Computational Complexity: A Modern Approach

Draft of a book: Dated January 2007
Comments welcome!

Sanjeev Arora and Boaz Barak
Princeton University
complexitybook@gmail.com

This is an Internet draft. Some chapters are more finished than others. References and attributions are very preliminary and we apologize in advance for any omissions (but hope you will nevertheless point them out to us).

Please send us bugs, typos, missing references or general comments to
complexitybook@gmail.com — **Thank You!!**

Chapter 20

Quantum Computation

"Turning to quantum mechanics.... secret, secret, close the doors! we always have had a great deal of difficulty in understanding the world view that quantum mechanics represents ... It has not yet become obvious to me that there's no real problem. I cannot define the real problem, therefore I suspect there's no real problem, but I'm not sure there's no real problem. So that's why I like to investigate things."
Richard Feynman 1964

"The only difference between a probabilistic classical world and the equations of the quantum world is that somehow or other it appears as if the probabilities would have to go negative.."
Richard Feynman, in "Simulating physics with computers", 1982

Quantum computers are a new computational model that may be physically realizable and may have an exponential advantage over 'classical" computational models such as probabilistic and deterministic Turing machines. In this chapter we survey the basic principles of quantum computation and some of the important algorithms in this model.

As complexity theorists, the main reason to study quantum computers is that they pose a serious challenge to the *strong Church-Turing thesis* that stipulates that any physically reasonable computation device can be simulated by a Turing machine with polynomial overhead. Quantum computers seem to violate no fundamental laws of physics and yet currently we do not know any such simulation. In fact, there is some evidence to the contrary: as we will see in Section 20.7, there is a polynomial-time algorithm for quantum computers to factor integers, where despite much effort no such algorithm is known for deterministic or probabilistic Turing machines. In fact, the conjectured hardness of this problem underlies of several cryptographic schemes (such as the RSA cryptosystem) that are currently widely used for electronic commerce and other applications. Physicists are also interested in quantum computers as studying them may shed light on quantum mechanics, a theory which, despite its great success in predicting experiments, is still not fully understood.

This chapter utilizes some basic facts of linear algebra, and the space $\mathbb{C}^n$. These are reviewed in Appendix A. See also Note 20.8 for a quick reminder of our notations.

20.1 Quantum weirdness

It is beyond this book (and its authors) to fully survey quantum mechanics. Fortunately, only very little physics is needed to understand the main results of quantum computing. However, these results do use some of the more counterintuitive notions of quantum mechanics such as the following:

> Any object in the universe, whether it is a particle or a cat, does not have definite properties (such as location, state, etc..) but rather has a kind of *probability wave* over its potential properties. The object only achieves a definite property when it is *observed*, at which point we say that the probability wave *collapses* to a single value.

At first this may seem like philosophical pontification analogous to questions such as "if a tree falls and no one hears, does it make a sound?" but these probability waves are in fact very real, in the sense that they can interact and interfere with one another, creating experimentally measurable effects. Below we describe two of the experiments that led physicists to accept this counterintuitive theory.

20.1.1 The 2-slit experiment

In the 2-slit experiment a source that fires electrons one by one (say, at the rate of one electron per second) is placed in front of a wall containing two tiny slits (see Figure **??**). Beyond the wall we place an array of detectors that light up whenever an electron hits them. We measure the number of times each detector lights up during an hour.

When we cover one of the slits, we would expect that the detectors that are directly behind the open slit will receive the largest number of hits, and as Figure **??** shows, this is indeed the case. When both slits are uncovered we expect that the number of times each detector is hit is the sum of the number of times it is hit when the first slit is open and the number of times it is hit when the second slit is open. In particular, uncovering both slits should only *increase* the number of times each location is hit.

Surprisingly, this is not what happens. The pattern of hits exhibits the "interference" phenomena depicted in Figure **??**. In particular, at several detectors the total electron flux is *lower* when both slits are open as compared to when a single slit is open. This defies explanation if electrons behave as particles or "little balls".

According to Quantum mechanics, the explanation is that it is wrong to think of an electron has a "little ball" that can either go through the first slit or the second (i.e., has a definite property). Rather, somehow the electron instantaneously explores all possible paths to the detectors (and so "finds out" how many slits are open) and then decides on a distribution among the possible paths that it will take.

You might be curious to see this "path exploration" in action, and so place a detector at each slit that light up whenever an electron passes through that slit. When this is done, one can see that every electron passes through only one of the slits like a nice little ball. But furthermore, the interference phenomenon disappears and the graph of electron hits becomes, as originally expected, the sum of the hits when each slit is open. The explanation is that, as stated above, *observing* an

NOTE 20.1 (PHYSICALLY IMPLEMENTING QUANTUM COMPUTERS.)

object "collapses" their distribution of possibilities, and so changes the result of the experiment. (One moral to draw from this is that quantum computers, if they are ever built, will have to be carefully isolated from external influences and noise, since noise may be viewed as a "measurement" performed by the environment on the system. Of course, we can never completely isolate the system, which means we have to make quantum computation tolerant of a little noise. This seems to be possible under some noise models, see the chapter notes.)

20.1.2 Quantum entanglement and the Bell inequalities.

Even after seeing the results of the 2-slit experiment, you might still be quite skeptical of the explanation that quantum mechanics offers. If you do, you are in excellent company. Albert Einstein didn't buy it either. While he agreed that the 2-slit experiment means that electrons are not exactly "little balls", he didn't think that it is sufficient reason to give up such basic notions of physics such as the existence of an independent reality, with objects having definite properties that do not depend on whether one is observing them. To show the dangerous outcomes of giving up such notions, in a 1951 paper with Podosky and Rosen (EPR for short) he described a thought experiment showing that accepting Quantum mechanics leads to the seemingly completely ridiculous conclusion that systems in two far corners of the universe can instantaneously coordinate their actions.

In 1964 John Bell showed how the principles behind EPR thought experiment can be turned into an *actual* experiment. In the years since, Bell's experiment has been repeated again and again with the same results: quantum mechanics' predictions were verified and, contrary to Einstein's expectations, the experiments refuted his intuitions about how the universe operates. In an interesting twist, in recent years the ideas behind EPR's and Bell's experiments were used for a practical goal: encryption schemes whose security depends only on the principles of quantum mechanics, rather than any unproven conjectures such as $\mathbf{P} \neq \mathbf{NP}$.

For complexity theorists, probably the best way to understand Bell's experiment is as a *two prover game*. Recall that in the two prover setting, two provers are allowed to decide on a strategy and then interact separately with a polynomial-time verifier which then decides whether to accept or reject the interaction (see Chapters 8 and 18). The provers' strategy can involve arbitrary computation and even be randomized, with the only constraint being that the provers are not allowed to communicate during their interaction with the verifier.

Bell's game. In Bell's setting, we have an extremely simple interaction between the verifier and two provers (that we'll name Alice and Bob): there is no statement that is being proven, and all the communication involves the verifier sending and receiving one bit from each prover. The protocol is as follows:

1. The verifier chooses two random bits $x, y \in_R \{0,1\}$.
2. It sends x to Alice and y to Bob.
3. Let a denote Alice's answer and b Bob's answer.
4. The verifier accepts if and only if $a \oplus b = x \wedge y$.

It is easy for Alice and Bob to ensure the verifier accepts with probability $3/4$ (e.g., by always sending $a = b = 0$). It turns out this is the best they can do:

THEOREM 20.2 ([?])
Regardless of the strategy the provers use, the verifier will accept with probability at most $3/4$.

PROOF: Assume for the sake of contradiction that there is a (possibly probabilistic) strategy that causes the verifier to accept with probability more than $3/4$. By a standard averaging argument there is a fixed set of provers' coins (and hence a deterministic strategy) that causes the verifier to accept with at least the same probability, and hence we may assume without loss of generality that the provers' strategy is deterministic.

A deterministic strategy for the two provers is a pair of functions $f, g : \{0,1\} \to \{0,1\}$ such as the provers' answers a, b are computed as $a = f(x)$ and $b = g(y)$. The function f can be one of only four possible functions: it can be either the constant function zero or one, the function $f(x) = x$ or the function $f(y) = 1 - y$. We analyze the case that $f(x) = x$; the other case are similar.

If $f(x) = x$ then the verifier accepts iff $b = (x \wedge y) \oplus x$. On input y, Bob needs to find b that makes the verifier accept. If $y = 1$ then $x \wedge y = x$ and hence $b = 0$ will ensure the verifier accepts with probability 1. However, if $y = 0$ then $(x \wedge y) \oplus x = x$ and since Bob does not know x, the probability that his output b is equal to x is at most $1/2$. Thus the total acceptance probability is at most $3/4$. ■

What does this game have to do with quantum mechanics? The main point is that according to "classical" pre-quantum physics, it is possible to ensure that Alice and Bob are isolated from one another. Suppose that you are given a pair of boxes that implement some arbitrary strategy for Bell's game. How can you ensure that these boxes don't have some secret communication mechanism that allows them to coordinate their answers? We might try to enclose the devices in lead boxes, but even this does not ensure complete isolation. However, Einstein's theory of relativity allows us a foolproof way to ensure complete isolation: place the two devices very far apart (say at a distance of a 1000 miles from one another), and perform the interaction with each prover at a breakneck speed: toss each of the coins x, y and demand the answer within less than one millisecond. Since according to the theory of relativity, nothing travels faster than light (that only covers about 200 miles in a millisecond), there is no way for the provers to communicate and coordinate their answers, no matter what is inside the box.

The upshot is that if someone provides you with such devices that consistently succeed in this experiment with more than $3/4 = 0.75$ probability, then she has refuted Einstein's theory. As we will see later in Section 20.3.2, quantum mechanics, with its instantaneous effects of measurements, can be used to actually build devices that succeed in this game with probability at least 0.8 (there are other games with more dramatic differences of probabilities) and this has been experimentally demonstrated.

20.2 A new view of probabilistic computation.

To understand quantum computation, it is helpful to first take a different viewpoint of a process we are already familiar with: probabilistic computation.

Suppose that we have an m-bit register. Normally, we think of such a register as having some definite value $x \in \{0,1\}^m$. However, in the context of probabilistic computation, we can also think of the register's state as actually being a *probability distribution* over its possible values. That is, we think of the register's state as being described by a 2^m-dimensional vector $\mathbf{v} = \langle \mathbf{v}_{0^m}, \mathbf{v}_{0^{m-1}1}, \ldots, \mathbf{v}_{1^m} \rangle$, where for every $x \in \{0,1\}^m$, $\mathbf{v_x} \in [0,1]$ and $\sum_x \mathbf{v}_x = 1$. When we read, or *measure*, the register, we will obtain the value x with probability $\mathbf{v}_x$.

For every $x \in \{0,1\}^n$, we denote by $|x\rangle$ the vector that corresponds to the degenerate distribution that takes the value x with probability 1. That is, $|x\rangle$ is the 2^m-dimensional vector that has zeroes in all the coordinates except a single 1 in the x^{th} coordinate. Note that $\mathbf{v} = \sum_{x\in\{0,1\}^m} \mathbf{v}_x |x\rangle$. (We think of all these vectors as column vectors in the space $\mathbb{R}^m$.)

Example 20.3
If a 1-bit register's state is the distribution that takes the value 0 with probability p and 1 with probability $1-p$, then we describe the state as the vector $p|0\rangle + (1-p)|1\rangle$.

The uniform distribution over the possible values of a 2-bit register is represented by $1/4\,(|00\rangle + |01\rangle + |10\rangle + |11$
The distribution that is uniform on every individual bit, but always satisfies that the two bits are equal is represented by $1/2\,(|00\rangle + |11\rangle)$.

