Hence, we have

$$|\varphi_{3b}\rangle = \begin{array}{c} \begin{array}{cccccccc} \mathbf{000} & \mathbf{001} & \mathbf{010} & \mathbf{011} & \mathbf{100} & \mathbf{101} & \mathbf{110} & \mathbf{111} \end{array} \\ \left[\begin{array}{cccccccc} \frac{-1}{4\sqrt{8}}, & \frac{-1}{4\sqrt{8}}, & \frac{-1}{4\sqrt{8}}, & \frac{-1}{4\sqrt{8}}, & \frac{-1}{4\sqrt{8}}, & \frac{11}{4\sqrt{8}}, & \frac{-1}{4\sqrt{8}}, & \frac{-1}{4\sqrt{8}} \end{array} \right]^T \end{array}. \tag{6.146}$$

For the record, $\frac{11}{4\sqrt{8}} = 0.97227$ and $\frac{-1}{4\sqrt{8}} = -0.08839$. Squaring the numbers gives us the probability of measuring those numbers. When we measure the state in Step 4, we will most likely get the state

$$|\varphi_4\rangle = \begin{array}{c} \begin{array}{cccccccc} \mathbf{000} & \mathbf{001} & \mathbf{010} & \mathbf{011} & \mathbf{100} & \mathbf{101} & \mathbf{110} & \mathbf{111} \end{array} \\ \left[\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array} \right]^T \end{array}, \tag{6.147}$$

which is exactly what we wanted. □

Exercise 6.4.5 Do a similar analysis for the case where $n = 4$ and f chooses the "1101" string. ■

A classical algorithm will search an unordered array of size n in n steps. Grover's algorithm will take time $\sqrt{n}$. This is what is referred to as a quadratic speedup. Although this is very good, it is not the holy grail of computer science: an exponential speedup. In the next section we shall meet an algorithm that does have such a speedup.

What if we relax the requirements that there be only one needle in the haystack? Let us assume that there are t objects that we are looking for (with $t < \frac{2^n}{2}$). Grover's algorithm still works, but now one must go through the loop $\sqrt{\frac{2^n}{t}}$ times. There are many other types of generalizations and assorted changes that one can do with Grover's algorithm. Several references are given at the end of the chapter. We discuss some complexity issues with Grover's algorithm at the end of Section 8.3.

6.5 SHOR'S FACTORING ALGORITHM

The problem of factoring integers is very important. Much of the World Wide Web's security is based on the fact that it is "hard" to factor integers on classical computers. Peter Shor's amazing algorithm factors integers in polynomial time and really brought quantum computing into the limelight.

Shor's algorithm is based on the following fact: the factoring problem can be reduced to finding the period of a certain function. In Section 6.3 we learned how to find the period of a function. In this section, we employ some of those periodicity techniques to factor integers.

We shall call the number we wish to factor N. In practice, N will be a large number, perhaps hundreds of digits long. We shall work out all the calculations for the numbers 15 and 371. For exercises, we ask the reader to work with the number 247. We might as well give away the answer and tell you that the only nontrivial factors of 247 are 19 and 13.

We assume that the given N is not a prime number but is a composite number. There now exists a deterministic, polynomial algorithm that determines if N is prime

(Agrawal, Kayal, and Saxena, 2004). So we can easily check to see if N is prime before we try to factor it.

Reader Tip. There are several different parts of this algorithm and it might be too much to swallow in one bite. If you are stuck at a particular point, may we suggest skipping to the next part of the algorithm. At the end of this section, we summarize the algorithm. ♡

Modular Exponentiation. Before we go on to Shor's algorithm, we have to remind ourselves of some basic number theory. We begin by looking at some **modular arithmetic.** For a positive integer N and any integer a, we write a Mod N for the remainder (or residue) of the quotient a/N. (For C/C++ and Java programmers, Mod is recognizable as the % operation.)

Example 6.5.1 Some examples:

- 7 Mod 15 = 7 because 7/15 = 0 remainder 7.
- 99 Mod 15 = 9 because 99/15 = 6 remainder 9.
- 199 Mod 15 = 4 because 199/15 = 13 remainder 4.
- 5,317 Mod 371 = 123 because 5,317/371 = 14 remainder 123.
- 2,3374 Mod 371 = 1 because 2,3374/371 = 63 remainder 1.
- 1,446 Mod 371 = 333 because 1,446/371 = 3 remainder 333. □

Exercise 6.5.1 Calculate

(i) 244,443 Mod 247
(ii) 18,154 Mod 247
(iii) 226,006 Mod 247. ■

We write

$$a \equiv a' \text{ Mod } N \quad \text{if and only if } (a \text{ Mod } N) = (a' \text{ Mod } N), \tag{6.148}$$

or equivalently, if N is a divisor of $a - a'$, i.e., $N|(a - a')$.

Example 6.5.2 Some examples:

- $17 \equiv 2$ Mod 15
- $126 \equiv 1{,}479{,}816$ Mod 15
- $534 \equiv 1{,}479$ Mod 15
- $2{,}091 \equiv 236$ Mod 371
- $3{,}350 \equiv 2237$ Mod 371
- $3{,}325{,}575 \equiv 2{,}765{,}365$ Mod 371. □

Exercise 6.5.2 Show that

(i) $1{,}977 \equiv 1$ Mod 247
(ii) $16{,}183 \equiv 15{,}442$ Mod 247
(iii) $2{,}439{,}593 \equiv 238{,}082$ Mod 247.

