DETECTING PERFECT POWERS BY FACTORING INTO COPRIMES

DANIEL J. BERNSTEIN, HENDRIK W. LENSTRA, JR., AND JONATHAN PILA

Abstract. This paper presents an algorithm that, given an integer $n > 1$, finds the largest integer k such that n is a kth power. A previous algorithm by the first author took time $b^{1+o(1)}$ where $b = \lg n$; more precisely, time $b \exp(O(\sqrt{\lg b \lg \lg b}))$; conjecturally, time $b(\lg b)^{O(1)}$. The new algorithm takes time $b(\lg b)^{O(1)}$. It relies on relatively complicated subroutines—specifically, on the first author's fast algorithm to factor integers into coprimes—but it allows a proof of the $b(\lg b)^{O(1)}$ bound without much background; the previous proof of $b^{1+o(1)}$ relied on transcendental number theory.

The computation of k is the first step, and occasionally the bottleneck, in many number-theoretic algorithms: the Agrawal-Kayal-Saxena primality test, for example, and the number-field sieve for integer factorization.

Here is an algorithm that, given an integer $n > 1$, finds the largest integer k such that n is a kth power:

1. For each prime power q such that $2^q \le n$, write down a positive integer r_q such that if n is a qth power then $n = r_q^q$.
2. Find a finite coprime set P of integers larger than 1 such that each of $n, r_2, r_3, r_4, r_5, r_7, \ldots$ is a product of powers of elements of P. (In this paper, "coprime" means "pairwise coprime.")
3. Factor n as $\prod_{p \in P} p^{n_p}$, and compute $k = \gcd\{n_p : p \in P\}$.

It is easy to see that the algorithm is correct. Say n is an ℓth power. Take any prime power q dividing ℓ. Then n is a qth power, so $n = r_q^q$; but r_q is a product $\prod_{p \in P} p^{a_p}$ for some exponents a_p, so n is a product $\prod_{p \in P} p^{q a_p}$. Factorizations over P are unique, so $n_p = q a_p$ for each p. Thus q divides $\gcd\{n_p : p \in P\} = k$. This is true for all q, so ℓ divides k. Conversely, n is certainly a kth power.

Take, for example, $n = 49787136 < 2^{26}$. Compute approximations

$$
\begin{array}{lll}
r_2 = 7056 \approx n^{1/2} & r_8 = 9 \approx n^{1/8} & r_{17} = 3 \approx n^{1/17} \\
r_3 = 368 \approx n^{1/3} & r_9 = 7 \approx n^{1/9} & r_{19} = 3 \approx n^{1/19} \\
r_4 = 84 \approx n^{1/4} & r_{11} = 5 \approx n^{1/11} & r_{23} = 2 \approx n^{1/23} \\
r_5 = 35 \approx n^{1/5} & r_{13} = 4 \approx n^{1/13} & r_{25} = 2 \approx n^{1/25} \\
r_7 = 13 \approx n^{1/7} & r_{16} = 3 \approx n^{1/16} &
\end{array}
$$

where $\approx$ means "within 0.6." Factor $\{49787136, 7056, 368, 84, 35, 13, 9, 7, 5, 4, 3, 2\}$ into coprimes: each of these numbers is a product of powers of elements of $P =$

Date: 2005.05.09. Permanent ID of this document: `bbd41ce71e527d3c06295aadccf60979`.

2000 *Mathematics Subject Classification.* Primary 11Y16.

Initial work: Lenstra was supported by the National Science Foundation under grant DMS–9224205. Subsequent work: Bernstein was supported by the National Science Foundation under grant DMS–0140542. The authors thank the University of California at Berkeley and the Fields Institute for Research in Mathematical Sciences.

$\{2, 3, 5, 7, 13, 23\}$. In particular, $n = 2^8 3^4 5^0 7^4 13^0 23^0$, so $k = \gcd\{8, 4, 0, 4, 0, 0\} = 4$. In other words, n is a 4th power, and is not an ℓth power for $\ell > 4$.

As discussed below, the literature already shows how to perform each step of this algorithm in time $b(\lg b)^{O(1)}$, where $b = \lg n$. Computing $n^{1/k}$, which is used by some applications, also takes time $b(\lg b)^{O(1)}$.

