Journal of Algorithms, to appear. Final revisions are in progress.

FACTORING INTO COPRIMES IN ESSENTIALLY LINEAR TIME

DANIEL J. BERNSTEIN

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 apply to any free commutative monoid with fast algorithms for multiplication, division, and greatest common divisors; e.g., monic polynomials over a field. They can be used as a substitute for prime factorization in many applications.

1. INTRODUCTION

It is not easy to factor 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]; see section 4 of this paper.

Similar comments apply to monic polynomials over a finite field. Here the Bach-Driscoll-Shallit algorithm has been superseded, at least in theory, by an algorithm of Kaltofen and Shoup in [16], which factors polynomials into primes in subquadratic expected time. I can factor a set of polynomials into coprimes in essentially linear time.

Applications. Coprimes are often an adequate substitute for primes. The following example is due to Hendrik W. Lenstra, Jr.: given positive integers a, b, c, one can compute a basis for the $\mathbf{Z}$-module $\{(i, j, k) \in \mathbf{Z}^3 : a^i b^j c^k = 1\}$ by first factoring a, b, c into coprimes and then doing linear algebra on the exponents in the factorizations. See [14] for a generalization.

Other applications, listed here in historical order: eliminating quantifiers (see [11]), converting any set of congruences to a set with coprime moduli (see [13, Remark 6.8] and [2, page 201]), finding the prime factors of n given the sum of n's divisors (see [3]), computing in zero-dimensional rings by lazy localization (see [12] and [23, section 3.1]), lifting polynomial factorizations (see [15, section 3]), computing normal bases (see [18]), finding the maximal order in a number field by lazy factorization (see [19, section 4.6]), finding algebraic dependencies among

Date: 19980108.
1991 *Mathematics Subject Classification.* Primary 11Y16.
The author was supported by the National Science Foundation under grant DMS–9600083.

radicals (see [21]), and performing arithmetic on ideals in a number field by lazy localization (see [5]).

Acknowledgments. Thanks to Igor Shparlinski for his comments.

Notation. $\{\}$ is the empty set. $\#S$ is the cardinality of S. $S - T$, when S and T are sets, means all elements of S that are not in T. When a proposition appears inside brackets, it means 1 if the proposition is true, 0 otherwise; for example, $[2 < 3] = 1$.

2. Outline of the paper

This paper is organized into four parts.

Part I. Existence. Section 4 proves that one can construct a "coprime base" for any finite set S via division and greatest common divisors. The construction is the Bach-Driscoll-Shallit algorithm.

Section 6 establishes the useful concept of a "quasiprime." Section 7 shows that there is only one coprime base that can be obtained from S via multiplication, division, and greatest common divisors: the "natural coprime base" for S, written $\operatorname{cb} S$.

Rather than state every result twice, once for integers and once for polynomials, I have abstracted the algebraic properties that make the constructions work. The most general setting for the Bach-Driscoll-Shallit algorithm is a "Noetherian coid with cancellation," defined in section 3. The most general setting for natural coprime bases, and for my algorithms, is a "free coid," defined in section 5. Any free coid is Noetherian and has cancellation.

Readers not interested in maximum generality may skip sections 3 and 5. The semigroup of positive integers is a free coid; the semigroup of monic polynomials over a field is a free coid.

Part II. Two-element sets. Sections 10 through 13 give a series of constructions culminating in an algorithm, DCBA, to find the natural coprime base for any two-element set. Each construction takes essentially linear time, given essentially-linear-time subroutines for multiplication, division, and gcd, as discussed in section 8. Motivation for DCBA appears in section 9.

Part III. Finite sets. Sections 14 through 18 give several further essentially-linear-time constructions, culminating in an algorithm to find the natural coprime base for any finite set.

Part IV. Factorization. To factor a set S into coprimes, I 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

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 $1a = a = a1$.

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.

A coid H is **Noetherian** if any nonempty subset $S \subseteq H$ has a minimal element. (**Minimal** means not divisible by any other elements of S; **maximal** means not dividing any other elements of S; **maximum** or **greatest** means divisible by all elements of S.)

In a Noetherian coid H one may apply **Noetherian induction**, which states that if T is a subset of H, and if T contains a whenever T contains all other divisors of a, then $T = H$. Indeed, $H - T$ has no minimal element, so it must be empty.

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.

Example. The set of nonsingular ideals in a Noetherian domain, under ideal multiplication, is a Noetherian coid with cancellation.

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. □

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.

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. □

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

Under my definition of "Noetherian," any Noetherian coid is **combinatorial**: $a = b$ whenever a and b are associates (i.e., whenever a divides b and b divides a). One can systematically replace "equal" by "associate" and "minimal" by "minimal up to associates" and so on, 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.

"Coprime" is also known as "relatively prime."

4. Coprime bases

A set P is **coprime** if each element of P is coprime to every other element of P. If P and Q are coprime sets, and each $p \in P$ is coprime to each $q \in Q$, then $P \cup Q$ is a coprime set.

Let P and S be subsets of a Noetherian coid with cancellation. I call P a **base for** S if S is contained in the coid generated by P. If P is a base for S then P is a base for any subset of S.

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

Theorem 4.1. *Let a, b be elements of a Noetherian coid with cancellation. Then there is a finite set P with an element d such that (1) P is a coprime base for $\{a, b\}$; (2) d divides b; (3) if $p \in P - \{d\}$ then p divides a.*

Proof. Apply Noetherian induction to ab—i.e., to the set of c such that the desired conclusion holds whenever $ab = c$.

Let g be a maximal common divisor of $\{a, b\}$. If $g = 1$ then the set $\{a, b\}$ with element b satisfies the stated conditions. So assume that $g \neq 1$; this implies that $a \neq 1$ and $b \neq 1$.

Consider $(a/g)g$, a divisor of ab; it is different from ab since $b \neq 1$. By induction, there is a finite coprime base P for $\{a/g, g\}$, with an element d dividing g, and with all other elements dividing a/g.

Next consider $d(b/g)$, a divisor of ab; it is different from ab since $a \neq 1$. By induction, there is a finite coprime base Q for $\{d, b/g\}$, with an element e dividing b/g, and with all other elements dividing d.

Finally, consider $R = (P - \{d\}) \cup Q$, with element e. (1) If $p \in P - \{d\}$ and $q \in Q$ then p and q are coprime. (Indeed, if $q \neq e$ then q divides d, so q is coprime to p, since P is a coprime set. If $q = e$ then q divides b/g, but p divides a/g, so p and q are coprime by Theorem 3.2.) Hence R is a coprime set. (2) Since e divides b/g it also divides b. (3) Consider $q \in R - \{e\}$. If $q \in P - \{d\}$ then q divides a/g, hence a. If $q \in Q$ then q divides d, hence g, hence a. □

Theorem 4.2. *Let H be a Noetherian coid with cancellation. Let B be a finite coprime subset of H, and let a be an element of H. Then there is a finite set E such that (1) E is a coprime base for $B \cup \{a\}$; (2) each element of E divides an element of $B \cup \{a\}$.*

Proof. Induct on the size of B. If $B = \{\}$ then $\{a\}$ satisfies (1) and (2). Otherwise pick $b \in B$. By Theorem 4.1, there is a finite coprime base P for $\{a, b\}$, with an element d dividing a, and with all other elements dividing b. Next, by induction, there is a finite set Q such that Q is a coprime base for $(B - \{b\}) \cup \{d\}$, with every element of Q dividing an element of $(B - \{b\}) \cup \{d\}$.

