from being used to transmit information faster than light. Once Bob has learned the measurement outcome, Bob can 'fix up' his state, recovering $|\psi\rangle$, by applying the appropriate quantum gate. For example, in the case where the measurement yields 00, Bob doesn't need to do anything. If the measurement is 01 then Bob can fix up his state by applying the X gate. If the measurement is 10 then Bob can fix up his state by applying the Z gate. If the measurement is 11 then Bob can fix up his state by applying first an X and then a Z gate. Summing up, Bob needs to apply the transformation $Z^{M_1} X^{M_2}$ (note how time goes from left to right in circuit diagrams, but in matrix products terms on the *right* happen *first*) to his qubit, and he will recover the state $|\psi\rangle$.

There are many interesting features of teleportation, some of which we shall return to later in the book. For now we content ourselves with commenting on a couple of aspects. First, doesn't teleportation allow one to transmit quantum states faster than light? This would be rather peculiar, because the theory of relativity implies that faster than light information transfer could be used to send information backwards in time. Fortunately, quantum teleportation does not enable faster than light communication, because to complete the teleportation Alice must transmit her measurement result to Bob over a classical communications channel. We will show in Section 2.4.3 that without this classical communication, teleportation does not convey *any* information at all. The classical channel is limited by the speed of light, so it follows that quantum teleportation cannot be accomplished faster than the speed of light, resolving the apparent paradox.

A second puzzle about teleportation is that it appears to create a copy of the quantum state being teleported, in apparent violation of the no-cloning theorem discussed in Section 1.3.5. This violation is only illusory since after the teleportation process only the target qubit is left in the state $|\psi\rangle$, and the original data qubit ends up in one of the computational basis states $|0\rangle$ or $|1\rangle$, depending upon the measurement result on the first qubit.

What can we learn from quantum teleportation? Quite a lot! It's much more than just a neat trick one can do with quantum states. Quantum teleportation emphasizes the interchangeability of *different* resources in quantum mechanics, showing that one shared EPR pair together with two classical bits of communication is a resource at least the equal of one qubit of communication. Quantum computation and quantum information has revealed a plethora of methods for interchanging resources, many built upon quantum teleportation. In particular, in Chapter 10 we explain how teleportation can be used to build quantum gates which are resistant to the effects of noise, and in Chapter 12 we show that teleportation is intimately connected with the properties of quantum error-correcting codes. Despite these connections with other subjects, it is fair to say that we are only beginning to understand *why* it is that quantum teleportation is possible in quantum mechanics; in later chapters we endeavor to explain some of the insights that make such an understanding possible.

1.4 Quantum algorithms

What class of computations can be performed using quantum circuits? How does that class compare with the computations which can be performed using classical logical circuits? Can we find a task which a quantum computer may perform better than a classical computer? In this section we investigate these questions, explaining how to perform classical computations on quantum computers, giving some examples of problems for

which quantum computers offer an advantage over classical computers, and summarizing the known quantum algorithms.

1.4.1 Classical computations on a quantum computer

Can we simulate a classical logic circuit using a quantum circuit? Not surprisingly, the answer to this question turns out to be yes. It would be very surprising if this were not the case, as physicists believe that all aspects of the world around us, including classical logic circuits, can ultimately be explained using quantum mechanics. As pointed out earlier, the reason quantum circuits cannot be used to directly simulate classical circuits is because unitary quantum logic gates are inherently *reversible*, whereas many classical logic gates such as the NAND gate are inherently irreversible.

Any classical circuit can be replaced by an equivalent circuit containing only *reversible* elements, by making use of a reversible gate known as the *Toffoli gate*. The Toffoli gate has three input bits and three output bits, as illustrated in Figure 1.14. Two of the bits are *control bits* that are unaffected by the action of the Toffoli gate. The third bit is a *target bit* that is flipped if both control bits are set to 1, and otherwise is left alone. Note that applying the Toffoli gate twice to a set of bits has the effect $(a, b, c) \rightarrow (a, b, c \oplus ab) \rightarrow (a, b, c)$, and thus the Toffoli gate is a reversible gate, since it has an inverse – itself.

Inputs			Outputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

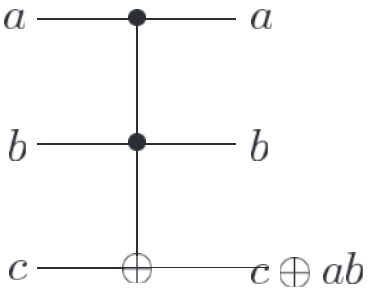

Figure 1.14. Truth table for the Toffoli gate, and its circuit representation.

The Toffoli gate can be used to simulate NAND gates, as shown in Figure 1.15, and can also be used to do FANOUT, as shown in Figure 1.16. With these two operations it becomes possible to simulate all other elements in a classical circuit, and thus an arbitrary classical circuit can be simulated by an equivalent reversible circuit.

