

Introduction to Quantum Computing

Lecture Notes

Daniel Grier

Contents

1	Foundations of Quantum Mechanics	3
1.1	The basics of quantum computation	3
1.2	Multi-qubit quantum computation	5
1.3	Dirac notation and inner products	8
1.4	Mixed states	10
1.5	Noteworthy quantum phenomena	15
2	Computation with Quantum Circuits	16
2.1	Introduction to quantum circuits	16
2.2	Universality, approximations, and circuit size	20
2.3	Principle of Deferred Measurement	21
3	Query Complexity	22
3.1	Defining a quantum oracle	22
3.2	Fourier sampling problems	23
3.3	Hidden subgroup problems	27
3.4	Grover's algorithm and the unstructured search problem	30
4	Shor's algorithm	39
4.1	Representing integers with qubits	39
4.2	Outline of Shor's algorithm	41
4.3	Simon's problem over the integers	42
4.4	Order finding: It's not just a toy problem any more	45

<i>CONTENTS</i>	2
Bibliography	52
A Complex numbers	53
A.1 Complex numbers as 2d vectors	53
A.2 Complex conjugate	54
A.3 Roots of unity	54

Chapter 1

Foundations of Quantum Mechanics

Before we can reason about the power of quantum computers, we must obviously first understand what kinds of computations they unlock. We will start with the foundations: What is a quantum state, and what kinds of operations can you perform on that state? As a warning, this exposition will be almost entirely mathematical, leaving the thorny physical questions about implementing such quantum systems to another course.

1.1 The basics of quantum computation

To describe a quantum state, let's start by analogy to one of the simplest classical objects—a two-sided coin. Let's call the two sides of the coin the 0-outcome and the 1-outcome (perhaps more traditionally these two outcomes would be called “heads” and “tails”).

To be even more concrete, let's suppose the coin is biased so that it lands on the 0-outcome with 30% probability and on the 1-outcome with 70% probability. Suppose we flip the coin in the air, and we want to describe the probability distribution over outcomes when the coin lands. We could represent it by the length-2 vector:

$$\begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix} \leftarrow \begin{array}{l} \text{Probability of 0-outcome} \\ \text{Probability of 1-outcome} \end{array}$$

In some sense, this represents the “state” of the coin if we know the coin has landed on one side or the other, but we have not yet looked at which outcome.

If we *were* to look at the outcome, then the state of the system immediately changes to whichever outcome we saw:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \leftarrow \begin{array}{l} \text{0-outcome} \\ \text{with certainty} \end{array} \quad \text{or} \quad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \leftarrow \begin{array}{l} \text{1-outcome} \\ \text{with certainty} \end{array}$$

since there is no ambiguity in the outcome once we've observed it.

Stepping back a bit, let's look at the full description of states and operations in this classical probability framework. First, notice that instead of a coin with just 2 outcomes, we could have as many outcomes as we like (think of rolling a biased die). In a system with d outcomes, the state of the system would be described by a vector of d probabilities. The key property of this vector is that each probability is non-negative and all probabilities sum up to 1.

The set of operations that we could perform on this system is the set of operations that take probability vectors to probability vectors. Specifically (and we will see how this changes in the quantum setting soon), these operations preserve the ℓ_1 -norm of the vector, where the ℓ_1 -norm of vector $v = (v_1, v_2, \dots, v_d) \in \mathbb{C}^d$ is defined as

$$\|v\|_1 := \sum_{i=1}^d |v_i|$$

1.1.1 Qubits

Let's now complete the analogy of the classical probabilistic bit discussed above with the quantum variant called a *qubit*. Instead of assigning two outcomes (0 and 1) a probability, we instead assign them a complex number called an *amplitude*. We represent a qubit as a column vector in $\mathbb{C}^2$. For example,

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{pmatrix} \leftarrow \begin{array}{l} \text{Amplitude on 0-outcome} \\ \text{Amplitude on 1-outcome} \end{array}$$

Let's now discuss what it means to “look” at a quantum state, which is called *measurement* in the quantum setting. The measurement axiom of quantum mechanics, called the *Born rule*, says that you see a particular outcome with probability equal to the squared magnitude of its amplitude. For the example above, this means we'd see the 0-outcome with probability $|1/\sqrt{2}|^2 = 1/2$ and the outcome will be 1 with probability $|i/\sqrt{2}|^2 = 1/2$. Once again, when you observe this outcome the qubit *collapses* to whichever outcome you observed.

From the Born rule, we can derive a condition on the amplitudes of a qubit. Suppose we have a qubit with amplitudes $\alpha, \beta \in \mathbb{C}$. The Born rule states that we see the 0 and 1 outcomes with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. Since there are only two outcomes, these two probabilities must sum up to 1 (i.e., we must see either the 0 or 1 outcome when we measure). We arrive at the following condition for the amplitudes of a qubit: $|\alpha|^2 + |\beta|^2 = 1$.

Stepping back again, let's give a complete mathematical description of a quantum state. We can generalize to a quantum state with d outcomes (called a *qudit* for $d > 2$), which is represented by a length- d complex vector. The key property of this vector is that the squared magnitudes of the amplitudes sum to 1. In other words, the ℓ_2 -norm of the vector is 1. The ℓ_2 -norm of any $v = (v_1, v_2, \dots, v_d) \in \mathbb{C}^d$ is defined as

$$\|v\|_2 := \sqrt{\sum_{i=1}^d |v_i|^2}.$$

It is an amazing fact that moving from the classical to the quantum setting is in some sense just moving from the ℓ_1 to the ℓ_2 norm.

1.1.2 Unitary matrices

Because the set of valid quantum states must have unit ℓ_2 -norm, the set of viable quantum operations must preserve the ℓ_2 -norm of the state. However, not all such operations are valid. An axiom of quantum mechanics dictates that quantum operations must also be *linear*. We

will see a slight generalization of this later, but for now, you can think of this linearity as implying that quantum operations are matrices. Applying a quantum operation to a quantum state simply means multiplying the vector of the state with the matrix of the quantum operation.

Matrices preserving the ℓ_2 -norm have a beautiful characterization—namely, they are the *unitary* matrices, i.e., matrices $U \in \mathbb{C}^{d \times d}$ such that $UU^\dagger = I$. Here, “ $\dagger$ ” is the conjugate transpose operation and “ I ” is the identity matrix.

1.2 Multi-qubit quantum computation

In general, we think of large classical computations as a sequence of operations on some bit string. In this way we can break up some large complex operation into a sequence of simpler operations. The number of operations required to build the more complex operation is a proxy for how complex that operation really is. Similarly, in quantum systems, we want to build up larger more complex operations from simpler ones. To do this, we first need to understand what a quantum system consisting of multiple qubits looks like, so that we can understand what it means to locally apply some quantum operation.

1.2.1 Tensor product of states

Once again, let’s start with a discussion of multiple classical random bits, and see how it generalizes to qubits. Let A, B be two random bits. Each bit has some probability of being in the 0 or 1 outcome. Together, the two bits give rise to a probability distribution over pairs of outcomes (i.e., 00, 01, 10, and 11). We can derive the probability of a particular pair of outcomes by multiplying the probabilities of the individual outcome for each bit. For example, let

$$A = \begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix} \begin{matrix} \leftarrow 0 \\ \leftarrow 1 \end{matrix}, \quad B = \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} \begin{matrix} \leftarrow 0 \\ \leftarrow 1 \end{matrix}$$

Then the product distribution associated to A and B together gives rise to the vector

$$AB = \begin{pmatrix} 0.18 \\ 0.12 \\ 0.42 \\ 0.28 \end{pmatrix} \begin{matrix} \leftarrow 00 \\ \leftarrow 01 \\ \leftarrow 10 \\ \leftarrow 11 \end{matrix}$$

Combining two separate qubits into a single system is exactly the same. Let $v, w \in \mathbb{C}^2$ be vectors representing two qubits. The vector of the joint system is called the *tensor product* $v \otimes w$ of the two vectors v and w . The tensor product operation yields the vector containing all products of amplitudes. The example looks identical to the classical setting:

$$v = \begin{pmatrix} \sqrt{0.3} \\ \sqrt{0.7} \end{pmatrix} \begin{matrix} \leftarrow 0 \\ \leftarrow 1 \end{matrix}, \quad w = \begin{pmatrix} \sqrt{0.6} \\ \sqrt{0.4} \end{pmatrix} \begin{matrix} \leftarrow 0 \\ \leftarrow 1 \end{matrix}$$

and

$$v \otimes w = \begin{pmatrix} \sqrt{0.18} \\ \sqrt{0.12} \\ \sqrt{0.42} \\ \sqrt{0.28} \end{pmatrix} \begin{matrix} \leftarrow 00 \\ \leftarrow 01 \\ \leftarrow 10 \\ \leftarrow 11 \end{matrix}$$

Formally, the tensor product operation $\otimes$ is defined over any pair of vectors $v \in \mathbb{C}^a$ and $w \in \mathbb{C}^b$ (not necessarily of the same length) as

$$v \otimes w := \begin{pmatrix} v_1 w \\ \vdots \\ v_a w \end{pmatrix} = \begin{pmatrix} v_1 w_1 \\ \vdots \\ v_1 w_b \\ v_2 w_1 \\ \vdots \\ v_a w_b \end{pmatrix}$$

From this definition, one can derive the following properties of the tensor product, which hold for all complex vectors v, w, z and scalars $\alpha, \beta \in \mathbb{C}$:

$$\text{Scalar multiplication: } (\alpha v) \otimes (\beta w) = (\alpha\beta)(v \otimes w)$$

$$\text{Associativity: } (v \otimes w) \otimes z = v \otimes (w \otimes z)$$

$$\text{Distributivity: } v \otimes (w + z) = v \otimes w + v \otimes z$$

We have that the tensor product of two qubits is represented by a length-4 complex vector, the tensor product of three qubits is represented by a length-8 vector, and so on. One of the key questions we will ask in these notes is: how much of this exponentiality is really there? Of course, when it comes to quantum states constructed from tensor products of qubits, the answer is... not much. To describe such a state, we simply need the 2 amplitudes for each individual qubit, a total of $2n$ amplitudes for an n -qubit state, rather than the 2^n amplitudes in the tensor product vector.

Critically, however, not all quantum states over qubits can be described in this way. That is, we can start with a tensor product of single-qubit quantum states, apply a sequence of quantum operations, and arrive at a state which cannot be described by any tensor product of single-qubit states. Such states are called *entangled*.

Our first example of an entangled 2-qubit state is the following:

$$\begin{pmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 1/\sqrt{2} \end{pmatrix}$$

which is known (amongst other names) as the *Bell state*. Before we prove this state is entangled, let's take a moment to consider what would happen if we measured this state. We would see the 00 outcome with probability $1/2$ and the 11 outcome with probability $1/2$. In other words, if we made the measurement and we saw that the first qubit was 0, we would immediately know the second qubit was also 0. This description gets even stranger when we consider the possibility that we could dramatically separate the first and second qubits, putting each on either end of the galaxy (hard to do in practice, of course!). Measuring at one end of the galaxy immediately tells us the outcome of the qubit at the other end.¹

¹A significant amount of ink has been spilled on exactly what is happening at the physical level when a measurement like this is made. Look up the “quantum measurement problem”. Thankfully for one of the most cherished physical laws, this entanglement phenomenon does *not* allow for faster-than-light communication.

To prove the Bell state is entangled, we argue by contradiction. Suppose otherwise, then we would have

$$\begin{pmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 1/\sqrt{2} \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \otimes \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{pmatrix}$$

for some complex amplitudes $\alpha_0, \alpha_1, \beta_0, \beta_1$. Comparing the left and right equations, we get the constraints:

$$\frac{1}{\sqrt{2}} = \alpha_0\beta_0, \quad 0 = \alpha_0\beta_1, \quad 0 = \alpha_1\beta_0, \quad \frac{1}{\sqrt{2}} = \alpha_1\beta_1.$$

One can check this system of equations has no feasible solution, and therefore, the Bell state must be entangled.

1.2.2 Tensor product of matrices

The tensor product of matrices is the unique operator which respects the tensor product of the underlying states. That is, for unitaries $U \in \mathbb{C}^{a \times a}$ and $V \in \mathbb{C}^{b \times b}$, the tensor product unitary $U \otimes V$ is the unique linear operator such that

$$(U \otimes V)(v \otimes w) = (Uv) \otimes (Vw)$$

for all states $v \in \mathbb{C}^a$ and $w \in \mathbb{C}^b$. This definition lines up with our intuition that if we apply a unitary to a specific qubit, then it should not affect any other qubit.

Formally, one can give a (rather more cumbersome) definition of the tensor product of arbitrary matrices $U \in \mathbb{C}^{a \times a}$ and $V \in \mathbb{C}^{b \times b}$ as:

$$U \otimes V = \begin{pmatrix} u_{11}V & u_{12}V & \cdots & u_{1a}V \\ u_{21}V & u_{22}V & \cdots & u_{2a}V \\ \vdots & \vdots & \ddots & \vdots \\ u_{a1}V & u_{a2}V & \cdots & u_{aa}V \end{pmatrix}.$$

Written out somewhat more explicitly when $a = b = 2$, we have

$$U \otimes V = \begin{pmatrix} u_{11} \begin{pmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{pmatrix} & u_{12} \begin{pmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{pmatrix} \\ u_{21} \begin{pmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{pmatrix} & u_{22} \begin{pmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{pmatrix} \end{pmatrix} = \begin{pmatrix} u_{11}v_{11} & u_{11}v_{12} & u_{12}v_{11} & u_{12}v_{12} \\ u_{11}v_{21} & u_{11}v_{22} & u_{12}v_{21} & u_{12}v_{22} \\ u_{21}v_{11} & u_{21}v_{12} & u_{22}v_{11} & u_{22}v_{12} \\ u_{21}v_{21} & u_{21}v_{22} & u_{22}v_{21} & u_{22}v_{22} \end{pmatrix}.$$

1.2.3 Partial measurement

With the tensor product, we can now talk about unitary matrices applied to a subset of qubits in our computation. As it turns out, it also makes sense to measure a subset of qubits. Once again, we can appeal to our classical intuition. Suppose we have the following classical distribution over outcomes:

$$\begin{pmatrix} 0.3 \\ 0.1 \\ 0.3 \\ 0.3 \end{pmatrix} \leftarrow \begin{matrix} 00 \\ 01 \\ 10 \\ 11 \end{matrix}$$

Suppose we look at the second coin, but not the first. The probability of seeing the 0-outcome for the second coin is

$$\Pr[00\text{-outcome}] + \Pr[10\text{-outcome}] = .3 + .3 = .6$$

since both of those outcomes are consistent with seeing 0 for the second coin. By an identical calculation, we see the 1-outcome for the second coin with 40% probability.

Let's suppose we do see the second coin in the 0-outcome. Now we must calculate the distribution on the first coin conditioned on seeing the second coin in the 0-outcome. For either outcome $b \in \{0, 1\}$, we have

$$\Pr[b \text{ for first coin} \mid 0 \text{ for second coin}] = \frac{\Pr[(b \text{ for first coin}) \wedge (0 \text{ for second coin})]}{\Pr[0 \text{ for second coin}]}.$$

In our example, the probability we see the 0-outcome on the first coin conditioned on having seen 0 for the second coin is just $.3/.6 = .5$. In practice, it's often easiest to do these calculations by simply removing the outcomes that are inconsistent with the partial measurement, and then renormalizing the vector. For our example where we've seen the 0-outcome on the second coin, we have

$$\begin{pmatrix} 0.3 \\ 0.1 \\ 0.3 \\ 0.3 \end{pmatrix} \xrightarrow{\text{Remove inconsistent outcomes}} \begin{pmatrix} 0.3 \\ 0 \\ 0.3 \\ 0 \end{pmatrix} \xrightarrow{\text{Renormalize}} \begin{pmatrix} 0.5 \\ 0 \\ 0.5 \\ 0 \end{pmatrix}.$$

Once again, the quantum setting is identical except everything is done with respect to the ℓ_2 -norm rather than the ℓ_1 -norm. For completeness, let's look at a similar example with a quantum state:

$$\begin{pmatrix} \sqrt{0.3} \\ \sqrt{0.1} \\ \sqrt{0.3} \\ \sqrt{0.3} \end{pmatrix} \begin{matrix} \leftarrow 00 \\ \leftarrow 01 \\ \leftarrow 10 \\ \leftarrow 11 \end{matrix}$$

The probability we see the 0-outcome for the second qubit is $|\sqrt{.3}|^2 + |\sqrt{.3}|^2 = .6$, and the resulting state conditioned on this outcome is

$$\begin{pmatrix} \sqrt{0.3} \\ \sqrt{0.1} \\ \sqrt{0.3} \\ \sqrt{0.3} \end{pmatrix} \xrightarrow{\text{Remove inconsistent outcomes}} \begin{pmatrix} \sqrt{0.3} \\ 0 \\ \sqrt{0.3} \\ 0 \end{pmatrix} \xrightarrow{\text{Renormalize}} \begin{pmatrix} \sqrt{0.5} \\ 0 \\ \sqrt{0.5} \\ 0 \end{pmatrix}.$$

This procedure will be easier to describe more formally once we've introduced the notation in the following section.

