Shor's algorithm

Now we'll turn our attention to the integer factorization problem, and see how it can be solved efficiently on a quantum computer using phase estimation. The algorithm we'll obtain is *Shor's algorithm for integer factorization*. Shor didn't describe his algorithm specifically in terms of phase estimation, but it is a natural and intuitive way to explain how it works.

We'll begin by discussing an intermediate problem known as the *order-finding problem* and see how phase estimation provides a solution to this problem. We'll then see how an efficient solution to the order-finding problem gives us an efficient solution to the integer factorization problem. (When a solution to one problem provides a solution to another problem like this, we say that the second problem *reduces* to the first — so in this case we're reducing integer factorization to order finding.) This second part of Shor's algorithm doesn't make use of quantum computing at all; it's completely classical. Quantum computing is only needed to solve order finding.

The order-finding problem

Some basic number theory

To explain the order-finding problem and how it can be solved using phase estimation, it will be helpful to begin with a couple of basic number theory concepts, and to introduce some handy notation along the way.

To begin, for any given positive integer N, define the set $\mathbb{Z}_N$ like this.

$$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$$

For instance, $\mathbb{Z}_1 = \{0\},\ \mathbb{Z}_2 = \{0, 1\},\ \mathbb{Z}_3 = \{0, 1, 2\},$ and so on.

These are sets of numbers, but we can think of them as more than sets. In particular, we can think about *arithmetic operations* on $\mathbb{Z}_N$ such as

addition and multiplication — and if we agree to always take our answers modulo N (that is, divide by N and take the remainder as the result), we'll always stay within this set when we perform these operations. The two specific operations of addition and multiplication, both taken modulo N, turn $\mathbb{Z}_N$ into a *ring*, which is a fundamentally important type of object in algebra.

For example, 3 and 5 are elements of $\mathbb{Z}_7$, and if we multiply them together we get $3 \cdot 5 = 15$, which leaves a remainder of 1 when divided by 7. Sometimes we express this as follows.

$$3 \cdot 5 \equiv 1 \pmod 7$$

But we can also simply write $3 \cdot 5 = 1$, provided that it's been made clear that we're working in $\mathbb{Z}_7$, just to keep our notation as simple as possible.

As an example, here are the addition and multiplication tables for $\mathbb{Z}_6$.

$+$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

$\cdot$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

Among the N elements of $\mathbb{Z}_N$, the elements $a \in \mathbb{Z}_N$ that satisfy $\gcd(a, N) = 1$ are special. Frequently the set containing these elements is denoted with a star like so.

$$\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N : \gcd(a, N) = 1\}$$

If we focus our attention on the operation of multiplication, the set $\mathbb{Z}_N^*$ forms a *group* — specifically an *abelian group* — which is another important type of object in algebra. It's a basic fact about these sets (and finite groups in general), that if we pick any element $a \in \mathbb{Z}_N^*$ and repeatedly multiply a to itself, we'll always eventually get the number 1.

For a first example, let's take $N = 6$. We have that $5 \in \mathbb{Z}_6^*$ because $\gcd(5, 6) = 1$, and if we multiply 5 to itself we get 1, as the table above confirms.

$$5^2 = 1 \quad (\text{working within } \mathbb{Z}_6)$$

As a second example, let's take $N = 21$. If we go through the numbers from 0 to 20, the ones having GCD equal to 1 with 21 are as follows.

$$\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$$

For each of these elements, it is possible to raise that number to a positive integer power to get 1. Here are the smallest powers for which this works:

$$\begin{array}{lll} 1^1 = 1 & 8^2 = 1 & 16^3 = 1 \\ 2^6 = 1 & 10^6 = 1 & 17^6 = 1 \\ 4^3 = 1 & 11^6 = 1 & 19^6 = 1 \\ 5^6 = 1 & 13^2 = 1 & 20^2 = 1 \end{array}$$

Naturally we're working within $\mathbb{Z}_{21}$ for all of these equations, which we haven't bothered to write — we take it to be implicit to avoid cluttering things up. We'll continue to do that throughout the rest of the lesson.

Problem statement and connection to phase estimation

Now we can state the order-finding problem.

