How to find smooth parts of integers

D. J. Bernstein

Thanks to:
University of Illinois at Chicago
NSF DMS–0140542
Alfred P. Sloan Foundation
Integer-factorization bottleneck: Given sequence of numbers, find nonempty subsequence with square product.
e.g. given 6, 7, 8, 10, 15,
discover $6 \cdot 10 \cdot 15 = 30^2$.

Discrete-log bottleneck: Given sequence of numbers, find 1 as nontrivial product of powers.
e.g. given 6, 7, 8, 10, 15,
discover $6^3 7^0 8^{-2} 10^3 15^{-3} = 1$.

More generally: find $k$th power.
This is a bottom-up talk aiming at these bottlenecks.
Will focus on integers. Can use same techniques, and more, for polynomials in function-field sieve etc.
Will focus on conventional architectures: e.g. multitape Turing machines. Optimization is very different for mesh architectures.
Multiplication and division

Given \( r, s \in \mathbb{Z} \), can compute \( rs \) in time \( \leq b(\lg b)^{1+o(1)} \) where \( b \) is number of input bits.

(1971 Pollard; independently 1971 Nicholson; independently 1971 Schönhage Strassen)

Also time \( \leq b(\lg b)^{1+o(1)} \) where \( b \) is number of input bits:

Given \( r, s \in \mathbb{Z} \) with \( s \neq 0 \), compute \( \lfloor r/s \rfloor \) and \( r \mod s \).

(reduction to product: 1966 Cook)
Product trees

Time $\leq b(\lg b)^{2+o(1)}$

where $b$ is number of input bits:

Given $x_1, x_2, \ldots, x_n \in \mathbb{Z}$,
compute $x_1x_2 \cdots x_n$.

Actually compute

**product tree** of $x_1, x_2, \ldots, x_n$.

Root is $x_1x_2 \cdots x_n$.

Has left subtree if $n \geq 2$:
product tree of $x_1, \ldots, x_{\lceil n/2 \rceil}$.

Also right subtree if $n \geq 2$:
product tree of $x_{\lceil n/2 \rceil}+1, \ldots, x_n$. 
e.g. tree for 23, 29, 84, 15, 58, 19:

Tree has \( \leq (\lg b)^{1+o(1)} \) levels.
Each level has \( \leq b(\lg b)^{0+o(1)} \) bits.

Obtain each level
in time \( \leq b(\lg b)^{1+o(1)} \)
by multiplying lower-level pairs.
Remainder trees

Remainder tree
of $r, x_1, x_2, \ldots, x_n$ has one node $r \mod t$ for each node $t$ in product tree of $x_1, x_2, \ldots, x_n$.

e.g. remainder tree of $223092870, 23, 29, 84, 15, 58, 19$:
Time $\leq b(\lg b)^{2+o(1)}$:
Given $r \in \mathbb{Z}$ and nonzero $x_1, \ldots, x_n \in \mathbb{Z}$, compute remainder tree of $r, x_1, \ldots, x_n$.

In particular, compute $r \mod x_1, \ldots, r \mod x_n$.

In particular, see which of $x_1, \ldots, x_n$ divide $r$.

(1972 Moenck Borodin, for “single precision” $x_i$’s, whatever exactly that means)
Small primes, union

Time $\leq b(\lg b)^{2+o(1)}$:

Given $x_1, x_2, \ldots, x_n \in \mathbb{Z}$ and finite set $Q \subseteq \mathbb{Z} - \{0\}$, compute

$\{p \in Q : x_1x_2\cdots x_n \mod p = 0\}$.

In particular, when $p$ is prime, see whether $p$ divides any of $x_1, x_2, \ldots, x_n$.

Algorithm:

1. Use a product tree to compute $r = x_1x_2\cdots x_n$.

2. Use a remainder tree to see which $p \in Q$ divide $r$. 
Small primes, separately

Time $\leq b(\lg b)^{3+o(1)}$:

Given $x_1, x_2, \ldots, x_n \in \mathbb{Z}$ and finite set $Q$ of primes, compute $\{p \in Q : x_1 \mod p = 0\}$, $\ldots$, $\{p \in Q : x_n \mod p = 0\}$.

(2000 Bernstein)

Algorithm for $n \geq 1$:
1. Replace $Q$ with $\{p \in Q : x_1 \cdots x_n \mod p = 0\}$.
2. If $n = 1$, print $Q$ and stop.
3. Recurse on $x_1, \ldots, x_{\lceil n/2 \rceil}, Q$.
4. Recurse on $x_{\lceil n/2 \rceil + 1}, \ldots, x_n, Q$. 
Factor 2543, 6766, 8967, 7598 over \{2, 3, 5, 7, 11, 13, 17\}

2543, 6766
over \{2, 3, 7, 17\}

8967, 7598
over \{2, 3, 7, 17\}

2543
over \{2, 17\}

6766
over \{2, 17\}

8967
over \{2, 3, 7\}

7598
over \{2, 3, 7\}

Each level has \(\leq b(\lg b)^{0+o(1)}\) bits.
Exponents of a small prime

\[ \text{Time } \leq b(\lg b)^{2+o(1)}: \]

Given nonzero \( p, x \in \mathbb{Z} \), find \( e, p^e, x/p^e \) with maximal \( e \).