1.3 Dirac notation and inner products

Let's start this section by introducing a method for writing quantum states, called *Dirac notation*. While this notation may at first seem somewhat unnecessary, it turns out to be quite

natural. The most basic notational idea is that we will use a “ket”, which looks like $|\cdot\rangle$, to describe a vector that is supposed to be a quantum state (i.e., a unit vector with respect to the ℓ_2 -norm). Importantly, we reserve special notation for certain states. In particular, the 0-outcome and 1-outcome states, which we have previously been referring to somewhat awkwardly, are now associated with the following vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

So, for example, we can write an arbitrary single-qubit quantum state $|\psi\rangle$ as

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

for amplitudes $\alpha, \beta \in \mathbb{C}$. To write multi-qubit states in this notation, we employ another useful shorthand for bit strings $x \in \{0, 1\}^n$:

$$|x\rangle := |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle$$

We call such states the *classical basis states*. Now, any n -qubit state $|\psi\rangle$ can be written as a linear combination of the classical basis states:

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$$

where $\alpha_x \in \mathbb{C}$ is some complex amplitude for each $x \in \{0, 1\}^n$. For example, we can write the Bell state introduced in the previous section as

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

As a note of caution, you will very often see that the tensor product symbol is dropped when using Dirac notation. For example, all of the following might be used for the classical basis states:

$$|0\rangle \otimes |1\rangle \otimes |0\rangle = |010\rangle = |0\rangle |1\rangle |0\rangle.$$

For general states $|\psi\rangle$ and $|\varphi\rangle$, you will often see

$$|\psi\rangle \otimes |\varphi\rangle = |\psi\rangle |\varphi\rangle.$$

1.3.1 Inner products

Every quantum state lives in a vector space $\mathbb{C}^d$. We will often use that this vector space is actually a *Hilbert space*, meaning that it is equipped with an inner product: for vectors $v, w \in \mathbb{C}^d$, their *inner product* is defined as

$$v^\dagger w = \sum_{i=1}^d \overline{v_i} w_i.$$

In Dirac notation, we write $\langle\psi|$ (pronounced “bra”- ψ) to denote the conjugate transpose of the state $|\psi\rangle$. Therefore, the inner product between two states $|\psi\rangle$ and $|\varphi\rangle$ is written as

$$\langle\psi|\varphi\rangle := \overset{\text{bra}}{\downarrow} \langle\psi| \cdot \overset{\text{ket}}{\downarrow} |\varphi\rangle$$

where the lefthand side shows yet another shorthand. Now we can finally see the reason for the weird names “bra” and “ket”. When you put them together to form an inner product, you get the phrase “braket”, which looks like “bracket” if you squint.

Why go through all this trouble to create a shorthand for inner products? Perhaps most importantly, the inner product induces a natural distance measure on quantum states. If the inner product of two states is 1, then the states are identical. If the inner product is 0, then the states are perfectly distinguishable.

1.3.2 Outer products

We can also use Dirac notation to denote the outer product between states in the natural way. For states $|\psi\rangle, |\varphi\rangle \in \mathbb{C}^d$, their outer product is

$$|\psi\rangle\langle\varphi| := \begin{pmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_d \end{pmatrix} \begin{pmatrix} \overline{\varphi_1} & \overline{\varphi_2} & \cdots & \overline{\varphi_d} \end{pmatrix} = \begin{pmatrix} \psi_1\overline{\varphi_1} & \psi_1\overline{\varphi_2} & \cdots & \psi_1\overline{\varphi_d} \\ \psi_2\overline{\varphi_1} & \psi_2\overline{\varphi_2} & \cdots & \psi_2\overline{\varphi_d} \\ \vdots & \vdots & \ddots & \vdots \\ \psi_d\overline{\varphi_1} & \psi_d\overline{\varphi_2} & \cdots & \psi_d\overline{\varphi_d} \end{pmatrix}$$

The outer product is useful for describing quantum operations. For example, an arbitrary n -qubit unitary U can be written as

$$U = \sum_{x,y \in \{0,1\}^n} u_{x,y} |x\rangle\langle y|$$

where $u_{x,y} = \langle x|U|y\rangle \in \mathbb{C}$ is the amplitude the unitary places on the state $|x\rangle$ on input $|y\rangle$. In this case, $|x\rangle\langle y|$ is just the matrix which is 1 at entry (x, y) and 0 everywhere else.

Summary – Quantum computation over n qubits

States: $|\psi\rangle \in \mathbb{C}^{2^n}$ such that $\sum_{x \in \{0,1\}^n} |\langle x|\psi\rangle|^2 = 1$

Operations: $U \in \mathbb{C}^{2^n \times 2^n}$ such that $U^\dagger U = U^\dagger U = I$
Applying U to $|\psi\rangle$ results in the state $U|\psi\rangle$

Measurement: State collapses to $|x\rangle$ with probability $|\langle x|\psi\rangle|^2$

1.4 Mixed states

For many questions in quantum computation, the formalism of states and operations we’ve previously developed is sufficient. For example, most quantum algorithms start with some classical basis state, apply some unitary operation, and then measure. However, there is

actually a more general form of a quantum state that is useful in a variety of contexts, like when you have noise in your quantum computer.

The quantum states $|\psi\rangle$ we have defined previously are called *pure states*. What makes a state “impure”, or as it’s traditionally called “mixed”? We say that a state is *mixed* when it represents a probability distribution over pure states. To see why these two notions are different, it’s helpful to look at an example.

On the one hand, let’s take the pure state $|+\rangle := \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ which is in some sense equal parts $|0\rangle$ and $|1\rangle$. On the other hand, let’s take the mixed state which is either $|0\rangle$ or $|1\rangle$ with 50% probability. These states may superficially seem to be the same (after all, they have the same probability over outcomes when measured), but are actually quite different. To see this, let’s examine what happens when we apply the following unitary H , which is called the *Hadamard gate*:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Applying H to our pure state $|+\rangle$, we get

$$H|+\rangle = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

In other words, if we were to measure our pure state *after* the application of the unitary operation H , then we are guaranteed to see the outcome $|0\rangle$. This will not be true in our mixed state picture. Let’s do the calculation. Applying H to the mixed state, we get

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = |+\rangle$$

and

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} =: |-\rangle,$$

each of which happens with 50% probability. What is the probability we measure $|0\rangle$ now? Given the calculation above of what the Hadamard transformation does to each of our starting states, we have

$$\begin{aligned} \Pr[\text{measure } |0\rangle] &= \Pr[\text{Original state was } |0\rangle] \cdot \Pr[\text{measure } |0\rangle \text{ on state } |+\rangle] \\ &\quad + \Pr[\text{Original state was } |1\rangle] \cdot \Pr[\text{measure } |0\rangle \text{ on state } |-\rangle] \\ &= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2} \end{aligned}$$

We can now see that when our state was a statistical mixture of $|0\rangle$ and $|1\rangle$, the Hadamard transformation didn’t change our measurement probabilities at all. In fact, this is a general phenomenon. One can show that no matter what unitary transformation you apply to this mixed state, you will always get $|0\rangle$ and $|1\rangle$ with 50% probability. This will be easy to show using the formalism we now introduce.

1.4.1 Density matrices

General quantum systems are fully described by statistical mixtures of quantum states—that is, an ensemble of pure states $\{|\psi_i\rangle\}_i$ each of which is prepared with probability $p_i \in [0, 1]$. The *density matrix* corresponding to this ensemble is

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i| \in \mathbb{C}^{2^n \times 2^n}$$

where $\sum_i p_i = 1$. One can show that if you have a density matrix ρ and apply a unitary U , then the new density matrix is given by $U\rho U^\dagger$. Furthermore, measurement results in outcome $|x\rangle$ with probability $\langle x|\rho|x\rangle$, whereupon ρ collapses to the state $|x\rangle\langle x|$.

Let's revisit our example of an even statistical mixture of the states $|0\rangle$ and $|1\rangle$. The corresponding density matrix is

$$\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \frac{I}{2}.$$

As it turns out, this state is called the *maximally mixed state* since it represents that we essentially have no knowledge of what the underlying state is. To see this, imagine applying any unitary U to this state. We would get

$$U\left(\frac{I}{2}\right)U^\dagger = \frac{UU^\dagger}{2} = \frac{I}{2},$$

as the new state, which is the same state we started with. In other words, no unitary operation changes how the state looks. This proves the claim we made earlier that any unitary followed by measurement would result in outcomes $|0\rangle$ and $|1\rangle$ with equal probability.

What matrices correspond to ensembles of pure states? As it turns out, there is a very nice characterization: ρ is a valid density matrix if and only if ρ is a trace-1 positive semidefinite matrix. Trace-1 implies that $\text{Tr}(\rho) = 1$. Positive semidefinite implies that $\langle\psi|\rho|\psi\rangle \geq 0$ for all pure states $|\psi\rangle$.

The forward direction of this claim can be shown by reasoning directly about the types of matrices that an ensemble of states can give rise to. The reverse direction can be shown by taking the spectral decomposition of ρ , which is valid since we have assumed that ρ is positive semidefinite. The eigenvectors of this decomposition will be the pure states in the decomposition, and the eigenvalues will be the associated probabilities.

1.4.2 Quantum channels

As one might have now guessed, unitary transformations are also not the most general transformation on quantum states. Quantum transformations that work on the level of density matrices are called *quantum channels*. That said, it is not true that every channel which preserves density matrices corresponds to a valid quantum operation. Most importantly, as required by the axioms of quantum mechanics, the channel must be linear. Furthermore, for technical reasons having to do with applying the channel to a restricted set of qubits, we must also require that the quantum channel still maps density matrices to density matrices when it is tensored with the identity map. Maps satisfying all the above conditions are called *completely positive trace preserving (CPTP)*.

1.4.3 Measurement

While there is a more general form of quantum measurements, it turns out that these more general measurements can be simulated by the measurements that we have already introduced. So, for simplicity, we will always assume that we measure our qubits the usual way.

Summary – Quantum computation with n -qubit mixed states

States:	$\rho \in \mathbb{C}^{2^n \times 2^n}$ such that $\text{Tr}(\rho) = 1$ and ρ is positive semidefinite
Operations:	Completely positive trace-preserving maps Φ If Φ is a unitary channel, then $\Phi(\rho) = U\rho U^\dagger$ for unitary $U \in \mathbb{C}^{2^n \times 2^n}$
Measurement:	State collapses to $ x\rangle\langle x $ with probability $\langle x \rho x\rangle$

1.4.4 Partial Trace

One of the most important reasons to introduce the density matrix formalism is to be able to talk about parts of a quantum state in isolation. That is, even if we have an n -qubit pure state, it is not necessarily the case that the state restricted to, say, the first $n/2$ qubits is a pure state.

We now introduce a way to “trace out” part of a density matrix of a large system to describe the state on the leftover qubits. To start, consider a composite system $\mathcal{H}_A \otimes \mathcal{H}_B$. For simplicity, you can at first just assume that $\mathcal{H}_A$ and $\mathcal{H}_B$ are the Hilbert spaces for two different qubits. Formally, the partial trace Tr_B is the unique linear map satisfying

$$\text{Tr}_B(|a_i\rangle\langle a_j| \otimes |b_k\rangle\langle b_l|) = |a_i\rangle\langle a_j| \text{Tr}(|b_k\rangle\langle b_l|),$$

where $a_i, a_j \in \mathcal{H}_A$ and $b_k, b_l \in \mathcal{H}_B$ are basis elements for the two subsystems.

So, if we have some state ρ_{AB} that lives in the Hilbert space $\mathcal{H}_A \otimes \mathcal{H}_B$, then the density matrix for the subsystem A after ignoring the subsystem B is given by

$$\rho_A = \text{Tr}_B(\rho_{AB}).$$

If we apply the partial trace operator to a product state we get, unsurprisingly,

$$\text{Tr}_B(\rho_A \otimes \rho_B) = \rho_A.$$

What happens when we take the partial trace of the Bell state? The density matrix is given by

$$\rho_{\text{Bell}} := \left(\frac{|00\rangle + |11\rangle}{\sqrt{2}} \right) \left(\frac{\langle 00| + \langle 11|}{\sqrt{2}} \right) = \frac{|00\rangle\langle 00| + |00\rangle\langle 11| + |11\rangle\langle 00| + |11\rangle\langle 11|}{2}$$

so tracing out the second qubit, we get (by linearity of the partial trace)

$$\begin{aligned} \text{Tr}_2(\rho_{\text{Bell}}) &= \frac{1}{2} (\text{Tr}_2(|00\rangle\langle 00|) + \text{Tr}_2(|00\rangle\langle 11|) + \text{Tr}_2(|11\rangle\langle 00|) + \text{Tr}_2(|11\rangle\langle 11|)) \\ &= \frac{1}{2} (|0\rangle\langle 0| \text{Tr}(|0\rangle\langle 0|) + |0\rangle\langle 1| \text{Tr}(|0\rangle\langle 1|) + |1\rangle\langle 0| \text{Tr}(|1\rangle\langle 0|) + |1\rangle\langle 1| \text{Tr}(|1\rangle\langle 1|)) \\ &= \frac{1}{2} (|0\rangle\langle 0| \cdot 1 + |0\rangle\langle 1| \cdot 0 + |1\rangle\langle 0| \cdot 0 + |1\rangle\langle 1| \cdot 1) \\ &= \frac{|0\rangle\langle 0| + |1\rangle\langle 1|}{2}. \end{aligned}$$

That is, if we take the Bell state and trace out a qubit, we are left with the maximally mixed state. This may give you some sense of the fragility of quantum computations. If you take an entangled state and lose a single qubit, it may become completely useless.

1.4.5 Reconciling the pure and mixed states

Often it will be easier to reason about pure states rather than mixed ones. As we've seen before, this is in some sense fundamentally impossible—there are mixed states which behave completely differently from pure ones. That said, there is also some sense in which there is an equivalence between the two settings. Namely, for every n -qubit mixed state ρ , there is a $(2n)$ -qubit pure state $|\psi\rangle$ such that tracing out the last n qubits of $|\psi\rangle$ leaves the state ρ . This process is called *purification*.

We will give an explicit purification procedure. First, let ρ be an arbitrary n -qubit mixed state:

$$\rho = \sum_{x \in \{0,1\}^n} p_x |\psi_x\rangle\langle\psi_x|$$

The following state will be a purification of ρ :

$$|\psi\rangle := \sum_{x \in \{0,1\}^n} \sqrt{p_x} |\psi_x\rangle \otimes |x\rangle$$

Let B be the system consisting of the last n qubits. Tracing out B , we get the density matrix:

$$\begin{aligned} \text{Tr}_B (|\psi\rangle\langle\psi|) &= \text{Tr}_B \left(\sum_{x,y} \sqrt{p_x p_y} |\psi_x\rangle\langle\psi_y| \otimes |x\rangle\langle y| \right) \\ &= \sum_{x,y} \sqrt{p_x p_y} \text{Tr}_B (|\psi_x\rangle\langle\psi_y| \otimes |x\rangle\langle y|) && \text{(Linearity of partial trace)} \\ &= \sum_{x,y} \sqrt{p_x p_y} |\psi_x\rangle\langle\psi_y| \text{Tr}(|x\rangle\langle y|) && \text{(Definition of partial trace)} \\ &= \sum_x p_x |\psi_x\rangle\langle\psi_x| && \text{(Trace is 1 iff } x = y) \end{aligned}$$

which is precisely the mixed state ρ that we wanted to embed into $|\psi\rangle$.

Are purifications unique? Unfortunately, no. To see this, notice that we can generalize our purification procedure above by applying any n -qubit unitary U to the second register:

$$|\psi\rangle := \sum_{x \in \{0,1\}^n} \sqrt{p_x} |\psi_x\rangle \otimes (U|x\rangle).$$

Intuitively, it makes sense that changing the basis of the second register shouldn't affect the partial trace since we never used anything special about the classical basis states. Formally, you can check that the computation is agnostic to the choice of unitary U because of the following equalities:

$$\text{Tr}(U|x\rangle\langle y|U^\dagger) = \text{Tr}(U^\dagger U|x\rangle\langle y|) = \text{Tr}(|x\rangle\langle y|)$$

where the first equality uses the cyclic property of the trace and the second uses the fact that U is unitary.

1.5 Noteworthy quantum phenomena

Let's start to use the quantum formalism to take note of some interesting phenomena. We start with a classic result which implies that quantum information cannot be copied.

