Last revised 3/28/06

LECTURE NOTES ON QUANTUM COMPUTATION

Cornell University, Physics 481-681, CS 483; Spring, 2006

III. Breaking RSA Encryption with a Quantum Computer: Shor's Factoring Algorithm

In Simon's problem we are presented with a subroutine which calculates a function $f(x)$. We are told that f satisfies $f(x) = f(y)$ for distinct x and y if and only if $y = x \oplus a$, where $\oplus$ denotes the bitwise modulo-2 sum of the n-bit integers a and x. The number of times a classical computer must invoke the subroutine to determine a grows exponentially with n, but with a quantum computer it grows only linearly.

This is a rather artificial problem, of interest primarily because it gives a simple demonstration of the remarkable computational power a quantum computer can possess. Simon's problem amounts to finding the unknown period of a function on n-bit integers that is "periodic" under bitwise modulo-2 addition. A much more natural problem is to find the period r of a function on the integers that is periodic under *ordinary* addition. Such a function f satisfies $f(x) = f(y)$ for distinct x and y if and only if x and y differ by an integral multiple of r. Finding the period of such a periodic function turns out to be the key to factoring products of large prime numbers, a mathematically natural problem with quite practical applications.

One might think that finding the period of a periodic function ought to be easy, but that is only because when one thinks of periodic functions one tends to think of slowly varying continuous functions (like the sine function) whose structure at a small subset of points within a period can give powerful clues about what that period might be. A better kind of periodic function to keep in mind from the beginning is a function on the integers whose values within a period r are completely random, and therefore give no hint whatever of the value of r.

The best known classical algorithms for finding the period r of such a periodic function take a time that grows faster than any power of the number n of bits of r (exponentially with $n^{1/3}$). But in 1994 Peter Shor discovered that one can exploit the power of a quantum computer to learn the period r, with probability arbitrarily close to one, in a time that scales only a little faster than n^3.

Shor's discovery is of considerable practical interest because the ability to find periods efficiently, combined with some number-theoretic tricks, enables one to factor efficiently the product of two large prime numbers. The very great computational effort required by all known classical factorization techniques underlies the security of the widely used

RSA[1] method of encryption. Any computer that can efficiently find periods would be an enormous threat to the security of both military and commercial comunications. This is why research into the feasibility of quantum computers is a matter of considerable interest in the worlds of war and business.

A. Number theoretic preliminaries

Although the elementary number-theoretic tricks that underly the RSA method of encryption have nothing directly to do with how a quantum computer finds periods, they motivate the problem that Shor's quantum-computational algorithm so effectively solves. Furthermore, examining the number-theoretic basis of RSA encryption reveals that Shor's period-finding algorithm can be used to decode it directly, without having to take a detour into factorization. We therefore defer the additional number-theoretic connection between period finding and factoring to Section H and Appendices A3 and A4. If you are interested only in applying Shor's period-finding algorithm to decoding RSA encryption, these can be skipped. If you are not interested in this application of period-finding to espionage and commerce, you can also skip the number theory here and in section B, and go directly to the quantum-computational part of the problem — super-efficient period finding — in section C.

The basic algebraic entities that underly RSA encryption are finite groups, where the group operation is multiplication modulo some fixed integer N. In modulo-N arithmetic all integers that differ by multiples of N are identified, so there are only N distinct quantities which can be represented by $0, 1, \dots N-1$. For example $5 \times 6 \equiv 2 \pmod 7$ since $5 \times 6 = 30 = 4 \times 7 + 2$. One writes $\equiv \pmod N$ to emphasize that the equality is only up to a multiple of N, reserving $=$ for strict equality. One can develop the results that follow using ordinary arithmetic rather than group theory, but the group theoretic approach is simpler and uses properties of groups so elementary that they can all be derived from the basic definitions in a single page. This is done in Appendix A1.

Let G_N be the set of all positive integers less than N (including 1) that have no factors in common with N. Since factoring into primes is unique, the product of two numbers in G_N (either the ordinary or the modulo-N product) also has no factors in common with N, so G_N is closed under multiplication modulo N. If a, b, and c are in G_N with $ab \equiv ac \pmod N$, then $a(b-c)$ is a multiple of N, and since a has no factors in common with N, it must be that $b-c$ is a multiple of N, so $b \equiv c \pmod N$. It follows that multiplication modulo N by a fixed member a of G_N simply permutes all the members of

[1] Named for the people who invented it in 1977, Ronald Rivest, Adi Shamir, and Leonard Adleman. RSA encryption was independently invented by Clifford Cocks four years earlier, but his discovery was classified top secret by British Intelligence and he was not allowed to reveal his priority until 1997. For this and other fascinating tales about cryptography, see Simon Singh, *The Code Book*, Doubleday, New York, 1999.

the finite set G_N. Since 1 is a member of G_N, there must be some d in G_N satisfying $ad = 1$ — i.e. a must have a multiplicative inverse in G_N. Thus G_N satisfies the conditions, listed in Appendix A1, for it to be a group under modulo-N multiplication.

Every member a of a group G is characterized by its *order* k, the smallest integer for which (in the case of G_N)

$$a^k \equiv 1 \pmod{N}. \tag{3.1}$$

As shown in Appendix A1, the order of every member of G is a divisor of the number of members of G, which is called the *order* of G. If p is a prime number, then the group G_p contains $p-1$ numbers, since no positive integer less than p has factors in common with p. Since $p-1$ is then a multiple of the order k of any a in G_p, it follows from (3.1) that any integer a less than p satisfies

$$a^{p-1} \equiv 1 \pmod{p}. \tag{3.2}$$

This relation, known as *Fermat's little theorem,* extends to arbitrary integers a not divisible by p, since any such a is of the form $a = mp + a'$ with m an integer and a' less than p.

RSA encryption exploits an extension of Fermat's little theorem to a case characterized by *two* distinct primes, p and q. If an integer a is divisible by neither p nor q, then no power of a is divisible by either p or q. Since, in particular, a^{q-1} is not divisible by p, we conclude from (3.2) that

$$[a^{q-1}]^{p-1} \equiv 1 \pmod{p}. \tag{3.3}$$

For the same reason

$$[a^{p-1}]^{q-1} \equiv 1 \pmod{q}. \tag{3.4}$$

The relations (3.3) and (3.4) state that there are integers m and n for which

$$a^{(q-1)(p-1)} = 1 + mp \tag{3.5}$$

and

$$a^{(q-1)(p-1)} = 1 + nq. \tag{3.6}$$

Taken together (3.5) and (3.6) require that $mp = nq$, which is possible with p and q distinct primes only if m is a multiple of q, and n is the same multiple of p. So

$$a^{(q-1)(p-1)} = 1 + kpq, \tag{3.7}$$

and therefore

$$a^{(q-1)(p-1)} \equiv 1 \pmod{pq}, \tag{3.8}$$

for any integer a divisible by neither of the distinct primes p and q.

As an alternative derivation of (3.8), note that since a is divisible by neither p nor q, it has no factors in common with pq and is therefore in G_{pq}. The number of elements of

G_{pq} is $pq-1-(p-1)-(q-1)=(p-1)(q-1)$, since there are $pq-1$ integers less than pq among which are $p-1$ multiples of q and another $q-1$ multiples of p. Eq. (3.8) follows because the order $(p-1)(q-1)$ of G_{pq} must be a multiple of the order of a.

We get the version of (3.8) that is the basis for RSA encryption by taking any integral power s of (3.8) and multiplying both sides by a:

$$a^{1+s(q-1)(p-1)} \equiv a \pmod{pq}. \tag{3.9}$$

(An unimportant remark: the relation (3.9), unlike (3.8), holds even for integers a that are divisible by p or q. It holds trivially when a is a multiple of pq. And if a is divisible by just one of p or q, let $a=kq$. Since a is not divisible by p neither is any power of a, and therefore Fermat's little theorem tells us that $[a^{s(q-1)}]^{p-1}=1+np$ for some integer n. Multiplying both sides by a we have $a^{1+s(q-1)(p-1)} \equiv a+nap \equiv a+nkqp$, so (3.9) continues to hold.)

Note finally that if c is an integer having no factor in common with $(p-1)(q-1)$ then c is in $G_{(p-1)(q-1)}$ and therefore has an inverse in $G_{(p-1)(q-1)}$; i.e. there is a d in $G_{(p-1)(q-1)}$ satisfying

$$cd \equiv 1 \big(\mathrm{mod}\ (p-1)(q-1)\big). \tag{3.10}$$

So for some integer s,

$$cd = 1+s(p-1)(q-1). \tag{3.11}$$

In view of (3.11) and (3.9), any integer a must satisfy

$$a^{cd} \equiv a \pmod{pq}. \tag{3.12}$$

So if

$$b \equiv a^c \pmod{pq}, \tag{3.13}$$

then

$$b^d \equiv a \pmod{pq}. \tag{3.14}$$

The elementary numerical facts summarized in this single paragraph constitute the entire arithmetic basis for RSA encryption.

B. RSA encryption

Bob wants to receive a message from Alice encoded so that he alone can read it. To do this he picks two large (say 200 digit) prime numbers p and q. He gives Alice, through a public channel, their product, $N=pq$ and a large encoding number c that he has picked to have no factors in common with [2] $(p-1)(q-1)$. He does *not*, however, reveal the

[2] As noted below, the probability that two random numbers have no common factors is greater than $\frac{1}{2}$, so such c are easily found. Whether two numbers do have any factors in common (and what their greatest common factor is) can be determined by a simple algorithm known to Euclid and easily executed by Bob on a classical computer. The Euclidean algorithm is described in Appendix A2.

separate values of p and q and, given the practical impossibility of factoring a 400 digit number with currently available computers, he is quite confident that neither Alice nor any eavesdropper Eve will ever be able to calculate p and q knowing only their product N. Bob, however, because he does know p and q, and therefore $(p-1)(q-1)$, can find[3] the multiplicative inverse d of c mod $(p-1)(q-1)$, satisfying (3.10). He keeps d strictly to himself for use in decoding.

Alice encodes a message by representing it as a string of fewer than 400 decimal digits using, for example, some version of ASCII coding. If her message requires more than 400 digits she chops it up into smaller pieces. She interprets each such string as a decimal number a less than N. Using the coding number c and the value of $N = pq$ she received from Bob, she then calculates $b \equiv a^c \pmod{pq}$, and sends it on to Bob through a public channel. With c typically a 200 digit number, you might think that this would itself be a huge computational task, but it is not, as noted below. When he receives b, Bob exploits his private knowledge of d to calculate $b^d \pmod{pq}$, which (3.14) assures him is Alice's original message a.

Were the eavesdropper Eve able to find the factors p and q of N, she could calculate $(p-1)(q-1)$ and find the decoding integer d from the publicly available coding integer c, the same way Bob did. But factoring a number as large as N is far beyond her classical computational powers. Period finding is of interest in this cryptographic setting not only because it can form the basis for efficient factoring (as described in Appendix A3), but also because it can lead Eve directly to an alternative way to decode Alice's message b without knowing or having to compute the factors p and q of N. Here is how it works:

Eve uses her efficient period finding machine to calculate the order r of Alice's publicly available encoded message $b = a^c$ in[4] G_{pq}. Since the order of any integer in G_{pq} divides the order $(p-1)(q-1)$ of G_{pq}, this gives her a *divisor* of $(p-1)(q-1)$, and it turns out that knowing such a divisor is just as useful for decoding Alice's message as is knowing $(p-1)(q-1)$ itself.