> **Order finding**
> Input: positive integers N and a satisfying $\gcd(N, a) = 1$
> Output: the smallest positive integer r such that $a^r \equiv 1 \pmod N$

Alternatively, in terms of the notation we just introduced above, we're given $a \in \mathbb{Z}_N^*$, and we're looking for the smallest positive integer r such that $a^r = 1$. This number r is called the *order* of a modulo N.

To connect the order-finding problem to phase estimation, let's think about the operation defined on a system whose classical states correspond to $\mathbb{Z}_N$, where we multiply by a fixed element $a \in \mathbb{Z}_N^*$.

$$M_a|x\rangle = |ax\rangle \qquad (\text{for each } x \in \mathbb{Z}_N)$$

To be clear, we're doing the multiplication in $\mathbb{Z}_N$, so it's implicit that we're taking the product modulo N inside of the ket on the right-hand side of the equation.

For example, if we take $N = 15$ and $a = 2$, then the action of M_2 on the standard basis $\{|0\rangle, \ldots, |14\rangle\}$ is as follows.

$$
\begin{array}{lll}
M_2|0\rangle = |0\rangle & M_2|5\rangle = |10\rangle & M_2|10\rangle = |5\rangle \\
M_2|1\rangle = |2\rangle & M_2|6\rangle = |12\rangle & M_2|11\rangle = |7\rangle \\
M_2|2\rangle = |4\rangle & M_2|7\rangle = |14\rangle & M_2|12\rangle = |9\rangle \\
M_2|3\rangle = |6\rangle & M_2|8\rangle = |1\rangle & M_2|13\rangle = |11\rangle \\
M_2|4\rangle = |8\rangle & M_2|9\rangle = |3\rangle & M_2|14\rangle = |13\rangle
\end{array}
$$

This is a unitary operation provided that $\gcd(a, N) = 1$; it shuffles the elements of the standard basis $\{|0\rangle, \ldots, |N-1\rangle\}$, so as a matrix it's a permutation matrix. It's evident from its definition that this operation is deterministic, and a simple way to see that it's invertible is to think about the order r of a modulo N, and to recognize that the inverse of M_a is M_a^{r-1}.

$$M_a^{r-1} M_a = M_a^r = M_{a^r} = M_1 = \mathbb{I}$$

There's another way to think about the inverse that doesn't require any knowledge of r (which, after all, is what we're trying to compute). For every element $a \in \mathbb{Z}_N^*$ there's always a unique element $b \in \mathbb{Z}_N^*$ that satisfies $ab = 1$. We denote this element b by a^{-1}, and it can be computed efficiently; an extension of Euclid's GCD algorithm does it with cost quadratic in $\lg(N)$. And thus

$$M_{a^{-1}} M_a = M_{a^{-1}a} = M_1 = \mathbb{I}.$$

So, the operation M_a is both deterministic and invertible. That implies that it's described by a permutation matrix, and is therefore unitary.

Now let's think about the eigenvectors and eigenvalues of the operation M_a, assuming that $a \in \mathbb{Z}_N^*$. As was just argued, this assumption tells us that M_a is unitary.

There are N eigenvalues of M_a, possibly including the same eigenvalue repeated multiple times, and in general there's some freedom in selecting corresponding eigenvectors — but we won't need to worry about all of the possibilities. Let's start simply and identify just one eigenvector of M_a.

$$|\psi_0\rangle = \frac{|1\rangle + |a\rangle + \cdots + |a^{r-1}\rangle}{\sqrt{r}}$$

The number r is the order of a modulo N, here and throughout the remainder of the lesson. The eigenvalue associated with this eigenvector is 1 because it isn't changed when we multiply by a.

$$M_a|\psi_0\rangle = \frac{|a\rangle + \cdots + |a^{r-1}\rangle + |a^r\rangle}{\sqrt{r}} = \frac{|a\rangle + \cdots + |a^{r-1}\rangle + |1\rangle}{\sqrt{r}} =$$

This happens because $a^r = 1$, so each standard basis state $|a^k\rangle$ gets shifted to $|a^{k+1}\rangle$ for $k \leq r-1$, and $|a^{r-1}\rangle$ gets shifted back to $|1\rangle$. Informally speaking, it's like we're slowly stirring $|\psi_0\rangle$, but it's already completely stirred so nothing changes.

