 divides z. For $n\geq 2$: By hypothesis 1 is a maximal common divisor of $\{x_1,x_2\}$, so x_1x_2 divides z by Theorem 5.2. Define $(y_1,y_2,y_3,\ldots,y_{n-1})=(x_1x_2,x_3,x_4,\ldots,x_n)$. Then $y_1,y_2,\ldots,y_{n-1}$ divide z, and y_i is coprime to y_j whenever $i\neq j$, so $y_1y_2\cdots y_{n-1}$ divides z by induction; i.e., $x_1x_2x_3\cdots x_n$ divides z. $\square$

Theorem 5.5. *Every irreducible in a free coid is a prime.*

Here p is a **prime** if $p\neq 1$ and the non-multiples of p are closed under multiplication: i.e., if p divides ab then p divides a or p divides b.

Proof. Let p be an irreducible in the coid. Let a, b be elements of the coid such that p divides ab. Define $g = \gcd\{p, a\}$. Then $p = (p/g)g$. By definition of irreducibles, $p/g = 1$ or $g = 1$. If $p/g = 1$ then $g = p$ so p divides a as desired. If $g = 1$ then p is coprime to a so p divides b by Theorem 5.1. $\square$

Theorem 5.6. *Every element of a free coid can be written as a product $\prod_{\text{prime}\, p} p^{a_p}$ where each a_p is a nonnegative integer and $\{p : a_p \neq 0\}$ is finite.*

Proof. By Theorem 3.3, every element can be written as a product $q_1 q_2 \cdots q_n$ where each q_i is an irreducible. By Theorem 5.5, each q_i is a prime. Define $a_p = \#\{i : q_i = p\}$; then $q_1 q_2 \cdots q_n = \prod_{\text{prime}\, p} p^{a_p}$, and $\{p : a_p \neq 0\} = \{q_1, q_2, \ldots, q_n\}$. $\square$

6. The ord function

Let p be a prime in a free coid, and let a be an element of the coid. By Theorem 3.4, there is a unique integer $m \geq 0$ such that p^m divides a and p^{m+1} does not. This integer is denoted $\operatorname{ord}_p m$.

Theorem 6.1. *Let a, b, p be elements of a free coid. If p is a prime then* $\operatorname{ord}_p ab = \operatorname{ord}_p a + \operatorname{ord}_p b$.

Proof. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$. Then p^e divides a and p^f divides b so p^{e+f} divides ab. Furthermore, p divides neither a/p^e nor b/p^f; by definition of prime, p does not divide $(a/p^e)(b/p^f)$; so p^{e+f+1} does not divide ab. $\square$

Theorem 6.2. *Let a, b, p be elements of a free coid. If p is a prime then* $\operatorname{ord}_p \gcd\{a, b\} = \min\{\operatorname{ord}_p a, \operatorname{ord}_p b\}$.

Proof. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$. Without loss of generality assume $e \leq f$. Then p^e divides both a and b so p^e divides $\gcd\{a, b\}$. Furthermore, p^{e+1} does not divide a, so p^{e+1} does not divide $\gcd\{a, b\}$. $\square$

Theorem 6.3. *Let a be an element of a free coid. Then $\{\text{prime}\, p : \operatorname{ord}_p a \neq 0\}$ is finite, and $a = \prod_{\text{prime}\, p} p^{\operatorname{ord}_p a}$.*

Consequently, if $\operatorname{ord}_p a = \operatorname{ord}_p b$ for every prime p, then $a = b$.

Proof. By Theorem 5.6, a can be written as a product $\prod_{\text{prime}\, p} p^{a_p}$ where each a_p is a nonnegative integer and $\{p : a_p \neq 0\}$ is finite. If q is a prime then $\operatorname{ord}_q q = 1$ and $\operatorname{ord}_q p = 0$ for all primes $p \neq q$, so $\operatorname{ord}_q a = \sum_{\text{prime}\, p} a_p \operatorname{ord}_q p = a_q$. Hence $\{p : \operatorname{ord}_p a \neq 0\}$ is finite and $a = \prod_{\text{prime}\, p} p^{\operatorname{ord}_p a}$. $\square$

Theorem 6.4. *Let a, b be elements of a free coid. If $\min\{\operatorname{ord}_p a, \operatorname{ord}_p b\} = 0$ for every prime p then a is coprime to b.*

Proof. If p is a prime then $\operatorname{ord}_p \gcd\{a,b\} = 0$ by Theorem 6.2. Hence $\gcd\{a,b\} = 1$ by Theorem 6.3. $\square$

Theorem 6.5. *Let a, b be elements of a free coid. If $\operatorname{ord}_p a \le \operatorname{ord}_p b$ for every prime p then a divides b.*

Proof. If p is a prime then $\operatorname{ord}_p \gcd\{a,b\} = \operatorname{ord}_p a$ by Theorem 6.2. Hence $\gcd\{a,b\} = a$ by Theorem 6.3. $\square$

Theorem 6.6. *Let P be a coprime subset of a free coid. Define $a = \prod_{p\in P} p^{a_p}$ where each a_p is a nonnegative integer and $\{p : a_p \ne 0\}$ is finite. Define $b = \prod_{p\in P} p^{b_p}$ where each b_p is a nonnegative integer and $\{p : b_p \ne 0\}$ is finite. Then $\gcd\{a,b\} = \prod_{p\in P} p^{\min\{a_p,b_p\}}$.*

Proof. I will show for every prime q that $\operatorname{ord}_q \prod_p p^{\min\{a_p,b_p\}} = \operatorname{ord}_q \gcd\{a,b\}$. Hence $\prod_{p\in P} p^{\min\{a_p,b_p\}} = \gcd\{a,b\}$ by Theorem 6.3.

If q does not divide any $r \in P$ then $\operatorname{ord}_q a = 0$, $\operatorname{ord}_q b = 0$, and $\operatorname{ord}_q \prod_p p^{\min\{a_p,b_p\}} = 0 = \operatorname{ord}_q \gcd\{a,b\}$. Assume from now on that q divides some $r \in P$.

If $r' \in P - \{r\}$ then $\gcd\{r,r'\} = 1$ so $\min\{\operatorname{ord}_q r, \operatorname{ord}_q r'\} = 0$ by Theorem 6.2 so $\operatorname{ord}_q r' = 0$. Thus $\operatorname{ord}_q a = a_r \operatorname{ord}_q r$, $\operatorname{ord}_q b = b_r \operatorname{ord}_q r$, and $\operatorname{ord}_q \prod_p p^{\min\{a_p,b_p\}} = \min\{a_r, b_r\} \operatorname{ord}_q r = \min\left\{\operatorname{ord}_q a, \operatorname{ord}_q b\right\} = \operatorname{ord}_q \gcd\{a,b\}$. $\square$

Theorem 6.7. *Let P be a coprime subset of a free coid. Define $a = \prod_{p\in P} p^{a_p}$ where each a_p is a nonnegative integer and $\{p : a_p \ne 0\}$ is finite. Define $b = \prod_{p\in P} p^{b_p}$ where each b_p is a nonnegative integer and $\{p : b_p \ne 0\}$ is finite. If a divides b then $a_p \le b_p$ for every $p \in P - \{1\}$ and $b/a = \prod_{p\in P-\{1\}} p^{b_p - a_p}$.*

Proof. Select $r \in P - \{1\}$. Select a prime q dividing r. Then $\operatorname{ord}_q r > 0$, so $\operatorname{ord}_q p = 0$ for all $p \in P - \{r\}$. Thus $\operatorname{ord}_q a = a_r \operatorname{ord}_q r$ and $\operatorname{ord}_q b = b_r \operatorname{ord}_q r$; but $\operatorname{ord}_q a \le \operatorname{ord}_q b$, so $a_r \le b_r$. Also $a \prod_{p\in P-\{1\}} p^{b_p - a_p} = \prod_{p\in P-\{1\}} p^{a_p + b_p - a_p} = b$. $\square$

7. The natural coprime base

Fix a subset S of a free coid. The **closure of** S is the smallest subset T of the free coid such that

- $S \cup \{1\} \subseteq T$;
- $ab \in T$ if $a, b \in T$;
- $a \in T$ if $ab, b \in T$; and
- $\gcd\{a,b\} \in T$ if $a, b \in T$.

The **natural coprime base for** S, denoted $\operatorname{cb} S$, is the set of minimal elements of $T - \{1\}$. This is, in fact, a coprime base for S; see Theorem 7.1. It is the only coprime base for S inside $T - \{1\}$; see Theorem 7.3.

There are other ways to characterize $\operatorname{cb} S$. In drafts of this paper I defined $\operatorname{cb} S$ as the set of maximal quasiprimes for S; here p is a **quasiprime for** S if every element of S can be written in the form up^m with p coprime to u. A referee suggested defining $\operatorname{cb} S$ as a minimal subset of T that is a base for T. Bach, Driscoll, and Shallit proved in [2, Theorem 3] that there is a maximum coprime base for S in a particular ordering of coprime bases for S, and that this maximum coprime base is a subset of $T - \{1\}$. Another characterization appears without proof in [20, Lemma 3.2]. An incorrect characterization appears without proof in [16, Remark 6.8].

Theorem 7.1. *Let S be a subset of a free coid. Then $\operatorname{cb} S$ is a coprime base for S.*

Proof. Define T as the closure of S. Define $P = \operatorname{cb} S$.

P is coprime: Say $a \in P$ is not coprime to $b \in P$. Define $g = \gcd\{a, b\}$. Then $g \in T - \{1\}$, and g divides a; but a is a minimal element of $T - \{1\}$ by definition of P, so $g = a$. Similarly $g = b$. Thus $a = b$.

P is a base for T (hence for S): Suppose not. Find a minimal element z of T outside the coid generated by P. Then $z \notin P$. Thus z is not a minimal element of $T - \{1\}$, by definition of P; but $z \in T - \{1\}$; so z has a divisor $y \in T - \{1\}$ with $y \neq z$. Also $z/y \in T - \{1\}$ and $z/y \neq z$. The minimality of z implies that both y and z/y are in the coid generated by P; hence z is in the coid generated by P. Contradiction. $\square$

Theorem 7.2. *Let S and Q be subsets of a free coid. If Q is a coprime base for S then Q is a base for the closure of S.*

Proof. Define T' as the intersection of T with the coid generated by Q. Then $1 \in T'$; $ab \in T'$ if $a, b \in T'$; $S \subseteq T'$ since Q is a base for S; $a \in T'$ if $ab, b \in T'$, by Theorem 6.7; and $\gcd\{a, b\} \in T'$ if $a, b \in T'$, by Theorem 6.6. Thus $T \subseteq T'$; i.e., Q is a base for T. $\square$

Theorem 7.3. *Let S be a subset of a free coid. Let P be a coprime base for S. Let T be the closure of S. If $P \subseteq T - \{1\}$ then $P = \operatorname{cb} S$.*

Proof. P is a base for T by Theorem 7.2, so P is a base for $\operatorname{cb} S$, so each element of $\operatorname{cb} S$ is divisible by an element of P. Furthermore, $\operatorname{cb} S$ is a base for P, so each element of P is divisible by an element of $\operatorname{cb} S$.

Starting from any $p \in P$, find $q \in \operatorname{cb} S$ such that q divides p, and then find $p' \in P$ such that p' divides q. Then p' divides p, so by coprimality $p' = p$, so $q = p$, so $p \in \operatorname{cb} S$. Hence $P \subseteq \operatorname{cb} S$. Similarly $\operatorname{cb} S \subseteq P$. $\square$

Theorem 7.4. *Let S be a finite subset of a free coid. Then $\operatorname{cb} S$ is a finite coprime base for S.*

Proof. The proof of Theorem 4.1 recursively constructs a finite coprime base Q for S using exact division and gcd. Thus $Q \subseteq T$, where T is the closure of S. Define $P = Q - \{1\}$. Then P is a finite coprime base for S, and $P \subseteq T - \{1\}$, so $P = \operatorname{cb} S$ by Theorem 7.3, so $\operatorname{cb} S$ is a finite coprime base for S. $\square$

Theorem 7.5. *Let S be a finite subset of a free coid H. Let z be an element of H. If every element of S divides z then $\prod_{p \in \operatorname{cb} S} p$ divides z.*

Proof. Take $p \in \operatorname{cb} S$. If p does not divide any element of S then p is coprime to all elements of S, so p is coprime to all elements of the closure of S, so p is coprime to p, contradiction. Thus p divides z. By Theorem 5.4, $\prod_{p \in \operatorname{cb} S} p$ divides z. $\square$

PART II. TWO-ELEMENT SETS

8. Logarithms and M-time

The algorithms in this paper work for any free coid H. They are given elements of H (represented in some way as strings) and oracles that perform the following operations:

- Multiplication: compute $ab \in H$ given $a, b \in H$.
- Exact division: compute $a \in H$ given $ab, b \in H$.
- Greatest common divisor: compute $\gcd\{a, b\} \in H$ given $a, b \in H$.
- Equality testing: compute $[a = b] \in \{0, 1\}$ given $a, b \in H$.