The Toffoli gate has been described as a classical gate, but it can also be implemented as a quantum logic gate. By definition, the quantum logic implementation of the Toffoli gate simply permutes computational basis states in the same way as the classical Toffoli gate. For example, the quantum Toffoli gate acting on the state $|110\rangle$ flips the third qubit because the first two are set, resulting in the state $|111\rangle$. It is tedious but not difficult to write this transformation out as an 8 by 8 matrix, U, and verify explicitly that U is a unitary matrix, and thus the Toffoli gate is a legitimate quantum gate. The quantum Toffoli gate can be used to simulate irreversible classical logic gates, just as the classical

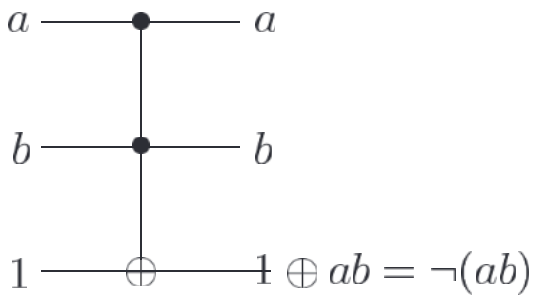

Figure 1.15. Classical circuit implementing a NAND gate using a Toffoli gate. The top two bits represent the input to the NAND, while the third bit is prepared in the standard state 1, sometimes known as an *ancilla* state. The output from the NAND is on the third bit.

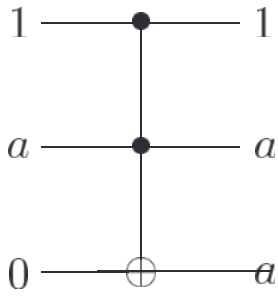

Figure 1.16. FANOUT with the Toffoli gate, with the second bit being the input to the FANOUT (and the other two bits standard ancilla states), and the output from FANOUT appearing on the second and third bits.

Toffoli gate was, and ensures that quantum computers are capable of performing any computation which a classical (deterministic) computer may do.

What if the classical computer is non-deterministic, that is, has the ability to generate random bits to be used in the computation? Not surprisingly, it is easy for a quantum computer to simulate this. To perform such a simulation it turns out to be sufficient to produce random fair coin tosses, which can be done by preparing a qubit in the state $|0\rangle$, sending it through a Hadamard gate to produce $(|0\rangle + |1\rangle)/\sqrt{2}$, and then measuring the state. The result will be $|0\rangle$ or $|1\rangle$ with 50/50 probability. This provides a quantum computer with the ability to efficiently simulate a non-deterministic classical computer.

Of course, if the ability to simulate classical computers were the only feature of quantum computers there would be little point in going to all the trouble of exploiting quantum effects! The advantage of quantum computing is that much more powerful functions may be computed using qubits and quantum gates. In the next few sections we explain how to do this, culminating in the Deutsch–Jozsa algorithm, our first example of a quantum algorithm able to solve a problem faster than any classical algorithm.

1.4.2 Quantum parallelism

Quantum parallelism is a fundamental feature of many quantum algorithms. Heuristically, and at the risk of over-simplifying, quantum parallelism allows quantum computers to evaluate a function $f(x)$ for many *different* values of x simultaneously. In this section we explain how quantum parallelism works, and some of its limitations.

Suppose $f(x) : \{0,1\} \rightarrow \{0,1\}$ is a function with a one-bit domain and range. A

convenient way of computing this function on a quantum computer is to consider a two qubit quantum computer which starts in the state $|x, y\rangle$. With an appropriate sequence of logic gates it is possible to transform this state into $|x, y \oplus f(x)\rangle$, where $\oplus$ indicates addition modulo 2; the first register is called the 'data' register, and the second register the 'target' register. We give the transformation defined by the map $|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$ a name, U_f, and note that it is easily shown to be unitary. If $y = 0$, then the final state of the second qubit is just the value $f(x)$. (In Section 3.2.5 we show that given a classical circuit for computing f there is a quantum circuit of comparable efficiency which computes the transformation U_f on a quantum computer. For our purposes it can be considered to be a black box.)

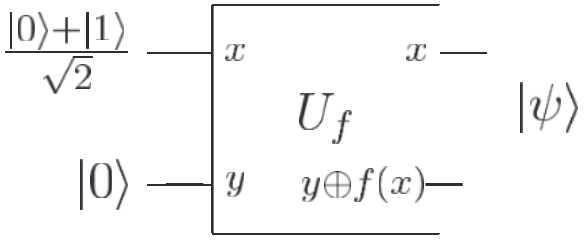

Figure 1.17. Quantum circuit for evaluating $f(0)$ and $f(1)$ simultaneously. U_f is the quantum circuit which takes inputs like $|x, y\rangle$ to $|x, y \oplus f(x)\rangle$.

Consider the circuit shown in Figure 1.17, which applies U_f to an input not in the computational basis. Instead, the data register is prepared in the superposition $(|0\rangle + |1\rangle)/\sqrt{2}$, which can be created with a Hadamard gate acting on $|0\rangle$. Then we apply U_f, resulting in the state:

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}} . \tag{1.37}$$

This is a remarkable state! The different terms contain information about both $f(0)$ and $f(1)$; it is almost as if we have evaluated $f(x)$ for two values of x simultaneously, a feature known as 'quantum parallelism'. Unlike classical parallelism, where multiple circuits each built to compute $f(x)$ are executed simultaneously, here a *single* $f(x)$ circuit is employed to evaluate the function for multiple values of x simultaneously, by exploiting the ability of a quantum computer to be in superpositions of different states.

