An Introductory Tutorial to Quantum Computing Algorithms

Author: Advay Mansingka[1*]
Faculty Advisor: Dmitri Feldman[2]
Language Editor: Aditi Marshan[1]

Dated: 12th July 2021

[1] Undergraduate Student, Brown University, Providence, Rhode Island, 02912, USA
[2] Department of Physics, Brown University, Providence, Rhode Island, 02912, USA

* Corresponding Author. Email Address: advay_mansingka@brown.edu

Table of Contents

0. Abstract

The goal of this tutorial is to serve as a brief introduction to quantum computing, as well as to some algorithms and circuits used in quantum computers. This paper presents an overview of some mathematical concepts in classical and quantum computing, including some linear algebra, ket notation, and logical operations in computing. It then uses this foundation to demonstrate how the Deutsch-Jozsa algorithm and Shor's algorithm allow quantum computers to solve some types of problems exponentially faster than classical computers. Quantum circuitry and operations are briefly discussed to aid the understanding of these algorithms. Next, the tutorial details some conventional sorting algorithms, and one possible solution for a quantum sorting algorithm. Finally, the tutorial presents some more esoteric concepts: quantum entanglement and quantum teleportation. These can be unintuitive, but they can allow quantum computers to coordinate and communicate data in ways that were previously impossible. The tutorial is primarily aimed at those with a basic understanding of quantum mechanics and linear algebra, however most of the concepts discussed are self contained in the paper.

1. Introduction

Since the 1960s, computers have slowly become more and more powerful, and increasingly integral to life as we know it. Now, computers are ubiquitous in society, and almost everything we do these days is influenced by technology in some way. In the last few decades, processors have become exponentially smaller, and more efficient. Some circuit components are even approaching the size of a single atom [1]. This is incredibly exciting — but also problematic. If transistors continue to decrease in size, their ability to reliably regulate the flow of electrons in a circuit may become compromised by quantum effects, such as tunnelling [2]. So, how do we continue to make our machines more efficient and powerful, if we are reaching the limits of what is possible with conventional computers?

One possible solution is to rethink the fundamental way in which computers deal with data. In classical computers, data is stored in transistors using bits. These bits are a binary/Boolean data type, with two states: 0 and 1. This means that you need n bits to store 2^n pieces of information. Perhaps, if we can increase the number of possible states for our data type to a number m, where $m > 2$, we could store m^n pieces of information with the same number of data stores. However, actually implementing a ternary ($m = 3$) or quaternary ($m = 4$) computer has not been very practical [3]. We need to think of an alternate approach to making computing more efficient.

This leads us to the idea of a quantum bit [4], often called a "qubit" for short. Similar to classical bits, a qubit is also a binary data type, however unlike bits, a qubit doesn't have to be in just one of the two states. Rather, it has a certain probability of being in the 0 state, and a certain probability of being in the 1 state. Until the qubit is measured, it is in any proportion of both states simultaneously owing the to the principle of superposition. A qubit can be any two level quantum system, such as the spin of an electron, or the polarisation of a photon. As long as the qubit is unobserved, it has a possibility of being in either state, but then collapses to a single definite state once it is observed. Thus, n qubits can be in 2^n configurations simultaneously [5]. Figure 1 is a graphical representation of this.

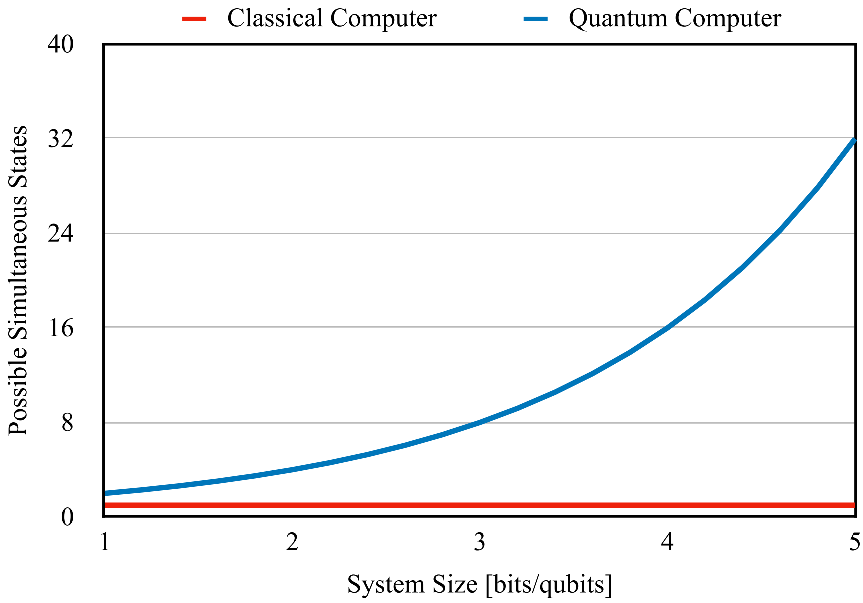

Figure 1: Qubits allow data to be in 2^n states simultaneously, as opposed to classical computers, which can only compute with 1 state at a time. [5]

This change in the data management means that a quantum computer could make certain calculations in exponentially less time compared to a classical computer. Of course, in practical applications qubits will not be error free, and we will have to build in redundancies in the system to account for potential errors in reading and storing qubits, but the gain in compute speed is still very significant.

This may not be helpful in many day to day tasks, like checking your email, or writing papers, but it has amazing implications for tasks that require a large amount of data to be processed. Because of the way compute time scales for some quantum algorithms, running progressively larger data sets and problems results in a relatively small change in processing time. Some examples of such tasks are sorting through massive lists, factorising large numbers to break encryption, and running simulations. Many algorithms exist that allow for the use of qubits to achieve these goals. In this tutorial, I will explore some of these algorithms, and demonstrate how quantum computing can outperform classical computing in certain scenarios.

This topic might become very relevant in the future, as companies like Google and IBM have demonstrated that it is possible to build quantum computers that can outperform classical computers — albeit this is currently only true on a small scale [6].

2. Bits, Logic, and Linear Algebra

As I briefly touched upon in the introduction, bits are the fundamental building block for any task in computing. In this section, I will elaborate on how we can represent them mathematically, as well as some operations we can perform on them. That will serve as a jumping off point to understand some more intriguing concepts, such as quantum logic operations and the unit circle state machine. These concepts serve as the foundation for the rest of this tutorial.

2.1 Classical Bits and Operations

A bit is a boolean data structure. This means it can have one of two states — false or true. In computer science we assign the false state as a 0. We can also represent this as a vector:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{1}$$

Similarly, we assign the true state as a 1, which can be represented by the vector:

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2}$$

For brevity, I use Dirac ket notation in some places in this paper. In this notation, we represent the false as $|0\rangle$ and true as $|1\rangle$. These simple values can be used to represent everything in classical computing. There are four ways we can operate on a single bit:

1. Identity, which keeps the output bit the same as the input bit.
2. Negation, which makes the output bit the opposite of the input bit.
3. Constant-0, which sets the output to $|0\rangle$ regardless of the input bit.
4. Constant-1, which sets the output to $|1\rangle$ regardless of the input bit.

