# Quantum Complexity Theory

*Lecture Notes*

Daniel Grier

# Contents

## Preamble

These lecture notes derive from a sequence of scribe notes taken from the Fall 2022 iteration of CSE 291 / Math 277A (Quantum Complexity Theory). This document represents a continual and iterative process to bring those notes into a more cohesive whole. Any mistakes can be assumed to have been introduced by me. Please feel free to email me (`dgrier@ucsd.edu`) if you notice any.

These course notes are written for graduate students with a strong mathematical background, but not necessarily any previous experience with quantum computing. Some previous exposure to complexity theory will be extremely useful. I recommend the excellent Arora-Barak textbook for those looking to brush up on that background.

## Overview

The goal of these notes is to rigorously compare, using the tools of complexity theory, the power of quantum and classical computers. We will see some settings in which quantum computers outperform their classical counterparts and some settings in which quantum computers are no better than brute force classical approaches. Given the recent explosion of progress in actually building a quantum computer, it is becoming more important than ever that we understand the difference in the quantum and classical worlds and what they allow us to compute. The exploration in these notes will take us to the forefront of quantum computing research, where we'll look at the complexity-theoretic foundations of these recent experiments.

# 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 pure foundations: What is a quantum state, and what kinds of operations can you perform on that state?

From there, we will define a computational model (analogous to the classical Turing machine) that captures the essence of a quantum computer.

## 1.1 The basics of quantum computation

What is the state of a quantum system? Let's start by analogy to one of the simplest classical objects—a biased coin. Since it will be convenient later, let's suppose the coin has two sides, corresponding to a $0$-outcome and a $1$-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:

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

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:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \leftarrow \begin{matrix} 0\text{-outcome} \\ \text{with certainty} \end{matrix} \qquad \text{or} \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \leftarrow \begin{matrix} 1\text{-outcome} \\ \text{with certainty} \end{matrix}$$

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 a biased die); but for simplicity, let's assume there are only finitely many. 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 are 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 $\ell_1$-norm of the vector, where the $\ell_1$-norm of vector $v = (v_1, v_2, \ldots, v_d) \in \mathbb{C}^d$ is defined as

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

## 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,

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

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 the squared magnitude of the 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 outcome with probability $|\alpha|^2$ 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 $O$ 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 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 $\ell_2$-norm of the vector is $1$. The $\ell_2$-norm of any $v = (v_1, v_2, \ldots, v_d) \in \mathbb{C}^d$ 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 $\ell_1$ to the $\ell_2$ norm.

## Unitary matrices

Because the set of valid quantum states must have unit $\ell_2$-norm, the set of viable quantum operations must preserve the $\ell_2$-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 $\ell_2$-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 systems consisting of multiple qubits, so that we can understand what it means to locally apply some quantum operation.

### 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

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

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:

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

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

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

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

| | |
|---|---|
| Scalar multiplication: | $(\alpha v) \otimes (\beta w) = (\alpha\beta)(v \otimes w)$ |
| Associativity: | $(v \otimes w) \otimes z = v \otimes (w \otimes z)$ |
| 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 exponentially 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 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 $00$ outcomes 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 outcome of the qubit at the other end.[1]

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:

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

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

### 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 $U \in \mathbb{C}^a$ and $V \in \mathbb{C}^b$, 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 $U \in \mathbb{C}^a$ and $V \in \mathbb{C}^b$ 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] A significant amount of ink has been spilled on exactly what is happening at a physical layer when a measurement like this is made. Look up the "quantum measurement problem". Thankfully for one of the most cherished pysical laws, this entanglement phenomenon does *not* allow for faster than light communication.

### 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 is 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:

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

Suppose we look at the second coin, but not the first. The probability the see 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 outcome is just $.3/.6 = .5$. In practice, its 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 $\ell_2$-norm rather than the $\ell_1$-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 second qubit is $|\sqrt{.3}|^2 + |\sqrt{.3}|^2 = .6$, and the distribution on the first qubit 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 $\ell_2$-norm). Importantly, we reserve certain vectors special 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:

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

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 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

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

### 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"-$\psi$) to denote the conjugate transpose of the state $|\psi\rangle$. Therefore, the inner product between two state $|\psi\rangle$ 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.

### 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 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 $U^\dagger U = U^\dagger U = I$<br>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 distrubtion of 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 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

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

and

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

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

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

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.

### 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 $\rho$ and apply a unitary $U$, that 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 $\rho$ 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 stated 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: $\rho$ is a valid density matrix if and only if $\rho$ is a trace-1 positive semidefinite matrix. Trace-1 implies that $\mathrm{Tr}(\rho) = 1$. 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 $\rho$, which is valid since we have assumed that $\rho$ is positive semidefinite. The eigenvectors of this decomposition will be the pure states in the decomposition, and the eigenvalues will be the associated probabilities.

### 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)*.

### 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 $\mathrm{Tr}(\rho) = 1$ and $\rho$ is positive semidefinite |
| *Operations*: | Completely positive trace-preserving maps $\Phi$<br>If $\Phi$ 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 $\lvert x\rangle\langle x\rvert$ with probability $\langle x\rvert \rho \lvert x\rangle$ |

## 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, let's imagine we start with a composite system $\mathcal{H}_A \otimes \mathcal{H}_B$. For simplicitly, 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 $\mathrm{Tr}_B$ is the unique linear map satisfying

$$\mathrm{Tr}_B(|a_i\rangle\langle a_j| \otimes |b_i\rangle\langle b_j|) = |a_i\rangle\langle a_j| \mathrm{Tr}(|b_i\rangle\langle b_j|),$$

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

So, if we have some state $\rho_{AB}$ 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

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

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

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

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)

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

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.

### 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 $\rho$, there is a $(2n)$-qubit pure state such that tracing out the last $n$ qubits of $|\psi\rangle$ leaves the state $\rho$. This process is called *purification*.

We will give an explicit purification procedure. First, let $\rho$ 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 $\rho$:

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

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

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

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

Are purifications unique? Unfortunately, not. To see this, notice that we can generalize our purification procedure above my multiplying the second register by any $n$-qubit unitary $U$:

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

Intuitively, it makes sense that changing the basis of the second register shouldn't affect 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:

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

where the first equality uses the cyclic property of the trace and the second using 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** (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 pure states $|\psi\rangle$.*

*Proof.* We argue by constradiction. Suppose such at $U$ 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 $U(|\psi_2\rangle \otimes |\varphi\rangle)$

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

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! □

# Chapter 2

# Computation with Quantum Circuits

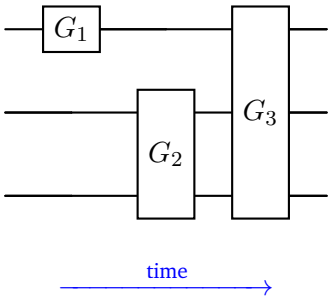
## 2.1 The quantum circuit model

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*.

### Introduction to quantum circuits

A $n$-qubit quantum circuit is a collection of unitary operations $G_1, \ldots, G_m$, 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:



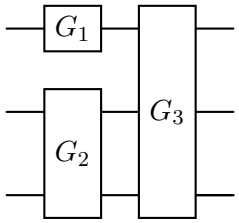

The above diagram is a circut on 3 qubits with 3 gates: the single-qubit gate $G_1$ is applied first; the 2-qubit gate $G_2$ is applied next; and finally $G_3$ is applied as a 3-qubit gate. The unitary matrix representing this circuit is

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

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

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

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 $2$.

## Examples with common gates

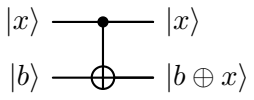
Let's take a look at some of the most common gates used in quantum circuits and the special notation that we use to denote them.

### 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:

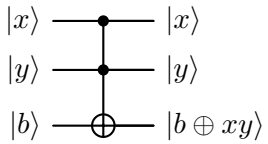
$$|00\rangle \mapsto |00\rangle\,, \quad |01\rangle \mapsto |01\rangle\,, \quad |10\rangle \mapsto |11\rangle\,, \quad |11\rangle \mapsto |10\rangle\,.$$

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



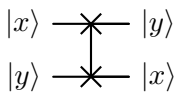

where $x, y, b \in \{0, 1\}$ are arbitrary bits ($xy$ is the product of $x$ and $y$). Notice that the third bit is flipped exactly when both controls are 1.

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.

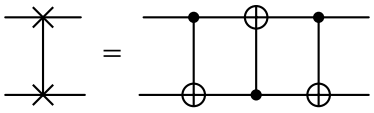
Finally, let's discuss the SWAP gate, another important "classical reversible" gate on 2 qubits. Aplty named, the SWAP gate swaps qubits, i.e., for all $x, y \in \{0, 1\}$ it maps:

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

In a circuit diagram, it is depicted as




One can check the following nice identity:



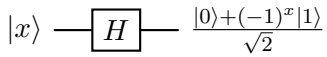

In other words, we can replace every SWAP gate in a circuit with 3 CNOT gates. This is a common theme we will continue to see—we can take some gates as the fundamental ones that will generate the rest.

**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

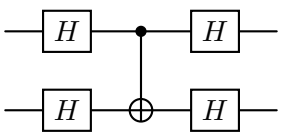
$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} := |+\rangle \qquad\qquad H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} := |-\rangle$$

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




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 $(H \otimes H)\mathrm{CNOT}(H \otimes H)$, 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:

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

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

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

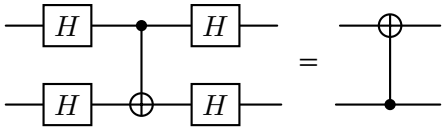
Therefore, the final layer of Hadamard gates returns the state to $|00\rangle$. That is, after all that computation, we see that the circuit acts as the identity on the $|00\rangle$. For completeness, let's see one more case (the input $|01\rangle$) in its entirety:

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

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 $i$:

$$S\,|0\rangle = |0\rangle \qquad\qquad S\,|1\rangle = i\,|1\rangle$$

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

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

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.

### 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 $\epsilon$-accuracy with respect to the operator norm [DN06]. That is, a unitary $U$ is $\epsilon$-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.