Details of Step 1. Here is one of several standard ways to handle Step 1.

Given n and q, use binary search and Newton's method to compute a floating-point number guaranteed to be within 2^{-32} of $n^{1/q}$, as explained in [4, Sections 8 and 10]. The algorithms of [4] rely on FFT-based integer multiplication; see [6, Sections 2–4].

Define r_q as an integer within 2^{-32} of this floating-point number. If no such integer exists, define $r_q = 1$.

Each r_q has $O(b/q)$ bits. Together the r_q's have $\sum_{q \le \lg n} O(b/q) = O(b \lg \lg b)$ bits by Mertens's theorem. The algorithms of [4] take time $(\lg b)^{O(1)}$ per bit.

Another standard way to handle Step 1 is to define r_q as an integer 2-adically close to $n^{1/q}$, as explained in [4, Section 21].

One can change the bound 2^{-32}. We caution the reader that the two numerical examples in this paper use different bounds. A smaller bound requires a higher-precision computation of $n^{1/q}$ but—for typical distributions of n—is more likely to produce $r_q = 1$, reducing the load on subsequent steps of the algorithm. The typical behavior of the algorithm is discussed below in more detail.

Details of Step 2. Given a finite set of positive integers, the algorithm of [5, Section 18] computes the "natural coprime base" for that set. The algorithm takes time $s(\lg s)^{O(1)}$ where s is the number of input bits. The algorithm relies on FFT-based multiplication, division, and gcd; see [6, Sections 17 and 22].

Use this algorithm to compute the "natural coprime base" P for $\{n, r_2, \ldots\}$. Together $n, r_2, \ldots$ have $O(b \lg \lg b)$ bits, so this takes time $b(\lg b)^{O(1)}$.

Details of Step 3. Given a finite coprime set P of integers larger than 1, and given a positive integer that has a factorization over P, the algorithm of [5, Section 20] finds that factorization. The algorithm takes time $s(\lg s)^{O(1)}$ where s is the number of input bits. The algorithm relies on FFT-based arithmetic.

Use this algorithm to factor n over P. Together n and P have $O(b \lg \lg b)$ bits, so this takes time $b(\lg b)^{O(1)}$.

Competition. Previous work by the first author in [4] had already shown that k could be computed in time $b^{1+o(1)}$. The algorithm of [4] computes r_q for prime numbers q, and then computes several increasingly precise approximations to r_q^q, stopping when an approximation demonstrates that $r_q^q \ne n$.

The run-time bound for the algorithm in this paper has two advantages over the run-time bound for the algorithm in [4]:

- The new bound is smaller. The old bound was $b \exp(O(\sqrt{\lg b \lg \lg \lg b}))$; the new bound is $b(\lg b)^{O(1)}$.
- The new proof requires considerably less background. The new proof relies on the first author's results in [5] on factoring into coprimes, but the old proof relied on deep results in transcendental number theory.

The old algorithm is conjectured to take time $b(\lg b)^{O(1)}$, as discussed in [4, Section 15], but this conjecture seems very difficult to prove.

Performance in the typical case. For most values of n, computing a floating-point number within 2^{-32} of $n^{1/2}$ reveals immediately that n is not a square, because the floating-point number is not within 2^{-32} of an integer.

Similarly, for almost all values of n, computing reasonably precise floating-point approximations to $n^{1/2}, n^{1/3}, \ldots$ reveals immediately that $k = 1$. Here one can define "reasonably precise" as, e.g., "within $2^{-32}/b$." For example, take $n = 31415926535897932384626433833$, and compute