An *probabilistic operation* on the register involves reading its value, and, based on the value read, modifying it in some deterministic or probabilistic way. If F is some probabilistic operation, then we can think of F as a function from $\mathbb{R}^{2^m}$ to $\mathbb{R}^{2^m}$ that maps the previous state of the register to its new state after the operation is performed. There are certain properties that *every* such operation must satisfy:

- Since F depends only on the contents of the register, and not on the overall distribution, for every $\mathbf{v}$, $F(\mathbf{v}) = \sum_{x\in\{0,1\}^m} \mathbf{v}_x F(|v\rangle)$. That is, F is a *linear* function. (Note that this means that F can be described by a $2^n \times 2^n$ matrix.)
- If $\mathbf{v}$ is a distribution vector (i.e., a vector of non-negative entries that sum up to one), then so is $F(\mathbf{v})$. That is, F is a *stochastic* function. (Note that this means that viewed as a matrix, F has non-negative entries with each column summing up to 1.)

Example 20.4
The operation that, regardless of the register's value, writes into it a uniformly chosen random string, is described by the function F such that $F(|x\rangle) = 2^{-m} \sum_{x\in\{0,1\}^m} |x\rangle$ for every $x \in \{0,1\}^m$. (Because the set $\{|x\rangle\}_{x\in\{0,1\}^n}$ is a *basis* for $\mathbb{R}^{2^m}$, a linear function is completely determined by its output on these vectors.)

The operation that flips the first bit of the register is described by the function F such that $F(|x_1 \ldots x_m\rangle) = |(1-x_1)x_2 \ldots x_m\rangle$ for every $x_1, \ldots, x_m \in \{0,1\}$.

Of course there are many probabilistic operations that cannot be efficiently computed, but there are very simple operations that can certainly be computed. An operation F is *elementary* if it only reads and modifies at most three bits of the register, leaving the rest of the register untouched. That is, there is some operation $G : \mathbb{R}^{2^3} \to \mathbb{R}^{2^3}$ and three indices $j, k, \ell \in [m]$ such that for every $x_1, \ldots, x_m \in \{0,1\}$, $F(|x_1 \ldots x_m\rangle) = |y_1 \ldots y_m\rangle$ where $|y_j y_k y_\ell\rangle = G(|x_j x_k x_\ell\rangle)$ and $y_i = x_i$ for every $i \notin \{j,k,\ell\}$. Note that such an operation can be represented by a $2^3 \times 2^3$ matrix and three indices in $[m]$.

Example 20.5
Here are some examples for operations depending on at most three bits:

AND function
F|xyz> = |xy(x∧y)>

	000	001	010	011	100	101	110	111
000	1	1	0	0	0	0	0	0
001	0	0	0	0	0	0	0	0
010	0	0	1	1	0	0	0	0
011	0	0	0	0	0	0	0	0
100	0	0	0	0	1	1	0	0
101	0	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0	0
111	0	0	0	0	0	0	1	1

Coin Tossing
F|x> = 1/2|0>+1/2|1>

	0	1
0	1/2	1/2
1	1/2	1/2

Constant zero function
F|x> = |0>

	0	1
0	1	1
1	0	0

For example, if we apply the coin tossing operation to the second bit of the register, then this means that for every $z = z_1 \ldots z_m \in \{0,1\}^m$, the vector $|z\rangle$ is mapped to $1/2\,|z_1 0 z_3 \ldots z_m\rangle + 1/2\,|z_1 1 z_3 \ldots z_m\rangle$.

We define a probabilistic computation to be a sequence of such elementary operations applied one after the other (see Definition 20.6 below). We will later see this corresponds exactly to our previous definition of probabilistic computation as in the class **BPP** defined in Chapter 7.

Definition 20.6 (Probabilistic Computation)
Let $f : \{0,1\}^* \to \{0,1\}^*$ and $T : \mathbb{N} \to \mathbb{N}$ be some functions. We say that f is *computable in probabilistic $T(n)$-time* if for every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, $f(x)$ can be computed by the following process:

1. Initialize an m bit register to the state $|x0^{n-m}\rangle$ (i.e., x padded with zeroes), where $m \leq T(n)$.
2. Apply one after the other $T(n)$ elementary operations $F_1, \ldots, F_T$ to the register (where we require that there is a polynomial-time TM that on input $1^n, 1^{T(n)}$ outputs the descriptions of $F_1, \ldots, F_T$).
3. Measure the register and let Y denote the obtained value. (That is, if $\mathbf{v}$ is the final state of the register, then Y is a random variable that takes the value y with probability $\mathbf{v}_y$ for every $y \in \{0,1\}^n$.)

Denoting $\ell = |f(x)|$, we require that the first ℓ bits of Y are equal to $f(x)$ with probability at least $2/3$.

PROBABILISTIC COMPUTATION: SUMMARY OF NOTATIONS.
The *state* of an m-bit register is represented by a vector $\mathbf{v} \in \mathbb{R}^{2^m}$ such that the register takes the value x with probability $\mathbf{v}_x$.
An *operation* on the register is a function $F : \mathbb{R}^{2^m} \to \mathbb{R}^{2^m}$ that is linear and stochastic.
An *elementary operation* only reads and modifies at most three bits of the register.
A *computation* of a function f on input $x \in \{0,1\}^n$ involves initializing the register to the state $|x0^{m-n}\rangle$, applying a sequence of elementary operations to it, and then measuring its value.

Now, as promised, we show that our new notion of probabilistic computation is equivalent to the one encountered in Chapter 7.

THEOREM 20.7
A Boolean function $f : \{0,1\}^* \to \{0,1\}$ is in **BPP** *iff it is computable in probabilistic $p(n)$-time for some polynomial $p : \mathbb{N} \to \mathbb{N}$.*

PROOF: ($\Rightarrow$) Suppose that $f \in$ **BPP**. As we saw before (e.g., in the proof of Theorem 6.7) this means that f can be computed by a polynomial-sized Boolean circuit C (that can be found by a deterministic poly-time TM) if we allow the circuit C access to random coins. Thus we can compute f as follows: we will use a register of $n+r+s$ bits, where r is the number of random coins C uses, and s is the number of gates C uses. That is, we have a location in the register for every coin and every gate of C. The elementary coin tossing operation (see Example 20.5) can transform a location initialized to 0 into a random coin. In addition, we have an elementary operation that transforms three bits x, y and z into $x, y, x \wedge y$ and can similarly define elementary operations for the OR and NOT functions. Thus, we can use these operations to ensure that for every gate of C, the corresponding location in the register contains the result of applying this gate when the circuit is evaluated on input x.
($\Leftarrow$) We will show a probabilistic polynomial-time algorithm to execute an elementary operation on a register. To simulate a $p(n)$-time probabilistic computation we can execute this algorithm $p(n)$ times one after the other. For concreteness, suppose that we need to execute an operation on the first three bits of the register, that is specified by an 8×8 matrix A. The algorithm will read the three bits to obtain the value $z \in \{0,1\}^3$, and then write to them a value chosen according to the distribution specified in the z^{th} column of A.

The only issue remaining is how to pick a value from an arbitrary distribution $(p_1, \ldots, p_8)$ over $\{0,1\}^3$ (which we identify with the set $[8]$). One case is simple: suppose that for every $i \in [8]$, $p_i = K_i/2^\ell$ where ℓ is polynomial in ℓ and $K_1, \ldots, K_8 \in [2^\ell]$. In this case, the algorithm will choose using ℓ random coins a number X between 1 and 2^ℓ and output the largest i such that $X \leq \sum_{j=1}^{i} K_i$.

However, this essentially captures general case as well: every number $p \in [0,1]$ can be approximated by a number of the form $K/2^\ell$ within $2^{-\ell}$ accuracy. This means that every general

NOTE 20.8 (A FEW NOTIONS FROM LINEAR ALGEBRA)
We use in this chapter several elementary facts and notations involving the space $\mathbb{C}^M$. These are reviewed in Appendix A, but here is a quick reminder:

- If $z = a + ib$ is a complex number (where $i = \sqrt{-1}$), then $\overline{z} = a - ib$ denotes the *complex conjugate* of z. Note that $z\overline{z} = a^2 + b^2 = |z|^2$.
- The *inner product* of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{C}^m$, denoted by $\langle \mathbf{u}, \mathbf{v} \rangle$, is equal to $\sum_{x \in [M]} \mathbf{u}_x \overline{\mathbf{v}}_x$.
- The *norm* of a vector $\mathbf{u}$, denoted by $\|\mathbf{u}\|_2$, is equal to $\sqrt{\langle \mathbf{u}, \mathbf{u} \rangle} = \sqrt{\sum_{x \in [M]} |\mathbf{u}_x|^2}$.
- If $\langle \mathbf{u}, \mathbf{v} \rangle = 0$ we say that $\mathbf{u}$ and $\mathbf{v}$ are *orthogonal*. More generally, $\odot \mathbf{u}, \mathbf{v} = \cos\theta \|\mathbf{u}\|_2 \|\mathbf{v}\|_2$, where θ is the angle between the two vectors $\mathbf{u}$ and $\mathbf{v}$.
- If A is an $M \times M$ matrix, then $A^\dagger$ denotes the *conjugate transpose* of A. That is, $A^\dagger_{x,y} = \overline{A}_{y,x}$ for every $x, y \in [M]$.
- An $M \times M$ matrix A is *unitary* if $AA^\dagger = I$, where I is the $M \times M$ identity matrix.

Note that if z is a *real* number (i.e., z has no imaginary component) then $\overline{z} = z$. Hence, if all vectors and matrices involved are real then the inner product is equal to the standard inner product of $\mathbb{R}^n$ and the conjugate transpose operation is equal to the standard transpose operation. Also a real matrix is unitary if and only if it is symmetric.

distribution can be well approximated by a distribution over the form above, and so by choosing a good enough approximation, we can simulate the probabilistic computation by a **BPP** algorithm. ■

20.3 Quantum superposition and the class BQP

A *quantum register* is also composed of m bits, but in quantum parlance they are called "qubits". In principle such a register can be implemented by any collection of m physical systems that can have an ON and OFF states, although in practice there are significant challenges for such an implementation (see Note 20.1). According to quantum mechanics, the state of such a register can be described by a 2^m-dimensional vector that, unlike the probabilistic case, can actually contain negative and even complex numbers. That is, the state of the register is described by a vector $\mathbf{v} \in \mathbb{C}^{2^m}$. Once again, we denote $\mathbf{v} = \sum_{x \in \{0,1\}^n} \mathbf{v}_x |x\rangle$ (where again $|x\rangle$ is the column vector that has all zeroes and a single one in the x^{th} coordinate). However, according to quantum mechanics,

when the register is *measured*, the probability that we see the value x is given not by $\mathbf{v}_x$ but by $|\mathbf{v}_x|^2$. This means that $\mathbf{v}$ has to be a *unit vector*, satisfying $\sum_x |\mathbf{v}_x|^2 = 1$ (see Note 20.8).

EXAMPLE 20.9
The following are two legitimate state vectors for a two-qubit quantum register: $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ and $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$. Even though in both cases, if the register is measured it will contain either 0 or 1 with probability $1/2$, these are considered distinct states and we will see that it is possible to differentiate between them using quantum operations.

Because states are always unit vectors, we often drop the normalization factor and so, say, use $|0\rangle - |1\rangle$ to denote the state $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$.