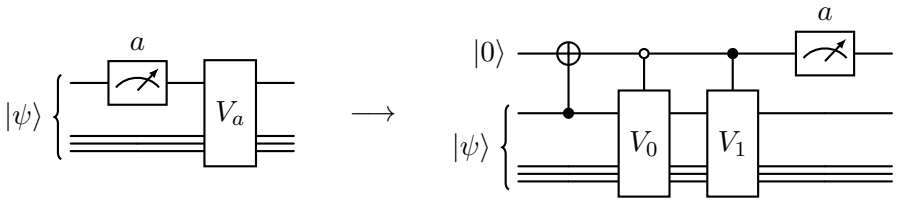
**Theorem 2** (Solovay-Kitaev [Kit97])**.** *Given an approximately universal gate set, there is a circuit to appoximate any unitary to $\epsilon$-accuracy with $\mathcal{O}(4^n \text{polylog}(\frac{1}{\epsilon}))$ gates.*

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 $O(4^n \log^{1.441}(1/\epsilon))$ gates [Kup23].

### 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 probably distribution over all measurments 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.

## 2.2 The complexity class BQP

This section relies on a background in classical complexity theory. For a short review of some of the fundamental classical complexity classes, see Appendix A.

To properly define the quantum complexity class BQP, we need to first discuss how a quantum circuit is encoded. Let us suppose that the circuit is constructed from some reasonable universal gate set (i.e., all the amplitudes used in the gates are efficiently computable). We will use the notation $\langle Q \rangle$ to denote the encoding of a circuit $Q$ as a bit string. We now discuss the requirement for a proper encoding:

1. The encoding is unique: mapping from a circuit to its encoding must be *injective*.

2. The encoding is not too big: if $Q$ has $m$ gates, then $\langle Q \rangle$ has at most $\mathsf{poly}(m)$ bits.

3. The encoding is not too small: if $Q$ has $m$ gates, then $\langle Q \rangle$ has at least $m$ bits.

Equipped with an encoding, we can now talk about Turing machines whose output is (an encoding of) a quantum circuit. A circuit family $\{Q_n\}_{n=1}^{\infty}$ is *poly-time uniform* if there exists a poly-time Turing machine such that on input $1^n$ outputs $\langle Q_n \rangle$. We are now ready to define the complexity class capturing efficient quantum computation:

**Bounded-error Quantum Polynomial Time (BQP):**
Languages $L$ such that there exists poly-time uniform class of quantum circuits $\{Q_n\}_{n=1}^{\infty}$ and a polynomial $q$ such that for all $x \in \{0,1\}^n$:

- If $x \in L$, probability of measuring $|1\rangle$ on the first qubit of $Q_n |x\rangle \otimes \left|0^{q(n)}\right\rangle$ is at least $\frac{2}{3}$.
- If $x \notin L$, probability of measuring $|1\rangle$ on the first qubit of $Q_n |x\rangle \otimes \left|0^{q(n)}\right\rangle$ is at most $\frac{1}{3}$.

## 2.3 How does BQP compare to its classical complexity friends?

In this section, we will see how BQP fits into the zoo of classical complexity classes. To start, let's start with an "obvious" claim—namely, efficient quantum computation is at least as powerful as efficient classical computation. That is, $\mathsf{BPP} \subseteq \mathsf{BQP}$.

To do this, it will be useful to also define BPP in terms of circuits:

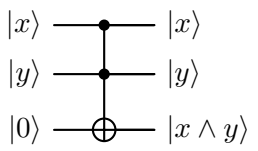
**Bounder-error probibalistic polynomial time (BPP)**
The class of languages $L$ for which there is a poly-time uniform family of classical circuits $\{C_n\}_{n=1}^{\infty}$ and a polynomial $q$ such that for all $x \in \{0,1\}^n$:

- If $x \in L$, $C(x,r) = 1$ for at least 2/3 of strings $r \in \{0,1\}^{q(n)}$.
- If $x \notin L$, $C(x,r) = 1$ for at most 1/3 of strings $r \in \{0,1\}^{q(n)}$.

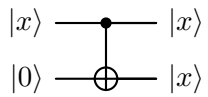
Once again, we assume that the classical circuits are compiled from a reasonable gates like AND and NOT gates.

**Theorem 3.** $\mathsf{BPP} \subseteq \mathsf{BQP}$.

*Proof.* We will show that quantum circuits can directly simulate AND and NOT gates, which are universal for classical computation. NOT is already unitary operator, so there is nothing to do. To simulate AND, we simply observe that the Toffoli computes AND on the target. That is,




for all $x, y \in \{0,1\}$. Therefore, every gate in a classical circuit can be replaced by a quantum one. Technically, classical circuits can "fan out" the output of a gate to serve as input for several other gates. We can simulate this behavior in a quantum circuit through a simple CNOT gate that acts as a copy gate:




We also need to deal with the random bits in the definition of BPP that are not present in the definition of BQP. This is also straightforward, if we want to simulate a random bit, we simply start the corresponding qubit of the quantum circuit in the $|+\rangle$ state.

Finally, we note that the reduction sketched above can be done efficiently. That is, a poly-time uniform family of classical circuits for a language in BPP implies there is also a poly-time uniform family of quantum circuits for the same language. Therefore, BPP $\subseteq$ BQP. □

What about classical upper bounds on the power of poly-time quantum computation? Let's start with the most basic bound, exponential time:

**Theorem 4.** BQP $\subseteq$ EXP.

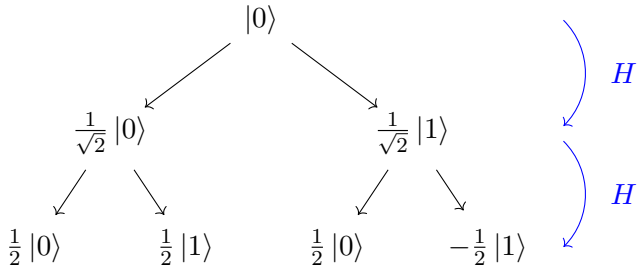
*Proof.* Any $n$-qubit state can be represented as a length-$2^n$ complex vector. Every quantum gate on an $n$-qubit state is unitary matrix of size $2^n \times 2^n$. The state after application of the gate is obtained by simply multiplying the vector by the matrix. Since such matrix-vector products can be computed in $O(4^n)$ time, and simulating the circuit requires at most polynomially many matrix-vector product operations, we then have BQP $\subseteq$ EXP. □

In some sense, the above containment above showcases what many think to be true about quantum computation—to classical simulate a general quantum computer, you fundamentally need some kind of brute-force exponential time computation. However, notice that the exponential-time algorithm is both time inefficient *and* space inefficient. It turns out that a classical simulation is possible with polynomial space, or in other words, BQP $\subseteq$ PSPACE.

The proof of this fact will be to consider a quantum computation as a sum over paths in a tree. Let's explore this idea with an exmaple. Consider the (very simple) circuit $H^2$. Let's look how the state $|0\rangle$ evolves gate by gate:

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

We can arrange these same steps in a tree:




Of course, for this simple example, we know the result is simply $|0\rangle$ after simplification. but we don't have to simplify the expression. Instead, we can keep it as a linear combination of $|0\rangle, |1\rangle, |0\rangle, |1\rangle$ with respective coefficients $\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, -\frac{1}{2}$. Why might we want to do this? The

key is that in the tree picture, the final states in the tree are just computational basis states rather than superpositions of computational basis states (as would be the case for a general quantum state). We can keep track of the amplitude and the classical basis state in polynomial space.

To do a generic quantum computation, we will traverse the entire computation tree, summing up the amplitudes when we get to leaves. Let's see how these ideas fit together.

**Theorem 5.** BQP $\subseteq$ PSPACE.

*Proof.* Let's start by making the simplifying assumption that the quantum circuit is constructed from Toffoli and Hadmard gates since those gates are computationally universal. This means that we can specify any leaf in the tree by a bit string indicating the path taken on all Hadamard gates from the root to the leaf (e.g., a 0 says to take the left branch, and a 1 says taking the right branch). Since the quantum circuit is of polynomial size, these leaf pointers are also of polynomial size.

Specifically, if the quantum circuit has $h$ Hadamard gates, then we can designate a path as a length-$h$ bit string. If we iterate through all of these leaf pointers, we can traverse the entire computation tree. Since the leaf pointers are of polynomial size and the intermediate states of the computation tree consist of a computational basis state and a single amlitude, we can traverse the entire tree using only polynomial space. To finish, the classical simulation algorithm, we need to specify what information we collect at the leaves of the tree once the traversal processes them.

We will devise a classical algorithm that computes the acceptance probability of the quantum circuit. If the final state of the quantum circuit is

$$|\psi\rangle = \sum_{y\in\{0,1\}^{n-1}} \alpha_{0y}\,|0y\rangle + \alpha_{1y}\,|1y\rangle\,,$$

then our goal is compute the probability of measuring $|1\rangle$ on the first qubit:

$$p_{\text{acc}} = \sum_{y\in\{0,1\}^{n-1}} |\alpha_{1y}|^2.$$

Our plan will now simple—for each $y \in \{0,1\}^{n-1}$, we traverse the entire tree looking for computation paths that end in a $|1y\rangle$ state, adding up the amplitudes as we go. The full pseudocode for the algorithm is below:

$p_{\text{acc}} \leftarrow 0$
**for** $y \in \{0,1\}^{n-1}$ **do**
  $\alpha_{1y} \leftarrow 0$
  **for** path $p \in \{0,1\}^h$ in computation tree **do**
    **if** path $p$ ends in leaf state $|1y\rangle$ with amplitude $\beta$ **then**
      $\alpha_{1y} \leftarrow \alpha_{1y} + \beta$
  $p_{\text{acc}} \leftarrow p_{\text{acc}} + |\alpha_{1y}|^2$
**return** $p_{\text{acc}}$

If the calculated the probability of acceptance is greater than 2/3, the classical algorithm accepts; otherwise, it rejects. □

Can this be improved? Somewhat surprisingly, the answer is "yes". Poly-size quantum circuits can be simulated in PP, which captures decision problems solvable by counting the number of satisfying instances of an NP problem. Formally, recall the definition of PP: the class of languages $L$ such that there exists a deterministic poly-time Turing machine $M$ and polynomial $q$ such that for all $x \in \{0,1\}^n$

- If $x \in L$, then $M(x,y)$ accepts for more than $1/2$ of strings $y \in \{0,1\}^{q(n)}$.
- If $x \notin L$, then $M(x,y)$ accepts at most $1/2$ of strings $y \in \{0,1\}^{q(n)}$.