To see this note first that the order r of Alice's encoded message $b = a^c$ in G_{pq} is the same as the order of a. This is because the subgroup of G_{pq} generated by a contains $a^c = b$, and hence it contains the subgroup generated by b; but the subgroup generated by b contains $b^d = a$, and hence the subgroup generated by a. Since each subroup contains the other they must be identical. Since the order of a or b is the number of elements in the

[3] This can easily be done classically as a straightforward embellishment of the Euclidean algorithm. See Appendix A2.

[4] I assume that Alice's unencoded message a, and hence her coded message b, is in G_{pq} — i.e. that a is not a multiple of p or q. Since p and q are huge numbers the odds against a being such a multiple are astronomical. But if Eve wants to be insanely careful she can find the greatest common factor of b and N, using the Euclidean algorithm. In the grossly improbable case that it turns out not to be 1, Eve will have factored N and can decode Alice's message the samed way Bob does.

subgroup it generates, their orders are the same. So if Eve can find the order r of Alice's code message b, then she has also learned the order of Alice's original text a.

Since Bob has picked c to have no factors in common with $(p-1)(q-1)$, and since r divides $(p-1)(q-1)$, the coding integer c can have no factors in common with r. So c is congruent modulo r to a member c' of G_r, which has an inverse d' in G_r, and d' is also a modulo-r inverse of c:

$$cd' \equiv 1 \pmod{r}. \tag{3.15}$$

Therefore given c (which Bob has publicly announced) and r (which Eve can get with her period-finding program from Alice's encoded message b and the publicly announced value of $N = pq$), it is easy for Eve to calculate d' with a classical computer, using, modulo r, the same extension of the Euclidean algorithm that Bob used to find d, modulo $(p-1)(q-1)$. It then follows that for some integer m

$$b^{d'} \equiv a^{cd'} = a^{1+mr} = a\big(a^r\big)^m \equiv a \pmod{pq}. \tag{3.16}$$

Eve has thus used her ability to find periods efficiently to decode Alice's encoded message $b = a^c$ to reveal Alice's original message a.

This use of period finding to defeat RSA encryption is summarized in Table 1.

C. Quantum period-finding: setting things up

So we can crack the RSA code if we have a fast way to find the period r of the known periodic function

$$f(x) = b^x \pmod{N}. \tag{3.17}$$

As remarked earlier, this might appear to be a simple task, especially since periodic functions of the special form (3.17) have the simplifying feature that $f(x+s) = f(x)$ *only* if s is a multiple of the period r. If we can find two different values of x at which f has the same value, then we have found a multiple of the period r, and given a few random multiples of r we can, with high probability find r itself.[5] But unlike the smooth, uncomplicated periodic functions one is used to dealing with, the function $b^x \pmod{N}$ looks like random noise within a period. Its structure over any stretch of integers less than the period offers no clue about the value of that period. In trying to determine the period with a classical computer by this direct approach one can do no better than to calculate f repeatedly for

[5] Given two random multiples kr and $k'r$, if k and k' have no common factors then r is their greatest common divisor, and is easily extracted using the Euclidean algorithm. The probability of two random numbers having no common factors is greater than 1/2, for the probability that they are both divisible by any prime p is $1/p^2$, and therefore the probability that they share no prime factors at all is $\prod_{\text{primes}}(1-1/p^2) = .6079\ldots\ (= 6/\pi^2)$. So with a fairly small number of random multiples of r one can, with high probability, extract r.

BOB KNOWS	**ALICE KNOWS**	**PUBLIC KNOWS**
p and q (primes) c and d, $cd \equiv 1 \pmod{(p-1)(q-1)}$	a (her message) only c (not d) and only $N = pq$ $b \equiv a^c \pmod N$ (encoded message)	b (encoded message) only c (not d), and only $N = pq$
Decoding: $a \equiv b^d \pmod N$		**Quantum decoding:** Quantum computer finds r: $b^r \equiv 1 \pmod N$; Classical computer finds d': $cd' \equiv 1 \pmod r$; $a \equiv b^{d'} \pmod N$

Table 1. Summary of RSA encryption and how to break it with a fast period-finding routine on a quantum computer. Bob has chosen the encoding number c to have an inverse d modulo $(p-1)(q-1)$ so c can have no factors in common with $(p-1)(q-1)$. Since Alice's encoded message b is in G_{pq}, its order r is a factor of the order $(p-1)(q-1)$ of G_{pq}. So c can have no factors in common with r, and therefore has an inverse d' modulo r. Because b is a power of a and vice-versa, each has the same order r in G_{pq}. Therefore $b^{d'} \equiv a^{cd'} \equiv a^{1+mr} \equiv a$ modulo N.

a random collection of integers until one finally gets a value that agrees with one of the values already calculated.

The scale of the problem is best measured by the number of bits n_0 in $N = pq$; 2^{n_0} is the smallest power of 2 that exceeds N. If N is a 500 digit number — a typical size for cryptographic applications — n_0 will be around 1700. This sets the scale for the number of bits in N and in the other relevant numbers a, b, and their modulo N period r. To have an appreciable probability of finding r by random searching requires a number of evaluations of f that is exponential in n_0 (just like the classical situation in Simon's problem, described in Chapter 2.) Although there are ways to improve on random searching, using, for example, Fourier analysis, no classical approach is known that does not require a time that grows faster than any power of n_0. With a quantum computer, however, quantum parallelism gets us tantalizingly close (but, as in Simon's problem, not close enough) to solving the problem with a single application of $\mathbf{U}_f$, and enables us to solve it completely with probability arbitrarily close to 1 in a time that grows only as a polynomial in n_0.

To deal with values of x and $f(x)$ between 0 and N, both the input and output registers must contain at least n_0 Qbits. For reasons that will emerge in Secton E, however, to find the period r efficiently the input register must actually have $n = 2n_0$ Qbits. Doubling the number of Qbits in the input register ensures that the range of values of x for which $f(x)$ is calculated contains at least N full periods of f. This redundancy turns out to be absolutely essential for a successful determination of the period by Shor's method. (We shall see in Section E that if p and q both happen to be primes of the form $2^j + 1$ then — and only then — the method works without doubling the size of the input register. Thus $N = 15$ can be a highly atypical test case for laboratory attempts to demonstrate Shor's algorithm for small p and q with real Qbits.)

We begin by using our quantum computer in the usual way to construct the state

$$\frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle_n |f(x)\rangle_{n_0} \tag{3.18}$$

with a single application of $\mathbf{U}_f$. In Section F we take a closer look at how this might efficiently be done in the case of interest, $f(x) = b^x \pmod N$. Once the state of the registers has become (3.18), we can measure the n-Qbit output register.[6] If the measurement yields the value f_0, then the generalized Born rule tells us that the state of the n-Qbit input register can be taken to be

$$|\Psi\rangle_n = \frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} |x_0 + kr\rangle_n. \tag{3.19}$$

Here x_0 is the smallest value of x $(0 \le x_0 < r)$ at which $f(x_0) = f_0$, and m is the smallest integer for which $mr + x_0 \ge 2^n$, so

$$m = \left[\frac{2^n}{r}\right] \quad \text{or} \quad m = \left[\frac{2^n}{r}\right] + 1, \tag{3.20}$$

depending on the value of x_0 (where $[x]$ is the integral part of x). As in the examples of Chapter 2, if we could produce a small number of identical copies of the state (3.19) we would be done, for a measurement in the computational basis would yield a random one of the values $x_0 + kr$, and the difference between the results of pairs of measurements on such identical copies would give us a collection of random multiples of r from which r itself could straightforwardly be extracted, as noted above. But we are again done in by the

[6] It is not, in fact, necessary to measure the output register. One can continue to work with the full state (3.18) in which one breaks down the sum on x into a sum over all the different values of f and a sum over all the values of x associated with each value of f. The only function of the measurement is to eliminate a lot of uninteresting additional structure, coming from the sum on the values of f, that plays no role beyond making many of the subsequent expressions somewhat lengthier.

no-cloning theorem. All we can extract is a single value of $x_0 + kr$ for unknown random x_0, which is completely useless for determining r. And, of course, if we ran the whole algorithm again, we would end up with a state of the form (3.19) for another random value of x_0, which would permit no useful comparison with what we had learned from the first run.

But, as with Simon's problem, we can do something more clever to the state (3.19) before making our final measurement. The problem is the displacement by the unknown random x_0, which prevents any information about r from being extracted in a single measurement. We need a unitary transformation that transforms the x_0 dependence into a harmless overall phase factor. This is accomplished with the *Quantum Fourier Transform.*

D. The Quantum Fourier Transform

The heart of Shor's algorithm is a superfast quantum Fourier transform procedure, carried out by an efficient quantum circuit built out of one- and 2-Qbit gates. The n-Qbit quantum Fourier transform is defined to be that unitary transformation $\mathbf{U}_{FT}$ whose action on the computational basis is given by

$$\mathbf{U}_{FT}|x\rangle_n = \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} e^{2\pi i xy/2^n} |y\rangle_n. \tag{3.21}$$

The product xy is here ordinary multiplication.

A warning to physicists (which others can ignore): This looks deceptively like a (discretized) transformation from a position to a momentum representation, and one's first reaction might be that it is (perhaps disappointingly) familiar. But it has, in fact, an entirely different character. The number x is the integer represented by the state $|x\rangle$, and not the position of anything. Changing x to $x + 1$ induces an arithmetically natural but physically quite unnatural transformation on the computational basis states, determined by the laws of binary addition, including carrying. It bears no resemblance to anything that could be associated with a spatial translation in the physical space of Qbits. So your eyes should not glaze over, and you should regard $\mathbf{U}_{FT}$ as a new and unfamiliar physical transformation of Qbits.