We can think of all four of these operations as multiplying a 2x2 matrix which represents the operator with a vector representing the input bit. For example, performing negation on a $|0\rangle$ bit may be written as:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3}$$

These four operations can be classified into two types. The identity and negation operators are both variable operations. If I apply either of these operation to an unknown bit, and then I measure the output, I can easily deduce what the unknown input bit was. I can also apply the same operator a second time, to convert the measured output bit back into the input bit. The operations are reversible, and are their own inverse.

The Constant-0 and Constant-1 operations, on the other hand, are non-reversible. If an unknown bit is run through either of these operators, and then I measure the output bit, I will have no way of knowing what the input was. The input information has been lost forever, and the thus these operations cannot be reversed.

While single bit operations are intriguing, operations involving multiple bits are what make computers as potent as they are today. To represent multiple bits as a vector, we take the tensor product of the individual bits it contains. As a quick refresher, the formula used to compute the tensor product of two vectors is:

$$\begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix} \Rightarrow \begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix} \tag{4}$$

For example, we can write the (base 10) number 6 in binary as 110. In vector form this looks like:

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \Rightarrow |110\rangle \tag{5}$$

This tensored result is called the product state, and it can be factored back down into each of the three individual bits that we used to build it. The factored out notation on the left hand side of the above equation is called the individual state representation. If n bits are being represented, the product state vector will have a length of 2^n. Thus, it would be very challenging to write any real-world data as a product state vector. This demonstrates how convenient ket notation can be when writing documents involving bits and bit operations.

In a multiple bit vector, the first digit is called the 'most significant bit' (MSB for short) and the last digit is called the 'least significant bit' (LSB for short). For example in $|10\rangle$, 1 is the MSB and 0 is the LSB.

Operations on multiple bits are particularly powerful: the logical operators AND, OR, NAND, NOR, XOR, and XNOR are the backbone of all modern technology. Since these are fairly straightforward, and widely known, I will not waste time detailing them too much. Instead, I will focus on explaining the conditional not gate [5][7] (also sometimes called the controlled not gate), as this is the most relevant for the rest of this tutorial. We write it as CNOT for short.

CNOT operates on a pair of bits, one of which becomes the "control" bit, and the other becomes the "target" bit. By convention, we make the MSB the controls and the LSB the target. If the control bit in a CNOT is 1, the target bit gets put through a negation operator. If the control bit is 0, the target bit gets put through an identity operator. The control bit is always unchanged, as it passes through a CNOT gate. A graphical representation of this is shown in Figure 2:

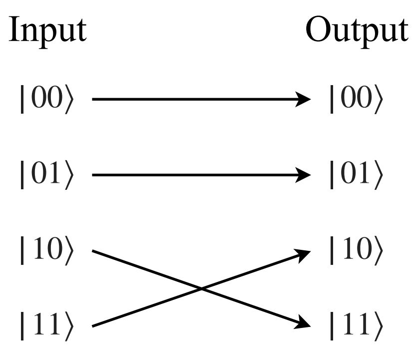

Figure 2: The working of a CNOT gate, using the MSB as the control and the LSB as the target. [7]

Just like with the other operators, we can represent the CNOT as a matrix. Since the CNOT gate operates on two bits, it is a 4x4 matrix:

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{6}$$

As we can see from the matrix, the CNOT operation is reversible, and is its own inverse. This operation is not very common in classical computers (they are usually built around NAND [8]), but it is extremely useful within the field of quantum computing.

2.2 Quantum Bits and Superposition

In the last section we saw how we could easily represent classical bits as vectors. This brings us to our next big question — how do we represent qubits? The simple answer is that they follow exactly the same notation as classical bits. In fact, classical bits are just a special case of qubits. To represent a qubit, we can take a vector:

$$\begin{pmatrix} a \\ b \end{pmatrix} \tag{7}$$

Where a and b are both complex numbers. We also need to use the normalisation condition:

$$|a|^2 + |b|^2 = 1 \tag{8}$$

Considering these constraints and the format, we see that classical bits satisfy all these criteria. However, the classical bits are just one of an infinite number of solutions to the condition. We know that our qubit is boolean, so it must result in either a $|0\rangle$ or a $|1\rangle$. How, then, can a qubit have a value that is neither?

The answer is superposition. Instead of being a $|0\rangle$ or a $|1\rangle$, the qubit has a certain probability of being a $|0\rangle$, and a certain probability of being a $|1\rangle$. Thus, the qubit is in both states (and neither state) at the same time. It has a probability $|a|^2$ of being a $|0\rangle$ and a probability $|b|^2$ of being a $|1\rangle$. When the qubit is measured, it collapses into one of the possible states, but until then, its value is unknown. For example a qubit with a value:

$$\begin{pmatrix} \frac{\sqrt{3}}{2} \\ \frac{i}{2} \end{pmatrix} \tag{9}$$

has a 75% chance of collapsing to a $|0\rangle$ and a 25% chance of collapsing to a $|1\rangle$ when it is measured. We can simplify the notation representing this superposition as: $|\psi\rangle = a|0\rangle + b|1\rangle$, where $|\psi\rangle$ now represents the probabilistic state of the qubit.

Representing multiple qubits also works similarly to the representation of multiple classical bits, using the tensor product of each individual bit to find a product state. The key difference, is that a value with length n qubits can now actually represent 2^n classical bits at the same time, thanks to superposition, as shown in Figure 1. For example the two qubit system:

$$\begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix} \otimes \begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix} \Rightarrow \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \tag{10}$$

has an equal $\left|\frac{1}{2}\right|^2 \Rightarrow 25\,\%$ chance of being $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$ at the same time.

2.3 Working with Qubits

In section 2.1, we introduced two kinds of operations: reversible operations and non-reversible operations. Fortunately for us, all quantum computing operations are reversible [9]. Additionally, just like identity and negation, most operations in quantum computers are their own inverses. This means that we don't need to worry about losing information when dealing with quantum algorithms, and we can always double check our input bits by running our measured bits through our operations a second time. Unlike in classical computing, the values of input qubits in a system are almost always unknown, as they can exist in multiple states at the same time. Thus, the ability to reverse our operations and double check out operations makes our processes a lot more straightforward.

Mathematically, the same matrix multiplication method we used on classical bits can be used to apply the identity, negation, and CNOT operations on qubits. Here, the operators don't deterministically change the state of the qubit from a $|0\rangle$ to a $|1\rangle$, or vice-versa. Rather, they change the probabilities that the qubit will collapse into either of the states.

Additionally, there are some operators that can't be applied in the classical computing world, but are extremely useful in quantum computing. One such operator is the Hadamard gate [10]. This gate operates on only one qubit at a time. The gate takes any qubit that is in a $|0\rangle$ state or a $|1\rangle$ state, and puts it into exact superposition so that the qubit now has an equal probability of collapsing into a $|0\rangle$ or a $|1\rangle$. We can see this mathematically as:

$$H|0\rangle \Rightarrow \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix} \tag{11}$$

$$H|1\rangle \Rightarrow \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix} \tag{12}$$

This is very useful, as we can take any collapsed signal, and put it into a probabilistic state, which can then be used to perform other quantum computing operations. It is worth noting that while both of the above expressions give us the same probability of a $|0\rangle$ or a $|1\rangle$, they are not giving us the same output. In the $H|0\rangle$ case, the second index of the output is positive, but in the $H|1\rangle$ case, it is negative. Thus, the identity of the input is not completely wiped out from the system, and the operation satisfies reversibility.