**Theorem 6.** BQP $\subseteq$ PP.

*Proof.* Once again, let's look at the computation tree of a poly-size quantum circuit built from Toffoli and Hadmard gates. Once again, the number of leaves in the tree is given by $2^h$ where $h$ is the number of Hadamard gates. The key idea is it that because the magnitude of the amplitude at every leaf is exactly $1/\sqrt{2^h}$, the final amplitude on any given basis state is proportional to the number of leaves that have a positive sign minus the number of leaves that have a negative sign. Let the final state before measurement of the quantum algorithm be

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

Let $a_x$ be the fraction of all leaf nodes that are in state $|x\rangle$ and have positive weight, and let $b_x$ be the fraction of all leaf nodes that are in state $|x\rangle$ and have positive weight. We can express the amplitude of the state as

$$\alpha_x = \sqrt{2^h}(a_x - b_x)$$

for real numbers $a_x, b_x \geq 0$.

Now consider the following classical algorithm. Randomly and uniformly follow one branch of the tree and record the state $|x\rangle$ at the leaf, then return to the beginning of the tree and randomly follow another branch of the tree and record the state $|y\rangle$ at the leaf. If $x \neq y$, then flip a coin to determine if we accept or reject the input. However, if $x = y$, then we want to determine if the signs of these states are likely to add constructively or destructively. Explicitly, if the first qubit of $x = y$ is a 1, then accept if the coefficients on $x$ and $y$ have the same sign and reject if they have opposite signs, and if the first qubit of $x = y$ is a 0, then accept if the coefficients have opposite signs and reject if they have the same sign.

Let's now see why this combination of accept/reject rules would give you an algorithm which accepts more than $50\%$ of the time only when the quantum algorithm accepts. Notice that when $x \neq y$, our decision rule is simply a coin flip, which succeeds with probability $50\%$ regardless of the truth. It suffices to show our decision rule succeeds with probability strictly more than $1/2$ when $x = y$. Notice that the probability of seeing the state $|x\rangle$ twice with constructive weights is given by

$$\Pr[\text{seeing } |x\rangle \text{ twice with constructive weights}] = a_x^2 + b_x^2.$$

The probability of seeing the state $|x\rangle$ twice with destructive weights is given by

$$\Pr[\text{seeing } |x\rangle \text{ twice with destructive weights}] = 2a_x b_x$$

Suppose the quantum algorithm accepts (i.e., measures "1" on the first qubit) with probability $p_{\text{acc}}$. We get

$$p_{\text{acc}} = \sum_{y \in \{0,1\}^{n-1}} \alpha_{1y}^2 = 2^h \sum_{y \in \{0,1\}^{n-1}} (a_{1y} - b_{1y})^2 . \tag{2.1}$$

On the other hand, this implies that the algorithm rejects with probability

$$1 - p_{\text{acc}} = \sum_{y \in \{0,1\}^{n-1}} \alpha_{0y}^2 = 2^h \sum_{y \in \{0,1\}^{n-1}} (a_{0y} - b_{0y})^2 \tag{2.2}$$

Dividing by $2^h$ and subtracting Equation (2.2) from (2.1), we get

$$\begin{aligned} \frac{2p_{\text{acc}} - 1}{2^h} &= \sum_y (a_{1y} - b_{1y})^2 - \sum_y (a_{0y} - b_{0y})^2 \\ &= \sum_y (a_{1y}^2 + b_{1y}^2 + 2a_{0y}b_{0y}) - \sum_y (a_{0y}^2 + b_{0y}^2 + 2a_{1y}b_{1y}) \end{aligned}$$

Notice that the first sum on the right hand side corresponds exactly to the probability of seeing either (i) two states starting with $1$ and with constructive weights, or (ii) two states starting with $0$ and with destructive weights. Ignoring the 50-50 coin tosses from the classical simulation procedure that don't end in the same state, this is exactly the probability that the procedure outputs YES. Similarly, the second sum is the probability that our classic simulation procedure outputs NO. This then gives

$$\frac{2p_{\text{acc}} - 1}{2^h} = \Pr[\text{simulation outputs YES}] - \Pr[\text{simulation outputs NO}],$$

Hence, when $p_{\text{acc}} \geq 2/3$ left hand side is positive. Since the classic simulation is more likely to output YES, and so the PP machine accepts. When $p_{\text{acc}} \leq 1/3$, the left hand side is negative, so the classical simulation is more likely to output NO, and therefore the PP machine rejects. This concludes the proof. □

# 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., in BQP), and therefore, form the basis for some of the most promising quantum algorithms.

For some intuition for the power of this setting, imagine a kind of satisfiability problem captured by a function $f\colon \{0,1\}^n \to \{0,1\}$. That is, we'd like to know if there is some input $x \in \{0,1\}$ 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 $f\colon \{0,1\}^n \to \{0,1\}^m$ is pretty straightforward in the classical circuit setting—the queries consist of $n$-input $m$-output gates, and the query complexity is the number 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 a output register:

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

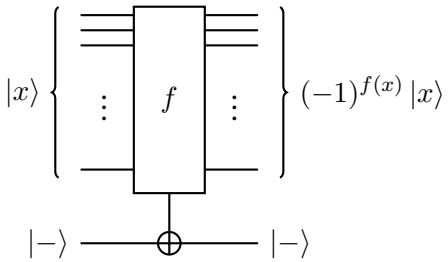
for all $x \in \{0,1\}^n$ and $b \in \{0,1\}^m$. That is, the standard oracle just computes the value of the function on the input and dumps it (reversibly) into the output register.

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 it 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 circle can implement the oracle $B_f$ by straightforwardly implenting 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 $f\colon \{0,1\}^n \to \{0,1\}$, 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:




The circuit diagram shows the $B_f$ oracle where the function $f$ is computed on the $|x\rangle$ register and XOR'd into the $|-\rangle$ register. We have

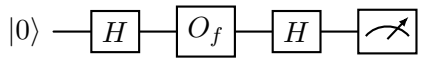
$$|x\rangle |-\rangle \xrightarrow{B_f} |x\rangle \left( \frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} \right) = (-1)^{f(x)} |x\rangle |-\rangle = (O_f |x\rangle) |-\rangle .$$

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 $f\colon \{0,1\} \to \{0,1\}$ 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

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

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.

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

**Deutsch-Jozsa problem**

*Function:* $f\colon \{0,1\}^n \to \{0,1\}$

*Promise:* $f$ is either

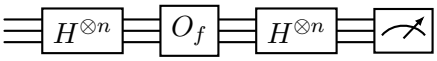
Constant: All outputs of $f$ are equal. $\forall x, y \in \{0,1\}^n$, $f(x) = f(y)$; or

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

*Goal:* Determine if $f$ is 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

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

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 Chernov bound). That is, quantum algorithms are at best only marginally better than classical randomized algorithms for the Deutsch-Jozsa problem.

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

**Berstein-Vazirani problem**

| | |
|---|---|
| *Function:* | $f\colon \{0,1\}^n \to \{0,1\}$ |
| *Promise:* | $f$ is linear. For all $x \in \{0,1\}^n$, $f(x) = x \cdot s$ for some secret string $s \in \{0,1\}^n$ |
| *Goal:* | Find $s$. |

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 $e_1 := 10\cdots0$, $e_2 := 010\cdots0$, and so on up to $e_n := 0\cdots01$. 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-Josza problem—a layer of Hadamards, followed by the phase oracle, followed by another layer of Hadamards. 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-Josza 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 amplidude 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 Berstein-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.

As a final remark, we note that both the Deutsch-Josza and Berstein-Vazirani algorithms are instances of *Fourier sampling*. To see this, let's quickly introduce the Fourier basis. First, for all $y \in \{0,1\}^n$, we define the basis functions

$$\chi_y(s) := y \cdot s \pmod 2$$

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

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

To see orthonormality of these basis vectors, we compute

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

Since we've defined a basis of $2^n$ independent functions, every Boolean function $f \colon \{0,1\}^n \to \{0,1\}$ 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

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

Let's now consider what the Deutsch-Josza/Berstein-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

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

In other words, what our quantum algorithm is actually doing for the Deutsch-Josza and Berstein-Vazirani problems is sampling a $y \in \{0,1\}^n$ with probability equal to $|\hat{f}(y)|^2$. Therefore, any 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**

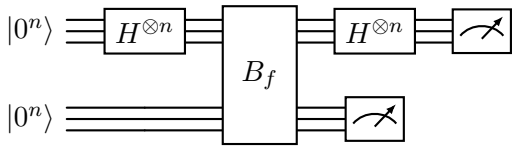
| | |
|---|---|
| *Function:* | $f \colon \{0,1\}^n \to \{0,1\}^n$ |
| *Promise:* | Outputs of $f$ are paired by secret $s \in \{0,1\}^n$. That is, $f(x) = f(y)$ iff $x = y \oplus s$. |
| *Goal:* | Find $s$. |

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:

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

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 determistic algorithm [CQ18]. Intuitively, after $k$ queries, we've looked at $\binom{k}{2} \approx k^2$ 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:

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

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

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

   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)**

| | |
|---|---|
| *Function:* | $f : G \to \{0,1\}^*$ where $G$ is a group |
| *Promise:* | $f$ is constant on a hidden subgroup $H \leq G$. That is, $f(x) = f(y)$ iff $xH = yH$. |
| *Goal:* | Find $H$. |

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 7** (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 8** (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 wasn't previously defined. 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. If fact, such a strategy provably cannot work:

**Theorem 9** (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 determinstic 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**
*Function:* $f\colon \{0,1\}^n \to \{0,1\}$
*Goal:* 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 NP $\subseteq$ BQP. Of course, by Theorem 9, we already know such a simple strategy for solving NP problems won't work. That is, since classical computers require expoentially 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.

## BBBV lower bound for search

While Theorem 9 leaves open the possibility that unstructured search can be solved with $O(2^{n/4})$ queries, this is unfortunately still too optimistic.

**Theorem 10** (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

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

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 $U_0, U_1, \ldots, U_t$ 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 $\ell_2$ 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 $p, q$:

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

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 $1/2 + \mathrm{TV}(p, q)/2$.

**Lemma 11.** *If $\||\phi\rangle - |\psi\rangle\|_2 < \epsilon$, then the total variation distance from measuring $|\phi\rangle$ and $|\psi\rangle$ is at most $2\epsilon$.*

*Proof.* Suppose $|\phi\rangle = \sum \alpha_x |x\rangle$, $|\psi\rangle = \sum \beta_x |x\rangle$. For ease of notation, assume $\alpha_x, \beta_x$ 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 $4\epsilon$ 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 $2\epsilon$. □

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 $T = \Omega(2^{n/2})$.