The algorithms combine these four operations to perform more complicated operations. For example, Algorithm 10.1 computes $a^4 \in H$, given $a \in H$, by first feeding a, a to the multiplication oracle to obtain a^2, and then feeding a^2, a^2 to the multiplication oracle to obtain a^4. As another example, to check whether a is a divisor of b, Algorithm 19.2 checks whether $\gcd\{a, b\}$ equals a.

Definition of M-time. I count the number of multiplications, divisions, and gcds in each algorithm, with a weight of $(1 + \lg ab)\mu(\lg ab)$ for each multiplication $a, b \mapsto ab$, each exact division $ab, b \mapsto a$, and each greatest common divisor $a, b \mapsto \gcd\{a, b\}$. The total is called **$M$-time**. Here $\mu : \mathbf{R} \to \mathbf{R}$ is any nondecreasing positive function, and $\lg : H \to \mathbf{R}$ is any function satisfying (1) $\lg ab = \lg a + \lg b$ and (2) $\lg a \geq 1$ for every $a \neq 1$. Note that $\lg 1 = 0$.

Each algorithm is accompanied by a theorem stating an upper bound, parametrized by lg and μ, on the M-time used by that algorithm. In particular, if $\mu(x) \in x^{o(1)}$, then the M-time to compute $\operatorname{cb} S$ is essentially linear in $\lg \operatorname{prod} S$ by Theorem 18.2, and the M-time to factor S over any coprime base P for S is essentially linear in $\lg \operatorname{prod} S + \lg \operatorname{prod} P$ by Theorem 21.3.

Why M-time is useful. These bounds on M-time imply bounds on algorithm time for various coids H that arise in practice.

In particular, if H is the set of positive integers (represented in the usual way as base-2 strings), then there are algorithms (for, e.g., multitape Turing machines) that perform multiplication, exact division, and gcd in time at most $(1 + \lg ab)\mu(\lg ab)$. Here $\lg : H \to \mathbf{R}$ is the usual logarithm base 2, and μ is a nondecreasing positive function with $\mu(x) \in x^{o(1)}$. The time spent by my algorithms inside these subroutines for multiplication, exact division, and gcd is bounded by M-time for these functions $\lg, \mu$; the reader can check that my algorithms spend negligible time in other operations; the M-time for factoring into coprimes is essentially linear in the input size. Conclusion: factoring positive integers into coprimes takes time essentially linear in the input size.

Similarly, if H is the set of monic polynomials in one variable over a finite field, then there are algorithms that perform multiplication, exact division, and gcd in time at most $(1+\lg ab)\mu(\lg ab)$. Here $\lg : H \to \mathbf{R}$ is the degree map, and μ is a nondecreasing positive function with $\mu(x) \in x^{o(1)}$. Conclusion: factoring monic polynomials into coprimes takes time essentially linear in the input size.

See my paper [7] for a survey of the standard essentially-linear-time algorithms for multiplication, exact division, and gcd.

Integers and polynomials support many other useful arithmetic operations: division with remainder, for example, and size inspection (approximation of lg). Algorithms that save time by using these extra operations are beyond the scope of this paper, as explained in Section 1. My M-time bounds are expressed in much more detail than "essentially linear time" solely because those details simplify the proofs; the level of detail is not meant to suggest that these are near-optimal bounds for near-optimal algorithms.

Termination. Another consequence of the M-time bounds is that each of these algorithms terminates for any free coid H. Proof: Define $\mu(x) = 1$. Define $\lg : H \to \mathbf{R}$ by setting $\lg p = 1$ for each prime p in H. The M-time bounds are then upper bounds on the number of multiplications, divisions, and gcds in the algorithm. Every algorithm loop involves at least one multiplication, division, or gcd. Hence the algorithm terminates.

Another way to prove termination is to observe that the relevant subcoid of H (the coid generated by a finite coprime base for the inputs to the algorithm) is isomorphic to a subcoid of the set of positive integers. The algorithm terminates for positive integers, so it terminates for H.

9. From CBA to DCBA

The natural coprime base for $\left\{p^e, p^f\right\}$ is $\{p^g\} - \{1\}$ where $g = \gcd\{e, f\}$. Thus *natural coprime bases act on exponents as greatest common divisors*.

This section compares two algorithms to compute $\operatorname{cb}\{a,b\}$. One algorithm, which I call CBA (the "coprime base algorithm"), is the quadratic-time algorithm introduced by Bach, Driscoll, and Shallit in [2]. The other algorithm, which I call DCBA, is the essentially-linear-time algorithm introduced in this paper.

CBA replaces (a,b) with $(a/\gcd\{a,b\}, \gcd\{a,b\}, b/\gcd\{a,b\})$; it focuses on the left pair in this vector and then focuses on the right pair. In particular, it replaces (p^e, p^f) with $(p^{e-f}, p^f, 1)$ if $e > f$ or $(1, p^e, p^{f-e})$ if $e \le f$. The exponent pair (e, f) has been replaced with $(e - f, f)$ or $(e, f - e)$. Thus *CBA uses Euclid's subtractive algorithm to compute greatest common divisors of exponents.*

Euclid's subtractive algorithm is a dangerous way to compute greatest common divisors: the number of steps is the sum of the quotients in the continued fraction for e/f. A much safer alternative is Euclid's repeated-division algorithm, where the number of steps is at worst logarithmic in $e + f$. See generally [26, Sections 4.5.2 and 4.5.3].

Writing out the standard base-2 integer-division algorithm inside Euclid's repeated-division algorithm produces Brent's left-shift binary gcd algorithm, which replaces (e, f) with $(e - 2^k f, f)$ for the largest possible value of k, or with (f, e) if $e < f$. It will turn out that *DCBA uses Brent's left-shift algorithm to compute greatest common divisors of exponents.*

DCBA actually uses a slightly different exponent transformation, namely the function τ defined below, to simplify the computations.

The exponent transformation. Define a function τ on pairs of nonnegative integers as follows:

$$\tau(e,f) = \begin{cases} (f-e,e) & \text{if } e \le f \\ (e-f,f) & \text{if } f < e \le 2f \\ (e-2f,f) & \text{if } 2f < e \le 4f \\ (e-4f,f) & \text{if } 4f < e \le 8f \\ \vdots & \\ (e,0) & \text{if } 0 = f < e. \end{cases}$$

Write τ^n for the nth iterate of τ.

Define $\lambda(a,b)$ as the smallest $n \ge 0$ such that the second component of $\tau^n(\operatorname{ord}_p a, \operatorname{ord}_p b)$ is 0 for every prime p. Existence of $\lambda(a,b)$ follows from Theorem 9.3.

Theorem 9.1. *Let e,f,n be nonnegative integers. If $f=0$ or $e+\sqrt{2}f < \sqrt{2}^{n+1}$ then $\tau^n(e,f)$ has second component* 0*.*

Proof. Induct on n. For $n=0$: If $e+\sqrt{2}f < \sqrt{2}$ then $f<1$, so in any case $f=0$, so $\tau^n(e,f)=(e,0)$ as desired. For $n \ge 1$: Define $(e',f')=\tau(e,f)$. I will show that $f'=0$ or $e'+\sqrt{2}f' < \sqrt{2}^n$; hence $\tau^n(e,f)=\tau^{n-1}(e',f')$ has second component 0 by induction.

Case 1: $f=0$ and $e=0$. Then $(e',f')=(f-e,e)=(0,0)$ by definition of τ.

Case 2: $f=0$ and $e>0$. Then $(e',f')=(e,0)$ by definition of τ.

Case 3: $f>0$ and $e \le f$. Then $(e',f')=(f-e,e)$ by definition of τ, so $\sqrt{2}e'+2f'-e-\sqrt{2}f=(1-\sqrt{2})e \le 0$, so $e'+\sqrt{2}f' \le (e+\sqrt{2}f)/\sqrt{2} < \sqrt{2}^n$.

Case 4: $f>0$ and $e>f$. Then $2^k f < e \le 2^{k+1} f$ for some integer $k \ge 0$. Now $(e',f')=(e-2^k f, f)$ by definition of τ, so

$$\begin{aligned}\sqrt{2}e'+2f'-e-\sqrt{2}f &= (\sqrt{2}-1)e-(2^k\sqrt{2}+\sqrt{2}-2)f \\ &\le 2^{k+1}(\sqrt{2}-1)f-(2^k\sqrt{2}+\sqrt{2}-2)f \\ &= (1-2^k)(2-\sqrt{2})f \le 0,\end{aligned}$$

so again $e'+\sqrt{2}f' < \sqrt{2}^n$. $\square$

Theorem 9.2. *Let a be an element of a free coid. Then $\operatorname{ord}_p a \le \lg a$ for every prime p.*

Proof. Write $n=\operatorname{ord}_p a$. Then $a=up^n$ for some u, and $\lg p \ge 1$ since $p \ne 1$, so $\lg a = \lg u + n \lg p \ge n \lg p \ge n$. $\square$

Theorem 9.3. *Let a,b be elements of a free coid. Let n be a nonnegative integer. If $\lg a + \sqrt{2}\lg b < \sqrt{2}^{n+1}$ then $\lambda(a,b) \le n$.*

Proof. Define $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$ where p is a prime. By Theorem 9.2, $e + \sqrt{2} f \le \lg a + \sqrt{2} \lg b < \sqrt{2}^{n+1}$. By Theorem 9.1, $\tau^n(e, f)$ has second component 0. □

Theorem 9.4. *Let a, b be elements of a free coid. If $\lambda(a, b) = 0$ then $b = 1$.*

Proof. If p is a prime then $\operatorname{ord}_p b = 0$ by definition of λ. Hence $b = 1$. □

10. Computing powers

Algorithm 10.1. Given (a, n), with n a nonnegative integer, to print a^{2^n}:

1. If $n = 0$: Print a and stop.
2. Set $a \leftarrow a^2$. Set $n \leftarrow n - 1$. Return to Step 1.

Theorem 10.2. *Algorithm 10.1 uses M-time at most $(n + 2(2^n - 1) \lg a)\mu(2^n \lg a)$.*

Proof. For $n = 0$, Algorithm 10.1 uses no M-time, and $n + 2(2^n - 1) \lg a = 0$.

For $n \ge 1$, Algorithm 10.1 first computes a^2, using M-time at most $(1 + 2 \lg a)\mu(2 \lg a) \le (1 + 2 \lg a)\mu(2^n \lg a)$. It then computes $(a^2)^{2^{n-1}}$, using M-time at most

$$(n - 1 + 2(2^{n-1} - 1) \lg a^2)\mu(2^{n-1} \lg a^2) = (n - 1 + 2(2^n - 2) \lg a)\mu(2^n \lg a)$$

by induction. Finally $1 + 2 \lg a + n - 1 + 2(2^n - 2) \lg a = n + 2(2^n - 1) \lg a$. □

11. The ppi, ppo, ppg, and pple functions

The defining properties of $\operatorname{ppi}(a, b)$, $\operatorname{ppo}(a, b)$, $\operatorname{ppg}(a, b)$, and $\operatorname{pple}(a, b)$ are that

$$\begin{aligned}
\operatorname{ord}_p \operatorname{ppi}(a, b) &= (\operatorname{ord}_p a)[\operatorname{ord}_p b > 0] \\
\operatorname{ord}_p \operatorname{ppo}(a, b) &= (\operatorname{ord}_p a)[\operatorname{ord}_p b = 0] \\
\operatorname{ord}_p \operatorname{ppg}(a, b) &= (\operatorname{ord}_p a)[\operatorname{ord}_p a > \operatorname{ord}_p b] \\
\operatorname{ord}_p \operatorname{pple}(a, b) &= (\operatorname{ord}_p a)[\operatorname{ord}_p a \le \operatorname{ord}_p b]
\end{aligned}$$

for all primes p. Existence of $\operatorname{ppi}, \operatorname{ppo}, \operatorname{ppg}, \operatorname{pple}$ is proven constructively in Theorem 11.1 and Theorem 11.2. Uniqueness follows from Theorem 6.3.