One easily verifies that $\mathbf{U}_{FT}|x\rangle$ is normalized to unity and that $\mathbf{U}_{FT}|x\rangle$ is orthogonal to $\mathbf{U}_{FT}|x'\rangle$ unless $x = x'$, so $\mathbf{U}_{FT}$ is unitary. Unitarity also emerges directly from the analysis that follows, which explicitly constructs $\mathbf{U}_{FT}$ out of one- and 2-Qbit unitary gates. The unitary $\mathbf{U}_{FT}$ is useful because, as one also easily verifies, applied to a superposition of states $|x\rangle$ with complex amplitudes $\gamma(x)$,it produces another superposition with amplitudes that are related to $\gamma(x)$ by the appropriate discrete Fourier transform:

$$\mathbf{U}_{FT}\Big(\sum_{x=0}^{2^n-1} \gamma(x)|x\rangle\Big) = \sum_{x=0}^{2^n-1} \tilde{\gamma}(x)|x\rangle, \tag{3.22}$$

where

$$\tilde{\gamma}(x) = \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} e^{2\pi i x y/2^n} \gamma(y). \tag{3.23}$$

The celebrated classical fast Fourier transform is an algorithm requiring a time that grows with the number of bits as $n2^n$ (rather than $\big(2^n\big)^2$ as the obvious direct approach would require) to evaluate $\tilde{\gamma}$. But there is a quantum algorithm for executing the unitary transformation $\mathbf{U}_{FT}$ exponentially faster than fast, in a time that grows only as n^2. The catch as usual, is that one does not end up knowing the complete set of Fourier coefficients, as one does after applying the classical fast Fourier transform. One only has n Qbits described by the state given by the right side of (3.22), and as we have repeatedly noted, having a collection of Qbits in a given state does not enable one to learn what that state actually is. There is no way to extract all the Fourier coefficients $\tilde{\gamma}$, given an n-Qbit register in the state (3.22). Nevertheless, we shall see in Section E that if γ is a periodic function with a period that is no bigger than $2^{n/2}$, then a register in the state (3.22) can give powerful clues about the precise value of the period r, even though r can be hundreds of digits long,

Notice the resemblance of the quantum Fourier transform (3.21) to the n-fold Hadamard transformation (Eq. (2.29)). Since $-1 = e^{\pi i}$, the the n-fold Hadamard assumes the form

$$\mathbf{H}^{\otimes n}|x\rangle_n = \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} e^{\pi i x\cdot y}|y\rangle_n. \tag{3.24}$$

Aside from the different powers of 2 appearing in the quantum Fourier transform (3.21) — so the factors of modulus 1 in the superposition are not just 1 and -1 — the only other difference between the two transforms is that xy is ordinary multiplication in the quantum Fourier transform, while $x\cdot y$ is the bitwise inner product in the n-fold Hadamard. Because the arithmetic product xy is a more elaborate function of x and y than $x \cdot y$, the quantum Fourier transformation cannot be built entirely out of 1-Qbit unitary gates as the n-fold Hadamard is. But, remarkably, it can be constructed entirely out of 1-Qbit and 2-Qbit gates. Even more remarkably we shall see that under certain conditions of practical interest, all of the 2-Qbit gates can be replaced by 1-Qbit measurement gates followed by additional 1-Qbit unitary gates whose application is contingent on the measurement outcomes.

To construct such a circuit to execute the quantum Fourier transform (QFT) $\mathbf{U}_{FT}$, it is convenient to introduce an n-Qbit unitary operator $\mathcal{Z}$, diagonal in the computational basis:

$$\mathcal{Z}|y\rangle_n = e^{2\pi i y/2^n}|y\rangle_n. \tag{3.25}$$

This can be viewed as a generalization to n Qbits of the 1-Qbit operator $\mathbf{Z}$, to which it reduces when $n = 1$. Using the familiar relation

$$\mathbf{H}^{\otimes n}|0\rangle_n = \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} |y\rangle_n, \tag{3.26}$$

we can reexpress the definition (3.21) as

$$\mathbf{U}_{FT}|x\rangle_n = \mathcal{Z}^x \mathbf{H}^{\otimes n}|0\rangle_n. \tag{3.27}$$

This gives $\mathbf{U}_{FT}|x\rangle_n$ as an x-dependent operator acting on the state $|0\rangle$.

We next reexpress the right side of (3.27) as an x-independent operator acting on the state $|x\rangle_n$. Since the computational basis states $|x\rangle_n$ are a basis, we will then have found an alternative expression for $\mathbf{U}_{FT}$ itself. The construction of this alternative form for (3.27) is made much more transparent by specializing to the case of 4 Qbits. We will find that the structure that emerges in the case $n = 4$ has an entirely obvious extension to general n; dealing with the case of general n from the start would only obscure this simple structure.

When $n = 4$ we want to find an appropriate form for

$$\mathbf{U}_{FT}|x_3\rangle|x_2\rangle|x_1\rangle|x_0\rangle = \mathcal{Z}^x \mathbf{H}_3\mathbf{H}_2\mathbf{H}_1\mathbf{H}_0|0\rangle|0\rangle|0\rangle|0\rangle. \tag{3.28}$$

As usual, we number the Qbits by the power of 2 with which they are associated, with the least significant on the right, so that, reading from right to left, the Qbits are labeled 0, 1, 2, and 3; $\mathbf{H}_i$ acts on the Qbit labeled i (and as the identity on all other Qbits). If $|y\rangle_4 = |y_3\rangle|y_2\rangle|y_1\rangle|y_0\rangle$ in (3.25) so that $y = 8y_3 + 4y_2 + 2y_1 + y_0$, then the operator $\mathcal{Z}$ can be constructed out of single-Qbit number operators:

$$\mathcal{Z} = \exp\big[\tfrac{i\pi}{8}\big(8\mathbf{n}_3 + 4\mathbf{n}_2 + 2\mathbf{n}_1 + \mathbf{n}_0\big)\big]. \tag{3.29}$$

The operator $\mathcal{Z}^x$ appearing in (3.28) then becomes

$$\mathcal{Z}^x = \exp\big[\tfrac{i\pi}{8}\big(8x_3 + 4x_2 + 2x_1 + x_0\big)\big(8\mathbf{n}_3 + 4\mathbf{n}_2 + 2\mathbf{n}_1 + \mathbf{n}_0\big)\big]. \tag{3.30}$$

Because the 1-Qbit operator $\exp(2\pi i\mathbf{n})$ acts as the identity on either of the 1-Qbit states $|0\rangle$ or $|1\rangle$, and because any 1-Qbit state is a superposition of these two, $\mathbf{n}$ obeys the operator identity

$$\exp(2\pi i\mathbf{n}) = \mathbf{1}. \tag{3.31}$$

Therefore in multiplying out the two terms

$$\big(8x_3 + 4x_2 + 2x_1 + x_0\big)\big(8\mathbf{n}_3 + 4\mathbf{n}_2 + 2\mathbf{n}_1 + \mathbf{n}_0\big) \tag{3.32}$$

appearing in the exponential (3.30), we can drop those whose coefficient is a power of 2 greater than 8, getting

$$\mathcal{Z}^x = \exp\bigl[i\pi\bigl(x_0\mathbf{n}_3+(x_1+\tfrac{1}{2}x_0)\mathbf{n}_2+(x_2+\tfrac{1}{2}x_1+\tfrac{1}{4}x_0)\mathbf{n}_1+(x_3+\tfrac{1}{2}x_2+\tfrac{1}{4}x_1+\tfrac{1}{8}x_0)\mathbf{n}_0\bigr)\bigr]. \quad (3.33)$$

Note next that the number and Hadamard operators for any single Qbit obey the relation

$$\exp(i\pi x\mathbf{n})\mathbf{H}|0\rangle = \mathbf{H}|x\rangle. \quad (3.34)$$

This is trivial when $x = 0$, and when $x = 1$ it reduces to the correct statement

$$(-1)^{\mathbf{n}}\tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \quad (3.35)$$

(Alternatively, note that $\exp(i\pi\mathbf{n}) = \mathbf{Z}$ and $\mathbf{ZH} = \mathbf{HX}$.) The effect on (3.28) of the four terms in (3.33) that do not contain factors of $\frac{1}{2}$, $\frac{1}{4}$, or $\frac{1}{8}$ is to produce the generalization of (3.34) to several Qbits:[7]

$$\begin{aligned}
&\exp\bigl[i\pi\bigl(x_0\mathbf{n}_3 + x_1\mathbf{n}_2 + x_2\mathbf{n}_1 + x_3\mathbf{n}_0\bigr)\bigr]\mathbf{H}_3\mathbf{H}_2\mathbf{H}_1\mathbf{H}_0|0\rangle|0\rangle|0\rangle|0\rangle \\
&= \bigl[\exp\bigl(i\pi x_0\mathbf{n}_3\bigr)\mathbf{H}_3\bigr]\bigl[\exp\bigl(i\pi x_1\mathbf{n}_2\bigr)\mathbf{H}_2\bigr]\bigl[\exp\bigl(i\pi x_2\mathbf{n}_1\bigr)\mathbf{H}_1\bigr]\bigl[\exp\bigl(i\pi x_3\mathbf{n}_0\bigr)\mathbf{H}_0\bigr]|0\rangle|0\rangle|0\rangle|0\rangle \\
&= \mathbf{H}_3\mathbf{H}_2\mathbf{H}_1\mathbf{H}_0|x_0\rangle|x_1\rangle|x_2\rangle|x_3\rangle. \quad (3.36)
\end{aligned}$$

(Note the inversion: because the number operator $\mathbf{n}_i$ is multiplied by x_{3-i} on the left side of (3.36), the state of the Qbit labeled i on the right is $|x_{3-i}\rangle$.)

The remaining six terms in (3.33) (containing fractional coefficients) further convert (3.28) to the form

$$\begin{aligned}
&\mathbf{U}_{FT}|x_3\rangle|x_2\rangle|x_1\rangle|x_0\rangle = \\
&\exp\bigl[i\pi\bigl(\tfrac{1}{2}x_0\mathbf{n}_2+(\tfrac{1}{2}x_1+\tfrac{1}{4}x_0)\mathbf{n}_1+(\tfrac{1}{2}x_2+\tfrac{1}{4}x_1+\tfrac{1}{8}x_0)\mathbf{n}_0\bigr)\bigr]\mathbf{H}_3\mathbf{H}_2\mathbf{H}_1\mathbf{H}_0|x_0\rangle|x_1\rangle|x_2\rangle|x_3\rangle. \quad (3.37)
\end{aligned}$$

Since the Hadamard transformation $\mathbf{H}_i$ commutes with the number operator $\mathbf{n}_j$ when $i \neq j$, we can regroup the terms in (3.37) so that each number operator $\mathbf{n}_i$ appears immediately to the left of its corresponding Hadamard operator $\mathbf{H}_i$:

$$\begin{aligned}
\mathbf{U}_{FT}|x_3\rangle|x_2\rangle|x_1\rangle|x_0\rangle &= \mathbf{H}_3\exp\bigl[i\pi\mathbf{n}_2\tfrac{1}{2}x_0\bigr]\mathbf{H}_2\exp\bigl[i\pi\mathbf{n}_1(\tfrac{1}{2}x_1 + \tfrac{1}{4}x_0)\bigr]\mathbf{H}_1\times \\
&\times\exp\bigl[i\pi\mathbf{n}_0(\tfrac{1}{2}x_2 + \tfrac{1}{4}x_1 + \tfrac{1}{8}x_0)\bigr]\mathbf{H}_0|x_0\rangle|x_1\rangle|x_2\rangle|x_3\rangle. \quad (3.38)
\end{aligned}$$

The state $|x_0\rangle|x_1\rangle|x_2\rangle|x_3\rangle$ is an eigenstate of the number operators $\mathbf{n}_3, \mathbf{n}_2, \mathbf{n}_1, \mathbf{n}_0$ with respective eigenvalues x_0, x_1, x_2, x_3. If we did not have to worry about Hadamard operators interposing themselves between number operators and their eigenstates, we could replace each x_i in (3.38) by the number operator $\mathbf{n}_{3-i}$ of which it is the eigenvalue to get:

[7] We use the fact that number operators associated with different Qbits comute with one another. Since $\mathbf{n}_i$ has the coefficient x_{3-i}, the Qbit labeled i in the final form is in the state $|x_{3-i}\rangle$.

$$\mathbf{U}_{FT}|x_3\rangle|x_2\rangle|x_1\rangle|x_0\rangle = \mathbf{H}_3 \exp\bigl[i\pi\tfrac{1}{2}\mathbf{n}_2\mathbf{n}_3\bigr]\mathbf{H}_2 \exp\bigl[i\pi\mathbf{n}_1(\tfrac{1}{2}\mathbf{n}_2+\tfrac{1}{4}\mathbf{n}_3)\bigr]\times$$
$$\times\mathbf{H}_1 \exp\bigl[i\pi\mathbf{n}_0(\tfrac{1}{2}\mathbf{n}_1+\tfrac{1}{4}\mathbf{n}_2+\tfrac{1}{8}\mathbf{n}_3)\bigr]\mathbf{H}_0|x_0\rangle|x_1\rangle|x_2\rangle|x_3\rangle. \qquad (3.39)$$

But as (3.39) makes clear, we do indeed not have to worry, because every $\mathbf{H}_i$ appears safely to the *left* of every $\mathbf{n}_i$ that has replaced an x_{3-i}.