### Grover's algorithm

While a $\Omega(\sqrt{2^n})$ query lower bound for search is an unpleasant reality, notice that the situation is not as bad as it could be—after all, the classical algorithm requires $\Omega(2^n)$ queries. Can we devise a devise a quantum algorithm that gets this quadratic improvement over the classical algorithm? We can!

**Theorem 12** (Grover's algorithm)**.** *There is a $O(\sqrt{2^n})$ time quantum algorithm for unstructured search.*

It will turn out that 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.

The entirety of Grover's algorithm is simply alternating between the phase oracle (i.e., $O_f$) and the "Grover diffusion operator" defined as

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

where $\left|u\right\rangle := H^{\otimes n}\left|0^n\right\rangle$ is the uniform superposition.

**Claim 13.** *The diffusion operator $D := 2\left|u\right\rangle\!\left\langle u\right| - I$ is a unitary operation that reflects[1] about $\left|u\right\rangle$. Furthermore, $D$ can be constructed with linearly-many gates in log depth.*

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

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

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

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

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

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

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: $D_0 = 2\left|0^n\right\rangle\!\left\langle 0^n\right| - I$. Therefore, we just need a circuit for $D_0$. On the computational basis states, we have $D_0\left|x\right\rangle = (-1)^{x_1\vee\cdots\vee x_n}\left|x\right\rangle$ 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. □

[1] By "reflect" about $\left|u\right\rangle$, we mean that $D$ flips the sign of every vector in the subspace orthogonal to $\left|u\right\rangle$.

**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** $i \in \{1, \ldots, T\}$ **do**
3: $\quad |\psi_i\rangle \leftarrow DO_f |\psi\rangle_{i-1}$
4: $s^* \leftarrow$ measurement of $|\psi_T\rangle$
5: **return** $s^*$ if $f(s^*) = 1$; otherwise, `null`

Examining Grover's algorithm, we see that the final state before we measure is given by

$$DO_f \cdots DO_f DO_f |u\rangle$$

To understand why this algorithm works, it will be extremely useful to take a 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:

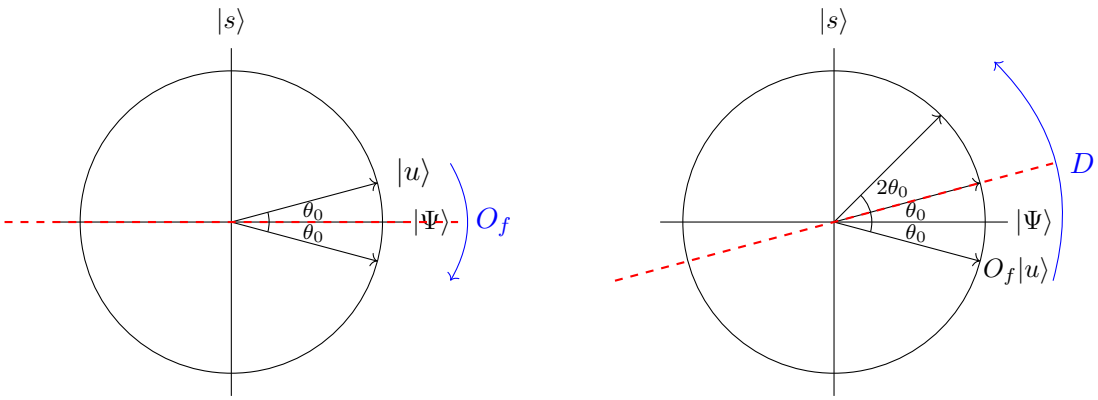
**Observation 14.** *Each Grover iteration keeps the state in the span of $|s\rangle$ and $|\Psi\rangle$.*

*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

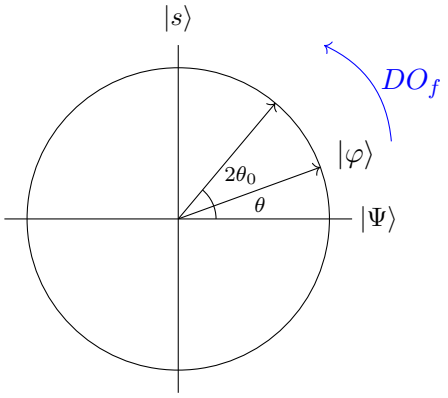
$$\alpha |s\rangle + \beta |\Psi\rangle \xrightarrow{D} (2 |u\rangle\langle u| - I)(\alpha |s\rangle + \beta |\Psi\rangle) = 2(\alpha \langle u|s\rangle + \beta \langle u|\Psi\rangle) |u\rangle - \alpha |s\rangle - \beta |\Psi\rangle$$

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

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 arbitray state $|\varphi\rangle$, we simply get a rotation in this space of $2\theta_0$, where $\theta_0$ is the initial angle between $|u\rangle$ and $|\Psi\rangle$:




That is, the evolution of the angle is given by $\theta_0, 3\theta_0, 5\theta_0, \ldots, (2T+1)\theta_0$. 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 our the entire algorithm hinges on the angle $\theta_0$ 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 $\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$ 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 to 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 $m$: $2^n, 2^{n-1}, 2^{n-2}, \ldots, 4, 2$. 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 the 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.

### Consequences of Grover's algorithm

Consider a variant of Simon's problem:

**Collision**
*Function:* $f\colon \{0,1\}^n \to \{0,1\}^n$
*Promise:* $f$ is 1-to-1 or 2-to-1
*Goal:* Decide which

**Fact 15.** *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 16** (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. □

## 3.5 Polynomial method

Let's now look a generic technique for proving quantum query lower bounds known as the polynomial method. To start, it will be convenient to reinterpret the meaning of a quantum query. Namely, instead of using the oracle to apply a function, we use it to reveal a bit of a hidden string. To be clear, these models are identical, only the language we use is different. That is, for a function $f\colon \{0,1\}^n \to \{0,1\}$, we consider the (exponentially long) bit string defined by all its outputs: $x = x_1 x_2 \dots x_{2^n}$ where $x_i$ is the $i$th output of $f$ when the inputs are ordered in binary. The input to the oracle is now an index into this string of length $N := 2^n$.

Using this language, we will show that the acceptance probability of every quantum query algorithm can be viewed as a polynomial in the $x_i$ variables. To start, recall the generic form of a quantum query algorithm:

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

which alternates between unitary operations and the phase oracle for the hidden bitstring $x$ (once again, we've hidden the ancilla register for clarity). Let's start by looking at the affect of the first unitary and oracle call:

$$|0\rangle \xrightarrow{U_0} \sum_{i=1}^{N} \alpha_i |i\rangle \xrightarrow{O_x} \sum_{i=1}^{N} (-1)^{x_i} \alpha_i |i\rangle .$$

Our first observation is that for $x_i \in \{0,1\}$ we can rewrite $(-1)^{x_i}$ as $1 - 2x_i$, so the state becomes

$$\sum_{i=1}^{N} (1 - 2x_i)\alpha_i |i\rangle .$$

Repeating this argument for each query, we arrive at one of the central ideas for the polynomial method:

**Claim 17.** *After $t$ oracle calls, the amplitude of each classical basis state is a polynomial of degree $t$ in the variables $x_1, \ldots, x_N$.*

*Proof.* The proof follows immediately from the following ideas we saw before: 1) applying a phase oracle increases the degree of the polynomial by at most 1; and 2) applying any unitary does not increase the degree. □

Concretely, after $t$ oracle calls, let's write the state of our system as

$$\sum_{i=1}^{N} \alpha_i(x) |i\rangle ,$$

where each $\alpha_i(x)$ is a degree $t$ polynomial in $x_1, \ldots, x_N$. Let's imagine that we use the quantum algorithm to solve some decision problem where we accept if we measurement outcome falls within some set $S$. The acceptance probability in that case, would be

$$p(x) := \sum_{i \in S} |\alpha_i(x)|^2.$$

Notice that each $|\alpha_i(x)|^2$ is a polynomial of degree $2t$, so $p(x)$ must also be a polynomial of degree $2t$.

The central idea behind the polynomial method is that low-degree polynomials are "well-behaved" in many respects. If we're trying to devise a quantum algorithm that is solving a complex problem, but the acceptance probability of the algorithm is given by a low-degree polynomial, the polynomial might not be expressive enough to capture the complexity of the problem.

### Grover lower bound via the polynomial method

Let's use the polynomial method to tackle the unstructured search problem. We'll focus on the decision variant where we'd like to know if our hidden bitstring $x_1 \cdots x_N$ contains a 1. This variant is sometimes called "OR" since we're trying to determine if $x_1 \vee x_2 \vee \cdots \vee x_N$ is satisfied.

Our first key observation will be that the OR problem is symmetric: no matter what permutation we apply to the string $x$, the answer to "does there exist a 1 in the string?" should not change. This will allow us to create a *symmetric* polynomial that captures the acceptance probability of the quantum algorithm.

Specifically, let $p(x)$ be the degree $2t$ polynomial for the acceptance probability of a $t$-query quantum algorithm for the OR problem. Define $q(x)$ to be the polynomial which averages the $p(x)$ polynomials after applying a permutation:

$$q(x) := \frac{1}{N!} \sum_{\pi \in S_N} p(\pi \cdot x)$$

where we denote by $\pi \cdot x$ as the result of applying the permutation to the bit string $x$, i.e., $\pi \cdot (x_1 \cdots x_N) = x_{\pi^{-1}(1)} \cdots x_{\pi^{-1}(N)}$. By our previous observation (the problem's answer is invariant under permutation of the input string), $q(x)$ should also be at least the acceptance probability of the algorithm. Furthermore, $q(x)$ is now symmetric.

To be clear, $q(x)$ is a multivariate polynomial in $x_1, \ldots, x_N$. However, we now claim that because $q(x)$ is symmetric it can be written as a univariate polynomial $r(z)$ of the same degree where $z = \sum_{i=1}^{N} x_i$.