This procedure can easily be generalized to functions on an arbitrary number of bits, by using a general operation known as the *Hadamard transform*, or sometimes the *Walsh–Hadamard transform*. This operation is just n Hadamard gates acting in parallel on n qubits. For example, shown in Figure 1.18 is the case $n = 2$ with qubits initially prepared as $|0\rangle$, which gives

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) = \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2} \tag{1.38}$$

as output. We write $H^{\otimes 2}$ to denote the parallel action of two Hadamard gates, and read '$\otimes$' as 'tensor'. More generally, the result of performing the Hadamard transform on n

qubits initially in the all $|0\rangle$ state is

$$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle \,, \tag{1.39}$$

where the sum is over all possible values of x, and we write $H^{\otimes n}$ to denote this action. That is, the Hadamard transform produces an equal superposition of all computational basis states. Moreover, it does this extremely efficiently, producing a superposition of 2^n states using just n gates.

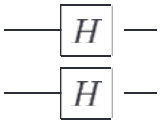

Figure 1.18. The Hadamard transform $H^{\otimes 2}$ on two qubits.

Quantum parallel evaluation of a function with an n bit input x and 1 bit output, $f(x)$, can thus be performed in the following manner. Prepare the $n + 1$ qubit state $|0\rangle^{\otimes n}|0\rangle$, then apply the Hadamard transform to the first n qubits, followed by the quantum circuit implementing U_f. This produces the state

$$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle|f(x)\rangle \,. \tag{1.40}$$

In some sense, quantum parallelism enables all possible values of the function f to be evaluated simultaneously, even though we apparently only evaluated f once. However, this parallelism is *not* immediately useful. In our single qubit example, measurement of the state gives only *either* $|0, f(0)\rangle$ *or* $|1, f(1)\rangle$! Similarly, in the general case, measurement of the state $\sum_x |x, f(x)\rangle$ would give only $f(x)$ for a single value of x. Of course, a classical computer can do this easily! Quantum computation requires something more than just quantum parallelism to be useful; it requires the ability to *extract* information about more than one value of $f(x)$ from superposition states like $\sum_x |x, f(x)\rangle$. Over the next two sections we investigate examples of how this may be done.

1.4.3 Deutsch's algorithm

A simple modification of the circuit in Figure 1.17 demonstrates how quantum circuits can outperform classical ones by implementing *Deutsch's algorithm* (we actually present a simplified and improved version of the original algorithm; see 'History and further reading' at the end of the chapter). Deutsch's algorithm combines quantum parallelism with a property of quantum mechanics known as *interference*. As before, let us use the Hadamard gate to prepare the first qubit as the superposition $(|0\rangle + |1\rangle)/\sqrt{2}$, but now let us prepare the second qubit y as the superposition $(|0\rangle - |1\rangle)/\sqrt{2}$, using a Hadamard gate applied to the state $|1\rangle$. Let us follow the states along to see what happens in this circuit, shown in Figure 1.19.

The input state

$$|\psi_0\rangle = |01\rangle \tag{1.41}$$

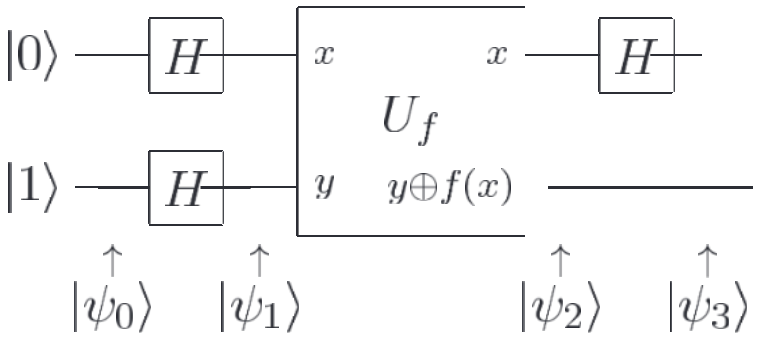

Figure 1.19. Quantum circuit implementing Deutsch's algorithm.

is sent through two Hadamard gates to give

$$|\psi_1\rangle = \left[\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right]\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{1.42}$$

A little thought shows that if we apply U_f to the state $|x\rangle(|0\rangle - |1\rangle)/\sqrt{2}$ then we obtain the state $(-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle)/\sqrt{2}$. Applying U_f to $|\psi_1\rangle$ therefore leaves us with one of two possibilities:

$$|\psi_2\rangle = \begin{cases} \pm\left[\dfrac{|0\rangle + |1\rangle}{\sqrt{2}}\right]\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{if } f(0) = f(1) \\ \pm\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right]\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{if } f(0) \neq f(1). \end{cases} \tag{1.43}$$

The final Hadamard gate on the first qubit thus gives us

$$|\psi_3\rangle = \begin{cases} \pm|0\rangle\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{if } f(0) = f(1) \\ \pm|1\rangle\left[\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}\right] & \text{if } f(0) \neq f(1). \end{cases} \tag{1.44}$$

Realizing that $f(0) \oplus f(1)$ is 0 if $f(0) = f(1)$ and 1 otherwise, we can rewrite this result concisely as

$$|\psi_3\rangle = \pm|f(0) \oplus f(1)\rangle\left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right], \tag{1.45}$$

so by measuring the first qubit we may determine $f(0) \oplus f(1)$. This is very interesting indeed: the quantum circuit has given us the ability to determine a *global property* of $f(x)$, namely $f(0) \oplus f(1)$, using only *one* evaluation of $f(x)$! This is faster than is possible with a classical apparatus, which would require at least two evaluations.

