# An Introduction to Quantum Circuits and Shor's Algorithm

**A literature survey**

Ben Hamlin

## 1 Introduction

Quantum computing systems are nothing more than a collection of Turing-equivalent programming systems. They differ from classical programming systems, however, in that they rely on the features of quantum mechanics to perform computations. It's an open question whether this gives them a fundamental advantage in complexity over classical methods (a more than polynomial speedup over "physically realizable" classical machines), but there's enough evidence to make studying them worthwhile.

In order to take take advantage of — or, put another way, to model — quantum mechanics, a system must have some way to represent all of the following [6]:

1. *probabilistic states:* A closed quantum system is characterized by the probability that it will be in each of a set of observable states when it is measured;
2. *change in state:* The probabilities that characterize a state can be manipulated to change over time, making computation possible. This is frequently called *time evolution* in the literature;
3. *measurement:* When a quantum system is measured, its state resolves into a particular observable outcome. This effect is distinct from time evolution;
4. *interference:* The probabilities associated with each component of a closed quantum system can affect or "interfere" with the probabilities associated with other components. This results in the two components having non-independent distributions.

A variety of systems with these properties have been proposed. Bernstein and Vazirani have suggested a quantum analog of the classical Turing machine [2], and quantum cellular automata, proposed by Richard Feynmann, have been formalized more recently by Delgado and others [3]. Quantum circuits remain the most popular system for representing quantum algorithms, however, and those are what this paper will cover.

The goal of this paper is to introduce the reader to quantum circuits from the perspective of computational theory. In the process, I'll survey some of the literature on this topic and walk through some examples of how quantum circuits can be applied. In sections 2 and 3, I'll introduce the basic abstractions in the quantum circuit model: qubits and qubit ensembles. I'll also introduce the concept of "mixed" or

"entangled" states, which is one of the principle ways in which quantum computation differs from classical computation. Next, in section 4 I'll talk about the operations that can be applied to qubits and ensembles, and with these preliminaries out of the way, I'll describe the system of quantum circuits and how they differ from classical boolean circuits. In sections 5 and 6, I'll describe some algorithms that can be expressed using quantum circuits, including — to the limited degree to which I understand it — Shor's famous integer factorization algorithm.

I began writing this paper in the hopes of answering the question of whether quantum algorithms allow a fundamental speedup over classical algorithms, in contradiction of the strong form of the Church-Turing thesis, and if so, what this speedup entails. In section 7, I'll briefly sum up my answer to this question, to the extent that I've succeded in finding one.

# 2 Qubits

Qubits are the basic unit of quantum information, whether you're working with QTMs, quantum circuits, or even the bloch sphere representation used in quantum physics. That is to say, they're not specific to quantum circuits. Nonetheless, qubits and their canonical representation as column or "ket" vectors are central to understanding quantum circuits.

A qubit represents a single bit of classical information. When measured, it will be observed to be a zero or a one. However between measurements, a qubit exists in a "superposition" of classical states. That is, when measured, it will be a zero with probability $P(0)$ and a one with probability $P(1)$. Obviously, $P(0) + P(1) = 1$ in a single qubit system.

Rather than express the state of our system using $P(0)$ and $P(1)$ themselves, it's useful to think of the one-state and the zero-state as having signed *amplitudes* whose squares represent their respective probabilities. I.e.,

$$P(0) = a_0^2$$

$$P(1) = a_1^2$$

This means that

$$\begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

is a unit vector. As it turns out, letting $a_0, a_1 \in \mathbb{C}$ allows us to accurately model the kinds of time evolution we can perform on quantum systems. For complex-valued amplitudes,

$$P(0) = |a_0|^2 = |x_0 + iy_0|^2 = x_0^2 + y_0^2$$

and similarly for $P(1)$.