Here's another example of an eigenvector of M_a. This one happens to be more interesting in the context of order finding and phase estimation.

$$|\psi_1\rangle = \frac{|1\rangle + \omega_r^{-1}|a\rangle + \cdots + \omega_r^{-(r-1)}|a^{r-1}\rangle}{\sqrt{r}}$$

Alternatively, we can write this vector using a summation as follows.

$$|\psi_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \omega_r^{-k} |a^k\rangle$$

Here we're seeing the complex number $\omega_r = e^{2\pi i/r}$ showing up naturally, due to the way that multiplication by a works modulo N. This time the corresponding eigenvalue is ω_r. To see this, we can first compute as follows.

$$M_a|\psi_1\rangle = \sum_{k=0}^{r-1} \omega_r^{-k} M_a|a^k\rangle = \sum_{k=0}^{r-1} \omega_r^{-k}|a^{k+1}\rangle = \sum_{k=1}^{r} \omega_r^{-(k-1)}|a^k\rangle = \omega$$

Then, because $\omega_r^{-r} = 1 = \omega_r^0$ and $|a^r\rangle = |1\rangle = |a^0\rangle$, we see that

$$\sum_{k=1}^{r} \omega_r^{-k}|a^k\rangle = \sum_{k=0}^{r-1} \omega_r^{-k}|a^k\rangle = |\psi_1\rangle,$$

so $M_a|\psi_1\rangle = \omega_r|\psi_1\rangle$.

Using the same reasoning, we can identify additional eigenvector/eigenvalue pairs for M_a. For any choice of $j \in \{0, \ldots, r-1\}$ we have that

$$|\psi_j\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \omega_r^{-jk} |a^k\rangle$$

is an eigenvector of M_a whose corresponding eigenvalue is ω_r^j.

$$M_a|\psi_j\rangle = \omega_r^j|\psi_j\rangle$$

There are other eigenvectors of M_a, but we don't need to concern ourselves with them — we'll focus solely on the eigenvectors $|\psi_0\rangle, \ldots, |\psi_{r-1}\rangle$ that we've just identified.

Order finding through phase estimation

To solve the order-finding problem for a given choice of $a \in \mathbb{Z}_N^*$, we can apply the phase-estimation procedure to the operation M_a.

To do this, we need to implement not only M_a efficiently with a quantum circuit, but also M_a^2, M_a^4, M_a^8, and so on, going as far as needed to obtain a precise enough estimate from the phase estimation procedure. Here we'll explain how this can be done, and we'll figure out exactly how much precision is needed later.

Let's start with the operation M_a by itself. Naturally, because we're working with the quantum circuit model, we'll use binary notation to encode the numbers between 0 and $N-1$. The largest number we need to encode is $N-1$, so the number of bits we need is

$$n = \lg(N-1) = \lfloor \log(N-1) \rfloor + 1.$$

For example, if $N = 21$ we have $n = \lg(N-1) = 5$. Here's what the encoding of elements of $\mathbb{Z}_{21}$ as binary strings of length 5 looks like.

$$\begin{aligned} 0 &\mapsto 00000 \\ 1 &\mapsto 00001 \\ &\vdots \\ 20 &\mapsto 10100 \end{aligned}$$

And now, here's a precise definition of how M_a is defined as an n-qubit operation.

$$M_a|x\rangle = \begin{cases} |ax \pmod N\rangle & 0 \leq x < N \\ |x\rangle & N \leq x < 2^n \end{cases}$$

The point is that although we only care about how M_a works for $|0\rangle, \ldots, |N-1\rangle$, we do have to specify how it works for the remaining $2^n - N$ standard basis states — and we need to do this in a way that still

gives us a unitary operation. Defining M_a so that it does nothing to the remaining standard basis states accomplishes this.

Using the algorithms for integer multiplication and division discussed in the previous lesson, together with the methodology for reversible, garbage-free implementations of them, we can build a quantum circuit that performs M_a, for any choice of $a \in \mathbb{Z}_N^*$, at cost $O(n^2)$. Here is one way this can be done.

1. Build a circuit for performing the operation

   $$|x\rangle|y\rangle \mapsto |x\rangle|y \oplus f_a(x)\rangle$$

   where

   $$f_a(x) = \begin{cases} ax \pmod N & 0 \leq x < N \\ x & N \leq x < 2^n \end{cases}$$

   using the method described in the previous lesson. This gives us a circuit of size $O(n^2)$.