Since we know that quantum operators are their own inverses, we realise that the Hadamard gate can also be used for the opposite effect — to take a qubit in superposition and collapse it back into a $|0\rangle$ or a $|1\rangle$ without measuring it.

This gives us a good general procedure to build any quantum algorithm: start with classical signals, and put them into superposition using the Hadamard gate. Once in superposition, run other quantum operations on them, and then transform the result of that computation back into a $|0\rangle$ or a $|1\rangle$ using a Hadamard gate. The results of a quantum algorithm using this structure will be deterministic, not probabilistic, thereby yielding the same results on every run.

Armed with just these three basic one-qubit operations (identity, negation, and the Hadamard gate), along with the two-qubit operation (CNOT gate), we can learn how to build some amazing algorithms that can outperform classical computers. Figure 3 shows some diagrams that I will be using to denote these operators when making circuit diagrams in the rest of this paper [11].

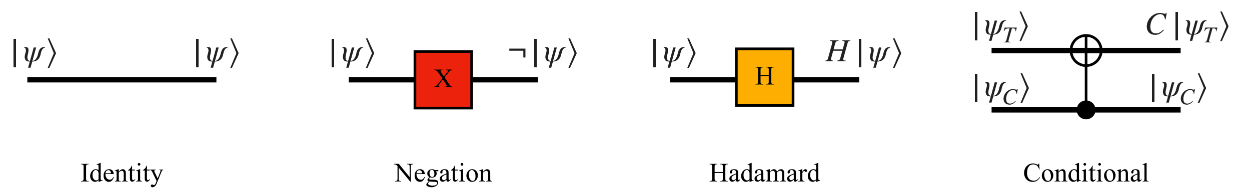

Figure 3: Circuit diagrams for some quantum computing operators. The right hand side qubits are the results of the operation. For example, $C|\Psi_T\rangle$ represents the result of the conditional gate on $|\Psi_T\rangle$. [11]

2.4 The Unit State Machine

Since a qubit is represented as a two-dimensional vector consisting of a pair of complex numbers, the set of all possible values it can take is a unit sphere [12]. This is called the Bloch sphere, and is named after the physicist Felix Bloch.

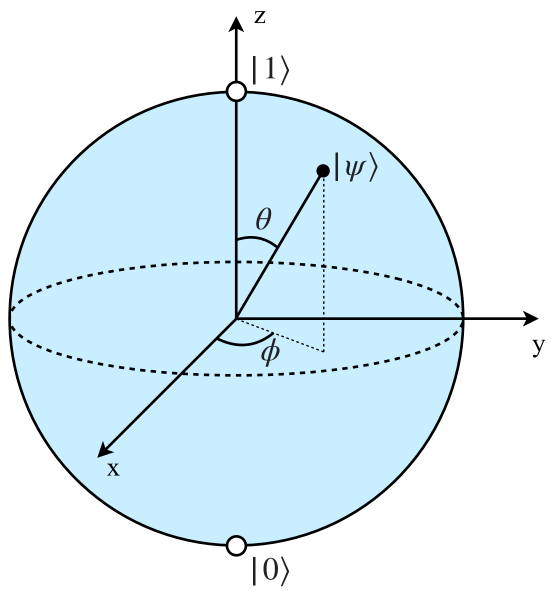

Figure 4: The Bloch sphere representation for all possible values of a qubit [12]. All points on the surface of the unit sphere are valid values for a qubit.

In this representation, the poles of the sphere are the exact values $|0\rangle$ and $|1\rangle$. At any other point on the sphere, the qubit is in superposition, with a probability of being both states. To find these probabilities, let's break the qubit vector down into two vectors in superposition: $|\psi\rangle = a|0\rangle + b|1\rangle$.

Looking at Figure 4, one simple solution from the spherical geometry jumps out immediately. If we assume a to be purely real and b to be complex, we can find:

$$a = \cos\frac{\theta}{2} \tag{13}$$

$$b = e^{i\phi}\sin\frac{\theta}{2} \tag{14}$$

Here we can see that our choice for any qubit $|\psi\rangle$ has two degrees of freedom, namely θ where $\theta \in [0, \pi]$, and ϕ such that $\phi \in [0, 2\pi)$. Those familiar with spherical coordinates may notice that we are missing the degree of freedom associated with the radial coordinate r. Recall that we applied the normalisation condition $|a|^2 + |b|^2 = 1$, which necessitates that $r = 1$, thereby eliminating it as a degree of freedom.

This is very useful in quantum computing, since modifying either θ or ϕ can change the value of the qubit itself. In Sections 2, 3 and 6 of this tutorial however, I will be assuming that all the qubits I am using have only real values. This will simplify things a lot, especially when working with diagrams, as the Bloch sphere simplifies into a two-dimensional "Bloch circle", with just a single degree of freedom Θ, where Θ represents the angular coordinate on the unit circle in Figure 5 such that $\Theta \in [0, 2\pi)$. We can now find the values of the probabilities a and b as:

$$a = \cos\Theta \tag{15}$$

$$b = \sin\Theta \tag{16}$$

Figure 5 shows this simplified representation along with some sample values. We can now also represent the one-qubit operations that discussed on this diagram. This diagram is called the unit state machine. We know that each input into the state operations only has one distinct output. Thus, if we are given a qubit with a known value, and a series of operations, we can use the unit state diagram to trace out all the transformations that the qubit undergoes, and find the resultant output qubit.