The standard notation for a qubit in the classical state zero is

$$|0\rangle$$

The implication here is that, even though the qubit happens to be in a classical state, it still has amplitudes:

$$|0\rangle = 1|0\rangle + 0|1\rangle$$

More generally, a qubit is always in the state

$$a_0|0\rangle + a_1|1\rangle$$

where $a_0$ and $a_1$ are amplitudes, as before. This is called Dirac notation, and the symbol $|n\rangle$ is called a *ket*[1].

[1] There's also a "bra", hence together they're "bra-ket" notation. A bra, written $\langle n|$, is just a row vector, the complex transpose of a ket.

This notation avoids some messy typesetting, but at first glance, it hides an important fact: A ket is nothing more than a column vector. That is,

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

. More generally, you'll see when we talk about ensembles of qubits that $|b\rangle$, where $b \in \{0,1\}^n$ is the binary representation of the number $B$, is the n-dimensional column vector in which the $B$th element is a one, and the rest are zeroes. Thus

$$a_0|0\rangle + a_1|1\rangle = a_0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + a_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} a_0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ a_1 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

which is identical to our unit vector representation above.

The ket vectors $|0\rangle$ and $|1\rangle$ are called *bases* — in linear algebra terms, orthonormal basis vectors. It's worth noting that they're not unique. These particular bases — called the computational bases in the literature — are useful because their amplitudes correspond to the probabilities of observing the classical states when measured. Nonetheless, we could as easily express the state of our system using any set of orthonormal bases. For example,

$$|+\rangle = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$|-\rangle = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

are called the Bell bases.

We can derive a Bell representation of a qubit using the following identities:

$$|0\rangle = \frac{|+\rangle + |-\rangle}{\sqrt{2}}$$

$$|1\rangle = \frac{|+\rangle - |-\rangle}{\sqrt{2}}$$

As we'll see in section 4, viewing a quantum system in terms of the Bell bases can make the effects of some quantum operations seem less mystifying[2].

# 3 Ensembles of Qubits

One of the principle notions in quantum computation — called interference — is that the amplitudes of one qubit can affect the amplitudes of another qubit in the same closed system before they are measured. This results in qubits whose probability distributions are intertwined. Put in the terminology of quantum mechanics, they are *entangled,* or in a *mixed* as opposed to a *pure* state. The upshot of this is that being able to represent qubits in isolation isn't enough. We need to be able to represent groups or *ensembles* of qubits.

The computational bases for qubit ensembles are analogous to the ones used for individual qubits. That is,

$$|...000\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ . \\ . \\ . \end{bmatrix}, |...001\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ . \\ . \\ . \end{bmatrix}, ...$$

More precisely, the states of an ensemble's component qubits are combined using the

[2]In fact from a physical perspective, we could even *measure* the qubit with respect to these bases. That is, we could measure the qubit $|\phi\rangle = a_+|+\rangle + a_-|-\rangle$ in such a way that we would see a $|+\rangle$ with probability $|a_+|^2$ and a $|-\rangle$ with probability $|a_-|^2$. This would tell us nothing about whether the qubit was a $|0\rangle$ or a $|1\rangle$ [5]. For the purposes of quantum computation, all measurement happens with respect to the computational bases.

tensor product ($\otimes$), defined as

$$\begin{bmatrix} a_0 \\ a_1 \\ . \\ . \\ . \end{bmatrix} \otimes \begin{bmatrix} b_0 \\ b_1 \\ . \\ . \\ . \end{bmatrix} = \begin{bmatrix} a_0 \times b_0 \\ a_0 \times b_1 \\ a_1 \times b_0 \\ . \\ . \\ . \end{bmatrix}$$

Note that tensor distributes over addition, so that

$$a \otimes (b + c) = a \otimes b + a \otimes c$$

If we have two qubits $|\phi_a\rangle = a_0|0\rangle + a_1|1\rangle$ and $|\phi_b\rangle = b_0|0\rangle + b_1|1\rangle$, the ensemble $|\phi_0\rangle \otimes |\phi_1\rangle$, often expressed as $|\phi_0\rangle|\phi_1\rangle$, or even $|\phi_0\phi_1\rangle$, is

$$a_0b_0|00\rangle + a_0b_1|01\rangle + a_1b_0|10\rangle + a_1b_1|11\rangle$$

$$= \begin{bmatrix} a_0b_0 \\ a_0b_1 \\ a_1b_0 \\ a_1b_1 \end{bmatrix}$$

Thus an ensemble with $n$ qubits will be represented using $2^n$ basis vectors and amplitudes. Again, the square of the amplitude of each state corresponds to the probability of observing that state when the system is measured. That is $P(00) = |a_0b_0|^2$, etc.