■

With Mod understood we can start discussing the algorithm. Let us randomly choose an integer a that is less than N but does not have a nontrivial factor in common with N. One can test for such a factor by performing Euclid's algorithm to calculate GCD(a, N). If the GCD is not 1, then we have found a factor of N and we are done. If the GCD is 1, then a is called **co-prime** to N and we can use it. We shall need to find the powers of a modulo N, that is,

$$a^0 \text{ Mod } N, \quad a^1 \text{ Mod } N, \quad a^2 \text{ Mod } N, \quad a^3 \text{ Mod } N, \quad \ldots \tag{6.149}$$

In other words, we shall need to find the values of the function

$$f_{a,N}(x) = a^x \text{ Mod } N. \tag{6.150}$$

Some examples are in order.

Example 6.5.3 Let $N = 15$ and $a = 2$. A few simple calculations show that we get the following:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
$f_{2,15}(x)$	1	2	4	8	1	2	4	8	1	2	4	8	1	$\cdots$

. (6.151)

For $a = 4$, we have

x	0	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
$f_{4,15}(x)$	1	4	1	4	1	4	1	4	1	4	1	4	1	$\cdots$

. (6.152)

For $a = 13$, we have

x	0	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
$f_{13,15}(x)$	1	13	4	7	1	13	4	7	1	13	4	7	1	$\cdots$

. (6.153)

□

The first few outputs of $f_{13,15}$ function can be viewed as the bar graph in Figure 6.3.

Example 6.5.4 Let us work out some examples with $N = 371$. This is a little harder and probably cannot be done with a handheld calculator. The numbers simply get too large. However, it is not difficult to write a small program, use MATLAB or Microsoft Excel. Trying to calculate a^x Mod N by first calculating a^x will not go very far, because the numbers will usually be beyond range. Rather, the trick is to calculate a^x Mod N from a^{x-1} Mod N by using the standard number theoretic fact

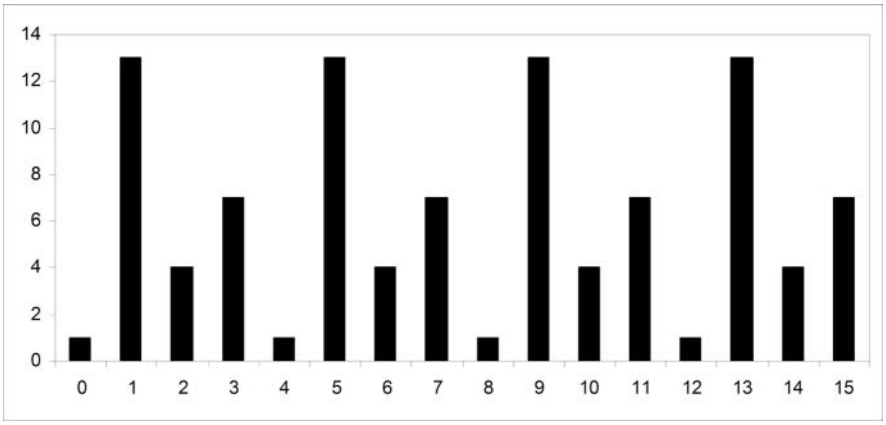


Figure 6.3. The first few outputs of $f_{13,15}$.

that

$$\text{if } a \equiv a' \text{ Mod } N \text{ and } b \equiv b' \text{ Mod } N, \text{ then } a \times b \equiv a' \times b' \text{ Mod } N. \tag{6.154}$$

Or, equivalently

$$a \times b \text{ Mod } N = (a \text{ Mod } N) \times (b \text{ Mod } N) \text{ Mod } N. \tag{6.155}$$

From this fact we get the formula

$$a^x \text{ Mod } N = a^{x-1} \times a \text{ Mod } N = ((a^{x-1} \text{ Mod } N) \times (a \text{ Mod } N)) \text{ Mod } N. \tag{6.156}$$

Because $a < N$ and a Mod $N = a$, this reduces to

$$a^x \text{ Mod } N = ((a^{x-1} \text{ Mod } N) \times a) \text{ Mod } N. \tag{6.157}$$

Using this, it is easy to iterate to get the desired results. For $N = 371$ and $a = 2$, we have

x	0	1	2	3	4	5	6	7	$\cdots$	78	$\cdots$	155	156	157	158	$\cdots$
$f_{2,371}(x)$	1	2	4	8	16	32	64	128	$\cdots$	211	$\cdots$	186	1	2	4	$\cdots$

.

(6.158)

For $N = 371$ and $a = 6$, we have

x	0	1	2	3	4	5	6	7	$\cdots$	13	$\cdots$	25	26	27	28	$\cdots$
$f_{6,371}(x)$	1	6	36	216	183	356	281	202	$\cdots$	370	$\cdots$	62	1	6	36	$\cdots$

.

(6.159)

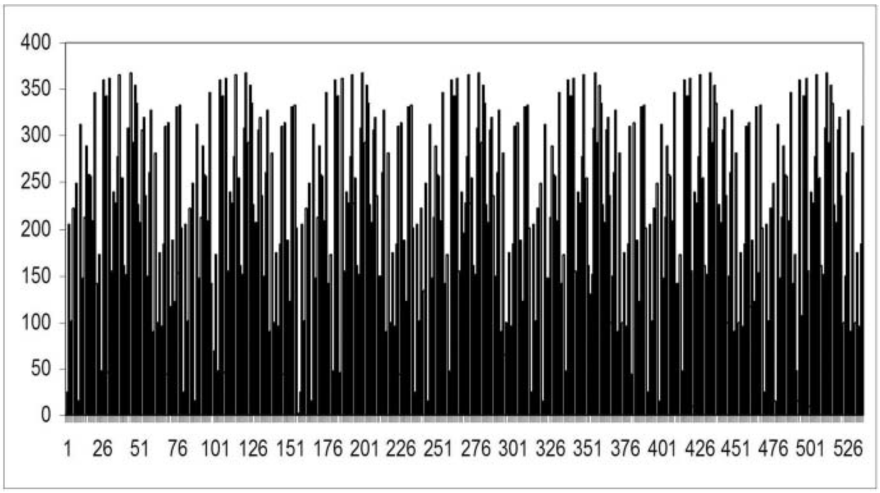


Figure 6.4. The output of $f_{24,371}$.