Algorithm:
1. If \( x \mod p \neq 0 \):
   Print 0, 1, \( x \) and stop.
2. Find \( f, (p^2)^f, r = (x/p)/(p^2)^f \) with maximal \( f \).
3. If \( r \mod p = 0 \): Print
   \[ 2f + 2, (p^2)^f p^2, r/p \] and stop.
4. Print \( 2f + 1, (p^2)^f p, r \).
Exponents of small primes

Time \leq b(\lg b)^{3+o(1)}:

Given finite set \( Q \) of primes and nonzero \( x \in \mathbb{Z} \), find maximal \( e, \prod_{p \in Q} p^e(p), x/\prod_{p \in Q} p^e(p) \).

Algorithm:

1. Replace \( Q \) with 
   \[ \{ p \in Q : x \mod p = 0 \} \].
2. Find maximal \( f, s, r \) with 
   \[ s = \prod(p^2)^f(p^2), r = (x/\prod p)/s. \]
3. Find \( T = \{ p \in Q : r \mod p = 0 \} \).
4. Answer is \( e, s \prod_{p \in T} p, r/\prod_{p \in T} p \) where \( e(p) = 2f(p^2) + [p \in T] \).
Smooth parts, old approach

Time $\leq b(\log b)^{3+o(1)}$: 
Given nonzero $x_1, x_2, \ldots, x_n \in \mathbb{Z}$ and finite set $Q$ of primes, compute $Q$-smooth part of $x_1$, $Q$-smooth part of $x_2$, $\ldots$, $Q$-smooth part of $x_n$.

$Q$-smooth means product of powers of elements of $Q$.

$Q$-smooth part means largest $Q$-smooth divisor. In particular, see which of $x_1, x_2, \ldots, x_n$ are smooth.
Algorithm:
1. Find \( Q_1 = \{p : x_1 \mod p = 0\}\),
   \[\ldots, Q_n = \{p : x_n \mod p = 0\}\].
2. For each \( i \) separately:
   Find maximal \( e, s, r \) with
   \[s = \prod_{p \in Q_i} p^e(p), \quad r = x_i / s.\]
   Print \( s \).

E.g. factoring 2543, 6766, 8967, 7598 over \( \{2, 3, 5, 7, 11, 13, 17\} \):
2543 over \( \{\} \), smooth part 1;
6766 over \( \{2, 17\} \), smooth part 34;
8967 over \( \{3, 7\} \), smooth part 147;
7598 over \( \{2\} \), smooth part 2.
Smooth multiplicative dependencies

Recall cryptanalytic bottleneck: find $k$th power nontrivially as product of powers of $x_1, x_2, \ldots, x_n$.

Choose $y$; imagine $y = 2^{40}$. Define $Q$ as set of primes $\leq y$. See which of $x_1, x_2, \ldots, x_n$ are $y$-smooth, i.e., $Q$-smooth. Know their factorizations. Do linear algebra over $\mathbb{Z}/k$ on the exponent vectors.
**Sieving**

In linear sieve (1977 Schroeppel), number-field sieve, etc., \( x \)'s are consecutive values of a low-degree polynomial.

Choose \( \theta \); imagine \( \theta = 0.5 \).
Sieve to discover primes \( \leq y^\theta \); say time \( S \) per number.
Keep most promising \( x \)'s.
See which ones are \( y \)-smooth; say time \( T \) per number.

Time to find each smooth number is roughly \( S^\theta T^{1-\theta} \) after optimization.
Smooth parts, new approach

Given nonzero $x_1, x_2, \ldots, x_n \in \mathbb{Z}$ and finite set $Q$ of primes:
Time typically $\leq b(\lg b)^{2+o(1)}$ to obtain smooth parts of $x$’s.
(2004 Franke Kleinjung Morain Wirth, in ECPP context)

Algorithm:
Compute $r = \prod_{p \in Q} p$.
Compute $r \mod x_1, \ldots, r \mod x_n$.
For each $i$ separately:
Replace $x_i$ by $x_i / \gcd \{x_i, r \mod x_i\}$ repeatedly until $\gcd$ is 1.
Slight variant (2004 Bernstein): Time always $\leq b(\lg b)^{2+o(1)}$.

Compute smooth part of $x_i$ as $\gcd \{x_i, (r \mod x_i)^{2^k} \mod x_i\}$ where $k = \lceil \lg \lg x_i \rceil$.

Subroutine: Computing gcd takes time $\leq b(\lg b)^{2+o(1)}$.

(1971 Schönhage; core idea: 1938 Lehmer; $b(\lg b)^{5+o(1)}$: 1971 Knuth)

Or, to see if $x_i$ is smooth, see if $(r \mod x_i)^{2^k} \mod x_i = 0$. 
Minor problem: New algorithm finds the smooth numbers but doesn’t factor them.

Solution: Feed the smooth numbers to the old algorithm. Very few smooth numbers, so this is very fast.

Bottom line: \( T \), time per number to find and factor smooth numbers, has dropped by \((\lg b)^{1+o(1)}\).