The notation $\operatorname{ppi}(a, b)$ stands for "powers in a of primes inside b"; $\operatorname{ppo}(a, b)$ is "powers in a of primes outside b"; $\operatorname{ppg}(a, b)$ is "prime powers in a greater than those in b"; and $\operatorname{pple}(a, b)$ is "prime powers in a less than or equal to those in b." I originally used the notation $\gcd\{a, b^\infty\}$ for $\operatorname{ppi}(a, b)$, but the shorter name ppi will be helpful later.

The ppg function is the subject of [30, Chapter 1, Section 19]; Stieltjes's algorithm in [30] takes quadratic time on inputs $(2^{n+1}, 2^n)$. The ppo function is the subject of [27]; Lüneburg's algorithm in [27] takes quadratic time on inputs $(2^n, 2)$.

Theorem 11.1. *Let a,c be elements of a free coid. Define $x_0 = \gcd\{a,c\}$ and $y_0 = a/x_0$. For $n \ge 0$ define $x_{n+1} = x_n \gcd\{x_n,y_n\}$ and $y_{n+1} = y_n/\gcd\{x_n,y_n\}$. (1) If $n \ge 0$ and $2^n \ge \lg a$ then $\gcd\{x_n,y_n\} = 1$. (2) If $\gcd\{x_n,y_n\} = 1$ then $x_n = \operatorname{ppi}(a,c)$ and $y_n = \operatorname{ppo}(a,c)$.*

Proof. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p c$ where p is a prime. Note that $x_n y_n = a$ by induction on n, so $\operatorname{ord}_p x_n + \operatorname{ord}_p y_n = e$.

The point is that $\operatorname{ord}_p x_n = \min\{e, 2^n f\}$ for all $n \ge 0$. Proof: $\operatorname{ord}_p x_0 = \min\{e, f\}$. For $n \ge 1$, assume inductively that $\operatorname{ord}_p x_{n-1} = \min\left\{e, 2^{n-1} f\right\}$. If $e < 2^{n-1} f$ then $\operatorname{ord}_p x_{n-1} = e$ so $\operatorname{ord}_p y_{n-1} = 0$ so $\operatorname{ord}_p \gcd\{x_{n-1}, y_{n-1}\} = \min\{e, 0\} = 0$ so $\operatorname{ord}_p x_n = e = \min\{e, 2^n f\}$. If $e \ge 2^{n-1} f$ then $\operatorname{ord}_p x_{n-1} = 2^{n-1} f$ so $\operatorname{ord}_p y_{n-1} = e - 2^{n-1} f$ so $\operatorname{ord}_p \gcd\{x_{n-1}, y_{n-1}\} = \min\left\{2^{n-1} f, e - 2^{n-1} f\right\}$ so $\operatorname{ord}_p x_n = 2^{n-1} f + \min\left\{e - 2^{n-1} f, 2^{n-1} f\right\} = \min\{e, 2^n f\}$.

(1) If $f = 0$ then $\operatorname{ord}_p x_n = 0 = \operatorname{ord}_p x_{n+1}$. If $f > 0$ then $2^n f \ge 2^n \ge \lg a \ge e$ by Theorem 9.2 so $\operatorname{ord}_p x_n = e = \operatorname{ord}_p x_{n+1}$. In both cases, $\operatorname{ord}_p \gcd\{x_n, y_n\} = \operatorname{ord}_p x_{n+1} - \operatorname{ord}_p x_n = 0$.

(2) By hypothesis $x_n = x_{n+1}$, so $\min\{e, 2^n f\} = \min\left\{e, 2^{n+1} f\right\}$. If $f = 0$ then $\operatorname{ord}_p x_n = \min\{e, 0\} = 0$. If $f > 0$ then $2^n f < 2^{n+1} f$ so $e \le 2^n f$; thus $\operatorname{ord}_p x_n = e$. In both cases, $\operatorname{ord}_p x_n = e[f > 0]$ and $\operatorname{ord}_p y_n = e - \operatorname{ord}_p x_n = e[f = 0]$. $\square$

Theorem 11.2. *Let a,b be elements of a free coid. Define $y_0 = \gcd\{a,b\}$ and $x_0 = a/y_0$. For $n \ge 0$ define $x_{n+1} = x_n \gcd\{x_n,y_n\}$ and $y_{n+1} = y_n/\gcd\{x_n,y_n\}$. (1) If $n \ge 0$ and $2^n \ge \lg a$ then $\gcd\{x_n,y_n\} = 1$. (2) If $\gcd\{x_n,y_n\} = 1$ then $x_n = \operatorname{ppg}(a,b)$ and $y_n = \operatorname{pple}(a,b)$.*

Proof. Define $c = x_0$. Then $x_0 = \gcd\{a,c\}$ and $y_0 = a/x_0$.

(1) If $n \ge 0$ and $2^n \ge \lg a$ then $\gcd\{x_n,y_n\} = 1$ by Theorem 11.1.

(2) If $\gcd\{x_n,y_n\} = 1$ then $x_n = \operatorname{ppi}(a,c)$ and $y_n = \operatorname{ppo}(a,c)$ by Theorem 11.1. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$ where p is a prime. Then $\operatorname{ord}_p y_0 = \min\{e, f\}$, so $\operatorname{ord}_p c$ is 0 exactly when $e \le f$. Thus $\operatorname{ord}_p x_n = e[\operatorname{ord}_p c > 0] = e[e > f]$ and $\operatorname{ord}_p y_n = e[\operatorname{ord}_p c = 0] = e[e \le f]$. $\square$

The following two algorithms include n solely for expository purposes.

Algorithm 11.3. Given (a,c), to print $\gcd\{a,c\}$, $\operatorname{ppi}(a,c)$, and $\operatorname{ppo}(a,c)$:

1. Set $x \leftarrow \gcd\{a,c\}$. Print x. Set $y \leftarrow a/x$. Set $n \leftarrow 0$.
2. (Now $(x,y) = (x_n,y_n)$ in Theorem 11.1.) Set $g \leftarrow \gcd\{x,y\}$. If $g = 1$: print x, print y, and stop.
3. Set $x \leftarrow xg$ and $y \leftarrow y/g$. Set $n \leftarrow n+1$. Return to Step 2.

Algorithm 11.4. Given (a,b), to print $\gcd\{a,b\}$, $\operatorname{ppg}(a,b)$, and $\operatorname{pple}(a,b)$:

1. Set $y \leftarrow \gcd\{a,b\}$. Print y. Set $x \leftarrow a/y$. Set $n \leftarrow 0$.
2. (Now $(x,y) = (x_n,y_n)$ in Theorem 11.2.) Set $g \leftarrow \gcd\{x,y\}$. If $g = 1$: print x, print y, and stop.
3. Set $x \leftarrow xg$ and $y \leftarrow y/g$. Set $n \leftarrow n+1$. Return to Step 2.

Theorem 11.5. *Algorithm 11.3 computes $\gcd\{a,c\}, \operatorname{ppi}(a,c), \operatorname{ppo}(a,c)$ in M-time at most $(3k + 3 + (2k+4)\lg a + \lg c)\mu(\lg ac)$ if $k \ge 0$ and $2^k \ge \lg a$.*

Proof. By Theorem 11.1, $\gcd\{x_k, y_k\} = 1$, so the algorithm stops when $n = k$ if not earlier. It thus performs Step 2 for $n \in \{0, 1, \ldots, k\}$ at most, and Step 3 for $n \in \{0, 1, \ldots, k-1\}$ at most.

Step 1 uses M-time at most $(1 + \lg ac)\mu(\lg ac)$ to compute $\gcd\{a, c\}$ and M-time at most $(1 + \lg a)\mu(\lg a)$ to compute a/x.

Each iteration of Step 2 uses M-time at most $(1 + \lg a)\mu(\lg a)$ since $xy = x_n y_n = a$. The total is at most $(k+1)(1 + \lg a)\mu(\lg a)$.

Each iteration of Step 3 uses M-time at most $(1 + \lg xg)\mu(\lg xg) + (1 + \lg y)\mu(\lg y) \le (2 + \lg x_{n+1} + \lg y_n)\mu(\lg a)$. The total for $n \in \{0, 1, 2, \ldots, k-1\}$ telescopes to

$$(2k + \lg x_k + (k-1)\lg a + \lg y_0)\mu(\lg a) \le (2k + (k+1)\lg a)\mu(\lg a).$$

Add: $(1 + \lg ac) + (1 + \lg a) + (k+1)(1 + \lg a) + (2k + (k+1)\lg a) = 3k + 3 + \lg c + (2k+4)\lg a$. □

Theorem 11.6. *Algorithm 11.4 computes* $\gcd\{a, b\}, \operatorname{ppg}(a, b), \operatorname{pple}(a, b)$ *in M-time at most* $(3k + 3 + (2k+4)\lg a + \lg b)\mu(\lg ab)$ *if $k \ge 0$ and $2^k \ge \lg a$.*

Proof. Same analysis as in Theorem 11.5. □

12. The ls function

Fix a, b in a free coid. Define $\operatorname{ls}_n(a, b)$ as (g_n, h_n, c_n), where

$$\begin{aligned}(g_0, h_0, c_0) &= (\gcd, \operatorname{ppg}, \operatorname{pple})(\operatorname{ppi}(a, b), b),\\ (g_{n+1}, h_{n+1}, c_{n+1}) &= (\gcd, \operatorname{ppg}, \operatorname{pple})(h_n, g_n^2).\end{aligned}$$

The name ls stands for “left shift.” This function separates the primes in a and b according to the cases in the definition of τ in Section 9.

Theorem 12.1. *Let a, b be elements of a free coid. Define $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$ where p is a prime. Define $(g_n, h_n, c_n) = \operatorname{ls}_n(a, b)$. Then*

$$(\operatorname{ord}_p g_n, \operatorname{ord}_p h_n, \operatorname{ord}_p c_n) = [0 < 2^{n-1} f < e]\big(\min\{e, 2^n f\}, e[e > 2^n f], e[e \le 2^n f]\big)$$

for $n \ge 1$.

Proof. Notice that $\operatorname{ord}_p g_{n+1}$, $\operatorname{ord}_p h_{n+1}$, and $\operatorname{ord}_p c_{n+1}$ are all bounded by $\operatorname{ord}_p h_n$; so if $\operatorname{ord}_p h_m = 0$ then $\operatorname{ord}_p g_n = \operatorname{ord}_p h_n = \operatorname{ord}_p c_n = 0$ for all $n > m$.

Case 1: $f = 0$. Then $\operatorname{ord}_p \operatorname{ppi}(a, b) = 0$ so $\operatorname{ord}_p g_n = \operatorname{ord}_p h_n = \operatorname{ord}_p c_n = 0$ for all n.

Case 2: $e \le f$. Then $\operatorname{ord}_p \operatorname{ppi}(a, b) = e$, so $\operatorname{ord}_p h_0 = e[e > f] = 0$, so $\operatorname{ord}_p g_n = \operatorname{ord}_p h_n = \operatorname{ord}_p c_n = 0$ for all $n \ge 1$.

Case 3: $e > f > 0$. Then $\operatorname{ord}_p \operatorname{ppi}(a, b) = e$, so $\operatorname{ord}_p h_0 = e$ and $\operatorname{ord}_p g_0 = f$. Thus $\operatorname{ord}_p g_1 = \min\{e, 2f\}$; $\operatorname{ord}_p h_1 = e[e > 2f]$; and $\operatorname{ord}_p c_1 = e[e \le 2f]$.