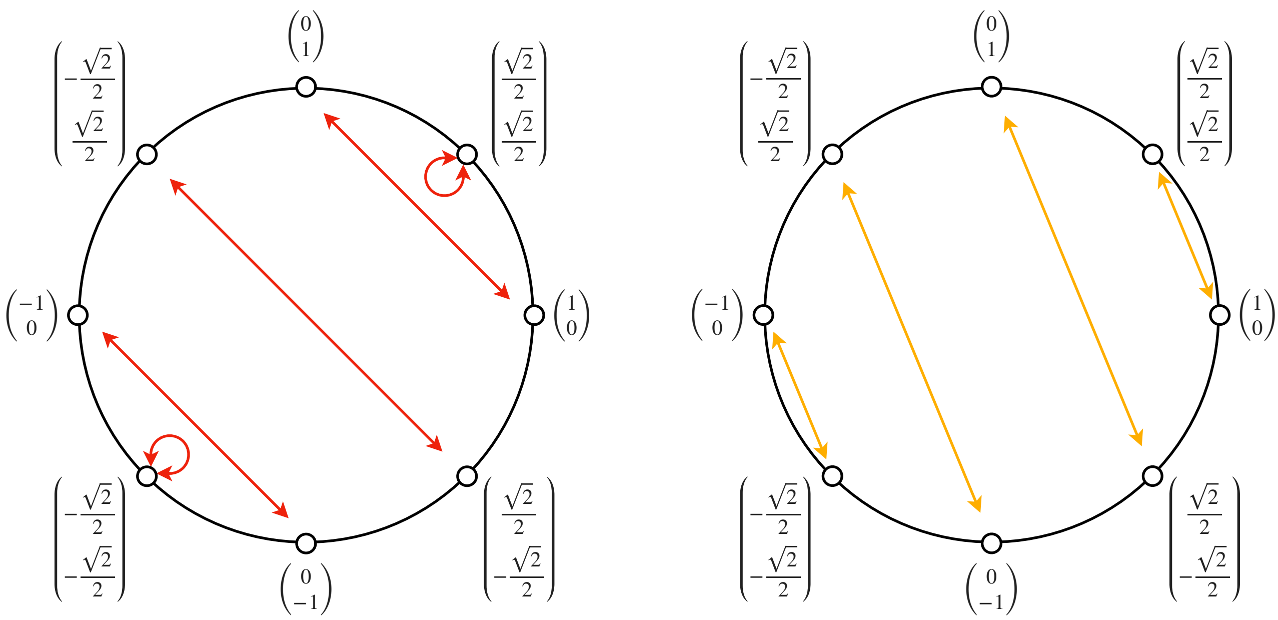

Figure 5: The unit state machine [27]. Red arrows show the result of the negation operation, and yellow arrows show the results of the Hadamard gate. As the identity operation on a qubit simply returns the same qubit, it is not shown here.

3. The Deutsch-Jozsa Algorithm

We now have all the tools necessary to understand quantum computing algorithms! For most day-to-day computational tasks, such as adding or multiplying numbers, quantum computers fare no better than classical computers, as they have to restrict the values of qubits to $|0\rangle$ and $|1\rangle$, effectively making them identical to classical bits. Quantum computers will only outperform classical computers if we give them a problem where they can exploit the full range of possible values of a qubit. The Deutsch oracle is the most basic problem on which a quantum computer outperforms a classical computer [13].

In this section, I will first detail the Deutsch oracle problem. Then, I will use the circuit notation and the state machine defined in Section 2 of this project, to demonstrate the implementation of a solution to the problem. Finally, I will briefly show a generalisation of solution to the Deutsch problem, which allows us to solve for multiple bits at the same time. This generalised solution is called the Deutsch-Jozsa Algorithm.

3.1 The Oracle

In the Deutsch oracle, we are given an unknown function, and we need to determine the operations it is performing. Let's call the function a "black box". Let us assume that this function has to be one of the four single-bit operations discussed in Section 2.1, but we don't know which one. We can send in a known input $|\psi\rangle$, and receive a known output $f\,|\psi\rangle$.

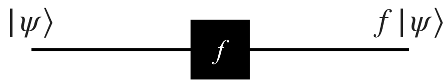

Figure 6: The circuit diagram for the Deutsch Oracle problem in a classical setup. [13]

In a classical system, we need two queries to find out what the black box function is. If we send in a $|0\rangle$ and a $|1\rangle$, and then compare the outputs, we can easily determine what the black box function is. This is the same in a quantum computer — it is not computing with all possibilities simultaneously, and then collapsing it to a single value when measured. Since it is not possible to represent four possible outcomes (as there are four possible operations in the box) on a single collapsed qubit, we still need two queries and two measurements to find out what the function is.

However, what if we want to determine what type of function is in the box? Again, the classical computer would require two queries. If the input is $|0\rangle$ and the output is $|0\rangle$, there is no way of knowing whether the box contains a Constant-0 operator or an identity operator. The only way to distinguish between the constant operation and the variable operation is to send in a $|1\rangle$ as an input, and then measure what the output of that query is.

On a quantum computer, we only need a single query to tell what kind of function is in the black box. To see why, let's first look at what each of the four single bit operators does to an input qubit.

The first problem we face is that the constant operations are not reversible, and so we cannot implement them in a quantum computer. We need to reframe the way in which we set up this problem, so that the black box takes in two qubits at the same time, instead of one.

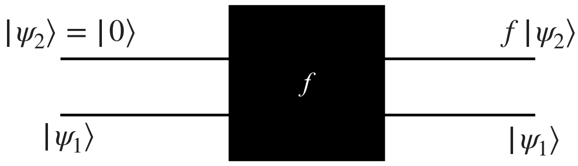

Figure 7: The quantum circuit diagram for the Deutsch Oracle problem in a quantum setup. [13][27]

Figure 7 shows this revised circuit diagram. Here, the value of $|\psi_1\rangle$ remains unchanged by the black box, and the effects of the black box function are written onto the $|\psi_2\rangle$ qubit. The $|\psi_2\rangle$ qubit is usually assumed to be set to $|0\rangle$ on the input side, and the output $f\,|\psi_2\rangle$ is the true output of the function, and the only one we actually need to worry about when working with these functions. This allows new set-up allows us to write non-reversible functions in a reversible way.

Let's now imagine what the inside of the black box would look like for each of the four operations. I will start with the Constant-0 operator:

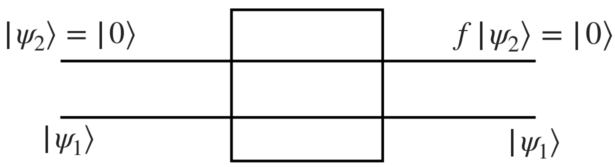

Figure 8: The quantum circuit diagram for the Constant-0 operator. [27]

This is very straightforward. Since we assume that $|\psi_2\rangle$ is set to $|0\rangle$, all we need to do is apply the identity operator on it, to return a constant $|0\rangle$ every time. A key difference between the quantum Constant-0 and the classical version, is that no information is being overwritten in the quantum circuit and the input $|\psi_1\rangle$ is returned unchanged. Measuring $|\psi_1\rangle$ would allow us to determine what the inputs were, and thus, the operation is reversible.

Constant-1 is also quite simple to set up:

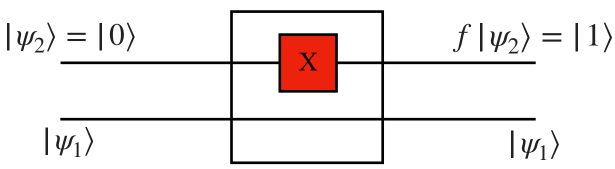

Figure 9: The quantum circuit diagram for the Constant-1 operator. [27]

All we need to do here is run $|\psi_2\rangle$ through a negation, and we will always get an output of $|1\rangle$. This effectively creates a reversible version of the Constant-1 operator.

Creating an identity function in the quantum set up is where things get more interesting. Here, we want the output of the function to be the same as the input $|\psi_1\rangle$. Thus, the operations that we need to run on $|\psi_2\rangle$ are themselves dependent on the input. Luckily, we've already seen a tool that can do something like this: the CNOT gate.

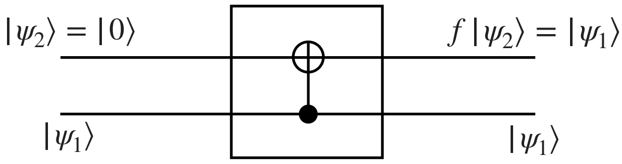

Figure 10: The quantum circuit diagram for the identity operator. [27]

If we follow the CNOT function's truth table from Figure 2, using $|\psi_1\rangle$ as the MSB and $|\psi_2\rangle$ as the LSB, we see that this perfectly returns the identity of $|\psi_1\rangle$ for any given inputs.