We call the state where all coefficients are equal the *uniform* state. For example, the uniform state for a 4-qubit register is

$$|00\rangle + |01\rangle + |10\rangle + |11\rangle\,,$$

(where we dropped the normalization factor of $\frac{1}{2}$.) We will also use the notation $|x\rangle\,|y\rangle$ to denote the standard basis vector $|xy\rangle$. It is easily verified that this operation respects the distributive law, and so we can also write the uniform state of a 4-qubit register as

$$(|0\rangle + |1\rangle)\,(|0\rangle + |1\rangle)$$

Once again, we can view an *operation* applied to the register as a function F that maps its previous state to the new state. That is, F is a function from $\mathbb{C}^{2^m}$ to $\mathbb{C}^{2^m}$. According to quantum mechanics, such an operation must satisfy the following conditions:

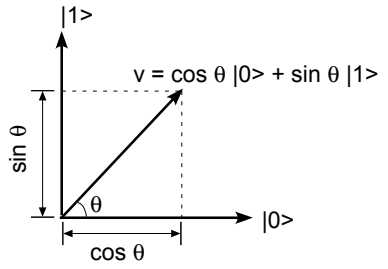
1. F is a linear function. That is, for every $\mathbf{v} \in \mathbb{C}^{2^n}$, $F(\mathbf{v}) = \sum_x \mathbf{v}_x F(|x\rangle)$.
2. F maps unit vectors to unit vectors. That is, for every $\mathbf{v}$ with $\|\mathbf{v}\|_2 = 1$, $\|F(\mathbf{v})\|_2 = 1$.

Together, these two conditions imply that F can be described by a $2^m \times 2^m$ *unitary* matrix. That is, a matrix A satisfying $AA^\dagger = I$ (see Note 20.8). We recall the following simple facts about unitary matrices (left as Exercise 1):

CLAIM 20.11
For every $M \times M$ complex matrix A, the following conditions are equivalent:

1. *A is unitary (i.e., $AA^\dagger = I$).*
2. *For every vector $\mathbf{v} \in \mathbb{C}^M$, $\|A\mathbf{v}\|_2 = \|\mathbf{v}\|_2$.*
3. *For every orthonormal basis $\{\mathbf{v}^i\}_{i\in[M]}$ of $\mathbb{C}^M$ (see below), the set $\{A\mathbf{v}^i\}_{i\in[M]}$ is an orthonormal basis of $\mathbb{C}^M$.*
4. *The columns of A form an orthonormal basis of $\mathbb{C}^M$.*

NOTE 20.10 (THE GEOMETRY OF QUANTUM STATES)
It is often helpful to think of quantum states geometrically as vectors in space. For example, consider a single qubit register, in which case the state is a unit vector in the two-dimensional plane spanned by the orthogonal vectors $|0\rangle$ and $|1\rangle$. For example, the state $\mathbf{v} = \cos\theta\,|0\rangle + \sin\theta\,|1\rangle$ corresponds to a vector making an angle θ with the $|0\rangle$ vector and an angle $\pi/2 - \theta$ with the $|1\rangle$ vector. When $\mathbf{v}$ is measured it will yield 0 with probability $\cos^2\theta$ and 1 with probability $\sin^2\theta$.



Although it's harder to visualize states with complex coefficients or more than one qubit, geometric intuition can still be useful when reasoning about such states.

5. *The rows of A form an orthonormal basis of $\mathbb{C}^M$.*

(Recall that a set $\{\mathbf{v}^i\}_{i\in[M]}$ of vectors in $\mathbb{C}^M$ is an *orthonormal basis* of $\mathbb{C}^M$ if for every $i, j \in [M]$, $\langle \mathbf{v}^i, \mathbf{v}^j \rangle$ is equal to 1 if $i = j$ and equal to 0 if $i \neq j$, where $\langle \mathbf{v}, \mathbf{u} \rangle$ is the standard inner product over $\mathbb{C}^M$. That is, $\langle \mathbf{v}, \mathbf{u} \rangle = \sum_{j=1}^{M} \mathbf{v}_i \overline{\mathbf{u}}_i$.)

As before, we define an *elementary quantum operation* to be an operation that only depends and modifies at most three qubits of the register.

EXAMPLE 20.12

Here are some examples for quantum operations depending on at most three qubits. (Because all the quantum operations are linear, it suffices to describe their behavior on any linear basis for the space $\mathbb{C}^{2^m}$ and so we often specify quantum operations by the way they map the standard basis.)

- The standard NOT operation on a single bit can be thought of as the unitary operation that maps $|0\rangle$ to $|1\rangle$ and vice versa.

- The *Hadamard* operation is the single qubit operation that (up to normalization) maps $|0\rangle$ to $|0\rangle + |1\rangle$ and $|1\rangle$ to $|0\rangle - |1\rangle$. (More succinctly, the state $|b\rangle$ is mapped to $|0\rangle + (-1)^b |1\rangle$.)

  It turns out to be a very useful operation in many algorithms for quantum computers. Note that if we apply an Hadamard operation to every qubit of an n-qubit register, then for every $x \in \{0,1\}^n$, the state $|x\rangle$ is mapped to

$$(|0\rangle + (-1)^{x_1}|1\rangle)(|0\rangle + (-1)^{k_2}|1\rangle)\cdots(|0\rangle + (-1)^{x_n}|1\rangle) = \sum_{y\in\{0,1\}^n} (\pi_{i\,:\,y_i=1}(-1)^x_i)\,|y\rangle = \sum_{y\in\{0,1\}^n} -1^{x\odot y}\,|y\rangle\,,$$

  where $x \odot y$ denotes the inner product modulo 2 of x and y. That is, $x \odot y = \sum_{i=1}^{n} x_i y_i \pmod 2$.[1]

- Since we can think of the state of a single qubit register as a vector in two dimensional space, a natural operation is for any angle θ, to rotate the single qubit by θ. That is, map $|0\rangle$ to $\cos\theta\,|0\rangle + \sin\theta\,|1\rangle$, and map $|1\rangle$ to $-\sin\theta\,|0\rangle + \cos\theta\,|1\rangle$. Note that rotation by an angle of π (i.e., 180°) is equal to flipping the sign of the vector (i.e., the map $\mathbf{v} \mapsto -\mathbf{v}$).

- One simple two qubit operation is exchanging the two bits with one another. That is, mapping $|01\rangle \mapsto |10\rangle$ and $|10\rangle \mapsto |01\rangle$, with $|00\rangle$ and $|11\rangle$ being mapped to them selves. Note that by combining these operations we can reorder the qubits of an n-bit register in any way we see fit.

- Another two qubit operation is the controlled-NOT operation: it performs a NOT on the first bit iff the second bit is equal to 1. That is, it maps $|01\rangle \mapsto |11\rangle$ and $|11\rangle \mapsto |01\rangle$, with $|10\rangle$ and $|11\rangle$ being mapped to themselves.

[1] Note the similarity to the definition of the Walsh-Hadamard code described in Chapter 17, Section 17.5.

- The *Tofolli operation* is the three qubit operation that can be called a "controlled-controlled-NOT": it performs a NOT on the first bit iff both the second and third bits are equal to 1. That is, it maps $|011\rangle$ to $|111\rangle$ and vice versa, and maps all other basis states to themselves.

Exercise 2 asks you to write down explicitly the matrices for these operations.

We define quantum computation as consisting of a sequence of elementary operations in an analogous way to our previous definition of probabilistic computation (Definition 20.6):

DEFINITION 20.13 (QUANTUM COMPUTATION)
Let $f : \{0,1\}^* \to \{0,1\}^*$ and $T : \mathbb{N} \to \mathbb{N}$ be some functions. We say that f is *computable in quantum $T(n)$-time* if for every $n \in \mathbb{N}$ and $x \in \{0,1\}^n$, $f(x)$ can be computed by the following process:

1. Initialize an m qubit quantum register to the state $|x0^{n-m}\rangle$ (i.e., x padded with zeroes), where $m \leq T(n)$.
2. Apply one after the other $T(n)$ elementary quantum operations $F_1, \ldots, F_T$ to the register (where we require that there is a polynomial-time TM that on input $1^n, 1^{T(n)}$ outputs the descriptions of $F_1, \ldots, F_T$).
3. Measure the register and let Y denote the obtained value. (That is, if $\mathbf{v}$ is the final state of the register, then Y is a random variable that takes the value y with probability $|\mathbf{v}_y|^2$ for every $y \in \{0,1\}^n$.)

Denoting $\ell = |f(x)|$, we require that the first ℓ bits of Y are equal to $f(x)$ with probability at least $2/3$.

The following class aims to capture the decision problems with efficient algorithms on quantum computers:

DEFINITION 20.14 (THE CLASS **BQP**)
A Boolean function $f : \{0,1\}^* \to \{0,1\}$ is in **BQP** if there is some polynomial $p : \mathbb{N} \to \mathbb{N}$ such that f is computable in quantum $p(n)$-time.

QUANTUM COMPUTATION: SUMMARY OF NOTATIONS.
The *state* of an m-qubit register is represented by a unit vector $\mathbf{v} \in \mathbb{C}^{2^m}$ such that the register takes the value x with probability $|\mathbf{v}_x|^2$.
An *operation* on the register is a function $F : \mathbb{C}^{2^m} \to \mathbb{C}^{2^m}$ that is unitary. (i.e., linear and norm preserving).
An *elementary operation* only depends and modifies at most three bits of the register.
A *computation* of a function f on input $x \in \{0,1\}^n$ involves initializing the register to the state $|x0^{m-n}\rangle$, applying a sequence of elementary operations to it, and then measuring its value.

REMARK 20.15
Readers familiar with quantum mechanics or quantum computing may notice that we did not allow in our definition of quantum computation several features that are allowed by quantum mechanics. These include *mixed* states, that involve both quantum superposition and probability, measuring in different basis than the standard basis, and performing partial measurements during the computation. However, none of these features adds to the computing power of quantum computers.

20.3.1 Universal quantum operations

Can we actually implement quantum computation? This is an excellent question, and no one really knows. However, one hurdle can be overcome: even though there is an infinite set of possible elementary operations, all of them can be generated (or at least sufficiently well approximated) by the Hadamard and Tofolli operations described in Example 20.12. In fact, every operation that depends on k qubits can be approximated by composing $2^{O(k)}$ of these four operations (times an additional small factor depending on the approximation's quality). Using a counting/dimension argument, it can be shown that some unitary transformations do indeed require an exponential number of elementary operations to compute (or even approximate).

One useful consequence of universality is the following: when designing quantum algorithms we can assume that we have at our disposal the all operations that depend on k qubits as elementary operations, for every constant k (even if $k > 3$). This is since these can be implemented by 3 qubit elementary operations incurring only a constant (depending on k) overhead.

20.3.2 Spooky coordination and Bell's state

To get our first glimpse of how things behave differently in the quantum world, we will now show how quantum registers and operations help us win the game described in Section 20.1.2 with higher probability than can be achieved according to pre-quantum "classical" physics.
Recall that the game was the following:

1. The verifier chooses random $x, y \in_R \{0,1\}$ and sends x to Alice and y to Bob, collecting their respective answers a and b.

2. It accepts iff $x \wedge y = a \oplus b$. In other words, it accepts if either $\langle x, y\rangle \neq \langle 1,1\rangle$ and $a = b$ or $x = y = 1$ and $a \neq b$.

It was shown in Section 20.1.2 that if Alice and Bob can not coordinate their actions then the verifier will accept with probability at most 3/4, but quantum effects can allow them to bypass this bound as follows:

1. Alice and Bob prepare a 2-qubit quantum register containing the EPR state $|00\rangle + |11\rangle$. (They can start with a register initialized to $|00\rangle$ and then apply an elementary operation that maps the initial state to the EPR state.)

2. Alice and Bob split the register - Alice takes the first qubit and Bob takes the second qubit. (The components containing each qubit of a quantum register do not necessarily need to be adjacent to one another.)

3. Alice receives the qubit x from the verifier, and if $x = 1$ then she applies a rotation by $\pi/8$ (22.5°) operation to her qubit. (Since the operation involves only her qubit, she can perform it even after the register was split.)

4. Bob receives the qubit y from the verifier, and if $y = 1$ he applies a rotation by by $-\pi/8$ ($-22.5°$) operation to his qubit.

5. Both of them measure their respective qubits and output the values obtained as their answers a and b.

Note that the order in which Alice and Bob perform their rotations and measurements does not matter - it can be shown that all orders yield exactly the same distribution (e.g., see Exercise 3). While splitting a quantum register and applying unitary transformations to the different parts may sound far fetched, this experiment had been performed several times in practice, verifying the following predictions of quantum mechanics:

THEOREM 20.16
Given the above strategy for Alice and Bob, the verifier will accept with probability at least 0.8.