If we define 2-Qbit unitary operators by

$$\mathbf{V}_{ij} = \exp\bigl(i\pi\mathbf{n}_i\mathbf{n}_j/2^{|i-j|}\bigr), \qquad (3.40)$$

then (3.39) assumes the more readable form

$$\mathbf{U}_{FT}|x_3\rangle|x_2\rangle|x_1\rangle|x_0\rangle = \mathbf{H}_3\bigl(\mathbf{V}_{32}\mathbf{H}_2\bigr)\bigl(\mathbf{V}_{31}\mathbf{V}_{21}\mathbf{H}_1\bigr)\bigl(\mathbf{V}_{30}\mathbf{V}_{20}\mathbf{V}_{10}\mathbf{H}_0\bigr)|x_0\rangle|x_1\rangle|x_2\rangle|x_3\rangle. \qquad (3.41)$$

(I have put in unnecessary parenthesis to guide the eye to the simple structure, whose generalization to more than four Qbits is, as promised, obvious.)

If we define the unitary operator $\mathbf{P}$ to bring about the permutation of computational basis states

$$\mathbf{P}|x_3\rangle|x_2\rangle|x_1\rangle|x_0\rangle = |x_0\rangle|x_1\rangle|x_2\rangle|x_3\rangle \qquad (3.42)$$

then (3.41) becomes

$$\mathbf{U}_{FT}|x_3\rangle|x_2\rangle|x_1\rangle|x_0\rangle = \mathbf{H}_3\bigl(\mathbf{V}_{32}\mathbf{H}_2\bigr)\bigl(\mathbf{V}_{31}\mathbf{V}_{21}\mathbf{H}_1\bigr)\bigl(\mathbf{V}_{30}\mathbf{V}_{20}\mathbf{V}_{10}\mathbf{H}_0\bigr)\mathbf{P}|x_3\rangle|x_2\rangle|x_1\rangle|x_0\rangle. \qquad (3.43)$$

Since (3.43) holds for all computational basis states it holds for arbitrary states and is therefore equivalent to the operator identity

$$\mathbf{U}_{FT} = \mathbf{H}_3(\mathbf{V}_{32}\mathbf{H}_2)(\mathbf{V}_{31}\mathbf{V}_{21}\mathbf{H}_1)(\mathbf{V}_{30}\mathbf{V}_{20}\mathbf{V}_{10}\mathbf{H}_0)\mathbf{P}. \qquad (3.44)$$

The form (3.44) expresses $\mathbf{U}_{FT}$ as a product of unitary operators, thereby independently establishing what we already noted directly from its definition, that $\mathbf{U}_{FT}$ is unitary. More importantly it gives an explicit construction of $\mathbf{U}_{FT}$ entirely out of *one*- and *two*-Qbit unitary gates, whose number grows only quadratically with the number n of Qbits. (The permutation $\mathbf{P}$ can be constructed out of cNOT gates and one additional Qbit, initially in the state $|0\rangle$ — an instructive exercise to think about — but in the application that follows it is much easier to build directly into the circuitry the rearranging of Qbits accomplished by $\mathbf{P}$.)

A circuit diagram that compactly expresses the content of (3.43) is shown in Figure 3.1. Note that the diagram introduces an artificial asymmetry into the 2-Qbit unitary gate $\mathbf{V}_{ij}$, by treating one Qbit as a control bit, which determines whether or not the unitary operator $e^{i\pi\mathbf{n}/2^{|i-j|}}$ acts on the target Qbit. Although this is the conventional way of representing the circuit for the quantum Fourier transform, the figure could equally well have been drawn as it is in Figure 3.2. This second, less conventional form, reveals a further simplification of great practical interest, if all the Qbits are measured as soon as the action of the quantum Fourier transformation is completed. The simplification allows the 2-Qbit controlled-V gates to be replaced by single-Qbit gates that act or not, depending on the outcome of a prior measurement of the control Qbit. This is described in the caption of Figure 3.2.

This replacement of controlled operators (2-Qbit operators) by measurements followed by 1-Qbit operators is shown explicitly in Figure 3.3. It is made possible by the (easily verified) fact that if a controlled operation, or a series of consecutive controlled operations all with the same control Qbit, is immediately followed by a measurement of the control Qbit, then the possible final states of all the Qbits and the probabilities of those states are exactly the same as they would be if the measurement of the control Qbit took place before the application of the controlled operation, and the target Qbit(s) were then acted upon or not, depending on whether the result of the prior measurement was 0 or 1.

We shall see that if one's aim is to find the period of the function f, one can indeed measure each Qbit immediately after applying the quantum Fourier transform. So this replacement of controlled unitary gates by 1-Qbit unitary gates, which act or not depending on the outcome of the measurement, is a major simplification from the technological point of view, 1-Qbit unitaries being far easier to implement than 2-Qbit controlled gates.

The most attractive (but least common) way of representing the quantum Fourier transform with a circuit diagram is shown in Figure 3.4. In this form the inversion in order from most to least significant Qbits between the input and the output is shown by bending the Qbit lines, rather than by inverting the order in the state symbols. The 2-Qbit gates V are also displayed in a way that does not suggesting a nonexistent asymmetry between control and target Qbits.

The permutation operator P plays a crucial role in establishing the operator form of the inverse Fourier transform operator $\mathsf{U}_{FT}^\dagger$. Since the adjoint of a product is the product of the adjoints in the opposite order, and since Hadamards and P are self-adjoint, we have from (3.44)

$$\mathsf{U}_{FT}^\dagger = \mathsf{P}(\mathsf{H}_0\mathsf{V}_{10}^\dagger\mathsf{V}_{20}^\dagger\mathsf{V}_{30}^\dagger)(\mathsf{H}_1\mathsf{V}_{21}^\dagger\mathsf{V}_{31}^\dagger)(\mathsf{H}_2\mathsf{V}_{32}^\dagger)\mathsf{H}_3. \tag{3.45}$$

One can insert $\mathbf{1} = \mathsf{PP}$ on the extreme right of (3.45) and then note that the effect of sandwiching all the Hadamard's and 1-Qbit unitaries between two P's is simply to alter all their indices by the permutation taking 0123→3210. Therefore

$$\mathsf{U}_{FT}^\dagger = (\mathsf{H}_3\mathsf{V}_{23}^\dagger\mathsf{V}_{13}^\dagger\mathsf{V}_{03}^\dagger)(\mathsf{H}_2\mathsf{V}_{12}^\dagger\mathsf{V}_{02}^\dagger)(\mathsf{H}_1\mathsf{V}_{01}^\dagger)\mathsf{H}_0\mathsf{P}. \tag{3.46}$$

If we now move every $\mathsf{V}^\dagger$ to the right past as many Hadamard's as we can, keeping in mind that each V commutes with all Hadamards except those sharing either of its indices, then we have

$$\mathsf{U}_{FT}^\dagger = (\mathsf{H}_3\mathsf{V}_{23}^\dagger)(\mathsf{H}_2\mathsf{V}_{13}^\dagger\mathsf{V}_{12}^\dagger)(\mathsf{H}_1\mathsf{V}_{03}^\dagger\mathsf{V}_{02}^\dagger\mathsf{V}_{01}^\dagger)\mathsf{H}_0\mathsf{P}. \tag{3.47}$$

Finally, if we note from (3.40) that each $\mathbf{V}$ is symmetric in its indices, and rearrange the parentheses in (3.47) to make the comparison with (3.44) easier, we have

$$\mathbf{U}_{FT}^\dagger = \mathbf{H}_3(\mathbf{V}_{32}^\dagger\mathbf{H}_2)(\mathbf{V}_{31}^\dagger\mathbf{V}_{21}^\dagger\mathbf{H}_1)(\mathbf{V}_{30}^\dagger\mathbf{V}_{20}^\dagger\mathbf{V}_{10}^\dagger\mathbf{H}_0)\mathbf{P}. \tag{3.48}$$

This is precisely the form (3.44) of $\mathbf{U}_{FT}$ itself, except that each $\mathbf{V}$ is replaced by its adjoint, which (3.40) shows amounts to replacing each i by $-i$ in the arguments of all the phase factors. This is exactly what one does to invert the ordinary functional Fourier transform.

E. Getting the period from the quantum Fourier transform

As noted above, the spectacular increase in computational speed of the quantum Fourier transform over the classical fast Fourier transform is undermined by the usual proviso: having a specimen of a system described by the state (3.22) does not enable one to learn what the state is, and therefore there is no way to learn what all the Fourier-transformed amplitudes $\tilde{\gamma}$ are. But if $\gamma(x)$ has a period r which is smaller than $2^{n/2}$, then if we have a small number of specimens of states of the form (3.22), even though we cannot learn the values of the $\tilde{\gamma}$, we can, with high probability, learn the value of the period r.

The period r of f appears in the state (3.19) of the input-register Qbits produced from a single application of $\mathbf{U}_f$. To get valuable information about r we apply the quantum Fourier transformation (3.21) to the input register:

$$\mathbf{U}_{FT}\frac{1}{\sqrt{m}}\sum_{k=0}^{m-1}|x_0+kr\rangle = \frac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}\frac{1}{\sqrt{m}}\sum_{k=0}^{m-1}e^{2\pi i(x_0+kr)y/2^n}|y\rangle =$$

$$\sum_{y=0}^{2^n-1}e^{2\pi i x_0 y/2^n}\frac{1}{\sqrt{2^n m}}\Big(\sum_{k=0}^{m-1}e^{2\pi i k r y/2^n}\Big)|y\rangle. \tag{3.49}$$

If we now make a measurement, the probability $p(y)$ of getting the result y is just the squared magnitude of the coefficient of $|y\rangle$ in (3.49). The factor $e^{2\pi i x_0 y/2^n}$, in which the formerly troublesome x_0 explicitly occurs, drops out of this probability[8] and we are left with

$$p(y) = \frac{1}{2^n m}\Big|\sum_{k=0}^{m-1}e^{2\pi i k r y/2^n}\Big|^2. \tag{3.50}$$

This completes the quantum mechanical part of the process, except that, as noted below, we may have to repeat the procedure a small number of times (of order 10 or so) to achieve a high probability of learning the period r. To see why this is so, we require some

[8] The random value of $x_0 < r$ also determines whether m is given by rounding the enormous value of $2^n/r$ up or down to the nearest integer — see Eq. (3.20) and the surrounding text — but this turns out to be of negligible importance.

more purely mathematical analysis, that includes the use of another branch of elementary number theory.