Theorem 1.1 (No-Cloning Theorem). *There is no $(2n)$ -qubit unitary U and n -qubit state $|\varphi\rangle$ such that*

$$U(|\psi\rangle \otimes |\varphi\rangle) = |\psi\rangle \otimes |\psi\rangle$$

for all n -qubit pure states $|\psi\rangle$.

Proof. We argue by contradiction. Suppose such a U and $|\varphi\rangle$ existed, and let $|\psi_1\rangle, |\psi_2\rangle$ be two states we want to copy. In other words, we have

$$U(|\psi_i\rangle \otimes |\varphi\rangle) = |\psi_i\rangle \otimes |\psi_i\rangle$$

for $i \in \{1, 2\}$. Let's now take the inner product of the two states $U(|\psi_1\rangle \otimes |\varphi\rangle)$ and $U(|\psi_2\rangle \otimes |\varphi\rangle)$

$$(\langle\psi_1| \otimes \langle\varphi|)U^\dagger U(|\psi_2\rangle \otimes |\varphi\rangle) = \langle\psi_1|\psi_2\rangle \langle\varphi|\varphi\rangle = \langle\psi_1|\psi_2\rangle$$

and compare it to the inner product of $|\psi_1\rangle \otimes |\psi_1\rangle$ and $|\psi_2\rangle \otimes |\psi_2\rangle$:

$$(\langle\psi_1| \otimes \langle\psi_1|)(|\psi_2\rangle \otimes |\psi_2\rangle) = \langle\psi_1|\psi_2\rangle^2.$$

Cloning implies that these two expressions are equal:

$$\langle\psi_1|\psi_2\rangle = \langle\psi_1|\psi_2\rangle^2.$$

However, for any states such that $\langle\psi_1|\psi_2\rangle \notin \{0, 1\}$, the above equation will not hold. That is, cloning breaks for any distinct pair of non-orthogonal states! $\square$

Chapter 2

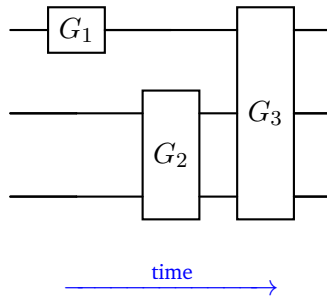
Computation with Quantum Circuits

How do we describe a quantum algorithm? One might think that something like a generalization of the classical Turing machine may be a particularly apt choice, given the centrality of that model to the story of classical theory of computation. While it is possible to define a quantum Turing machine, it turns out to be rather cumbersome to work with.

Instead, we will use a model of computation that more-or-less is the straightforward realization of applying a sequence of unitaries—the *quantum circuit*.

2.1 Introduction to quantum circuits

A n -qubit quantum circuit is a collection of unitary operations $G_1, \dots, G_m$, called *gates*, applied in sequence to a subset of n *wires*. The composition of the gates in the circuit generates a $2^n \times 2^n$ unitary operation. We assume that each gate is in tensor product with the identity operation on each wire that it does not touch. Let's look at a simple example:



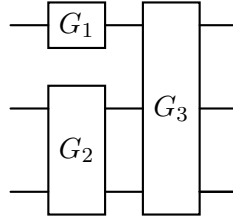
The above diagram is a circuit on 3 qubits with 3 gates: the single-qubit gate G_1 is applied first; the 2-qubit gate G_2 is applied next; and finally G_3 is applied as a 3-qubit gate. The unitary matrix representing this circuit is

$$G_3 (I \otimes G_2) (G_1 \otimes I \otimes I).$$

Beware: matrix multiplication happens the reverse order of the circuit, which is why G_1 appears last the composition of unitaries. Since G_1 and G_2 act on different wires, we get that

$$(I \otimes G_2) (G_1 \otimes I \otimes I) = G_1 \otimes G_2.$$

Therefore, in the diagram, we can put G_1 and G_2 on the same *layer*.



That is, a layer of the circuits consists of a set of gates that can be applied simultaneously since they act on different qubits. The *depth* of a circuit is the number of layers of gates it has. Therefore, the example circuit above has depth 2.

2.1.1 Examples with common gates

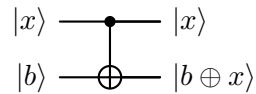
Let's take a look at some of the most common gates used in quantum circuits and the special notation that we use to denote them.

Classical reversible gates

One of the most common two-qubit gates is the *controlled-NOT* or *CNOT* gate. Recall that by linearity, it suffices to define the action of any gate on the computational basis. CNOT has the following action:

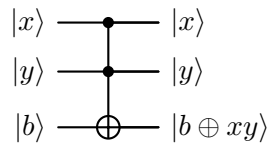
$$|00\rangle \mapsto |00\rangle, \quad |01\rangle \mapsto |01\rangle, \quad |10\rangle \mapsto |11\rangle, \quad |11\rangle \mapsto |10\rangle.$$

Notice that CNOT maps any computational basis state to another computational basis state. That is, the CNOT gate is “classical” in the sense that it cannot be used to create superposition of inputs. A CNOT gate in a circuit is depicted as a $\bullet$ symbol (the *control*) connected to a $\oplus$ symbol (the *target*):



Here, we've shown how the CNOT gate acts on general computational basis states, where $x, b \in \{0, 1\}$ are arbitrary bits and $b \oplus x$ denotes their XOR (i.e., addition modulo 2).

Another related gate is the version of the CNOT gate with an extra control, that is, the *controlled-controlled-NOT* gate, most commonly referred to as the *Toffoli* gate. As a circuit, it looks like



where $x, y, b \in \{0, 1\}$ are arbitrary bits (xy is the product of x and y). Notice that the third bit is flipped exactly when both controls are 1.

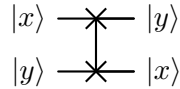
The Toffoli gate is in some sense more powerful than the CNOT gate since it can be used to generate the CNOT gate. Notice that if we set the second input qubit above to $|1\rangle$ (i.e., $y = 1$),

then the remaining effect on the remaining two qubits is exactly the CNOT gate. We will see later however, that the reverse is not true—we cannot just use the CNOT gate to generate a Toffoli gate.

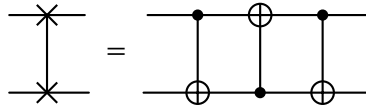
Finally, let's discuss the SWAP gate, another important “classical reversible” gate on 2 qubits. Aptly named, the SWAP gate swaps qubits, i.e., for all $x, y \in \{0, 1\}$ it maps:

$$|xy\rangle \mapsto |yx\rangle.$$

In a circuit diagram, it is depicted as



One can check the following nice identity:



In other words, we can replace every SWAP gate in a circuit with 3 CNOT gates. This is a common theme we will continue to see—we can take some gates as the fundamental ones that will generate the rest.

Change of basis operations

Evidently, we need a gate that can create a superposition of inputs from a classical basis state. The *Hadamard gate* is the canonical choice for such an operation. It has the action

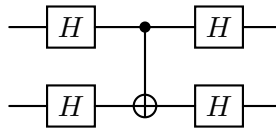
$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} := |+\rangle \quad H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} := |-\rangle$$

on the computational basis. Notice that Hadamard gate has given rise to a new basis, the $\{|+\rangle, |-\rangle\}$ basis. In fact, Hadamard switches back and forth between the computational basis and this new basis. That is, the Hadamard gate is its own inverse: $H^2 = I$. As a circuit, it is shown as

$$|x\rangle \xrightarrow{H} \frac{|0\rangle + (-1)^x |1\rangle}{\sqrt{2}}$$

for any $x \in \{0, 1\}$.

As another example, let's consider a circuit built from Hadamard and CNOT gates:



One way of understanding this circuit would be to just explicitly compute the unitary matrix $(H \otimes H)\text{CNOT}(H \otimes H)$, but it is often more helpful to instead look at how the system evolves over a basis. Let's see how it acts on the computational basis, considering one gate at a time:

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

That is, after applying $H \otimes H$, we have the uniform superposition over 2-qubit computational basis states. We know that the CNOT gate just permutes the elements of the computational basis, or, in other words, it must do nothing to do the above state:

$$\frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2} \xrightarrow{\text{CNOT}} \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}.$$

Of course, if we've done nothing to the state, then it must also factorize as

$$|+\rangle \otimes |+\rangle = (H \otimes H) |00\rangle.$$

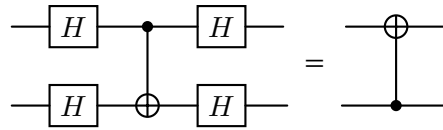
Therefore, the final layer of Hadamard gates returns the state to $|00\rangle$. That is, after all that computation, we see that the circuit acts as the identity on the $|00\rangle$. For completeness, let's see one more case (the input $|01\rangle$) in its entirety:

$$\begin{aligned} |01\rangle &\xrightarrow{H \otimes H} \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) \otimes \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = \frac{|00\rangle - |01\rangle + |10\rangle - |11\rangle}{2} \\ &\xrightarrow{\text{CNOT}} \frac{|00\rangle - |01\rangle - |10\rangle + |11\rangle}{2} = |-\rangle \otimes |-\rangle \\ &\xrightarrow{H \otimes H} |11\rangle \end{aligned}$$

If we were to continue with the entire computational basis, we would see

$$|00\rangle \mapsto |00\rangle, \quad |01\rangle \mapsto |11\rangle, \quad |10\rangle \mapsto |10\rangle, \quad |11\rangle \mapsto |01\rangle.$$

We've seen this gate before. It's just the CNOT gate with the control on the second qubit instead of the first! That is, we've derived the following circuit identity:



Notice that up until this point, every gate that we've introduced is *real*—the all elements of the unitary matrix representing the gate are real numbers. Let's now introduce some gates that have complex entries.

Phase gates

The gate most commonly referred to as the “phase gate” is the single-qubit diagonal gate S that simply multiplies the $|1\rangle$ state by a phase of i :

$$S|0\rangle = |0\rangle \quad S|1\rangle = i|1\rangle$$

Another type of phase gate, most commonly called a T -gate, is the square root of this operation:

$$T|0\rangle = |0\rangle \quad T|1\rangle = e^{i\frac{\pi}{4}}|1\rangle$$

Unsurprisingly, there are many other common gates that we have yet to define. Thankfully, the gates we have already allow us to do essentially everything we want.

2.2 Universality, approximations, and circuit size

A *gate set* is the collections of gates that one can use in the construction of a circuit. Typically, when a gate is included in a gate set, then you're allowed to apply that gate as many times you like on whichever subset of qubits that you like.

Universality captures the notion that a particular gate set can be used to construct any possible quantum operation. There are several different kinds of universality you might want:

- *Exact Universality*: For any n -qubit unitary, there is a circuit that exactly compute the unitary.
- *Approximate Universality*: For any n -qubit unitary, there is a circuit that approximately computes the unitary. One common measure of closeness is the operator norm.
- *Computational Universality*: For any n -qubit unitary, the probability distribution resulting from measuring the first qubit can be approximated by measuring the first qubit of the circuit. For example, it turns out that real quantum gates (without complex entries) are sufficient for computational universality, whereas they clearly fail on the other two notions of universality.

Given that we have a universal gate set, how many gates do we actually need to construct an arbitrary unitary? Let's look at the exact case, where we can get an estimate based on the number of parameters it takes to specify arbitrary unitary matrix. The first claim is that an arbitrary complex $d \times d$ unitary matrix U is specified by d^2 real parameters.

To see this, first note there are d^2 entries in the matrix, each with a complex part and a real part, that is, $2d^2$ real parameters total. However, the unitary constraint $UU^\dagger = I$ imposes d^2 algebraically independent real conditions (d for the fact that the norm of each column should be 1, and $d(d-1)$ conditions for the fact that the complex inner product of each row should be 0).

If our gate set consists of gates that only act on a constant number of qubits (which is often the convention), then each such gate only contributes constantly many real parameters to the construction of the unitary. Therefore, we must have $\Omega(4^n)$ gates to construct an arbitrary n -qubit unitary exactly.

A generalization of this result shows that this lower bound is essentially tight for approximation computation as well— $\Omega(4^n \log(1/\epsilon))$ gates are required to approximately compute any unitary to within ϵ -accuracy with respect to the operator norm [DN06]. That is, a unitary U is ϵ -close to unitary V if

$$\|U - V\|_{\text{op}} = \sup_{\psi} \|(U - V)|\psi\rangle\|_2 \leq \epsilon.$$

Thankfully, there is a matching (up to polylog factors) circuit building algorithm as well.

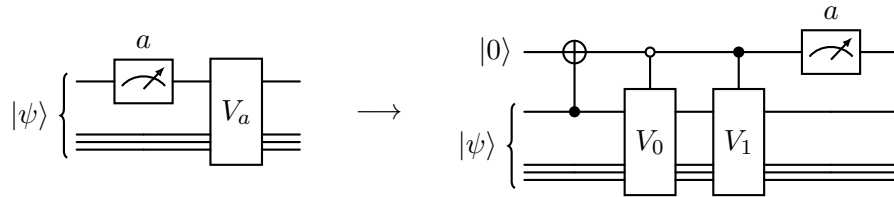
Theorem 2.1 (Solovay-Kitaev [Kit97]). *Given an approximately universal gate set, there is a circuit to approximate any unitary to ϵ -accuracy with $\mathcal{O}(4^n \text{polylog}(\frac{1}{\epsilon}))$ gates.*

The version of the Solovay-Kitaev theorem stated above is actually the result of Bouland and Giurgica-Tiron [BGT21], who show how to work with general gate sets. Unfortunately, their result suffers in the exponent of the log factor. The original and most-efficient Solovay-Kitaev theorems require that if a gate is in the gate set, then its inverse is also in the gate set. The current best result in this setting is by Kuperberg, who shows a bound of $\mathcal{O}(4^n \log^{1.441}(1/\epsilon))$ gates [Kup23].

2.3 Principle of Deferred Measurement

So far in this section, we’ve used unitary gates as the only operations in a quantum circuit. That is, we have been implicitly assuming that measurements are performed at the end of the circuit. Measurement is a non-unitary operation, so is it possible that by using intermediate measurements in the “middle” of the circuit, we might be able to more efficiently construct a particular unitary operation?

The *principle of deferred measurement* says that each intermediate measurements can essentially be pushed to the end of circuit by introducing a new ancillary qubit. The resulting probability distribution over all measurements made in the circuit will be the same before and after the transformation. The figure below depicts a general form of this transformation:



The left side shows a quantum circuit where the intermediate measurement has outcome $a \in \{0, 1\}$ and unitary V_a is applied as a result. The right side shows a quantum circuit where this measurement has been deferred to the end of the circuit by pushing it onto an ancilla. (Note: a control gate with an open circle is usually used to denote that the gate is controlled on 0, rather than 1.)

To verify correctness of this procedure, it suffices to check that tracing out the first qubit on the right side circuit results in the same density matrix as you get from the left.

Chapter 3

Query Complexity

In this chapter we explore one of our first tools for comparing the power of quantum and classical computation—*query complexity*. In the query complexity setting, we imagine there is some property of a function that we are trying to compute. The catch is that we can only learn things about the function by “querying” its value on a single input at a time. The number of times we need to query the function to learn the property is new measure of complexity (rather than something like the gate count in the quantum circuit).

The benefit of this approach is that the blackbox nature of the function greatly restricts the kinds of algorithms (both quantum and classical) that you could use to solve the problem. This will allow us to prove tight bounds on the query complexity for both the quantum and classical computers. When the quantum computation requires significantly fewer queries, we have evidence of a quantum advantage. In fact, many of the efficient query algorithms we will discuss in this chapter are also efficient in the more traditional sense (i.e., have polynomial-size quantum circuits), and therefore, form the basis for some of the most promising quantum algorithms.

For some intuition for the power of this setting, imagine a satisfiability problem captured by a function $f: \{0, 1\}^n \rightarrow \{0, 1\}$. That is, we want to find some input $x \in \{0, 1\}^n$ such that $f(x) = 1$. Now imagine the function is instantiated by polynomially many constraints, e.g., an NP-complete problem like 3-SAT. It is a famously open problem if NP-hard problem like this can be solved in polynomial time. However, when we look at the query version of this problem—that is, we can only query f on inputs $x \in \{0, 1\}^n$ one at a time—the problem is obviously hopelessly difficult for a classical polynomial-time machine. If there is at most one possible input x such that $f(x) = 1$, we have to make exponentially many queries to determine if such an x exists.

Despite the restricted access model for query algorithms, we will see that there can still be nontrivial algorithms for a variety of different problems. The purpose of this section is to showcase how quantum algorithms fare in this setting in comparison to their classical counterparts.

3.1 Defining a quantum oracle

Querying a function $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$ is pretty straightforward in the classical circuit setting—the queries consist of n -input m -output gates, and the query complexity is the num-

ber of these query gates that are required to solve the problem.