PROOF: Recall that Alice and Bob win the game if they output the same answer when $\langle x, y\rangle \neq \langle 1, 1\rangle$ and a different answer otherwise. The intuition behind the proof is that in the case that $\langle x, y\rangle \neq \langle 1, 1\rangle$ then the states of the two qubits will be "close" to one another (the angle between them is less than $\pi/8$ or 22.5°) and in the other case the states will be "far" (having angle $\pi/4$ or 45°). Specifically we will show that:

(1) If $x = y = 0$ then $a = b$ with probability 1.

(2) If $x \neq y$ then $a = b$ with probability $\cos^2(\pi/8) \geq 0.85$

(3) If $x = y = 1$ then $a = b$ with probability 1/2.

Implying that the overall acceptance probability is at least $\frac{1}{4} + \frac{1}{2}0.85 + \frac{1}{8} = 0.8$.

In the case **(1)** both Alice and Bob perform no operation on their register, and so when measured it will be either in the state $|00\rangle$ or $|11\rangle$, both resulting in Alice and Bob's outputs being equal. To analyze case **(2)**, it suffices to consider the case that $x = 0, y = 1$ (the other case is symmetrical).

In this case Alice applies no transformation to her qubit, and Bob rotates his qubit in a $-\pi/8$ angle. Imagine that Bob first makes the rotation, then Alice measures her qubit and then Bob measures his (this is OK as the order of measurements does not change the outcome). With probability $1/2$, Alice will get the value 0 and Bob's qubit will collapse to the state $|0\rangle$ rotated by a $-\pi/8$ angle, meaning that when measuring Bob will obtain the value 0 with probability $\cos^2(\pi/8)$. Similarly, if Alice gets the value 1 then Bob will also output 1 with $\cos^2(\pi/8)$ probability.

To analyze case **(3)**, we just use direct computation. In this case, after both rotations are performed, the register's state is

$$\begin{aligned}(\cos(\pi/8)\,|0\rangle + \sin(\pi/8)\,|1\rangle)\,(\cos(\pi/8)\,|0\rangle - \sin(\pi/8)\,|1\rangle) + \\ (-\sin(\pi/8)\,|0\rangle + \cos(\pi/8)\,|1\rangle)\,(\sin(\pi/8)\,|0\rangle + \cos(\pi/8)\,|1\rangle) = \\ \left(\cos^2(\pi/8) - \sin^2(\pi/8)\right)|00\rangle - 2\sin(\pi/8)\cos(\pi/8)\,|01\rangle + \\ 2\sin(\pi/8)\cos(\pi/8)\,|10\rangle + \left(\cos^2(\pi/8) - \sin^2(\pi/8)\right)|11\rangle\,.\end{aligned}$$

But since

$$\cos^2(\pi/8) - \sin^2(\pi/8) = \cos(\pi/4) = \tfrac{1}{\sqrt{2}} = \sin(\pi/4) = 2\sin(\pi/8)\cos(\pi/8)\,,$$

all coefficients in this state have the same absolute value and hence when measured the register will yield either one of the four values $00, 01, 10$ and 11 with equal probability $1/4$. ■

20.4 Quantum programmer's toolkit

Quantum algorithms have some peculiar features that classical algorithm designers are not used to. The following observations can serve as a helpful "bag of tricks" for designing quantum algorithms:

- If we can compute an n-qubit unitary transformation U in T steps then we can compute the transformation Controlled-U in $O(T)$ steps, where Controlled-U maps a vector $|x_1 \ldots x_n x_{n+1}\rangle$ to $|U(x_1 \ldots x_n)x_{n+1}\rangle$ if $x_{n+1} = 1$ and to itself otherwise.

  The reason is that we can transform every elementary operation F in the computation of U to the analogous "Controlled-F" operation. Since the "Controlled-F" operation depends on at most 4 qubits, it can be considered also as elementary.

  For every two n-qubit transformations U, U', we can use this observation twice to compute the transformation that invokes U on $x_1 \ldots x_n$ if $x_{n+1} = 1$ and invokes U' otherwise.

- Every *permutation* of the standard basis is unitary. That is, any operation that maps a vector $|x\rangle$ into $|\pi(x)\rangle$ where π is a permutation of $\{0,1\}^n$ is unitary. Of course, this does not mean that all such permutations are efficiently computable in quantum polynomial time.

- For every function $f : \{0,1\}^n \to \{0,1\}^\ell$, the function $x, y \mapsto x, (y \oplus f(x))$ is a permutation on $\{0,1\}^{n+\ell}$ (in fact, this function is its own inverse). In particular, this means that we can use as elementary operations the following "permutation variants" of AND, OR and

copying: **(1)** $|x_1x_2x_3\rangle \mapsto |x_1x_2(x_3 \oplus (x_1 \wedge x_2))\rangle$, **(2)** $|x_1x_2x_3\rangle \mapsto |x_1x_2(x_3 \oplus (x_1 \vee x_2))\rangle$, and **(3)** $|x_1x_2\rangle \mapsto |x_1(x_1 \oplus x_2)\rangle$. Note that in all these cases we compute the "right" function if the last qubit is initially equal to 0.

- If $f : \{0,1\}^n \to \{0,1\}^\ell$ is a function computable by a size-T Boolean circuit, then the following transformation can be computed by a sequence of $O(T)$ elementary operations: for every $x \in \{0,1\}^m, y \in \{0,1\}^\ell, z \in \{0,1\}^T$

$$|x, y, z\rangle \mapsto |x, (y \oplus f(x)), 0^T\rangle \text{ if } z = 0^T.$$

  (We don't care on what the mapping does for $z \neq 0^T$.)

  The reason is that by transforming every AND, OR or NOT gate into the corresponding elementary permutation we can ensure that the i^{th} qubit of z contains the result of the i^{th} gate of the circuit when executed on input x. We can then XOR the result of the circuit into y using ℓ elementary operations and run the entire computation backward to return the state of z to 0^T. .

- We can assume that we are allowed to make a *partial measurement* in the course of the algorithm, and then proceed differently according to its outcome. That is, we can measure a some of the qubits of the register. Note that if the register is at state $\mathbf{v}$ and we measure its i^{th} qubit then with probability $\sum_{z:z_i=1} |\mathbf{v}_z|^2$ we will get the answer "1" and the register's state will change to (the normalized version of) the vector $\sum_{z:z_i=1} \mathbf{v}_z |z\rangle$. Symmetrically, with probability $\sum_{z:z_i=0} |\mathbf{v}_z|^2$ we will get the answer "0" and the new state will be $\sum_{z:z_i=0} \mathbf{v}_z |z\rangle$.

  This is allowed since an algorithm using partial measurement can be replaced with an algorithm not using it with at most a constant overhead (see Exercise 4).

- Since the 1-qubit Hadamard operation maps $|0\rangle$ to the uniform state $|0\rangle + |1\rangle$, it can be used to simulate tossing a coin: we simply take a qubit in our workspace that is initialized to 0, apply Hadamard to it, and measure the result.

Together, the last three observations imply that quantum computation is at least as powerful as "classical" non-quantum computation:

THEOREM 20.17
BPP $\subseteq$ **BQP**.

20.5 Grover's search algorithm.

Consider the **NP**-complete problem SAT of finding, given an n-variable Boolean formula φ, whether there exists an assignment $a \in \{0,1\}^n$ such that $\varphi(a) = 1$. Using "classical" deterministic or probabilistic TM's, we do not know how to solve this problem better than the trivial $\text{poly}(n)2^n$-time algorithm.[2] We now show a beautiful algorithm due to Grover that solves SAT in $\text{poly}(n)2^{n/2}$-time on a quantum computer. This is a significant improvement over the classical case, even if it falls

[2]There are slightly better algorithms for special cases such as 3SAT.

way short of showing that **NP** $\subseteq$ **BQP**. In fact, Grover's algorithm solves an even more general problem: for *every* polynomial-time computable function $f : \{0,1\}^n \to \{0,1\}$ (even if f is not expressed as a small Boolean formula[3]), it finds in $\mathrm{poly}(n)2^{n/2}$ time a string a such that $f(a) = 1$ (if such a string exists).

Grover's algorithm is best described geometrically. We assume that the function f has a *single* satisfying assignment a. (The techniques described in Chapter 9, Section 9.3.1 allow us to reduce the general problem to this case.) Consider an n-qubit register, and let $\mathbf{u}$ denote the *uniform state vector* of this register. That is, $\mathbf{u} = \frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle$. The angle between $\mathbf{u}$ and $|a\rangle$ is equal to the inverse cosine of their inner product $\langle \mathbf{u}, |a\rangle \rangle = \frac{1}{2^{n/2}}$. Since this is a positive number, this angle is smaller than $\pi/2$ (90°), and hence we denote it by $\pi/2 - \theta$, where $\sin\theta = \frac{1}{2^{n/2}}$ and hence, assuming n is sufficiently large, $\theta \geq \frac{1}{2\cdot 2^{n/2}}$ (since for small θ, $\sin\theta \sim \theta$).

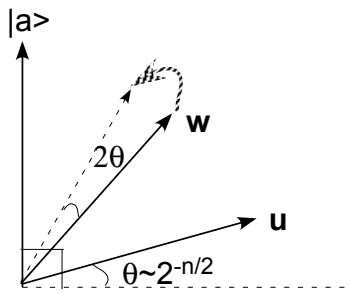


Figure 20.1: Grover's algorithm finds the string a such that $f(a) = 1$ as follows. It starts with the uniform vector $\mathbf{u}$ whose angle with $|a\rangle$ is $\pi/2 - \theta$ for $\theta \sim 2^{-n/2}$ and at each step transforms the state of the register into a vector that is 2θ radians closer to $|a\rangle$. After $O(1/\theta)$ steps, the state is close enough so that measuring the register yields $|a\rangle$ with good probability.

The algorithm starts with the state $\mathbf{u}$, and at each step it gets nearer the state $|a\rangle$ by transforming its current state to a state whose angle with $|a\rangle$ is smaller by 2θ (see Figure 20.1). Thus, in $O(1/\theta) = O(2^{n/2})$ steps it will get to a state $\mathbf{v}$ whose inner product with $|a\rangle$ is larger than, say, $1/2$, implying that a measurement of the register will yield a with probability at least $1/4$.

The main idea is that to rotate a vector $\mathbf{w}$ towards the unknown vector $|a\rangle$ by an angle of θ, it suffices to take two *reflections* around the vector $\mathbf{u}$ and the vector $\mathbf{e} = \sum_{x\neq a} |a\rangle$ (the latter is the vector orthogonal to $|a\rangle$ on the plane spanned by $\mathbf{u}$ and $|a\rangle$). See Figure 20.2 for a "proof by picture".

To complete the algorithm's description, we need to show how we can perform the reflections around the vectors $\mathbf{u}$ and $\mathbf{e}$. That is, we need to show how we can in polynomial time transform a state $\mathbf{w}$ of the register into the state that is $\mathbf{w}$'s reflection around $\mathbf{u}$ (respectively, $\mathbf{e}$). In fact, we will not work with an n-qubit register but with an m-qubit register for m that is polynomial in n. However, the extra qubits will only serve as "scratch workspace" and will always contain zero except during intermediate computations, and hence can be safely ignored.

[3] We may assume that f is given to the algorithm in the form of a polynomial-sized circuit.