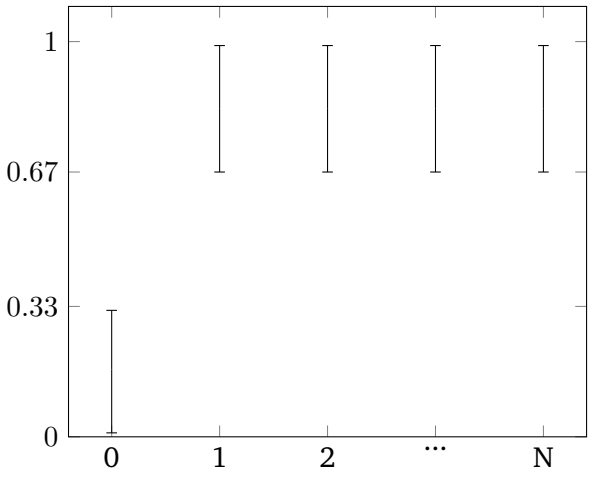
To show this, we will use the following fact: for every symmetric polynomial over Boolean variables, the coefficients of all terms of the same degree are equal. To prove this, simply take the smallest degree for which this is not true and consider the two terms with different coefficients. Setting the variables to be all 1's in one term and the rest 0's gives a different result from setting the variables to be all 1's in the other term and the rest 0's. Since both terms have the same number of variables, this is a contradiction because the function was supposed to be symmetric.

Let $\beta_i$ be the coefficient of any term in $q(x)$ which has degree $i$. Notice that our new variable $z = \sum_{i=1}^{N} x_i$ counts the number of variables which are 1. Therefore, using the above argument, we can write

$$q(x) = \sum_{i=0}^{\deg q} \beta_i \binom{z}{i} = \sum_{i=0}^{\deg q} \beta_i \frac{z(z-1)\cdots(z-i+1)}{i!} = r(z)$$

as the expression that counts how many terms in the original expansion of $q(x)$ had the same degree.

Let's return to our specific problem. To summarize, we have a polynomial $r(z)$ of degree $2t$ which captures the acceptance probability of the quantum algorithm after $t$ queries. If the quantum algorithm were perfect (i.e., had no error), then $r(0) = 0$ and $r(z) = 1$ for all $z \neq 0$. Since the quantum algorithm can err with probability at most $1/3$, the acceptance probabilities must have values in the following ranges:




To be clear, the polynomial $r(z)$ can do whatever it likes on non-integer points, but on the values $0, 1, \ldots, N$, it must fall within the specified ranges. We now want to show that any polynomial which has that behavior must necessarily have relatively high degree. Specifically, we can apply the Markov brothers' inequality:

**Theorem 18** (Markov brothers' inequality)**.** *If $p(x)$ is a polynomial, then*

$$\max |p'(x)| \leq \left| \frac{\max p(x) - \min p(x)}{N} \right| (\deg p)^2,$$

*where the max and min values are for $0 \leq x \leq N$ and $p'(x)$ denotes the first derivative.*

**Theorem 19.** *If $r(z)$ as above has the desired properties, then $t = \Omega(\sqrt{N})$.*

*Proof.* Plugging in $r$ to the Markov brothers' inequality and rearranging and weakening slightly gives us

$$\frac{N}{4t^2}\left(\max_{0\leq z\leq N}|r'(z)|\right) \leq \max_{0\leq z\leq N} r(z)$$

Let $M = \max_{0\leq z\leq N} r(z)$, and pick $z_0$ with $r(z_0) = M$ (we can do so since $[0, N]$ is compact). Let's analyze two possible cases:

$M < 2$: Note this is the "most likely" case, since we don't expect our function to go skyrocketing. In order to have $r(0) \leq 1/3$ and $r(1) \geq 2/3$, by the mean value theorem there must be some $z \in [0, 1]$ with $r'(z) \geq 1/3$. Plugging this into the brothers' inequality, we get

$$\frac{N}{12t^2} \leq \frac{N}{4t^2}\max|r'(z)| \leq \max r(z) < 2$$

so in this case we know that $t = \Omega(\sqrt{N})$.

$M \geq 2$: In this case, the mean value theorem implies that $|r'(c)| \geq 2(M-1)$ for some $c \in [\lfloor z_0 \rfloor, \lceil z_0 \rceil]$, since $r$ must return down to at most 1 for each integer, and the closest integer is at most $1/2$ away. We get $2N(M-1) \leq M(2t)^2$, so

$$\frac{N}{2t^2} \leq \frac{M}{M-1} \leq 2$$

and hence again we know that $t = \Omega(\sqrt{N})$. □

Combining everything together, we get a new proof of Theorem 10 using the polynomial method:

*Proof of Theorem 10:* Any quantum query algorithm that approximates OR with $t$ queries gives rise to a polynomial $r$ of degree $2t$. By Theorem 19, any such polynomial must have degree $\Omega(\sqrt{N})$. Hence, the total number of queries must be $\Omega(\sqrt{N})$. □

**Observation 20.** *Suppose we want to construct a quantum algorithm for OR that has perfect accuracy. That is, $r(z) = 1$ for $z = 1, 2, \ldots N$. The fundamental theorem of algebra requires a polynomial of $\deg r \geq N$, so we need at least $N/2$ queries.*

## Complexity of Parity

Considering the following simple problem:

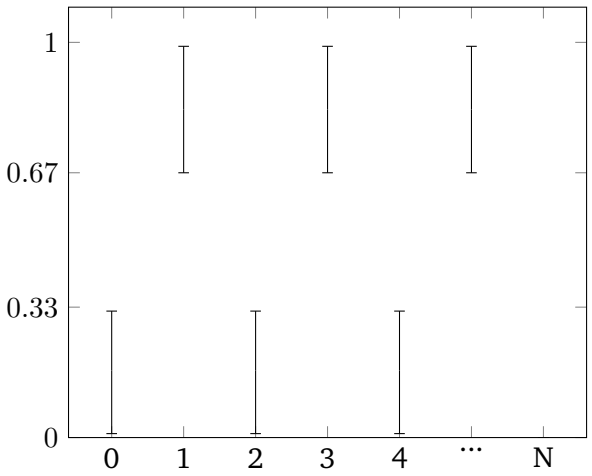
**Parity**
*Hidden string:* $x_1 \cdots x_N \in \{0,1\}^N$
*Goal:* Compute $x_1 \oplus \cdots \oplus x_N$

First, notice that we can use Deutsch's algorithm to compute parity with $N/2$ queries. To see this, first notice Deutsch algorithm can compute $x_j \oplus x_j$ for any pair of bits—in the

"constant" case, $x_i = x_j$ which implies $x_i \oplus x_j = 0$; in the "balanced" case, $x_i \neq x_j$ which implies $x_i \oplus x_j = 1$. Pairing up all the bits, we get an $N/2$ query bound. Somewhat surprisingly, this is exaclty tight:

**Theorem 21.** *The quantum query complexity of the parity function is precisely $N/2$.*

*Proof.* The parity function is symmetric, so running the polynomial method as above again gives us $r(z)$ which must now have values in these ranges:




In particular, $r(z) = 1/2$ at $N$ distinct values, one each between $i$ and $i+1$ for $0 \leq i < N$. Thus by the fundamental theorem of algebra, $\deg r \geq N$. Since the degree of $r$ is at most twice the number of queries, we must have made at least $N/2$ queries. □

## 3.6 Adversary method

Perhaps the simplest way to prove query lower bounds is through a powerful technique called the "adversary method". It is a general technique that can apply to all decision problems $\mathcal{P}\colon \{0,1\}^N \to \{0,1\}$, where $\mathcal{P}(x)$ indicates whether or not the hidden bitstring $x$ is part of the language.

For example, for the OR language, $\mathcal{P}(x) = 1$ iff $x$ contains a 1.

**Theorem 22** (Ambainis Adversary Method [Amb00])**.** *Suppose $\mathcal{P}\colon \{0,1\}^N \to \{0,1\}$. Let $X \subseteq \{0,1\}^N$ be a subset of 0-inputs and let $Y \subseteq \{0,1\}^N$ be a set of 1-inputs—that is, $\mathcal{P}(x) = 0$ for all $x \in X$; and $\mathcal{P}(y) = 1$ for all $y \in Y$. Let $R \subseteq X \times Y$ be any relation over the sets $X$ and $Y$ satisfying the following conditions:*

1. *For every $x \in X$, there are at least $m_0$ inputs $y \in Y$ such that $(x, y) \in R$.*
2. *For every $y \in Y$, there are at least $m_1$ inputs $x \in X$ such that $(x, y) \in R$.*
3. *For every $x \in X$ and $i \in \{1, \ldots, N\}$, there are at most $s_0$ inputs $y \in Y$ with $(x, y) \in R$ and $x_i \neq y_i$.*
4. *For every $y \in Y$ and $i \in \{1, \ldots, N\}$, there are at most $s_1$ inputs $x \in X$ with $(x, y) \in R$ and $x_i \neq y_i$.*

*The quantum query complexity of $\mathcal{P}$ is* $\Omega\left(\sqrt{\frac{m_0 m_1}{s_0 s_1}}\right)$.

Once again, let's use this to prove a lower bound for OR.

*Proof of Theorem 10:* Let $X = \{0^N\}$ be the set that only contains the all-zeros string and $Y = \{x : |x| = 1\}$ be the set containing those strings with exactly one 1. We also let $R = X \times Y$ be all pairs of strings in $X$ and $Y$. We have that, $m_0 = N$, $m_1 = 1$, $s_0 = 1$, $s_1 = 1$, so the query complexity is $\Omega(\sqrt{N})$. □

## Query lower bounds via reduction

Let's define a promise-free variant of the Collision problem:

**Element Distinctness**
*Function:* $f\colon \{0,1\}^n \to \{0,1\}^n$
*Goal:* Determine if there are distinct inputs $x, y \in \{0,1\}^n$ such that $f(x) = f(y)$

Recall that for the Collision problem, the goal was to find a collision when we were promised that $f$ was either 2-to-1 or 1-to-1. For Element Distinctness, we no longer have that promise. Indeed, $f$ could very close to a 1-to-1 function, and yet there could still be inputs which collide. Element Distinctness is clearly a harder problem than Collision, but... how much harder?

It turns out that proving a quantum query lower bound from scratch for Element Distinctness is quite challenging, but suppose we knew the following lower bound for the Collision problem:

**Theorem 23** ([Shi02])**.** *The quantum query complexity of Collision is* $\Omega(2^{n/3})$.

Our goal will be to reduce the Collision problem to the Element Distinctness problem, so that an efficient query algorithm for Element Distinctness implies an efficient query algorithm for Collision.