For $N = 371$ and $a = 24$, we have

x	0	1	2	3	4	5	6	7	$\cdots$	39	$\cdots$	77	78	79	80	$\cdots$
$f_{24,371}(x)$	1	24	205	97	102	222	134	248	$\cdots$	160	$\cdots$	201	1	24	205	$\cdots$

. (6.160)

□

We can see the results of $f_{24,371}$ as a bargraph in Figure 6.4.

Exercise 6.5.3 Calculate the first few values of $f_{a,N}$ for $N = 247$ and

(i) $a = 2$
(ii) $a = 17$
(iii) $a = 23$.

■

In truth, we do not really need the values of this function, but rather we need to find the **period** of this function, i.e., we need to find the smallest r such that

$$f_{a,N}(r) = a^r \text{ Mod } N = 1. \tag{6.161}$$

It is a theorem of number theory that for any co-prime $a \leq N$, the function $f_{a,N}$ will output a 1 for some $r < N$. After it hits 1, the sequence of numbers will simply repeat. If $f_{a,N}(r) = 1$, then

$$f_{a,N}(r + 1) = f_{a,N}(1) \tag{6.162}$$

and in general

$$f_{a,N}(r + s) = f_{a,N}(s). \tag{6.163}$$

Example 6.5.5 Charts (6.151), (6.152), and (6.153) show us that the periods for $f_{2,15}$, $f_{4,15}$, and $f_{13,15}$ are 4, 2, and 4, respectively. Charts (6.158), (6.159), and (6.160) show us that the periods for $f_{2,371}$, $f_{6,371}$, and $f_{24,371}$ are 156, 26, and 78, respectively. In fact, it is easy to see the periodicity of $f_{24,371}$ in Figure 6.4. □

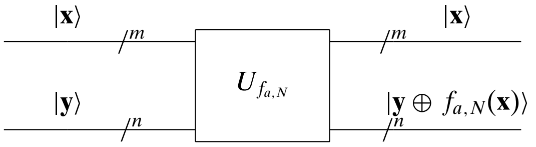
Exercise 6.5.4 Find the period of the functions $f_{2,247}$, $f_{17,247}$, and $f_{23,247}$. ■

The Quantum Part of the Algorithm. For small numbers like 15, 371, and 247, it is fairly easy to calculate the periods of these functions. But what about a large N that is perhaps hundreds of digits long? This will be beyond the ability of any conventional computers. We will need a quantum computer with its ability to be in a superposition to calculate $f_{a,N}(x)$ for *all* needed x.

How do we get a quantum circuit to find the period? First we have to show that there is a quantum circuit that can implement the function $f_{a,N}$. The output of this function will always be less than N, and so we will need $n = \log_2 N$ output bits. We will need to evaluate $f_{a,N}$ for at least the first N^2 values of x and so will need at least

$$m = \log N^2 = 2 \log N = 2n \tag{6.164}$$

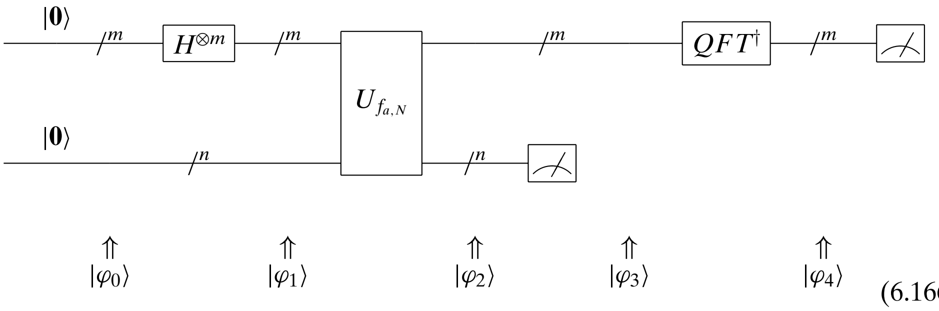
input qubits. The quantum circuit that we would get will be the operator $U_{f_{a,N}}$, which we may visualize as



(6.165)

where $|\mathbf{x}, \mathbf{y}\rangle$ goes to $|\mathbf{x}, \mathbf{y} \ominus f_{a,N}(\mathbf{x})\rangle = |\mathbf{x}, \mathbf{y} \oplus a^{\mathbf{x}} \text{ Mod } N\rangle$.[2] How is this circuit formed? Rather than destroying the flow of the discussion, we leave that technical discussion for a mini appendix at the end of this section.

With $U_{f_{a,N}}$, we can go on to use it in the following quantum algorithm. The first thing is to evaluate *all* the input at one time. From earlier sections, we know how to put $\mathbf{x}$ into an equally weighted superposition. (In fact, the beginning of this algorithm is very similar to Simon's algorithm.) We shall explain all the various parts of this quantum circuit:



$|\mathbf{0}\rangle$
$/^m$
$H^{\otimes m}$
$|\mathbf{0}\rangle$
$/^n$
$U_{f_{a,N}}$
$QFT^\dagger$
$\Uparrow$ $|\varphi_0\rangle$
$\Uparrow$ $|\varphi_1\rangle$
$\Uparrow$ $|\varphi_2\rangle$
$\Uparrow$ $|\varphi_3\rangle$
$\Uparrow$ $|\varphi_4\rangle$

(6.166)

[2] Until now, we have thought of x as any number and now we are dealing with x as its binary expansion $\mathbf{x}$. This is because we are thinking of x as described in a (quantum) computer. We shall use both notations interchangeably.