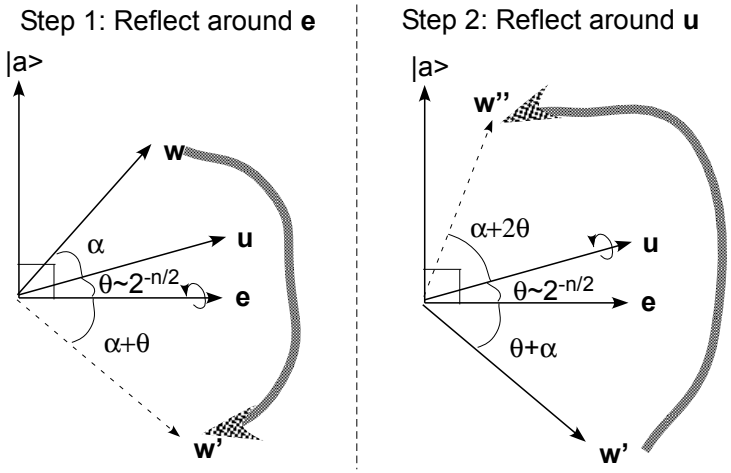


Figure 20.2: We transform a vector $\mathbf{w}$ in the plane spanned by $|a\rangle$ and $\mathbf{u}$ into a vector $\mathbf{w}''$ that is 2θ radians close to $|a\rangle$ by performing two reflections. First, we reflect around $\mathbf{e} = \sum_{x \neq a} |x\rangle$ (the vector orthogonal to $|a\rangle$ on this plane), and then we reflect around $\mathbf{u}$. If the original angle between $\mathbf{w}$ and $|a\rangle$ was $\pi/2 - \theta - \alpha$ then the new angle will be $\pi/2 - \theta - \alpha - 2\theta$.

Reflecting around e. Recall that to reflect a vector $\mathbf{w}$ around a vector $\mathbf{v}$, we express $\mathbf{w}$ as $\alpha\mathbf{v} + \mathbf{v}^\perp$ (where $\mathbf{v}^\perp$ is orthogonal to $\mathbf{v}$) and output $\alpha\mathbf{v} - \mathbf{v}^\perp$. Thus the reflection of $\mathbf{w}$ around $\mathbf{e}$ is equal to $\sum_{x \neq a} \mathbf{w}_x |x\rangle - \mathbf{w}_a |a\rangle$. Yet, it is easy to perform this transformation:

1. Since f is computable in polynomial time, we can compute the transformation $|x\sigma\rangle \mapsto |x(\sigma \oplus f(x))\rangle$ in polynomial (this notation ignores the extra workspace that may be needed, but this won't make any difference). This transformation maps $|x0\rangle$ to $|x0\rangle$ for $x \neq a$ and $|a0\rangle$ to $|a1\rangle$.

2. Then, we apply the elementary transformation that multiplies the vector by -1 if $\sigma = 1$, and does nothing otherwise. This maps $|x0\rangle$ to $|x0\rangle$ for $x \neq a$ and maps $|a1\rangle$ to $-|a1\rangle$.

3. Then, we apply the transformation $|x\sigma\rangle \mapsto |x(\sigma \oplus f(x))\rangle$ again, mapping $|x0\rangle$ to $|x0\rangle$ for $x \neq a$ and maps $|a1\rangle$ to $|a0\rangle$.

The final result is that the vector $|x0\rangle$ is mapped to itself for $x \neq a$, but $|a0\rangle$ is mapped to $-|a0\rangle$. Ignoring the last qubit, this is exactly a reflection around $|a\rangle$.

Reflecting around u. To reflect around $\mathbf{u}$, we first apply the Hadamard operation to each qubit, mapping $\mathbf{u}$ to $|0\rangle$. Then, we reflect around $|0\rangle$ (this can be done in the same way as reflecting around $|a\rangle$, just using the function $g : \{0,1\}^n \to \{0,1\}$ that outputs 1 iff its input is all zeroes instead of f). Then, we apply the Hadamard operation again, mapping $|0\rangle$ back to $\mathbf{u}$.

Together these operations allow us to take a vector in the plane spanned by $|a\rangle$ and $\mathbf{u}$ and rotate it 2θ radians closer to $|a\rangle$. Thus if we start with the vector $\mathbf{u}$, we will only need to repeat them

$O(1/\theta) = O(2^{n/2})$ to obtain a vector that, when measured, yields $|a\rangle$ with constant probability. For the sake of completeness, Figure 20.3 contains the full description of Grover's algorithm. ■

Grover's Search Algorithm.
Goal: Given a polynomial-time computable $f : \{0,1\}^n \to \{0,1\}$ with a unique $a \in \{0,1\}^n$ such that $f(a) = 1$, find a.
Quantum register: We use an $n + 1 + m$-qubit register, where m is large enough so we can compute the transformation $|x\sigma 0^m\rangle \mapsto |x(\sigma \oplus f(x))0^m\rangle$.

Operation	**State** (neglecting normalizing factors)
	Initial state: $\left|0^{n+m+1}\right\rangle$
Apply Hadamard operation to first n qubits.	$\mathbf{u}\left|0^{m+1}\right\rangle$ (where $\mathbf{u}$ denotes $\sum_{x\in\{0,1\}^n}|x\rangle$)
For $i = 1, \ldots, 2^{n/2}$ do:	$\mathbf{v}^i\left|0^{m+1}\right\rangle$ We let $\mathbf{v}^1 = \mathbf{u}$ and maintain the invariant that $\langle \mathbf{v}^i, |a\rangle\rangle = \sin(i\theta)$, where $\theta \sim 2^{-n/2}$ is such that $\langle \mathbf{u}, |a\rangle\rangle = \sin(\theta)$
Step 1: *Reflect around* $\mathbf{e} = \sum_{x\neq a}|x\rangle$:	
1.1 Compute $\left|x\sigma 0^{m+1}\right\rangle \mapsto \left|x(\sigma \oplus f(x))0^{m+1}\right\rangle$	$\sum_{x\neq a}\mathbf{v}^i_x|x\rangle\left|0^{m+1}\right\rangle + \mathbf{v}^i_a|a\rangle\left|10^{m+1}\right\rangle$
1.2 If $\sigma = 1$ then multiply vector by -1, otherwise do not do anything.	$\sum_{x\neq a}\mathbf{v}^i_x|x\rangle\left|0^{m+1}\right\rangle - \mathbf{v}^i_a|a\rangle\left|10^{m+1}\right\rangle$
1.3 Compute $\left|x\sigma 0^{m+1}\right\rangle \mapsto \left|x(\sigma \oplus f(x))0^{m+1}\right\rangle$.	$\mathbf{w}^i\left|0^{m+1}\right\rangle = \sum_{x\neq a}\mathbf{v}^i_x|x\rangle\left|0^{m+1}\right\rangle - \mathbf{v}^i_a|a\rangle|00^m\rangle$. ($\mathbf{w}^i$ is $\mathbf{v}^i$ reflected around $\sum_{x\neq a}|x\rangle$.)
Step 2: *Reflect around* $\mathbf{u}$:	
2.1 Apply Hadamard operation to first n qubits.	$\langle \mathbf{w}^i, \mathbf{u}\rangle|0^n\rangle\left|0^{m+1}\right\rangle + \sum_{x\neq 0^n}\alpha_x|x\rangle\left|0^{m+1}\right\rangle$, for some coefficients α_x's (given by $\alpha_x = \sum_z(-1)^{x\odot z}\mathbf{w}^i_z|z\rangle$).
2.2 Reflect around $\|0\rangle$:	
2.2.1 If first n-qubits are all zero then flip $n + 1^{st}$ qubit.	$\langle \mathbf{w}^i, \mathbf{u}\rangle|0^n\rangle|10^m\rangle + \sum_{x\neq 0^n}\alpha_x|x\rangle\left|0^{m+1}\right\rangle$
2.2.2 If $n + 1^{st}$ qubit is 1 then multiply by -1	$-\langle \mathbf{w}^i, \mathbf{u}\rangle|0^n\rangle|10^m\rangle + \sum_{x\neq 0^n}\alpha_x|x\rangle\left|0^{m+1}\right\rangle$
2.2.3 If first n-qubits are all zero then flip $n + 1^{st}$ qubit.	$-\langle \mathbf{w}^i, \mathbf{u}\rangle|0^n\rangle\left|0^{m+1}\right\rangle + \sum_{x\neq 0^n}\alpha_x|x\rangle\left|0^{m+1}\right\rangle$
2.3 Apply Hadamard operation to first n qubits.	$\mathbf{v}^{i+1}\left|0^{m+1}\right\rangle$ (where $\mathbf{v}^{i+1}$ is $\mathbf{w}^i$ reflected around $\mathbf{u}$)
Measure register and let a' be the obtained value in the first n qubits. If $f(a') = 1$ then output a'. Otherwise, repeat.	

Figure 20.3: Grover's Search Algorithm

20.6 Simon's Algorithm

Although beautiful, Grover's algorithm still has a significant drawback: it is merely quadratically faster than the best known classical algorithm for the same problem. In contrast, in this section we show *Simon's algorithm* that is a polynomial-time quantum algorithm solving a problem for which the best known classical algorithm takes *exponential* time.

Simon's algorithm solves the following problem: given a polynomial-time computable function $f : \{0,1\}^n \to \{0,1\}^n$ such that there exists $a \in \{0,1\}^*$ satisfying $f(x) = f(y)$ iff $x = y \oplus a$ for every $x, y \in \{0,1\}^n$, find this string a.

Two natural questions are (1) why is this problem interesting? and (2) why do we believe it is hard to solve for classical computers? The best answer to (1) is that, as we will see in Section 20.7, a generalization of Simon's problem turns out to be crucial in the quantum polynomial-time algorithm for famous *integer factorization* problem. Regarding (2), of course we do not know for certain that this problem does not have a classical polynomial-time algorithm (in particular, if $\mathbf{P} = \mathbf{NP}$ then there obviously exists such an algorithm). However, some intuition why it may be hard can be gleaned from the following *black box* model: suppose that you are given access to a black box (or oracle) that on input $x \in \{0,1\}^n$, returns the value $f(x)$. Would you be able to learn a by making at most a subexponential number of queries to the black box? It is not hard to see that if a is chosen at random from $\{0,1\}^n$ and f is chosen at random subject to the condition that $f(x) = f(y)$ iff $x = y \oplus a$ then no algorithm can successfully recover a with reasonable probability using significantly less than $2^{n/2}$ queries to the black box. Indeed, an algorithm using fewer queries is very likely to never get the same answer to two distinct queries, in which case it gets no information about the value of a.

20.6.1 The algorithm

Simon's algorithm is actually quite simple. It uses a register of $2n + m$ qubits, where m is the number of workspace bits needed to compute f. (Below we will ignore the last m qubits of the register, since they will be always set to all zeroes except in intermediate steps of f's computation.) The algorithm first uses n Hadamard operations to set the first n qubits to the uniform state and then apply the operation $|xz\rangle \mapsto |x(z \oplus f(x)\rangle$ to the register, resulting (up to normalization) in the state

$$\sum_{x\in\{0,1\}^n} |x\rangle\,|f(x)\rangle = \sum_{x\in\{0,1\}^n} (|x\rangle + |x \oplus a\rangle)\,|f(x)\rangle\,. \tag{1}$$

We then *measure* the second n bits of the register, collapsing its state to

$$|xf(x)\rangle + |(x \oplus a)f(x)\rangle \tag{2}$$

for some string x (that is chosen uniformly from $\{0,1\}^n$). You might think that we're done as the state (2) clearly encodes a, however we cannot directly learn a from this state: if we measure the first n bits we will get with probability $1/2$ the value x and with probability $1/2$ the value $x \oplus a$. Even though a can be deduced from these two values combined, each one of them on its own yields no information about a. (This point is well worth some contemplation, as it underlies the subtleties

involved in quantum computation and demonstrates why a quantum algorithm is *not* generally equivalent to performing exponentially many classical computation in parallel.)