It's also possible to measure just one qubit of an ensemble [5]. Given the ensemble

$$a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle$$

measuring just the first qubit gives us

$$P(0) = |a_{00} + a_{01}|^2$$

$$P(1) = |a_{10}|^2 + |a_{11}|^2$$

and assuming the outcome of our measurement was $X$, the new state of the ensemble is

$$\frac{a_{X0}|X0\rangle + a_{X1}|X1\rangle}{\sqrt{P(X)}}$$

A simple example of an entangled state is $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$, sometimes called the Bell state. Say we measure the first qubit in an ensemble in the Bell state and, with probability $\frac{1}{2}$, we observe a $|0\rangle$. The new state of the ensemble is

$$\frac{\frac{1}{\sqrt{2}}|00\rangle}{\frac{1}{\sqrt{2}}} = |00\rangle$$

so the second qubit must be a $|0\rangle$. On the other hand, if we observe the first qubit to be a $|1\rangle$, the second qubit must also be a $|1\rangle$. Measuring the first qubit changes the probabilities associated with the second. There is no way to represent this with single-qubit ket vectors.

It's important to note that increasing the number of qubits in the ensemble by one multiplies the number of amplitudes by two. This makes quantum circuits — and quantum mechanics in general — difficult to simulate with a classical computer. It also hints at the power of quantum computing: An ensemble of $n$ qubits seems to contain $2^n$ complex-valued amplitudes — significantly more than $n$ bits of information. This information isn't immediately useful, since measuring the qubit resolves the ensemble into a classical state, and $n$ bits of information are lost. Quantum algorithm design is an exercise in trying to use interference to perform calculations using this extra information before the system is measured.

# 4 Operations and Gates

Ensembles of qubits allow us to represent the probabilistic state of a quantum computational system at a single moment in time. In addition, we need to be able to represent the way the system changes

over time — its so-called "time evolution." In fact, in order to perform computations, we need to be able to execute these changes. Since we're representing qubits and ensembles with $2^n$-dimensional unit vectors, it's natural to represent operations as dimension-preserving linear transformations, that is, by left-multiplying our state vectors by $2^n \times 2^n$ matrices.

The laws of probability and physics place some limits on the kinds of operations we can perform. The laws of thermodynamics demand that we only perform reversible transformations. That is, we can only multiply our state vectors by invertible matrices. Fortunately, any classical computation can be made reversible with only a constant increase in time complexity [8]. In addition, it would make no sense if the probabilities associated with the observable outcomes of our system didn't add up to one. Thus we are limited to operations that can be described by unitary matrices. Since state vectors have elements in $\mathbb{C}$, our operations must be unitary in the complex-valued sense that $P^{-1} = P^\dagger$, where $P^\dagger$ is the adjoint[3] of $P$. In quantum circuits, operations are often refered to as "gates". From here on, I'll use the terms interchangably.

Any $2^n \times 2^n$ unitary matrix over $\mathbb{C}$ is a valid transformation on an $n$-bit ensemble. This provides us with an infinite number of possible operations — an undesirable state if we ever hope to implement a quantum computer. A wide variety of sets of "universal" operations have been proposed to deal with this problem — "universal" in the sense that in combination, they can approximate any quantum operation to an arbitrary degree of accuracy. The most influential approach seems to be the one taken by Barenco, et al. [1], who show that the set of one qubit "rotations", which they simulate using five one-qubit operations, along with a single two-qubit operation, form a universal set.