Finally, consider $E = (P - \{d\}) \cup Q$. (1) If $p \in P - \{d\}$ and $q \in Q$ then p and q are coprime. (Indeed, if q divides d then p and q are coprime since P is a coprime set. If q does not divide d then it divides an element of $B - \{b\}$, but p divides b, so p and q are coprime since B is a coprime set.) Thus E is a coprime set. Furthermore, the coid generated by E contains d (via Q), hence all of P, hence a and b, hence all of B. So E is a coprime base for $B \cup \{a\}$. (2) Consider $q \in E$. If $q \in Q$ then q divides d, which divides a, or q divides an element of $B - \{b\}$. If $q \in P - \{d\}$ then q divides b. Hence in any case q divides an element of $B \cup \{a\}$. □

Theorem 4.3. *Let S be a finite subset of a Noetherian coid with cancellation. Then there is a finite coprime base for S.*

Proof. Induct on the size of S. If $S = \{\}$ then $\{\}$ is a coprime base for S. Otherwise select $a \in S$. By induction, there is a finite coprime base B for $S - \{a\}$. By Theorem 4.2, there is a finite coprime base for $B \cup \{a\}$, and hence for S. □

Notes. The proof of Theorem 4.1 is an algorithm introduced by Bach, Driscoll, and Shallit in [2]. This algorithm is neither the first algorithm for constructing coprime bases (see [11]) nor the fastest (see section 9), but it is certainly the simplest. I call it the **coprime base algorithm** (CBA). Here is a C program that reads two small positive integers from its input and prints the CBA results:

```
gcd(a,b) { return b ? gcd(b,a%b) : a; }
cba(a,b) { int g = gcd(a,b);
  if (g == 1) { printf("%d ",a); return b; }
  return cba(cba(a/g,g),b/g); }
main() { int a, b; scanf("%d %d",&a,&b);
  printf("%d\n",cba(a,b)); }
```

The proof of Theorem 4.2 is also an algorithm in [2]; I call it the **coprime base extension algorithm** (ECBA).

It is stated in [2] that Gaussian semigroups form a "suitable abstract setting" for these algorithms. ("Gaussian semigroup," modulo associates, means free coid; see section 5.) Noetherian coids with cancellation are substantially more general.

Some authors say "P is pairwise coprime" instead of "P is coprime," reserving "P is coprime" for the relatively unimportant concept that $\gcd P = 1$.

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.

Example. The nonzero ideals in a Dedekind domain form a free coid. Indeed, in a Dedekind domain, the sum of two ideals a and b is a common divisor of $\{a,b\}$, hence a maximal common divisor; if $a+b=1$ and $a+c=1$ then $1=a+b(a+c)=a+ba+bc=a+bc$ by elementary ideal arithmetic.

Theorem 5.1. *In a free coid, if a divides bc, with a coprime to b, then a divides c.*

Proof. Let g be a maximal common divisor of $\{a,c\}$. By Theorem 3.2, a/g is coprime to c/g. By hypothesis a/g is coprime to b; so 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. □

Theorem 5.2. *In 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. Write $z=qy$. Then x/h divides $z/h=q(y/h)$. By Theorem 3.2, x/h and y/h are coprime. By Theorem 5.1, x/h divides q. Hence $xy/h=(x/h)y$ divides $qy=z$. □

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. Let g be a maximal common divisor of B, and let d be any common divisor of B. 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. □

Notes. Theorem 5.1, in the case of the positive integers, is the **fundamental theorem of arithmetic**. It implies that factorizations into primes are unique; see Theorem 6.4. The reader may enjoy showing that every element of a Noetherian coid with cancellation has a factorization into irreducibles, and that irreducibles in free coids are primes; hence every element of a free coid has a unique factorization into primes. Thus a free coid is free in the usual sense. The converse is also true.

6. Quasiprimes and the ord function

Let p and a be elements of a free coid, with $p \neq 1$. If a/p^m is coprime to p for some integer $m \geq 0$ then p is a **quasiprime for** a. Observe that m is determined by p and a; it is denoted $\operatorname{ord}_p a$.

If p is a quasiprime for every element of a set S then p is a **quasiprime for** S.

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

Proof. By assumption $a = up^m$ and $b = vp^n$ where p is coprime to u and v. Then p is coprime to uv, so p is a quasiprime for $ab = uvp^{m+n}$, with $\operatorname{ord}_p ab = m + n = \operatorname{ord}_p a + \operatorname{ord}_p b$. □

Theorem 6.2. *Let a, b be elements of a free coid. If p is a quasiprime for $\{a, ab\}$ then p is a quasiprime for b.*

Proof. By assumption $a = up^m$ and $ab = vp^n$ where p is coprime to u and v. Then p^m is coprime to v, and p^m divides vp^n, so p^m divides p^n, so $m \leq n$. Next u divides vp^{n-m}, and u is coprime to p^{n-m}, so u divides v. Finally $b = (v/u)p^{n-m}$. □

Theorem 6.3. *Let a, b be elements of a free coid. Define $g = \gcd\{a, b\}$. If p is a quasiprime for $\{a, b\}$ then p is a quasiprime for g with $\operatorname{ord}_p g = \min\{\operatorname{ord}_p a, \operatorname{ord}_p b\}$.*

Proof. By assumption $a = up^m$ and $b = vp^n$ where p is coprime to u and v. Without loss of generality assume $m \leq n$. Since p^m is a common divisor of a and b, it divides g; write $g = xp^m$. Then x divides u, so p is coprime to x. □

Theorem 6.4. *Let P be a finite coprime subset of a free coid. Define $a = \prod_{p \in P} p^{a_p}$ where each a_p is a nonnegative integer. Fix $q \in P - \{1\}$. Then q is a quasiprime for a, and $\operatorname{ord}_q a = a_q$.*

Hence factorizations into elements of P are unique if P is a finite coprime set not containing 1: the only factorization of a is $\prod_{p \in P} p^{\operatorname{ord}_p a}$.

Proof. If $p \in P$, $p \neq q$, then p is coprime to q by assumption, so $\operatorname{ord}_q p = 0$. Also $\operatorname{ord}_q q = 1$. By Theorem 6.1, $\operatorname{ord}_q \prod_p p^{a_p} = \sum_p a_p \operatorname{ord}_q p = a_q$. □

Theorem 6.5. *Let a, b be elements of a free coid H. Let S be a finite subset of H. If $\operatorname{ord}_p a = \operatorname{ord}_p b$ whenever p is a quasiprime for S then $a = b$.*

Proof. By Theorem 4.3, there is a finite coprime base P for $S \cup \{a, b\}$. Write $a = \prod_{p \in P} p^{a_p}$ and $b = \prod_{p \in P} p^{b_p}$. If $p \in P - \{1\}$ then p is a quasiprime for S so $a_p = \operatorname{ord}_p a = \operatorname{ord}_p b = b_p$ by Theorem 6.4. Hence $a = b$. □

Theorem 6.6. *Let a, b be elements of a free coid H. Let S be a finite subset of H. If $\operatorname{ord}_p a \leq \operatorname{ord}_p b$ whenever p is a quasiprime for S then a divides b.*

Proof. Define $g = \gcd\{a, b\}$. If p is a quasiprime for S then $\operatorname{ord}_p a \le \operatorname{ord}_p b$ so $\operatorname{ord}_p g = \operatorname{ord}_p a$ by Theorem 6.3. Hence $g = a$ by Theorem 6.5. □

Theorem 6.7. *Let a, b be elements of a free coid H. Let S be a finite subset of H. If $\min\{\operatorname{ord}_p a, \operatorname{ord}_p b\} = 0$ whenever p is a quasiprime for S then a is coprime to b.*

Proof. Define $g = \gcd\{a, b\}$. If p is a quasiprime for S then $\min\{\operatorname{ord}_p a, \operatorname{ord}_p b\} = 0$ so $\operatorname{ord}_p g = 0 = \operatorname{ord}_p 1$ by Theorem 6.3. Hence $g = 1$ by Theorem 6.5. □

7. The natural coprime base

Fix a subset S of a free coid. The **natural coprime base for** S, denoted $\operatorname{cb} S$, is the set of maximal quasiprimes for S.

The coid generated by $\operatorname{cb} S$ is everything that can be constructed from S with multiplication, division, and greatest common divisors; see Theorem 7.5.

Theorem 7.1. *Let S be a subset of a free coid. Let P be a coprime base for S, not containing 1, such that any quasiprime for S is a quasiprime for P. Then $P = \operatorname{cb} S$.*

Proof. Take $p \in P$. By Theorem 6.4, p is a quasiprime for S. Say p divides another quasiprime d for S. Then d is a quasiprime for p by assumption; since d and p are not coprime, d must divide p, so $d = p$. Hence p is maximal.