Lastly, we look at the negation operator. Here, we want the black box to return the opposite of $|\psi_1\rangle$ as the output. To create this, we can simply take circuit from the identity operation in Figure 10, and add a negation to the $|\psi_2\rangle$ qubit to ensure that it is always the opposite of $|\psi_1\rangle$.

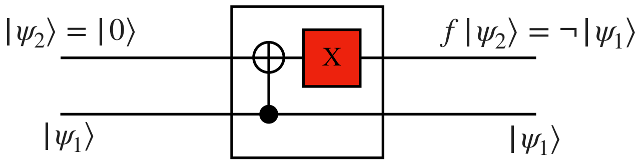

Figure 11: The quantum circuit diagram for the negation operator. [27]

Given what we know about qubit representations from Section 2, we can see that the black boxes in this section are performing operations which can each be shown mathematically as a 4x4 matrix, which is operating on a 4x1 vector that represents the product state of $|\psi_1\rangle$ and $|\psi_2\rangle$. Writing out these computations in matrix form looks quite cluttered, and hard to follow, so instead I will be showing the effects of the operations on qubits using the unit state machine.

3.2 Implementing the Deutsch Algorithm

Now that we have understood how the black box behaves given any function it can hold, we can create a circuit algorithm to solve the Deutsch oracle problem with a single query.

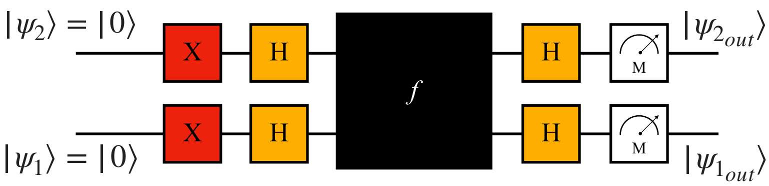

Figure 12: Quantum circuit for solving the Deutsch Oracle problem in single query. [27]

This solution is shown in Figure 12. Let's break down what is happening step by step. First, let's look at the preprocessing steps. Both $|\psi_1\rangle$ and $|\psi_2\rangle$ start off with values of $|0\rangle$. We put them both through a negation, effectively hard setting them to $|1\rangle$, and then send them through a Hadamard gate, which puts them into superposition. We can better see the prepossessing steps on a units state diagram:

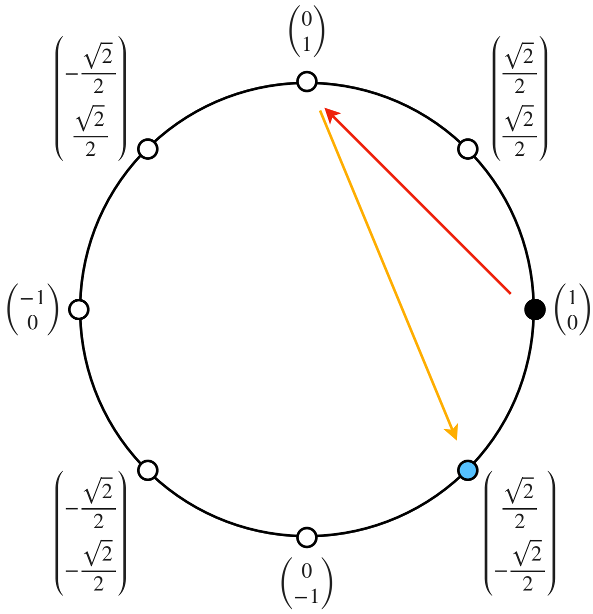

Figure 13: The preprocessing steps for the Deutsch algorithm. The black circle represents the initial value of the qubit and the blue circle represents the final value of the qubit. [13][27]

Now, both the qubits enter into the black box function, in a probabilistic state. The black box then operates on the qubits, and returns them out in a probabilistic state. On Figure 13, we also see a Hadamard gate applied to both the qubits once it exits the black box function. This gate is running post processing on the outputs of the black box, to transform the qubits with probabilistic values into qubits with deterministic values. This helps avoid errors due to collapsing the wave functions when we measure them.

We can now run these qubit values through all four of the quantum operator circuits described in Section 3.1, to see the effects that they have on the qubit. I show these along with the post processing step for brevity. The values at the points are the same as in other figures showing the state machine.

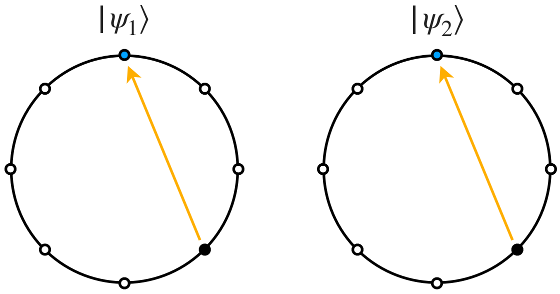

Figure 14: Result of the Deutsch Algorithm if Constant-0 is in the black box.

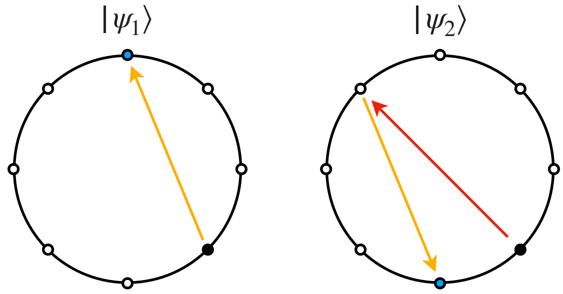

Figure 15: Result of the Deutsch Algorithm if Constant-1 is in the black box. [27]

From Figures 14 and 15, we see that the final states of the qubits when they are put through the black box and the post processing are not the same when they are put through Constant-0 and Constant-1. However, when we measure the qubits, the probabilities collapse to give us the same values, since the probabilities make use of the absolute value function. Thus, for both Constant-0 and Constant-1, the two qubit measurement of the system is $|11\rangle$.

We can now see what happens if the black box contains one of the variable functions. Again, let's make use of the unit state machine to visualise what is happening.

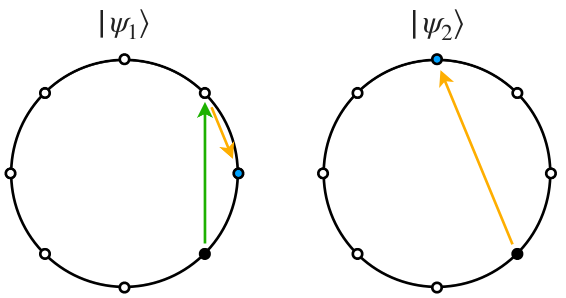

Figure 16: Result of the Deutsch Algorithm if identity is in the black box. [27]

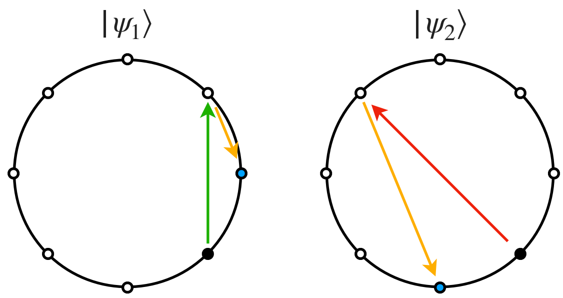

Figure 17: Result of the Deutsch Algorithm if negation is in the black box. [27]

Here, the green lines represent the transformation that results from the CNOT gate within both of the variable type operators. This is rather odd, didn't the CNOT gate leave the control bit untouched when we defined it back in Section 2.1? This was because we were still talking about classical bits when discussing CNOT in that section. Here, both the inputs to the CNOT gate are qubits in a superposition. This means that the CNOT gate will in fact have an effect on the control qubit.