The choice of operations is important because although any universal set of quantum operations can be combined to approximate any other quantum operation, they may not be able to do so efficiently. In fact, it is conjectured that "almost all" operations require a number of applications of the universal gates that is exponential in the number of input qubits. The choice of a universal set thus determines which operations can be efficiently approximated. In practice, it seems to be possible to implement useful circuits using a relatively small — efficiently approximable — set of operations. This is discussed more fully in [1] and [5].

Only one one-qubit gate has an obvious analog in classical computation: the "not" or $X$ gate. This has the effect of swapping the amplitudes of the computational bases. The corresponding matrix representation is

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The effect this has on the classical states is

$$X|0\rangle = |1\rangle$$

$$X|1\rangle = |0\rangle$$

Note that the interpretation of $X$ as "not" only makes sense as long as we intend to measure in the computational bases, since

$$X|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle$$

$$X|-\rangle = X\frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{|1\rangle - |0\rangle}{\sqrt{2}} = -1|-\rangle$$

[3] "Adjoint" is just a quantum mechanical term for conjugate transpose.

So $X|+\rangle \neq |-\rangle$, and vice versa.

The other one-qubit gates used in the circuits described below are the Hadamard or $H$ gate and the $T$ gate. The $H$ gate takes a qubit in one of the computational base states and places it reversibly into a "uniform" state — a state that has equal probability of resolving to a $|0\rangle$ or a $|1\rangle$ on measurement. The matrix for $H$ is

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

and its effect is

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

It is its own inverse, so its effects can be reversed simply by applying it a second time. It's interesting to note that

$$H|0\rangle = |+\rangle$$

$$H|1\rangle = |-\rangle$$

$$H|+\rangle = |0\rangle$$

$$H|-\rangle = |1\rangle$$

So another way to characterize the Hadamard gate is that it converts between the computational and Bell bases.

The effects of the $T$ gate are harder to describe in an intuitive way. If the complex-valued amplitudes of a qubit are taken to represent angles of the form

$$a = cos(\theta) + isin(\theta)$$

then the $T$ gate rotates $\theta$ in the $|1\rangle$ amplitude of the qubit 45° counterclockwise[4]. Its only application in the algorithms discussed in this paper is to perform the quantum Fourier transform as part of Shor's algorithm.

In order to model interference — the property of quantum mechanical systems that the amplitudes of one qubit can affect the amplitudes of another before measurement occurs — we need multi-qubit gates. As discussed by [1] and others, it suffices to talk about two qubit gates, since they can be used to construct gates over an arbitrary number of qubits.

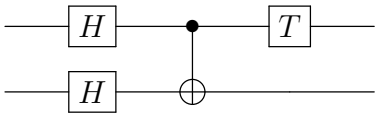
In particular, we can construct arbitrary n-qubit gates using single-qubit gates and the two-qubit gate "controlled-not," or $CNOT$. $CNOT$ has a control qubit and a target qubit, and its effect is to perform $X$ on the target qubit if the control qubit is a $|1\rangle$, and do nothing otherwise. Assuming the first qubit in a two-qubit ensemble is the control and the second is the target, its matrix representation is

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Now that we have introduced all of the components of a quantum circuit, we can start talking about the circuits themselves. Quantum circuits, also called quantum gate arrays, were introduced by D. Deutsch in [4]. They are analogous to classical boolean circuits, and analogously, they can be used to judge the time complexity of an algorithm. The time complexity of a quantum circuit is the number of gates it contains as a function of its input size. In addition, we require that a family of circuits describing the algorithm for arbitrary input sizes be constructible by a Turing machine. This allows us to rule out circuit families that solve uncomputable problems.

[4]This relates to a representation of qubits called the Bloch sphere, which can be useful, but which I won't talk about further here.