This is big news for cryptanalysis!
Is smooth the right question?

After finding smooth numbers, do first step of linear algebra: Throw away primes that appear only once; throw away numbers with those primes; repeat until stable.

Don’t want all smooth numbers. Want smooth numbers only if they are built from primes that divide the other numbers.
An alternate approach

Given nonzero $x_1, x_2, \ldots, x_n \in \mathbb{Z}$:
Compute $r = x_1 x_2 \cdots x_n$.
Compute $(r/x_1) \mod x_1, \ldots, (r/x_n) \mod x_n$.
For each $i$ separately: see if 
$((r/x_i) \mod x_i)^{2^k} \mod x_i = 0$
where $k = \lceil \lg \lg x_i \rceil$.

Finds $x_i$ iff all primes in $x_i$
are divisors of other $x$’s.
Time $\leq b(\lg b)^{2+o(1)}$.

(2004 Bernstein)
Compute \((r/x_1) \mod x_1, \ldots, (r/x_n) \mod x_n\) by computing \(r \mod x_1^2, \ldots, r \mod x_n^2\).

(1972 Moenck Borodin)

Problem: Recognizing the interesting \(x\)'s is not enough; also need their factorizations.

Solution: Again, very few of them. Have ample time to use rho method (1974 Pollard) or use ECM (1987 Lenstra) or factor into coprimes.
Factoring into coprimes

Time $\leq b(\lg b)^{O(1)}$:

Given positive $x_1, x_2, \ldots, x_n$, find coprime set $Q$ and complete factorization of each $x_i$ over $Q$.

(announced 1995 Bernstein; now at second-galley stage for J. Algorithms)

Immediately gives $b(\lg b)^{O(1)}$ for the other factoring problems. Subsequent research: $\lg$ speedups, constant-factor speedups, etc.
Speedup: aligning roots

Original FFT (1805 Gauss, et al.): $(4.5 + o(1)) n \lg n$ operations in $\mathbb{C}$ to multiply in $\mathbb{C}[x]/(x^n - 1)$; or $(15 + o(1)) n \lg n$ operations in $\mathbb{R}$.

Split-radix FFT (1968 Yavne; Duhamel, Hollmann, Martens, Stasinski, Vetterli, Nussbaumer): $(4.5 + o(1)) n \lg n$ operations in $\mathbb{C}$ to multiply in $\mathbb{C}[x]/(x^n - 1)$; only $(12 + o(1)) n \lg n$ operations in $\mathbb{R}$.

Why fewer operations in $\mathbb{R}$?
Multiplications in $\mathbb{C}$ for original FFT:
$1.5n$ by primitive 4th roots of 1,
$1.5n$ by primitive 8th roots of 1,
$1.5n$ by primitive 16th roots of 1,
e.tc.

For split-radix FFT:
$0.5n \lg n$ by primitive 4th roots of 1,
$n$ by primitive 8th roots of 1,
$n$ by primitive 16th roots of 1,
e.tc.

Split-radix FFT
aligns many of the roots
to be 4th roots of 1.
In Schönhage-Strassen context, aligning roots produces much larger speedups. (2000 Bernstein)

Consider size-65536 FFT over \( \mathbb{A} \) where \( \mathbb{A} = \mathbb{Z}/(2^{16384} + 1); \) \( 2^{12288} - 2^{4096} \) is a square root of 2 in \( \mathbb{A} \).

Multiplications by powers of 2 usually mean annoying shifts across word boundaries. Alignment avoids almost all of this. Also sometimes makes slightly larger FFT sizes practical.
Speedup: better caching

Multiply in \( \mathbb{Z}/(2^{1048576000} - 1) \) by lifting to \( \mathbb{Z}[x]/(x^{65536} - 1) \), mapping to \( A[x]/(x^{65536} - 1) \), using FFT. (1971 Schönhage Strassen for negacyclic case)

Reorganize FFT operations to reduce communication costs. (1966 Gentleman Sande, et al.)

Can reduce communication costs even more by aligning roots and violating A operation atomicity. (2004 Bernstein)
Speedup: FFT doubling

(2004 Kramer)

Consider product tree for \(x_1, x_2, x_3, x_4\), each \(b/4\) bits.

Compute \(x_1x_2\) as
\[
\text{FFT}^{-1}_{b/2}(\text{FFT}_{b/2}(x_1) \text{FFT}_{b/2}(x_2)).
\]

Compute \(x_1x_2x_3x_4\) as
\[
\text{FFT}^{-1}_b(\text{FFT}_b(x_1x_2) \text{FFT}_b(x_3x_4)).
\]
First half of \(\text{FFT}_b(x_1x_2)\) is
\[
\text{FFT}_{b/2}(x_1x_2), \text{already known!}
\]

For large product trees,
\[
1.5 + o(1) \text{ speedup}.
\]
Some additional speedups

Start Newton for $1/x_1x_2$
at product of approximations
to $1/x_1$ and $1/x_2$.

Remove redundancy in division.

Use $2$-adic division.

Eliminate tiny primes.

Further reduce the $2^k$
by using powers of small primes.

Balance gcd and powering.