Let's now discuss what exactly it means to query the function quantumly. We will use two oracle models. First, we define the *standard oracle* B_f which acts on two registers, an input register and an output register:

$$B_f |x\rangle |b\rangle = |x\rangle |b \oplus f(x)\rangle$$

for all $x \in \{0, 1\}^n$ and $b \in \{0, 1\}^m$. That is, the standard oracle just computes the value of the function on the input and dumps it (reversibly) into the output register.

Is this a reasonable model? In other words, how can we justify that we are not cheating by giving the quantum algorithm a query model which is fundamentally more powerful than the query model in the classical setting (in a way which is unrelated to the power of quantum computation)? One way to see this is to imagine what it would look like to instantiate the function f for a practical problem. For any setting where there is a classical circuit for f (e.g., in our example where f was a 3-SAT formula), then the quantum circuit can implement the oracle B_f by straightforwardly implementing the classical circuit in superposition. On the other hand, if there's no classical circuit for f , then the classical oracle also doesn't make any sense!

As it turns out, it's often useful to have another query model where the output of f is computed in the phase of the input. For every function $f: \{0, 1\}^n \rightarrow \{0, 1\}$, the *phase oracle* O_f is defined so that

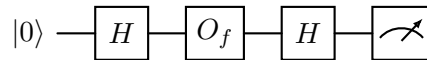
$$O_f |x\rangle = (-1)^{f(x)} |x\rangle$$

for all $x \in \{0, 1\}^n$. The phase oracle is only marginally different from the standard oracle. In fact, the standard oracle can simulate the phase oracle with a single ancilla qubit (which we leave as an exercise). An almost-identical construction shows that the B_f oracle can be simulated by a single query to a controlled- O_f oracle.

3.2 Fourier sampling problems

Let's start with one of the simplest examples of a quantum-classical query separation. Our goal will be determine if a function $f: \{0, 1\} \rightarrow \{0, 1\}$ is constant or not, i.e., if $f(0) = f(1)$ or not. Classically, it is clear that we need two queries. We must check both $f(0)$ and $f(1)$ to know if they are equal.

We claim there is a simple 1-query quantum circuit (Deutsch's algorithm) for this task:



Tracing through the circuit, we get

$$\begin{aligned} |0\rangle &\xrightarrow{H} \frac{|0\rangle + |1\rangle}{\sqrt{2}} \xrightarrow{O_f} \frac{(-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle}{\sqrt{2}} = (-1)^{f(0)} \left(\frac{|0\rangle + (-1)^{f(0) \oplus f(1)} |1\rangle}{\sqrt{2}} \right) \\ &\xrightarrow{H} (-1)^{f(0)} |f(0) \oplus f(1)\rangle \end{aligned}$$

Therefore, if $f(0) = f(1)$, we will measure $|0\rangle$ with 100%, and if $f(0) \neq f(1)$, we will measure $|1\rangle$ with 100% probability. That is, we have a quantum algorithm that distinguishes between constant and non-constant functions with 100% probability.

A 1 vs. 2-query separation may not seem like a big deal, but essentially the exact same algorithm can lead to a much more impressive separation. To get these impressive separations, however, we will have to make a sacrifice. Namely, we will need to look at *promise problems*, that is, problems where the input function f has some specific property, called the “promise”. Importantly, we will never judge our algorithm’s correctness on functions f that don’t satisfy the promise. This will allow us to devise algorithms that exploit some very specific structure for a query advantage.

3.2.1 Deutsch-Jozsa problem

Let’s begin with a problem that’s the n -qubit generalization of Deutsch’s problem:

Deutsch-Jozsa problem

Input: $f: \{0, 1\}^n \rightarrow \{0, 1\}$

Promise: f is either

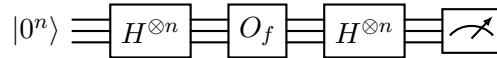
Constant: All outputs of f are equal. $\forall x, y \in \{0, 1\}^n, f(x) = f(y)$

Balanced: f has equal number of 0 and 1 outputs. $|\{x | f(x) = 1\}| = 2^{n-1}$

Question: Is f constant or balanced?

Notice that a classical deterministic machine requires $2^{n-1} + 1$ queries to f . In the worst case, the first 2^{n-1} queries to f yield the same output. It could be that all other unqueried inputs yield the same value (i.e., the function is constant) or all unqueried values yield the other value (i.e., the function is balanced). Therefore, we need 1 more query to solve the problem.

Miraculously, the quantum algorithm still only needs 1 query, and in fact, the quantum circuit is nearly identical to the one we saw previously:



Let’s step through the circuit, one layer of gates at a time:

1. Apply a layer of Hadamard gates:

$$|0^n\rangle \xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

2. Apply the phase oracle:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \xrightarrow{O_f} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle$$

3. Apply final layer of Hadamard gates:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \xrightarrow{H^{\otimes n}} \frac{1}{2^n} \sum_{x,y \in \{0,1\}^n} (-1)^{f(x)+x \cdot y} |y\rangle$$

Here, we are using “ $\cdot$ ” to denote the inner product between x and y as binary vectors (i.e., $x \cdot y = \sum_{i=1}^n x_i y_i$). To see why this is true, we remark that the result of applying an n -fold Hadamard gate on an arbitrary classical state $|x\rangle$ can be written as

$$\begin{aligned} H^{\otimes n} |x\rangle &= \bigotimes_{i=1}^n (|0\rangle + (-1)^{x_i} |1\rangle) = \bigotimes_{i=1}^n ((-1)^{x_i \cdot 0} |0\rangle + (-1)^{x_i \cdot 1} |1\rangle) \\ &= \bigotimes_{i=1}^n \left(\sum_{y_i \in \{0,1\}} (-1)^{x_i \cdot y_i} |y_i\rangle \right) = \sum_{y \in \{0,1\}^n} (-1)^{\sum_{i=1}^n x_i y_i} |y\rangle. \end{aligned}$$

What happens when we measure the state in Step 3? Let’s look specifically at the probability we measure the all-zeros state (i.e., $y = 0^n$). Since $x \cdot 0^n = 0$ for all $x \in \{0,1\}^n$, the amplitude on $|0^n\rangle$ is

$$\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)}.$$

If the function f is constant, then $f(x) = f(0^n)$ for all $x \in \{0,1\}^n$, so

$$\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(0^n)} = \frac{(-1)^{f(0^n)}}{2^n} \left(\sum_{x \in \{0,1\}^n} 1 \right) = (-1)^{f(0^n)}$$

In other words, if we were to measure the state, then we would observe the all-zeros state with 100% probability.

If f is balanced, instead of all the amplitudes on the all-zeros state adding up constructively, they all cancel each other out:

$$\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} = \frac{|\{x \mid f(x) = 0\}| - |\{x \mid f(x) = 1\}|}{2^n} = 0.$$

That is, if f is balanced, then we measure a state which is *not* the all-zeros state with 100% probability. Combining the two cases above, we can see that whether or not we measure the all-zeros state immediately solves the Deutsch-Jozsa problem.

At first glance, this 1 vs. $\Theta(2^n)$ quantum-classical query separation seems quite amazing, and possibly the best we could hope for. However, recall that the classical lower bound was for *deterministic* classical computation. If we were to allow for classical randomness, the problem becomes dramatically simpler. The classical algorithm would simply query f on a few uniformly random inputs. If the function is balanced, then you are likely to see two different outputs using only constantly many queries (to formalize this argument, use the Chernoff bound). That is, quantum algorithms are at best only marginally better than classical randomized algorithms for the Deutsch-Jozsa problem.

3.2.2 Bernstein-Vazirani

Let’s now consider a similar problem where the classical algorithm will struggle a bit more:

Bernstein-Vazirani problem

Input: $f: \{0, 1\}^n \rightarrow \{0, 1\}$

Promise: f is linear. For all $x \in \{0, 1\}^n$, $f(x) = x \cdot s$ for some secret string $s \in \{0, 1\}^n$

Question: Determine the value of the secret string s .

Once again, let's start by discussing the best classical query algorithm. Unlike the Deutsch-Jozsa problem, there is a fairly efficient algorithm—only n queries are needed. To see this, consider the algorithm that the queries f on the inputs $e_1 := 10 \cdots 0$, $e_2 := 010 \cdots 0$, and so on up to $e_n := 0 \cdots 01$. Notice that $f(e_i) = e_i \cdot s = s_i$, so each query reveals one of the n bits of s .

Can we do better? perhaps by using randomness? Unfortunately, not. To see this, consider that each query $x \cdot s = f(x)$ gives us a linear equation (over $\mathbb{F}_2$) where there are n unknown variables (i.e., the n bits of s). A linear system of equations with n -variables can only have a unique solution if there are at least n equations. Therefore, we require at least n queries.

As it turns out, the quantum algorithm is identical to the one for the Deutsch-Jozsa problem—a layer of Hadamard gates, followed by the phase oracle, followed by another layer of Hadamard gates. The only thing that changes is what we conclude from the measurement. Therefore, let's start from the state we constructed in Step 3 in our algorithm for the Deutsch-Jozsa problem:

$$\frac{1}{2^n} \sum_{x, y \in \{0, 1\}^n} (-1)^{f(x) + x \cdot y} |y\rangle.$$

Using the fact that $f(x) = x \cdot s$, we get

$$\frac{1}{2^n} \sum_{x, y \in \{0, 1\}^n} (-1)^{x \cdot s + x \cdot y} |y\rangle = \frac{1}{2^n} \sum_{x, y \in \{0, 1\}^n} (-1)^{x(s \oplus y)} |y\rangle$$

Looking at the amplitude on state $|s\rangle$, we see that the term $(-1)^{x(s \oplus y)} = 1$ for all x since $s \oplus s = 0$. Since there are 2^n values for x , we immediately get that amplitude on $|s\rangle$ is 1. In other words, we measure $|s\rangle$ with 100% probability, but s was exactly what we were looking for!

Therefore, the Bernstein-Vazirani problem gives us a 1 vs. n quantum-classical query separation. While this is less impressive than the initial separation we obtained for the Deutsch-Jozsa problem, it's worth emphasizing that this separation even holds against randomized classical algorithms.

3.2.3 Fourier sampling

As a final remark, we note that both the Deutsch-Jozsa and Bernstein-Vazirani algorithms are instances of *Fourier sampling*. To see this, let's quickly introduce the Fourier basis for Boolean functions. First, we define the basis functions

$$\chi_y(s) := (-1)^{y \cdot s}$$

for all $y \in \{0, 1\}^n$. These functions are orthonormal with respect to following inner product on real-valued functions $f, g: \{0, 1\}^n \rightarrow \mathbb{R}$:

$$\langle f, g \rangle := \frac{1}{2^n} \sum_{x \in \{0, 1\}^n} f(x)g(x)$$

To see orthonormality of these basis vectors, we compute

$$\langle \chi_y, \chi_z \rangle = \frac{1}{2^n} \sum_{x \in \{0, 1\}^n} (-1)^{x \cdot y + x \cdot z} = \frac{1}{2^n} \sum_{x \in \{0, 1\}^n} (-1)^{x \cdot (y \oplus z)} = \begin{cases} 1 & \text{if } y = z \\ 0 & \text{if } y \neq z \end{cases}.$$

Since we've defined a basis of 2^n independent functions, every Boolean function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ can be written uniquely as

$$f(x) = \sum_{y \in \{0, 1\}^n} \hat{f}(y) \chi_y(x)$$

for coefficients $\hat{f}(y) \in \mathbb{R}$. Using the inner product, we can explicitly compute the Fourier coefficients as

$$\hat{f}(y) = \langle f, \chi_y \rangle = \frac{1}{2^n} \sum_{x \in \{0, 1\}^n} f(x) \chi_y(x).$$

Let's now consider what the Deutsch-Jozsa/Bernstein-Vazirani algorithm does when we expand f in the Fourier basis. Once again, explicitly computing $H^{\otimes n} O_f H^{\otimes n} |0^n\rangle$, we get

$$\frac{1}{2^n} \sum_{x, y \in \{0, 1\}^n} (-1)^{f(x) + x \cdot y} |y\rangle = \sum_{y \in \{0, 1\}^n} \left(\frac{1}{2^n} \sum_{x \in \{0, 1\}^n} (-1)^{f(x)} \chi_y(x) \right) |y\rangle = \sum_{y \in \{0, 1\}^n} \hat{F}(y) |y\rangle.$$

where we've defined the function $F(x) = (-1)^{f(x)}$ to essentially convert from the $\{0, 1\}$ outputs to $\{-1, +1\}$ outputs. In other words, what our quantum algorithm is actually doing for the Deutsch-Jozsa and Bernstein-Vazirani problems is sampling a $y \in \{0, 1\}^n$ with probability equal to $|\hat{F}(y)|^2$, the magnitude squared of the Fourier coefficients of F . Therefore, any Boolean function f that has simple Fourier expansion is immediately a promising candidate for an efficient quantum query algorithm.

3.3 Hidden subgroup problems

Let's now deviate slightly from our Fourier sampling framework to obtain a problem on which the classical algorithm will really struggle:

Simon's problem

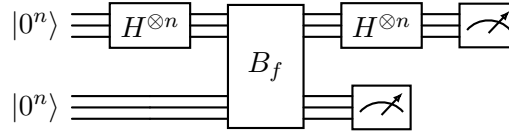
- Input:** $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$
- Promise:** Outputs of f are paired by secret $s \in \{0, 1\}^n$.
That is, $f(x) = f(y)$ iff $x \oplus y = s$.
- Question:** Determine the secret string s .

Notice that to solve Simon's problem it suffices to find a collision, a pair of strings $x \neq y$ such that $f(x) = f(y)$. If we find such an input pair, we can deduce s by taking their difference:

$$f(x) = f(y) \implies x \oplus y = s.$$

Let's first consider how efficient a classical query algorithm can be for this task. Notice that if we query random inputs, we can expect to find a collision after only $O(\sqrt{2^n})$ queries via the birthday paradox bound. In fact, this algorithm can be derandomized so that $O(\sqrt{2^n})$ queries are sufficient for a classical deterministic algorithm [CQ18]. Intuitively, after k queries, we've looked at $\binom{k}{2} \approx k^2/2$ pairs of inputs, so we need $k \approx 2^{n/2}$ queries to find one of the 2^{n-1} pairs. This same argument also suffices to give a lower bound of $\Omega(\sqrt{2^n})$ queries.

The quantum algorithm proceeds by running the following circuit $O(n)$ times:



Let's once again analyze this circuit layer by layer:

1. Apply a layer of Hadamard gates:

$$|0^n\rangle |0^n\rangle \xrightarrow{H^{\otimes n} \otimes I^{\otimes n}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0^n\rangle.$$

2. Apply the standard oracle:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0^n\rangle \xrightarrow{B_f} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$$

3. Measure the second register, getting outcome $f(x)$:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle \xrightarrow{\text{measure}} \frac{|x\rangle + |x \oplus s\rangle}{\sqrt{2}} |f(x)\rangle$$

The idea is that there are only two inputs that are consistent with a measurement of $f(x)$ in the second register, both x and $x \oplus s$. Therefore, the first register is a superposition over those inputs. We can now drop the second register since it is unentangled with the first.

4. Apply another layer of Hadamard gates:

$$\frac{|x\rangle + |x \oplus s\rangle}{\sqrt{2}} \xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} \left((-1)^{x \cdot y} + (-1)^{(x \oplus s) \cdot y} \right) |y\rangle$$

5. Measure first register to obtain uniformly random $y \in \{0, 1\}^n$ such that $y \cdot s = 0$:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} \left((-1)^{x \cdot y} + (-1)^{y \cdot (x \oplus s)} \right) |y\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} (1 + (-1)^{y \cdot s}) |y\rangle$$

From the right hand side, we can see that if $y \cdot s = 1$, the amplitude on state $|y\rangle$ is 0. On the other hand, if $y \cdot s = 0$, then the amplitude is $(-1)^{x \cdot y} / \sqrt{2^{n-1}}$. Taken together, this implies that the measurement returns a uniformly random $y \in \{0, 1\}^n$ such that $y \cdot s = 0$.

To complete the quantum algorithm for Simon's problem, we note that the measurement result y gives us a random linear equation (over $\mathbb{F}_2$), $y \cdot s = 0$. If we could collect the $n - 1$ linearly independent equations that span the subspace orthogonal to s , we could solve for the bits of s . Since our measurement results are uniformly random within this space, we will collect $n - 1$ linearly independent equations with only $O(n)$ measurements with high probability.

To conclude, we finally have a problem in which quantum computers are getting an *exponential* advantage over classical computers— $O(n)$ vs. $O(\sqrt{2^n})$ queries. In fact, Simon's problem is special case of a wider class of problems which (sometimes) admit fast quantum algorithms.