Assume inductively that $\operatorname{ord}_p g_n = [2^{n-1} f < e] \min\{e, 2^n f\}$ and $\operatorname{ord}_p h_n = e[e > 2^n f]$. If $e \le 2^n f$ then $\operatorname{ord}_p h_n = 0$ so $\operatorname{ord}_p g_{n+1} = \operatorname{ord}_p h_{n+1} = \operatorname{ord}_p c_{n+1} = 0$ as desired. If $e > 2^n f$ then $(\operatorname{ord}_p g_n, \operatorname{ord}_p h_n) = (2^n f, e)$ so $\operatorname{ord}_p g_{n+1} = \min\{e, 2^{n+1} f\}$ as desired; $\operatorname{ord}_p h_{n+1} = e[e > 2^{n+1} f]$ as desired; and $\operatorname{ord}_p c_{n+1} = e[e \le 2^{n+1} f]$ as desired. □

Theorem 12.2. *Let a, b be elements of a free coid. Define $(g_n, h_n, c_n) = \operatorname{ls}_n(a, b)$. If $m \ge 1$ and $h_m = 1$ then $a = c_0 c_1 \cdots c_m \operatorname{ppo}(a, b)$. Furthermore, any two members of the sequence $c_0, c_1, \ldots, c_m, \operatorname{ppo}(a, b)$ are coprime.*

Proof. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$ where p is a prime. I will show that the sum of the numbers $\operatorname{ord}_p c_0, \operatorname{ord}_p c_1, \ldots, \operatorname{ord}_p c_m, \operatorname{ord}_p \operatorname{ppo}(a, b)$ is e, and that at most one of the numbers is nonzero.

Case 1: $f = 0$. Then $\operatorname{ord}_p \operatorname{ppi}(a, b) = 0$ so $\operatorname{ord}_p c_0 = 0$; $\operatorname{ord}_p c_n = 0$ for $n \ge 1$; and $\operatorname{ord}_p \operatorname{ppo}(a, b) = e$.

Case 2: $e \le f$. Then $\operatorname{ord}_p \operatorname{ppo}(a, b) = 0$; $\operatorname{ord}_p c_0 = e[e \le f] = e$; and $\operatorname{ord}_p c_n = 0$ for $n \ge 1$.

Case 3: $e > f > 0$. Then $\operatorname{ord}_p c_0 = 0$; $\operatorname{ord}_p c_n = e[2^{n-1} f < e \le 2^n f]$ for $n \ge 1$; and $\operatorname{ord}_p \operatorname{ppo}(a, b) = 0$. There is exactly one value of $k \ge 1$ for which $2^{k-1} f < e \le 2^k f$; and $\operatorname{ord}_p h_m = 0$ so $e \le 2^m f$ so $k \le m$. □

Theorem 12.3. *Let a, b be elements of a free coid. Define $(g_n, h_n, c_n) = \operatorname{ls}_n(a, b)$. Define $d_n = \gcd\{c_n, b\}$ for $n \ge 1$. Then $d_n^{2^{n-1}}$ divides c_n; $\lambda(c_n / d_n^{2^{n-1}}, d_n) \le \lambda(a, b) - 1$ if $b \ne 1$; and $c_0 d_1 d_2 \cdots d_n$ divides b. Furthermore, if $m \ge 1$ and $h_m = 1$, then $b / d_1 d_2 \cdots d_m$ is coprime to a / c_0, and $\lambda(b / c_0 d_1 d_2 \cdots d_m, c_0) \le \lambda(a, b) - 1$ if $b \ne 1$.*

Proof. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$ where p is a prime.

By Theorem 12.1, $\operatorname{ord}_p c_n = e[2^{n-1} f < e \le 2^n f]$, so $\operatorname{ord}_p d_n = f[2^{n-1} f < e \le 2^n f]$, so $2^{n-1} \operatorname{ord}_p d_n \le e[2^{n-1} f < e \le 2^n f] = \operatorname{ord}_p c_n$. Thus $d_n^{2^{n-1}}$ divides c_n.

Next, $\operatorname{ord}_p c_0 d_1 d_2 \cdots d_n = e[e \le f] + f[f < e \le 2^n f] \le f[e \le f] + f[f < e \le 2^n f] = f[e \le 2^n f] \le f$. Thus $c_0 d_1 d_2 \cdots d_n$ divides b.

If $h_m = 1$ then $f = 0$ or $e \le 2^m f$, so $\operatorname{ord}_p d_1 d_2 \cdots d_m = f[f < e]$, so $\operatorname{ord}_p(b / d_1 d_2 \cdots d_m) = f[e \le f]$; but $\operatorname{ord}_p(a / c_0) = e[e > f]$. Thus $b / d_1 d_2 \cdots d_m$ and a / c_0 are coprime.

Assume from now on that $b \ne 1$. Write $k = \lambda(a, b)$. Then $\tau^k(e, f) = (\ldots, 0)$ by definition of λ, and $k \ge 1$ by Theorem 9.4.

If $2^{n-1} f < e \le 2^n f$ then $(\operatorname{ord}_p(c_n / d_n^{2^{n-1}}), \operatorname{ord}_p d_n) = (e - 2^{n-1} f, f) = \tau(e, f)$ so

$$\tau^{k-1}(\operatorname{ord}_p(c_n / d_n^{2^{n-1}}), \operatorname{ord}_p d_n) = \tau^k(e, f) = (\ldots, 0).$$

Otherwise $(\operatorname{ord}_p(c_n / d_n^{2^{n-1}}), \operatorname{ord}_p d_n) = (0, 0)$ so $\tau^{k-1}(\operatorname{ord}_p(c_n / d_n^{2^{n-1}}), \operatorname{ord}_p d_n) = (0, 0)$.

Finally, if $h_m = 1$ and $e \le f$ then $(\operatorname{ord}_p(b / c_0 d_1 d_2 \cdots d_m), \operatorname{ord}_p c_0) = (f - e, e) = \tau(e, f)$; if $h_m = 1$ and $e > f$ then $(\operatorname{ord}_p(b / c_0 d_1 d_2 \cdots d_m), \operatorname{ord}_p c_0) = (0, 0)$; in either case, as above, $\tau^{k-1}(\operatorname{ord}_p(b / c_0 d_1 d_2 \cdots d_m), \operatorname{ord}_p c_0) = (\ldots, 0)$. □

Theorem 12.4. *Let a, b be elements of a free coid. Define $(g_n, h_n, c_n) = \operatorname{ls}_n(a, b)$. If $m \ge 1$ and $2^{m-1} \ge \lg a$ then $h_m = 1$.*

Proof. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$ where p is a prime. Then $e \le \lg a \le 2^{m-1}$ by Theorem 9.2. If $f = 0$ then $\operatorname{ord}_p h_m = 0$; if $f \ge 1$ then $e \le 2^{m-1} f$ so $\operatorname{ord}_p h_m = 0$. □

13. Computing a coprime base for a two-element set

This section introduces a fast algorithm to compute $\operatorname{cb}\{a,b\}$.

Theorem 13.1. *Let a,b be elements of a free coid. Define $(g_n,h_n,c_n) = \operatorname{ls}_n(a,b)$. Define $d_n = \gcd\{c_n,b\}$ for $n \ge 1$. Assume that $h_m = 1$ with $m \ge 1$. Define $P_n = \operatorname{cb}\{c_n/d_n^{2^{n-1}}, d_n\}$ for $1 \le n \le m$. Define $Q = \operatorname{cb}\{b/c_0 d_1 d_2 \cdots d_m, c_0\}$. Define $R = \operatorname{cb}\{\operatorname{ppo}(a,b)\}$. Then $P = \bigcup P_n$ is a disjoint union; $P \cup Q \cup R$ is a disjoint union; and $P \cup Q \cup R = \operatorname{cb}\{a,b\}$.*

Proof. By construction d_n divides c_n, so each element of P_n divides c_n. Hence, by Theorem 12.2, elements of P_n are coprime to elements of P_k for $k \ne n$. Natural coprime bases do not contain 1, so $P_1, P_2, \ldots, P_m$ are disjoint, and their union P is a coprime set.

Next, by Theorem 12.2 again, $c_1, c_2, \ldots, c_m$ are coprime to $\operatorname{ppo}(a,b)$, so P is disjoint from R, and $P \cup R$ is a coprime set.

Next, each element of Q divides $b/d_1 d_2 \cdots d_m$, hence is coprime to a/c_0 by Theorem 12.3, hence is coprime to $c_1 c_2 \cdots c_m \operatorname{ppo}(a,b)$ by Theorem 12.2. Thus Q is disjoint from $P \cup R$, and $P \cup Q \cup R$ is a coprime set.

Next, the coid generated by $P \cup Q \cup R$ contains $b/c_0 d_1 d_2 \cdots d_m$ (via Q), c_0 (via Q), and each d_n (via P_n), so it contains b. It also contains $c_n/d_n^{2^{n-1}}$ and thus c_n (via P_n), plus $\operatorname{ppo}(a,b)$ (via R), so it contains $c_0 c_1 \cdots c_m \operatorname{ppo}(a,b) = a$ by Theorem 12.2.

Finally, naturalness follows from Theorem 7.3. □

Algorithm 13.2. Given (a,b), to print $\operatorname{cb}\{a,b\}$:

1. If $b = 1$: Print a if $a \ne 1$. Stop.
2. Compute $(a,r) \leftarrow (\operatorname{ppi}, \operatorname{ppo})(a,b)$ by Algorithm 11.3.
3. Print r if $r \ne 1$.
4. Compute $(g,h,c) \leftarrow (\gcd, \operatorname{ppg}, \operatorname{pple})(a,b)$ by Algorithm 11.4.
5. Set $c_0 \leftarrow c$. Set $x \leftarrow c_0$.
6. Set $n \leftarrow 1$.
7. Compute $(g,h,c) \leftarrow (\gcd, \operatorname{ppg}, \operatorname{pple})(h,g^2)$ by Algorithm 11.4.
8. Set $d \leftarrow \gcd\{c,b\}$. (Now $(g,h,c,d) = (g_n,h_n,c_n,d_n)$.)
9. Set $x \leftarrow xd$. (Now $x = c_0 d_1 d_2 \cdots d_n$.)
10. Compute $y \leftarrow d^{2^{n-1}}$ by Algorithm 10.1.
11. Recursively apply Algorithm 13.2 to $(c/y, d)$.
12. If $h \ne 1$: Set $n \leftarrow n+1$. Return to Step 7.
13. Recursively apply Algorithm 13.2 to $(b/x, c_0)$.

Beware that, even though $\operatorname{cb}\{a,b\} = \operatorname{cb}\{b,a\}$, the M-time used by Algorithm 13.2 may depend on the order of inputs. The order of arguments in Algorithm 13.2's recursive calls is important for the time analysis below.

Theorem 13.3. *Algorithm 13.2 computes* $\mathrm{cb}\{a,b\}$ *in* M*-time at most*

$$\lambda(a,b)(4m^2+12m+4)(\lg ab)\mu(3\lg ab)$$

if $m \geq 1$ *and* $2^{m-1} \geq \lg ab$.

Proof. If $b=1$ then Algorithm 13.2 stops immediately in Step 1, using no M-time; and $\lambda(a,b)(4m^2+12m+4)(\lg ab)\mu(3\lg ab) \geq 0$. So assume that $b \neq 1$.

The point is that $\lambda(a,b)$ decreases at each level of recursion: $\lambda(c/y,d) \leq \lambda(a,b)-1$ in Step 11, and $\lambda(b/x,c_0) \leq \lambda(a,b)-1$ in Step 13, by Theorem 12.3. So induct on $\lambda(a,b)$. The product of all the inputs in Steps 11 and 13 is $b\prod_{n\geq 1}(c_n/d_n^{2^{n-1}})$, which divides ab by Theorem 12.2. Hence the inputs a',b' to the recursive calls satisfy $\sum \lg a'b' \leq \lg ab$; in particular, $2^{m-1} \geq \lg a'b'$. By induction, the recursive call for a',b' uses M-time at most $(\lambda(a,b)-1)(4m^2+12m+4)(\lg a'b')\mu(3\lg a'b')$; so all the recursive calls together use M-time at most $(\lambda(a,b)-1)(4m^2+12m+4)(\lg ab)\mu(3\lg ab)$.