Conversely, let q be a maximal quasiprime for S. Then q divides some element of P. (If not, then q must be coprime to every element of P, since q is a quasiprime for P. Thus q is coprime to every element of S. But then q^2 is a quasiprime for S, so q is not maximal.) Say q divides $p \in P$. By Theorem 6.4, p is a quasiprime for S, so by maximality $q = p$. □

Theorem 7.2. *Let S be a finite subset of a free coid. Then there is a finite coprime base P for S such that any quasiprime for S is a quasiprime for P.*

Proof. According to Theorem 6.2 and Theorem 6.3, if p is a quasiprime for S, then p is a quasiprime for anything that can be created from S with division and gcd. The constructions in Theorems 4.1, 4.2, and 4.3 use solely division and gcd. □

Theorem 7.3. *Let S be a finite subset of a free coid. Then $\operatorname{cb} S$ is a finite coprime base for S. Furthermore, any quasiprime for S is a quasiprime for $\operatorname{cb} S$.*

Proof. By Theorem 7.2, there is a finite coprime base P for S such that any quasiprime for S is a quasiprime for P. Define $Q = P - \{1\}$; then Q is a coprime base for S, not containing 1, and any quasiprime for S is a quasiprime for Q. By Theorem 7.1, $Q = \operatorname{cb} S$. □

Theorem 7.4. *Let Q be a coprime subset of a free coid. Then $\operatorname{cb} Q = Q - \{1\}$.*

Proof. $Q - \{1\}$ is a coprime base for Q not containing 1. Any quasiprime for Q is certainly a quasiprime for $Q - \{1\}$. By Theorem 7.1, $Q - \{1\} = \operatorname{cb} Q$. □

Theorem 7.5. *Let Q be a finite coprime subset of a free coid. Then Q is a base for $\{c\}$ if and only if every quasiprime for Q is a quasiprime for c.*

In particular, if Q is a base for $\{a, b\}$, then it is also a base for $\{ab, \gcd\{a, b\}\}$, since any quasiprime for Q is a quasiprime for $\{ab, \gcd\{a, b\}\}$ by Theorem 6.4, Theorem 6.1, and Theorem 6.3. Similarly, if Q is a base for $\{a, ab\}$ then it is also a base for $\{b\}$.

Proof. If Q is a base for $\{c\}$ then every quasiprime for Q is a quasiprime for c by Theorem 6.4.

For the converse, define $P = \operatorname{cb}(Q \cup \{c\})$. Any quasiprime for Q is a quasiprime for $Q \cup \{c\}$ by assumption, hence for P. So $P = \operatorname{cb} Q$ by Theorem 7.1. Thus $P = Q - \{1\}$ by Theorem 7.4. Since P is a base for $\{c\}$, Q is too. □

Theorem 7.6. *Let S be a finite subset of a free coid. Define $P = \operatorname{cb} S$. If every element of S divides a then $\prod_{p \in P} p$ divides a.*

Proof. Take $p \in P$. If p does not divide any element of S then p is coprime to all elements of S so p^2 is a quasiprime for S, contradicting the maximality of p. Thus p divides a. This is true for all p, so $\prod_{p \in P} p$ divides a by Theorem 5.2. □

Notes. My definition of the natural coprime base as the set of maximal quasiprimes is simpler than the characterizations in [15], [13, Remark 6.8], and [2, Theorem 3].

Part II. Two-element sets

8. Logarithms and M-time

Fix a free coid H. Fix a function $\lg : H \to \mathbf{R}$ such that (1) $\lg ab = \lg a + \lg b$ and (2) $\lg a \geq 1$ for any $a \neq 1$. Observe that $\lg 1 = 0$.

My algorithms use a black box that can multiply a and b, divide ab by b, check whether a divides b, or compute $\gcd\{a, b\}$ in time at most $(1 + \lg ab)\mu(\lg ab)$, where μ is a nondecreasing positive function. I write **M-time** for time spent inside this black box. I say that the black box supports **fast arithmetic** when $\mu(x) \in x^{o(1)}$.

Examples. For the coid of positive integers, one may take $\lg a$ as the usual logarithm base 2. For the coid of monic polynomials over a field, one may take $\lg a$ as the degree of a. In each case there is an algorithm that achieves $\log \mu(x) \in O(\log \log x)$.

Theorem 8.1. *Let p, a be elements of a free coid. If p is a quasiprime for a then $\operatorname{ord}_p a \leq \lg a$.*

Proof. Put $n = \operatorname{ord}_p a$. Then $a = up^n$ for some u, so $\lg a = \lg u + n \lg p \geq n$. □

Notes. See [20] for a fast gcd algorithm for integers. See [7] for a fast gcd algorithm for polynomials.

Subsequent sections use the above assumptions on lg and μ to bound the M-time used by various algorithms. One can attach a lg function to any free coid by setting $\lg p = 1$ for each prime p, and one can imagine a black box that performs each arithmetic operation in constant time. Conclusion: In any free coid, each algorithm terminates after finitely many arithmetic operations.

My M-time bounds are somewhat pessimistic in practice, for two reasons. First, one can actually divide ab by b in time depending only on the length of a. Second, multiplication, division, and divisibility testing are actually much faster than gcd. (This is also why I treat divisibility testing as a separate primitive, even though it could be implemented with gcd.)

9. From CBA to DCBA

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

Consider CBA, the algorithm described in section 4 for computing $\text{cb}\{a, b\}$. CBA replaces (a, b) with $(a/\gcd\{a, b\}, \gcd\{a, b\}, b/\gcd\{a, b\})$; it focuses on the left pair in this triple and then on the right pair. If $(a, b) = (p^e, p^f)$ then this triple is $(p^{e-f}, p^f, 1)$ if $e \ge f$ or $(1, p^e, p^{f-e})$ if $e \le f$. Observe that 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 $e + f$. Writing out the usual base-2 division algorithm inside Euclid's 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 $e < f$.

How DCBA operates on exponents. Define a function κ on pairs of nonnegative integers as follows:

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

Write κ^i for the ith iterate of κ. Define $\lambda(a, b)$ as the smallest $n \ge 0$ such that, for every quasiprime p for $\{a, b\}$, the second component of $\kappa^n(\text{ord}_p\, a, \text{ord}_p\, b)$ is 0.

Theorem 9.1. *If $(e', f') = \kappa(e, f)$ then $e' + \sqrt{2} f' \le (e + \sqrt{2} f)/\sqrt{2}$.*

Proof. If $e \le f$ then $\sqrt{2}e' + 2f' - e - \sqrt{2}f = (1 - \sqrt{2})e \le 0$. Otherwise $2^k f < e \le 2^{k+1} f$ for some $k \ge 0$, so $\sqrt{2}e' + 2f' - e - \sqrt{2}f = (\sqrt{2} - 1)e - (2^k\sqrt{2} + \sqrt{2} - 2)f \le 2^{k+1}(\sqrt{2} - 1)f - (2^k\sqrt{2} + \sqrt{2} - 2)f = (1 - 2^k)(2 - \sqrt{2})f \le 0$. □

Theorem 9.2. *If $n \ge 0$ and $\lg a + \sqrt{2} \lg b < \sqrt{2}^{n+1}$ then $\lambda(a, b) \le n$.*

Proof. Let p be a quasiprime for $\{a, b\}$. Define $e = \text{ord}_p\, a$, $f = \text{ord}_p\, b$, and $(e', f') = \kappa^n(e, f)$. Then $e' + \sqrt{2} f' \le (e + \sqrt{2} f)/\sqrt{2}^n < \sqrt{2}$ by Theorem 9.1, so $f' = 0$. □

Theorem 9.3. *If $\lambda(a, b) = 0$ then $b = 1$.*

Proof. If p is a quasiprime for $\{a, b\}$ then $\text{ord}_p\, b = 0$ by definition of λ. By Theorem 6.5, $b = 1$. □

Notes. See [17, sections 4.5.2 and 4.5.3] for a thorough discussion of Euclid's subtractive gcd algorithm and Euclid's repeated-division gcd algorithm.

10. Computing powers

Algorithm T. 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 10.1. *Algorithm T computes a^{2^n} in M-time at most*

$$(n + 2(2^n - 1) \lg a)\mu(2^n \lg a).$$

Proof. For $n = 0$, Algorithm T uses no M-time, and $n + 2(2^n - 1)\lg a = 0$. For $n \geq 1$, Algorithm T first computes a^2, using M-time at most $(1 + 2\lg a)\mu(2\lg a) \leq (1 + 2\lg a)\mu(2^n \lg a)$. 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 $1 + 2\lg a + n - 1 + 2(2^n - 2)\lg a = n + 2(2^n - 1)\lg a$. □

Notes. Algorithm T is well known. For generalizations see [17, section 4.6.3].

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

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

whenever p is a quasiprime for both a and b. Constructive proofs of existence appear in Theorem 11.2 and Theorem 11.3 below. Uniqueness follows from Theorem 6.5.