Graphically, a quantum circuit looks much like a classical boolean circuit. In the circuit



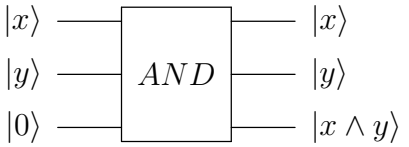

the gates applied are $H$, $CNOT$ with the upper qubit as control and the lower as target, and $T$, in that order. Time proceeds left to right along the x-axis. Sometimes, like the Hadamards in the above diagram, gates will be applied to multiple wires at the same "time". This is strictly for convenience of representation. They both count toward the complexity of the overall circuit. Also, many operations require extra scratch, or "ancilla" qubits that are not part of the input and that start in a known state. These to not count toward the input size. A comprehensive coverage of quantum circuit complexity is provided in [4].

Quantum circuits are lacking some features of classical circuits. In particular fanout is impossible. The number of wires is constant throughout the circuit. Even using scratch qubits, it's impossible to copy the value of a qubit. Consider a circuit of the form

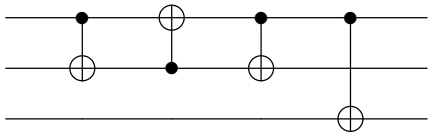

The value of $y$ is lost, so this gate is not invertible. On the other hand, this circuit is realizable


and as long as we make sure that $y$ is $|0\rangle$ to start out with, this has the same effect.

Similarly, there is no way to implement an exact analog of classical *and*. On the other hand, we can do this

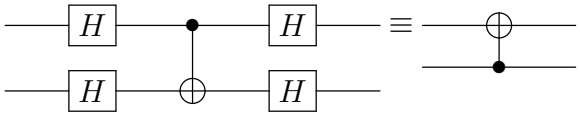



This is called a Toffoli gate, and it can be implemented using $CNOT$s as follows:



An *or* gate can be constructed similarly, and along with $X$ (*not*) and $CNOT$ (*xor*), these can be used to construct any classical circuit. On the other hand, every *and* in a classical circuit requires a scratch qubit in a quantum circuit. This could result in an unbounded increase in the number of qubits required for a computation, so quantum algorithms have to be carefully designed to reset their scratch qubits to $|0\rangle$ between calculations so they can be reused. This is frequently accomplished by performing the computation and then performing its inverse, as in [7].

Finally, it's worthwhile to note a possibly counterintuitive property of $CNOT$:




This is unexpected, because our description of $CNOT$ entails that it leaves the control qubit unchanged. This characterization applies to ensembles in classical states, however ($|00\rangle$, $|01\rangle$, ...). Remember that the effect of Hadamard is to make a qubit undetermined with respect to the computational bases but determined with respect to the Bell bases. The effect of $CNOT$ on an ensemble represented with respect to the Bell bases is

$$CNOT|{+}{+}\rangle = |{+}{+}\rangle$$

$$CNOT|+-\rangle = |--\rangle$$
$$CNOT|-+\rangle = |-+\rangle$$
$$CNOT|--\rangle = |+-\rangle$$

In other words, its effect is to change the first qubit when the second qubit is a $|-\rangle$ and do nothing otherwise, the exact opposite of what it does when the qubit is determined with respect to the computational bases.

# 5 An Example: Simon's Problem

Simon's problem is a relatively simple example of the power of quantum computing. In fact, it was published with the express interest of demonstrating the potential of quantum algorithms [8]. The problem it solves is of less practical interest than integer factorization or discrete logarithm. Nonetheless, it is thought not to be in $BPP$, whereas the quantum-computational solution runs in expected polynomial time.

The problem is the following: Let

$$f : \{0,1\}^n \to \{0,1\}^m$$

such that for some constant $c$

$$\forall x \neq y, f(x) = f(y) \text{ iff } x = c \oplus y$$

We wish to find $c$. Note that $f$ is an injection in the case that $c = 0$. Otherwise, each output of $f$ corresponds to exactly two inputs, $x$ and $x \oplus c$.