It therefore suffices to prove that the non-recursive work in Algorithm 13.2 takes M-time at most $(4m^2+12m+4)(\lg ab)\mu(3\lg ab)$.

By Theorem 11.5, Step 2 uses M-time at most $(3m+(2m+2)\lg a+\lg b)\mu(\lg ab)$; this is at most $((2m+2)\lg a+(3m+1)\lg b)\mu(\lg ab)$ since $1 \leq \lg b$. Similarly, by Theorem 11.6, Step 4 uses M-time at most $((2m+2)\lg a+(3m+1)\lg b)\mu(\lg ab)$.

Steps 7 through 12 are performed for $n \in \{1,2,\ldots,m\}$ at most, since $h_m = 1$ by Theorem 12.4.

The squaring of g in Step 7 uses M-time at most $(1+2\lg g)\mu(2\lg g)$ per iteration. The total across iterations is at most $(2m\lg a+m\lg b)\mu(2\lg a)$ since g divides a.

By Theorem 11.6, the invocation of Algorithm 11.3 in Step 7 uses M-time at most $(3m+(2m+2)\lg h+\lg g^2)\mu(\lg hg^2)$ per iteration, since $2^{m-1} \geq \lg a \geq \lg h$. The total across iterations is at most $(m(2m+4)\lg a+3m^2\lg b)\mu(3\lg a)$ since g and h divide a.

Step 8 uses M-time at most $(1+\lg ab)\mu(\lg ab)$ per iteration since c divides a. The total across iterations is at most $(m\lg a+2m\lg b)\mu(\lg ab)$.

Step 9 uses M-time at most $(1+\lg b)\mu(\lg b)$ per iteration since the final value of x divides b. The total across iterations is at most $(2m\lg b)\mu(\lg b)$.

Step 10 uses M-time at most $(n-1+2(2^{n-1}-1)\lg d)\mu(2^{n-1}\lg d)$ by Theorem 10.2; this is at most $(2\lg a+m\lg b)\mu(\lg a)$ since $2^{n-1}\lg d \leq \lg a$. The total across iterations is at most $(2m\lg a+m^2\lg b)\mu(\lg a)$.

The division in Step 11 uses M-time at most $(1+\lg a)\mu(\lg a)$ per iteration since c divides a. The total across iterations is at most $(m\lg a+m\lg b)\mu(\lg a)$.

The division in Step 13 uses M-time at most $(1+\lg b)\mu(\lg b) \leq (2\lg b)\mu(\lg b)$.

The grand total is at most $\mu(3\lg ab)$ times $(2m^2+14m+4)\lg a+(4m^2+12m+4)\lg b \leq (4m^2+12m+4)\lg a+(4m^2+12m+4)\lg b = (4m^2+12m+4)\lg ab$ as claimed. □

Theorem 13.4. *Algorithm 13.2 computes* $\mathrm{cb}\{a,b\}$ *in* M*-time at most* $8m(m^2+3m+1)\cdot(\lg ab)\mu(3\lg ab)$ *if* $m \geq 1$ *and* $2^{m-1} \geq \lg ab$.

Proof. $\lambda(a,b) \leq 2m$ by Theorem 9.3. □

14. Computing prod

The following algorithm is the standard way to compute $\operatorname{prod} S$ in essentially linear time. See [7, Section 12] for credits.

Algorithm 14.1. Given a finite set S, to print $\operatorname{prod} S$:

1. If $S = \{\}$: Print 1. Stop.
2. If $\#S = 1$: Find $a \in S$. Print a. Stop.
3. Select $T \subseteq S$ with $\#T = \lfloor \#S/2 \rfloor$.
4. Compute $X \leftarrow \operatorname{prod} T$ by Algorithm 14.1 recursively.
5. Compute $Y \leftarrow \operatorname{prod}(S - T)$ by Algorithm 14.1 recursively.
6. Print XY.

Theorem 14.2. *Algorithm 14.1 computes* $\operatorname{prod} S$ *in* M*-time at most* $(\#S - 1 + m \lg \operatorname{prod} S) \cdot \mu(\lg \operatorname{prod} S)$ *if* $2^m \ge \#S \ge 1$.

Proof. Induct on m.

Case 1: $\#S = 1$. Algorithm 14.1 uses no M-time; and $\#S - 1 + m \lg \operatorname{prod} S \ge 0$.

Case 2: $\#S \ge 2$. Then $m \ge 1$. Now $\#T = \lfloor \#S/2 \rfloor$, so $\#T \le 2^{m-1}$ and $\#(S - T) \le 2^{m-1}$. In Steps 4 and 5, by induction, Algorithm 14.1 uses M-time at most

$$\begin{aligned}&(\#T - 1 + (m-1) \lg \operatorname{prod} T)\mu(\lg \operatorname{prod} T)\\ &\qquad + (\#(S-T) - 1 + (m-1) \lg \operatorname{prod}(S-T))\mu(\lg \operatorname{prod}(S-T))\\ &\quad \le (\#S - 2 + (m-1) \lg \operatorname{prod} S)\mu(\lg \operatorname{prod} S)\end{aligned}$$

since $\lg \operatorname{prod} T + \lg \operatorname{prod}(S - T) = \lg \operatorname{prod} S$. In Step 6, Algorithm 14.1 uses M-time at most $(1 + \lg \operatorname{prod} S)\mu(\lg \operatorname{prod} S)$. Add: $\#S - 2 + (m-1) \lg \operatorname{prod} S + 1 + \lg \operatorname{prod} S = \#S - 1 + m \lg \operatorname{prod} S$. $\square$

15. The split function

Define $\operatorname{split}(a, P) = \{(p, \operatorname{ppi}(a, p)) : p \in P\}$ when P is coprime. This section presents a fast algorithm to compute $\operatorname{split}(a, P)$. See Sections 16 and 21 for applications.

Theorem 15.1. *Let b be an element of a free coid H. Let P be a finite coprime subset of H. Then* $b = \operatorname{ppo}(b, \operatorname{prod} P) \prod_{p \in P} \operatorname{ppi}(b, p)$.

Proof. Write $x = \operatorname{prod} P$. I will show that $\operatorname{ord}_q b = \operatorname{ord}_q \operatorname{ppo}(b, x) + \sum_{p \in P} \operatorname{ord}_q \operatorname{ppi}(b, p)$ for every prime q: one of the terms on the right side is $\operatorname{ord}_q b$ while the others are all 0.

Case 1: q divides some $p \in P$. Then $\operatorname{ord}_q x > 0$ so $\operatorname{ord}_q \operatorname{ppo}(b, x) = 0$, and $\operatorname{ord}_q p > 0$ so $\operatorname{ord}_q \operatorname{ppi}(b, p) = \operatorname{ord}_q b$. If $p' \in P$ with $p' \ne p$ then p' is coprime to p by hypothesis, so $\operatorname{ord}_q p' = 0$, so $\operatorname{ord}_q \operatorname{ppi}(b, p') = 0$.

Case 2: q does not divide any $p \in P$. Then $\operatorname{ord}_q p = 0$ so $\operatorname{ord}_q \operatorname{ppi}(b,p) = 0$. Also $\operatorname{ord}_q x = 0$, so $\operatorname{ord}_q \operatorname{ppo}(b,x) = \operatorname{ord}_q b$. $\square$

Theorem 15.2. *Let a be an element of a free coid H. Let P be a finite coprime subset of H. Define $b = \operatorname{ppi}(a, \operatorname{prod} P)$. Then $\operatorname{split}(a,P) = \operatorname{split}(b,P)$, and $\operatorname{split}(a,P) = \{(p,b)\}$ if $P = \{p\}$.*

Proof. Fix $p \in P$. If q is a prime then $\operatorname{ord}_q b = (\operatorname{ord}_q a)[\operatorname{ord}_q \operatorname{prod} P > 0]$ so $\operatorname{ord}_q \operatorname{ppi}(b,p) = (\operatorname{ord}_q a)[\operatorname{ord}_q \operatorname{prod} P > 0][\operatorname{ord}_q p > 0] = (\operatorname{ord}_q a)[\operatorname{ord}_q p > 0] = \operatorname{ord}_q \operatorname{ppi}(a,p)$. Therefore $\operatorname{ppi}(a,p) = \operatorname{ppi}(b,p)$.

If $P = \{p\}$ then $\operatorname{prod} P = p$ so $b = \operatorname{ppi}(a,p)$ so $\operatorname{split}(a,P) = \{(p,b)\}$. $\square$

Algorithm 15.3. Given (a,P) with P coprime, to print $\operatorname{split}(a,P)$:

1. If $P = \{\}$: Stop.
2. Compute $b \leftarrow \operatorname{ppi}(a, \operatorname{prod} P)$ by Algorithm 14.1 and Algorithm 11.3.
3. If $\#P = 1$: find $p \in P$, print (p,b), and stop.
4. Select $Q \subseteq P$ with $\#Q = \lfloor \#P/2 \rfloor$.
5. Print $\operatorname{split}(b,Q)$ by Algorithm 15.3 recursively.
6. Print $\operatorname{split}(b,P-Q)$ by Algorithm 15.3 recursively.

Theorem 15.4. *Write $x = \operatorname{prod} P$ and $b = \operatorname{ppi}(a,x)$. Algorithm 15.3 computes $\operatorname{split}(a,P)$ in M-time at most $((2k+4)(\lg a + 2m \lg b) + \frac{(m+1)(m+2)}{2} \lg x + (6k+m+5)\#P - 3k - 2) \cdot \mu(\lg ax)$ if $k \geq 0$, $2^k \geq \lg a$, and $2^m \geq \#P \geq 1$.*

Proof. Induct on m.

Step 2 uses M-time at most $(\#P - 1 + m \lg x)\mu(\lg x)$ to compute x by Theorem 14.2, and M-time at most $(3k + 3 + (2k+4)\lg a + \lg x)\mu(\lg ax)$ to compute b by Theorem 11.5. I claim that the rest of Algorithm 15.3 uses M-time at most $(2k+4)(2m \lg b) + \frac{m(m+1)}{2} \lg x + (6k+m+4)\#P - 6k - 4$ times $\mu(\lg ax)$.

Case 1: $\#P = 1$. Then Algorithm 15.3 stops in Step 3, so there is no additional M-time; and $(2k+4)(2m \lg b) + \frac{m(m+1)}{2} \lg x + (6k+m+4)\#P - 6k - 4 \geq 0$ since $m \geq 0$.

Case 2: $\#P \geq 2$. Write $y = \operatorname{prod} Q$ and $z = \operatorname{prod}(P-Q)$. Then y is coprime to z, so $b = \operatorname{ppo}(b,yz)\operatorname{ppi}(b,y)\operatorname{ppi}(b,z)$ by Theorem 15.1; hence $\lg \operatorname{ppi}(b,y) + \lg \operatorname{ppi}(b,z) \leq \lg b$. By induction, Step 5 uses M-time at most

$$(2k+4)(\lg b + 2(m-1)\lg \operatorname{ppi}(b,y)) + \frac{m(m+1)}{2} \lg y + (6k+m+4)\#Q - 3k - 2$$

times $\mu(\lg by) \leq \mu(\lg ax)$, and Step 6 uses M-time at most

$$(2k+4)(\lg b + 2(m-1)\lg \operatorname{ppi}(b,z)) + \frac{m(m+1)}{2} \lg z + (6k+m+4)\#(P-Q) - 3k - 2$$

times $\mu(\lg bz) \leq \mu(\lg ax)$, so Steps 5 and 6 together use M-time at most

$$(2k+4)(2\lg b + 2(m-1)\lg b) + \frac{m(m+1)}{2} \lg x + (6k+m+4)\#P - 6k - 4$$

times $\mu(\lg ax)$ as claimed. □

16. Extending a coprime base

Algorithm 16.2 finds $\operatorname{cb}(P \cup \{b\})$ when P is coprime.