Theorem 11.1. *Let a, c, x_0, y_0 be elements of a free coid, with $x_0 y_0 = a$ and $x_0 = \gcd\{a, c\}$. For $n \geq 0$ define $g_{n+1} = \gcd\{x_n, y_n\}$, $x_{n+1} = x_n g_{n+1}$, and $y_{n+1} = y_n / g_{n+1}$. If p is a quasiprime for $\{a, c\}$ then $\operatorname{ord}_p x_n = \min\{\operatorname{ord}_p a, 2^n \operatorname{ord}_p c\}$.*

Note that $x_n y_n = a$.

Proof. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p c$. Then $\operatorname{ord}_p x_0 = \min\{e, f\}$.

Assume inductively that $\operatorname{ord}_p x_n = \min\{e, 2^n f\}$. If $e < 2^n f$ then $\operatorname{ord}_p x_n = e$ so $\operatorname{ord}_p y_n = 0$ so $\operatorname{ord}_p g_{n+1} = \min\{0, e\} = 0$; thus $\operatorname{ord}_p x_{n+1} = e = \min\{e, 2^{n+1} f\}$. If $e \geq 2^n f$ then $\operatorname{ord}_p x_n = 2^n f$ so $\operatorname{ord}_p y_n = e - 2^n f$ so $\operatorname{ord}_p g_{n+1} = \min\{e - 2^n f, 2^n f\}$; thus $\operatorname{ord}_p x_{n+1} = 2^n f + \min\{e - 2^n f, 2^n f\} = \min\{e, 2^{n+1} f\}$. □

Theorem 11.2. *Let a, c be elements of a free coid. Define $x_0 = \gcd\{a, c\}$ and $y_0 = a/x_0$. For $n \geq 0$ define $g_{n+1} = \gcd\{x_n, y_n\}$, $x_{n+1} = x_n g_{n+1}$, and $y_{n+1} = y_n / g_{n+1}$. If $g_{n+1} = 1$ then $x_n = \operatorname{ppi}(a, c)$ and $y_n = \operatorname{ppo}(a, c)$.*

Proof. Select a quasiprime p for $\{a, c\}$; write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p c$. If $g_{n+1} = 1$ then $x_n = x_{n+1}$ so $\min\{e, 2^n f\} = \min\{e, 2^{n+1} f\}$ by Theorem 11.1. 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 $e \leq 2^n f$; thus $\operatorname{ord}_p x_n = e$. Hence in general $\operatorname{ord}_p x_n = e[f > 0]$ and $\operatorname{ord}_p y_n = e - \operatorname{ord}_p x_n = e[f = 0]$. □

Theorem 11.3. *Let a, b be elements of a free coid. Define $y_0 = \gcd\{a, b\}$ and $x_0 = a/y_0$. For $n \geq 0$ define $g_{n+1} = \gcd\{x_n, y_n\}$, $x_{n+1} = x_n g_{n+1}$, and $y_{n+1} = y_n / g_{n+1}$. If $g_{n+1} = 1$ then $x_n = \operatorname{ppg}(a, b)$ and $y_n = \operatorname{pple}(a, b)$.*

Proof. Define $c = x_0$. Then $x_n = \operatorname{ppi}(a, c)$ and $y_n = \operatorname{ppo}(a, c)$ by Theorem 11.2. Select a quasiprime p for $\{a, b\}$; write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$. Then $\operatorname{ord}_p y_0 = \min\{e, f\}$, so $\operatorname{ord}_p c$ is 0 exactly when $e \leq f$. Thus $\operatorname{ord}_p x_n = e[\operatorname{ord}_p c > 0] = e[e > f]$ and $\operatorname{ord}_p y_n = e[\operatorname{ord}_p c = 0] = e[e \leq f]$. □

Theorem 11.4. *In the situation of Theorem 11.1, if $k \geq 0$ and $2^k \geq \lg a$ then $g_{k+1} = 1$.*

Proof. Say p is a quasiprime for $\{a, c\}$. By Theorem 8.1, $\operatorname{ord}_p a \le \lg a$. By Theorem 11.1, $\operatorname{ord}_p x_k = \min\{\operatorname{ord}_p a, 2^k \operatorname{ord}_p c\}$. If $\operatorname{ord}_p c = 0$ then $\operatorname{ord}_p x_k = 0 = \operatorname{ord}_p x_{k+1}$. If $\operatorname{ord}_p c > 0$ then $2^k \operatorname{ord}_p c \ge 2^k \ge \lg a \ge \operatorname{ord}_p a$ so $\operatorname{ord}_p x_k = \operatorname{ord}_p a = \operatorname{ord}_p x_{k+1}$. Hence $\operatorname{ord}_p g_{k+1} = \operatorname{ord}_p x_{k+1} - \operatorname{ord}_p x_k = 0$. This is true for all p, so $g_{k+1} = 1$. □

The following two algorithms include n solely for expository purposes.

Algorithm PPIO. 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 11.2.) 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.
```

Algorithm PPGLE. 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 11.3.) 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. *Let a, c be elements of a free coid. Algorithm PPIO computes $(\gcd, \operatorname{ppi}, \operatorname{ppo})(a, c)$ 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 11.4, $g_{k+1} = 1$, 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 $(k + 1)(1 + \lg a)\mu(\lg a)$.

An 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 $(2k + \lg x_k + (k - 1)\lg a + \lg y_0)\mu(\lg a)$, which is at most $(2k + (k + 1)\lg a)\mu(\lg a)$.

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

Theorem 11.6. *Let a, b be elements of a free coid. Algorithm PPGLE computes $(\gcd, \operatorname{ppg}, \operatorname{pple})(a, b)$ 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. □

Notes. The ppo function is the subject of [18]. Write $r = \operatorname{ppo}(a, b)$. Then r is a divisor of a, coprime to b, such that each quasiprime for $\{a, b\}$ dividing a/r also divides b. Lüneburg's algorithm in [18] takes quadratic time on inputs (p^n, p).

The ppg function is the subject of [22, section 19]. Write $A = \operatorname{ppg}(a, b)$ and $B = b/\gcd\{A, b\}$. Then A divides a, and B divides b; A and B are coprime; and AB is the least common multiple of a and b. Stieltjes's algorithm in [22] takes quadratic time on inputs (p^{n+1}, p^n).

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."

12. The ls function

Fix a, b in a free coid. Define $\operatorname{ls}_n(a,b)$ as the triple (g_n, h_n, c_n), where $(g_0, h_0, c_0) = (\gcd, \operatorname{ppg}, \operatorname{pple})(\operatorname{ppi}(a,b), b)$ and $(g_{n+1}, h_{n+1}, c_{n+1}) = (\gcd, \operatorname{ppg}, \operatorname{pple})(h_n, g_n^2)$.

Theorem 12.1. *Let a, b be elements of a free coid. Let p be a quasiprime for $\{a, b\}$. Define $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$. 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 $\operatorname{ord}_p h_n$. Therefore 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 $\operatorname{ord}_p g_n = [2^{n-1}f < e]\min\{e, 2^n f\}$ 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 $a = c_0 c_1 \dots c_m \operatorname{ppo}(a,b)$. Furthermore, any two members of the sequence $c_0, c_1, \dots, c_m, \operatorname{ppo}(a,b)$ are coprime.*

Proof. Let p be a quasiprime for $\{a, b\}$. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$. I will show that the sum of the numbers $\operatorname{ord}_p c_0, \operatorname{ord}_p c_1, \dots, \operatorname{ord}_p c_m, \operatorname{ord}_p \operatorname{ppo}(a,b)$ 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 $\operatorname{ord}_p \operatorname{ppo}(a,b) = 0$; $\operatorname{ord}_p c_0 = 0$; and $\operatorname{ord}_p c_n = e[2^{n-1}f < e \le 2^n f]$ for $n \ge 1$. There is exactly one value of $k \ge 1$ for which $2^{k-1}f < e \le 2^k f$. Furthermore $\operatorname{ord}_p h_m = 0$ 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. Furthermore, if $m \ge 1$ and $h_m = 1$, then $c_0 d_1 d_2 \dots d_m$ divides b, and $b/d_1 d_2 \dots d_m$ is coprime to a/c_0.*

Proof. Let p be a quasiprime for $\{a, b\}$. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$. Then $\operatorname{ord}_p c_n = e[2^{n-1}f < e \le 2^n f]$ by Theorem 12.1, 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, $\operatorname{ord}_p d_1 d_2 \dots d_m = f[f < e \le 2^m f]$, and $\operatorname{ord}_p c_0 = e[e \le f] \le f[e \le f]$, so $\operatorname{ord}_p c_0 d_1 d_2 \dots d_m \le f[e \le 2^m f] \le f$. Thus $c_0 d_1 d_2 \dots d_m$ divides b.

If $h_m = 1$ then $f = 0$ or $e \le 2^m f$. Either way $\operatorname{ord}_p d_1 d_2 \dots d_m = f[f < e]$, so $\operatorname{ord}_p(b/d_1 d_2 \dots d_m) = f[e \le f]$. On the other hand $\operatorname{ord}_p(a/c_0) = e[e > f]$. Thus $b/d_1 d_2 \dots d_m$ and a/c_0 are coprime. □

Notes. The name ls stands for "left shift." This function splits up the quasiprimes in a according to the cases in the definition of κ in section 9.

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

This section introduces a fast algorithm, DCBA, to compute the natural coprime base for $\{a, b\}$. For the correctness of DCBA, see Theorem 13.1. For speed, see Theorem 13.3.

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 $n \geq 1$. Assume that $h_m = 1$ with $m \geq 1$. Define $P_n = \operatorname{cb}\{c_n/d_n^{2^{n-1}}, d_n\}$ for $1 \leq n \leq m$. Define $Q = \operatorname{cb}\{b/c_0d_1d_2\ldots d_m, c_0\}$. 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 $k \neq n$. 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 $b/d_1d_2\ldots d_m$, hence is coprime to $a/c_0 = c_1c_2\ldots c_m \operatorname{ppo}(a, b)$ by Theorem 12.3 and 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 $b/c_0d_1d_2\ldots d_m$ (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), as well as $\operatorname{ppo}(a, b)$ (via R), so it contains $c_0c_1\ldots c_m \operatorname{ppo}(a, b) = a$ by Theorem 12.2.

Finally, naturalness follows from Theorem 7.1. □

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