2. Swap the two n-qubit systems using n swap gates to swap the qubits individually.

3. Along similar lines to the first step, build a circuit for the operation

   $$|x\rangle|y\rangle \mapsto |x\rangle\big|y \oplus f_{a^{-1}}(x)\big\rangle$$

   where a^{-1} is the inverse of a in $\mathbb{Z}_N^*$.

By initializing the bottom n qubits and composing the three steps, we obtain this transformation:

$$|x\rangle|0^n\rangle \overset{\text{step 1}}{\mapsto} |x\rangle|f_a(x)\rangle \overset{\text{step 2}}{\mapsto} |f_a(x)\rangle|x\rangle \overset{\text{step 3}}{\mapsto} |f_a(x)\rangle\big|x \oplus f_{a^{-1}}(f_a(x$$

The method requires workspace qubits, but they're returned to their initialized state at the end, which allows us to use these circuits for phase estimation. The total cost of the circuit we obtain is $O(n^2)$.

To perform M_a^2, M_a^4, M_a^8, and so on, we can use exactly the same method, except that we replace a with a^2, a^4, a^8, and so on, as elements of $\mathbb{Z}_N^*$. That is, for any power k we choose, we can create a circuit for M_a^k not by iterating k times the circuit for M_a, but instead by computing $b = a^k \in \mathbb{Z}_N^*$ and then using the circuit for M_b.

The computation of powers $a^k \in \mathbb{Z}_N$ is the *modular exponentiation* problem mentioned in the previous lesson. This computation can be done

classically, using the algorithm for modular exponentiation mentioned in the previous lesson (often called the *power algorithm* in computational number theory). In fact, we only require *power-of-2* powers of a, in particular $a^2, a^4, \ldots a^{2^{m-1}} \in \mathbb{Z}_N^*$, and we can obtain these powers by iteratively squaring $m-1$ times. Each squaring can be performed by a Boolean circuit of size $O(n^2)$.

In essence, what we're effectively doing here is offloading the problem of iterating M_a as many as 2^{m-1} times to an efficient classical computation. And it's good fortune that this is possible! For an arbitrary choice of a quantum circuit in the phase estimation problem, this is not likely to be possible — and in that case the resulting cost for phase estimation grows *exponentially* in the number of control qubits m.

Solution given a convenient eigenvector

To understand how we can solve the order-finding problem using phase estimation, let's start by supposing that we run the phase estimation procedure on the operation M_a using the eigenvector $|\psi_1\rangle$. Getting our hands on this eigenvector isn't easy, as it turns out, so this won't be the end of the story — but it's helpful to start here.

The eigenvalue of M_a corresponding to the eigenvector $|\psi_1\rangle$ is

$$\omega_r = e^{2\pi i \frac{1}{r}}.$$

That is, $\omega_r = e^{2\pi i \theta}$ for $\theta = 1/r$. So, if we run the phase estimation procedure on M_a using the eigenvector $|\psi_1\rangle$, we'll get an approximation to $1/r$. By computing the reciprocal we'll be able to learn r — provided that our approximation is good enough.

In more detail, when we run the phase-estimation procedure using m control qubits, what we obtain is a number $y \in \{0, \ldots, 2^m - 1\}$. We then take $y/2^m$ as a guess for θ, which is $1/r$ in the case at hand. To figure out what r is from this approximation, the natural thing to do is to compute the reciprocal of our approximation and round to the nearest integer.

$$\left\lfloor \frac{2^m}{y} + \frac{1}{2} \right\rfloor$$

For example, let's suppose $r = 6$ and we perform phase estimation on M_a with the eigenvector $|\psi_1\rangle$ using $m = 5$ control bits. The best 5-bit approximation to $1/r = 1/6$ is $5/32$, and we have a pretty good chance

(about 68% in this case) to obtain the outcome $y = 5$ from phase estimation. We have

$$\frac{2^m}{y} = \frac{32}{5} = 6.4,$$

and rounding to the nearest integer gives 6, which is the correct answer.

On the other hand, if we don't use enough precision, we might not get the right answer. For instance, if we take $m = 4$ control qubits in phase estimation, we might obtain the best 4-bit approximation to $1/r = 1/6$, which is $3/16$. Taking the reciprocal yields

$$\frac{2^m}{y} = \frac{16}{3} = 5.333\cdots$$

and rounding to the nearest integer gives an incorrect answer of 5.