The corresponding decision problem — whether the $c$ value for $f$ is within a specified range — is clearly in $NP$. The $c$ value acts as the certificate, and because we are guaranteed that $c$ exists, the verifier can simply

1. Select some $x$,
2. Calculate $a = f(x)$ and $b = f(x \otimes c)$,
3. Return true iff $a = b$.

which runs in polynomial time assuming $f$ runs in polynomial time.

On the other hand, there is no known decider for the problem in $P$. The naive algorithm would be

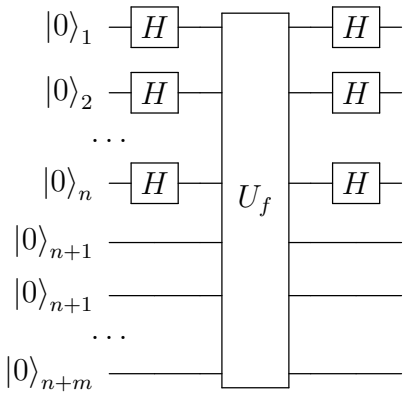
```
for x from 0 to 2^n
    for y from 0 to 2^n
        if x ≠ y
            a ← f(x)
            b ← f(y)
            if a = b return x ⊕ y
return 0
```

Assuming $c$ is non-zero, the likelyhood that $a = b$ at each comparison is

$$\frac{1}{2^{2n} - 2^n - ((x+1)(y+1))}$$

and it can can be shown that no classical probabilistic Turing machine that runs $f$ fewer than $2^{\frac{1}{4}}$ times can correctly guess even whether $c$ is zero or non-zero ith greater than $\frac{1}{2} + 2^{-\frac{n}{2}}$ probability [8].

The quantum circuit for Simon's problem looks as follows, where $U_f$ finds $a_0, a_1, ..., a_m = f(x)$ on the first $n$ qubits of the ensemble and sets each $b_i$ of the last $m$ qubits to $b_i = b_i \oplus a_i$:




Only the upper $n$ qubits are measured. I provide examples below, but the gist is

that after the second Hadamard, assuming $c \neq 0$, the upper $n$ qubits will contain, with equal likelyhood, exactly one of the two inputs corresponding to each output[5]. On the other hand if $c = 0$, the upper $n$ bits will be uniformly distributed over $2^n$. Sampling $k$ times and solving the resulting set of linear equations will give us $c$. A detailed statistical analysis of this, including a proof that the number of samples required is polynomial, can be found in [8].

In broad strokes, the technique here is a common approach in quantum algorithms:

1. Put a set of qubits into a uniform superposition.
2. Run some function on them and store the output, generating a uniform superposition of pairs of the form $(x, f(x))$ for all possible input values, $x$.

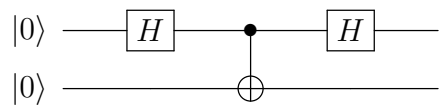
This is sometimes called "quantum parallelism". Note that this does let us obtain all possible $(x, f(x))$ pairs using a single run of $f$. If we measure our ensemble at this point, we'll see a single such pair, which we could have obtained by simply running $f$ classically. The tricky part is using this uniform superposition to give us some information about the function.

To take a minimal one-bit example, say

$$f(0) = 1$$

$$f(1) = 0$$

In this case, $c = 0$. The circuit implementing Simon's algorithm for this problem description is simply




[5]E.g., if $f(00) = f(01) = 00$ and $f(10) = f(11) = 11$, the upper $n$ bits will contain a superposition of $|00\rangle$ and $|10\rangle$

After the first Hadamard gate, we have

$$\frac{|00\rangle + |10\rangle}{\sqrt{2}}$$

The $CNOT$ takes this to

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

And the final Hadamard gives us

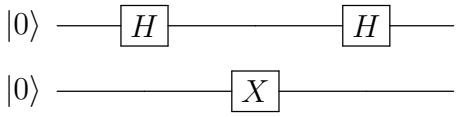
$$\frac{|00\rangle + |10\rangle + |01\rangle - |11\rangle}{2}$$

Since each input to $f$ maps to a different output, nothing cancels, and measuring the first qubit gives us $|0\rangle$ or $|1\rangle$ with equal probability.