```
1. If $b = 1$: Print $a$ if $a \neq 1$. Stop.
2. Compute $(a, r) \leftarrow (\operatorname{ppi}, \operatorname{ppo})(a, b)$ by Algorithm PPIO.
3. Print $r$ if $r \neq 1$.
4. Compute $(g, h, c) \leftarrow (\gcd, \operatorname{ppg}, \operatorname{pple})(a, b)$ by Algorithm PPGLE.
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 PPGLE.
8. Set $d \leftarrow \gcd\{c, b\}$.
9. (Now $(g, h, c, d) = (g_n, h_n, c_n, d_n)$.) Set $x \leftarrow xd$.
10. Compute $y \leftarrow d^{2^{n-1}}$ by Algorithm T.
11. Recursively apply DCBA to $(c/y, d)$.
12. If $h \neq 1$: Set $n \leftarrow n + 1$. Return to step 7.
13. Recursively apply DCBA to $(b/x, c_0)$.
```

Beware that the order of arguments is important in DCBA's recursive calls.

Theorem 13.2. *Let a, b be elements of a free coid. Define $(g_n, h_n, c_n) = \operatorname{ls}_n(a, b)$. If $m \geq 1$ and $2^{m-1} \geq \lg a$ then $h_m = 1$.*

Proof. Say p is a quasiprime for $\{a, b\}$. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$. Then $e \leq \lg a \leq 2^{m-1}$ by Theorem 8.1. If $f = 0$ then $\operatorname{ord}_p h_m = 0$; if $f \geq 1$ then $e \leq 2^{m-1}f$ so again $\operatorname{ord}_p h_m = 0$. □

Theorem 13.3. *Let a, b be elements of a free coid. If $m \geq 1$ and $2^{m-1} \geq \lg ab$ then DCBA finishes in M-time at most $\lambda(a, b)(4m^2 + 12m + 4)(\lg ab)\mu(3 \lg ab)$.*

Proof. If $b = 1$ then DCBA stops immediately, using no M-time. Otherwise $\lambda(a,b) \ge 1$ by Theorem 9.3.

The point here is that $\lambda(a,b)$ decreases at each level of the recursion: $\lambda(c/y,d) \le \lambda(a,b) - 1$ in step 11, and $\lambda(b/x,c_0) \le \lambda(a,b) - 1$ in step 13. Indeed, let p be a quasiprime for $\{a,b\}$. Write $e = \operatorname{ord}_p a$ and $f = \operatorname{ord}_p b$. Then the pair $(\operatorname{ord}_p(c_n/d_n^{2^{n-1}}), \operatorname{ord}_p d_n)$ is $(e - 2^{n-1}f, f) = \kappa(e,f)$ if $2^{n-1}f < e \le 2^n f$, $(0,0)$ otherwise. The pair $(\operatorname{ord}_p(b/c_0d_1d_2\dots d_m), \operatorname{ord}_p c_0)$ is $(f-e,e) = \kappa(e,f)$ if $e \le f$, $(0,0)$ otherwise.

So induct on $\lambda(a,b)$. The product of the inputs to DCBA in step 11 and step 13 is $b\prod_{n\ge1}(c_n/d_n^{2^{n-1}})$, which divides ab by Theorem 12.2. By induction, 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 DCBA takes M-time at most $(4m^2+12m+4)(\lg ab)\mu(3\lg ab)$.

Step 2 and step 4 each use M-time at most $(3m+(2m+2)\lg a + \lg b)\mu(\lg ab)$ by Theorem 11.5 and Theorem 11.6 respectively. The division in step 13 uses M-time at most $(1+\lg b)\mu(\lg b)$.

Steps 7 through 12 are performed for $n \in \{1,2,\dots,m\}$ at most, since $h_m = 1$ by Theorem 13.2. Step 7 uses M-time at most $(3m+(2m+2)\lg h + \lg g^2)\mu(\lg hg^2) \le (3m+(2m+4)\lg a)\mu(\lg a^3)$ per iteration by Theorem 11.5 since h and g divide a.

Step 8 uses M-time at most $(1+\lg ab)\mu(\lg ab)$ per iteration since c divides a.

Step 9 uses M-time at most $(1+\lg b)\mu(\lg b)$ per iteration since the final value of x divides b.

By Theorem 10.1, step 10 uses M-time at most $(n-1+(2^n-2)\lg d)\mu(2^{n-1}\lg d) \le (m-1+2\lg a)\mu(\lg a)$ per iteration.

Finally, the division in step 11 uses M-time at most $(1+\lg a)\mu(\lg a)$ per iteration.

The total is at most $\mu(\lg a^3b)$ times $4m^2+8m+1+(2m^2+12m+4)\lg a + (2m+3)\lg b$, which is at most $(2m^2+12m+4)\lg a + (4m^2+10m+4)\lg b \le (4m^2+12m+4)\lg ab$ since $\lg b \ge 1$. □

Theorem 13.4. *Let a,b be elements of a free coid. If $m \ge 1$ and $2^{m-1} \ge \lg ab$ then DCBA finishes in M-time at most $8m(m^2+3m+1)(\lg ab)\mu(3\lg ab)$.*

Proof. By Theorem 9.2, $\lambda(a,b) \le 2m$. □

Notes. Although DCBA is much faster than CBA in the worst case, CBA is faster on average for many input distributions. In practice one should start with CBA, switching to DCBA after a few steps.

Part III. Finite sets

14. The prod function

For any finite set S define $\operatorname{prod} S = \prod_{a\in S} a$. The following algorithm computes $\operatorname{prod} S$ by **binary splitting**:

Algorithm P. 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 P recursively.
5. Compute $Y \leftarrow \operatorname{prod}(S-T)$ by Algorithm P recursively.
6. Print $XY$.
```

Theorem 14.1. *Let S be a finite subset of a free coid. If $2^m \ge \#S \ge 1$ then Algorithm P computes* $\operatorname{prod} S$ *in M-time at most* $(\#S - 1 + m \lg \operatorname{prod} S)\mu(\lg \operatorname{prod} S)$.

Proof. Induct on m.

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

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

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

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

Notes. Algorithm P is well known; it appears in, e.g., [6, exercise 6.4–10(a)].