Hidden Subgroup Problem (HSP)

Input: $f: G \rightarrow \{0, 1\}^*$ where G is a group
Promise: f is constant on a hidden subgroup $H \leq G$.
 That is, $f(x) = f(y)$ iff $xH = yH$.
Question: Determine the hidden subgroup H .

Notice that Simon's problem is HSP for the group $G = \mathbb{Z}_2^n$ and the subgroup $H = \{0^n, s\}$. In fact, the discrete log problem in $\mathbb{Z}_N^\times$ (a crucial step in Shor's integer factorization algorithm) can be cast as an instance of HSP for the additive abelian group $G = \mathbb{Z}_N \times \mathbb{Z}_N$. Both of these algorithms fall within a wide class of efficiently solvable HSP instances:

Theorem 3.1 (Kitaev [Kit95]). *HSP for finite abelian groups is in quantum polynomial time.*

What about non-abelian groups? The story is surprisingly subtle. While we don't know of any efficient quantum algorithms for such groups, there are efficient algorithms as measured by the query complexity:

Theorem 3.2 (Ettinger, Høyer, Knill [EHK04]). *The query complexity of HSP for any finite group G is polynomial in $\log |G|$.*

As some small taste for the power of such HSP instances, if the Ettinger-Høyer-Knill algorithm could be made time-efficient, then there would be an efficient quantum algorithm for the graph isomorphism problem, which has long evaded fast classical techniques.

3.4 Grover’s algorithm and the unstructured search problem

So far, we’ve seen some huge quantum speedup for various query problems. Importantly, however, these exponential speedups have been for promise problems where the input instance comes from some restricted class. Let’s now move on to consider *total problems*, where the problem must be well-defined over all possible instances.

One might wonder why we cannot just take any promise problem for which a quantum computer had some kind of advantage and extend it to inputs for which it was previously undefined. Unfortunately, the issue is that we cannot easily detect the inputs for which the original promise held. Since we *must* be able to detect those inputs to answer consistently on all inputs, it’s unclear how to make such a strategy work. In fact, such a strategy provably cannot work:

Theorem 3.3 (Aaronson, Ben-David, Kothari, and Tal [ABDKT20]). *The deterministic query complexity of a total function is at most the quantum query complexity of that function to the fourth power.*

In other words, total functions can only yield polynomial query speedups. That said, the bound in the theorem is tight up to log factors [ABB⁺17]—there is a total problem on which deterministic algorithms require quartically many more queries than the best quantum algorithm. Instead of looking at total problems in general, let’s look at a specific total problem that has shaped a lot of the discussion around quantum computers.

Unstructured search

Input: $f: \{0, 1\}^n \rightarrow \{0, 1\}$

Question: Find $x \in \{0, 1\}^n$ such that $f(x) = 1$ (or report none exists)

It’s worth taking a moment to appreciate how monumental a fast quantum algorithm for unstructured search would be. Since problems in NP can be phrased as unstructured search problems (e.g., given a SAT formula, find a satisfying assignment), a poly-time quantum algorithm for unstructured search would immediately imply that $\text{NP} \subseteq \text{BQP}$. Of course, by Theorem 3.3, we already know such a simple strategy for solving NP problems won’t work. That is, since classical computers require exponentially many queries to solve unstructured search, so must quantum computers.

To see that exponentially many classical queries are required, consider the case where there is at most one input which evaluates to 1. Any classical deterministic algorithm will need to make 2^n queries since it might get unlucky and query $2^n - 1$ zeroes. Randomness doesn’t help—even if you query half of the inputs, you only have a $1/2$ chance at choosing the input that evaluates to 1.

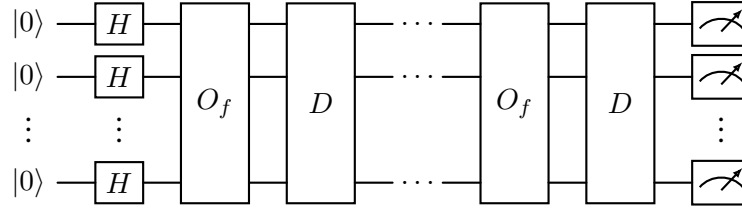
So the classical algorithm requires $\Omega(2^n)$ queries. Can we devise a quantum algorithm that does better? Somewhat shockingly, we can.

3.4.1 Grover’s algorithm

In 1996, Lov Grover surprised the quantum community when he released a $O(\sqrt{2^n})$ time quantum algorithm for unstructured search [Gro96]. That is, the algorithm only requires

$O(\sqrt{2^n})$ quantum queries, but is also similarly efficient in time. The algorithm will turn out to have quite a slick proof when viewed from a geometric perspective, which is the proof we'll describe here.

The entirety of Grover's algorithm is simply alternating between the phase oracle (i.e., O_f) and a unitary D called the “Grover diffusion operator” that we will define shortly:



Let's take a closer look at this circuit.

Grover diffusion operator

The Grover diffusion operator D has the following form:

$$D := 2|u\rangle\langle u| - I$$

where $|u\rangle := H^{\otimes n}|0^n\rangle$ is the uniform superposition. Unpacking this definition, it will often be easier to interpret D in the following way:

$$D|\psi\rangle = \begin{cases} +|\psi\rangle & \text{if } \langle\psi|u\rangle = 1 \\ -|\psi\rangle & \text{if } \langle\psi|u\rangle = 0 \end{cases}$$

To see that these are equivalent, we simply need to consider both cases. First, if we apply D to $|u\rangle$, we have

$$D|u\rangle = (2|u\rangle\langle u| - I)|u\rangle = 2|u\rangle\langle u|u\rangle - |u\rangle = 2|u\rangle - |u\rangle = |u\rangle,$$

that is, D does nothing. On the other hand, for any state $|\psi\rangle$ orthogonal to $|u\rangle$, we have

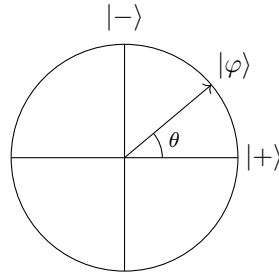
$$D|\psi\rangle = (2|u\rangle\langle u| - I)|\psi\rangle = 2|u\rangle\langle u|\psi\rangle - |\psi\rangle = -|\psi\rangle,$$

that is, D flips the sign. Since we've checked that the two definitions agree over a basis (e.g., the $\{+, -\}$ -basis), they agree everywhere.

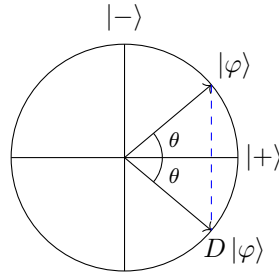
To see how the diffusion operation acts geometrically, let's consider a simplified case where we only have a single qubit. In this case $D = 2|+\rangle\langle +| - I$. Moreover, let's suppose that we apply D to a single-qubit state $|\psi\rangle$ with real amplitudes. Writing $|\psi\rangle$ in the basis of $|+\rangle$ and $|-\rangle$, we get

$$|\psi\rangle = \cos\theta|+\rangle + \sin\theta|-\rangle$$

for some angle $\theta \in [0, 2\pi)$. If, say, $\cos\theta$ and $\sin\theta$ are both positive, we can draw our state as a vector lying the first quadrant of the unit circle:



When we apply the Grover diffusion operator, our state becomes $D|\psi\rangle = \cos\theta|+\rangle - \sin\theta|-\rangle$, or other words, we flip the sign of the $|-\rangle$ component. In our picture, this is going to reflect the state about the horizontal axis, because the component along the vertical axis has been negated:



Unsurprisingly, the behavior that we see for a single-qubit applies more generally:

Claim 3.4. *The diffusion operator $D := 2|u\rangle\langle u| - I$ is a unitary operation that reflects¹ about $|u\rangle$. Furthermore, D can be constructed with linearly-many gates in log depth.*

Proof. To verify that D is unitary, we can simply compute

$$DD^\dagger = (2|u\rangle\langle u| - I) \cdot (2|u\rangle\langle u| - I)^\dagger = 4|u\rangle\langle u| - 2|u\rangle\langle u| - 2|u\rangle\langle u| + I = I.$$

To see why D is a reflection about $|u\rangle$, first notice that we can decompose an arbitrary state $|\psi\rangle$ as its component aligned with $|u\rangle$ and its component orthogonal to $|u\rangle$.

$$|\psi\rangle = \alpha|u\rangle + \beta|v\rangle,$$

where $\langle u|v\rangle = 0$ and $|\alpha|^2 + |\beta|^2 = 1$. Then, we can verify

$$D|\psi\rangle = \alpha(2(|u\rangle\langle u| - I)|u\rangle) + \beta(2(|u\rangle\langle u| - I)|v\rangle) = \alpha|u\rangle - \beta|v\rangle,$$

where we use the fact that $\langle u|u\rangle = 1$ and $\langle u|v\rangle = 0$.

To see that D can be constructed with linearly-many gates in log depth, notice that if we conjugate D by Hadamard, we get the reflection about the all-zeros state: $D_0 = 2|0^n\rangle\langle 0^n| - I$. Therefore, we just need a circuit for D_0 . On the computational basis states, we have $D_0|x\rangle = (-1)^{x_1 \vee \dots \vee x_n} |x\rangle$ so we just need to be able to detect if any of the qubits are 1 (which can be done with a linear-size, log-depth reversible circuit) and apply a phase gate depending on the answer. $\square$

¹By “reflect” about $|u\rangle$, we mean that D flips the sign of every vector in the subspace orthogonal to $|u\rangle$.

Algorithm 1 Grover's algorithm**Input:** 2^n unknown input bits accessed through the oracle O_f .**Output:** $s \in \{0, 1\}^n$ such that $f(s) = 1$, or null if none exists.

- 1: $|\psi_0\rangle = H^{\otimes n} |0^n\rangle$
- 2: **for** $i \in \{1, \dots, T\}$ **do**
- 3: $|\psi_i\rangle \leftarrow DO_f |\psi_{i-1}\rangle$
- 4: $s^* \leftarrow$ measurement of $|\psi_T\rangle$
- 5: **return** s^* if $f(s^*) = 1$; otherwise, null

The simplest version of Grover's algorithm depends on the number of *marked items*, that is, inputs x such that $f(x) = 1$. Therefore, let's assume for now that there is only a single marked item. We will see in the analysis that this is the “hard” case.

To understand why this algorithm works, we will use the geometric perspective. To start, notice that our initial state $|u\rangle$ lies in a particular 2-dimensional subspace that is spanned by $|s\rangle$ (our marked item) and $|\Psi\rangle = \frac{1}{\sqrt{2^n-1}} \sum_{x \neq s} |x\rangle$ (the uniform superposition over all unmarked states):

$$|u\rangle = \frac{1}{\sqrt{2^n}} \sum_x |x\rangle = \frac{1}{\sqrt{2^n}} |s\rangle + \frac{1}{\sqrt{2^n}} \sum_{x \neq s} |x\rangle = \frac{1}{\sqrt{2^n}} |s\rangle + \sqrt{1 - \frac{1}{2^n}} |\Psi\rangle.$$

First, we make the following intriguing observation about each Grover iteration (i.e., DO_f) in the algorithm:

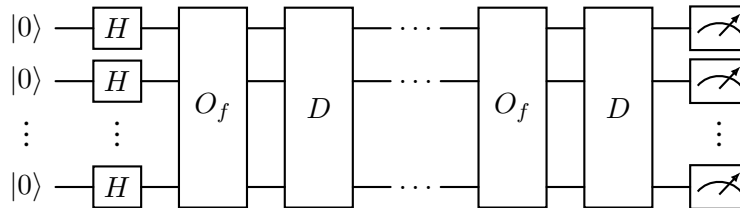
Observation 3.5. *Each Grover iteration keeps the state in the span of $|s\rangle$ and $|\Psi\rangle$.*

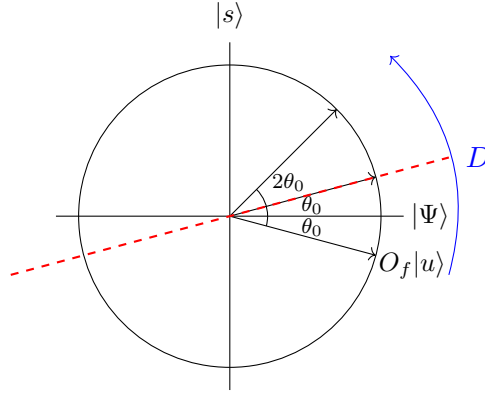
Proof. This is easy to see for the phase oracle: $\alpha |s\rangle + \beta |\Psi\rangle \xrightarrow{O_f} -\alpha |s\rangle + \beta |\Psi\rangle$. For the diffusion operator, we have

$$\alpha |s\rangle + \beta |\Psi\rangle \xrightarrow{D} (2|u\rangle\langle u| - I)(\alpha |s\rangle + \beta |\Psi\rangle) = 2(\alpha \langle u|s\rangle + \beta \langle u|\Psi\rangle) |u\rangle - \alpha |s\rangle - \beta |\Psi\rangle$$

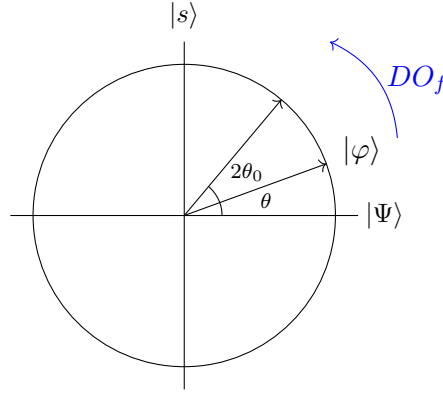
but we've already seen above that $|u\rangle$ can be expressed a linear combination of $|s\rangle$ and $|\Psi\rangle$. $\square$

In other words, each Grover operation is a rotation in the plane spanned by $|s\rangle$ and $|\Psi\rangle$. We have that O_f reflects about $|\Psi\rangle$, and the diffusion operation reflects about $|u\rangle$:





If we compose the two operations (i.e., DO_f) and apply them to any arbitrary state $|\varphi\rangle$, we simply get a rotation in this space of $2\theta_0$, where θ_0 is the initial angle between $|u\rangle$ and $|\Psi\rangle$:



That is, the evolution of the angle is given by $\theta_0, 3\theta_0, 5\theta_0, \dots, (2T+1)\theta_0$. Notice that we want to reach the angle $\pi/2$, so we get that we need $T \approx \pi/(4\theta_0)$ steps. In other words, performance of the entire algorithm hinges on the angle θ_0 between our initial state $|u\rangle$ and the unmarked state $|\Psi\rangle$. We have

$$\sin(\theta_0) = \langle u|s \rangle = \frac{1}{\sqrt{2^n}} \implies \theta_0 \approx \frac{1}{\sqrt{2^n}}$$

where we have used that $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$ is approximately x for small x . Therefore, to rotate our initial state to the $|s\rangle$ state we need $T = O(\sqrt{2^n})$. After that many queries, we simply measure to obtain the marked state s with high probability.

Of course, this analysis only holds if there was indeed a marked element. However, after we've done this procedure, we measure to obtain some candidate marked item s^* . We can use one more query to our oracle to check that $f(s^*) = 1$. This completes the analysis of Grover's algorithm for a single marked element.

What happens if there are more than 1 marked items? In this case, let $|s\rangle$ be the uniform superposition over all marked items. If we have m marked elements, then initial angle is $\langle s|u \rangle \approx \sqrt{m/2^n}$ at least when there aren't too many marked items (if there are so many marked items, we can just randomly sample until we find one). Therefore, with the same analysis, the number of queries required to rotate our state to $|s\rangle$ is $O(\sqrt{2^n/m})$. When we

measure, we get a uniformly random marked item. This speedup follows our intuition that if there are more marked elements, it should be easier to find one of them.

There is one final question to address. Namely, the above analysis only works when we know the number marked elements. Indeed, if we continue to do more Grover iterations, then our state continues to rotate around the unit circle. If the number of marked items is unknown, how do we know when to stop and measure? The trick is something called “exponential search.” We make the following sequence of guesses for m : $2^n, 2^{n-1}, 2^{n-2}, \dots, 4, 2$. Notice that if we make all n possible guesses, then we are at most a factor of 2 off from the true answer. One can check that this does not dramatically affect the analysis. The reason that we search in decreasing order is because we want to obtain a speedup in the case that there are actually many marked items. If at any point we find a marked item, then we stop.

3.4.2 Consequences of Grover’s algorithm

Consider a variant of Simon’s problem:

Collision	
<i>Input:</i>	$f: \{0, 1\}^n \rightarrow \{0, 1\}^n$
<i>Promise:</i>	f is 1-to-1 or 2-to-1
<i>Question:</i>	Decide which

Fact 3.6. Suppose f is 2-to-1. Then for randomly chosen $A, B \subseteq \{0, 1\}^n$ with $|A||B| = 2^n$ there is a constant probability that there exists $a \in A$ and $b \in B$ such that $f(a) = f(b)$.