The probability (3.50) is a simple explicit function of y, whose magnitude has maxima when y is close[9] to integral multiples of $2^n/r$. Indeed we now show that the probability is at least 40% that the measured value of y will be as close as possible to — i.e. within $\frac{1}{2}$ of — an integral multiple of $2^n/r$. To see this we calculate a lower bound for $p(y)$ when

$$y = y_j = j\,2^n/r + \delta_j, \tag{3.51}$$

with $|\delta_j| \leq 1/2$. Only the term in δ_j contributes to the exponentials in (3.50). The summation is a geometric series which can be explicitly summed to give

$$p(y_j) = \frac{1}{2^n m}\frac{\sin^2(\pi\delta_j mr/2^n)}{\sin^2(\pi\delta_j r/2^n)}. \tag{3.52}$$

Since (3.20) tells us that m is within an integer of $2^n/r$, and since $2^n/r \geq N^2/r > N$, we can with negligible error replace $mr/2^n$ by 1 in the numerator of (3.52), and replace the sine in the denominator by its (extremely small) argument. This gives

$$p(y_j) = \frac{1}{2^n m}\Big(\frac{\sin(\pi\delta_j)}{\pi\delta_j r/2^n}\Big)^2 = \frac{1}{r}\Big(\frac{\sin(\pi\delta_j)}{\pi\delta_j}\Big)^2. \tag{3.53}$$

When x is between 0 and $\pi/2$, the graph of $\sin x$ lies above the straight line connecting the origin to the maximum at $x = \pi/2$:

$$x/\tfrac{1}{2}\pi \leq \sin x, x \quad 0 \leq x \leq \pi/2. \tag{3.54}$$

Since $\delta_j \leq \frac{1}{2}$ we can therefore bound the probability (3.53) below by

$$p(y_j) \geq (4/\pi^2)/r. \tag{3.55}$$

Since there are at least $r-1$ different values of j, and since r is a large number[10] one has at least a 40% chance ($4/\pi^2 = .4053$) of getting one of the special values (3.51) for y — a value that is within 1/2 of an integral multiple of $2^n/r$.

[9] Such sums of phase factors are familiar to physicists (to whom this cautionary footnote is addressed), particularly in the context of time-dependent perturbation theory, where one becomes quite used to waving ones hands and treating them as Dirac delta-functions concentrated in the neighborhood of the maximum values. Physicists should not let this familiarity distract them from the fact that the analysis here is significantly different. Because we require highly precise information about the integer r, we must pay much more careful attention to just how much of the probability is concentrated in those special values of y. We must also worry about how to get from such maximum values to the period r itself.

[10] We can easily test with a classical computer all values of r less than, say, a hundred, to see if they are periods of f; one only need resort to the quantum computation if r itself is enormous.

Note, in passing, that as $\delta_j \to 0$ in (3.53) the probability $p(y_j)$ becomes $1/r$, so that if all the δ_j are 0 — i.e. if the period r is exactly a power of 2 — then the probability of measuring an integral multiple of $2^n/r$ is essentially 1. Indeed, you can easily check that in this (highly unlikely) case the probability remains 1 even if we do not double the number of Qbits in the input register and take $n = n_0$. Thus the case $r = 2^j$ avoids many of the mathematical and practical (i.e. having to double the size of the input register) difficulties of quantum period finding. Since r divides $(p-1)(q-1)$, all periods modulo pq will necessarily be powers of 2 if p and q are both primes of the form $2^n + 1$. The smallest such primes are 3, 5, 17, and 257. Hence claims to have realized the Shor algorithm for factoring 15 are to be taken *cum grano salis*, as should possible future claims to have factored 51, 85, and 771.

Note also that the derivation of (3.55) only requires the argument of the sine in the denominator of (3.52) to be small. This will be the case provided 2^n is any large multiple of N — i.e. provided the input register is large enough to contain many periods of b^x (mod N). The stronger requirement that 2^n should be as large as N^2 — that the input register should actually be able to accomodate at least N full periods — emerges when we examine whether it is possible to learn r itself, given an integral multiple of $2^n/r$.

Suppose, then, that we have found a y that is within 1/2 of $j2^n/r$ for some integer j. It follows that

$$\left|\frac{y}{2^n} - \frac{j}{r}\right| \le \frac{1}{2^{n+1}}. \tag{3.56}$$

Since y is the result of our measurement and we know n, the number of input-register Qbits, we have an estimate for the fraction j/r. It is here that our use of an n-Qbit input register with $2^n > N^2$ is crucial. By using twice as many Qbits as needed to represent all the integers up to N, we have ensured that our estimate (3.56) of j/r is off by no more than $1/2N^2$. But since $r < N$, and since any two distinct fractions with denominators less than N must differ[11] by at least $1/N^2$, the measured value of y and the fact that r is less than N is enough to determine a unique value of the rational number j/r.

That value of j/r can be efficiently extracted from the known value of $y/2^n$ by a simple application of the theory of continued fractions.[12] This gives us the fraction j/r reduced to lowest terms — i.e. it gives us integers j_0 and r_0 which have no common factors satisfying $j_0/r_0 = j/r$. Since r_0 is r divided by the factors it has in common with j, if we were lucky enough to get a value of j that is coprime to r, then we have r. Since, as noted above, two random numbers j and r have a better than even chance of having no common factors, we do not have to be terribly lucky. We can easily check to see if r_0 is

[11] For $|\frac{a}{b} - \frac{c}{d}| \ge \frac{1}{bd}$ unless the two fractions are identical.

[12] This is described in the number-theoretic Appendix of Nielsen and Chuang. It is based on the theorem that if x is an estimate for j/r that differs from it by less than $1/2r^2$, then j/r will appear as one of the partial sums in the continued fraction expansion of x. This will be illustrated in Assignment #5.

the sought for period by computing (with a classical computer) b^{r_0} modulo (N) and seeing whether or not it is b. If it is not, we can try several low multiples $2r_0, 3r_0, 4r_0, \ldots$ since the probability of j sharing a large common factor with r is not large.

If this fails, we can repeat the whole procedure. We now get j'/r where j' is another (random) integer, yielding an r_0' which is r divided by the factors it has in common with j'. So r is the product of r_0 with the factors r has in common with j, and r is also the product of r_0' with the factors r has in common with j'. If j and j' have no factors in common — which again has a better than even chance of happening — then r will be the least common multiple[13] of r_0 and r_0'. We can again test to see if we have the right r by evaluating b^r (mod N) to see if it is indeed equal to b. If it is not, we can again try some of the lower multiples of our candidate for r and, if necessary, go through the whole business one more time to get yet another random multiple of $1/r$.

Because we are only 40% sure that our measurement gives us one of the y_j, we may have to repeat the whole procedure several (but not a great many) times before succeeding. Some not terribly taxing mathematical detective work, carried out with the aid of a classical computer, will quickly give us the period r.

F. Calculating the periodic function

We have assumed the existence of an efficient subroutine that calculates b^x (mod N). You might think that calculating $f(x) = b^x$ (mod N) for arbitrary values of x less than, say, $2^n = 10^{800}$ would require astronomical numbers of multiplications, but it does not. We simply square b (mod N), square the result (mod N), square that, etc., calculating the comparatively small number of powers b^{2^j} (mod N) with $j < n$. The binary expansion of $x = x_{n-1}x_{n-2}\ldots x_1x_0$ tells us which of these must be multiplied together to get $b^x = \prod_j (b^{2^j})^{x_j}$.

So if we start with x in the input register, 1 (i.e. $000\ldots001$) in the output register, and b in an additional work register, then we can proceed as follows:

> (a) multiply the ouput register by the work register if and only if $x_0 = 1$; (b) replace the contents of the work register by its modulo-N square; (a$'$) repeat (a) with the multiplication now conditional on $x_1 = 1$; (b$'$) repeat (b); (a$''$) repeat (a) with the multiplication now conditional on $x_2 = 1$; etc.

At the end of this process we will still have x in the input register (which serves only as a set of control bits for the n controlled multiplications), and we will have b^x mod (N) in the output register. The work register will contain b^{2^n} independent of the value of x in the input register, and it will therefore be unentangled with the input and output registers and can be ignored when we take our starting point to be a superposition of classical inputs.[14]

[13] The least common multiple of two numbers is their product divided by their greatest common divisor; the greatest common divisor can be found with the Euclidean algorithm, as noted above.

[14] As noted in Chapter 2, any additional registers used in the squaring and multiplica-

Note the striking difference between classical and quantum programming styles. One's classical computational instincts would direct one to make a look-up table of all n modulo-N multiple squares of b, since (a) Cbits are cheap and stable and (b) otherwise to get b^x modulo (N) for all the needed values of x one would have to recalculate the successive squares so many times that this would become ridiculously inefficient. But the situation is quite the opposite with a quantum computer, since (a) Qbits are expensive and fragile and (b) "quantum parallelism" makes it possible to produce the state (3.18) with only a single execution of the procedure that does the successive squarings, thereby relieving us of any need to store all the modulo-N squares, at a substantial saving in Qbits.

As usual with quantum parallelism, there is the major catch that a measurement can reveal the value of only a single one of the modulo-N powers of b. But, as we have now seen, by applying $\mathbf{U}_{FT}$ to the input register of the state (3.18) and only then making a measurement, one can get important collective information about the modulo-N values of b^x — in this case clues about the crucial period r — at the price of loosing all information about the individual values.

G. The unimportance of unavoidable small phase errors

To execute the quantum Fourier transform one needs 2-Qbit gates $\mathbf{V}_{ij} = e^{i\pi \mathbf{n}_i \mathbf{n}_j / 2^{|i-j|}}$ or, if one exploits the Griffiths-Niu trick, 1-Qbit gates $\mathbf{V}_j = e^{i\pi \mathbf{n}_j / 2^j}$. Since we need to deal with numbers of many hundreds of digits, the 2^j appearing in these phase gates can be larger than 10^{100}. Producing such tiny phase shifts requires a degree of control over the gates that is impossible to achieve. Typically such phase-shift gates would allow two Qbits to interact in a carefully controlled way for an interval of time that was very precisely specified, but obviously not to hundreds of significant figures. It is therefore crucial that the effectiveness of the period finding algorithm not be greatly affected by small (but not absurdly small) errors in the phase shifts.