15. The split function

For any coprime set P define $\operatorname{split}(a, P) = \{(p, \operatorname{ppi}(a, p)) : p \in P\}$. Algorithm S below computes $\operatorname{split}(a, P)$.

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. Let q be a quasiprime for $P \cup \{b\}$. I will show that $\operatorname{ord}_q b = \operatorname{ord}_q \operatorname{ppo}(b, x) + \sum_{p \in P} \operatorname{ord}_q \operatorname{ppi}(b, p)$ where $x = \operatorname{prod} P$.

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 assumption, 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 $\operatorname{ord}_q \operatorname{ppo}(b, x) = \operatorname{ord}_q b$. □

Theorem 15.2. *Let a, x, y be elements of a free coid. Set $b = \operatorname{ppi}(a, xy)$. Then* $\operatorname{ppi}(b, x) = \operatorname{ppi}(a, x)$.

Proof. Let p be a quasiprime for $\{a, x, y\}$, and write $e = \operatorname{ord}_p a$. Then $\operatorname{ord}_p b = e[\operatorname{ord}_p xy > 0]$, so $\operatorname{ord}_p \operatorname{ppi}(b, x) = e[\operatorname{ord}_p xy > 0][\operatorname{ord}_p x > 0] = e[\operatorname{ord}_p x > 0] = \operatorname{ord}_p \operatorname{ppi}(a, x)$. □

Theorem 15.3. *Let a be an element of a free coid H. Let P be a finite subset of H. Define $b = \operatorname{ppi}(a, \operatorname{prod} P)$. Then* $\operatorname{split}(a, P) = \operatorname{split}(b, P)$.

Proof. If $p \in P$ then p divides $\operatorname{prod} P$ so $\operatorname{ppi}(b, p) = \operatorname{ppi}(a, p)$ by Theorem 15.2. □

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

```
1. If $P = \{\}$: Stop.
2. Compute $b \leftarrow \operatorname{ppi}(a, \operatorname{prod} P)$.
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 S recursively.
6. Print $\operatorname{split}(b, P - Q)$ by Algorithm S recursively.
```

Theorem 15.4. *Let a be an element of a free coid H. Let P be a finite coprime subset of H. Define $x = \operatorname{prod} P$ and $b = \operatorname{ppi}(a, x)$. Then Algorithm S computes* $\operatorname{split}(a, P)$ *in M-time at most*

$$(2k+4)(\lg a + 2m \lg b) + \frac{(m+1)(m+2)}{2} \lg x + (m+5+6k)\#P - 3k - 2$$

times $\mu(\lg ax)$ if $k \geq 0$, $2^k \geq \lg a$, and $2^m \geq \#P \geq 1$.

Proof. Induct on m. Write $y = \operatorname{prod} Q$ and $z = \operatorname{prod}(P - Q)$. Then y and z are coprime since P is coprime, so $\lg \operatorname{ppi}(b, y) + \lg \operatorname{ppi}(b, z) \leq \lg b$ by Theorem 15.1.

Algorithm S spends M-time at most $(\#P - 1 + m \lg x)\mu(\lg x)$ computing $\operatorname{prod} P$ by Theorem 14.1, and M-time at most $(3k+3+(2k+4)\lg a + \lg x)\mu(\lg ax)$ computing b by Theorem 11.5.

Case 1: $\#P = 1$. Then Algorithm S stops in step 3. The total M-time is at most $\mu(\lg ax)$ times

$$3k+3+(2k+4)\lg a + \lg x \leq (2k+4)(\lg a + 2m \lg b) + \frac{(m+1)(m+2)}{2} \lg x + m + 3 + 3k$$

as claimed.

Case 2: $\#P \geq 2$. Then Algorithm S performs two recursive calls. By induction the first uses M-time at most

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

times $\mu(\lg by)$ and the second uses M-time at most

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

times $\mu(\lg bz)$. Thus the total M-time is at most $\mu(\lg ax)$ times

$$\begin{aligned}&(\#P - 1 + m \lg x) + (3k+3+(2k+4)\lg a + \lg x)\\ &\quad + (2k+4)(2\lg b + 2(m-1)\lg b) + \frac{m(m+1)}{2} \lg x + (m+4+6k)\#P - 6k - 4,\end{aligned}$$

which equals the claimed bound. □

Notes. There is some redundancy in Algorithm S. Step 5 and step 6 recursively calculate products of various subsets of P; so does step 2. One could save time by storing products in step 2 for later use.

16. Extending a coprime base

Given a coprime set P, EDCBA below finds the natural coprime base for $P \cup \{b\}$.

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 and p' are coprime so $\operatorname{ppi}(b, p)$ and $\operatorname{ppi}(b, p')$ are coprime. 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 7.1. □

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

```
1. Compute $x \leftarrow \operatorname{prod} P$ by Algorithm P.
2. Compute $(a, r) \leftarrow (\operatorname{ppi}, \operatorname{ppo})(b, x)$ by Algorithm PPIO.
3. Print $r$ if $r \neq 1$.
4. Compute $S \leftarrow \operatorname{split}(a, P)$ by Algorithm S.
5. For each $(p, c) \in S$: Apply DCBA to $(p, c)$.
```

Theorem 16.2. *Let b be an element of a free coid H. Let P be a nonempty finite coprime subset of H. Define $x = \operatorname{prod} P$. EDCBA finishes in M-time at most $7m\#P + (8m^3 + 28m^2 + 16m + 4)\lg b + (8m^3 + 24.5m^2 + 10.5m + 2)\lg x$ times $\mu(3\lg bx)$ if $m \geq 1$ and $2^{m-1} \geq \lg bx$.*

A simpler bound when $1 \notin P$ is $(8m^3 + 28m^2 + 18m + 4)(\lg bx)\mu(3\lg bx)$.

Proof. Note that $\#P \leq 1 + \lg x \leq 1 + 2^{m-1} \leq 2^m$.

Step 1 takes M-time at most $(\#P - 1 + m\lg x)\mu(\lg x)$ by Theorem 14.1.

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

Step 4 takes M-time at most $(2m+2)(2m+1)\lg b + (1/2)(m+1)(m+2)\lg x + (7m-1)\#P - 3m + 1$ times $\mu(\lg bx)$ by Theorem 15.4.

By Theorem 13.4, the application of DCBA to (p, c) in step 5 takes M-time at most $8m(m^2 + 3m + 1)(\lg cp)\mu(3\lg cp)$. The sum of $\lg cp$ is at most $\lg bx$, since the product of c's divides b by Theorem 15.1. □

17. Merging coprime bases

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

Let k be a nonnegative integer. I define $\operatorname{bit}_i k$ as the ith bit in k's binary expansion; thus $k = \sum_{i \geq 0} 2^i \operatorname{bit}_i k$, with $\operatorname{bit}_i k \in \{0, 1\}$.

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

Proof. The point is that $q_k = \gcd\{x(\operatorname{bit}_i k, i) : 0 \leq i < b\}$. Indeed, the gcd is a product of q's by Theorem 7.5. It is divisible by q_j if and only if q_j divides each $x(\operatorname{bit}_i k, i)$, i.e., $\operatorname{bit}_i j = \operatorname{bit}_i k$ for each i, i.e., $j = k$.

Now if P is a base for all $x(e, i)$ then P is a base for $\gcd\{x(\operatorname{bit}_i k, i) : 0 \leq i < b\} = q_k$ by Theorem 7.5.