**Theorem 24.** *The quantum query complexity of Element Distinctness is* $\Omega(2^{2n/3})$.

*Proof.* Suppose there is a quantum query algorithm for Element Distinctness with $o(2^{2n/3})$ queries. We claim that this implies a quantum query algorithm for Collision with $o(2^{n/3})$ queries, contradicting the lower bound of Theorem 23.

Therefore, suppose we have some instance $f$ of Collison. Sample a random subset $R \subset \{0,1\}^n$ of size $|R| = 2^{n/2}$. By the Birthday Paradox (see Fact 15), with high probability, there exists $x, y \in R$ such that $f(x) = f(y)$ if $f$ is 2-to-1. Now run the Element Distinctness algorithm with $f$ restricted to the subset $R$ to determine whether a collision exists. The query complexity of the algorithm is then $o((2^{n/2})^{2/3}) = o(2^{n/3})$. We conclude that any quantum query algorithm for Element Distinctness must make $\Omega(2^{\frac{2n}{3}})$ oracle queries. □

# Chapter 4

# Complexity of Clifford circuits

Clifford operations are just those unitaries constructed from circuits of CNOT, Hadamard, and Phase gates. As we've seen throughout these notes, the Clifford gates have appeared in many quantum algorithms, and indeed, that have and will feature in many future quantum subroutines. As we will soon see, this is not always a coincidence. The Clifford gates have some extremely nice properties. Perhaps too nice! In fact, Clifford circuits can be efficiently simulated on a classical computer. Nevertheless, we will eventually see that this is not the end of the story. Clifford circuits show a surprising advantage over classical circuits when the comparision point is the depth of the circuit.

## 4.1 Clifford circuits, the gate definition

Let's start by defining Clifford circuits in the simplest possible way, by the list of gates that comprise them—controlled-not $(\mathrm{CNOT})$, Hadmard $(H)$, and Phase $(S)$. The set of all unitaries that arise from Clifford circuits is called the *Clifford group*. A *Cifford state* or a *stabilizer state* is obtained by applying a Clifford circuit to the all-zeros state.

To recap, the fundamental Clifford gates are

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}.$$

One way to start to understand the power and limitations of Clifford circuits is just to play around with them, applying variations to see what kinds of states you can create. One thing you will notice is that you get a discrete set of gates, and with a bit more mathematics, you can give an exact characterization of what these states look like. This will be our first approach to proving the famous Gottesman-Knill theorem:

**Theorem 25** (Gottesman–Knill [Got98])**.** *There is a classical polynomial-time algorithm to sample from the output distribution of any Clifford circuit.*

As previously mentioned, our plan to prove this theorem will be keep track of the state vector of the quantum state as we apply each Clifford gate in the circuit. Of course, we

can't write out the entire state vector, as this would require exponential space. It turns out, however, that Clifford states have the following special form:

**Lemma 26.** *[DM03, VDN10] Every $n$-qubit Clifford state can be written as*

$$|\psi\rangle = \frac{1}{\sqrt{|\mathcal{A}|}} \sum_{x \in \mathcal{A}} (-1)^{q(x)} i^{\ell(x)} |x\rangle$$

*where*

- *$\mathcal{A}$ is an affine space: $\mathcal{A} = \{My + b \pmod 2 \mid y \in \{0,1\}^r\}$ for $M \in \{0,1\}^{n \times r}$ and $b \in \{0,1\}^n$.*
- *$q(x)$ is a quadratic form: $q(x) = \sum_{i<j} q_{ij} x_i x_j$ with $q_{ij} \in \{0,1\}$.*
- *$\ell(x)$ is a linear form: $\ell(x) = \sum_i \ell_i x_i$ with $\ell_i \in \{0,1,2,3\}$.*

**Note:** I wanted to include a proof sketch here for completeness, but this is not the most common method of Clifford simulation, nor is it one that we'll be using much in this class. It's fine to skip to section 4.2, which is integral to much of what we'll talk about later in the class.

The natural proof of this statement also gives a classical simulation algorithm. That is, starting with the all-zeroes state (which is trivially of the above form), show how it evolves under the application of each one of the fundamental Clifford operations. Since each update to the state takes polynomial time, the entire computation will take polynomial time. By induction, we need to understand the following cases:

- **Apply $S$ on qubit $i$:**

  $$S|\psi\rangle = \frac{1}{\sqrt{|\mathcal{A}|}} \sum_{x \in \mathcal{A}} (-1)^{q(x)} i^{\ell(x)} S|x\rangle = \frac{1}{\sqrt{|\mathcal{A}|}} \sum_{x \in \mathcal{A}} (-1)^{q(x)} i^{\ell(x)} i^{x_i} |x\rangle$$

  In other words, if we let $\ell'(x) = l(x) + x_i \pmod 4$, then we have an updated representation of the state with Affine space $\mathcal{A}$, quadratic form $q(x)$, and linear form $\ell'(x)$.

- **Apply $\mathrm{CNOT}$ from qubit $i$ to qubit $j$:**

  $$\mathrm{CNOT}|\psi\rangle = \frac{1}{\sqrt{|\mathcal{A}|}} \sum_{x \in \mathcal{A}} (-1)^{q(x)} i^{\ell(x)} \mathrm{CNOT}|x\rangle = \frac{1}{\sqrt{|\mathcal{A}|}} \sum_{x \in \mathcal{A}} (-1)^{q(x)} i^{\ell(x)} |\mathrm{CNOT}x\rangle$$

  where the last equation reflects the fact that $\mathrm{CNOT}$ can be identified with an $n \times n$ Boolean matrix which XOR's the $i$th bit into the $j$th bit of the $n$-bit vector $x$. Therefore, we now have

  - *Affine space:* $\mathcal{A}' = \{M'y + b' \mid \forall y \in \{0,1\}^n$ for $M' = \mathrm{CNOT}M$ and $b' = \mathrm{CNOT}b$.
  - *Quadratic form:* Notice that we can write $q(x) = x^T Q x$ for some upper triangular matrix $Q$. Therefore, the updated quadratic form can be written as $q'(x) = q(\mathrm{CNOT}x) = x^T Q' x$ with $Q' = \mathrm{CNOT}^T \cdot Q \cdot \mathrm{CNOT}$.
  - *Linear form:* Similar to above, we can write $\ell(x) = \ell^T x$ for the vector $\ell = (\ell_1, \ldots, \ell_n)$. Therefore, the updated linear form can be written as $\ell'(x) = \ell(\mathrm{CNOT}x) = (\ell')^T x$ where $(\ell')^T = \ell^T \mathrm{CNOT}$.

- **Apply $H$:** The proof for applying $H$ is nontrivial, so we leave it for now. See proof in [VDN10].

In conclusion every Clifford state $|\psi\rangle$ can be written of the form in Lemma 26. Furthermore, the inductive proof reveals a polynomial-time algorithm to compute $\mathcal{A}, \ell, q$ of a state given the sequence of Clifford gates that construct the state.

## 4.2 Clifford circuits, the stabilizer picture

In the stabilizer picture of simulation [Got98], we do not represent a quantum state by its state vector, but rather as a list of "stabilizers" of the state, i.e., unitary matrices for which the original state vector is an eigenvector. It will turn out that this is a particularly useful representation for states generated by a Clifford circuits. Before we do this, however, let us describe a important subgroup of the Clifford group called the Pauli group that will be the basis of this representation.

### Pauli Group

The single-qubit Pauli group is generated by Pauli matrices, which are:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

The Pauli matrices also have a bunch of nice properties ($I$ is the $2 \times 2$ identity matrix):

- Hermitian: $X = X^\dagger$, $Y = Y^\dagger$, $Z = Z^\dagger$
- Square to the identity: $X^2 = Y^2 = Z^2 = I$
- Traceless: $\mathrm{Tr}(X) = \mathrm{Tr}(Y) = \mathrm{Tr}(Z) = 0$
- Same determinant: $\det(X) = \det(Y) = \det(Z) = -1$
- Anticommutation: $XY = -YX$, $XZ = -ZX$, $YZ = -ZY$
- Cyclic structure: $\begin{array}{lll} XY = iZ & YZ = iX & ZX = iY \\ YX = -iZ & ZY = -iX & XZ = -iY \end{array}$

These Pauli matrices form a group $\mathcal{P}_1$ of order 16 under matrix multiplication, where each element is of the form $\alpha P$ with $\alpha \in \{\pm 1, \pm i\}$ and $P \in \{I, X, Y, Z\}$. More generally, each element of the $n$-qubit Pauli group $\mathcal{P}_n$ is of the form $\alpha P_1 \otimes P_2 \otimes \cdots \otimes P_n$ where $\alpha \in \{\pm 1, \pm i\}$ and $P_i \in \{I, X, Y, Z\}$.

Although $|\mathcal{P}_n| = 4 \cdot 4^n$, $\mathcal{P}_n$ is generated by just $2n$ elements: Pauli elements with a single $Z$ and Pauli elements with a single $X$. For example, on 3-qubits, the generators are

| $Z$-elements | $X$-elements |
|---|---|
| $Z \otimes I \otimes I$ | $X \otimes I \otimes I$ |
| $I \otimes Z \otimes I$ | $I \otimes X \otimes I$ |
| $I \otimes I \otimes Z$ | $I \otimes I \otimes X$ |

and using the multiplication properties of the Pauli matrices, one can check that these do indeed generate the entire group.

We will often refer to the Pauli elements without the phase as the $n$-qubit Pauli matrices. In fact, the $n$-qubit Pauli matrices are particularly nice because they form a basis for all complex matrices:

**Fact 27.** *The $n$-qubit Pauli matrices form a basis for all complex $2^n \times 2^n$ matrices.*

*Proof.* Treat each $2^n \times 2^n$ matrix $A$ as a vector of length $4^n$ denoted by $\mathrm{vec}(A)$. Then, we can express inner products between matrices $A$ and $B$ as $\mathrm{vec}(A) \cdot \mathrm{vec}(B) = \mathrm{Tr}(AB^\dagger)$.