In terms of matrices this is

$$(Measure \otimes I)(QFT^{\dagger} \otimes I)(I \otimes Measure)U_{f_{a,N}}(H^{\otimes m} \otimes I)|\mathbf{0}_m, \mathbf{0}_n\rangle, \qquad (6.167)$$

where $\mathbf{0}_m$ and $\mathbf{0}_n$ are qubit strings of length m and n, respectively.

Let us look at the states of the system. We start at

$$|\varphi_0\rangle = |\mathbf{0}_m, \mathbf{0}_n\rangle. \qquad (6.168)$$

We then place the input in an equally weighted superposition of all possible inputs:

$$|\varphi_1\rangle = \frac{\sum_{\mathbf{x}\in\{0,1\}^m} |\mathbf{x}, \mathbf{0}_n\rangle}{\sqrt{2^m}}. \qquad (6.169)$$

Evaluation of f on all these possibilities gives us

$$|\varphi_2\rangle = \frac{\sum_{\mathbf{x}\in\{0,1\}^m} |\mathbf{x}, f_{a,N}(\mathbf{x})\rangle}{\sqrt{2^m}} = \frac{\sum_{\mathbf{x}\in\{0,1\}^m} |\mathbf{x}, a^{\mathbf{x}} \text{ Mod } N\rangle}{\sqrt{2^m}}. \qquad (6.170)$$

As the examples showed, these outputs repeat and repeat. They are periodic. We have to figure out what is the period. Let us meditate on what was just done. It is right here where the fantastic power of quantum computing is used. We have evaluated *all* the needed values at one time! Only quantum parallelism can perform such a task.

Let us pause and look at some examples.

Example 6.5.6 For $N = 15$, we will have $n = 4$ and $m = 8$. For $a = 13$, the state $|\varphi_2\rangle$ will be

$$\frac{|0, 1\rangle + |1, 13\rangle + |2, 4\rangle + |3, 7\rangle + |4, 1\rangle + \cdots + |254, 4\rangle + |255, 7\rangle}{\sqrt{256}}. \qquad (6.171)$$

□

Example 6.5.7 For $N = 371$, we will have $n = 9$ and $m = 18$. For $a = 24$, the state $|\varphi_2\rangle$ will be

$$\frac{|0, 1\rangle + |1, 24\rangle + |2, 205\rangle + |3, 97\rangle + |4, 102\rangle + \cdots + |2^{18} - 1, 24^{2^{18}-1} \text{ Mod } 371\rangle}{\sqrt{2^{18}}}. \qquad (6.172)$$

□

Exercise 6.5.5 Write the state $|\varphi_2\rangle$ for $N = 247$ and $a = 9$. ■

Going on with the algorithm, we measure the bottom qubits of $|\varphi_2\rangle$, which is in a superposition of many states. Let us say that after measuring the bottom qubits we find

$$a^{\overline{\mathbf{x}}} \text{ Mod } N \qquad (6.173)$$

for some $\overline{\mathbf{x}}$. However, by the periodicity of $f_{a,N}$ we also have that

$$a^{\overline{\mathbf{x}}} \equiv a^{\overline{\mathbf{x}}+r} \text{ Mod } N \tag{6.174}$$

and

$$a^{\overline{\mathbf{x}}} \equiv a^{\overline{\mathbf{x}}+2r} \text{ Mod } N. \tag{6.175}$$

In fact, for any $s \in \mathbb{Z}$ we have

$$a^{\overline{\mathbf{x}}} \equiv a^{\overline{\mathbf{x}}+sr} \text{ Mod } N. \tag{6.176}$$

How many of the 2^m superpositions $\mathbf{x}$ in $|\varphi_2\rangle$ have $a^{\overline{\mathbf{x}}}$ Mod N as the output? Answer: $\lfloor \frac{2^m}{r} \rfloor$. So

$$|\varphi_3\rangle = \frac{\sum_{a^{\mathbf{x}} \equiv a^{\overline{\mathbf{x}}} \text{ Mod } N} |\mathbf{x}, a^{\overline{\mathbf{x}}} \text{ Mod } N\rangle}{\lfloor \frac{2^m}{r} \rfloor}. \tag{6.177}$$

We might also write this as

$$|\varphi_3\rangle = \frac{\sum_{j=0}^{2^m/r-1} |t_0 + jr, a^{\overline{\mathbf{x}}} \text{ Mod } N\rangle}{\left[\frac{2^m}{r} \right]}, \tag{6.178}$$

where t_0 is the first time that $a^{t_0} \equiv a^{\overline{\mathbf{x}}}$ Mod N, i.e., the first time that the measured value occurs. We shall call t_0 the **offset** of the period for reasons that will soon become apparent.

It is important to realize that this stage employs entanglement in a serious fashion. The top qubits and the bottom qubits are entangled in a way that when the top is measured, the bottom stays the same.