So how much precision do we need to get the right answer? We know that the order r is an integer, and intuitively speaking what we need is enough precision to distinguish $1/r$ from nearby possibilities, including $1/(r+1)$ and $1/(r-1)$. The closest number to $1/r$ that we need to be concerned with is $1/(r+1)$, and the distance between these two numbers is

$$\frac{1}{r} - \frac{1}{r+1} = \frac{1}{r(r+1)}.$$

So, if we want to make sure that we don't mistake $1/r$ for $1/(r+1)$, it suffices to use enough precision to guarantee that a best approximation $y/2^m$ to $1/r$ is closer to $1/r$ than it is to $1/(r+1)$. If we use enough precision so that

$$\left| \frac{y}{2^m} - \frac{1}{r} \right| < \frac{1}{2r(r+1)},$$

so that the error is less than half of the distance between $1/r$ and $1/(r+1)$, then $y/2^m$ will be closer to $1/r$ than to any other possibility, including $1/(r+1)$ and $1/(r-1)$.

We can double-check this as follows. Suppose that

$$\frac{y}{2^m} = \frac{1}{r} + \varepsilon$$

for ε satisfying

$$|\varepsilon| < \frac{1}{2r(r+1)}.$$

When we take the reciprocal we obtain

$$\frac{2^m}{y} = \frac{1}{\frac{1}{r} + \varepsilon} = \frac{r}{1 + \varepsilon r} = r - \frac{\varepsilon r^2}{1 + \varepsilon r}.$$

By maximizing in the numerator and minimizing in the denominator, we can bound how far away we are from r as follows.

$$\left| \frac{\varepsilon r^2}{1 + \varepsilon r} \right| \leq \frac{\frac{r^2}{2r(r+1)}}{1 - \frac{r}{2r(r+1)}} = \frac{r}{2r + 1} < \frac{1}{2}$$

We're less than $1/2$ away from r, so as expected we'll get r when we round.

Unfortunately, because we don't yet know what r is, we can't use it to tell us how much accuracy we need. What we can do instead is to use the fact that r must be smaller than N to ensure that we use enough precision. In particular, if we use enough accuracy to guarantee that the best approximation $y/2^m$ to $1/r$ satisfies

$$\left| \frac{y}{2^m} - \frac{1}{r} \right| \leq \frac{1}{2N^2},$$

then we'll have enough precision to correctly determine r when we take the reciprocal. Taking $m = 2\lg(N) + 1$ ensures that we have a high chance to obtain an estimation with this precision using the method described previously. (Taking $m = 2\lg(N)$ is good enough if we're comfortable with a lower bound of 40% on the probability of success.)

General solution

As we just saw, if we have the eigenvector $|\psi_1\rangle$ of M_a, we can learn r through phase estimation, so long as we use enough control qubits to do this with sufficient precision. Unfortunately, it's not easy to get our hands on the eigenvector $|\psi_1\rangle$, so we need to figure out how to proceed.

Let's suppose momentarily that we proceed just like above, except with the eigenvector $|\psi_k\rangle$ in place of $|\psi_1\rangle$, for any choice of $k \in \{0, \ldots, r - 1\}$ that we choose to think about. The result we get from the phase estimation procedure will be an approximation

$$\frac{y}{2^m} \approx \frac{k}{r}.$$

Working under the assumption that we don't know either k or r, this might or might not allow us to identify r. For example, if $k = 0$ we'll get an approximation $y/2^m$ to 0, which unfortunately tells us nothing. This, however, is an unusual case; for other values of k, we'll at least be able to learn something about r.

We can use an algorithm known as the *continued fraction algorithm* to turn our approximation $y/2^m$ into nearby fractions — including k/r if the approximation is good enough. We won't explain the continued fraction algorithm here. Instead, here's a statement of a known fact about this algorithm.

> **Fact**
> Given an integer $N \geq 2$ and a real number $\alpha \in (0, 1)$, there is at most one choice of integers $u, v \in \{0, \ldots, N-1\}$ with $v \neq 0$ and $\gcd(u, v) = 1$ satisfying $|\alpha - u/v| < \frac{1}{2N^2}$. Given α and N, the *continued fraction algorithm* finds u and v, or reports that they don't exist. This algorithm can be implemented as a Boolean circuit having size $O((\lg(N))^3)$.