On the face of it this seems worrisome. Since we need to know the period r to a hundred or more digits of precision, don't we have to get the phase shifts right to a comparable accuracy? Here the fundamentally digital character of the actual output of a quantum computation saves the day. To learn r we require the outcomes of several hundreds of 1-Qbit measurements, each of which has just two outcomes (0 or 1). While the action of the unitary gates that precede the measurements is like that of an analog computer, involving continuously variable phase shifts that cannot be controlled with perfect precision, this analog evolution only affects the *probabilities* of the sharply defined digital outputs. Small alterations in the phases will produce small alterations in the probabilities of getting that extremely precise digital information, but not the precision of the information itself, once it is acquired.[15]

---

tion subroutines must also be restored to their initial states to insure that they are also disentangled from the input and output registers.

[15] For a long time this quite crucial point seems to have been discussed only in an

Suppose that the phase of each term in the quantum Fourier transform (3.21) is incorrect by an amount $\varphi(x,y)$, and that each of these altered phases is bounded in magnitude by $\varphi \ll 1$. The probability $p(y)$ in (3.50) will be changed to

$$p_\varphi(y) = \frac{1}{2^n m}\Big|\sum_{k=0}^{m-1} e^{2\pi ikry/2^n} e^{i\varphi_k(y)}\Big|^2, \tag{3.57}$$

where $\varphi_k(y) = \varphi(x_0 + kr, y)$. Since all the phases $\varphi_k(y)$ are small compared with unity,

$$e^{i\varphi_k(y)} \approx 1 + i\varphi_k(y), \tag{3.58}$$

and therefore

$$p_\varphi(y) \approx \frac{1}{2^n m}\Big|\sum_{k=0}^{m-1} e^{2\pi ikry/2^n}\big(1 + i\varphi_k(y)\big)\Big|^2. \tag{3.59}$$

What affect does this have on the probability of learning from the measurement one of the special values y^j give in (3.51)?

We have

$$p_\varphi(y_j) \approx \frac{1}{2^n m}\Big|\sum_{k=0}^{m-1} e^{2\pi ikr\delta_j/2^n}\big(1 + i\varphi_{jk}\big)\Big|^2, \tag{3.60}$$

where $\varphi_{jk} = \varphi_k(y_j)$. If we expand to linear order in the small quantities φ_{jk}, we get

$$p_\varphi(y_j) \approx p(y_j) + \frac{2}{2^n m}\mathrm{Im}\Big[\big(\sum_{k=0}^{m-1} e^{-2\pi ikr\delta_j/2^n}\varphi_{jk}\big)\big(\sum_{k'=0}^{m-1} e^{2\pi ik'r\delta_j/2^n}\big)\Big]. \tag{3.61}$$

We can get an upper bound on the magnitude of the difference between the exact and approximate probabilities by replacing the imaginary part of the product of the two sums by the product of the absolute values of the sums, and then replacing each term in each sum by its absolute value. Since the absolute value of each φ_{jk} is bounded by φ, we can conclude that

$$|p(y_j) - p_\varphi(y_j)| \le \frac{2m}{2^n}\varphi = \frac{2}{r}\varphi. \tag{3.62}$$

Since there are r different y_j, the probability of getting one of the special values y_j is altered by less than 2φ. So if one is willing to settle for a probability of getting a special value that is at worst 1% less than the ideal value of about 0.4, one can tolerate phase errors up to $\varphi = 0.4/200 = 1/500$. If one leaves out of the quantum Fourier transform circuit all controlled phase gates $e^{\pi i\mathbf{n}_i\mathbf{n}_j/2^{|i-j|}}$ with $|i-j| > \ell$, the maximum phase error φ this can produce in any term is $\varphi = n\pi/2^\ell$, and therefore the probability will be within 1% of its ideal value if $1/2^\ell < 1/500n\pi$.

unpublished internal IBM report by D. Coppersmith. In 2002 that 1994 report was finally posted on `www.arXiv.org` as quant-ph/0201067.

The number of n of Qbits in the input register might be as large as 3000 for problems of interest (factoring a 500 digit N). Consequently for all practical purposes one can omit from the quantum Fourier transform all controlled-phase gates connecting Qbits that are more than about $\ell = 22$ wires apart in the circuit diagram. This has two major advantages. Of crucial importance, quantum engineers will not have to produce impossibly precise phase changes. Furthermore the size of the circuit executing the quantum Fourier transform only has to grow linearly with large n rather than quadratically. Since n is likely to be of order 10^3 for practical code-breaking, this too is a significant improvement.

H. Period finding and factoring

Since Shor's period-finding quantum algorithm is always described as a factoring algorithm, I conclude by noting how period finding leads to factoring. I consider only the case relevant to RSA encryption, where one wants to factor the product of two large primes, $N = pq$, although the connection between period finding and factoring is more general.

If we have a way to determine periods (such as Shor's algorithm) and want to find the large prime factors of $N = pq$, we pick a random number a coprime to N. The odds that a random a happens to be a multiple of p or of q are minuscule when p and q are enormous, but if you are the worrying kind you can check that it isn't using the Euclidean algorithm. (In the overwhelmingly unlikely event that it is, then the Euclidean algorithm will give you p or q directly, and you will have factored N.) Using our period finding routine, we find the order of a in G_{pq}: the smallest r for which

$$a^r \equiv 1 \ (\text{mod}\ pq). \tag{3.63}$$

We can use this information to factor N provided our choice of a was lucky in two ways:

Suppose first that we are fortunate enough to get an r that is even. We can then calculate

$$x = a^{r/2} \ (\text{mod}\ pq) \tag{3.64}$$

and note that

$$0 \equiv x^2 - 1 \equiv (x-1)(x+1) \quad (\text{mod}\ pq). \tag{3.65}$$

Now $x - 1 = a^{r/2} - 1$ is not congruent to 0 mod pq, since r is the *smallest* power of a congruent to 1. Suppose in addition — our second piece of good fortune — that

$$x + 1 = a^{r/2} + 1 \not\equiv 0 \ (\text{mod}\ pq). \tag{3.66}$$

In that case neither $x - 1$ nor $x + 1$ is divisible by $N = pq$, but (3.65) tells us that their product is. Since p and q are prime this is only possible if one of them, say p, divides $x - 1$ and the other, q, divides $x + 1$. Because the only divisors of N are p and q, it follows that p is the greatest common divisor of N and $x - 1$, while q is the greatest common divisor of N and $x + 1$. We can therefore find p or q by a straightforward application of the Euclidean algorithm.

So it all comes down to the likelihood of our being lucky. We show in Appendices A3 and A4 that the probability is at least 50% that a random number a in G_{pq} has an order r that is even with $a^{r/2} \not\equiv -1 \pmod{pq}$ so we do not have to repeat the procedure an enormous number of times to achieve a very high probability of success. If you're willing to accept the fact that you don't have to try out very many random a's to succeed, then this elementary argument is all you need to know about why period-finding enables you to factor $N = pq$.

But if you're curious about why the probability of good fortune is so high, then you must contend with Appendices A3 and A4, where I have constructed an elementary but rather elaborate argument, by condensing a fairly large body of number-theoretic lore into the comparatively simple form it assumes when applied to the special case in which the number N is the product of two primes.

Appendix to Chapter 3

A1. Some elementary group theory.

A set of positive integers less than N constitutes a *group* under multiplication modulo N if the set (a) contains 1, (b) contains the modulo-N inverse of any of its members, and (c) contains the the modulo-N products of all pairs of its members. A subset of a group meeting conditions (a)-(c) is called a *subgroup.* The number of members of a group is called the *order* of the group. An important result of the elementary theory of finite groups (Lagrange's theorem) is that the order of any of its subgroups is a divisor of the order of the group itself. The point of the indented paragraphs that follow is only to establish this.

> If S is any subset of a group G (not necessarily a subgroup) and a is any member of G (which might or might not be in S), define aS (called a *coset* of S) to be the set of all members of G of the form $g = as$ where s is any member of S. (Throughout this appendix equality will be taken to mean equality modulo N.) Distinct members of S give rise to distinct members of aS, for if s and s' are in S and $as = as'$, then multiplying both sides by the inverse of a gives $s = s'$. So any coset aS has the same number of members as S itself.
>
> If the subset S is a *subgroup* of G and a is a member s of S, then every member of the coset sS must be in S. Since sS has as many distinct members as S has, $sS = S$. If two cosets aS and bS of a subgroup S have a common member then there are members s and s' of S that satisfy $as = bs'$, so $(as)S = (bs')S$. But $(as)S = a(sS) = aS$, and similarly $(bs')S = bS$. Therefore $aS = bS$: two cosets of a *subgroup* are either identical or have no members in common.
>
> If S is a subgroup then since 1 is in S, any member a of G is in the coset aS. Since every member of G is thus in some coset, and since the cosets of a subgroup are either identical or disjoint, it follows that the distinct cosets of a subgroup S partition the whole group G into disjoint subsets, each of which has the same

number of members as S. Consequently the total number of members of G must be an integral multiple of the number of members of any of its subgroups S: *The order of any subgroup S is a divisor of the order of the whole group G.*

Of particular interest is the subgroup given by all the distinct powers of any particular member a of G. Since G is a finite set, the set of distinct powers of a is also finite, and therefore for some n and m with $n > m$ we must have $a^n = a^m$, or $a^{(n-m)} = 1$. The *order* of a is defined to be the smallest non-zero k with $a^k = 1$. The subset $a, a^2, \ldots, a^k$ of G is a subgroup of G, since it contains $1 = a^k$, and the inverses and products of all its members. It is called the subgroup *generated* by a, and its order is the order k of a. Since the order of any subgroup of G divides the order of G, we conclude that the order of any member of G divides the order of G.

This is all the group theory one needs to know to understand RSA encryption.

A2: The Euclidean algorithm.

We wish to find the greatest common divisor of two numbers f and c, with $f > c$. The Euclidean algorithm is the iterative procedure that replaces f and c by $f' = c$ and $c' = f - \mathrm{Int}[f/c]c$, where $\mathrm{Int}[x]$ is the largest integer less than or equal to x. Evidently any factors common to f and c are also common to f' and c'. Furthermore f' and c' decrease with each iteration and each iteration keeps $f' > c'$, until the procedure reaches $c' = 0$. Let f_0 and c_0 be the values of f and c at the last stage before $c = 0$. They are still divisible by all common factors of the original f and c, but in addition f_0 is divisible by c_0 (or the next stage would not be $c_0' = 0$.) Therefore c_0 is the greatest common divisor of the original f and c.

If f and c (less than f) have no common factors, then iterating the Euclidean algorithm eventually leads to $c_0 = 1$. This stage must have been immediately preceded by a pair f_1 and c_1 satisfying $f_1 - mc_1 = 1$ for some integer m. But f_1 and c_1 are given by explicit integral linear combinations of the pair at the preceding stage, f_2 and c_2. That pair in turn are explicit integral linear combinations of f_3 and c_3, etc. So one can work backwards through the iterations to construct integers j and k with $1 = jf + kc$. We can express k as $lf + d$ with $1 < d < f$ and with l a (possibly negative) integer; d is then the inverse of c modulo f.

A3. More on Factoring

We establish here that the probability is at least $\frac{1}{2}$ that if a is a random member of G_{pq}, then the order r of a satisfies both

$$r \text{ even} \tag{3.67}$$

and

$$a^{r/2} \not\equiv -1 \pmod{pq}. \tag{3.68}$$

In Section H it is shown that given such an a and its order r, the problem of factoring $N = pq$ is trivially solved.