Bach, Driscoll, and Shallit considered this problem in [2, page 211]. They used elements of $P = \{p_0, p_1, \ldots\}$ one at a time: factor b and p_0 into coprimes, all of which divide p_0 except for a divisor b_1 of b; then factor b_1 and p_1 into coprimes, all of which divide p_1 except for a divisor b_2 of b_1; and so on. This strategy inherently takes quadratic time.

Algorithm 16.2 instead uses Algorithm 15.3 to quickly factor b into one part for each element of P and one remaining part; see Theorem 15.1. Algorithm 16.2 then handles each part independently.

Theorem 16.1. *Let b be an element of a free coid H. Let P be a finite coprime subset of H. Define $x = \operatorname{prod} P$. For each $p \in P$ define $Q_p = \operatorname{cb}\{p, \operatorname{ppi}(b,p)\}$. Define $R = \operatorname{cb}\{\operatorname{ppo}(b,x)\}$. Then $Q = \bigcup Q_p$ is a disjoint union; $Q \cup R$ is a disjoint union; and $Q \cup R = \operatorname{cb}(P \cup \{b\})$.*

Proof. If p and p' are distinct elements of P then p is coprime to p' so $p \operatorname{ppi}(b,p)$ is coprime to $p' \operatorname{ppi}(b,p')$. Thus Q_p and $Q_{p'}$ are disjoint, and Q is a coprime set.

If $p \in P$ then p divides x so both p and $\operatorname{ppi}(b,p)$ are coprime to $\operatorname{ppo}(b,x)$. Thus Q and R are disjoint, and $Q \cup R$ is a coprime set.

The coid generated by $Q \cup R$ contains each $p \in P$ (via Q_p). It also contains each $\operatorname{ppi}(b,p)$ and $\operatorname{ppo}(b,x)$, hence b by Theorem 15.1.

Naturalness follows from Theorem 7.3. □

Algorithm 16.2. Given (P,b) with P coprime, to print $\operatorname{cb}(P \cup \{b\})$:

1. If $P = \{\}$: Print b if $b \neq 1$. Stop.
2. Compute $x \leftarrow \operatorname{prod} P$ by Algorithm 14.1.
3. Compute $(a,r) \leftarrow (\operatorname{ppi}, \operatorname{ppo})(b,x)$ by Algorithm 11.3.
4. Print r if $r \neq 1$.
5. Compute $S \leftarrow \operatorname{split}(a,P)$ by Algorithm 15.3.
6. For each $(p,c) \in S$: Apply Algorithm 13.2 to (p,c).

Theorem 16.3. *Write $x = \operatorname{prod} P$. Algorithm 16.2 computes $\operatorname{cb}(P \cup \{b\})$ in M-time at most $(8m^3 + 28m^2 + 18m + 4)(\lg bx)\mu(3\lg bx)$ if $1 \notin P$, $m \geq 1$, and $2^{m-1} \geq \lg bx$.*

Proof. If $P = \{\}$ then Algorithm 16.2 uses no M-time; and $(8m^3 + 28m^2 + 18m + 4)\lg bx \geq 0$. So assume that $\#P \geq 1$. Note that $\#P \leq \lg x \leq 2^{m-1}$ since $1 \notin P$.

By Theorem 14.2, Step 2 takes M-time at most $(\#P - 1 + m\lg x)\mu(\lg x)$.

By Theorem 11.5, Step 3 takes M-time at most $(3m + (2m+2)\lg b + \lg x)\mu(\lg bx)$.

By Theorem 15.4, Step 5 takes M-time at most $\mu(\lg bx)$ times $(2m+2)(2m+1)\lg b + \frac{1}{2}(m+1)(m+2)\lg x + (7m-1)\#P - 3m + 1$.

By Theorem 13.4, the application of Algorithm 13.2 to (p,c) in Step 6 takes M-time at most $8m(m^2+3m+1)(\lg cp)\mu(3\lg cp)$. The product of c's divides b by Theorem 15.1, so the sum of $\lg cp$ is at most $\lg bx$.

Add:

$$\begin{aligned}
&\#P - 1 + m\lg x + 3m + (2m+2)\lg b + \lg x \\
&\qquad + (2m+2)(2m+1)\lg b + \tfrac{1}{2}(m+1)(m+2)\lg x + (7m-1)\#P - 3m + 1 \\
&\qquad + 8m(m^2+3m+1)(\lg bx) \\
&\quad = (8m^3+28m^2+16m+4)\lg b + (8m^3+24.5m^2+10.5m+2)\lg x + 7m\#P \\
&\quad \le (8m^3+28m^2+16m+4)\lg b + (8m^3+24.5m^2+17.5m+2)\lg x \\
&\quad \le (8m^3+28m^2+18m+4)\lg bx.
\end{aligned}$$

□

17. Merging coprime bases

Algorithm 17.3 finds $\operatorname{cb}(P \cup Q)$ if P is coprime and Q is coprime.

This algorithm combines an old idea with a new idea. The old idea is to hit P with one element of Q at a time. For example, $\operatorname{cb}(P \cup \{q_0, q_1, q_2, q_3\})$ can be computed as $\operatorname{cb}(\operatorname{cb}(\operatorname{cb}(\operatorname{cb}(P \cup \{q_0\}) \cup \{q_1\}) \cup \{q_2\}) \cup \{q_3\})$ with four applications of Algorithm 16.2. This is how Bach, Driscoll, and Shallit compute $\operatorname{cb} Q$ for arbitrary sets Q in [2, page 211]; this idea does not need Q to be a coprime set.

The problem with the old idea is that it inherently takes quadratic time if $\#Q$ is large. The new idea is to first replace Q with a new set that has far fewer elements but has Q as its natural coprime base. See Theorem 17.1. The new set takes more space than Q, but the expansion is only logarithmic.

Define $\operatorname{bit}_i k$, where i and k are nonnegative integers, as the ith bit in k's binary expansion. In other words, write k as $\sum_{i \ge 0} 2^i \operatorname{bit}_i k$ with $\operatorname{bit}_i k \in \{0, 1\}$.

Theorem 17.1. *Let $q_0, q_1, \ldots, q_{n-1}$ be distinct elements of a free coid, with q_j coprime to q_k for $j \ne k$. Let $b \ge 1$ be an integer with $2^b \ge n$. Define $x(e,i) = \operatorname{prod}\{q_k : \operatorname{bit}_i k = e\}$. If a coprime set P is a base for $\{x(0,0), x(0,1), \ldots, x(0,b-1), x(1,0), x(1,1), \ldots, x(1,b-1)\}$ then it is a base for $\{q_0, \ldots, q_{n-1}\}$.*

Proof. If $j, k \in \{0, 1, \ldots, n-1\}$ satisfy $\operatorname{bit}_i k = \operatorname{bit}_i j$ for all $i \in \{0, 1, \ldots, b-1\}$ then $j = k$. Thus

$$\begin{aligned}
&\gcd\{x(\operatorname{bit}_i j, i) : 0 \le i < b\} \\
&\quad = \gcd\left\{\textstyle\prod_k q_k^{[\operatorname{bit}_i k = \operatorname{bit}_i j]} : 0 \le i < b\right\} && \text{by definition of } x \\
&\quad = \textstyle\prod_k q_k^{\min\{[\operatorname{bit}_i k = \operatorname{bit}_i j] : 0 \le i < b\}} && \text{by Theorem 6.6} \\
&\quad = \textstyle\prod_k q_k^{[k=j]} = q_j.
\end{aligned}$$

Hence P is a base for q_j by Theorem 7.2. □

Theorem 17.2. *Let H be a free coid. Let P be a finite coprime subset of H. Let $q_0, \ldots, q_{n-1}$ be distinct elements of H, with q_j coprime to q_k for $j \ne k$. Let $b \ge 1$ be an integer with*

$2^b \ge n$. *Define* $x(e,i) = \operatorname{prod}\{q_k : \operatorname{bit}_i k = e\}$. *Define*

$$
\begin{aligned}
S_0 &= P \\
S_1 &= \operatorname{cb}\Big(\operatorname{cb}\big(S_0 \cup \{x(0,0)\}\big) \cup \{x(1,0)\}\Big) \\
S_2 &= \operatorname{cb}\Big(\operatorname{cb}\big(S_1 \cup \{x(0,1)\}\big) \cup \{x(1,1)\}\Big) \\
S_3 &= \operatorname{cb}\Big(\operatorname{cb}\big(S_2 \cup \{x(0,2)\}\big) \cup \{x(1,2)\}\Big) \\
&\vdots \\
S_b &= \operatorname{cb}\Big(\operatorname{cb}\big(S_{b-1} \cup \{x(0,b-1)\}\big) \cup \{x(1,b-1)\}\Big).
\end{aligned}
$$

Then $S_b = \operatorname{cb}(P \cup \{q_0, q_1, \ldots, q_{n-1}\})$.

Proof. S_b is a base for $P \cup \{x(0,0), x(0,1), \ldots, x(0,b-1), x(1,0), x(1,1), \ldots, x(1,b-1)\}$ and is coprime. By Theorem 17.1, S_b is a base for $P \cup \{q_0, \ldots, q_{n-1}\}$. Naturalness follows from Theorem 7.3. □

Algorithm 17.3. Given (P, Q), with P coprime and Q coprime, to print $\operatorname{cb}(P \cup Q)$:

1. Set $n = \#Q$. Label the elements of Q as $q_0, q_1, \ldots, q_{n-1}$.
2. Find the smallest $b \ge 1$ with $2^b \ge n$. Set $S \leftarrow P$. Set $i \leftarrow 0$.
3. (Now $S = S_i$.) If $i = b$: Print S. Stop.
4. Compute $x \leftarrow \operatorname{prod}\{q_k : \operatorname{bit}_i k = 0\}$ by Algorithm 14.1.
5. Compute $T \leftarrow \operatorname{cb}(S \cup \{x\})$ by Algorithm 16.2.
6. Compute $x \leftarrow \operatorname{prod}\{q_k : \operatorname{bit}_i k = 1\}$ by Algorithm 14.1.
7. Compute $S \leftarrow \operatorname{cb}(T \cup \{x\})$ by Algorithm 16.2.
8. Set $i \leftarrow i+1$. Return to Step 3.

Theorem 17.4. *Write* $z = (\operatorname{prod} P)(\operatorname{prod} Q)^2$. *Algorithm 17.3 computes* $\operatorname{cb}(P \cup Q)$ *in M-time at most* $2m(8m^3 + 28m^2 + 19m + 4)(\lg z)\mu(3\lg z)$ *if* $1 \notin P$, $m \ge 1$, *and* $2^{m-1} \ge \lg z$.

Proof. Write $n = \#Q$. Then $n - 1 \le \lg \operatorname{prod} Q \le \lg z \le 2^{m-1}$, so $n \le 1 + 2^{m-1} \le 2^m$, so $b \le m$ in Step 2. Thus there are at most m iterations of Steps 4 through 7.

By Theorem 7.5, $\operatorname{prod} S$ and $\operatorname{prod} T$ divide $(\operatorname{prod} P)(\operatorname{prod} Q)$; also, x divides $\operatorname{prod} Q$. Thus Step 5 and Step 7 each use M-time at most $(8m^3 + 28m^2 + 18m + 4)(\lg z)\mu(3\lg z)$ per iteration by Theorem 16.3. Step 4 and Step 6 each use M-time at most $m(\lg z)\mu(\lg z)$ by Theorem 14.2. The total is at most $2(8m^3 + 28m^2 + 19m + 4)(\lg z)\mu(3\lg z)$ per iteration. □

18. Computing a coprime base for a finite set

Algorithm 18.1 computes the natural coprime base for any finite subset of a free coid. It uses Algorithm 17.3 to merge coprime bases for halves of the set.

Algorithm 18.1. Given S, to print $\operatorname{cb} S$:

1. If $S = \{\}$: Stop.
2. If $\#S = 1$: Find $a \in S$. Print a if $a \neq 1$. Stop.
3. Select $T \subseteq S$ with $\#T = \lfloor \#S/2 \rfloor$.
4. Compute $P \leftarrow \operatorname{cb} T$ by Algorithm 18.1 recursively.
5. Compute $Q \leftarrow \operatorname{cb}(S - T)$ by Algorithm 18.1 recursively.
6. Print $\operatorname{cb}(P \cup Q)$ by Algorithm 17.3.