Example 6.5.8 Continuing Example 6.5.6, let us say that after measurement of the bottom qubits, 7 is found. In that case $|\varphi_3\rangle$ would be

$$\frac{|3,7\rangle + |7,7\rangle + |11,7\rangle + |15,7\rangle + \cdots + |251,7\rangle + |255,7\rangle}{\left[\frac{256}{4} \right]}. \tag{6.179}$$

For example, if we looked at the $f_{13,15}$ rather than the bargraph in Figure 6.3, we would get the bargraph shown in Figure 6.5. □

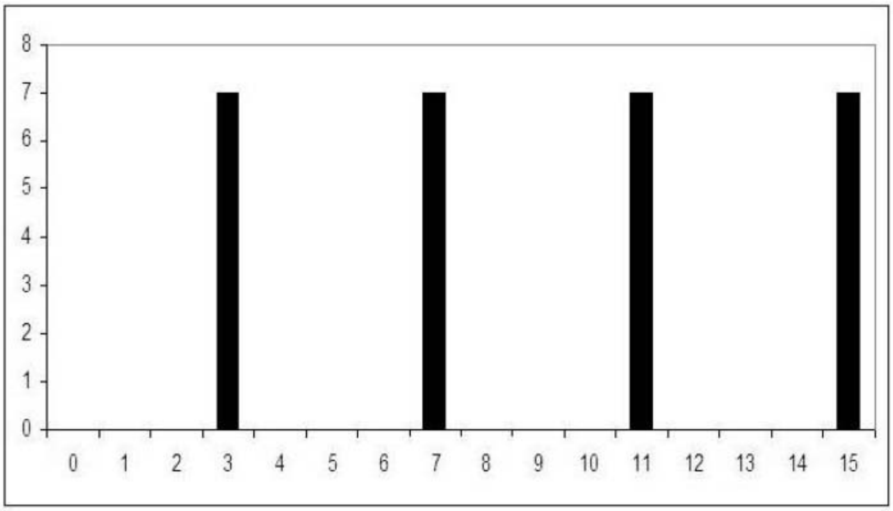


Figure 6.5. $f_{13,15}$ after a measurement of 7.

Example 6.5.9 Continuing Example 6.5.7, let us say that after measurement of the bottom qubits we find 222 (which is 24^5 Mod 371.) In that case $|\varphi_3\rangle$ would be

$$\frac{|5,222\rangle + |83,222\rangle + |161,222\rangle + |239,222\rangle + \cdots}{\left\lceil \frac{2^{18}}{78} \right\rceil}. \tag{6.180}$$

We can see the result of this measurement in Figure 6.6 □

Exercise 6.5.6 Continuing Exercise 6.5.5, let us say that after measuring the bottom qubits, 55 is found. What would $|\varphi_3\rangle$ be? ■

The final step of the quantum part of the algorithm is to take such a superposition and return its period. This will be done with a type of **Fourier transform**. We do not

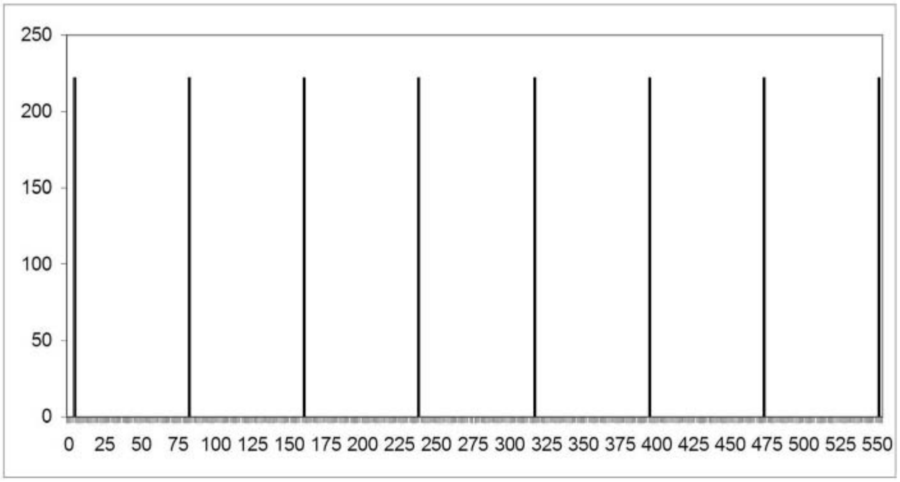


Figure 6.6. $f_{24,371}$ after a measurement of 222.

assume the reader has seen this before and some motivation is in order. Let us step away from our task at hand and talk about evaluating polynomials. Consider the polynomial

$$P(x) = a_0 + a_1x^1 + a_2x^2 + a_3x^3 + \cdots + a_{n-1}x^{n-1}. \tag{6.181}$$

We can represent this polynomial with a column vector $[a_0, a_1, a_2, \ldots, a_{n-1}]^T$. Suppose we wanted to evaluate this polynomial at the numbers $x_0, x_1, x_2, \ldots, x_{n-1}$, i.e., we wanted to find $P(x_0)$, $P(x_1)$, $P(x_2)$, $\ldots$, $P(x_{n-1})$. A simple way of performing the task is with the following matrix multiplication:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^j & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^j & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^j & \cdots & x_2^{n-1} \\ \vdots & & & & \vdots & & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^j & \cdots & x_k^{n-1} \\ \vdots & & & & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^j & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_k \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(x_0) \\ P(x_1) \\ P(x_2) \\ \vdots \\ P(x_k) \\ \vdots \\ P(x_{n-1}) \end{bmatrix}. \tag{6.182}$$