If we have a very close approximation $y/2^m$ to k/r, and we run the continued fraction algorithm for N and $\alpha = y/2^m$, we'll get u and v, as they're described in the fact. An analysis of the fact allows us to conclude that

$$\frac{u}{v} = \frac{k}{r}.$$

Notice in particular that we don't necessarily learn k and r, we only learn k/r in lowest terms.

For example, and as we've already noticed, we're not going to learn anything from $k = 0$. But that's the only value of k where that happens. When k is nonzero, it might have common factors with r, but the number v we obtain from the continued fraction algorithm must at least divide r.

It's far from obvious, but it is true that if we have the ability to learn u and v for $u/v = k/r$ for $k \in \{0, \ldots, r-1\}$ chosen *uniformly at random*, then we're very likely to be able to recover r after just a few samples. In particular, if our guess for r is the *least common multiple* of all the values for the denominator v that we observe, we'll be right with high probability. Intuitively speaking, some values of k aren't good because they share common factors with r, and those common factors are hidden from us when we learn u and v. But *random* choices of k aren't likely to

hide factors of r for long, and the probability that we don't guess r correctly by taking the least common multiple of the denominators we observe drops exponentially in the number of samples.

It remains to address the issue of how we get our hands on an eigenvector $|\psi_k\rangle$ of M_a on which to run the phase estimation procedure. As it turns out, we don't actually need to create them!

What we will do instead is to run the phase estimation procedure on the state $|1\rangle$, by which we mean the n-bit binary encoding of the number 1, in place of an eigenvector $|\psi\rangle$ of M_a. So far, we've only talked about running the phase estimation procedure on a particular eigenvector, but nothing prevents us from running the procedure on an input state that isn't an eigenvector of M_a, and that's what we're doing here with the state $|1\rangle$. (This isn't an eigenvector of M_a unless $a = 1$, which isn't a choice we'll be interested in.)

The rationale for choosing the state $|1\rangle$ in place of an eigenvector of M_a is that the following equation is true.

$$|1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle$$

One way to verify this equation is to compare the inner products of the two sides with each standard basis state, using formulas mentioned previously in the lesson to help to evaluate the results for the right-hand side. As a consequence, we will obtain precisely the same measurement results as if we had chosen $k \in \{0, \ldots, r-1\}$ uniformly at random and used $|\psi_k\rangle$ as an eigenvector.

In greater detail, let's imagine that we run the phase estimation procedure with the state $|1\rangle$ in place of one of the eigenvectors $|\psi_k\rangle$. After the inverse quantum Fourier transform is performed, this leaves us with the state

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\psi_k\rangle |\gamma_k\rangle,$$

where

$$|\gamma_k\rangle = \frac{1}{2^m} \sum_{y=0}^{2^m-1} \sum_{x=0}^{2^m-1} e^{2\pi i x (k/r - y/2^m)} |y\rangle.$$

The vector $|\gamma_k\rangle$ represents the state of the top m qubits after the inverse of the quantum Fourier transform has been performed on them.

So, by virtue of the fact that $\{|\psi_0\rangle, \ldots, |\psi_{r-1}\rangle\}$ is an orthonormal set, we find that a measurement of the top m qubits yields an approximation $y/2^m$ to the value k/r where $k \in \{0, \ldots, r-1\}$ is chosen uniformly at random. As we've already discussed, this allows us to learn r with a high degree of confidence after several independent runs, which was our goal.

Total cost

The cost to implement each controlled-unitary M_a^k is $O(n^2)$. There are m controlled-unitary operations, and we have $m = O(n)$, so the total cost for the controlled-unitary operations is $O(n^3)$. In addition, we have m Hadamard gates (which contribute $O(n)$ to the cost), and the inverse quantum Fourier transform contributes $O(n^2)$ to the cost. Thus, the cost of the controlled-unitary operations dominates the cost of the entire procedure — which is therefore $O(n^3)$.