This can be verified with some quick linear algebra. We know that CNOT is represented by:

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{17}$$

We also know that the inputs in the Deutsch algorithm are:

$$\begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix} \otimes \begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix} \Rightarrow \begin{pmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \tag{18}$$

We now multiply the CNOT operator with the product state of the inputs. This calculation clearly shows that the output product state of the CNOT is not the same as the input product state. The value of the control qubit has changed:

$$\left(\frac{1}{2}\right)\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\begin{pmatrix}1\\-1\\-1\\1\end{pmatrix} \Rightarrow \begin{pmatrix}\frac{\sqrt{2}}{2}\\\frac{\sqrt{2}}{2}\end{pmatrix} \otimes \begin{pmatrix}\frac{\sqrt{2}}{2}\\-\frac{\sqrt{2}}{2}\end{pmatrix} \tag{19}$$

This perfectly corresponds to the changes shown using the green lines on Figures 16 and 17. From these figures we can also see that the measured result of both the variable operations is $|01\rangle$.

This is a fantastic result: with a just a single query, we can now tell if the function within the black box is a constant function (returns $|01\rangle$) or a variable function (returns $|11\rangle$). As shown in Section 2.1, a classical computer would require two queries to make the same distinction. This problem seems rather convoluted, and doesn't have too many practical applications, but we can build on it significantly, to showcase real-world problems where quantum computers outperform classical ones.

The intuition behind what's happening here is that the differences in the operations due to the presence of negations within the black box are suppressed, while the differences in the operations due to the presence of the CNOT gate is amplified. This is because the effect of the negation is to put the qubit from one superposition state to another, but both of these states yield the same result once they are measured, due to the Hadamard gate used in post processing.

3.3 Generalising the Algorithm

We now expand the Deutsch Algorithm, to get its generalisation, which is called the Deutsch-Jozsa Algorithm. Here, we take the same basic setup as the Deutsch oracle, except now we have $n + 1$ bits as input, instead of just $1 + 1$ qubits. Using a quantum algorithm, the measured outputs either all map to $|0\rangle$, or they all map to $|1\rangle$. This is in the case of a constant function in the black box. If the function in the black box is a variable, the measured outputs can also map to exactly the same number of $|0\rangle$s and $|1\rangle$s. This is called a balanced result.

Let's first consider the classical case: here n qubits are unknown, and the final qubit is the known bit $|\psi_1\rangle$, which starts with a value of $|0\rangle$. Since $|\psi_1\rangle$ is negated, as per Figure 12, it is effectively a $|1\rangle$ in our setup.

This means that there are 2^n possible values. Let's try to solve this with a classical algorithm: suppose we take half the bits as an input and they all return $f(x) = 0$ as the output. Similar to section 3.1, assume that this function is a single-bit operation. We cannot know whether this was because the function is Constant-0 or because the function is negation and all the inputs are $|1\rangle$. We would need another input of a value $|0\rangle$ to make this distinction. So, in a worst case scenario, a classical computer would need $2^{n-1}+1$ queries to tell us what type of function was within the black box.

A quantum computer however, can solve this in just a single query [14]. Since we are now dealing with n qubits, working with matrices can be quite cumbersome. Instead, I will be showing this algorithm with the aid of sigma notation. Let's start off by rewriting the Hadamard gate. We know that it puts a qubit into a state where it has an equal probability of being a $|0\rangle$ or a $|1\rangle$, and already defined this as a 2x2 matrix operator back in Section 2:

$$H = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix} \Rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}} \langle 0| + \frac{|0\rangle - |1\rangle}{\sqrt{2}} \langle 1| \tag{20}$$

Now, since we have $n+1$ qubits, we need to run the Hadamard transformation on each of them. If we assume that the n qubits have a value of $|0\rangle$ (each of which goes through a negation, just like $|\psi_2\rangle$ in Figure 12, giving them values of $|1\rangle$) and the final $+1$ qubit already has a value of $|1\rangle$, we can write the product state of the bits after the Hadamard gates as:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle(|0\rangle - |1\rangle) \tag{21}$$

Where $|x\rangle$ represents that there are multiple possible states for the product, each of which needs to be accounted for in our black box function. This completes the same kind of preprocessing as we saw in Section 3.2.

We now put all of the different input qubits into the black box function. To see all the steps used to work this out, refer to the paper by Richard Cleve et. al. [14]. As per the paper, the last qubit can be ignored, so Equation 21 simplifies to:

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle \tag{22}$$

Here, for each of the possible states $|x\rangle$, the function $|f(x)\rangle$ has to return either a $|0\rangle$ or a $|1\rangle$, not a superposition of both. Finally, we put these outputs from the black box function back through Hadamard gates, in order to conduct post processing and make the entire computation deterministic instead of probabilistic.

$$\begin{gathered} \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \left[\frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} (-1)^{x \cdot y} |y\rangle \right] \\ \Rightarrow \frac{1}{2^n} \sum_{y=0}^{2^n-1} \left[\sum_{x=0}^{2^n-1} (-1)^{f(x)} (-1)^{x \cdot y} \right] |y\rangle \end{gathered} \tag{23}$$

where $x \cdot y = x_0 y_0 \oplus x_1 y_1 \oplus \cdots \oplus x_{n-1} y_{n-1}$. In this expression, $\oplus$ represents addition modulo 2. Therefore, $x \cdot y$ gives the sum of the scalar bitwise products, which is similar to using a Hadamard transform on each of the qubits individually.

Thus, Equation 23 is adding up all the different possibilities of the qubits put through the black box, and then summing them all up. In this expression $|y\rangle$ represents the final qubit vector that is measured after post processing. As a consequence of the above equation, the probability that $|y\rangle = |0\rangle$ can be found with:

$$\left| \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \right|^2 \tag{24}$$

The value of this probability would be 1 if the black box function $f(x)$ is a constant. Here, each of the terms in the summation makes the summation grow, as it adds some probability over and over again until the resulting probability is 1. We can also see that the probability of $|y\rangle = |0\rangle$ would be 0 if $f(x)$ is a variable function, as there would be an equal number of terms with a value of $|0\rangle$ and an equal number with a value of $|1\rangle$. Thus, the terms in the summation would cancel out.

A great way to visualise this is by imagining the interference patterns that would be created in the summation of all the states, as shown using an example where $n = 4$ in Figure 18. Here we clearly see that if $f(x)$ is a constant function, the output has maximum probability approaching 1, since all of the qubits are in phase with one another.

Alternatively, if $f(x)$ is a variable function, we see that the wave function for half the terms in the summation is out of phase with the other half of the terms. Since the output is balanced, the waves completely destructively interfere and thus the resultant probability is approaching the minimum value of 0.