$$\begin{aligned}
56049912163979.2869928550892 &\approx n^{1/2}, & r_2 = r_4 = r_8 = r_{16} = r_{32} = r_{64} = 1;\\
1464591887.5615232630107 &\approx n^{1/3}, & r_3 = r_9 = r_{27} = r_{81} = 1;\\
315812.9791837632319 &\approx n^{1/5}, & r_5 = r_{25} = 1;\\
8475.4793001649371 &\approx n^{1/7}, & r_7 = r_{49} = 1;\\
316.0391590557065 &\approx n^{1/11}, & r_{11} = 1;\\
130.3663105302392 &\approx n^{1/13}, & r_{13} = 1;\\
41.4456928612363 &\approx n^{1/17}, & r_{17} = 1;\\
28.0038933071808 &\approx n^{1/19}, & r_{19} = 1;\\
15.6865795173630 &\approx n^{1/23}, & r_{23} = 1;\\
8.8751884186190 &\approx n^{1/29}, & r_{29} = 1;\\
7.7091205087505 &\approx n^{1/31}, & r_{31} = 1;\\
5.5356192737976 &\approx n^{1/37}, & r_{37} = 1;\\
4.6844886605433 &\approx n^{1/41}, & r_{41} = 1;\\
4.3598204254547 &\approx n^{1/43}, & r_{43} = 1;\\
3.8463229122474 &\approx n^{1/47}, & r_{47} = 1;\\
3.3022819333873 &\approx n^{1/53}, & r_{53} = 1;\\
2.9245118649948 &\approx n^{1/59}, & r_{59} = 1;\\
2.8234034999139 &\approx n^{1/61}, & r_{61} = 1;\\
2.5727952305908 &\approx n^{1/67}, & r_{67} = 1;\\
2.4394043898716 &\approx n^{1/71}, & r_{71} = 1;\\
2.3805279554537 &\approx n^{1/73}, & r_{73} = 1;\\
2.2287696658789 &\approx n^{1/79}, & r_{79} = 1;\\
2.1443267449321 &\approx n^{1/83}, & r_{83} = 1;\\
2.0368391790628 &\approx n^{1/89}, & r_{89} = 1;
\end{aligned}$$

where now $\approx$ means "within 2^{-40}." Evidently $k = 1$.

For these typical values of n, there is no difference between the algorithm in this paper and the algorithm of [4]. All the time is spent computing approximate roots. Doing better means computing fewer roots—see [4, Section 22]—or computing the roots more quickly; these improvements apply equally to both algorithms.

For the other values of n—the atypical integers that are close to squares, cubes, etc.—the algorithms behave differently. It is not easy to analyze, or experiment with, the actual worst-case behavior of the algorithms, because it is not easy to find integers that are simultaneously close to many powers. We leave this as a challenge for the reader.

History. Bach, Driscoll, and Shallit in [2] introduced a quadratic-time algorithm to factor integers into coprimes. The obvious algorithm takes cubic time.

Bach and Sorenson in [3] published various algorithms to detect perfect powers, i.e., to check whether $k > 1$. One algorithm takes time $O(b^3)$. Another algorithm is conjectured to take time $O(b^2/(\lg b)^2)$ for most, but not all, n's.

The second and third authors of this paper observed in early 1994 that they could compute k in time $O(b^2(\lg\lg b)^2)$ by factoring $n, r_2, \ldots$ into coprimes with the Bach-Driscoll-Shallit algorithm; recall that $n, r_2, \ldots$ together have $O(b \lg\lg b)$ bits. This line of work was abandoned several months later when the first author announced that k could be computed in time $b^{1+o(1)}$ by the increasingly-precise-approximations-to-r_q^q method.

The first author later pointed out that this line of work deserved to be revived, since he had found an essentially-linear-time algorithm—see [5]—to factor integers into coprimes.

References

[1] Eric Bach, James Driscoll, Jeffrey Shallit, *Factor refinement*, in [7] (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, Jonathan Sorenson, *Sieve algorithms for perfect power testing*, Algorithmica **9** (1993), 313–328. ISSN 0178–4617. MR 94d:11103.

[4] 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`.

[5] Daniel J. Bernstein, *Factoring into coprimes in essentially linear time*, Journal of Algorithms **54** (2005), 1–30. ISSN 0196–6774. URL: `http://cr.yp.to/papers.html#dcba`. ID `f32943f0bb67a9317d4021513f9eee5a`.

[6] 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`. ID `8758803e61822d485d54251b27b1a20d`.

[7] 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.

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

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

Mathematisch Instituut, Universiteit Leiden, Postbus 9512, 2300 RA Leiden, The Netherlands

E-mail address: `hwl@math.leidenuniv.nl`

Department of Mathematics and Statistics, McGill University, Burnside Hall, Montreal, Quebec, H2A 2K6, Canada

E-mail address: `pila@math.mcgill.ca`