The matrix on the left, where every row is a geometric series, is called the **Vandermonde matrix** and is denoted $\mathcal{V}(x_0, x_1, x_2, x_{n-1})$. There is no restriction on the type of numbers we are permitted to use in the Vandermonde matrix, and hence, we are permitted to use complex numbers. In fact, we shall need them to be powers of the Mth roots of unity, ω_M (see page 26 of Chapter 1 for a quick reminder). Because M is fixed throughout this discussion, we shall simply denote this as ω. There is also no restriction on the size of the Vandermonde matrix. Letting $M = 2^m$, which is the amount of numbers that can be described with the top qubits, there is a need for the Vandermonde matrix to be an M-by-M matrix. We would like to evaluate the polynomials at $\omega^0 = 1, \omega, \omega^2, \ldots, \omega^{M-1}$. To do this, we need to look at $\mathcal{V}(\omega^0, \omega^1, \omega^2, \ldots, \omega^{M-1})$. In order to evaluate $P(x)$ at the powers of the Mth root of unity, we must multiply

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \cdots & \omega^j & \cdots & \omega^{M-1} \\ 1 & \omega^2 & \omega^{2\times 2} & \cdots & \omega^{2j} & \cdots & \omega^{2(M-1)} \\ \vdots & & & & \vdots & & \vdots \\ 1 & \omega^k & \omega^{k2} & \cdots & \omega^{kj} & \cdots & \omega^{k(M-1)} \\ \vdots & & & & \vdots & & \vdots \\ 1 & \omega^{M-1} & \omega^{(M-1)2} & \cdots & \omega^{(M-1)j} & \cdots & \omega^{(M-1)(M-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_k \\ \vdots \\ a_{M-1} \end{bmatrix} = \begin{bmatrix} P(\omega^0) \\ P(\omega^1) \\ P(\omega^2) \\ \vdots \\ P(\omega^k) \\ \vdots \\ P(\omega^{M-1}) \end{bmatrix}. \tag{6.183}$$

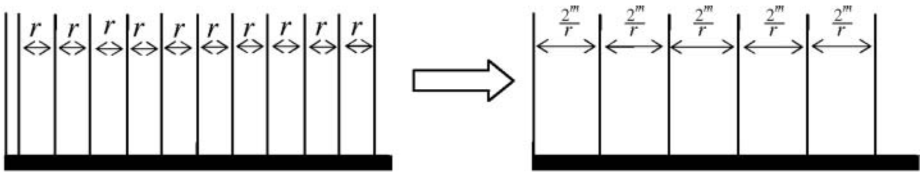


Figure 6.7. The action of $DFT^{\dagger}$.

$[P(\omega^0), P(\omega^1), P(\omega^2), \ldots, P(\omega^k), \ldots, P(\omega^{M-1})]^T$ is the vector of the values of the polynomial at the powers of the Mth root of unity.

Let us define the **discrete Fourier transform**, denoted DFT, as

$$DFT = \frac{1}{\sqrt{M}}\mathcal{V}(\omega^0, \omega^1, \omega^2, \ldots, \omega^{M-1}). \tag{6.184}$$

Formally, DFT is defined as

$$DFT[j,k] = \frac{1}{\sqrt{M}}\omega^{jk}. \tag{6.185}$$

It is easy to see that DFT is a unitary matrix: the adjoint of this matrix, $DFT^{\dagger}$, is formally defined as

$$DFT^{\dagger}[j,k] = \frac{1}{\sqrt{M}}\overline{\omega^{kj}} = \frac{1}{\sqrt{M}}\omega^{-jk}. \tag{6.186}$$

To show that DFT is unitary, let us multiply

$$(DFT \star DFT^{\dagger})[j,k] = \frac{1}{M}\sum_{i=0}^{M-1}(\omega^{ji}\omega^{-ik}) = \sum_{i=0}^{M-1}\omega^{-i(k-j)}. \tag{6.187}$$

If $k = j$, i.e., if we are along the diagonal, this becomes

$$\frac{1}{M}\sum_{i=0}^{M-1}\omega^0 = \frac{1}{M}\sum_{i=0}^{M-1}1 = 1. \tag{6.188}$$

If $k \neq j$, i.e., if we are off the diagonal, then we get a geometric progression which sums to 0. And so $DFT \star DFT^{\dagger} = I$.

What task does $DFT^{\dagger}$ perform? Our text will not get into the nitty-gritty of this important operation, but we shall try to give an intuition of what is going on. Let us forget about the normalization $\frac{1}{\sqrt{M}}$ for a moment and think about this intuitively. The matrix DFT acts on polynomials by evaluating them on different equally spaced points of the circle. The outcomes of those evaluations will necessarily have periodicity because the points go around and around the circle. So multiplying a column vector with DFT takes a sequence and outputs a periodic sequence. If we start with a periodic column vector, then the DFT will transform the periodicity. Similarly, the inverse of the Fourier transform, $DFT^{\dagger}$, will also change the periodicity. Suffice it to say that the $DFT^{\dagger}$ does two tasks as shown in Figure 6.7:

- It modifies the period from r to $\frac{2^m}{r}$.
- It eliminates the offset.

Circuit (6.166) requires a variant of a DFT called a **quantum Fourier transform** and denoted as QFT. Its inverse is denoted $QFT^\dagger$. The $QFT^\dagger$ performs the same operation but is constructed in a way that is more suitable for quantum computers. (We shall not delve into the details of its construction.) The quantum version is very fast and made of "small" unitary operators that are easy for a quantum computer to implement.[3]

The final step of the circuit is to measure the top qubits. For our presentation, we shall make the simplifying assumption that r evenly divides into 2^m. Shor's actual algorithm does not make this assumption and goes into details about finding the period for any r. When we measure the top qubit we will find it to be some multiple of $\frac{2^m}{r}$. That is, we will measure

$$x = \frac{\lambda 2^m}{r} \tag{6.191}$$

for some whole number λ. We know 2^m, and after measuring we will also know x. We can divide the whole number x by 2^m and get

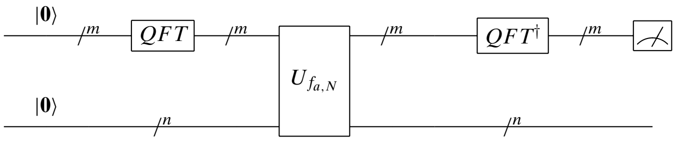
$$\frac{x}{2^m} = \frac{\lambda 2^m}{r 2^m} = \frac{\lambda}{r}. \tag{6.192}$$

One can then reduce this number to an irreducible fraction and take the denominator to be the long sought-after r. If we do not make the simplifying assumption that r evenly divides into 2^m, then we might have to perform this process several times and analyze the results.