Theorem 3.7 (Brassard, Høyer, and Tapp [BHT97]). The quantum query complexity of the Collision problem is $O(2^{n/3})$.

Proof. Pick a random A of size $2^{n/3}$ and B of size $2^{2n/3}$. First query each element of A , which takes $2^{n/3}$ queries. With this, construct the (single query) function $g(x)$ which returns true if there is $a \in A$ with $f(x) = f(a)$. Now run Grover’s algorithm on B , to see if g is ever true. This takes $O(\sqrt{2^{2n/3}}) = O(2^{n/3})$ queries. $\square$

3.4.3 BBBV lower bound for search

While Theorem 3.3 leaves open the possibility that unstructured search can be solved with $O(2^{n/4})$ queries, this is unfortunately still too optimistic.

Theorem 3.8 (Bennett, Bernstein, Brassard, Vazirani [BBBV97]). The quantum query complexity of unstructured search is $\Omega(\sqrt{2^n})$.

As we will see later, there are actually many possible ways to prove this lower bound, but the BBBV lower bound was the first and perhaps most intuitive lower bound technique, so let’s start with that. First, notice that a generic quantum query algorithm alternates between applying some unitary and applying the oracle. In other words, after t queries, the state of our system looks like

$$U_t O_f U_{t-1} \cdots O_f U_1 O_f U_0 |0^n\rangle.$$

To be fully rigorous here, we would also need to specify a set of ancillary workspace qubits, but this will not change the analysis and only make the notation more cumbersome, so we will drop these extra qubits.

A key point about this decomposition is that the unitaries $U_0, U_1, \dots, U_t$ are fixed and are independent of what the oracle does. When there are few oracle queries, our goal will be to show that for every choice of unitaries, there is some state $|y\rangle$ that always has small amplitude when queried by the oracle. Because of this, it will be very difficult for the algorithm to “see” whether or not this item is marked. Therefore, we can fool the algorithm into accepting/rejecting when it shouldn’t.

Let’s first consider what our algorithm does on the constant-zero function. In this case, the oracle is just the identity, and the algorithm should reject. The state of the algorithm after t queries is

$$|\psi_t\rangle := U_t U_{t-1} \cdots U_1 U_0 |0^n\rangle = \sum_{x \in \{0,1\}^n} \alpha_{x,t} |x\rangle.$$

Supposing there are T total queries, define the quantity

$$m_x := \sum_{t=0}^{T-1} |\alpha_{x,t}|^2.$$

to be the sum of the squares of the magnitudes on x over all states $|\psi_t\rangle$ we have right before the t th oracle call. We have that

$$\sum_{x \in \{0,1\}^n} m_x = \sum_{x \in \{0,1\}^n} \sum_{t=0}^{T-1} |\alpha_{x,t}|^2 = \sum_{t=0}^{T-1} \left(\sum_{x \in \{0,1\}^n} |\alpha_{x,t}|^2 \right) = \sum_{t=0}^{T-1} 1 = T.$$

Since m_x is non-negative, this implies that there must exist some $y \in \{0,1\}^n$ such that $m_y \leq T/2^n$ (otherwise, the sum is greater than T). This y will be the element that the algorithm fails to properly consider if T is too small. The above argument gives us a bound on the sum of the squares of the magnitudes for the input y , but it will turn out that we will actually need a bound on the sum of the magnitudes themselves. Fortunately, by Cauchy-Schwarz, we have

$$\sum_{t=0}^T |\alpha_{y,t}| \leq \sqrt{\sum_{t=1}^T |\alpha_{y,t}|^2 \cdot T} = \sqrt{m_y T} \leq \frac{T}{\sqrt{2^n}}.$$

Since we can refer to the all-zeros function as the identity, let f be the function which is 1 on y and 0 elsewhere. Our goal is to distinguish the oracle for f from the oracle for the identity, but for the purposes of analysis, let’s consider a set of rather strange oracles $\{O^{(t)}\}_{t=0}^T$. Here, $O^{(t)}$ is defined to be the identity for the first t queries and f on the remaining $T - t$ queries. In other words, the oracle is interpolates between our two function instances. Let’s define the set of states arising from the application of these oracles as

$$|\varphi^{(t)}\rangle := U_T O^{(t)} U_{T-1} \cdots O^{(t)} U_1 O^{(t)} U_0 |0^n\rangle = U_T O_f U_{T-1} \cdots O_f U_{t+1} O_f |\psi_t\rangle$$

So, for example, we have that $|\varphi^{(T)}\rangle = |\psi_T\rangle$ is the state for the complete execution of the quantum algorithm for the constant-zero function, and $|\varphi^{(0)}\rangle$ is the state for the execution of the quantum algorithm for f .

If we can show that $|\varphi^{(t+1)}\rangle$ is close to $|\varphi^{(t)}\rangle$ for all t , then by the triangle inequality, we will be able to conclude that the states from the two different problem instances are also close to each other. We have the following:

$$\begin{aligned} \||\varphi^{(t+1)}\rangle - |\varphi^{(t)}\rangle\| &= \|U_T O_f U_{T-1} \cdots O_f U_{t+2} O_f |\psi_{t+1}\rangle - U_T O_f U_{T-1} \cdots O_f U_{t+1} O_f |\psi_t\rangle\| \\ &= \|(U_T O_f U_{T-1} \cdots O_f U_{t+2} O_f U_{t+1}) |\psi_t\rangle - (U_T O_f U_{T-1} \cdots O_f U_{t+1}) O_f |\psi_t\rangle\| \\ &= \||\psi_t\rangle - O_f |\psi_t\rangle\| \\ &= 2|\alpha_{y,t}| \end{aligned}$$

where we have used the fact that unitaries preserves the 2-norm and the fact that $O_f |\psi_t\rangle = |\psi_t\rangle - 2\alpha_{y,t} |y\rangle$. Combining everything together, we get

$$\||\varphi^{(T)}\rangle - |\varphi^{(0)}\rangle\| \leq \sum_{t=0}^{T-1} \||\varphi^{(t+1)}\rangle - |\varphi^{(t)}\rangle\| \leq 2 \sum_{t=0}^{T-1} |\alpha_{y,t}| \leq \frac{2T}{\sqrt{2^n}}.$$

Hence, we see that for $T \ll \sqrt{2^n}$, the two states are close under ℓ_2 norm. We want to show that if the states are close, then all measurement procedures fail to distinguish them with high probability. To formalize this, let us define the *total variation distance* between two discrete probability distributions p, q :

$$\text{TV}(p, q) = \frac{1}{2} \|p - q\|_1 = \frac{1}{2} \sum_i |p_i - q_i|.$$

The total variation distance is important because it determines the maximum probability with which we can distinguish two probability distributions. That is, suppose with 50% probability we sample from p and with 50% probability we sample from q , the maximum probability with which we can guess which distribution was sampled from is $1/2 + \text{TV}(p, q)/2$.

Lemma 3.9. *If $\||\phi\rangle - |\psi\rangle\|_2 < \epsilon$, then the total variation distance from measuring $|\phi\rangle$ and $|\psi\rangle$ is at most 2ϵ .*

Proof. Suppose $|\phi\rangle = \sum \alpha_x |x\rangle$, $|\psi\rangle = \sum \beta_x |x\rangle$. For ease of notation, assume α_x, β_x are all real numbers, though the proof still works if we allow them to be complex. Let $\gamma_x = \beta_x - \alpha_x$. Now we write

$$\||\phi\rangle - |\psi\rangle\|_2 = \sqrt{\sum_x \gamma_x^2} \leq \epsilon.$$

Let p, q be the distributions of measuring $|\phi\rangle, |\psi\rangle$ respectively. Then, we have that (twice) their total variation distance is

$$\begin{aligned} \sum_x |\alpha_x^2 - \beta_x^2| &= \sum_x (\beta_x - \alpha_x)(\beta_x + \alpha_x) \\ &= \sum_x \gamma_x (\gamma_x + 2\alpha_x) \\ &\leq \sum_x \gamma_x^2 + 2|\gamma_x \alpha_x| && \text{(triangle inequality)} \\ &\leq \|\gamma\|_2^2 + 2\|\gamma\|_2 \|\alpha\|_2 && \text{(Cauchy-Schwarz)} \\ &\leq \epsilon^2 + 2\epsilon, \end{aligned}$$

which is at most 4ϵ since $\epsilon \leq 2$ by the triangle inequality ($\| |\phi\rangle - |\psi\rangle \|_2 \leq \| |\phi\rangle \|_2 + \| |\psi\rangle \|_2 = 2$). Hence the TV distance is at most 2ϵ . $\square$

Putting everything together, we have shown that for any quantum algorithm with T queries, there is a state we should accept and one we should reject which we can distinguish with probability at most $\frac{1}{2} + \frac{2T}{\sqrt{2^n}}$. To correctly answer at least $2/3$ of the time, this must be at least a constant larger than $1/2$, which requires $T = \Omega(2^{n/2})$.

Chapter 4

Shor's algorithm

One of the crowning achievements in all of quantum computing is the discovery that there are efficient quantum circuits to factor integers:

Integer Factorization

Input: Integer $N = pq$ where p and q are prime numbers

Output: Prime factors p and q

The algorithm for this problem is quite involved, so we will develop our intuition for it one step at a time. To start, since the Integer Factorization problem is a problem about integers, we need to get in the habit of interpreting the classical basis states as integers.

4.1 Representing integers with qubits

For many of the query algorithms we've developed so far, the classical basis states appearing in the computations of our quantum circuits directly correspond to the inputs of a Boolean function. A common motif is to prepare a superposition over all bit strings and then apply a function $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$ using the standard oracle on all inputs:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0^n\rangle \xrightarrow{B_f} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$$

For problems over integers, the inputs to our functions will (naturally) be integers. For example, a superposition of N integers would look like

$$\frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} |k\rangle.$$

To get superpositions like the one above, will just interpret every binary string $x \in \{0, 1\}^n$ as some integer $k \in \{0, \dots, 2^n - 1\}$ using its binary expansion:

$$x = x_{n-1}x_{n-2} \dots x_1x_0 \implies k = x_02^0 + x_12^1 + \dots + x_{n-2}2^{n-2} + x_{n-1}2^{n-1}$$

4.1.1 The complexity of integer operations

Thankfully, we won't have to dig into the details too much of how the binary representation affects our computation. We will rely heavily on our intuition of which sorts of operations on integers are and are not efficiently computable. For example, adding two integers together is an eminently easy operation to perform classically. We could define the function that adds two n -bit integers $\text{Add}: \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$:

$$\text{Add}(s, t) = s + t$$

for any integers $s, t \in \{0, \dots, 2^n - 1\}$. Since the adding function is not reversible (e.g., 4 is both $1 + 3$ and $2 + 2$), we will once again need to use the standard oracle to turn it into a valid unitary:

$$|s\rangle |t\rangle |b\rangle \xrightarrow{B_{\text{Add}}} |s\rangle |t\rangle |b \oplus (s + t)\rangle.$$

Because the Add function is easy to compute classically, it will turn out that B_{Add} will be easy to compute quantumly. If we wanted to make this kind of statement more formal, we would start with the following fact about classical circuits:

Fact 4.1. *Every Boolean function $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$ can be computed by a circuit of AND, OR, and NOT gates.*

The *gate complexity* of a Boolean function is the number of AND, OR, and NOT gates required for some classical circuit to compute that function. In particular, we will say that a function $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$ is efficiently computable if the number of gates required to compute the function is polynomial in the input and output size (i.e., n and m).

Luckily for us, the gate complexity of a classical circuit computing a Boolean function will directly translate to the gate complexity of a quantum circuit computing the standard oracle for that function:

Fact 4.2. *For any function $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$, the quantum gate complexity of computing the standard oracle B_f is at most linear in the classical gate complexity of computing f .*

Using the correspondence between binary strings and integers, we can conclude that if performing some classical operation on integers is efficient, then performing the same operation quantumly (through the standard oracle) is similarly efficient.

4.1.2 Modular arithmetic

Notice that when we defined the add function $\text{Add}: \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$, the output register needed to have $n + 1$ bits because when we add two integers together, the integer can double in size. Indeed, if we wanted to keep adding integers, the size of our register would need to grow and grow to accommodate larger and larger integers. While it may not be obvious now, this kind of unbounded growth tends to be bad for quantum algorithms since the underlying operation does not form a group in the formal mathematical sense.

To fix this, we define $\text{Add}_N: \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ to be addition modulo some integer N :

$$\text{Add}_N(s, t) = s + t \pmod{N}$$

Since it will be central to much of what we look at next, let's review some of the basics of modular arithmetic. First, for any integer $s \geq 0$, we define $s \bmod N$ to be the remainder of s when divided by N . That is, if we were to write $s = qN + r$ for some integers $q, r \geq 0$ and $b < N$, then

$$s \bmod N = r.$$

One of the nicest properties of the mod operation is that it is compatible with addition and multiplication in the following sense: for any integers $s, t \geq 0$

$$\begin{aligned}(s + t) \bmod N &= ((s \bmod N) + (t \bmod N)) \bmod N \\ (st) \bmod N &= ((s \bmod N)(t \bmod N)) \bmod N\end{aligned}$$

In other words, if you're planning to take the mod of the output of a sequence of operations, you might as well also take the mod of the inputs. Notice that this can dramatically shrink the number of qubits you need for some computations because each mod operation keeps the integer from getting too large.

We will use the notation $\mathbb{Z}_N = \{0, 1, 2, \dots, N - 1\}$ to denote the integers mod N .

4.2 Outline of Shor's algorithm

As it turns out, Shor's famous quantum algorithm for integer factorization is actually a quantum algorithm for a seemingly distinct problem called "order finding":

Order Finding

Input: $a \in \mathbb{Z}_M$ such that $\gcd(a, M) = 1$.

Output: Smallest nonnegative integer r such that $a^r \equiv 1 \pmod{M}$.

For example, suppose $a = 5$ and $M = 16$. We can compute the order of $a \bmod M$ by a somewhat brute force algorithm: repeatedly multiply a by itself until it is equal to 1 mod M :

$$\begin{aligned}a &\equiv 5 \\ a^2 &\equiv 5 \cdot 5 = 25 \equiv 9 \pmod{16} \\ a^3 &\equiv 9 \cdot 5 = 45 \equiv 13 \pmod{16} \\ a^4 &\equiv 13 \cdot 5 = 65 \equiv 1 \pmod{16}\end{aligned}$$

Here, you can see that the order is 4. The reason order finding is an attractive candidate for admitting a fast quantum algorithm is due to the fact that it possesses a certain kind of periodic structure. To see this, consider the sequence obtained by repeatedly multiplying by a , starting from 1. Using our example above, a has order 4, so we get the sequence of integers:

$$1 \rightarrow a \rightarrow a^2 \rightarrow a^3 \rightarrow a^4 \equiv 1 \rightarrow a \rightarrow a^2 \rightarrow a^3 \rightarrow 1 \rightarrow a \rightarrow a^2 \rightarrow \dots$$

The sequence repeats every 4 steps. If we could build a quantum algorithm to detect periodicities like these, then we could solve the order finding problem! This kind of periodicity will show up in a toy version of the order finding algorithm, which is a direct analog of Simon's algorithm. We explore this problem in the following section.

What ties this all together, of course, is the following remarkable theorem:

Theorem 4.3. *If there is an efficient algorithm to solve order finding, then there is also an efficient algorithm to factor integers.*

The proof of this theorem uses some standard number theory ideas, that we will omit for now.

4.3 Simon's problem over the integers

To understand Simon's problem over the integers, let's first revisit the [original formulation](#): given query access to some Boolean function $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$, you are promised that there is some secret string $s \in \{0, 1\}^n$ such that $f(x) = f(y)$ iff $x \oplus y = s$. In other words, if we start with some input string $x \in \{0, 1\}^n$, we can compute the bitwise XOR with the string s to get another string y such that $f(y) = f(x)$. We could continue this forever, taking the bitwise XOR in each iteration to get another input with output value $f(x)$. Of course, in this example, we just alternate between the inputs x and $x \oplus s$ since $x \oplus s \oplus s = x$. It turns out that it is exactly this kind of additive repeating pattern that quantum algorithms are good at detecting, and we can generalize Simon's problem to the integers below:

Simon's problem over $\mathbb{Z}_N$

Input: $f: \mathbb{Z}_N \rightarrow \{0, 1\}^m$
Promise: f has a secret period $s \in \mathbb{Z}_N$ with $s \neq 0$.
 That is, $f(x) = f(y)$ iff $y = x + as$ for some $a \in \mathbb{Z}_N$
Question: Determine the secret period s .