This example highlights the difference between quantum parallelism and classical randomized algorithms. Naively, one might think that the state $|0\rangle|f(0)\rangle + |1\rangle|f(1)\rangle$ corresponds rather closely to a probabilistic classical computer that evaluates $f(0)$ with probability one-half, or $f(1)$ with probability one-half. The difference is that in a classical computer these two alternatives forever exclude one another; in a quantum computer it is

possible for the two alternatives to *interfere* with one another to yield some global property of the function f, by using something like the Hadamard gate to recombine the different alternatives, as was done in Deutsch's algorithm. The essence of the design of many quantum algorithms is that a clever choice of function and final transformation allows efficient determination of useful global information about the function – information which cannot be attained quickly on a classical computer.

1.4.4 The Deutsch–Jozsa algorithm

Deutsch's algorithm is a simple case of a more general quantum algorithm, which we shall refer to as the Deutsch–Jozsa algorithm. The application, known as *Deutsch's problem*, may be described as the following game. Alice, in Amsterdam, selects a number x from 0 to $2^n - 1$, and mails it in a letter to Bob, in Boston. Bob calculates some function $f(x)$ and replies with the result, which is either 0 or 1. Now, Bob has promised to use a function f which is of one of two kinds; either $f(x)$ is *constant* for all values of x, or else $f(x)$ is *balanced*, that is, equal to 1 for exactly half of all the possible x, and 0 for the other half. Alice's goal is to determine with certainty whether Bob has chosen a constant or a balanced function, corresponding with him as little as possible. How fast can she succeed?

In the classical case, Alice may only send Bob one value of x in each letter. At worst, Alice will need to query Bob at least $2^n/2 + 1$ times, since she may receive $2^n/2$ 0s before finally getting a 1, telling her that Bob's function is balanced. The best deterministic classical algorithm she can use therefore requires $2^n/2 + 1$ queries. Note that in each letter, Alice sends Bob n bits of information. Furthermore, in this example, physical distance is being used to artificially elevate the cost of calculating $f(x)$, but this is not needed in the general problem, where $f(x)$ may be inherently difficult to calculate.

If Bob and Alice were able to exchange qubits, instead of just classical bits, and if Bob agreed to calculate $f(x)$ using a unitary transform U_f, then Alice could achieve her goal in just *one* correspondence with Bob, using the following algorithm.

Analogously to Deutsch's algorithm, Alice has an n qubit register to store her query in, and a single qubit register which she will give to Bob, to store the answer in. She begins by preparing both her query and answer registers in a superposition state. Bob will evaluate $f(x)$ using quantum parallelism and leave the result in the answer register. Alice then interferes states in the superposition using a Hadamard transform on the query register, and finishes by performing a suitable measurement to determine whether f was constant or balanced.

The specific steps of the algorithm are depicted in Figure 1.20. Let us follow the states through this circuit. The input state

$$|\psi_0\rangle = |0\rangle^{\otimes n}|1\rangle \tag{1.46}$$

is similar to that of Equation (1.41), but here the query register describes the state of n qubits all prepared in the $|0\rangle$ state. After the Hadamard transform on the query register and the Hadamard gate on the answer register we have

$$|\psi_1\rangle = \sum_{x\in\{0,1\}^n} \frac{|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{1.47}$$

The query register is now a superposition of all values, and the answer register is in an

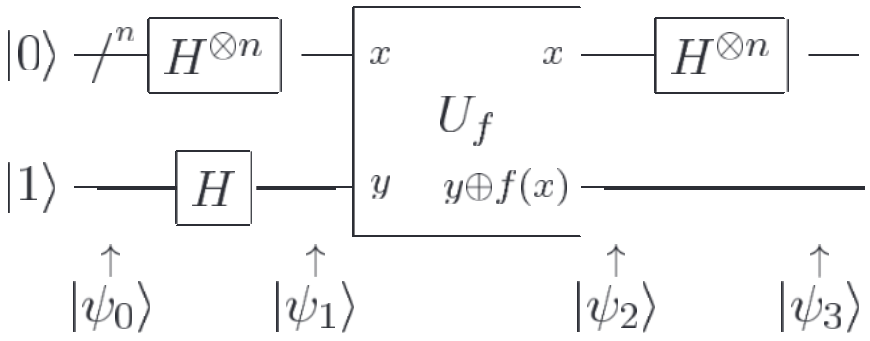

Figure 1.20. Quantum circuit implementing the general Deutsch–Jozsa algorithm. The wire with a '/' through it represents a set of n qubits, similar to the common engineering notation.

evenly weighted superposition of 0 and 1. Next, the function f is evaluated (by Bob) using $U_f : |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$, giving

$$|\psi_2\rangle = \sum_x \frac{(-1)^{f(x)}|x\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{1.48}$$

Alice now has a set of qubits in which the result of Bob's function evaluation is stored in the amplitude of the qubit superposition state. She now interferes terms in the superposition using a Hadamard transform on the query register. To determine the result of the Hadamard transform it helps to first calculate the effect of the Hadamard transform on a state $|x\rangle$. By checking the cases $x = 0$ and $x = 1$ separately we see that for a single qubit $H|x\rangle = \sum_z (-1)^{xz}|z\rangle/\sqrt{2}$. Thus

$$H^{\otimes n}|x_1, \ldots, x_n\rangle = \frac{\sum_{z_1,\ldots,z_n}(-1)^{x_1 z_1 + \cdots + x_n z_n}|z_1, \ldots, z_n\rangle}{\sqrt{2^n}}. \tag{1.49}$$