From the Period to the Factors. Let us see how knowledge of the period r will help us find a factor of N. We shall need a period that is an even number. There is a theorem of number theory that tells us that for the majority of a, the period of $f_{a,N}$ will be an even number. If, however, we do choose an a such that the period is an odd number, simply throw that a away and choose another one. Once an even r is found so that

$$a^r \equiv 1 \text{ Mod } N, \tag{6.193}$$

[3] There are slight variations of Shor's algorithm: For one, rather than using the $H^{\otimes m}$ to put the m qubits in a superposition in the beginning of circuit (6.166), we could have used QFT and get the same results. However, we leave it as is because at this point the reader has familiarity with the Hadamard matrix.
Another variation is not measuring the bottom qubits before performing the $QFT^\dagger$ operation. This makes the mathematics slightly more complicated. We leave it as is for simplicity sakes.
However, if we take both of these variants, our quantum circuit would look like



(6.189)

This would have been more in line with our discussion at the end of Section 2.3, where we wrote about solving problems using

$$Translation \mapsto Calculation \mapsto Reverse\ \ Translation \tag{6.190}$$

where QFT and $QFT^\dagger$ would be our two translations.

we may subtract 1 from both sides of the equivalence to get

$$a^r - 1 \equiv 0 \text{ Mod } N, \tag{6.194}$$

or equivalently

$$N|(a^r - 1). \tag{6.195}$$

Remembering that $1 = 1^2$ and $x^2 - y^2 = (x + y)(x - y)$ we get that

$$N|(\sqrt{a^r} + 1)(\sqrt{a^r} - 1) \tag{6.196}$$

or

$$N|(a^{\frac{r}{2}} + 1)(a^{\frac{r}{2}} - 1). \tag{6.197}$$

(If r was odd, we would not be able to evenly divide by 2.) This means that any factor of N is also a factor of either$(a^{\frac{r}{2}} + 1)$ or $(a^{\frac{r}{2}} - 1)$ or both. Either way, a factor for N can be found by looking at

$$\text{GCD}((a^{\frac{r}{2}} + 1), N) \tag{6.198}$$

and

$$\text{GCD}((a^{\frac{r}{2}} - 1), N). \tag{6.199}$$

Finding the GCD can be done with the classical Euclidean algorithm. There is, however, one caveat. We must make sure that

$$a^{\frac{r}{2}} \neq -1 \text{ Mod } N \tag{6.200}$$

because if $a^{\frac{r}{2}} \equiv -1$ Mod N, then the right side of Equation (6.197) would be 0. In that case we do not get any information about N and must throw away that particular a and start over again.

Let us work out some examples.

Example 6.5.10 In chart (6.151), we saw that the period of $f_{2,15}$ is 4, i.e., $2^4 \equiv 1$ Mod 15. From Equation (6.197), we get that

$$15|(2^2 + 1)(2^2 - 1). \tag{6.201}$$

And, hence, we have that GCD(5, 15) = 5 and GCD(3, 15) = 3. □

Example 6.5.11 In chart (6.159), we saw that the period of $f_{6,371}$ is 26, i.e., $6^{26} \equiv 1$ Mod 371. However, we can also see that $6^{\frac{26}{2}} = 6^{13} \equiv 370 \equiv -1$ Mod 371. So we cannot use $a = 6$. □

Example 6.5.12 In chart (6.160), we saw that the period of $f_{24,371}$ is 78, i.e., $24^{78} \equiv 1$ Mod 371. We can also see that $24^{\frac{78}{2}} = 24^{39} \equiv 160 \neq -1$ Mod 371. From Equation (6.197), we get that

$$371|(24^{39} + 1)(24^{39} - 1). \tag{6.202}$$

And, thus, GCD(161, 371) = 7 and GCD(159, 371) = 53 and $371 = 7 * 53$. □

Exercise 6.5.7 Use the fact that the period of $f_{7,247}$ is 12 to determine the factors of 247. ■

Shor's Algorithm. We are, at last, ready to put all the pieces together and formally state **Shor's algorithm**:
Input: A positive integer N with $n = \lceil \log_2 N \rceil$.
Output: A factor p of N if it exists.

Step 1. Use a polynomial algorithm to determine if N is prime or a power of prime. If it is a prime, declare that it is and exit. If it is a power of a prime number, declare that it is and exit.

Step 2. Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine GCD(a, N). If the GCD is not 1, then return it and exit.

Step 3. Use quantum circuit (6.166) to find a period r.

Step 4. If r is odd or if $a^r \equiv -1$ Mod N, then return to Step 2 and choose another a.

Step 5. Use Euclid's algorithm to calculate GCD$((a^{\frac{r}{2}} + 1), N)$ and GCD$((a^{\frac{r}{2}} - 1), N)$. Return at least one of the nontrivial solutions.