On the other hand, say $c = 1$, for example, our function might be

$$f(0) = f(1) = 1$$

The circuit to implement Simon's algorithm would be




After the final Hadamard, our ensemble is in the state

$$\frac{|01\rangle + |11\rangle + |01\rangle - |11\rangle}{2} = |01\rangle$$

Because an output value was repeated, one of our input values canceled out, and the only possible value for the first qubit is $|0\rangle$. This can be used to find $c$.

In this simple case, the outcome may seem trivial. After all, $H$ is its own identity, so of course $HH|0\rangle = |0\rangle$. Nonetheless, this generalizes to functions with inputs of arbitrary size. Moreover, it illustrates a feature that [8] uses to characterize why quantum machines seem to be more powerful — in

a complexity sense — than classical probabilistic machines:

The computation of a classical probabilistic machine can be characterized by a tree in which each node is a configuration. The probability of it being in configuration $a$ at step $n+1$ is determined by its configuration at step $n$ and the input it sees. Configurations can be duplicated at the $n+1$st level of the tree if they are reachable from more than one configuration at step $n$. For each leaf, the probability of ending at that leaf is the product of the probabilities along the path to that leaf, and if a configuration is duplicated in more than one leaf, the probability of ending in that configuration is the sum of those leaves' probabilities. All of the probabilities are, of course, positive.

A quantum computation can be described by a similar tree of configurations. The difference is that, until measurement is taken, each node is assigned amplitudes, not probabilities, and these amplitudes can be negative. If a configuration is duplicated, the probability of arriving at that leaf is the *square of the sums* of their amplitudes. This means that if one of leaf representing configuration $c$ has amplitude $\frac{1}{\sqrt{2}}$ and another has amplitude $^{-}\frac{1}{\sqrt{2}}$, they cancel out, and the configuration is unreachable. In other words, the amplitudes interfere with each other before they resolve to probabilities. It's interesting to note that, if one were to measure the system after each step, collapsing the amplitudes to probabilities, the result would have no more power than a classical probabilistic machine.

# 6 Shor's Algorithm

Shor's algorithm is the most famous quantum algorithm. It factors integers, a problem that is not only believed not to be in $P$, but whose hardness is the basis for RSA. Nonetheless, integer factorization is not $NP$ complete, and from the perspective of computational complexity, its solution is no more surprising than the solution to Simon's problem. I'll cover Shor's algorithm in significantly less depth than I did Simon's problem, without circuit diagrams or examples. This is in part because the circuits involved in Shor's algorithm are quite complex, but also because the final step in Shor's algorithm involves an element of Fourier analysis that I have yet to fully understand. Nonetheless, the general thrust of Shor's algorithm isn't hard to grasp, and is in fact quite similar to the algorithm for Simon's problem.

In [7], Shor presents a solution not just to integer factorization, but also to the discrete logarithm problem, which is the hard problem behind the Diffie-Hellman cryptographic key exchange algorithm. In fact, the problem Shor solves is neither of these, directly, but order-finding: determining the smallest $r$ such that $x^r \equiv 1 (\mathrm{mod}\ n)$. Such an $r$ for a uniformly chosen $x$ can be used to find a prime factor of $n$ with probability greater than $\frac{1}{3}$, and since the number of factors of $n$ is polynomial in $log(n)$, a polynomial time algorithm for order-finding equates to a polynomial time algorithm for integer factorization: All we need to do is choose values for $x$ and find their order until the prime factors derived can be multiplied together to get $n$. A complete coverage of the number-theoretic foundations of this can be found in [7], but an order-finding approach is not unique to Shor's algorithm. It is the basis for classical factoring algorithms as well.

Shor's algorithm can be broken up into two basic parts, one of which could easily be performed by a classical computer. The other two require quantum computation.

The first of these is merely a quantum version of fast exponentiation. The algorithm is fully described in [7], but it is identical in most respects to the classical algorithm for performing the same task. We calculate $x^y (\text{mod } n)$, where $y$ is treated as input, and $x$ is hard-wired into the circuit[6].