This can be summarized more succinctly in the very useful equation

$$H^{\otimes n}|x\rangle = \frac{\sum_z (-1)^{x \cdot z}|z\rangle}{\sqrt{2^n}}, \tag{1.50}$$

where $x \cdot z$ is the bitwise inner product of x and z, modulo 2. Using this equation and (1.48) we can now evaluate $|\psi_3\rangle$,

$$|\psi_3\rangle = \sum_z \sum_x \frac{(-1)^{x \cdot z + f(x)}|z\rangle}{2^n} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{1.51}$$

Alice now observes the query register. Note that the amplitude for the state $|0\rangle^{\otimes n}$ is $\sum_x (-1)^{f(x)}/2^n$. Let's look at the two possible cases – f constant and f balanced – to discern what happens. In the case where f is constant the amplitude for $|0\rangle^{\otimes n}$ is +1 or -1, depending on the constant value $f(x)$ takes. Because $|\psi_3\rangle$ is of unit length it follows that all the other amplitudes must be zero, and an observation will yield 0s for all qubits in the query register. If f is balanced then the positive and negative contributions to the amplitude for $|0\rangle^{\otimes n}$ cancel, leaving an amplitude of zero, and a measurement must yield a result other than 0 on at least one qubit in the query register. Summarizing, if Alice

measures all 0s then the function is constant; otherwise the function is balanced. The Deutsch–Jozsa algorithm is summarized below.

Algorithm: Deutsch–Jozsa

Inputs: (1) A black box U_f which performs the transformation $|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$, for $x \in \{0, \ldots, 2^n - 1\}$ and $f(x) \in \{0, 1\}$. It is promised that $f(x)$ is either *constant* for all values of x, or else $f(x)$ is *balanced*, that is, equal to 1 for exactly half of all the possible x, and 0 for the other half.

Outputs: 0 if and only if f is constant.

Runtime: One evaluation of U_f. Always succeeds.

Procedure:

1.	$\|0\rangle^{\otimes n}\|1\rangle$	initialize state
2.	$\rightarrow \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} \|x\rangle \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$	create superposition using Hadamard gates
3.	$\rightarrow \sum_x (-1)^{f(x)} \|x\rangle \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$	calculate function f using U_f
4.	$\rightarrow \sum_z \sum_x \frac{(-1)^{x \cdot z + f(x)} \|z\rangle}{\sqrt{2^n}} \left[\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}\right]$	perform Hadamard transform
5.	$\rightarrow z$	measure to obtain final output z

We've shown that a quantum computer can solve Deutsch's problem with one evaluation of the function f compared to the classical requirement for $2^n/2 + 1$ evaluations. This appears impressive, but there are several important caveats. First, Deutsch's problem is not an especially important problem; it has no known applications. Second, the comparison between classical and quantum algorithms is in some ways an apples and oranges comparison, as the method for evaluating the function is quite different in the two cases. Third, if Alice is allowed to use a probabilistic classical computer, then by asking Bob to evaluate $f(x)$ for a few randomly chosen x she can very quickly determine with high probability whether f is constant or balanced. This probabilistic scenario is perhaps more realistic than the deterministic scenario we have been considering. Despite these caveats, the Deutsch–Jozsa algorithm contains the seeds for more impressive quantum algorithms, and it is enlightening to attempt to understand the principles behind its operation.

Exercise 1.1: (Probabilistic classical algorithm) Suppose that the problem is not to distinguish between the constant and balanced functions *with certainty*, but rather, with some probability of error $\epsilon < 1/2$. What is the performance of the best classical algorithm for this problem?

1.4.5 Quantum algorithms summarized

The Deutsch–Jozsa algorithm suggests that quantum computers may be capable of solving some computational problems much more efficiently than classical computers. Unfortunately, the problem it solves is of little practical interest. Are there more interesting

problems whose solution may be obtained more efficiently using quantum algorithms? What are the principles underlying such algorithms? What are the ultimate limits of a quantum computer's computational power?

Broadly speaking, there are three classes of quantum algorithms which provide an advantage over known classical algorithms. First, there is the class of algorithms based upon quantum versions of the Fourier transform, a tool which is also widely used in classical algorithms. The Deutsch–Jozsa algorithm is an example of this type of algorithm, as are Shor's algorithms for factoring and discrete logarithm. The second class of algorithms is quantum search algorithms. The third class of algorithms is quantum simulation, whereby a quantum computer is used to simulate a quantum system. We now briefly describe each of these classes of algorithms, and then summarize what is known or suspected about the computational power of quantum computers.

Quantum algorithms based upon the Fourier transform

The discrete Fourier transform is usually described as transforming a set $x_0, \ldots, x_{N-1}$ of N complex numbers into a set of complex numbers $y_0, \ldots, y_{N-1}$ defined by

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N} x_j \, . \tag{1.52}$$

Of course, this transformation has an enormous number of applications in many branches of science; the Fourier transformed version of a problem is often easier than the original problem, enabling a solution.

The Fourier transform has proved so useful that a beautiful generalized theory of Fourier transforms has been developed which goes beyond the definition (1.52). This general theory involves some technical ideas from the character theory of finite groups, and we will not attempt to describe it here. What is important is that the Hadamard transform used in the Deutsch–Jozsa algorithm is an example of this generalized class of Fourier transforms. Moreover, many of the other important quantum algorithms also involve some type of Fourier transform.