What is the worst case complexity of this algorithm? To determine this, one needs to have an in-depth analysis of the details of how $U_{a,N}$ and $QFT^{\dagger}$ are implemented. One would also need to know what percentage of times things can go wrong. For example, what percentage of a would $f_{a,N}$ have an odd period? Rather than going into the gory details, let us just state that Shor's algorithm works in

$$O(n^2 \log n \log \log n) \tag{6.203}$$

number of steps, where n is the number of bits needed to represent the number N. That is polynomial in terms of n. This is in contrast to the best-known classical algorithms that demand

$$O(e^{cn^{1/3} \log^{2/3} n}) \tag{6.204}$$

steps, where c is some constant. This is exponential in terms of n. Shor's quantum algorithm is indeed faster.

Appendix: Implementing $U_{f_{a,N}}$ with quantum gates. In order for $U_{f_{a,N}}$ to be implemented with unitary matrices, we need to "break up" the operations into small little jobs. This is done by splitting up x. Let us write x in binary. That is,

$$\mathbf{x} = x_{n-1}x_{n-2}\cdots x_2x_1x_0. \tag{6.205}$$

Formally, x as a number is

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_2 2^2 + x_1 2 + x_0. \tag{6.206}$$

Using this description of x, we can rewrite our function as

$$f_{a,N}(x) = a^x \text{ Mod } N = a^{x_{n-1}2^{n-1}+x_{n-2}2^{n-2}+\cdots+x_2 2^2+x_1 2+x_0} \text{ Mod } N \tag{6.207}$$

or

$$a^{x_{n-1}2^{n-1}} \times a^{x_{n-2}2^{n-2}} \times \cdots \times a^{x_2 2^2} \times a^{x_1 2} \times a^{x_0} \text{ Mod } N. \tag{6.208}$$

We can convert this formula to an inductive definition[4] of $f_{a,N}(x)$. We shall define $y_0, y_1, y_2, \ldots, y_{n-2}, y_{n-1}$, where $y_{n-1} = f_{a,N}(x)$: the base case is

$$y_0 = a^{x_0}. \tag{6.209}$$

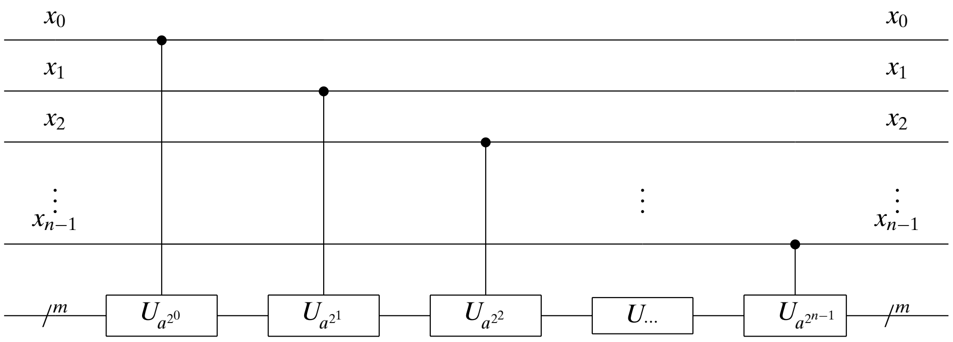
If we have y_{j-1}, then to get y_j we use the trick from Equation (6.157):

$$y_j = y_{j-1} \times a^{x_j 2^j} \text{ Mod } N. \tag{6.210}$$

Notice that if $x_j = 0$ then $y_j = y_{j-1}$. In other words, whether or not we should multiply y_{j-1} by a^{2^j} Mod N is dependent on whether or not $x_j = 1$. It turns out that as long as a and N are co-prime, the operation of multiplying a number times a^{2^j} Mod N is reversible and, in fact, unitary. So for each j, there is a unitary operator

$$U_{a^{2^j} \text{ Mod } N} \tag{6.211}$$

that we shall write as $U_{a^{2^j}}$. As we want to perform this operation conditionally, we will need controlled-$U_{a^{2^j}}$, or ${}^C U_{a^{2^j}}$, gates. Putting this all together, we have the following quantum circuit that implements $f_{a,N}$ in a polynomial number of gates:



(6.212)

Even if a real implementation of large-scale quantum computers is years away, the design and study of quantum algorithms is something that is ongoing and is an exciting field of interest.

References:

(i) A version of Deutsch's algorithm was first stated in Deutsch (1985).
(ii) Deutsch–Jozsa was given in Deutsch and Jozsa (1992).
(iii) Simon's algorithm was first presented in Simon (1994).

[4] This inductive definition is nothing more than the modular-exponentiation algorithm given in, say, Section 31.6 of Corman et al. (2001) or Section 1.2 of Dasgupta, Papadimitriou, and Vazirani (2006).

(iv) Grover's search algorithm was originally presented in Grover (1997). Further developments of the algorithm can be found in Chapter 6 of Nielsen and Chuang (2000). For nice applications of Grover's algorithm to graph theory, see Cirasella (2006).

(v) Shor's algorithm was first announced in Shor (1994). There is also a very readable presentation of it in Shor (1997). There are several slight variations to the algorithm and there are many presentations at different levels of complexity. Chapter 5 of Nielsen and Chuang (2000) goes through it thoroughly. Chapter 10 of Dasgupta, Papadimitriou, and Vazirani (2006) goes from an introduction to quantum computing through Shor's algorithm in 20 pages.

Every quantum computer textbook works through several algorithms. See, e.g., Hirvensalo (2001) and Kitaev, Shen, and Vyalyi (2002) and, of course, Nielsen and Chuang (2000). There is also a very nice short article by Shor that discusses several algorithms (Shor, 2002). Dorit Aharonov has written a nice survey article that goes through many of the algorithms (Aharonov, 1998)

Peter Shor has written a very interesting article on the seeming paucity of quantum algorithms in Shor (2003).