Conversely, if P is a base for all the q's then it is also a base for the x's, since each $x(e, i)$ is a product of q's. □

Algorithm U. 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 $b \geq 1$ with $2^b \geq n$. Set $S \leftarrow P$. Set $i \leftarrow 0$.
3. If $i \geq b$: Print $S$. Stop.
4. Compute $x \leftarrow \operatorname{prod}\{q_k : \operatorname{bit}_i k = 0\}$ by Algorithm P.
5. Compute $T \leftarrow \operatorname{cb}(S \cup \{x\})$ by EDCBA.
6. Compute $x \leftarrow \operatorname{prod}\{q_k : \operatorname{bit}_i k = 1\}$ by Algorithm P.
7. Compute $S \leftarrow \operatorname{cb}(T \cup \{x\})$ by EDCBA.
8. Set $i \leftarrow i + 1$. Return to step 3.
```

Theorem 17.2. *Let P and Q be finite coprime subsets of a free coid. Then Algorithm U prints* $\operatorname{cb}(P \cup Q)$.

Proof. Apply Theorem 17.1. The set S printed by Algorithm U is a coprime base for $P \cup \{x(0,0), x(0,1), \dots, x(0,b-1), x(1,0), x(1,1), \dots, x(1,b-1)\}$ by construction, and therefore a coprime base for $P \cup Q$. Naturalness follows from Theorem 7.1. $\square$

Theorem 17.3. *Let P and Q be finite coprime subsets of a free coid, with $1 \notin P \cup Q$. Define $z = (\operatorname{prod} P)(\operatorname{prod} Q)^2$. If $m \geq 1$ and $2^{m-1} \geq \lg z$ then Algorithm U finishes in M-time at most $2m(8m^3 + 28m^2 + 19m + 4)(\lg z)\mu(3 \lg z)$.*

Proof. By Theorem 7.6, $\operatorname{prod} S$ always divides $(\operatorname{prod} P)(\operatorname{prod} Q)$; and x divides $\operatorname{prod} Q$. Thus step 5 uses M-time at most $(8m^3 + 28m^2 + 18m + 4)(\lg z)\mu(3 \lg z)$ per iteration by Theorem 16.2. Step 4 uses M-time at most $m(\lg z)\mu(\lg z)$ by Theorem 14.1. Similar comments apply to step 7 and step 6. $\square$

Notes. Theorem 17.1 is crucial to my improvement over [2]. One can simply apply EDCBA to each element of Q in turn, as in Theorem 4.3, but this is far too slow when $\#Q$ is large. The trick here is to replace Q with a new set that has far fewer elements but has Q as its natural coprime base. One disadvantage of this trick is that the new set takes b times as much space as Q.

There are many ways to reduce the time spent in steps 4 and 6 of Algorithm U. However, steps 5 and 7 dominate the computation time.

18. Computing a coprime base for a finite set

The following algorithm uses binary splitting and Algorithm U to compute the natural coprime base for any finite set.

Algorithm M. 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 M recursively.
5. Compute $Q \leftarrow \operatorname{cb}(S - T)$ by Algorithm M recursively.
6. Print $\operatorname{cb}(P \cup Q)$ by Algorithm U.
```

Theorem 18.1. *Let S be a finite subset of a free coid. Then Algorithm M prints* $\operatorname{cb} S$.

Proof. By construction the output is a coprime base for T and for $S - T$, hence for S. Naturalness follows from Theorem 7.1. $\square$

Theorem 18.2. *Let S be a finite susbet of a free coid. Define $x = \operatorname{prod} S$. If $m \geq 1$, $2^{m-1} \geq 2 \lg x$, and $2^k \geq \#S \geq 1$ then Algorithm M finishes in M-time at most $4mk(8m^3 + 28m^2 + 19m + 4)(\lg x)\mu(6 \lg x)$.*

Proof. If $\#S = 1$ then Algorithm M 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 17.3. Add. $\square$

Part IV. Factorization

19. Computing ord

Let p and a be elements of a free coid, with $p \neq 1$. I define $\operatorname{reduce}(p, a) = (i, a/p^i)$, where i is the largest integer such that p^i divides a. In particular $i = \operatorname{ord}_p a$ if p is a quasiprime for a.

Algorithm E. Given (p, a), 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 E recursively.
3. If $p$ divides $b$: Print $(2j + 2, b/p)$ and stop.
4. Print $(2j + 1, b)$.
```

Theorem 19.1. *Let p and a be elements of a free coid, with $p \neq 1$. Then Algorithm E prints* $\operatorname{reduce}(p, a)$.

Proof. Induct on a.

If p does not divide a then $\operatorname{reduce}(p, a) = (0, a/p^0) = (0, a)$.

After step 2, by induction, $a/p = (p^2)^j b$, with p^2 not dividing 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 $\operatorname{reduce}(p, a) = (2j + 1, b)$. □

Theorem 19.2. *Let p and a be elements of a free coid, with $p \neq 1$. Define $(i, c) =$* $\operatorname{reduce}(p, a)$. *If $2^k > i + 1$ then Algorithm E computes (i, c) in M-time at most* $(4k - 3)(1 + \lg ap)\mu(\lg ap)$.

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

Case 1: i is zero. Algorithm E uses M-time at most $(1 + \lg ap)\mu(\lg ap)$ in step 1; it then stops, since p does not divide a. Finally $4k - 3 \geq 1$.

Case 2: i is odd. Then $j = (i-1)/2$ in step 2 of Algorithm E so $j+1 = (i+1)/2 < 2^{k-1}$. By induction Algorithm E uses M-time at most $(4k - 7)(1 + \lg ap)\mu(\lg ap)$ for the recursive call. It also uses M-time at most $(1 + \lg ap)\mu(\lg ap)$ in step 1, $(1 + \lg p^2)\mu(\lg p^2) + (1 + \lg a)\mu(\lg a)$ for the computations of p^2 and a/p in step 2, and $(1 + \lg bp)\mu(\lg bp)$ in step 3.

The total is at most $\mu(\lg ap)$ times $(4k-7)(1+\lg ap)+1+\lg ap+1+2\lg p+1+\lg a+1+\lg bp = 4k-3+(4k-4)\lg a+(4k-3-i)\lg p = (4k-3)(1+\lg ap)-\lg a-i\lg p$.

Case 3: i is even and nonzero. As before the recursive call uses M-time at most $(4k - 7)(1 + \lg ap)\mu(\lg ap)$. The other computations use M-time at most $(1 + \lg ap)\mu(\lg ap)$ in step 1, $(1 + \lg p^2)\mu(\lg p^2) + (1 + \lg a)\mu(\lg a)$ in step 2, and $(1 + \lg bp)\mu(\lg bp) + (1 + \lg b)\mu(\lg b)$ in step 3.

The total is at most $\mu(\lg ap)$ times $(4k-7)(1+\lg ap)+1+\lg ap+1+2\lg p+1+\lg a+1+\lg bp+1+\lg b = (4k-3)(1+\lg ap)+1+(2-2i)\lg p$. Finally $(2i-2)\lg p > 1$ since $\lg p \geq 1$. □

Notes. Algorithm E is a simplified version of the algorithm I outlined in [4, section 22].

20. Factoring over a coprime base

Fix a. Let P be a coprime set not containing 1. Algorithm F below factors a into powers of elements of P, if possible; otherwise it proclaims failure.

Algorithm F prints the factorization of a as a list of pairs (p, n) meaning p^n where $p \in P$. In practice one could represent p in this pair by a pointer into P.

Algorithm F. Given (a, P), with P coprime and $1 \notin P$, to print the factorization of a into elements of 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 E. 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 P.
5. Compute $(b, c) \leftarrow (\operatorname{ppi}, \operatorname{ppo})(a, y)$ by Algorithm PPIO.
6. Apply Algorithm F to $(b, Q)$ recursively. If Algorithm F fails, proclaim failure
   and stop.
7. Apply Algorithm F to $(c, P - Q)$ recursively. If Algorithm F fails, proclaim
   failure and stop.
```

Theorem 20.1. *Let H be a free coid. Let P be a finite coprime subset of H not containing* 1. *Let a be an element of H. If P is a base for $\{a\}$ then Algorithm F prints the factorization of a into elements of P. Otherwise Algorithm F proclaims failure.*

Proof. Induct on $\#P$.

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

Case 2: $P = \{p\}$. If Algorithm F 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 E returns $(n, 1)$, so Algorithm F 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 7.5. If $p \notin Q$ then $\operatorname{ord}_p y = 0$ so $\operatorname{ord}_p b = 0$; thus Q is a base for $\{b\}$. Similarly $P - Q$ is a base for $\{c\}$. By induction, Algorithm F 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 F does not proclaim failure.

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

Theorem 20.2. *Let H be a free coid. Let P be a finite coprime subset of H not containing* 1. *Define $x = \operatorname{prod} P$. Let a be an element of H. If $2^m \geq \#P \geq 1$ and $2^k > \lg a + 1$ then Algorithm F finishes in M-time at most*

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

times $\mu(\lg ax)$.

Proof. Induct on m.

Case 1: $\#P = 1$. Algorithm F uses M-time at most $(4k - 3)(1 + \lg ap)\mu(\lg ap)$ in step 2 by Theorem 19.2.

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 $U = 4k - 3 + (m - 1)m/2$.