As an example, consider the following function f with $N = 16$ and $s = 4$ where we've colored the outputs of f that are the same to highlight the periodic structure:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f(x)$	00	11	01	10	00	11	01	10	00	11	01	10	00	11	01	10

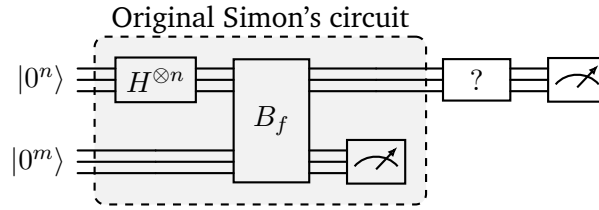
You might have noticed in this example that $s = 4$ divides $N = 16$. This is not a coincidence. The smallest period s will always divide N . Suppose we set $s = 6$ and $N = 16$. We will see that there is a smaller period than 6. Let's start with the value at 0. We know that if $s = 6$, then $f(0) = f(0 + 6) = f(0 + 2 \cdot 6) = \dots$. Let's look at all the distinct numbers we get by iterating this procedure. We get

$$0 \rightarrow 6 \rightarrow 12 \rightarrow 18 = 2 \rightarrow 8 \rightarrow 14 \rightarrow 20 = 4 \rightarrow 10 \rightarrow 16 = 0$$

Examining these numbers, we see that all even numbers are present, so in fact the smallest period of this function is $s = 2$. More generally, it will turn out that if s is a period of f , then the greatest common divisor of s and N (denoted $\gcd(s, N)$) is also a period of f . This fact follows from some basic number theory that we leave as an exercise.

One interesting takeaway from the above discussion is that the problem defined above is not actually exponentially difficult for classical computers when $N = 2^n$, which is the first setting in which we will demonstrate a quantum algorithm. To see this, notice that we can just guess that the period is $1, 2, 4, 8, \dots$ until we find it. This will only require n queries. Nevertheless, let's go ahead and describe the quantum algorithm for this problem anyways because it will contain many of the right ideas we will need for Shor's algorithm.

As you might suspect, the quantum algorithm for Simon's problem over $\mathbb{Z}_N$ takes a very similar form to our original algorithm for our original Simon's problem over $\mathbb{Z}_2^n$. The circuit is almost identical gate-for-gate except for the Fourier basis change $H^{\otimes n}$, which we replace with a mysterious box for now:



Let's now repeat the Simon's analysis for the shaded region using the fact that our oracle now operates over integer values. We first apply a layer of Hadamard gates to get superposition of inputs (note $N = 2^n$) and then apply our periodic function f in superposition:

$$|0^n\rangle |0^m\rangle \xrightarrow{H^{\otimes n} \otimes I} \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |0^m\rangle \xrightarrow{B_f} \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |f(x)\rangle$$

We now measure the second register to get some function value $f(x)$, which means the first register will be a superposition of inputs y such that $f(y) = f(x)$. Because f has period s , these are the inputs $x, x + s, x + 2s$, and so on. We showed earlier that the smallest period must divide N , so there are a total of N/s distinct inputs whose outputs are $f(x)$:

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |f(x)\rangle \xrightarrow{\text{measure}} \sqrt{\frac{s}{N}} \sum_{i=0}^{N/s-1} |x + is\rangle |f(x)\rangle$$

We can now drop the second register since it is unentangled with the first. How can we extract the period from this state? For Simon's algorithm over $\mathbb{Z}_2^n$, we applied a layer of Hadamard gates to perform the Boolean Fourier transform. If only we had an analogue of the Boolean Fourier transform over the integer domain...

4.3.1 The Quantum Fourier Transform

In general, Fourier transforms convert between two bases: if there is some kind of periodic structure in some basis, then that information may be revealed by computing the Fourier transform. Perhaps unsurprisingly, then, Quantum Fourier Transform (QFT) uses numbers that are also periodic in nature, the complex roots of unity:

$$\omega_N := e^{\frac{2i\pi}{N}} = \cos\left(\frac{2\pi}{N}\right) + i \sin\left(\frac{2\pi}{N}\right)$$

A review of complex numbers and roots of unity can be found in Chapter A. For now, let's jump straight to the definition of the quantum Fourier transform:

$$\text{QFT}_N |x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{xy} |y\rangle$$

Of course, simply defining some useful multi-qubit unitary is insufficient to create efficient quantum algorithms. Quite remarkably, however, there is an efficient quantum circuit for the QFT:

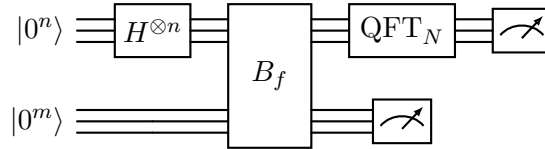
Theorem 4.4. *There is a quantum circuit with $O(n^2)$ gates that computes QFT_{2^n} .*

It can be useful to visualize the quantum Fourier transform as a matrix (for clarity, we just write ω for ω_N):

$$\frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{(N-1)} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}$$

4.3.2 Using QFT to solve Simon's problem

Let's now use the quantum Fourier transform to solve Simon's problem over $\mathbb{Z}_N$. Filling in the QFT for the "?" in the the Simon's circuit from before, we have



Starting with the superposition over states mapping to the same output in the first register, we now apply the Fourier transform:

$$\sqrt{\frac{s}{N}} \sum_{i=0}^{N/s-1} |x + is\rangle \xrightarrow{\text{QFT}_N} \frac{\sqrt{s}}{N} \sum_{i=0}^{N/s-1} \sum_{y=0}^{N-1} \omega_N^{(x+is)y} |y\rangle = \frac{\sqrt{s}}{N} \sum_{y=0}^{N-1} \omega_N^{xy} \left(\sum_{i=0}^{N/s-1} \omega_N^{isy} \right) |y\rangle$$

Let's now consider the probability on those state $|y\rangle$ such that y is a multiple of N/s . In other words, $y = k(N/s)$ for some nonnegative integer k . Recall that we've already shown that s divides N , so y will always be an integer. We get that the probability of measuring state $|kN/s\rangle$ is

$$\left| \frac{\sqrt{s}}{N} \omega_N^{x(kN/s)} \left(\sum_{i=0}^{N/s-1} \omega_N^{is(kN/s)} \right) \right|^2 = \frac{s}{N^2} \left| \sum_{i=0}^{N/s-1} (\omega_N^N)^{ik} \right|^2 = \frac{s}{N^2} \cdot \frac{N^2}{s^2} = \frac{1}{s}$$

where we've crucially used that ω is a N th root of unity: $\omega_N^N = 1$. Notice that the above calculation was independent of the choice of k . Since there are exactly s choices of k , we immediately conclude that the probability of measuring any other state is 0.

We now know that the measurement outcome is kN/s , but crucially we don't actually know the value of k , so we can't use it to solve directly for s . We *do* know the value of N since it's the size of our domain: $N = 2^n$. So, if we could somehow determine N/s , then could use our knowledge of N to solve for s .

To calculate N/s , we will repeat the above algorithm several times to get outcomes:

$$\frac{k_1 N}{s}, \frac{k_2 N}{s}, \frac{k_3 N}{s}, \dots$$

Once again, we don't know the precise values of any of the k_i , so we will need to use some number theory tools to help deduce them. Since N/s is an integer, the prime factorization of all $k_i N/s$ terms will contain the same primes in the factorization of N/s . The prime factorization of the k_i integers, however, will be different with high probability. Therefore, to compute N/s , we look at the prime factorization of each $k_i N/s$ term and only keep the prime factors that are common to all of them. The following fact shows that this process will succeed with high probability:

Fact 4.5. *For any $n \geq 0$, there is at least 40% probability that two randomly sampled integers in $\{0, 1, \dots, n-1\}$ do not share any common prime factors.*

Therefore, after sampling just a few $k_i N/s$ terms, we can be very certain that the only terms that are common to all of them are the ones belonging to N/s . To complete the proof, we just need to show that picking out the common factors from the $k_i N/s$ terms can be done efficiently. Indeed, a beautiful number theory algorithm guarantees that this is possible:

Theorem 4.6 (Euclidean algorithm). *The greatest common divisor of any two n -bit integers can be computed in time $O(n^2)$.*

It's worth emphasizing a key conceptual point from the Euclidean algorithm. While it is widely believed that factoring an integer is difficult for a classical computer (indeed, this is why Shor's algorithm is so monumental!), the process of finding common factors between integers is actually easy for a classical computer. Here, we are exploiting the fact that after we run our quantum algorithm, we can post-process the output with a classical algorithm to find these common factors and ultimately solve the problem.

4.4 Order finding: It's not just a toy problem any more

Now that we've done the hard work to build some intuition for why quantum algorithms are good at finding periodicities, let's use those ideas to solve the order finding problem. At a high level, order finding is just Simon's problem with the exception that we can no longer guarantee that s cleanly divides N . To see this, let's first talk about how we are going to create our periodic function $f: \mathbb{Z}_N \rightarrow \mathbb{Z}_M$ that we will use as an oracle in the Simon's circuit. Recall that in the order finding problem we are trying to find the order of $a \bmod M$, so we will make our function f depend on a :

$$f(x) = a^x \pmod{M}$$

Assuming the order of a modulo M is r , our function will have period r :

x	0	1	2	...	$r-1$	r	$r+1$	$r+2$	...	$2r-1$	$2r$	$2r+1$	$2r+2$	...
$f(x)$	1	a	a^2	...	a^{r-1}	1	a	a^2	...	a^{r-1}	1	a	a^2	...

In other words, if we could design an algorithm to determine the period of f , that immediately gives us an algorithm to find the order of a .

Unfortunately, we can now see where we depart from Simon's problem. In Simon's problem, all outputs of f have the same number of inputs. In contrast, we will show that some outputs of f for the order finding problem are slightly more common than other outputs. To see this, consider the input $N-1$, which is the largest possible input to f . We have that $f(N-1) = a^{N-1}$. Unless we get extremely lucky, the order r will not divide N . This will mean that a^{N-1} will not be equivalent to a^{r-1} modulo M . Since the function obeys the repeating pattern $1, a, a^2, \dots, a^{r-1}$, this means that the elements in the first part of the sequence are slightly more overrepresented in the output than the elements later in the sequence.

Here's an example of this phenomenon when $N = 16$, $M = 21$, and $a = 2$:

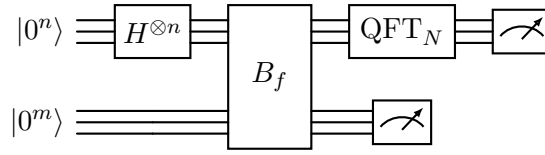
x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$2^x \bmod 21$	1	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8

Dealing with this imbalance will be surprisingly unpleasant. However, we can already see some intuitive ways to mitigate its effect. As we make N larger, the imbalance becomes less pronounced. Let's do a concrete calculation and calculate how many times a^x appears as an output of f for $x \in \{0, 1, \dots, r-1\}$. Dividing up N into blocks of size r , we can see that a^x either appears $D_x := \lfloor N/r \rfloor + \delta_x$ times where $\delta_x \in \{0, 1\}$. As N gets larger, the fraction of inputs that have output a^x approaches $1/r$:

$$\lim_{N \rightarrow \infty} \frac{\lfloor N/r \rfloor + \delta_x}{N} = \frac{1}{r}.$$

4.4.1 Analyzing Shor's algorithm for order finding

Let's now formally analyze the quantum algorithm for order finding. We will set the specific value of N later, but let's keep our intuitive understanding that the larger N is, the more likely our algorithm is to succeed. First, we need to properly define what our algorithm for order finding is going to be. In fact, the circuit we use is identical to the one we used for Simon's problem!



Carrying out the same first steps of the algorithm, we get

$$|0^n\rangle |0^m\rangle \xrightarrow{H^{\otimes n} \otimes I} \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |0^m\rangle \xrightarrow{B_f} \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |f(x)\rangle$$

When we measure the second register, we obtain some function value a^x for some $x \in \{0, 1, \dots, r-1\}$. What remains in the first register is a superposition over all inputs that have that function value (recall, $D_x := \lfloor N/r \rfloor + \delta_x$):

$$\frac{1}{\sqrt{D_x}} \sum_{i=0}^{D_x-1} |x + ri\rangle.$$

When we apply the Fourier transform, we get

$$\frac{1}{\sqrt{D_x}} \sum_{i=0}^{D_x-1} |x + ri\rangle \xrightarrow{\text{QFT}_N} \frac{1}{\sqrt{ND_x}} \sum_{i=0}^{D_x-1} \sum_{y=0}^{N-1} \omega_N^{(x+ir)y} |y\rangle = \frac{1}{\sqrt{ND_x}} \sum_{y=0}^{N-1} \omega_N^{xy} \left(\sum_{i=0}^{D_x-1} \omega_N^{iry} \right) |y\rangle$$

From here, we would like to mirror the analysis of Simon's problem and consider outputs of the form $y = k(N/r)$ so that the N part rotates ω_N back to 1. Unfortunately, N/r is no longer guaranteed to be an integer, so $y = k(N/r)$ may not even be a valid output. Instead, we look at the probability that y is close to some value kN/r .

Claim 4.7. For each $k \in \{0, 1, \dots, r-1\}$, the probability of measuring y such that

$$\left| y - \frac{kN}{r} \right| \leq \frac{1}{2}$$

is at least $O(1/r)$.

Proof. To prove this claim, let's first look at the probability of measuring any output y :

$$\left| \frac{\omega_N^{xy}}{\sqrt{ND_x}} \sum_{i=0}^{D_x-1} \omega_N^{iry} \right|^2 = \frac{1}{ND_x} \left| \sum_{i=0}^{D_x-1} \omega_N^{iry} \right|^2 = \frac{1}{ND_x} \left| \sum_{i=0}^{D_x-1} (\omega_N^{ry})^i \right|^2$$

The key to analyzing the above sum is to view the term in the absolute value as a geometric series:

$$1 + \alpha + \alpha^2 + \alpha^3 + \dots$$

where $\alpha = \omega_N^{ry}$. The idea will be to apply the formula for a geometric series, but we need to be slightly careful here because the formula only applies when $\alpha \neq 1$. However, since ω_N is a primitive N th root of unity, this can only happen when ry divides N . Since $y \in \{0, 1, \dots, N-1\}$, this can only happen if $y = 0$ or r divides N . However, as we saw earlier,

this is the easy case—it is the exact setting of Simon's problem over the integers. Therefore, we can safely assume that $\omega_N^{ry} \neq 1$ and use the formula for a geometric series to get

$$\sum_{i=0}^{\ell-1} \alpha^i = \frac{1 - \alpha^\ell}{1 - \alpha} \implies \sum_{i=0}^{D_x-1} (\omega_N^{ry})^i = \frac{1 - (\omega_N^{ry})^{D_x}}{1 - \omega_N^{ry}}$$

and so we get the probability of measuring a y is given by

$$\frac{1}{ND_x} \left| \frac{1 - \omega_N^{ryD_x}}{1 - \omega_N^{ry}} \right|^2$$

Evidently, we want to show that this probability is high whenever y is close to kN/r . To do this, notice we need to analyze two expressions of the form $|1 - e^{i\theta}|^2$ for an angle θ . In particular, $\theta = 2\pi ryD_x/N$ for the numerator and $\theta = 2\pi ry/N$ in the denominator. First, let's invoke the formula¹

$$|1 - e^{i\theta}|^2 = 4 \sin^2(\theta/2)$$

to get that the probability of measuring a y is given by

$$\frac{1}{ND_x} \left| \frac{1 - \omega_N^{ryD_x}}{1 - \omega_N^{ry}} \right|^2 = \frac{1}{ND_x} \frac{\sin^2(\pi ryD_x/N)}{\sin^2(\pi ry/N)}.$$

Recall that we are trying to show that the y outputs cluster around the values kN/r . For any such $k \in \{1, \dots, r-1\}$, we can write $y = kN/r + \epsilon$ where $\epsilon \in \mathbb{R}$ is used to round kN/r to the nearest integer. Since the nearest integer is at most distance $1/2$ from kN/r , we can assume that $|\epsilon| \leq 1/2$. We want to show that the probability of such a y is high. Plugging this into the above, and noticing that multiples of π do not change the sign of the sin function, we get

$$\frac{\sin^2(\pi r(kN/r + \epsilon)D_x/N)}{\sin^2(\pi r(kN/r + \epsilon)/N)} = \frac{\sin^2(\pi rkD_x + \epsilon\pi rD_x/N)}{\sin^2(\pi rk + \epsilon\pi r/N)} = \left(\frac{\sin(\epsilon\pi rD_x/N)}{\sin(\epsilon\pi r/N)} \right)^2$$