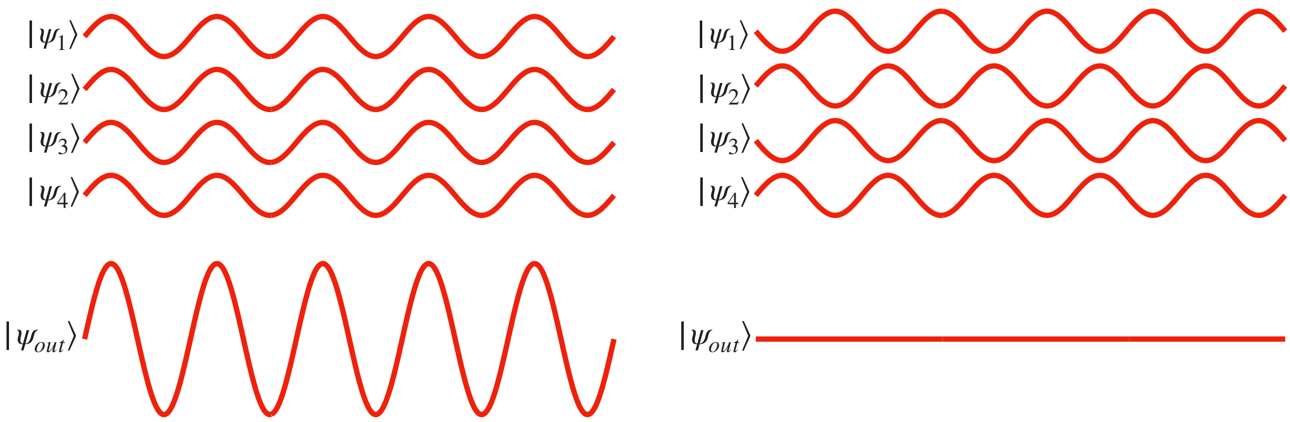

Figure 18: An interference based representation of the results of the Deutsch-Jozsa Algorithm. The left side shows a constant function in the black box, where the input waves are made to constructively interfere. The right side shows a variable function in the black box, where the input waves are destructively interfering with one another. [13]

Figure 18 also reveals something that I haven't discussed yet: the Hadamard gate is just a one-qubit version of a quantum Fourier transform. It selects and emphasises the wave functions with the frequencies that are the solution, and deemphasises all other wave functions by having them destructively interfere with each other. This way of thinking about the Hadamard gate has not been very relevant until we got to the Deutsch-Jozsa algorithm, but is something that we will have to keep in mind for the rest of this tutorial.

The Deutsch-Jozsa algorithm gives us a really neat solution, allowing us to find out what the type of any unknown function that operates on any number of bits is, using just a single query. This is an exponential improvement over classical computing. By itself, this algorithm doesn't have too many practical uses, but it serves as a great first step to demonstrate quantum computing supremacy.

4. Shor's Algorithm

Data on computers, smartphones, and internet accounts, are an integral part of life for billions of people in the modern age. Along with memes, cat videos, and movies, our data often holds sensitive and private information — such as healthcare records, personal conversations, and photos documenting our memories. Encryption is what keeps our private data private.

Many of the most common encryption methods, such as RSA encryption [15], make use of very large numbers (call this number N). To unlock an encrypted file, your device needs to provide the encryption's code with a list of all the prime factors of N. This acts as a "key" unlocking the encryption.

The only way for a third party to break the encryption is to find all the prime factors of N, by attempting to factor it. Since N is usually 1024 base 10 digits long, this can take thousands of years of computational time, effectively making the encryption unbreakable for classical computers. As you've probably guessed by now, quantum computers have no trouble quickly factoring N, and can effectively break most of our widespread encryption schemes with relative ease.

The method I demonstrate in this section is called Shor's algorithm [16]. It takes advantage of the superposition properties of qubits to factor large numbers very efficiently. I will first detail the math behind such a factorisation method. Next, I introduce some new quantum operations that have been alluded to previously in this paper, but not fully explored. Finally, we will build the quantum circuit diagram that can allow Shor's Algorithm to destroy encryption and internet piracy as we know it.

4.1 Factorising Large Numbers

The first part of Shor's algorithm relies on some clever number theory. Here, we reduce the problem of factorisation into a problem of period identification and order finding. This reduction can be run on a classical system, and allows us to break down the factorisation problem into a form that we can feed into the quantum computer.

In the first step, we simply pick a random number g, which satisfies the condition $1 < g < N$. Then, we need to find $\gcd(g, N)$, the greatest common divisor of the pair of numbers. To find the gcd, we can use the Euclidean algorithm.

The Euclidean algorithm tells us that to find a factor of N, we don't have to guess a factor of N; guessing numbers that share a factor with N yields a correct result too. Let's call this shared factor f_1. We can now divide N with this shared factor, giving us N/f_1, which itself would be a factor of N. We can repeat this process, and find f_2, f_3, et cetera, until all factors of the number are found.

For extremely large N however, this is very time consuming, and unlikely to deliver the results. Instead, we can use a trick to transform our initial guess g into a pair of guesses. Together, this pair of guesses is far more likely to be correct.

From Euler's theorem, we know that for any pair of whole numbers A and B that don't share a factor, if we multiply one of the numbers by itself enough times, we will get the second number multiplied by a constant plus 1. We can write this as:

$$\text{for any given } A, B \rightarrow A^p = mB + 1 \tag{25}$$

where m and p are both positive integer constants. Going back to our large number and our guess, we can write:

$$g^p = mN + 1 \tag{26}$$

We can now rearrange this equation to give us our pair of guesses that have a high chance of resulting in a correct answer:

$$g^{\frac{p}{2}} \pm 1 = mN \tag{27}$$

Since we have the constant m on the right hand side, we aren't actually finding factors of N, rather we are finding factors of some multiple of N. This is not a problem, as we can now simply make use of the Euclidean algorithm again, which would provide us with factors of N.

So far, everything is perfectly valid for a classical computer to solve. The process seems simple enough, but, looking closer, a few problems can be identified. Firstly, one of the $g^{\frac{p}{2}} \pm 1$ solutions could be a multiple of N, instead of a factor of N. This would make the other $g^{\frac{p}{2}} \pm 1$ solution a factor of m. Here we have gotten no correct answers for factors of N.

Secondly, the power p might be an odd number, which would mean that $p/2$ is not an integer. This would yield a non integer when plugged into $g^{\frac{p}{2}} \pm 1$, which is an impossible solution given that N and m are both integers.

If we simply discard all solutions where p is odd and where one of the solutions is a multiple of N, then we can safely repeat the process over and over again until all the factors are known. Thus, the first two problems slow down our process, but are easy to fix.

This leads us to our third, and most significant problem: finding the correct values of p. For a classical computer, this can take a lot of time and computational power, as the only way to find values of p is by brute force. For most cases, this will actually take longer than just trying to factor N using other methods, like the Euclidean algorithm that I initially proposed. If we want to break the encryption efficiently, we need a way to check for all values of p simultaneously, and then return only the correct ones. Of course, the best way to do this is by using a quantum computer.