The most important quantum algorithms known, Shor's fast algorithms for factoring and discrete logarithm, are two examples of algorithms based upon the Fourier transform defined in Equation (1.52). The Equation (1.52) does not appear terribly quantum mechanical in the form we have written it. Imagine, however, that we define a linear transformation U on n qubits by its action on computational basis states $|j\rangle$, where $0 \leq j \leq 2^n - 1$,

$$|j\rangle \longrightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i j k/2^n} |k\rangle \, . \tag{1.53}$$

It can be checked that this transformation is unitary, and in fact can be realized as a quantum circuit. Moreover, if we write out its action on superpositions,

$$\sum_{j=0}^{2^n-1} x_j |j\rangle \longrightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \left[\sum_{j=0}^{2^n-1} e^{2\pi i j k/2^n} x_j \right] |k\rangle = \sum_{k=0}^{2^n-1} y_k |k\rangle \, , \tag{1.54}$$

we see that it corresponds to a vector notation for the Fourier transform (1.52) for the case $N = 2^n$.

How quickly can we perform the Fourier transform? Classically, the fast Fourier transform takes roughly $N \log(N) = n2^n$ steps to Fourier transform $N = 2^n$ numbers. On a quantum computer, the Fourier transform can be accomplished using about $\log^2(N) = n^2$ steps, an exponential saving! The quantum circuit to do this is explained in Chapter 5.

This result seems to indicate that quantum computers can be used to very quickly compute the Fourier transform of a vector of 2^n complex numbers, which would be fantastically useful in a wide range of applications. However, that is *not* exactly the case; the Fourier transform is being performed on the information 'hidden' in the amplitudes of the quantum state. This information is not directly accessible to measurement. The catch, of course, is that if the output state is measured, it will collapse each qubit into the state $|0\rangle$ or $|1\rangle$, preventing us from learning the transform result y_k directly. This example speaks to the heart of the conundrum of devising a quantum algorithm. On the one hand, we can perform certain calculations on the 2^n amplitudes associated with n qubits far more efficiently than would be possible on a classical computer. But on the other hand, the results of such a calculation are not available to us if we go about it in a straightforward manner. More cleverness is required in order to harness the power of quantum computation.

Fortunately, it does turn out to be possible to utilize the quantum Fourier transform to efficiently solve several problems that are believed to have no efficient solution on a classical computer. These problems include Deutsch's problem, and Shor's algorithms for discrete logarithm and factoring. This line of thought culminated in Kitaev's discovery of a method to solve the *Abelian stabilizer problem*, and the generalization to the *hidden subgroup problem*,

> Let f be a function from a finitely generated group G to a finite set X such that f is constant on the cosets of a subgroup K, and distinct on each coset. Given a quantum black box for performing the unitary transform $U|g\rangle|h\rangle = |g\rangle|h \oplus f(g)\rangle$, for $g \in G$, $h \in X$, and $\oplus$ an appropriately chosen binary operation on X, find a generating set for K.

The Deutsch–Jozsa algorithm, Shor's algorithms, and related 'exponentially fast' quantum algorithms can all be viewed as special cases of this algorithm. The quantum Fourier transform and its applications are described in Chapter 5.

Quantum search algorithms

A completely different class of algorithms is represented by the quantum search algorithm, whose basic principles were discovered by Grover. The quantum search algorithm solves the following problem: Given a search space of size N, and no prior knowledge about the structure of the information in it, we want to find an element of that search space satisfying a known property. How long does it take to find an element satisfying that property? Classically, this problem requires approximately N operations, but the quantum search algorithm allows it to be solved using approximately $\sqrt{N}$ operations.

The quantum search algorithm offers only a quadratic speedup, as opposed to the more impressive exponential speedup offered by algorithms based on the quantum Fourier transform. However, the quantum search algorithm is still of great interest, since searching heuristics have a wider range of application than the problems solved using the quantum Fourier transform, and adaptations of the quantum search algorithm may have utility

for a very wide range of problems. The quantum search algorithm and its applications are described in Chapter 6.

Quantum simulation

Simulating naturally occurring quantum mechanical systems is an obvious candidate for a task at which quantum computers may excel, yet which is believed to be difficult on a classical computer. Classical computers have difficulty simulating general quantum systems for much the same reasons they have difficulty simulating quantum computers – the number of complex numbers needed to describe a quantum system generally grows *exponentially* with the size of the system, rather than linearly, as occurs in classical systems. In general, storing the quantum state of a system with n distinct components takes something like c^n bits of memory on a classical computer, where c is a constant which depends upon details of the system being simulated, and the desired accuracy of the simulation.

By contrast, a quantum computer can perform the simulation using kn qubits, where k is again a constant which depends upon the details of the system being simulated. This allows quantum computers to efficiently perform simulations of quantum mechanical systems that are believed not to be efficiently simulatable on a classical computer. A significant caveat is that even though a quantum computer can simulate many quantum systems far more efficiently than a classical computer, this does not mean that the fast simulation will allow the desired information about the quantum system to be obtained. When measured, a kn qubit simulation will collapse into a definite state, giving only kn bits of information; the c^n bits of 'hidden information' in the wavefunction is not entirely accessible. Thus, a crucial step in making quantum simulations useful is development of systematic means by which desired answers can be efficiently extracted; how to do this is only partially understood.