We now claim that all $4^n$ Pauli matrices are linearly independent. To see this, let $P = P_1 \otimes \cdots \otimes P_n$ and $Q = Q_1 \otimes \cdots \otimes Q_n$ be two distinct $n$-qubit Pauli matrices. We have that

$$\mathrm{Tr}(PQ) = \mathrm{Tr}(P_1Q_1 \otimes \cdots \otimes P_nQ_n) = \mathrm{Tr}(P_1Q_1) \cdots \mathrm{Tr}(P_nQ_n) = 0$$

since there must exist at least some index $i$ for which $P_i \neq Q_i$. In more detail, notice that when $P_i \neq Q_i$ for $P_i, Q_i \in \{I, X, Y, Z\}$ that $P_iQ_i = \alpha R$ for $\alpha \in \{\pm 1, \pm i\}$ and $R \in \{X, Y, Z\}$. Therefore, $\mathrm{Tr}(P_iQ_i) = \alpha \,\mathrm{Tr}(R) = 0$ since the Pauli matrices are traceless.

Since we have a space of dimension $4^n$ and all $4^n$ Pauli matrices are linearly independent, we must have that the span of the Pauli matrices is the entire space. □

As a special case, we can look at the Pauli decomposition for density matrices of pure states.

**Fact 28.** *Let $|\psi\rangle$ be an $n$-qubit pure state. The density matrix $|\psi\rangle\langle\psi| = \sum_{P \in \{I,X,Y,Z\}^{\otimes n}} \alpha_P P$ with $\alpha_P \in \mathbb{R}$ and $\sum_P \alpha_P^2 = 2^{-n}$.*

*Proof.* By Fact 27, we can write $|\psi\rangle\langle\psi| = \sum_P \alpha_P P$ where $P \in \{I, X, Y, Z\}^{\otimes n}$ and $\alpha_P \in \mathbb{C}$. Since the Pauli matrices are Hermitian, we have

$$|\psi\rangle\langle\psi| = \sum_P \alpha_P P = \sum_P \alpha_P^* P.$$

This implies that $\alpha_P = \alpha_P^*$ since the $P$ are linearly independent, which in turn implies that the $\alpha_P$ coefficients are real. Furthermore, using the purity of $|\psi\rangle$ we have

$$1 = \mathrm{Tr}(|\psi\rangle\langle\psi|) = \mathrm{Tr}(|\psi\rangle\langle\psi| \cdot |\psi\rangle\langle\psi|) = \sum_{P,Q} \alpha_P \alpha_Q \,\mathrm{Tr}(PQ) = \sum_P \alpha_P^2 \,\mathrm{Tr}(I^{\otimes n}) = 2^n \sum_P \alpha_P^2$$

where we've used that $\mathrm{Tr}(PQ) = 0$ for $P \neq Q$, $P^2 = I^{\otimes n}$, and $\mathrm{Tr}(I^{\otimes n}) = 2^n$. □

## Pauli matrices and stabilizer groups

Now that we have defined the Pauli group, let's use it to help us represent a quantum state.

**Definition 29.** *For $n$-qubit state $|\psi\rangle$, we say unitary $U$ stabilizes $|\psi\rangle$ iff $U|\psi\rangle = |\psi\rangle$. Let the stabilizer group $\mathrm{Stab}(|\psi\rangle) \subseteq \mathcal{P}_n$ be the set of all Pauli elements that stabilize $|\psi\rangle$.*

**Fact 30.** *$\mathrm{Stab}(|\psi\rangle)$ is an Abelian group under matrix multiplication.*

*Proof.* If Pauli elements $P$ and $Q$ both stabilize $|\psi\rangle$, then so do $PQ$ and $P^\dagger$.

To argue that this group must be Abelian, notice that any two Pauli's $P$ and $Q$ either commute ($PQ = QP$) or anti-commute ($PQ = -QP$). Suppose that $P$ and $Q$ anti-commute. We get the following contradiction:

$$|\psi\rangle = PQ\,|\psi\rangle = -QP\,|\psi\rangle = -\,|\psi\rangle\,.$$

Therefore, $P$ and $Q$ must commute, and the stabilizer group is Abelian. □

Does the stabilizer group constitute a reasonable representation state? By a counting argument, one can see that there exist many quantum states whose stabilizer groups are empty, so this stabilizer representation won't be very good for them. On the other hand, if the stabilizer group is large enough, then it is a unique representation of the state:

**Fact 31.** *For any stabilizer group of size $2^n$, there is only one state (up to global phase) with that stabilizer group. Additionally, for any stabilizer state $|\psi\rangle$, we have*

$$|\psi\rangle\langle\psi| = \frac{1}{2^n} \sum_{P \in \mathrm{Stab}(|\psi\rangle)} P.$$

*Proof.* Let $|\psi\rangle$ be an $n$-qubit state with $|\mathrm{Stab}(|\psi\rangle)| = 2^n$. Let $|\varphi\rangle\langle\varphi|$ be the density matrix of any state stabilized by every element in $\mathrm{Stab}(|\psi\rangle)$. We claim that this density matrix is unique. To see this, first expand $|\varphi\rangle\langle\varphi|$ in the Pauli basis using Fact 28: $|\varphi\rangle\langle\varphi| = \sum_P \alpha_P P$. Now take any $Q \in \mathrm{Stab}(|\psi\rangle)$. We have

$$1 = \mathrm{Tr}(|\varphi\rangle\langle\varphi|) = \mathrm{Tr}(Q\,|\varphi\rangle\langle\varphi|) = \sum_P \alpha_P \,\mathrm{Tr}(QP) = \alpha_Q \,\mathrm{Tr}(I^{\otimes n}) = \frac{\alpha_Q}{2^n}$$

where we have used (in order) that $Q$ stabilizes $|\varphi\rangle$, that $\mathrm{Tr}(QP) = 0$ for $Q \neq P$, and that $Q^2 = I^{\otimes n}$ for any Pauli. In other words, for each of the $2^n$ stabilizers in $\mathrm{Stab}(|\psi\rangle)$, the corresponding coefficient in the Pauli expansion is $2^{-n}$. Notice that this implies all other Pauli coefficients must be zero since by Fact 28 we have

$$2^{-n} = \sum_P \alpha_P^2 = \sum_{Q \in \mathrm{Stab}(|\psi\rangle)} \alpha_Q^2 + \sum_{P \notin \mathrm{Stab}(|\psi\rangle)} \alpha_P^2 = 2^{-n} + \sum_{P \notin \mathrm{Stab}(|\psi\rangle)} \alpha_P^2.$$

Since the $\alpha_P$ coefficients are real, their squares must be non-negative. On the other hand, the above equation implies that $\sum_{P \notin \mathrm{Stab}(|\psi\rangle)} \alpha_P^2 = 0$, so they must all be zero. □

Because of this, let's focus our attention on states that have stabilizer groups of size $2^n$. This raises the obvious question: which states have these large stabilizer groups? Well, to start, notice that the all-zeroes state is stabilized by every Pauli matrix which is a tensor product of identity matrices ($I$) and Pauli-$Z$ matrices. There are $2^n$ such matrices, so they comprise the entire stabilizer group.

We now have a stabilizer representation of our initial state. A reasonable requirement is that we can determine how the stabilizer group changes when we apply a unitary to the state:

**Fact 32.** *$U$ stabilizes $|\psi\rangle$ iff $VUV^\dagger$ stabilizes $V\,|\psi\rangle$.*

*Proof.* $|\psi\rangle = U|\psi\rangle \iff V|\psi\rangle = VU|\psi\rangle = (VUV^\dagger)V|\psi\rangle$ □

In other words, if we apply a gate to our state, then we can update the stabilizer group representation by conjugating every stabilizer by the gate. In general, we don't have any guarantee on the form of $VUV^\dagger$. That is, even if $U$ is a Pauli matrix, it's conjugation under an arbitrary unitary might not be Pauli.

We are now ready to reveal the key feature of Clifford circuits:

**Theorem 33.** *The Clifford group is the* normalizer *of the Pauli group. That is, a unitary $U$ is Clifford iff $UPU^\dagger$ is in the Pauli group for all Pauli matrices $P$.*

For now, we just describe the direction which is important Clifford circuit simulation: if $U$ is Clifford, then $UPU^\dagger$ is in the Pauli group. To make our lives easier, we will simplify down to just a few special cases that we have to check:

- *Only have to check* CNOT*, $H$, and $S$:* Since $U$ is Clifford, we can write $U = g_1 \cdots g_m$ as a product of CNOT, $H$, and $S$ gates. Therefore, if each gate $g_i$ maps Pauli elements to Pauli elements under conjugation, then we have

  $$UPU^\dagger = g_1 \cdots g_{m-1}(g_m P g_m^\dagger) g_{m-1}^\dagger \cdots g_1 = g_1 \cdots g_{m-2}(g_{m-1} P' g_{m-1}^\dagger) g_{m-2}^\dagger \cdots g_1 = \cdots$$

  is another Pauli matrix.

- *Only have to check generators of the Pauli group:* Recall that every $n$-qubit Pauli $P$ can be expressed as the product of $2n$ different generators, which consist of the Pauli elements with a single $Z$ term and a single $X$ term. Therefore, if we specify how a unitary affects each such generator under conjugation, then we can determine its more general behavior. Namely, if $P = P_1 P_2 \cdots P_k$ for Pauli generators $P_i$, then

  $$UPU^\dagger = UP_1P_2 \cdots P_k U^\dagger = (UP_1U^\dagger)(UP_2U^\dagger)U \cdots U^\dagger(UP_kU^\dagger)$$

  for any unitary $U$.

To complete the proof, we can simply show how each of CNOT, $H$, and $S$ affects the Pauli generators:

| $P$ | $HPH^\dagger$ |
|---|---|
| $X$ | $Z$ |
| $Z$ | $X$ |

| $P$ | $SPS^\dagger$ |
|---|---|
| $X$ | $Y$ |
| $Z$ | $Z$ |

| $P$ | $\mathrm{CNOT}P\mathrm{CNOT}^\dagger$ |
|---|---|
| $X \otimes I$ | $X \otimes X$ |
| $I \otimes X$ | $I \otimes X$ |
| $Z \otimes I$ | $Z \otimes I$ |
| $I \otimes Z$ | $Z \otimes Z$ |

As a direct consequence, we can simulate Clifford circuits by keeping track of the stabilizer group and how it changes under the application of each gate in the circuit.

