Review of[1]
The Joy of Factoring
by Samuel Wagstaff
Publisher: AMS
$35.00 softcover , 293 pages, Year: 2013

Reviewer: William Gasarch `gasarch@cs.umd.edu`

1 Introduction

This is a book on algorithms for factoring. As you might expect, there is also a lot of number theory presented. What is more surprising is that there is not much in the way of analyzing algorithms. This is *not* a criticism of the book. The field in general seems to not have that much in the way of rigorous runtimes of algorithms. This is *not* a criticism of the field.

Reading the book was odd in that there was more concern for algorithms that could *actually* be coded up (many *actually* have been) that would *actually* run fast, and less concern for rigorous runtime analysis. This is unusual for computer science algorithms. Some *might* take that as a criticism of computer science research in algorithms, but I am not going to go there.

Why is factoring an area where practical algorithms are emphasized and rigorous analysis is not? I speculate that there are two reasons: (1) people really want algorithms that factor quickly (for cryptography), and (2) the number theory needed to prove runtimes of algorithms often involves open problems in umber theory.

Notation: Throughout this review N is the number we want to factor and $b = \left\lfloor \sqrt{N} \right\rfloor$.

2 Summary of Contents

The first five chapters a blend of number theory needed for factoring algorithms, simple factoring algorithms, and factoring algorithms for numbers of a certain type. We give some examples of each:

1. (r/p) is the Legendre symbol: it is 1 when r is a nonzero square mod p, -1 when r is not a square mod p, and 0 when r divides p. There is a section on how to compute this easily. Theorems from number theory are stated but not proven.

2. Trivial Algorithm: If N is small you certainly do not want to use a fancy algorithm on it. Hence a simple fast algorithm is needed for N small. This book discusses the so-called trivial algorithm: divide n by $2, 3, \ldots, b$, in some depth. The key is that this algorithm can be sped up considerably. The first observation is that you do not need to divide by any even larger than 2. Hence you need only divide by $2, 3, 5, 7, 9, \ldots, b$. AH- you need not divide by any multiple of 3 larger than 3. This leads to only trying numbers that are $\equiv 1, 5 \pmod 6$. If you avoid all multiples of 2,3, and 5 then you only try numbers that are $\equiv 1, 7, 11, 13, 17, 19, 23, 29 \pmod{30}$. The fact that they are all primes is an accident which no longer holes when you avoid numbers that are a multiple of $1, 3, 5, 7$ and use mod 210. One can go further and further with this but at some points the cost of making sure your numbers are in one of the congruence classes outweighs the benefits.

[1]©2016 William Gasarch

3. Fermat's difference of squares algorithm: For each of $b^2 - N$, $(b+1)^2 - N$, $(b+2)^2 - N$, $\ldots$ test if it is a square and stop if it is. If $(b+i)^2 - N$ is a square then $(b+i)^2 - N = y^2$ so $(b+i-y)(b+i+y) = N$. Hence a factor is found (unless $b+i-y = 1$ and $b+i+y = N$). This algorithm is analyzed rigorously and ways to speed it up are given. In the worst case it works in time $O(N^{2/3})$ but if one of the factors of N is close to $\sqrt{N}$ (which is common when trying to crack RSA) then this algorithm is very fast. Also, this technique of trying to find x, y such that $x^2 - y^2$ is N (or a multiple of N) is the starting point for many other algorithms.

4. Let p be the lowest prime factor of N. We do not know p but we do know that $p \le \sqrt{N}$. Pollard's $p-1$ method begins by noting that $2^{p-1} \equiv 1 \pmod{p}$, hence $2^{p-1} - 1 \equiv 0 \pmod{N}$. Let $p_1, \ldots, p_m$ be all primes less than a parameter B. If $p-1$ has all of its factors $\le B$ then $2^{p_1^{a_1} \cdots p_m^{a_m}} - 1 \equiv 0 \pmod{N}$ where $a_i = \lfloor \log B / \log p_i \rfloor$. Hence $GCD(2^{p_1^{a_1} \cdots p_m^{a_m}} - 1, N)$ will likely yield a factor of N.

5. Let $F_1, F_2, \ldots$ be the Fibonacci numbers. If m divides n then F_m divides F_n. Hence, if you are given n and F_n, to factor it you need only factor $n = ab$ and find F_a.

Chapter 6 presents a sequence of algorithms that use continued fractions to factor. The basic idea is to find x, y, z such that $(x+by)^2 - Ny^2 = z^2$. From this one easily gets $(x+by-z)(x+by-z) = Ny^2$ so $GCD(x+by-z, N)$ is a likely factor of N. Such an x, y, z can be found by looking at convergents of continued fractions. Getting this to actually work takes many more ideas. The algorithm was first proposed in 1970. A runtime of $e^{\sqrt{2(\ln N)(\ln\ln N)}}$ was proven by Pomerance in 1982. Note that, for all ϵ, this is $\le O(N^{\epsilon})$.

Chapter 7 uses Elliptic curve methods to factor. One of them takes Pollard's $p-1$ method and instead of using the integers mod p uses a set of elliptic curves of about the right size. This increases the chance of success.

Chapter 8 is on Sieve methods- both the Quadratic Sieve and the Number Field Sieve. The latter is the method of choice for large number factorization. Both use ideas from Fermat's difference method and the Trivial method but of course use much more sophisticated ideas as well.

Chapter 9 and 10 are a collection of chapters on practical and theoretical factoring, though mostly practical. There is also a discussion of why there have not been truly new factoring algorithms since 1995.

3 Opinion

This book has lots of information and lots of pointers to more information. The field is vast and spans both practical and theoretical concerns; hence I suspect anyone who is not an active researcher in the area will find things in this book of interest. This should certainly be read by people in cryptography to get a better sense of just how vulnerable their systems might be.

While there are chapters to teach some number theory, the reader really should already know some number theory. The book could be read by bright undergraduates. Good projects could be devised by coding up some of these algorithms.