Theorem 18.2. *Write $x = \operatorname{prod} S$. Algorithm 18.1 computes $\operatorname{cb} S$ in M-time at most*

$$4mk(8m^3 + 28m^2 + 19m + 4)(\lg x)\mu(6\lg x)$$

if $m \geq 1$, $2^{m-1} \geq 2\lg x$, and $2^k \geq \#S \geq 1$.

Proof. If $\#S = 1$ then Algorithm 18.1 uses no M-time. Otherwise, by induction on k, Step 4 uses M-time at most $4m(k-1)(8m^3 + 28m^2 + 19m + 4)(\lg \operatorname{prod} T)\mu(6\lg x)$, and Step 5 uses M-time at most $4m(k-1)(8m^3 + 28m^2 + 19m + 4)(\lg \operatorname{prod}(S - T))\mu(6\lg x)$. Step 6 uses M-time at most $2m(8m^3 + 28m^2 + 19m + 4)(2\lg x)\mu(6\lg x)$ by Theorem 17.4. Add. □

PART IV. FACTORIZATION

19. The reduce function

Let p and a be elements of a free coid, with $p \neq 1$. Define $\operatorname{reduce}(p, a) = (i, a/p^i)$, where i is the largest integer such that p^i divides a.

Algorithm 19.2 computes $\operatorname{reduce}(p, a)$. It is a simplified version of one of the algorithms that I outlined in [6, Section 22].

Theorem 19.1. *Let p, a be elements of a free coid, with $p \neq 1$. Assume that p divides a. Define $(j, b) = \operatorname{reduce}(p^2, a/p)$. If p divides b then $\operatorname{reduce}(p, a) = (2j+2, b/p)$. Otherwise $\operatorname{reduce}(p, a) = (2j+1, b)$.*

Proof. $a/p = (p^2)^j b$, with p^2 not dividing b, by definition of (j, b). If p divides b then $a = p^{2j+2}(b/p)$, with p not dividing b/p, so $\operatorname{reduce}(p, a) = (2j+2, b/p)$. Otherwise $a = p^{2j+1}b$, with p not dividing b, so $\operatorname{reduce}(p, a) = (2j+1, b)$. □

Algorithm 19.2. Given (p, a) with $p \neq 1$, to print $\operatorname{reduce}(p, a)$:
1. If p does not divide a: Print $(0, a)$ and stop.
2. Compute $(j, b) \leftarrow \operatorname{reduce}(p^2, a/p)$ by Algorithm 19.2 recursively.
3. If p divides b: Print $(2j+2, b/p)$ and stop.
4. Print $(2j+1, b)$.

Theorem 19.3. *Write $(i, c) = \operatorname{reduce}(p, a)$. Algorithm 19.2 computes (i, c) in M-time at most $(4k-3)(1 + \lg ap)\mu(\lg ap)$ if $2^k > i+1$.*

Proof. Induct on k. Note that $k \geq 1$ since $2^k \geq i+2 \geq 2$.

Case 1: $i = 0$. Algorithm 19.2 uses M-time at most $(1+\lg ap)\mu(\lg ap)$ in Step 1; it then stops, since p does not divide a. Also $4k-3 \geq 1$.

Case 2: $i > 0$. Then $j \leq (i-1)/2$ in Step 2 so $j+1 \leq (i+1)/2 < 2^{k-1}$. By induction, the recursive call in Algorithm 19.2 uses M-time at most $(4k-7)(1+\lg ap)\mu(\lg ap)$. The algorithm also uses M-time at most $(1+\lg ap)\mu(\lg ap)$ for the divisibility test in Step 1, $(1+\lg p^2)\mu(\lg p^2)$ for the computation of p^2 in Step 2, $(1+\lg a)\mu(\lg a)$ for the computation of a/p in Step 2, $(1+\lg bp)\mu(\lg bp)$ for the divisibility test in Step 3, and $(1+\lg b)\mu(\lg b)$ for the division in Step 3, if that division happens. The total is at most $\mu(\lg ap)$ times $(4k-3)(1+\lg ap)+2\lg b-2\lg a+1$; and $2\lg b-2\lg a+1 \leq 2\lg(a/p)-2\lg a+1 = 1-2\lg p < 0$ since $\lg p \geq 1$. $\square$

20. Factoring over a coprime base

Let a be an element of a free coid, and let P be a finite coprime set with $1 \notin P$. Algorithm 20.1 factors a as a product of powers of elements of P if possible; otherwise it proclaims failure. Algorithm 20.1 prints the factorization of a as a list of pairs (p,n) meaning p^n where $p \in P$.

The conventional approach to this problem, as in [2, Theorem 7], is to divide a by one element of P at a time. This approach inherently takes quadratic time. What I do instead is separate a into two pieces for two halves of P, and then handle each piece recursively, as in Algorithm 15.3.

Algorithm 20.1. Given (a,P), with P coprime and $1 \notin P$, to print the factorization of a over P:

1. If $P = \{\}$: Proclaim failure if $a \neq 1$. Stop.
2. If $\#P = 1$: Find $p \in P$. Compute $(n,c) \leftarrow \operatorname{reduce}(p,a)$ by Algorithm 19.2. If $c \neq 1$, proclaim failure and stop. Otherwise print (p,n) and stop.
3. Select $Q \subseteq P$ with $\#Q = \lfloor \#P/2 \rfloor$.
4. Compute $y \leftarrow \operatorname{prod} Q$ by Algorithm 14.1.
5. Compute $(b,c) \leftarrow (\operatorname{ppi},\operatorname{ppo})(a,y)$ by Algorithm 11.3.
6. Apply Algorithm 20.1 to (b,Q) recursively. If Algorithm 20.1 fails, proclaim failure and stop.
7. Apply Algorithm 20.1 to $(c,P-Q)$ recursively. If Algorithm 20.1 fails, proclaim failure and stop.

Theorem 20.2. *Let H be a free coid. Let P be a finite coprime subset of H with $1 \notin P$. Let a be an element of H. If P is a base for $\{a\}$ then Algorithm 20.1 prints the factorization of a over P. Otherwise Algorithm 20.1 proclaims failure.*

Proof. Induct on $\#P$.

Case 1: $P = \{\}$. Algorithm 20.1 correctly proclaims failure for $a \neq 1$, and correctly prints nothing for $a = 1$.

Case 2: $P = \{p\}$. If Algorithm 20.1 does not proclaim failure, then it prints (p,n); and $a/p^n = c = 1$ so $a = p^n$. Conversely, if $a = p^n$ for some n, then Algorithm 19.2 returns $(n,1)$, so Algorithm 20.1 does not proclaim failure.

Case 3: $\#P \geq 2$. Say P is a base for $\{a\}$. P is also a base for $\{y\}$, so P is a base for $\{b,c\}$ by Theorem 7.2. If $p \notin Q$ then p is coprime to y so p is coprime to b; thus Q is a base for $\{b\}$. Similarly $P-Q$ is a base for $\{c\}$. By induction, Algorithm 20.1 prints the factorizations of b and c into elements of Q and $P-Q$ respectively, which together form a factorization of a since $bc = a$; and Algorithm 20.1 does not proclaim failure.

Conversely, if Algorithm 20.1 does not proclaim failure, then P is a base for $\{b,c\}$ by induction, hence for $\{a\}$. □

Theorem 20.3. *Write $x = \operatorname{prod} P$. Algorithm 20.1 finishes in M-time at most $\mu(\lg ax)$ times*

$$(4k-3+m(2k+4))\lg a + \left(4k-3+\frac{m(m+1)}{2}\right)\lg x + \left(7k-1+\frac{m}{2}\right)\#P - 3k - 2$$

if $2^m \geq \#P \geq 1$ and $2^k > \lg a + 1$.

Proof. Induct on m.

Case 1: $P = \{p\}$. The claimed bound is at least

$$((4k-3)\lg a + (4k-3)\lg x + 7k - 1 - 3k - 2)\mu(\lg ax) = (4k-3)(1+\lg ap)\mu(\lg ap)$$

since $m \geq 0$. Step 2 uses M-time at most $(4k-3)(1+\lg ap)\mu(\lg ap)$ by Theorem 19.3.

Case 2: $\#P \geq 2$. Then $m \geq 1$, $2^{m-1} \geq \#Q \geq 1$, and $2^{m-1} \geq \#(P-Q) \geq 1$. Define $y = \operatorname{prod} Q$ and $z = \operatorname{prod}(P-Q)$. Also write $T = 4k-3+(m-1)(2k+4)$ and $U = 4k-3+(m-1)m/2$.

By Theorem 14.2, Step 4 uses M-time at most

$$(\#Q - 1 + (m-1)\lg y)\mu(\lg y) \leq \left(\tfrac{1}{2}\#P - 1 + (m-1)\lg x\right)\mu(\lg x)$$

since $\#Q \leq \frac{1}{2}\#P$ and $\lg y \leq \lg x$.

By Theorem 11.5, Step 5 uses M-time at most $(3k+3+(2k+4)\lg a + \lg x)\mu(\lg ax)$.

Step 6 uses M-time at most $\left(T\lg b + U\lg y + \left(7k-1+\frac{1}{2}(m-1)\right)\#Q - 3k - 2\right)\mu(\lg by)$, and Step 7 at most $\left(T\lg c + U\lg z + \left(7k-1+\frac{1}{2}(m-1)\right)\#(P-Q) - 3k - 2\right)\mu(\lg cz)$, by induction.

The total is at most $\mu(\lg ax)$ times

$$\begin{aligned} &\left(\tfrac{1}{2}\#P - 1 + (m-1)\lg x\right) + (3k+3+(2k+4)\lg a + \lg x) \\ &\quad + \left(T\lg a + U\lg x + \left(7k-1+\tfrac{1}{2}(m-1)\right)\#P - 6k - 4\right), \end{aligned}$$

which equals the claimed bound. □

21. Factoring a set over a coprime base

Let S be a finite set, and let P be a finite coprime set with $1 \notin P$. Algorithm 21.2 factors each element $a \in S$ over P if P is a base for S; otherwise it proclaims failure.

The conventional approach to this problem, as in [2, Theorem 7], is to separately factor each element of S. This approach inherently takes quadratic time. What I do instead is factor one number, $\operatorname{prod} S$, to identify the relevant elements of P; then I split S into two parts to handle separately.

Theorem 21.1. *Let S be a finite subset of a free coid. Let P be a finite coprime base for S with $1 \notin P$. Define $x = \operatorname{prod} P$, $y = \operatorname{prod} S$, $z = \operatorname{ppi}(x,y)$, and $Q = \{p \in P : \operatorname{ppi}(z,p) = p\}$. Then Q is a base for S, and each element of Q divides y.*

Proof. Take any $p \in P$ that divides some element of S. If q is a prime dividing p then q divides y so $\operatorname{ord}_q z = \operatorname{ord}_q x$ so $\operatorname{ord}_q \operatorname{ppi}(z,p) = \operatorname{ord}_q x = \operatorname{ord}_q p$; if q is a prime not dividing p then $\operatorname{ord}_q \operatorname{ppi}(z,p) = 0 = \operatorname{ord}_q p$. Thus $\operatorname{ppi}(z,p) = p$; i.e., $p \in Q$. Thus Q is a base for S.