Note first that the order r of a in G_{pq} is the least common multiple of the orders r_p and r_q of a in G_p and in G_q. That r must be *some* multiple of both r_p and r_q is immediate, since $a^r \equiv 1 \pmod{pq}$ implies that $a^r \equiv 1 \pmod{p}$ and $a^r \equiv 1 \pmod{q}$. Furthermore *any* common multiple r' of r_p and r_q satisfies $a^{r'} \equiv 1 \pmod{pq}$, because if $a^{r'} = 1 + mp$ and $a^{r'} = 1 + nq$, then $mp = nq$. But since the primes p and q have no common factors this requires $m = kq$ and $n = kp$, and hence $a^{r'} = 1 + kpq \equiv 1 \pmod{pq}$. Since r is the least integer with $a^r \equiv 1 \pmod{pq}$, since r must be a common multiple of r_p and r_q, and since any such multiple works, r must be the least common multiple of r_p and r_q.

Consequently condition (3.67) can fail only if r_p and r_q are both odd. Condition (3.68) can fail only if r_p and r_q are both odd multiples of the *same* power of 2; for if r_p contains a higher power of 2 than r_q, then since r is a common multiple of r_p and r_q, it will remain a multiple of r_q if a single factor of 2 is removed from it, and therefore that $a^{r/2} \equiv 1 \pmod{q}$. But this is inconsistent with a failure of condition (3.68), which would imply that $a^{r/2} \equiv -1 \pmod{q}$.

So a necessary condition for failure to factor $N = pq$ is that either r_p and r_q are both odd, or they are both odd multiples of the same power of 2. These neatly combine into a single necessary condition: Our effort to factor N can fail only if we have picked a random a for which r_p and r_q are both odd multiples of 2^j for any $j \geq 0$. We next show that the probability p_f of this happening is less than $\frac{1}{2}$.

To calculate a upper bound for the probability of failure p_f, note first that the mod-p and mod-q orders, r_p and r_q, of a are the same as the mod-p and mod-q orders of the numbers a_p and a_q in G_p and G_q, where

$$a \equiv a_p \pmod{p}, \qquad a \equiv a_q \pmod{q}. \tag{3.69}$$

Furthermore, every number a in G_{pq} is associated through (3.69) with a *distinct* pair from G_p and G_q. For if $a_p = b_p$ and $a_q = b_q$ then $a - b$ is a multiple of both p and q, and therefore, since p and q are distinct primes, $a - b$ is a multiple of pq itself, so $a \equiv b \pmod{pq}$.

Since the $(p-1)(q-1)$ different members of G_{pq} are thus in one-to-one correspondence with the number of distinct pairs, one from the $p-1$ members of G_p and one from the $q-1$ members of G_q, the modulo-p and modulo-q orders r_p and r_q of a random integer a in G_{pq} will have exactly the same statistical distribution as the orders r_p and r_q of randomly and independently selected integers in G_p and G_q. So to show that the probability of failure is at most $\frac{1}{2}$, we must show that the probability is at most $\frac{1}{2}$ that the orders r_p and r_q of such a randomly and independently selected pair are both odd multiples of 2^j for some $j \geq 0$..

We do this by showing that for any prime p, no more than half of the numbers in G_p can have orders r_p that are odd multiples of *any* given power of 2. [Given this, if $P_p(j)$ and $P_q(j)$ are the probabilities that random elements of G_p and G_q have orders that are odd multiples of 2^j, then the probability of failre p_f is less than $\sum_{j\geq 0} P_p(j)P_q(j) \leq \frac{1}{2}\sum_{j\geq 0} P_q(j) \leq \frac{1}{2}$.] This in turn follows from the following fact: Let the order $p-1$ of G_p be an odd multiple of 2^k for some $k \geq 0$; then exactly half the elements of G_p have orders that are odd multiples of 2^k.

This last fact follows from use the important theorem (proved in Section A4 below) that if p is a prime, then G_p has at least one *primitive* element b of order $p-1$, whose successive powers therefore generate the entire group. Given this theorem, we complete the argument by showing that the the orders of the odd powers of any such primitive b are odd multiples 2^k, but the orders of the even powers are not.

If r_0 is the order of b^j with j odd, then

$$1 \equiv (b^j)^{r_0} \equiv b^{jr_0} \pmod{p}, \tag{3.70}$$

so jr_0 must be a multiple of $p-1$, the order of b. Since j is odd r_0 must contain at least as many powers of 2 as does $p-1$. But since the order r_0 of any element must divide the order $p-1$ of the group, r_0 cannot contain more powers of 2 than $p-1$. So r_0 is an odd multiple of 2^k.

On the other hand if j is even, then b^j satisfies

$$(b^j)^{(p-1)/2} = \left(b^{p-1}\right)^{j/2} \equiv 1 \pmod{p}, \tag{3.71}$$

so the order r_0 of b^j divides $(p-1)/2$. Therefore r_0 must contain at least one less power of 2 than does $p-1$.

This completes the proof that the probability is at least 1/2 that a random choice of a in G_{pq} will satisfy both the conditions (3.67) and (3.68) that lead, with the aid of an efficient period-finding routine, to an easy factorization of $N = pq$.

A4. G_p has an element of order $p-1$ when p is prime.

It remains to prove the theorem that when p is prime, G_p contains at least one number of order $p-1$. The relevant property of the multiplicative group of integers $\{1, 2, 3, \ldots p-1\}$ modulo a *prime* is that together with 0 these integers also constitute a group under *addition* mBut sinceodulo p, which has all the structure necessary to ensure that a polynomial equation of degree d has at most d roots.[16] We can exploit this fact as follows:

[16] This is easily proved by induction on the degree of the equation, using the fact that every non-zero integer modulo p has a multiplicative inverse modulo p. It is obviously true for degree 1. Suppose it is true for degree $m-1$ and a polynomial $P(x)$ of degree m satisfies $P(a) = 0$. Then $P(x) = 0$ implies $P(x) - P(a) = 0$. Since $P(x) - P(a)$ has the form $\sum_j c_j(x^j - a^j)$, the factor $x - a$ can be extracted from each term leading to the form $(x-a)Q(x)$ where $Q(x)$ is a polynomial of degree $m-1$. So if $x \neq a$ then $Q(x) = 0$, and this has at most $m-1$ distinct solutions by the inductive assumption.

Write the order $s = p - 1$ of G_p in terms of its prime factors q_i:

$$s = p - 1 = q_1^{n_1} \cdots q_m^{n_m}. \tag{3.72}$$

For each q_i, the polynomial $x^{s/q_i} - 1$ has at most s/q_i roots, and since $s/q_i < s$, the number of elements in G_p, there must be elements a_i in G_p satisfying

$$a_i^{s/q_i} \not\equiv 1 \pmod{p}. \tag{3.73}$$

Given such an a_i, define

$$b_i = a_i^{s/(q_i^{n_i})}. \tag{3.74}$$

We next show that the order of b_i is $q_i^{n_i}$. This is because

$$b_i^{(q_i^{n_i})} \equiv a_i^s \equiv 1 \pmod{p}, \tag{3.75}$$

so the order of b_i must divide $q_i^{n_i}$ and therefore be a power of q_i, since q_i is prime. But if that order were any power of q_i less than n_i, then we would have $a_i^{s/q_i^k} \equiv 1 \pmod{p}$ with $k \geq 1$, which contradicts (3.73).

Because each b_i has order $q_i^{n_i}$, the product $b_1 b_2 \cdots b_m$ has order $q_1^{n_1} q_2^{n_2} \cdots q_m^{n_m} = p - 1$. This follows from the fact that if two numbers in G_p have orders that are coprime, then the order of their product is the product of their orders.[17] Therefore since $q_1^{n_1}$ and $q_2^{n_2}$ are coprime, $b_1 b_2$ has order $q_1^{n_1} q_2^{n_2}$. But since $q_1^{n_1} q_2^{n_2}$ and $q_3^{n_3}$ are coprime, it follows that $b_1 b_2 b_3$ has order $q_1^{n_1} q_2^{n_2} q_3^{n_3}$. Continuing in this way, we conclude that $b_1 b_2 \cdots b_m$ has order $q_1^{n_1} q_2^{n_2} \cdots q_m^{n_m} = s = p - 1$.

[17] For let u, v, and w be the orders of c, d, and cd. Since $c^u \equiv 1 \pmod{p}$ and $(cd)^w \equiv 1 \pmod{p}$, it follows that $d^{wu} \equiv 1 \pmod{p}$. So the order v of d divides wu, and since v and u have no common factors, v divides w. In the same way one concludes that u divides w. Therefore, since v and u are coprime, w must be a multiple of uv. Furthermore, $(cd)^{uv} \equiv c^{uv} d^{vu} \equiv 1 \pmod{p}$, uv must be a multiple of w. Therefore $w = uv$.

Figure 3.1

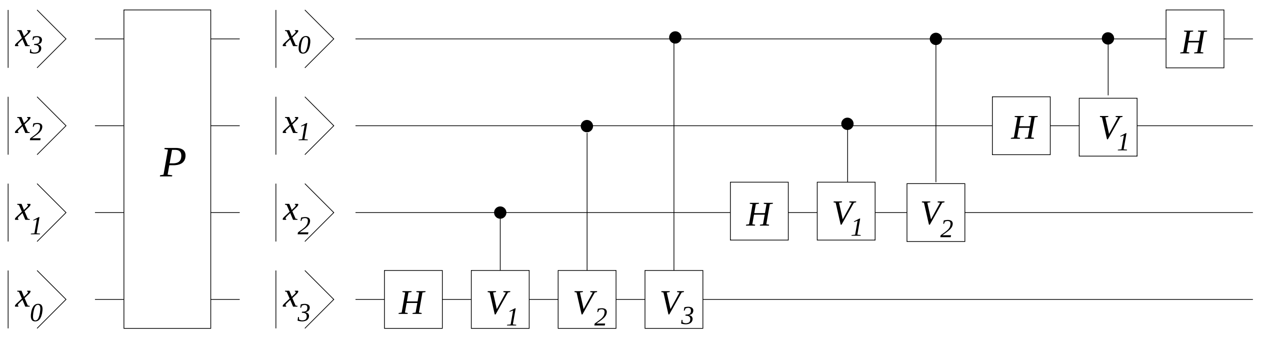

Figure 3.1. Diagram of a circuit that illustrates, for four Qbits, the construction of the quantum Fourier transform $\mathbf{U}_{FT}$ defined in (3.21), as the product of one- and two-Qbit gates given in (3.43). (Note that in the figure the order in which the gates act is from left to right, while in (3.43) the order in which they act is from right to left.)