However, consider now what happens if we perform another n Hadamard operations on the first n bits. Since this maps x to the vector $\sum_y (-1)^{x \odot y} |y\rangle$, the new state of the first n bits will be

$$\sum_y \left((-1)^{x \odot y} + (-1)^{(x \oplus a) \odot y} \right) |y\rangle = \sum_y \left((-1)^{x \odot y} + (-1)^{x \odot y} (-1)^{a \odot y} \right) |y\rangle . \quad (3)$$

For every $y \in \{0,1\}^m$, the y^{th} coefficient in the state (3) is nonzero if and only if if and only if $a \odot y = 0$, and in fact if measured, the state (3) yields a uniform $y \in \{0,1\}^n$ satisfying $a \odot y = 0$.

Repeating the entire process k times, we get k uniform strings $y_1, \ldots, y_k$ satisfying $y \odot a = 0$ or in other words, k *linear equations* (over the field GF(2)) on the variables $a_1, \ldots, a_n$. It can be easily shown that if, say, $k \geq 2n$ then with high probability there will be $n-1$ linearly independent equations among these (see Exercise 5), and hence we will be able to retrieve a from these equations using Gaussian elimination. For completeness, a full description of Simon's algorithm can be found in Figure 20.4.

Simon's Algorithm.
Goal: Given a polynomial-time computable $f : \{0,1\}^n \to \{0,1\}^n$ such that there is some $a \in \{0,1\}^n$ satisfying $f(x) = f(y)$ iff $y = x \oplus a$ for every $x, y \in \{0,1\}^n$, find a.
Quantum register: We use an $2n + m$-qubit register, where m is large enough so we can compute the transformation $|xz0^m\rangle \mapsto |x(z \oplus f(x))0^m\rangle$. (Below we ignore the last m qubits of the register as they will always contain 0^m except in intermediate computations of f.)

Operation	**State** (neglecting normalizing factors)
	Initial state: $\left\|0^{2n}\right\rangle$
Apply Hadamard operation to first n qubits.	$\sum_x \|x0^n\rangle$
Compute $\|xz\rangle \mapsto \|x(y \oplus f(x))\rangle$	$\sum_x \|xf(x)\rangle = \sum_x (\|x\rangle + \|x \oplus a\rangle) \|f(x)\rangle$
Measure second n bits of register.	$(\|x\rangle + \|x \oplus a\rangle) \|f(x)\rangle$
Apply Hadamard to first n bits.	$\left(\sum_y (-1)^{x \odot y} (1 + (-1)^{a \odot y}) \|y\rangle\right) \|f(x)\rangle$ $=$ $2 \sum_{y: a \odot y = 0} (-1)^{x \odot y} \|y\rangle \|f(x)\rangle$
Measure first n qubits of register to obtain a value y such that $y \odot a = 0$. Repeat until we get a sufficient number of linearly independent equations on a.	

Figure 20.4: Simon's Algorithm

20.7 Shor's algorithm: integer factorization using quantum computers.

The *integer factorization* problem is to find, given an integer N, the set of all *prime factors* of N (i.e., prime numbers that divide N). By a polynomial-time algorithm for this problem we mean an

algorithm that runs in time polynomial in the description of N, i.e., poly($\log(N)$) time. Although people have been thinking about the factorization problem in one form or another for at least 2000 years, we still do not know of a polynomial-time algorithm for it: the best classical algorithm takes roughly $2^{(\log N)^{1/3}}$ steps to factor N [**?**]. In fact, the presumed difficulty of this problem underlies many popular encryption schemes (such as RSA). Therefore, it was quite a surprise when in 1994 Peter Shor showed a quantum polynomial-time algorithm for this problem. To this day it remains the most famous algorithm for quantum computers, and the strongest evidence that **BQP** may contain problems outside of **BPP**.

The order-finding problem. Rather than showing an algorithm to factor a given number N, we will show an algorithm for a related problem: given a number A with $gcd(A, N) = 1$, find the *order* of A modulo N, defined to be the smallest positive integer r such that $A^r = 1 \pmod N$. Using elementary number theory, it is fairly straightforward to reduce the task of factoring N to solving this problem, and we defer the description of this reduction to Section 20.7.3.

REMARK 20.18
It is easy to see that for every positive integer k, if $A^k = 1 \pmod N$ then r divides k. (Indeed, otherwise if $k = cr + d$ for $c \in \mathbb{Z}$ and $d \in \{1, .., r-1\}$ then $A^d = 1 \pmod N$, contradicting the minimality of r.) Similarly, for every x, y it holds that $A^x = A^y \pmod N$ iff $x - y$ is a multiple of r. Therefore, the order finding problem can be defined as follows: given the function $f : \mathbb{N} \to \mathbb{N}$ that maps x to $A^x \pmod N$ and satisfies that $f(x) = f(y)$ iff $r|x - y$, find r. In this notation, the similarity to Simon's problem becomes more apparent.

20.7.1 Quantum Fourier Transform over $\mathbb{Z}_M$.

The main tool used to solve the order-finding problem is the *quantum Fourier transform*. We have already encountered the Fourier transform in Chapter 19, but will now use a different variant, which we call the *Fourier transform over* $\mathbb{Z}_M$ where $M = 2^m$ for some integer M. Recall that $\mathbb{Z}_M$ is the group of all number in $\{0, \ldots, M-1\}$ with the group operation being addition modulo M. The Fourier transform over this group, defined below, is a linear and in fact unitary operation from $\mathbb{C}^{2^m}$ to $\mathbb{C}^{2^m}$. The quantum Fourier transform is a way to perform this operation by composing $O(m^2)$ elementary quantum operations (operations that depend on at most three qubits). This means that we can transform a quantum system whose register is in state f to a system whose register is in the state corresponding to the Fourier transform $\hat{f}$ of f. This does *not* mean that we can compute in $O(m^2)$ the Fourier transform over $\mathbb{Z}_M$ - indeed this is not sufficient time to even write the output! Nonetheless, this transformation still turns out to be very useful, and is crucial to Shor's factoring algorithm in the same way that the Hadamard transformation (which is a Fourier transform over the group $\{0,1\}^n$ with the operation $\oplus$) was crucial to Simon's algorithm.

Definition of the Fourier transform over $\mathbb{Z}_M$.

Let $M = 2^m$ and let $\omega = e^{2\pi i/M}$. Note that $\omega^M = 1$ and $\omega^K \neq 1$ for every positive integer $K < N$ (we call such a number ω a primitive M^{th} root of unity). A function $\chi : \mathbb{Z}_M \to \mathbb{C}$ is called a *character* of $\mathbb{Z}_M$ if $\chi(y+z) = \chi(y)\chi(z)$ for every $y, z \in \mathbb{Z}_M$. $\mathbb{Z}_M$ has M characters $\{\chi_x\}_{x \in \mathbb{Z}_M}$ where

$\chi_x(y) = \omega^{xy}$. Let $\tilde{\chi}_x = \chi_x/\sqrt{M}$ (this factor is added for normalization), then the set $\{\tilde{\chi}_x\}_{x\in\mathbb{Z}_M}$ is an *orthonormal basis* of the space $\mathbb{C}^M$ since

$$\langle \tilde{\chi}_x, \tilde{\chi}_y \rangle = \tfrac{1}{M} \sum_{z=0}^{M-1} \omega^{xz}\overline{\omega^{yz}} = \tfrac{1}{M} \sum_{z=0}^{M-1} \omega^{(x-y)z}$$

which is equal to 1 if $x = y$ and to $\frac{1}{M}\frac{1-\omega^{(x-y)M}}{1-\omega^{x-y}} = 0$ if $x \neq y$ (the latter equality follows by the formula for the sum of a geometric series and the fact that $\omega^{\ell M} = 1$ for every ℓ).

Definition 20.19
For f a vector in $\mathbb{C}^M$, the *Fourier transform of* f is the representation of f in the basis $\{\tilde{\chi}_x\}$. We let $\hat{f}(x)$ denote the coefficient of $\tilde{\chi}_{-x}$ in this representation. Thus $f = \sum_{x=0}^{M-1} \hat{f}(x)\tilde{\chi}_{-x}$ and so $\hat{f}(x) = \langle f, \chi_{-x}\rangle = \frac{1}{\sqrt{M}}\sum_{y=0}^{M-1} \omega^{xy} f(x)$. We let $FT_M(f)$ denote the vector $(\hat{f}(0), \ldots, \hat{f}(M-1))$. The function FT_M is a unitary operation from $\mathbb{C}^M$ to $\mathbb{C}^M$ and is called the *Fourier transform over* $\mathbb{Z}^M$.

Fast Fourier Transform

Note that

$$\hat{f}(x) = \tfrac{1}{\sqrt{M}} \sum_{y\in\mathbb{Z}_M} f(y)\omega^{xy} = \tfrac{1}{\sqrt{M}} \sum_{y\in\mathbb{Z}_M, y \text{ even}} f(y)\omega^{-2x(y/2)} + \omega^x \tfrac{1}{\sqrt{M}} \sum_{y\in\mathbb{Z}_M, y \text{ odd}} f(y)\omega^{2x(y-1)/2}\,.$$

Now since ω^2 is an $M/2$th root of unity and $\omega^{M/2} = -1$, letting W be the $M/2$ diagonal matrix with diagonal $\omega^0, \ldots, \omega^{M/2-1}$, we get that

$$FT_M(f)_{low} = FT_{M/2}(f_{even}) + WFT_{M/2}(f_{odd}) \tag{4}$$
$$FT_M(f)_{high} = FT_{M/2}(f_{even}) - WFT_{M/2}(f_{odd}) \tag{5}$$

where for an M-dimensional vector $\mathbf{v}$, we denote by $\mathbf{v}_{even}$ (resp. $\mathbf{v}_{odd}$) the $M/2$-dimensional vector obtained by restricting $\mathbf{v}$ to the coordinates whose indices have least significant bit equal to 0 (resp. 1) and by $\mathbf{v}_{low}$ (resp. $\mathbf{v}_{high}$) the restriction of $\mathbf{v}$ to coordinates with most significant bit 0 (resp. 1).

Equations (4) and (5) are the crux of the well known *Fast Fourier Transform* (FFT) algorithm that computes the Fourier transform in $O(M \log M)$ (as opposed to the naive $O(M^2)$) time. We will use them for the *quantum* Fourier transform algorithm, obtaining the following lemma:

Lemma 20.20
There is an $O(m^2)$-step quantum algorithm that transforms a state $f = \sum_{x\in\mathbb{Z}_m} f(x)\,|x\rangle$ into the state $\hat{f} = \sum_{x\in\mathbb{Z}_m} \hat{f}(x)\,|x\rangle$, where $\hat{f}(x) = \frac{1}{\sqrt{M}}\sum_{y\in\mathbb{Z}_m} \omega^{xy} f(x)$.

Quantum Fourier transform: proof of Lemma 20.20

To prove Lemma 20.20, we use the following algorithm:

Quantum Fourier Transform FT_M **Initial state:** $f = \sum_{x\in\mathbb{Z}_M} f(x)\,|x\rangle$ **Final state:** $\hat{f} = \sum_{x\in\mathbb{Z}_M} \hat{f}(x)\,|x\rangle$.	
Operation	**State** (neglecting normalizing factors)
	$f = \sum_{x\in\mathbb{Z}_M} f(x)\,\|x\rangle$
Recursively run $FT_{M/2}$ on $m-1$ most significant qubits	$(FT_{M/2}f_{even})\,\|0\rangle + (FT_{M/2}f_{odd})\,\|1\rangle$
If LSB is 1 then compute W on $m-1$ most significant qubits (see below).	$(FT_{M/2}f_{even})\,\|0\rangle + (WFT_{M/2}f_{odd})\,\|1\rangle$
Apply Hadmard gate H to least significant qubit.	$(FT_{M/2}f_{even})(\|0\rangle + \|1\rangle) + (WWFT_{M/2}f_{odd})(\|0\rangle - \|1\rangle) =$ $(FT_{M/2}f_{even} + FT_{M/2}f_{odd})\,\|0\rangle +$ $(FT_{M/2}f_{even} - WFT_{M/2}f_{odd})\,\|1\rangle$
Move LSB to the most significant position	$\|0\rangle(FT_{M/2}f_{even} + FT_{M/2}f_{odd}) +$ $\|1\rangle(FT_{M/2}f_{even} - WFT_{M/2}f_{odd}) = \hat{f}$

The transformation W on $m-1$ qubits can be defined by $|x\rangle \mapsto \omega^x = \omega^{\sum_{i=0}^{m-2} 2^i x_i}$ (where x_i is the i^{th} qubit of x). It can be easily seen to be the result of applying for every $i \in \{0,\ldots,m-2\}$ the following elementary operation on the i^{th} qubit of the register: $|0\rangle \mapsto |0\rangle$ and $|1\rangle \mapsto \omega^{2^i}\,|1\rangle$.