In addition to the quantum circuit itself, there are a few classical computations that need to be performed along the way. This includes computing the powers a^k in $\mathbb{Z}_N$ for $k = 2, 4, 8, \ldots, 2^{m-1}$, which are needed to create the controlled-unitary gates, as well as the continued fraction algorithm that converts approximations of θ into fractions. These computations can be performed by Boolean circuits with a total cost of $O(n^3)$.

As is typical, all of these bounds can be improved using asymptotically fast algorithms; these bounds assume we're using standard algorithms for basic arithmetic operations.

Factoring by order finding

The very last thing we need to discuss is how solving the order-finding problem helps us to factor. This part is completely classical — it has nothing specifically to do with quantum computing.

Here's the basic idea. We want to factorize the number N, and we can do this *recursively*. Specifically, we can focus on the task of *splitting* N, which means finding any two integers $b, c \geq 2$ for which $N = bc$. This isn't possible if N is a prime number, but we can efficiently test to see if N is prime using a primality testing algorithm first, and if N isn't prime

we'll try to split it. Once we split N, we can simply recurse on b and c until all of our factors are prime and we obtain the prime factorization of N.

Splitting even integers is easy: we just output 2 and $N/2$.

It's also easy to split perfect powers, meaning numbers of the form $N = s^j$ for integers $s, j \geq 2$, just by approximating the roots $N^{1/2}$, $N^{1/3}$, $N^{1/4}$, and so on, and checking nearby integers as suspects for s. We don't need to go further than $\log(N)$ steps into this sequence, because at that point the root drops below 2 and won't reveal additional candidates.

It's good that we can do both of these things because order finding won't help us to factor even numbers or for *prime* powers, where the number s happens to be prime. If N is odd and not a prime power, however, order finding allows us to split N.

Probabilistic algorithm to split an odd, composite integer N that is not a prime power

1. Randomly choose $a \in \{2, \ldots, N-1\}$.
2. Compute $d = \gcd(a, N)$.
3. If $d > 1$ then output $b = d$ and $c = N/d$ and stop.
   Otherwise continue to the next step knowing that $a \in \mathbb{Z}_N^*$.
4. Let r be the order of a modulo N. (Here's where we need order finding.)
5. If r is even:
   5.1 Compute $x = a^{r/2} - 1$ modulo N
   5.2 Compute $d = \gcd(x, N)$.
   5.3 If $d > 1$ then output $b = d$ and $c = N/d$ and stop.
6. If this point is reached, the algorithm has failed to find a factor of N.

A run of this algorithm may fail to find a factor of N. Specifically, this happens in two situations:

- The order of a modulo N is odd.
- The order of a modulo N is even and $\gcd\left(a^{r/2} - 1, N\right) = 1$.

Using basic number theory it can be proved that, for a random choice of a, with probability at least $1/2$ neither of these events happen. In fact, the probability that either event happens is at most $2^{-(m-1)}$ for m being the number of distinct prime factors of N, which is why the assumption that N is not a prime power is needed. (The assumption that N is odd is also required for this fact to be true.)

This means that each run has at least a 50% chance to split N. Therefore, if we run the algorithm t times, randomly choosing a each time, we'll succeed in splitting N with probability at least $1 - 2^{-t}$.

The basic idea behind the algorithm is as follows. If we have a choice of a for which the order r of a modulo N is even, then $r/2$ is an integer and we can consider the numbers

$$a^{r/2} - 1 \pmod{N} \quad \text{and} \quad a^{r/2} + 1 \pmod{N}.$$

Using the formula $Z^2 - 1 = (Z + 1)(Z - 1)$, we conclude that

$$\left(a^{r/2} - 1\right)\left(a^{r/2} + 1\right) = a^r - 1.$$

Now, we know that $a^r \pmod{N} = 1$ by the definition of the order — which is another way of saying that N evenly divides $a^r - 1$. That means that N evenly divides the product

$$\left(a^{r/2} - 1\right)\left(a^{r/2} + 1\right).$$

For this to be true, all of the prime factors of N must also be prime factors of $a^{r/2} - 1$ or $a^{r/2} + 1$ (or both) — and for a random selection of a it turns out to be unlikely that all of the prime factors of N will divide one of the terms and none will divide the other. Otherwise, so long as some of the prime factors of N divide the first term and some divide the second term, we'll be able to find a non-trivial factor of N by computing the GCD with the first term.

Previous page

Start the next lesson