Algorithm F uses M-time at most $(\frac{1}{2}\#P - 1 + (m - 1) \lg x)\mu(\lg x)$ in step 4 by Theorem 14.1 since $\#Q \leq \frac{1}{2}\#P$ and $\lg y \leq \lg x$. In step 5 it uses M-time at most $(3k + 3 + (2k + 4) \lg a + \lg x)\mu(\lg ax)$ by Theorem 11.5. By induction it uses M-time at most $\big(T \lg b + U \lg y + \big(7k - 1 + \frac{1}{2}(m - 1)\big)\#Q - 3k - 2\big)\mu(\lg by)$ in step 6 and

$\big(T\lg c + U\lg z + \big(7k - 1 + \frac{1}{2}(m-1)\big)\#(P-Q) - 3k - 2\big)\mu(\lg cz)$ in step 7. The total is at most $\mu(\lg ax)$ times

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

which equals the claimed bound. □

21. Factoring a set over a coprime base

Let S be a set, and let P be a coprime set not containing 1. Algorithm G below factors each element $a \in S$ into elements of P if P is a base for S; otherwise it proclaims failure.

Theorem 21.1. *Let H be a free coid. Let P be a finite coprime subset of H not containing* 1. *Let S be a finite subset of H. Define $x = \operatorname{prod} P$, $y = \operatorname{prod} S$, and $z = \operatorname{ppi}(x,y)$. Define $Q = \{p \in P : \operatorname{ppi}(z,p) = p\}$. Then P is a base for S if and only if Q is a base for S. Furthermore each element of Q divides y.*

Thus Q contains only the elements of P that are relevant to S.

Proof. $Q \subseteq P$, so if Q is a base for S then P is a base for S.

Conversely, say P is a base for S. Take any $p \in P$ dividing some element of S. Then $\operatorname{ord}_p x = 1$ and $\operatorname{ord}_p y > 0$, so $\operatorname{ord}_p z = 1$, so $\operatorname{ord}_p \operatorname{ppi}(z,p) = 1$. For any $q \in P$ other than p, $\operatorname{ord}_q p = 0$ so $\operatorname{ord}_q \operatorname{ppi}(z,p) = 0$. Thus $\operatorname{ppi}(z,p) = p$; i.e., $p \in Q$.

Finally, if $p \in Q$ then $\operatorname{ppi}(z,p) = p$ so $\operatorname{ord}_p z = 1$ so $\operatorname{ord}_p y > 0$ so p divides y. □

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

```
1. If $S = \{\}$: Stop.
2. Compute $x \leftarrow \operatorname{prod} P$ by Algorithm P.
3. Compute $y \leftarrow \operatorname{prod} S$ by Algorithm P.
4. Compute $z \leftarrow \operatorname{ppi}(x,y)$ by Algorithm PPIO.
5. Compute $D \leftarrow \operatorname{split}(z,P)$ by Algorithm S.
6. Compute $Q \leftarrow \{p \in P : (p,p) \in D\}$.
7. If $\#S = 1$: Apply Algorithm F to $(y,Q)$, proclaiming failure if Algorithm F
   fails. Stop.
8. Select $T \subseteq S$ with $\#T = \lfloor \#S/2 \rfloor$.
9. Apply Algorithm G to $(T,Q)$ recursively.
10. Apply Algorithm G to $(S-T,Q)$ recursively.
```

Theorem 21.2. *Let S be a finite subset of a free coid. Let P be a finite coprime base for S, with $1 \notin P$. Then Algorithm G prints the factorization of each element of S into elements of P.*

Proof. By Theorem 21.1, Q is a base for S. Induct on $\#S$. If $\#S = 1$ then $S = \{y\}$ and step 7 prints the factorization of y. If $\#S \geq 2$ then Q is a base for T and for $S - T$; hence, by induction, Algorithm G prints the factorization of each element of T in step 9 and each element of $S - T$ in step 10. □

Theorem 21.3. *Let S be a finite subset of a free coid. Let P be a finite coprime base for S, with $1 \notin P$. Define $x = \operatorname{prod} P$ and $y = \operatorname{prod} S$. Algorithm G finishes* *in M-time at most $\mu(\lg x^2y)$ times*

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

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

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 14.1 for $\#P \ge 1$. (For $\#P = 0$, Algorithm P 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 S uses no M-time.)

The total so far is at most $\mu(\lg x^2y)$ times $3m+3+(4.5m^2+21.5m+15)\lg x+(k+2)\lg y$. This leaves

$$\big((6k+6)(\#S-1)+((9k^2+44k+32)n+2.5k^2+20k-7)\lg y\big)\,\mu(\lg x^2y)$$

unaccounted for in the claimed run time.

Case 1: $\#S = 1$. Then step 7 uses M-time at most $(2.5k^2+20k-7)\lg y$ times $\mu(\lg x^2y)$. This follows from Theorem 20.2 for $\#Q \ge 1$, since $2^k \ge \#Q$. (For $\#Q = 0$, Algorithm F 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 $\mu(\lg x^2y)$ times

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

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

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

Add. □

Notes. Algorithm G is faster than the algorithm of [2, Theorem 7].

In step 6 of Algorithm G one could proclaim failure if $(p,c) \in D$ for some $c \notin \{1,p\}$.

References

[1] —, *International symposium on symbolic and algebraic computation '90*, Association for Computing Machinery, New York, 1990.

[2] Eric Bach, James Driscoll, Jeffrey Shallit, *Factor refinement*, Journal of Algorithms **15** (1993), 199–222.

[3] Eric Bach, Gary Miller, Jeffrey Shallit, *Sums of divisors, perfect numbers, and factoring*, SIAM Journal on Computing **15** (1986), 1143–1154.

[4] Daniel J. Bernstein, *Detecting perfect powers in essentially linear time*, to appear, Mathematics of Computation.

[5] Daniel J. Bernstein, *Fast ideal arithmetic via lazy localization*, in [10], 27–34.

[6] Jonathan M. Borwein, Peter B. Borwein, *Pi and the AGM*, Wiley, New York, 1987.

[7] Richard P. Brent, Fred G. Gustavson, David Y. Y. Yun, *Fast solution of Topelitz systems of equations and computation of Padé approximants*, Journal of Algorithms **1** (1980), 259–295.

[8] Jacques Calmet (editor), *Algebraic algorithms and error-correcting codes 3*, Lecture Notes in Computer Science 229, Springer-Verlag, Berlin, 1986.
[9] Bob F. Caviness (editor), *Proceedings of EUROCAL '85, volume 2*, Lecture Notes in Computer Science 204, Springer-Verlag, Berlin, 1985.
[10] Henri Cohen (editor), *Proceedings of the algorithmic number theory symposium 2*, Lecture Notes in Computer Science 1122, Springer-Verlag, Berlin, 1996.
[11] George E. Collins, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition: preliminary report*, SIGSAM Bulletin **8** (1974), 80–90.
[12] Jean Della Dora, Claire DiCrescenzo, Dominique Duval, *About a new method for computing in algebraic number fields*, in [9], 289–290.
[13] Joachim von zur Gathen, *Representations and parallel computations for rational functions*, SIAM Journal of Computing **15** (1986), 432–452.
[14] Guoqiang Ge, *Recognizing units in number fields*, Mathematics of Computation **63** (1994), 377–387.
[15] Erich Kaltofen, *Sparse Hensel lifting*, in [9], 4–17.
[16] Erich Kaltofen, Victor Shoup, *Subquadratic-time factoring of polynomials over finite fields*, preprint.
[17] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 2nd edition, Addison-Wesley, Reading, Massachusetts, 1981.
[18] Heinz Lüneburg, *On a little but useful algorithm*, in [8], 296–301.
[19] Michael Pohst, Hans Zassenhaus, *Algorithmic algebraic number theory*, Cambridge University Press, Cambridge, 1989.
[20] Arnold Schönhage, *Schnelle Berechnung von Kettenbruchentwicklugen*, Acta Informatica **1** (1971), 139–144.
[21] Trevor J. Smedley, *Detecting algebraic dependencies between unnested radicals: extended abstract*, in [1], 292–293.
[22] Thomas Jan Stieltjes, *Sur la théorie des nombres*, Ann. Fac. Sci. Toulouse **4** (1890), 1–103.
[23] Jeremy T. Teitelbaum, *On the computational complexity of the resolution of plane curve singularities*, Mathematics of Computation **54** (1990), 797–837.

Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, Chicago, IL 60607–7045
E-mail address: `djb@pobox.com`