Let's now analyze the numerator and denominator of this function separately. The denominator is easier, so let's start there. Notice that if N is large, then $\epsilon\pi r/N$ is small, so we can safely approximate it using the small-angle formula ($\sin(\theta) \approx \theta$). In fact, since we want an upper bound on the denominator, we just need that $\sin(\theta) \leq \theta$ for all values θ :

$$\sin(\epsilon\pi r/N) \leq \epsilon\pi r/N$$

¹It's not too hard to derive this formula using some basic trig. We start by computing the magnitude-squared:

$$|1 - e^{i\theta}|^2 = (1 - e^{i\theta})(1 - e^{-i\theta}) = 1 - e^{i\theta} - e^{-i\theta} + e^{i\theta}e^{-i\theta} = 2 - (e^{i\theta} + e^{-i\theta})$$

To analyze the second term above, let's invoke Euler's formula— $e^{i\theta} = \cos(\theta) + i\sin(\theta)$:

$$e^{i\theta} + e^{-i\theta} = \cos \theta + i\sin \theta + \cos(-\theta) + i\sin(-\theta) = 2\cos \theta$$

where we've used $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$. We combine the above and use a half-angle formula to arrive at our desired formula:

$$|1 - e^{i\theta}|^2 = 2(1 - \cos \theta) = 2(2\sin^2(\theta/2)) = 4\sin^2(\theta/2).$$

Next, to analyze the numerator, we will use that $\sin(\pi\theta) \geq \theta$ as long as $\theta \leq 7/10$. Inspecting the term in the denominator and recalling that $D_x := \lfloor N/r \rfloor + \delta_x$ for some $\delta_x \in \{0, 1\}$, we get

$$\frac{\epsilon r D_x}{N} = \frac{\epsilon r (\lfloor N/r \rfloor + \delta_x)}{N} \leq \frac{\epsilon r (N/r + 1)}{N} = \epsilon \left(1 + \frac{r}{N}\right) \leq 7/10$$

whenever N is slightly bigger than r . Therefore, we can conclude that the numerator is at least $\epsilon r D_x / N$. Combining all of the above, we get a lower bound on the probability of measuring $y = k(N/r) + \epsilon$:

$$\frac{1}{N D_x} \left(\frac{\epsilon r D_x / N}{\epsilon \pi r / N} \right)^2 = \frac{D_x}{N \pi^2} \geq \frac{N/r - 1}{N \pi^2} \approx \frac{1}{r \pi^2}$$

□

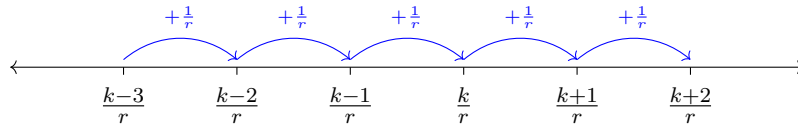
Suppose we manage to measure a string y such that y is close to kN/r . It seems like we might still be stuck because how could we possibly extract r ? The remarkable fact that we are going to use is that rational numbers are less dense than you might think. Specifically, we will show that if y is sufficiently close to kN/r , then in fact we can just determine its exact value. Recall that we showed that with high probability we measure a y such that

$$\left| y - \frac{kN}{r} \right| \leq \frac{1}{2}$$

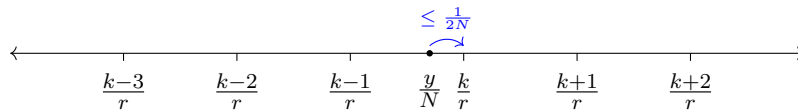
which means that if we divide both sides by N , we arrive at an estimate y/N that is very close to k/r :

$$\left| \frac{y}{N} - \frac{k}{r} \right| \leq \frac{1}{2N}$$

This is where we will finally use that N is very large. In particular, let's set $N = M^2$. The key here is that k/r must be some integer multiple of $1/r$ where $r < M$ (i.e., the order of $a \bmod M$ must be less than M itself). If we were to draw multiples of $1/r$ on a number line it might look something like



On the other hand, our estimate y/N getting very close to k/r —it is at a distance of $1/(2N)$, which is much smaller than the distance between multiples of $1/r$. Therefore, we should be able to round it to some value of k/r :



We can formalize this argument with the following claim:

Claim 4.8. Let $N \geq M^2$. For all $y \in \{0, \dots, N-1\}$, there is at most one rational number a/b such that $b < M$ and

$$\left| \frac{y}{N} - \frac{a}{b} \right| \leq \frac{1}{2N}.$$

Proof. Suppose there are two different fractions a_1/b_1 and a_2/b_2 that simultaneously satisfy the conditions. Since a_1/b_1 and a_2/b_2 are both close to y/N , they must be close to each other:

$$\left| \frac{a_1}{b_1} - \frac{a_2}{b_2} \right| = \left| \frac{a_1}{b_1} - \frac{y}{N} + \frac{y}{N} - \frac{a_2}{b_2} \right| \leq \left| \frac{a_1}{b_1} - \frac{y}{N} \right| + \left| \frac{y}{N} - \frac{a_2}{b_2} \right| = 2 \cdot \frac{1}{2N} = \frac{1}{N}.$$

On the other hand, we have

$$\left| \frac{a_1}{b_1} - \frac{a_2}{b_2} \right| = \left| \frac{a_1 b_2}{b_1 b_2} - \frac{a_2 b_1}{b_1 b_2} \right| \geq \frac{1}{b_1 b_2} > \frac{1}{M^2} \geq \frac{1}{N},$$

where we used that since $b_1, b_2 < M$ followed by $N \geq M^2$. Since the two inequalities directly contradict each other, it must be the case that there is a unique fraction satisfying the conditions. $\square$

Above, we just argued for existence of a unique fraction. In actuality, there is fast classical algorithm using something called continued fractions that will find the k/r in lowest terms. From here, we just invoke the a similar argument as in Simon's algorithm. We run the algorithm multiple times to get values $k_1/r, k_2/r, \dots$. Some of the k_i terms might share factors with r , but with reasonable probability the k_i terms will not share any common factors. Therefore, if we take the least common multiple of all the denominators returned by the continued fraction algorithm, then we arrive at the actual value of r .

4.4.2 Implementing the order finding oracle

One might reasonably suspect that we are now done with the analysis of Shor's algorithm. However, there's one final wrinkle: how do we actually implement the order finding oracle?

In Section 4.1.1, we sketched an argument that if there was an efficient classical algorithm to implement some function $f: \mathbb{Z}_N \rightarrow \mathbb{Z}_M$, then there should be an efficient quantum algorithm that implements the standard oracle B_f . For Shor's algorithm, we require the "powering" function, which takes as input some value $x \in \mathbb{Z}_N$ and computes $a^x \bmod M$.

At first glance, it seemed like there might not be an efficient algorithm for the powering function because x can be as large as $2^n - 1$ (recall that we are representing the N in binary, so x can be exponential in the size of the input). Indeed, if on input x , we ran the brute force algorithm of successively multiplying by a , we would be in trouble:

$$a \pmod{M} \rightarrow a^2 \pmod{M} \rightarrow a^3 \pmod{M} \rightarrow \dots \rightarrow a^{x-1} \pmod{M} \rightarrow a^x \pmod{M}$$

The above sequence is exponentially long for general values of x , so our classical algorithm would take exponential time! The trick is to realize that there is a shorter path through all of the intermediate calculations. We will use an idea called "repeated squaring": if we've already computed $a^i \bmod M$ for some value of i , then we can easily compute $a^{2i} \bmod M$ by taking the square of our computed value for $a^i \bmod M$. This allows us to jump forward in our calculation without doing all of the intermediate computations: $a^{i+1}, a^{i+2}, \dots, a^{2i-1}$.

To formalize this argument, write the input $x \in \mathbb{Z}_N$ in binary $x_{n-1} \cdots x_1 x_0$ so that

$$x = x_0 + x_1 2 + x_2 4 + \cdots x_{n-1} 2^{n-1}.$$

We can break the computation of a^x into powers of 2:

$$a^x = a^{x_0 + x_1 2 + x_2 4 + \cdots x_{n-1} 2^{n-1}} = a^{x_0} a^{x_1 2} a^{x_2 4} \cdots a^{x_{n-1} 2^{n-1}} = (a)^{x_0} (a^2)^{x_1} (a^4)^{x_2} \cdots (a^{2^{n-1}})^{x_{n-1}}$$

We can precompute $a, a^2, a^4, a^8, \dots$ using the repeated squaring trick, and then the bits of x just tell us which of those factors to multiply together.

Bibliography

- [ABB⁺17] Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. *Journal of the ACM (JACM)*, 64(5):1–24, 2017.
- [ABDKT20] Scott Aaronson, Shalev Ben-David, Robin Kothari, and Avishay Tal. Quantum implications of Huang’s sensitivity theorem. *arXiv:2004.13231*, 2020.
- [BBBV97] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.
- [BGT21] Adam Bouland and Tudor Giurgica-Tiron. Efficient universal quantum compilation: An inverse-free Solovay-Kitaev algorithm. *arXiv preprint arXiv:2112.02040*, 2021.
- [BHT97] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum algorithm for the collision problem. *arXiv quant-ph/9705002*, 1997.
- [CQ18] Guangya Cai and Daowen Qiu. Optimal separation in exact query complexities for simon’s problem. *Journal of computer and system sciences*, 97:83–93, 2018.
- [DN06] Christopher M Dawson and Michael A Nielsen. The Solovay-Kitaev algorithm. *Quantum Information & Computation*, 6(1):81–95, 2006.
- [EHK04] Mark Ettinger, Peter Høyer, and Emanuel Knill. The quantum query complexity of the hidden subgroup problem is polynomial. *Information Processing Letters*, 91(1):43–48, 2004.
- [Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.
- [Kit95] A Yu Kitaev. Quantum measurements and the abelian stabilizer problem. *arXiv preprint quant-ph/9511026*, 1995.
- [Kit97] A. Y. Kitaev. Quantum computations: algorithms and error correction. *Russ. Math. Surv.*, 52(6):1191–1249, 1997.
- [Kup23] Greg Kuperberg. Breaking the cubic barrier in the Solovay-Kitaev algorithm. *arXiv preprint arXiv:2306.13158*, 2023.

Appendix A

Complex numbers

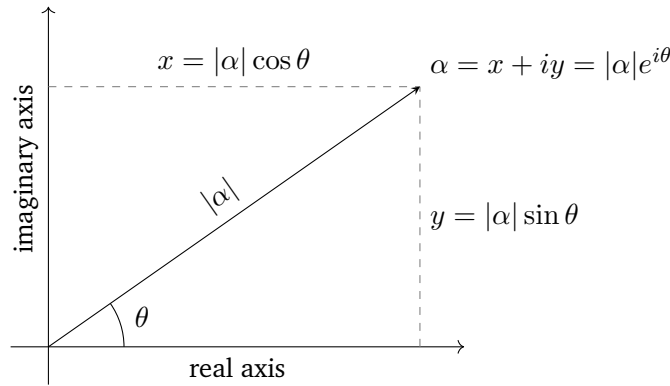
Quantum mechanical systems are described by vectors of complex numbers. Therefore, it is important to review some important properties of complex numbers. We will use the symbol $\mathbb{R}$ to denote the set of all real numbers and the symbol $\mathbb{C}$ to denote the set of all complex numbers. That is,

$$\mathbb{C} = \{x + iy \mid x, y \in \mathbb{R} \text{ and } i^2 = -1\}$$

For complex number $\alpha = x + iy$, we say that x is the *real part* of α and y is the *imaginary part* of α .

A.1 Complex numbers as 2d vectors

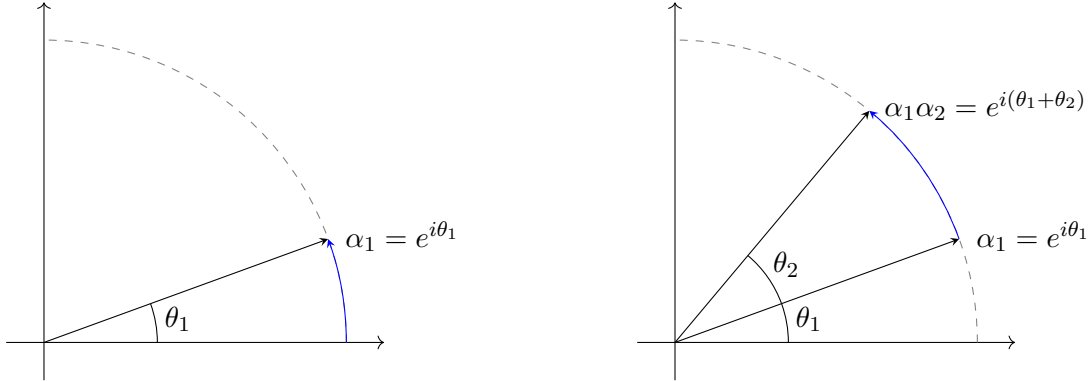
It turns out to be particularly use to visualize a complex number $\alpha = x + iy$ as a vector in 2-dimensional space, where the real and imaginary parts are the two components. The *absolute value* or *magnitude* of α , which we denote by $|\alpha|$, is just the length of this vector. That is, $|\alpha| = \sqrt{x^2 + y^2}$.



From this picture, we can see that we could alternatively capture all the information about our complex number α by specifying the angle θ and the magnitude $|\alpha|$. Somewhat surprisingly, this information is exactly captured using single complex number $|\alpha|e^{i\theta}$, which follows from the following remarkable fact:

Theorem A.1 (Euler's formula). $e^{ix} = \cos x + i \sin x$ for all $x \in \mathbb{R}$.

In summary, every complex number $\alpha \in \mathbb{C}$ can be written in *polar coordinates* as $re^{i\theta}$ for some radius $r \in \mathbb{R}$ and angle $\theta \in \mathbb{R}$. This representation gives us a particularly nice interpretation of the multiplication of two complex numbers: if $\alpha_1 = r_1e^{i\theta_1}$ and $\alpha_2 = r_2e^{i\theta_2}$, then $\alpha_1\alpha_2 = (r_1r_2)e^{i(\theta_1+\theta_2)}$. Notice that the angles add, so we can interpret multiplication of complex numbers as a rotation in complex space. This is especially nice when the radii r_1 and r_2 are both 1. We have the following visualization, starting with α_1 and multiplying by α_2 :



A.2 Complex conjugate

For every complex number $\alpha = x + iy$, we define its *complex conjugate*¹ as $\alpha^* := x - iy$. The complex conjugate helps us define the overlap of vectors in complex space. For example, using this definition, one can check that $|\alpha| = \sqrt{\alpha\alpha^*}$.

Fact A.2. *The complex conjugate is distributive. That is, for all $\alpha, \beta \in \mathbb{C}$, we have*

- *Distributive over multiplication:* $(\alpha\beta)^* = \alpha^*\beta^*$
- *Distributive over addition:* $(\alpha + \beta)^* = \alpha^* + \beta^*$

Another way to conceptualize the complex conjugate is to interpret it as a reverse rotation in complex space—that is, the complex conjugate flips the sign of the angle. To see this, we can take the conjugate of an arbitrary complex number $\alpha = re^{i\theta}$ as follows:

$$\alpha^* = (re^{i\theta})^* = (r(\cos \theta + i \sin \theta))^* = r(\cos \theta - i \sin \theta) = r(\cos(-\theta) + i \sin(-\theta)) = re^{-i\theta}$$

where we've used Euler's formula to expand $e^{i\theta}$, distributivity of the complex conjugate, and two standard trigonometric identities: $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$.

A.3 Roots of unity

An n th root of unity is a complex number z such that $z^n = 1$. A root of unity can be thought of a cycle around the unit sphere in complex space. As an example, let's consider a 4th root

¹As a note of warning, we will mostly write the complex conjugate as α^* , but it is also common to see it written as $\bar{\alpha}$.

of unity z . If were to start with z and repeatedly multiply by z , we would get the following sequence:

$$z \rightarrow z^2 \rightarrow z^3 \rightarrow z^4 = 1 \rightarrow z \rightarrow z^2 \rightarrow z^3 \rightarrow z^4 = z \rightarrow \dots$$

The sequence repeats every 4 iterations, using the fact that $z^4 = 1$. In this example, the smallest period of this sequence might actually be 2: if $z^2 = 1$, then $z^4 = 1$. To rule out such possibilities, we define a *primitive* n th root of unity to a complex number z such that $z^n = 1$ and there is no positive integer $k < n$ such that $z^k = 1$.

As it turns out, there is a nice explicit form for an n th primitive root of unity:

$$\omega_n := e^{\frac{2i\pi}{n}} = \cos\left(\frac{2\pi}{n}\right) + i \sin\left(\frac{2\pi}{n}\right)$$

where the second equality follows from Euler's formula.