Once we have a circuit that calculates $f(y) = x^y (\text{mod } n)$ on $log(n)$ qubits and sets the next $log(n)$ qubits to their original value *xor* the result, we do the following:

1. Prepare an ensemble of $2log(n)$ qubits plus scratch qubits to $|0\rangle$
2. Use Hadamard gates to bring the first $log(n)$ qubits into the uniform state
3. Apply $f$ with the first $log(n)$ qubits as control and the next $log(n)$ qubits as target

Our ensemble is now in the state

$$\sum_{i=0}^{n} \frac{|i\rangle|x^i (\text{mod } n)\rangle}{n^{\frac{1}{2}}}$$

This very similar to the approach taken in the algorithm for Simon's problem. Every pair of the form $(y, f(y))$ is now stored in our ensemble with equal amplitude. If we attempt to measure it at this stage, however, the amplitudes collapse to give us just one of these pairs — no more than we could have gotten with classical fast exponentiation. We need some way to tease out only those values for which $f(y) = 1$.

The solution to this — and the part of the algorithm I have yet to understand — involves performing quantum Fourier transform (QFT). The classical Fourier transform takes as input a list of numbers representing outputs to a function $f$ and produces a list of coefficients to *sine* and *cosine* functions whose sum approximates $f$. This can then be used to find the periodicity of $f$. QFT does exactly the same thing, but the list taken as input and used as output is the list of amplitudes to a qubit ensemble. That is, each amplitude of the resulting ensemble is a coefficient to a *sine* or *cosine* function, which together approximate the function described by the original coefficients. The final part of the algorithm, which can be performed by a classical computer, is to perform statistical analysis on a polynomial number of runs of the quantum algorithm. The results of this are used to find the order of $x$ mod $n$ with probability greater than $\frac{2}{3}$. More detail on this process can be found in [7].

# 7 Conclusion

I'm a little disappointed that I failed to fully understand fully Shor's algorithm. I feel like I learned a lot in the attempt, though. In general, I think the answer to my original question, "To what extent does quantum computation allow a more-than-polynomial speedup over classical algorithms, and what does that speedup entail," can be summed up as follows:

- There is evidence, albeit inconclusive, that quantum algorithms provide a fundamental algorithmic speedup in some cases.
- To the extent that such a speedup exists, it can be explained by interference phenomena and so-called "quantum parallelism". In other words, it stems from the fact that at a quantum mechanical level, it seems to be possible to execute a function on an exponential[7] number of inputs at once and then cause the outputs of those execu-

[6] Thus for each new value of $x$ whose order we wish to find, we must create a new circuit.

[7] That is, exponential in the number of qubits.

tions to affect the amplitudes of qubits in a limited way.

# References


[1] A. Barenco, C. Bennett, R. Cleve, D. DiVincenzo, N. Margolus, P. Shor, T. Sleater, J. Smolin, and H. Weinfurter. "Elementary Gates for Quantum Computing." In *Physical Review A.* March, 1995.

[2] E. Bernstein and U. Vazirani. "Quantum complexity theory." In *Siam Journal on Computing* 26(5). 1997.

[3] C. Delgado and D. Cheung. "Local unitary quantum cellular automata." In *Physical Review A* 76(3A). 2007.

[4] D. Deutsch. "Quantum theory, the Church-Turing principle and the universal quantum computer" In *Proceedings of the Royal Society of London A* 400, pages 97–117. 1985.

[5] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge, UK, 2000.

[6] A. Pittenger. *An Introduction to Quantum Computing Algrithms.* Birkhauser, Boston, MA, 2000.

[7] P. Shor. "Polynomial time algorithms for prime factorization and discrete logarithm on a quantum computer." In *SIAM Journal of Computing 26.* 1997.

[8] D. Simon. "On the power of quantum computation." In *SIAM Journal on Computing* 26(5). 1997.