The final state is equal to $\hat{f}$ by (4) and (5). (We leave verifying this and the running time to Exercise 9.) ■

20.7.2 The Order-Finding Algorithm.

We now present a quantum algorithm that on input a number $A < N$, finds the order of A modulo N (i.e., the smallest r such that $A^r = 1 \pmod N$). We let $m = 3\log N$ and $M = 2^m$. Our register will consist of $m + \log(N)$ qubits. Note that the function $x \mapsto A^x \pmod N$ can be computed in polylog(N) time (see Exercise 6) and so we will assume that we can compute the map $|x\rangle\,|y\rangle \mapsto |x\rangle\,|y \oplus \llcorner A^x \pmod N \lrcorner\rangle$ (where $\llcorner X \lrcorner$ denotes the representation of the number $X \in \{0,\ldots,N-1\}$ as a binary string of length $\log N$).[4] The order-finding algorithm is as follows:

[4] To compute this map we may need to extend the register by some additional qubits, but we can ignore them as they will always be equal to zero except in intermediate computations.

Order finding algorithm.
Goal: Given numbers N and $A < N$ such that $gcd(A, N) = 1$, find the smallest r such that $A^r = 1 \pmod N$.
Quantum register: We use an $m + n$-qubit register, where $m = 3 \log N$ (and hence in particular $M \geq N^3$). Below we treat the first m bits of the register as encoding a number in $\mathbb{Z}_M$.

Operation	**State** (*including* normalizing factors)
Apply Fourier transform to the first m bits.	$\frac{1}{\sqrt{M}} \sum_{x \in \mathbb{Z}_M} \|x\rangle \|0^n\rangle$
Compute the transformation $\|x\rangle \|y\rangle \mapsto \|x\rangle \|y \oplus (A^x \pmod N)\rangle$.	$\sum_{x \in \mathbb{Z}_M} \|x\rangle \|A^x \pmod N\rangle$
Measure the second register to get a value y_0.	$\frac{1}{\sqrt{\lceil M/r \rceil}} \sum_{\ell=0}^{\lceil M/r \rceil - 1} \|x_0 + \ell r\rangle \|y_0\rangle$ where x_0 is the smallest number such that $A^{x_0} = y_0 \pmod N$.
Apply the Fourier transform to the first register.	$\frac{1}{\sqrt{M}\sqrt{\lceil M/r \rceil}} \left(\sum_{x \in \mathbb{Z}_n} \sum_{\ell=0}^{\lceil M/r \rceil - 1} \omega^{(x_0 + \ell r)x} \|x\rangle \right) \|y_0\rangle$
Measure the first register to obtain a number $x \in \mathbb{Z}_M$. Find the best rational approximation a/b (with a, b coprime) for the fraction $\frac{x}{M}$ with denominator b at most $40M$ (see Section 20.A). If $A^b = A \pmod M$ then output b.	

In the analysis, it will suffice to show that this algorithm outputs the order r with probability at least $1/\text{poly}(\log(N))$ (we can always amplify the algorithm's success by running it several times and taking the smallest output).

Analysis: the case that $r|M$

We start by analyzing the algorithm in the (rather unrealistic) case that $M = rc$ for some integer c. In this case we claim that the value x measured will be equal to $c'c$ for random $c' \in 0, \ldots, r$. In this case, $x/M = c'/r$. However, with probability at least $\Omega(1/\log(r))$, the number c' will be prime (and in particular coprime to r). In this case, the denominator of the rational approximation for x/M is indeed equal to r.

Indeed, for every $x \in \mathbb{Z}_M$, the absolute value of $|x\rangle$'s coefficient before the measurement is equal (up to some normalization factor) to

$$\left| \sum_{\ell=0}^{c-1} \omega^{(x_0 + \ell r)x} \right| = \left| \omega^{x_0 c' c} \right| \left| \sum_{\ell=0}^{c-1} \omega^{r \ell x} \right| = 1 \cdot \left| \sum_{\ell=0}^{c-1} \omega^{r \ell x} \right| . \tag{6}$$

But if $x = cc'$ then $\omega^{r\ell cc'} = \omega^{Mc'} = 1$, and hence the coefficients of all such x's are equal to the same positive number. On the other hand, if c does not divide x then then since ω^r is a c^{th} root of unity, $\sum_{\ell=0}^{c-1} w^{r\ell x} = 0$ by the formula for sums of geometric progressions. Thus, such a number x would be measured with zero probability.

The case that $r \not| M$

In the general case, we will not be able to show that the value x measured satisfies $M|xr$. However, we will show that with $\Omega(1/\log r)$ probability, **(1)** xr will be "almost divisible" by M in the sense

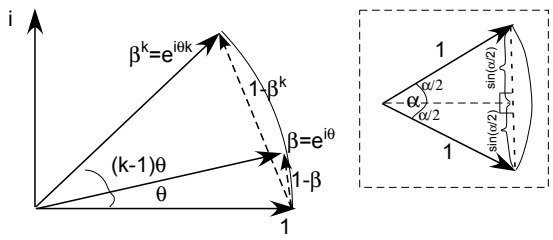


Figure 20.5: A complex number $z = a + ib$ can be thought of as the two-dimensional vector (a, b) of length $|z| = \sqrt{a^2 + b^2}$. The number $\beta = e^{i\theta}$ corresponds to a unit vector of angle θ from the x axis.For any such β, if k is not too large (say $k < 1/\theta$) then by elementary geometric considerations $\frac{|1-\beta^k|}{|1-\beta|} = \frac{2\sin(\theta/2)}{2\sin(k\theta/2)}$. We use here the fact (proved in the boxed figure) that in a unit cycle, the chord corresponding to an angle α is of length $2\sin(\alpha/2)$.

that $0 \leq xr \pmod M < r/10$ and **(2)** $\lceil xr/M \rceil$ is coprime to r. Condition **(1)** implies that $|xr - cM| < r/10$ for $c = \lceil xr/M \rceil$. Dividing by rM gives $\left|\frac{x}{M} - \frac{c}{r}\right| < \frac{1}{10M}$. Therefore, $\frac{c}{r}$ is a rational number with denominator at most N that approximates $\frac{x}{M}$ to within $1/(10M) < 1/(2N^2)$. It is not hard to see that such an approximation is unique (Exercise 7) and hence in this case the algorithm will come up with c/r and output the denominator r.

Thus all that is left is to prove the following two lemmas:

LEMMA 20.21
There exist $\Omega(r/\log r)$ values $x \in \mathbb{Z}_M$ such that:

1. $0 \leq xr \pmod M < r/10$
2. $\lceil xr/M \rceil$ *and* r *are coprime*

LEMMA 20.22
If x satisfies $0 \leq xr \pmod M < r/10$ then, before the measurement in the final step of the order-finding algorithm, the coefficient of $|x\rangle$ is at least $\Omega(\frac{1}{\sqrt{r}})$.

PROOF OF LEMMA 20.21: We prove the lemma for the case that r is coprime to M, leaving the general case as Exercise 10. In this case, the map $x \mapsto rx \pmod M$ is a permutation of $\mathbb{Z}_M^*$ and we have a set of at least $r/(20\log r)$ x's such that $xr \pmod M$ is a prime number p between 0 and $r/10$. For every such x, $xr + \lceil r/M \rceil M = p$ which means that $\lceil r/M \rceil$ can not have a nontrivial shared factor with r, as otherwise this factor would be shared with p as well. ■

PROOF OF LEMMA 20.22: Let x be such that $0 \leq xr \pmod M < r/10$. The absolute value of $|x\rangle$'s coefficient in the state before the measurement is

$$\frac{1}{\sqrt{\lceil M/r \rceil}\sqrt{M}} \left| \sum_{\ell=0}^{\lceil M/r \rceil - 1} \omega^{\ell r x} \right| . \tag{7}$$

Setting $\beta = \omega^{rx}$ (note that since $M \not| rx$, $\beta \neq 1$) and using the formula for the sum of a geometric series, this is at least

$$\frac{\sqrt{r}}{2M} \left| \frac{1-\beta^{\lceil M/r \rceil}}{1-\beta} \right| = \frac{\sqrt{r}}{2M} \frac{\sin(\theta \lceil M/r \rceil/2)}{\sin(\theta/2)} , \tag{8}$$

where $\theta = \frac{rx \pmod M}{M}$ is the angle such that $\beta = e^{i\theta}$ (see Figure 20.5 for a proof by picture of the last equality). Under our assumptions $\lceil M/r \rceil \theta < 1/10$ and hence (using the fact that $\sin\alpha \sim \alpha$ for small angles α), the coefficient of x is at least $\frac{\sqrt{r}}{4M}\lceil M/r \rceil \geq \frac{1}{8\sqrt{r}}$ ■

20.7.3 Reducing factoring to order finding.

The reduction of the factoring problem to the order-finding problem follows immediately from the following two Lemmas:

LEMMA 20.23
For every nonprime N, the probability that a random X in the set $\mathbb{Z}_N^* = \{X \in [N-1] : gcd(X,N) = 1\}$ has an even order r and furthermore, $X^{r/2} \neq +1 \pmod N$ and $X \neq -1 \pmod N$ is at least 1/4.

LEMMA 20.24
For every N and Y, if $Y^2 = 1 \pmod N$ but $Y \neq +1 \pmod N$ and $Y \neq -1 \pmod N$, then $gcd(Y-1,N) > 1$.

Together, Lemmas 20.23 and 20.24 show that the following algorithm will output a prime factor P of N with high probability: (once we have a single prime factor P, we can run the algorithm again on N/P)

1. Choose X at random from $[N-1]$.
2. If $gcd(X,N) > 1$ then let $K = gcd(X,N)$, otherwise compute the order r of X, and if r is even let $K = gcd(X^{r/2}-1, N)$.
3. If $K \in \{1, N\}$ then go back to Step 1. If K is a prime then output K and halt. Otherwise, use recursion to output a factor of K.

Note that if $T(N)$ is the running time of the algorithm then it satisfies the equation $T(N) \leq T(N/2) + \text{polylog}(N)$ leading to $\text{polylog}(N)$ running time.

PROOF OF LEMMA 20.24: Under our assumptions, N divides $Y^2 - 1 = (Y-1)(X+1)$ but does not divide neither $Y-1$ or $Y+1$. But this means that $gcd(Y-1,N) > 1Z$ since if $Y-1$ and N were coprime, then since N divides $(Y-1)(Y+1)$, it would have to divide $X+1$ (Exercise 8). ■

PROOF OF LEMMA 20.23: We prove this for the case that $N = PQ$ for two primes P, Q (the proof for the general case is similar and is left as Exercise **??**). In this case, by the Chinese Reminder Theorem, if we map every number $X \in \mathbb{Z}_N^*$ to the pair $\langle X \pmod P, X \pmod Q\rangle$ then this map is one-to-one. Also, the groups $\mathbb{Z}_P^*$ and $\mathbb{Z}_Q^*$ are known to by *cyclic* which means that there is a number $g \in [P-1]$ such that the map $j \mapsto g^j \pmod P$ is a permutation of $[P-1]$ and similarly there is a number $h \in [Q-1]$ such that the map $k \mapsto h^k \pmod P$ is a permutation of $[Q-1]$.