The convention, as usual, is that the Qbits representing more significant bits are represented by higher lines in the figure. Note, however, that acting on the computational basis, the first gate $\mathbf{P}$ permutes the states of the Qbits, exchanging the states of the most and least significant Qbits, and the states of the next most significant and next least significant Qbits. Rather than going to the trouble of constructing such a gate, it obviously makes more sense simply to reverse the convention for the input state, representing Qbits that represent more significant bits by lower lines in the figure. This is what one usually encounters in diagrams of the quantum Fourier transform: the gate $\mathbf{P}$ is omitted and the conventional ordering of significant bits is reversed for the input. The complete figure then reduces to the portion to the right of the permutation gate $\mathbf{P}$. For the output, of course, the conventional ordering remains in effect.

If the input on the left is the computational basis state $|x\rangle_4 = |x_3\rangle|x_2\rangle|x_1\rangle|x_0\rangle$ the output will be $\mathbf{U}_{FT}|x\rangle_4$, the superposition (3.21) of computational basis states $|y\rangle_4 = |y_3\rangle|y_2\rangle|y_1\rangle|y_0\rangle$, defined in (3.21).

There is no need for the figure to have subscripts on the Hadamard gates appearing in (3.44), since each is explicitly attached to the line associated with the Qbit on which it acts. For the same reason each two-Qbit controlled-V gate requires only a single subscript, which specifies the unitary operator $\mathbf{V}_k$ that acts on the target Qbit; the subscript is the number of "wires" the target Qbit is away from the control Qbit. The explicit form of $\mathbf{V}_k$ is $e^{i\pi\mathbf{n}/2^k}$ where $\mathbf{n}$ is the number operator for the target Qbit. In the computational basis the controlled-V gate acts as the identity unless control and target Qbits are both in the state $|1\rangle$, in which case it multiplies the state of the target Qbit by $e^{i\pi/2^k}$.

Figure 3.2

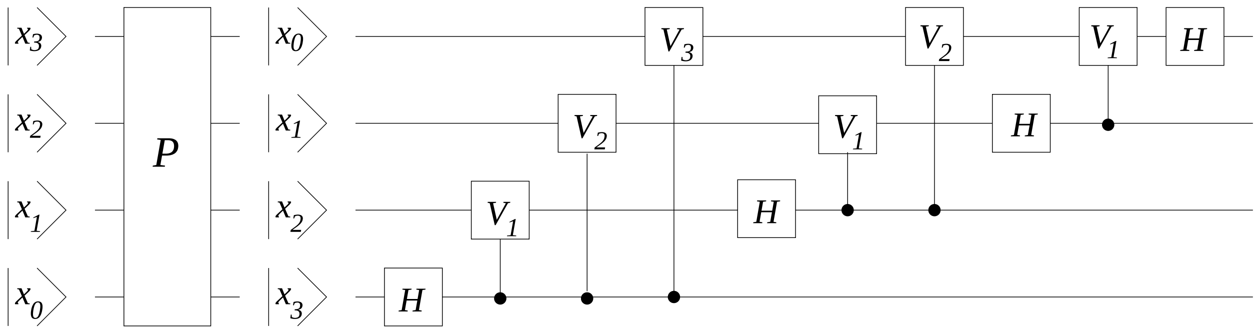

Figure 3.2. Since the action (3.40) of the controlled-V gates is symmetric in i and j, Figure 3.1 can be redrawn with control and target Qbits interchanged. If the quantum Fourier transformation is immediately followed by a measurement of all the Qbits, as it is when applied to period finding, then this equivalent representation of the quantum Fourier transformation reveals that the two-Qbit controlled-V gates can be replaced by one-Qbit unitary gates, which act or not depending on the outcome of earlier measurements.

To how this comes about consider first the bottom wire. Once $\mathbf{H}$ and the three controlled-V gates have acted on it, nothing further happens to it until the final measurement of the least significant output Qbit. If the result of that measurement is 1, the state of the four Qbits reduces to that component of the full superposition in which $\mathbf{V}_1$, $\mathbf{V}_2$, and $\mathbf{V}_3$ have acted on the three wires above the bottom wire; if the result of the measurement is 0, the four-Qbit state reduces to the component in which they have not acted. We can produce exactly the same effect if we measure the least significant output Qbit immediately after $\mathbf{H}$ has acted on the bottom wire, before any of the other gates have acted, and then do or do not apply the three unitary transformations to the other three Qbits, depending on whether the outcome of the measurement is 1 or 0. Next, we apply the Hadamard transformation to the second wire from the bottom. We then immediately measure the next least significant output Qbit and, depending on the outcome, apply or not apply the appropriate one-Qbit unitary transformations to each of the remaining two Qbits. Continuing in this way, we end up producing exactly the same statistical distribution of measurement results as we would have produced had we used the two-Qbit controlled-V gates, measuring none of the Qbits until the full unitary transformation $\mathbf{U}_{FT}$ was produced.

This simplification, which eliminates the need for any 2-Qbit gates, was pointed out by Robert B. Griffiths and Chi-Sheng Niu, Phys. Rev. Lett. **76**, 3228-3231 (1996). The procedure that results is pictured in Fig. 3.3.

Figure 3.3

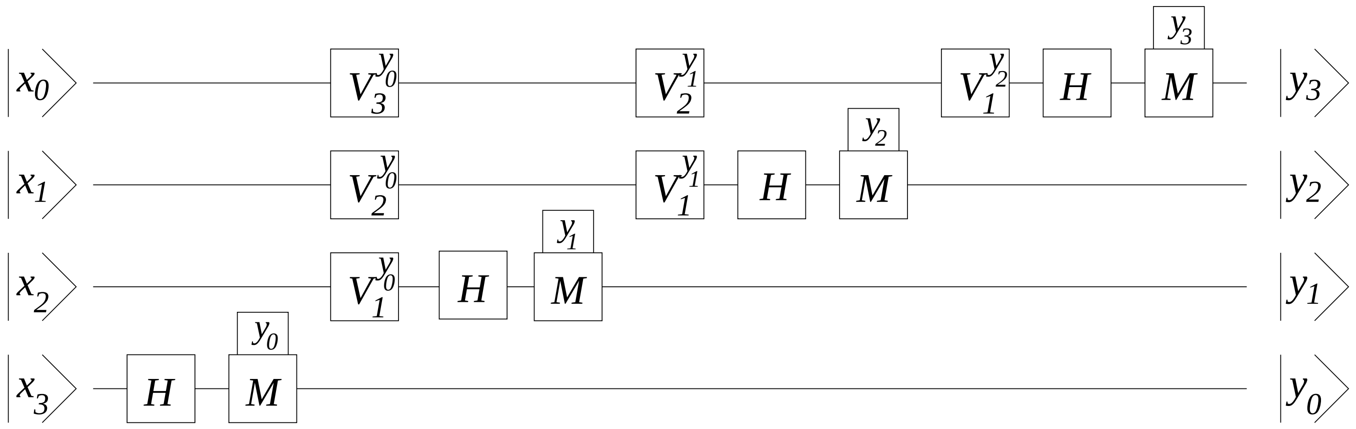

Figure 3.3. If the Qbits are all measured immediately after all the gates of the quantum Fourier transform have all acted, then the one-Qbit measurement gates can be applied to each Qbit immediately after the action of the Hadamard gate on that Qbit, and the controlled-V gates that follow the action of the Hadamard (see Figure 3.2) can be replaced by one-Qbit gates that act or not depending on whether the outcome y of the one-Qbit measurement is 1 or 0.

Note that the final state on the right can now be indicated, since it is just the computational basis state associated with the outcomes of the measurements of the individual Qbits. If the input state is a computational basis state, the output state is entirely random, all states $|y\rangle_4$ having equal probability. However the distribution of output states can have a highly informative structure when the input is an appropriate superposition of computational basis states.

Figure 3.4

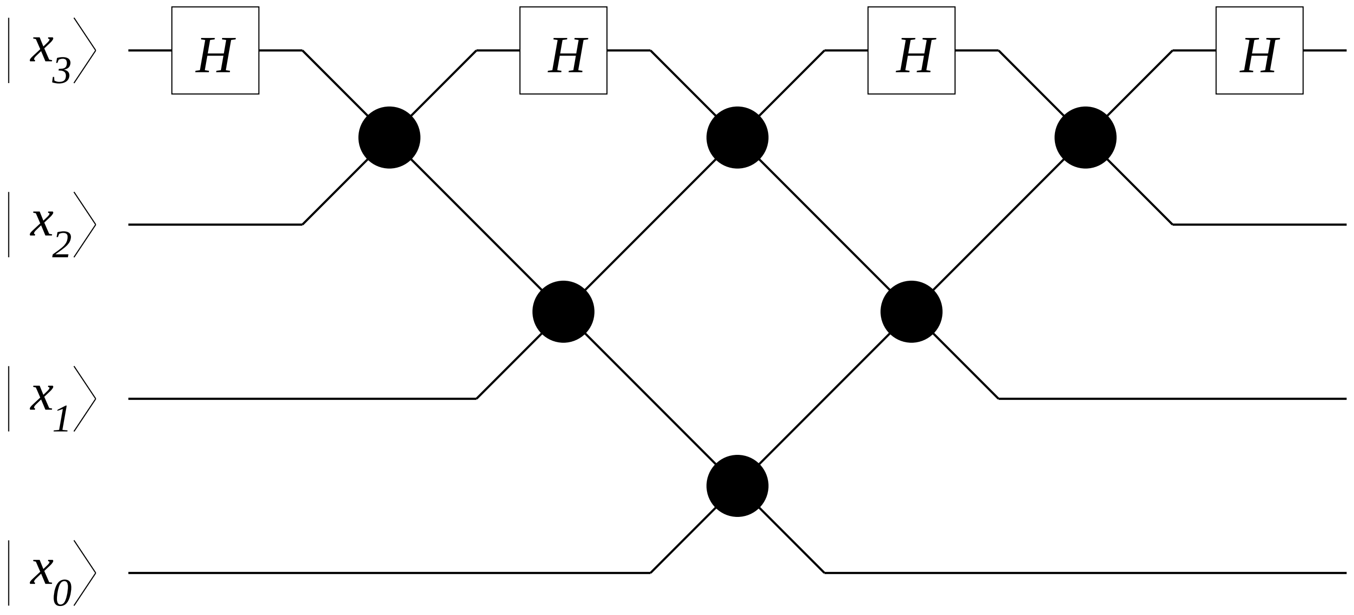

Figure 3.4. A much more symmetric way of drawing Figs. 1 or 2, due to Griffiths and Niu. (Although it is superior to the conventional diagram, it does not seem to have caught on.) The permutation $\mathbf{P}$ that in effect permutes the Qbits in the input register is now built into the diagram by using lines that no longer connect input register Qbits to output register Qbits at the same horizontal level. Because the lines now cross one another, the unitary operators $\mathbf{V}$ can be represented by the black circles at the intersection of the lines associated with the Qbits that they couple. This eliminates the artificial distinction between control and target Qbits used in Figs. 1 and 2. The form of each such operator is $\mathbf{V} = \exp\bigl(i\pi\mathbf{n}\mathbf{n}'/2^k\bigr)$, where $\mathbf{n}$ and $\mathbf{n}'$ are the Qbit number operators associated with the two lines that cross at the dot, and $k = 1, 2,$ or 3 depending on whether the dot lies in the first, second, or third horizontal row below the top row of Hadamard transformations.