## 4.3 Simulation of Clifford circuits using stabilizer groups

There are two remaining issues to address in order to obtain an efficient classical simulation of Clifford circuits using stabilizer groups. The first is a question of efficiency: if we need to keep track of all $2^n$ stabilizer elements, then our algorithm would take exponential time. However, once again, we only need to keep track of the *generators* of the stabilizer group:

**Fact 34.** *Let $|\psi\rangle$ be an $n$-qubit Clifford state. There exists $n$ Pauli generators $g_1, \dots g_n \in \mathcal{P}_n$ such that every $P \in \mathrm{Stab}(|\psi\rangle)$ can be expressed as the product of generator elements. Furthermore, the generators are independent in the sense that no generator can be expressed as the product of the other generators.*

*Proof.* Recall that the stabilizer group of the all-zeroes state consists of all the $Z$-type Pauli elements. One can check that this group is generated by the Pauli matrices with a single $Z$ term: $Z \otimes I \otimes \cdots \otimes I$, $I \otimes Z \otimes \cdots \otimes I$, ..., $I \otimes I \otimes \cdots \otimes Z$. There are $n$ such generators, and it is of minimal size.

Since every Clifford state is of the form $U|0^n\rangle$ for Clifford unitary $U$, we have that the stabilizer group is generated by $UZ_iU^\dagger$ where $Z_i$ is the Pauli matrix with a single $Z$ in the $i$th register. □

It's worth noting that although every stabilizer group can be represented by $n$ generators $g_1, \dots, g_n$, this representation is far from unique. In particular, one can check that multiplying the first generator into the second yields a new set of generators $g_1, g_1g_2, \dots, g_n$. In some cases, this idea will allow us to simplify our set of stabilizer generators using a multiplicative version of Gaussian elimination.

Let's turn our attention to the final issue: how do we deal with measurements? Without loss of generality, let's just focus on a computational basis measurement on the first qubit. There are two cases:

*Deterministic Measurement:* This occurs when the state is either $|0\rangle \otimes |\psi'\rangle$ or $|1\rangle \otimes |\psi'\rangle$. In other words, measuring the state results in $|0\rangle$ or $|1\rangle$ with probability 1, and it doesn't change the state. Even though we don't need to update our stabilizer group representation, there are still two potential issues: how do we determine if the measurement will be deterministic? and how can we determine the outcome of the measurement?

For the first problem, notice that there cannot be any Pauli stabilizers of the form $X \otimes P$ or $Y \otimes P$ for $P \in \mathcal{P}_{n-1}$ because both stabilizers would flip the first qubit. We claim that if these stabilizers are not present, then the measurement will be deterministic. To see this, notice that a computational basis measurement projects the first qubit onto $|0\rangle$ or $|1\rangle$. We can express this projection as $|0\rangle\langle 0| = (I + Z)/2$ and $|1\rangle\langle 1| = (I - Z)/2$, respectively. Notice, however that if $g$ stabilizes $|\psi\rangle$ and it is of the form $I \otimes P$ or $Z \otimes P$, then it still stabilizes the state after projection:

$$g\left(\frac{I \pm Z}{2} \otimes I \otimes \cdots \otimes I\right)|\psi\rangle = \left(\frac{I \pm Z}{2} \otimes I \otimes \cdots \otimes I\right) g\,|\psi\rangle = \left(\frac{I \pm Z}{2} \otimes I \otimes \cdots \otimes I\right)|\psi\rangle\,.$$

Since the stabilizer group has not changed and the stabilizer group is unique (Fact 31), the measurement must have been deterministic.

In summary, we can now easily detect whether or not the measurement will be deterministic by checking if all of the stabilizer generators start with either an $I$ or a $Z$. To determine the result of the measurement, we must now learn whether or not the state is of the form $|0\rangle \otimes |\psi'\rangle$ or $|1\rangle \otimes |\psi'\rangle$. Notice that the first state is stabilized by $Z \otimes I \otimes \cdots \otimes I$, while the second state is stabilized by $-Z \otimes I \otimes \cdots \otimes I$. We simply need to decide which. We can find it using Gaussian elimination in time $O(n^3)$.

*Random Measurement:* Since the measurement is not deterministic, there must exist some stabilizer generator that starts with either an $X$ or a $Y$. Using Lemma 26, one can check that the measurement result is either $|0\rangle$ or $|1\rangle$ with $50\%$ probability. Therefore, it is easy to output the result of the measurement. The difficulty is in updating the stabilizer representation.

Once again, the stabilizers that start with $I$ or $Z$ remain in the stabilizer group because they still stabilize the state after projection. On the other hand, we do need to remove the stabilizers which start with $X$ or $Y$ because they anti-commute with the projection. There is a fairly nice way of doing this: take one of the stabilizer generators that starts with an $X$ or $Y$ and multiply it into all of the remaining stabilizer generators that start with an $X$ or $Y$. One can check that now all the stabilizer generators start with either an $I$ or $Z$ except for one. Now, remove the remaining stabilizer generator that starts with an $X$ or $Y$. What's left are $n-1$ stabilizer generators, so we only need to add one more. If the measurement outcome was $|0\rangle$, add the stabilizer generator $Z \otimes I \otimes \cdots \otimes I$ and if it was $|1\rangle$ add the stabilizer generator $-Z \otimes I \otimes \cdots \otimes I$. This completes the measurement protocol. It takes time $O(n^2)$.

# 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.

[Amb00] Andris Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, pages 636–643, 2000.

[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.

[DM03] Jeroen Dehaene and Bart De Moor. Clifford group, stabilizer states, and linear and quadratic operations over GF(2). *Physical Review A*, 68(4), oct 2003.

[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.

[Got98] Daniel Gottesman. The Heisenberg representation of quantum computers, 1998.

[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.

[Shi02] Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 513–519. IEEE, 2002.

[VDN10] Maarten Van Den Nest. Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond. *Quantum Information & Computation*, 10(3):258–271, 2010.

# Appendix A

# Classical complexity

In this appendix, we review some of the important concepts from classical complexity theory.

## A.1 Complexity classes for decision problems

For this section, let's suppose all computation is done over the binary alphabet $\{0,1\}^*$. A *language* $L \subseteq \{0,1\}^*$ is simply a set of strings. In this section, we will focus on decision problems, where the goal is the compute membership in language or promise language. A *complexity class* is a collection of languages recognized by a particular model of computation.

The complexity classes below are defined in terms of classical Turing machines. The more resources we give the Turing machine (e.g., time, space, randomness, or nondeterminism), the more languages that model of Turing machine can recognize.

**Polynomial Time (P):**
Languages $L$ such that there exists a deterministic poly-time Turing machine $M$ such that $M$ accepts $x$ iff $x \in L$.

**Non-deterministic Polynomial Time (NP):**
Language $L$ such that there exists deterministic poly-time Turing machine $M$ and polynomial $q$ such that for all $x \in \{0,1\}^n$

- If $x \in L$, $\exists y \in \{0,1\}^{q(n)}$ such that $M(x,y)$ accepts
- If $x \notin L$, $\forall y \in \{0,1\}^{q(n)}$, $M(x,y)$ rejects.

NP is a generalization of P (just forget about the witness string $y$), so P $\subseteq$ NP. It is widely conjectured that P $\neq$ NP, but we do not have a proof!

**Bounded-error Probabalistic Polynomial Time (BPP):**
Languages $L$ such that there exists a deterministic poly-time Turing machine $M$ and polynomial $q$ such that for all $x \in \{0,1\}^n$

- If $x \in L$, then $M(x,y)$ accepts for at least $2/3$ of the strings $y \in \{0,1\}^{q(n)}$.
- If $x \notin L$, then $M(x,y)$ accepts for at most $1/3$ of the strings $y \in \{0,1\}^{q(n)}$.

That is, $y$ is a random string given to the Turing machine. Once again, it is clear that $\mathsf{P} \subseteq \mathsf{BPP}$ since we can just forget about the extra random bits. However, we do not know if $\mathsf{BPP} \subseteq \mathsf{NP}$ or if $\mathsf{NP} \subseteq \mathsf{BPP}$, though it is widely conjectured that $\mathsf{P} = \mathsf{BPP}$.

**Probabilistic Polynomial Time (PP):**
Languages $L$ such that there exists a deterministic poly-time Turing machine $M$ and polynomial $q$ such that for all $x \in \{0,1\}^n$

- If $x \in L$, then $M(x,y)$ accepts for more than $1/2$ of strings $y \in \{0,1\}^{q(n)}$.
- If $x \notin L$, then $M(x,y)$ accepts at most $1/2$ of strings $y \in \{0,1\}^{q(n)}$.

Notice that the definition PP to identical to that of BPP except with a smaller gap between acceptance and rejection probabilities. Therefore, we have that $\mathsf{BPP} \subseteq \mathsf{PP}$. In fact, PP is powerful enough even to contain NP. To see this, take any NP machine $M$, and alter it in the following way. If $M(x,y)$ accepts, then accept. If $M(x,y)$ rejects, then flip an unbiased coin (to be completely rigorous, one would need to extend the length of the random string $y$)—if heads, accept, and if tails, reject. Notice that if there are no accepting $y$ for the original machine, then the new machine accepts with exactly $50\%$ probability. However, if there is any accepting $y$, then the new machine accepts with greater than $50\%$, and so the inclusion $\mathsf{NP} \subseteq \mathsf{PP}$ follows.

**Polynomial Space (PSPACE):**
Languages $L$ such that there exists a deterministic Turing machine $M$ that uses at most polynomial space and $M$ accepts $x$ iff $x \in L$.

We have that $\mathsf{PP} \subseteq \mathsf{PSPACE}$ since a PSPACE machine can simply count all the $y \in \{0,1\}^{q(n)}$ that make a poly-time Turing machine accept. There are exponentially many such $y$, but this is not an issue since we can erase the previous computation as we are enumerating over all the $y$.

**Exponential Time (EXP):**
Languages $L$ such that there exists a deterministic Turing machine $M$ and a polynomial $q$ such $M$ halts in $2^{q(n)}$ time and $M$ accepts $x$ iff $x \in L$.

We have that $\mathsf{PSPACE} \subseteq \mathsf{EXP}$ because a Turing machine that uses polynomial space can only have exponentially many configurations. And, if you were to reach the same configuration twice, then you will be in an infinite loop, so you might as well halt.

Figure A.1 shows a summary of how the complexity classes introduced above relate to each other.

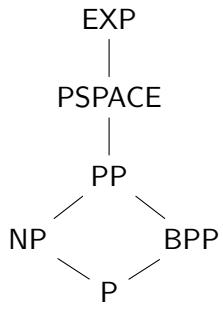



Figure A.1: Inclusion diagram of classical complexity classes. $A$ is below $B$ if $A \subseteq B$.