Despite this caveat, quantum simulation is likely to be an important application of quantum computers. The simulation of quantum systems is an important problem in many fields, notably quantum chemistry, where the computational constraints imposed by classical computers make it difficult to accurately simulate the behavior of even moderately sized molecules, much less the very large molecules that occur in many important biological systems. Obtaining faster and more accurate simulations of such systems may therefore have the welcome effect of enabling advances in other fields in which quantum phenomena are important.

In the future we may discover a physical phenomenon in Nature which cannot be efficiently simulated on a quantum computer. Far from being bad news, this would be wonderful! At the least, it will stimulate us to extend our models of computation to encompass the new phenomenon, and increase the power of our computational models beyond the existing quantum computing model. It also seems likely that very interesting new physical effects will be associated with any such phenomenon!

Another application for quantum simulation is as a general method to obtain insight into other quantum algorithms; for example, in Section 6.2 we explain how the quantum search algorithm can be viewed as the solution to a problem of quantum simulation. By approaching the problem in this fashion it becomes much easier to understand the origin of the quantum search algorithm.

Finally, quantum simulation also gives rise to an interesting and optimistic 'quantum corollary' to Moore's law. Recall that Moore's law states that the power of classical

computers will double once every two years or so, for constant cost. However, suppose we are simulating a quantum system on a classical computer, and want to add a single qubit (or a larger system) to the system being simulated. This doubles or more the memory requirements needed for a classical computer to store a description of the state of the quantum system, with a similar or greater cost in the time needed to simulate the dynamics. The quantum corollary to Moore's law follows from this observation, stating that quantum computers are keeping pace with classical computers provided a *single qubit* is added to the quantum computer every two years. This corollary should not be taken too seriously, as the exact nature of the gain, if any, of quantum computation over classical is not yet clear. Nevertheless, this heuristic statement helps convey why we should be interested in quantum computers, and hopeful that they will one day be able to outperform the most powerful classical computers, at least for some applications.

The power of quantum computation

How powerful are quantum computers? What gives them their power? Nobody yet knows the answers to these questions, despite the suspicions fostered by examples such as factoring, which strongly suggest that quantum computers are more powerful than classical computers. It is still possible that quantum computers are no more powerful than classical computers, in the sense that any problem which can be efficiently solved on a quantum computer can also be efficiently solved on a classical computer. On the other hand, it may eventually be proved that quantum computers are much more powerful than classical computers. We now take a brief look at what is known about the power of quantum computation.

Computational complexity theory is the subject of classifying the difficulty of various computational problems, both classical and quantum, and to understand the power of quantum computers we will first examine some general ideas from computational complexity. The most basic idea is that of a *complexity class*. A complexity class can be thought of as a collection of computational problems, all of which share some common feature with respect to the computational resources needed to solve those problems.

Two of the most important complexity classes go by the names **P** and **NP**. Roughly speaking, **P** is the class of computational problems that can be solved quickly on a classical computer. **NP** is the class of problems which have *solutions* which can be quickly checked on a classical computer. To understand the distinction between **P** and **NP**, consider the problem of finding the prime factors of an integer, n. So far as is known there is no fast way of solving this problem on a classical computer, which suggests that the problem is not in **P**. On the other hand, if somebody tells you that some number p is a factor of n, then we can quickly check that this is correct by dividing p into n, so factoring is a problem in **NP**.

It is clear that **P** is a subset of **NP**, since the ability to solve a problem implies the ability to check potential solutions. What is not so clear is whether or not there are problems in **NP** that are not in **P**. Perhaps the most important unsolved problem in theoretical computer science is to determine whether these two classes are different:

$$\mathbf{P} \stackrel{?}{\neq} \mathbf{NP}. \tag{1.55}$$

Most researchers believe that **NP** contains problems that are not in **P**. In particular, there is an important subclass of the **NP** problems, the **NP**-complete problems, that are

of especial importance for two reasons. First, there are thousands of problems, many highly important, that are known to be **NP**-complete. Second, any given **NP**-complete problem is in some sense 'at least as hard' as all other problems in **NP**. More precisely, an algorithm to solve a specific **NP**-complete problem can be adapted to solve any other problem in **NP**, with a small overhead. In particular, if $\mathbf{P} \neq \mathbf{NP}$, then it will follow that no **NP**-complete problem can be efficiently solved on a classical computer.

It is not known whether quantum computers can be used to quickly solve all the problems in **NP**, despite the fact that they can be used to solve some problems – like factoring – which are believed by many people to be in **NP** but not in **P**. (Note that factoring is not known to be **NP**-complete, otherwise we would already know how to efficiently solve all problems in **NP** using quantum computers.) It would certainly be very exciting if it were possible to solve all the problems in **NP** efficiently on a quantum computer. There is a very interesting negative result known in this direction which rules out using a simple variant of quantum parallelism to solve all the problems in **NP**. Specifically, one approach to the problem of solving problems in **NP** on a quantum computer is to try to use some form of quantum parallelism to search in parallel through all the possible solutions to the problem. In Section 6.6 we will show that no approach based upon such a search-based methodology can yield an efficient solution to all the problems in **NP**. While it is disappointing that this approach fails, it does not rule out that some deeper structure exists in the problems in **NP** that will allow them all to be solved quickly using a quantum computer.