Now write y as a product $q_1 q_2 \cdots q_n$ where $q_1, q_2, \ldots, q_n \in Q$. If $p \in P$ does not divide y, then $p \notin \{q_1, q_2, \ldots, q_n\}$, so p is coprime to $q_1, q_2, \ldots, q_n$, hence to y. Find a prime q dividing p; then $\operatorname{ord}_q y = 0$, so $\operatorname{ord}_q z = 0$, so $\operatorname{ord}_q \operatorname{ppi}(z,p) = 0 \neq \operatorname{ord}_q p$, so $p \notin Q$. Thus each element of Q divides y. $\square$

Algorithm 21.2. Given (S,P), with P coprime and $1 \notin P$, to print the factorization of each element of S over P:

1. If $S = \{\}$: Stop.
2. Compute $x \leftarrow \operatorname{prod} P$ by Algorithm 14.1.
3. Compute $y \leftarrow \operatorname{prod} S$ by Algorithm 14.1.
4. Compute $z \leftarrow \operatorname{ppi}(x,y)$ by Algorithm 11.3.
5. Compute $D \leftarrow \operatorname{split}(z,P)$ by Algorithm 15.3.
6. Compute $Q \leftarrow \{p \in P : (p,p) \in D\}$. (Now Q contains only the elements of P that are relevant to S, by Theorem 21.1.)
7. If $\#S = 1$: Apply Algorithm 20.1 to (y,Q), proclaiming failure if Algorithm 20.1 fails. Stop.
8. Select $T \subseteq S$ with $\#T = \lfloor \#S/2 \rfloor$.
9. Apply Algorithm 21.2 to (T,Q) recursively.
10. Apply Algorithm 21.2 to $(S-T,Q)$ recursively.

Theorem 21.3. *Write $x = \operatorname{prod} P$ and $y = \operatorname{prod} S$. Assume that P is a coprime base for S. Then Algorithm 21.2 finishes in M-time at most $\mu(\lg x^2 y)$ times*

$$\begin{aligned} &3m+3+(6k+6)(\#S-1)+(4.5m^2+21.5m+15)\lg x \\ &+((9k^2+44k+32)n+2.5k^2+21k-5)\lg y \end{aligned}$$

if $2^n \geq \#S \geq 1$, $2^k > \lg y + 1$, $2^m \geq \lg x$, and $m \geq 0$.

Proof. Induct on n. Note that $\#S - 1 \le \lg y$. Similarly $\#P \le \lg x$ since $1 \notin P$, and $\#Q \le \lg \operatorname{prod} Q \le \lg y$ by Theorem 21.1. Write $z = \operatorname{ppi}(x, y)$.

Step 2 uses M-time at most $(m+1)(\lg x)\mu(\lg x)$. This follows from Theorem 14.2 for $\#P \ge 1$. (For $\#P = 0$, Algorithm 14.1 uses no M-time.) Similarly, Step 3 uses M-time at most $(k+1)(\lg y)\mu(\lg y)$.

Step 4 uses M-time at most $(3m+3+(2m+4)\lg x + \lg y)\mu(\lg xy)$ by Theorem 11.5.

Step 5 uses M-time less than $(4.5m^2 + 18.5m + 10)(\lg x)\mu(\lg x^2)$. This follows from Theorem 15.4 for $\#P \ge 1$, since $\lg z \le \lg x$. (For $\#P = 0$, Algorithm 15.3 uses no M-time.)

The total so far is at most $(3m+3+(4.5m^2+21.5m+15)\lg x + (k+2)\lg y)\mu(\lg x^2 y)$, which is exactly $\left((6k+6)(\#S-1) + ((9k^2+44k+32)n + 2.5k^2 + 20k - 7)\lg y\right)\mu(\lg x^2 y)$ below the claimed bound.

Case 1: $\#S = 1$. Then Step 7 uses M-time at most $(2.5k^2 + 20k - 7)\lg y$ times $\mu(\lg x^2 y)$. This follows from Theorem 20.3 for $\#Q \ge 1$, since $2^k \ge \#Q$. (For $\#Q = 0$, Algorithm 20.1 uses no M-time; note that $2.5k^2 + 20k - 7 > 0$ since $k \ge 1$.)

Case 2: $\#S \ge 2$. Then Step 9 uses M-time at most $\mu(\lg x^2 y)$ times

$$\begin{aligned}3k+3+(6k+6)(\#T-1)+\left(4.5k^2+21.5k+15\right)\lg y\\ +\left((9k^2+44k+32)(n-1)+2.5k^2+21k-5\right)\lg \operatorname{prod} T\end{aligned}$$

by induction since $2^k \ge \lg \operatorname{prod} Q$. Similarly, Step 10 uses M-time at most $\mu(\lg x^2 y)$ times

$$\begin{aligned}3k+3+(6k+6)(\#(S-T)-1)+\left(4.5k^2+21.5k+15\right)\lg y\\ +\left((9k^2+44k+32)(n-1)+2.5k^2+21k-5\right)\lg \operatorname{prod}(S-T).\end{aligned}$$

Add. □

References

[1] Eric Bach, James Driscoll, Jeffrey Shallit, *Factor refinement*, in [19] (1990), 201–211; see also newer version [2]. URL: `http://cr.yp.to/bib/entries.html#1990/bach-cba`.

[2] Eric Bach, James Driscoll, Jeffrey Shallit, *Factor refinement*, Journal of Algorithms **15** (1993), 199–222; see also older version [1]. ISSN 0196–6774. MR 94m:11148. URL: `http://cr.yp.to/bib/entries.html#1993/bach-cba`.

[3] Eric Bach, Gary Miller, Jeffrey Shallit, *Sums of divisors, perfect numbers, and factoring*, in [15] (1984), 183–190; see also newer version [4].

[4] Eric Bach, Gary Miller, Jeffrey Shallit, *Sums of divisors, perfect numbers, and factoring*, SIAM Journal on Computing **15** (1986), 1143–1154; see also older version [3]. ISSN 0097–5397. MR 87k:11139. URL: `http://cr.yp.to/bib/entries.html#1986/bach`.

[5] Daniel J. Bernstein, *Fast ideal arithmetic via lazy localization*, in [12] (1996), 27–34. URL: `http://cr.yp.to/papers.html`.

[6] Daniel J. Bernstein, *Detecting perfect powers in essentially linear time*, Mathematics of Computation **67** (1998), 1253–1283. ISSN 0025–5718. MR 98j:11121. URL: `http://cr.yp.to/papers.html`.

[7] Daniel J. Bernstein, *Fast multiplication and its applications*, to appear in Buhler-Stevenhagen *Algorithmic number theory* book. URL: http://cr.yp.to/papers.html#multapps.

[8] Johannes A. Buchmann, Friedrich Eisenbrand, *On factor refinement in number fields*, Mathematics of Computation **68** (1999), 345–350. ISSN 0025–5718. MR 99e:11155. URL: http://www.ams.org/journal-getitem?pii=S0025571899010236.

[9] Johannes A. Buchmann, Hendrik W. Lenstra, Jr., *Approximating rings of integers in number fields*, Journal de Théorie des Nombres de Bordeaux **6** (1994), 221–260. ISSN 1246–7405. MR 96m:11092.

[10] Jacques Calmet (editor), *Algebraic algorithms and error-correcting codes 3*, Lecture Notes in Computer Science, 229, Springer-Verlag, Berlin, 1986. ISBN 0387167765.

[11] Bob F. Caviness (editor), *Proceedings of EUROCAL '85, volume 2*, Lecture Notes in Computer Science, 204, Springer-Verlag, Berlin, 1985. ISBN 3–540–15984–3. MR 87a:68007.

[12] Henri Cohen (editor), *Algorithmic number theory: second international symposium, ANTS-II, Talence, France, May 18–23, 1996, proceedings*, Lecture Notes in Computer Science, 1122, Springer-Verlag, Berlin, 1996. ISBN 3–540–61581–4. MR 97k:11001.

[13] George E. Collins, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition: preliminary report*, SIGSAM Bulletin **8** (1974), 80–90. ISSN 0163–5825.

[14] Jean Della Dora, Claire DiCrescenzo, Dominique Duval, *About a new method for computing in algebraic number fields*, in [11] (1985), 289–290. URL: http://cr.yp.to/bib/entries.html#1985/delladora.

[15] Richard A. DeMillo (editor), *Proceedings of the sixteenth annual ACM symposium on theory of computing. Held in Washington, D.C., April 30–May 2, 1984*, Association for Computing Machinery, New York, 1984. ISBN 0–89791–133–4. MR 87g:68005.

[16] Joachim von zur Gathen, *Representations and parallel computations for rational functions*, SIAM Journal on Computing **15** (1986), 432–452. ISSN 0097–5397. MR 88f:68055. URL: http://cr.yp.to/bib/entries.html#1986/vonzurgathen.

[17] Guoqiang Ge, *Recognizing units in number fields*, Mathematics of Computation **63** (1994), 377–387. ISSN 0025–5718. MR 94i:11107. URL: http://links.jstor.org/sici?sici=0025-5718(199407)63:207<377:RUINF>2.0.CO;2-8.

[18] Patrizia Gianni (editor), *Symbolic and algebraic computation: proceedings of the international symposium (ISSAC '88) held in Rome, July 4–8, 1988*, Lecture Notes in Computer Science, 358, Springer, Berlin, 1989. ISBN 3–540–51084–2. MR 90i:00005.

[19] David S. Johnson (editor), *Proceedings of the first annual ACM-SIAM symposium on discrete algorithms, January 22–24, 1990, San Francisco, California*, Society for Industrial and Applied Mathematics, Philadelphia, 1990. ISBN 0–89871–251–3.

[20] Erich Kaltofen, *Sparse Hensel lifting*, in [11] (1985), 4–17. MR 88b:12001. URL: http://cr.yp.to/bib/entries.html#1985/kaltofen.

[21] Erich Kaltofen, Victor Shoup, *Subquadratic-time factoring of polynomials over finite fields*, Mathematics of Computation **67** (1998), 1179–1197. ISSN 0025–5718. MR 99m:68097. URL: http://www.ams.org/journal-getitem?pii=S0025571898009442.

[22] William M. Kantor, Ákos Seress, *Prime power graphs for groups of Lie type*, Journal of Algebra **247** (2002), 370–434. MR 2003a:20026. URL: `http://dx.doi.org/10.1006/jabr.2001.9016`.

[23] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 1st edition, 1st printing, Addison-Wesley, Reading, 1969; see also newer version [24]. MR 44:3531.

[24] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 1st edition, 2nd printing, Addison-Wesley, Reading, 1971; see also older version [23]; see also newer version [25]. MR 44:3531.

[25] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 2nd edition, Addison-Wesley, Reading, 1981; see also older version [24]; see also newer version [26]. ISBN 0–201–03822–6. MR 83i:68003.

[26] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 3rd edition, Addison-Wesley, Reading, 1997; see also older version [25]. ISBN 0–201–89684–2.

[27] Heinz Lüneburg, *On a little but useful algorithm*, in [10] (1986), 296–301. URL: `http://cr.yp.to/bib/entries.html#1986/lueneburg`.

[28] Michael Pohst, Hans Zassenhaus, *Algorithmic algebraic number theory*, Cambridge University Press, Cambridge, 1989. ISBN 0–521–33060–2. MR 92b:11074.

[29] Trevor J. Smedley, *Detecting algebraic dependencies between unnested radicals: extended abstract*, in [34] (1990), 292–293. URL: `http://cr.yp.to/bib/entries.html#1990/smedley`.

[30] Thomas Jan Stieltjes, *Sur la théorie des nombres*, Annales de la Faculté des Sciences de Toulouse **4** (1890), 1–103. ISSN 0240–2963.

[31] Arne Storjohann, *Computing Hermite and Smith normal forms of triangular integer matrices*, Linear Algebra and its Applications **282** (1998), 25–45. ISSN 0024–3795. MR 99j:65074. URL: `http://dx.doi.org/10.1016/S0024-3795(98)10012-5`.

[32] Jeremy T. Teitelbaum, *On the computational complexity of the resolution of plane curve singularities*, in [18] (1989), 285–292; see also newer version [33]. MR 91g:14061. URL: `http://cr.yp.to/bib/entries.html#1989/teitelbaum`.

[33] Jeremy T. Teitelbaum, *On the computational complexity of the resolution of plane curve singularities*, Mathematics of Computation **54** (1990), 797–837; see also older version [32]. ISSN 0025–5718. MR 91g:14061. URL: `http://links.jstor.org/sici?sici=0025-5718(199004)54:190<797:TCCOTR>2.0.CO;2-R`.

[34] Shunro Watanabe, Morio Nagata (editors), *Proceedings of the international symposium on symbolic and algebraic computation ’90*, Association for Computing Machinery, New York, 1990. ISBN 0–201–54892–5.