This means that instead of choosing X at random, we can think of choosing two numbers j, k at random from $[P-1]$ and $[Q-1]$ respectively and consider the pair $\langle g^j \pmod P, h^k \pmod Q\rangle$ which is in one-to-one correspondence with the set of X's in $\mathbb{Z}_N^*$. The order of this pair (or equivalently, of X) is the smallest positive integer r such that $g^{jr} = 1 \pmod P$ and $h^{kr} = 1 \pmod Q$, which

means that $P-1|jr$ and $Q-1|kr$. Now suppose that j is odd and k is even (this happens with probability 1/4). In this case r is of the form $2r'$ where r' is the smallest number such that $P-1|2jr'$ and $Q-1|kr'$ (the latter holds since we can divide the two even numbers k and $Q-1$ by two). But this means that $g^{j(r/2)} \neq 1 \pmod{Q}$ and $h^{k(r/2)} = 1 \pmod{Q}$. In other words, if we let X be the number corresponding to $\langle g^j \pmod{P}, h^k \pmod{Q} \rangle$ then $X^{r/2}$ corresponds to a pair of the form $\langle a, 1 \rangle$ where $a \neq 1$. However, since $+1 \pmod{N}$ corresponds to the pair $\langle +1, +1 \rangle$ and $-1 \pmod{N}$ corresponds to the pair $\langle -1 \pmod{P}, -1 \pmod{Q} \rangle$ it follows that $X^{r/2} \neq \pm 1 \pmod{N}$. ■

20.8 BQP and classical complexity classes

What is the relation between **BQP** and the classes we already encountered such as **P**, **BPP** and **NP**? This is very much an open questions. It not hard to show that quantum computers are at least not infinitely powerful compared to classical algorithms:

THEOREM 20.25
BQP $\subseteq$ **PSPACE**

PROOF SKETCH: To simulate a T-step quantum computation on an m bit register, we need to come up with a procedure `Coeff` that for every $i \in [T]$ and $x \in \{0,1\}^m$, the x^{th} coefficient (up to some accuracy) of the register's state in the i^{th} execution. We can compute `Coeff` on inputs x, i using at most 8 recursive calls to `Coeff` on inputs $x', i-1$ (for the at most 8 strings that agree with x on the three bits that the F_i's operation reads and modifies). Since we can reuse the space used by the recursive operations, if we let $S(i)$ denote the space needed to compute $\texttt{Coeff}(x,i)$ then $S(i) \leq S(i-1) + O(\ell)$ (where ℓ is the number of bits used to store each coefficient).

To compute, say, the probability that if measured after the final step the first bit of the register is equal to 1, just compute the sum of $\texttt{Coeff}(x,T)$ for every $x \in \{0,1\}^n$. Again, by reusing the space of each computation this can be done using polynomial space. ■

Theorem 20.25 can be improved to show that **BQP** $\subseteq \mathbf{P}^{\#\mathbf{P}}$ (where #**P** is the counting version of **NP** described in Chapter 9), but this is currently the best upper bound we know on **BQP**.

Does **BQP** = **BPP**? The main reason to believe this is false is the polynomial-time algorithm for integer factorization. Although this is not as strong as the evidence for, say **NP** $\not\subseteq$ **BPP** (after all **NP** contains thousands of well-studied problems that have resisted efficient algorithms), the factorization problem is one of the oldest and most well-studied computational problems, and the fact that we still know no efficient algorithm for it makes the conjecture that none exists appealing. Also note that unlike other famous problems that eventually found an algorithm (e.g., linear programming [**?**] and primality testing [**?**]), we do not even have a heuristic algorithm that is conjectured to work (even without proof) or experimentally works on, say, numbers that are product of two random large primes.

What is the relation between **BQP** and **NP**? It seems that quantum computers only offer a quadratic speedup (using Grover's search) on **NP**-complete problems, and so most researchers believe that **NP** $\not\subseteq$ **BPP**. On the other hand, there is a problem in **BQP** (the Recursive Fourier Sampling or RFS problem [**?**]) that is not known to be in the polynomial-hierarchy , and so at the moment we do not know that **BQP** = **BPP** even if we were given a polynomial-time algorithm for SAT.

Chapter notes and history

We did not include treatment of many fascinating aspects of quantum information and computation. Many of these are covered in the book by Nielsen and Chuang [**?**]. See also Umesh Vazirani's excellent lecture notes on the topic (available from his home page).

One such area is *quantum error correction*, that tackles the following important issue: how can we run a quantum algorithm when at every possible step there is a probability of noise interfering with the computation? It turns out that under reasonable noise models, one can prove the following *threshold theorem*: as long as the probability of noise at a single step is lower than some constant threshold, one can perform arbitrarily long computations and get the correct answer with high probability [**?**].

Quantum computing has a complicated but interesting relation to cryptography. Although Shor's algorithm and its variants break many of the well known public key cryptosystems (those based on the hardness of integer factorization and discrete log), the features of quantum mechanics can actually be used for cryptographic purposes, a research area called *quantum cryptography* (see [**?**]). Shor's algorithm also spurred research on basing public key encryption scheme on other computational problems (as far as we know, quantum computers do not make the task of breaking most known *private key* cryptosystems significantly easier). Perhaps the most promising direction is basing such schemes on certain problems on integer lattices (see the book [**?**] and [**?**]).

While quantum mechanics has had fantastic success in predicting experiments, some would require more from a physical theory. Namely, to tell us what is the "actual reality" of our world. Many physicists are understandably uncomfortable with the description of nature as maintaining a huge array of possible states, and changing its behavior when it is observed. The popular science book [**?**] contains a good (even if a bit biased) review of physicists' and philosophers' attempts at providing more palatable descriptions that still manage to predict experiments.

On a more technical level, while no one doubts that quantum effects exist at microscopic scales, scientists questioned why they do not manifest themselves at the macrosopic level (or at least not to human consciousness). A *Scientific American* article by Yam [**?**] describes various explanations that have been advanced over the years. The leading theory is *decoherence*, which tries to use quantum theory to explain the absence of macroscopic quantum effects. Researchers are not completely comfortable with this explanation. The issue is undoubtedly important to quantum computing, which requires hundreds of thousands of particles to stay in quantum superposition for large-ish periods of time. Thus far it is an open question whether this is practically achievable. One theoretical idea is to treat decoherence as a form of noise, and to build noise-tolerance into the computation —a nontrivial process. For details of this and many other topics, see the books by Kitaev, Shen, and Vyalyi [**?**].

The original motivation for quantum computing was to construct computers that are able to simulate quantum mechanical systems, and this still might be their most important application if they are ever built. Feynman [**?**] was the first to suggest the possibility that quantum mechanics might allow Turing Machines more computational power than classical TMs. In 1985 Deutsch [**?**] defined a quantum Turing machine, though in retrospect his definition is unsatisfactory. Better definitions then appeared in Deutsch-Josza [**?**], Bernstein-Vazirani [**?**] and Yao [**?**], at which point quantum computation was firmly established as a field.

Exercises

§1 Prove Claim 20.11.

Hint: First prove that Condition 3 holds iff Condition 1 holds iff Condition 4 holds. This follows almost directly from the definition of the inner product and the fact that for every matrices A, B it holds that $(AB)^* = B^*A^*$ and $(A^*)^* = A$. Then prove that Condition 3 implies Condition 2, which follows from the fact that the norm is invariant under a change of basis. Finally, prove that Condition 2 implies Condition 3 by showing that if two orthogonal unit vectors $\mathbf{v}, \mathbf{u}$ are mapped to non-orthogonal unit vectors $\mathbf{v}', \mathbf{u}'$, then the norm of the vector $\mathbf{u} + \mathbf{v}$ is not preserved.

§2 For each one of the following operations: Hadamard, NOT, controlled-NOT, rotation by $\pi/4$, and Tofolli, write down the 8×8 matrix that describes the mapping induced by applying this operation on the first qubits of a 3-qubit register.

§3 Suppose that a two-bit quantum register is in an arbitrary state $\mathbf{v}$. Show that the following three experiments will yield the same probability of output:

(a) Measure the register and output the result.

(b) First measure the first bit and output it, then measure the second bit and output it.

(c) First measure the second bit and output it, then measure the first bit and output it.

§4 Suppose that f is computed in T time by a quantum algorithm that uses a partial measurements in the middle of the computation, and then proceeds differently according to the result of that measurement. Show that f is computable by $O(T)$ elementary operations.

§5 Prove that if for some $a \in \{0,1\}^n$, the strings $y_1, \ldots, y_{n-1}$ are chosen uniformly at random from $\{0,1\}^n$ subject to $y_i \odot a = 0$ for every $i \in [n-1]$, then with probability at least $1/10$, there exists no nonzero string $a' \neq a$ such that $y_i \odot a' = 0$ for every $i \in [n-1]$. (In other words, the vectors $y_1, \ldots, y_{n-1}$ are linearly independent.)

§6 Prove that given $A, x \in \{0, \ldots, M-1\}$, we can compute $A^x \pmod M$ in time polynomial in $\log M$.

Hint: Start by solving the case that $x = 2^k$ for some k. Then, show an algorithm for general x by using x's binary expansion.

§7 Prove that for every $\alpha < 1$, there is at most a single rational number a/b such that $b < N$ and $|\alpha - a/b| < 1/(2N^2)$.

§8 Prove that if A, B are numbers such that N and A are coprime but N divides AB, then N divides B.

Hint: Use the fact that if N and A are co-prime then there are whole numbers α, β such that $\alpha N + \beta A = 1$ and multiply this equation by B.

§9 Prove Lemma 20.20.

§10 Complete the proof of Lemma **??** for the case that r and M are not coprime. That is, prove that also in this case there exist at least $\Omega(r/\log r)$ values x's such that $0 \leq rx \pmod{M} \leq r/2$ and $\lceil M/x \rceil$ and r are coprime.

Hint: let $d = gcd(r, M)$, $r' = r/d$ and $M' = M/d$. Now use the same argument as in the case that M and r are coprime to argue that there exist $\Omega(\frac{r}{d \log r})$ values $x \in \mathbb{Z}_{M'}$ satisfying this condition, and that if x satisfies it then so does $x + cM$ for every c.

§11 (Uses knowledge of continued fractions) Suppose $j, r \leq N$ are mutually coprime and unknown to us. Show that if we know the first $2 \log N$ bits of j/r then we can recover j, r in polynomial time.

20.A Rational approximation of real numbers

A *continued fraction* is a number of the following form:

$$\alpha_1 + \cfrac{1}{\alpha_2 + \cfrac{1}{\alpha_2 + \cfrac{1}{\alpha_3 + \ldots}}}$$

Given a real number $\alpha > 0$, we can find its representation as an infinite fraction as follows: split α into the integer part $\lfloor \alpha \rfloor$ and fractional part $\alpha - \lfloor \alpha \rfloor$, find recursively the representation R of $1/(\alpha - \lfloor \alpha \rfloor)$, and then write

$$\alpha = \lfloor \alpha \rfloor + \frac{1}{R}\,.$$

If we stop after ℓ steps, we get a rational number that can be represented as a/b with a, b coprime. It can be verified that $b \in [2^{\ell/2}, 2^{2\ell}]$. The following theorem is also known:

THEOREM 20.26
*[**?**] If a/b is a rational number obtained by running the continued fraction algorithm on α for a finite number of steps then $|\alpha - a/b| > |\alpha - c/d$ for every rational number c/d with denominator $d \leq b$.*

This means that given any number α and bound N, we can use the continued fraction algorithm to find in polylog(N) steps a rational number a/b such that $b \leq 16N$ and a/b approximates α better than any other rational number with denominator at most b.