P and **NP** are just two of a plethora of complexity classes that have been defined. Another important complexity class is **PSPACE**. Roughly speaking, **PSPACE** consists of those problems which can be solved using resources which are few in spatial size (that is, the computer is 'small'), but not necessarily in time ('long' computations are fine). **PSPACE** is believed to be strictly larger than both **P** and **NP** although, again, this has never been proved. Finally, the complexity class **BPP** is the class of problems that can be solved using randomized algorithms in polynomial time, if a bounded probability of error (say $1/4$) is allowed in the solution to the problem. **BPP** is widely regarded as being, even more so than **P**, the class of problems which should be considered efficiently soluble on a classical computer. We have elected to concentrate here on **P** rather than **BPP** because **P** has been studied in more depth, however many similar ideas and conclusions arise in connection with **BPP**.

What of quantum complexity classes? We can define **BQP** to be the class of all computational problems which can be solved efficiently on a quantum computer, where a bounded probability of error is allowed. (Strictly speaking this makes **BQP** more analogous to the classical complexity class **BPP** than to **P**, however we will ignore this subtlety for the purposes of the present discussion, and treat it as the analogue of **P**.) Exactly where **BQP** fits with respect to **P**, **NP** and **PSPACE** is as yet unknown. What is known is that quantum computers can solve all the problems in **P** efficiently, but that there are no problems outside of **PSPACE** which they can solve efficiently. Therefore, **BQP** lies somewhere between **P** and **PSPACE**, as illustrated in Figure 1.21. An important implication is that if it is proved that quantum computers are strictly more powerful than classical computers, then it will follow that **P** is not equal to **PSPACE**. Proving this latter result has been attempted without success by many computer scientists, suggesting that it may be non-trivial to prove that quantum computers are more powerful than classical computers, despite much evidence in favor of this proposition.

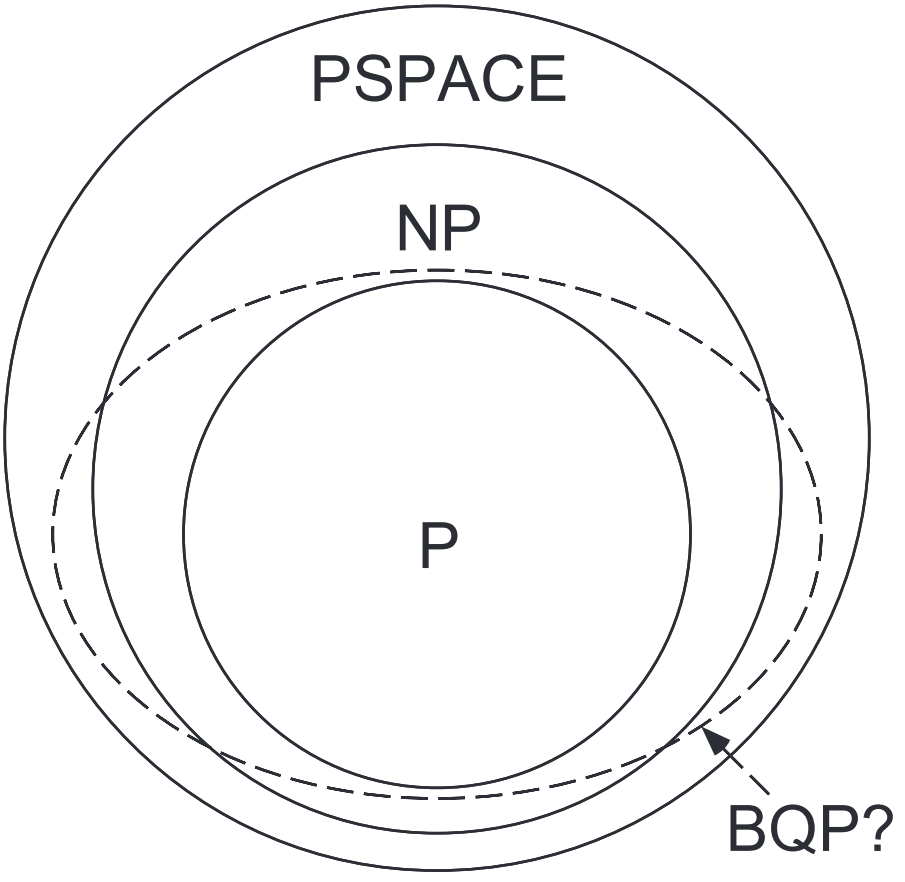

Figure 1.21. The relationship between classical and quantum complexity classes. Quantum computers can quickly solve any problem in **P**, and it is known that they can't solve problems outside of **PSPACE** quickly. Where quantum computers fit between **P** and **PSPACE** is not known, in part because we don't even know whether **PSPACE** is bigger than **P**!

We won't speculate further on the ultimate power of quantum computation now, preferring to wait until after we have better understood the principles on which fast quantum algorithms are based, a topic which occupies us for most of Part II of this book. What is already clear is that the *theory* of quantum computation poses interesting and significant challenges to the traditional notions of computation. What makes this an important challenge is that the theoretical model of quantum computation is believed to be *experimentally* realizable, because – to the best of our knowledge – this theory is consistent with the way Nature works. If this were not so then quantum computation would be just another mathematical curiosity.

1.5 Experimental quantum information processing

Quantum computation and quantum information is a wonderful theoretical discovery, but its central concepts, such as superpositions and entanglement, run counter to the intuition we garner from the everyday world around us. What evidence do we have that these ideas truly describe how Nature operates? Will the realization of large-scale quantum