4.2 The Quantum Fourier Transform

Before I elaborate on how we can find the correct values of p, we need to take another brief interlude into the topics of quantum logic gates and operators. Unfortunately, we can no longer restrict ourselves to the simplified "Bloch circle" and will have to make use of the full Bloch sphere for the rest of Section 4 and Section 5 of this tutorial.

A quantum Fourier transform (QFT) is a linear transformation on qubits, that acts on a set of quantum states and maps it onto a different state [17]. For an initial state consisting of n qubits, there are $2^n = S$ possible states. We can apply the QFT on it using:

$$y_k = \frac{1}{\sqrt{S}} \sum_{n=0}^{S-1} \psi_n \omega_S^{nk}, \quad k = 0, 1, 2, \ldots, S-1 \tag{28}$$

Here ω_S^n is a rotation representing a root of unity, giving us $\omega_S = e^{\frac{2\pi i}{S}}$. The ψ_n represents each row of the input vector ψ, and y_k represents each row of the output qubit vector y. We can also write the QFT in matrix form:

$$F_S = \frac{1}{\sqrt{S}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{S-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(S-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(S-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{S-1} & \omega^{2(S-1)} & \omega^{3(S-1)} & \cdots & \omega^{(S-1)(S-1)} \end{bmatrix} \tag{29}$$

Just like the other quantum computing operators we've seen so far, the QFT operator is reversible. The inverse QFT is actually what we will use when laying out Shor's algorithm in Section 4.3. By now, we are already familiar with the Hadamard gate. If we compare the matrix for the Hadamard gate, as presented in Section 2, to the above matrix for the QFT, we see that this gate is actually an example of a QFT being performed when the number of bits $n = 1$, giving $2^n = 2$ possible states. The Hadamard gate is also incredibly useful for the construction of a QFT operator in a quantum mechanical circuit.

Another component that is used in the construction of a QFT operator is called the phase shift gate [5]. This is a single qubit operator, which is used to modify the phase of the qubit. The probability of measuring a $|0\rangle$ or a $|1\rangle$ in that qubit is left completely unchanged. We can imagine this on the Bloch sphere as the cubit $|\psi\rangle$ being moved along a line of longitude, by an azimuthal angle ϕ. The polar angle θ is left unchanged. Referring to Figure 4 may help in visualising this transformation. We can represent it in matrix form as:

$$R_m = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^m}} \end{pmatrix} \tag{30}$$

where $e^{\frac{2\pi i}{2^m}} = \omega_{(2^m)}$, which is the 2^mth root of unity. The subscript m on the phase shift is also indicative of which of the roots is being used. The controlled phase shift gate is the specific version that we need to build the QFT. To get this, we simply multiply the CNOT operator on R_m. This results in a two bit operator, where the control qubit determines whether or not the target qubit undergoes the phase shift.

Using the Hadamard gate, and the controlled phase shift gate, we can now build a circuit that performs a QFT on a set of n qubits.

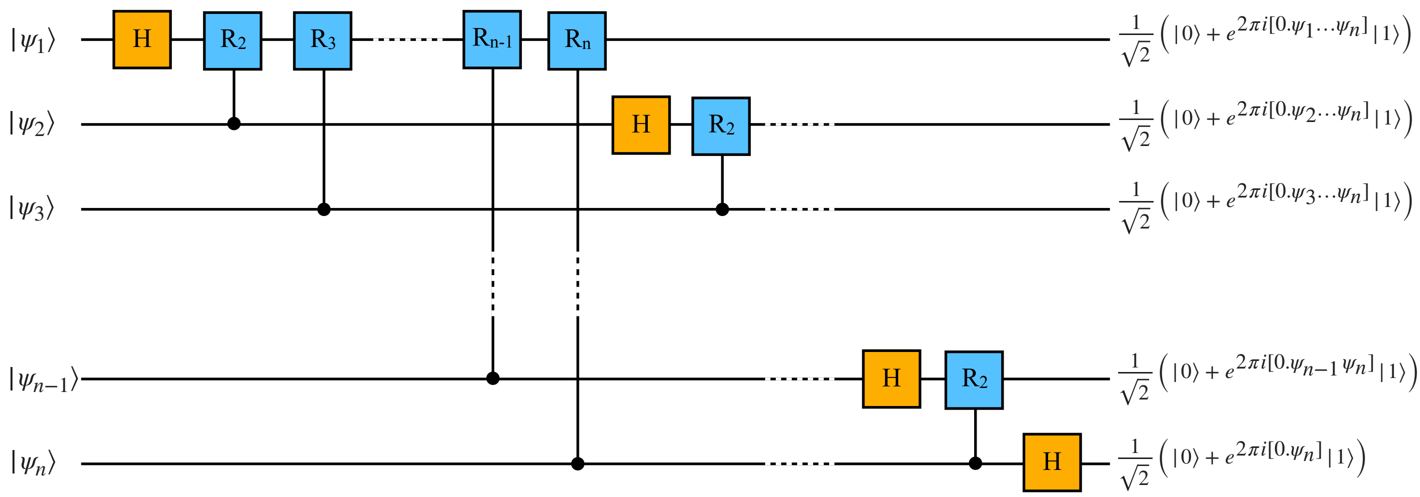

Figure 19: A quantum circuit diagram for building a quantum Fourier transform, being performed on a set of n qubits. [5]

Here I have used binary notation to simplify what is given on the output side. Binary notation represents a summation of the form:

$$[0.\psi_1 \ldots \psi_m] = \sum_{k=1}^{m} \psi_k 2^{-k} \tag{31}$$

Detailing all the steps that go on within this circuit diagram would become very complicated, and is out of the scope of this tutorial. Instead of further explaining the mathematics of the QFT, I will provide a qualitative understanding of it: the QFT selects and emphasises the wave functions with the frequencies that are in the solution to our problem, and deemphasises all other wave functions by having them destructively interfere with each other. This is very similar to what was shown in Section 3.3, and Figure 18.

Another way to understand the workings of the QFT without going through the math is to use an example to see its operation. Here, I take $n = 3$ and set all the inputs to $|0\rangle$.

Let's start by plugging $n = 3$ into the QFT summation equation:

$$|\psi\rangle \mapsto \frac{1}{\sqrt{2^3}} \sum_{k=0}^{2^3-1} \omega_{2^3}^{\psi k} |k\rangle \tag{32}$$

Since $n = 3$, we know the product state will be a vector of length 8x1, and the QFT will be represented by an 8x8 matrix. This is tedious to write out, but can help to visualise the procedure:

$$F_{2^3} = \frac{1}{\sqrt{2^3}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega^8 & \omega^{10} & \omega^{12} & \omega^{14} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \omega^{12} & \omega^{15} & \omega^{18} & \omega^{21} \\ 1 & \omega^4 & \omega^8 & \omega^{12} & \omega^{16} & \omega^{20} & \omega^{24} & \omega^{28} \\ 1 & \omega^5 & \omega^{10} & \omega^{15} & \omega^{20} & \omega^{25} & \omega^{30} & \omega^{35} \\ 1 & \omega^6 & \omega^{12} & \omega^{18} & \omega^{24} & \omega^{30} & \omega^{36} & \omega^{42} \\ 1 & \omega^7 & \omega^{14} & \omega^{21} & \omega^{28} & \omega^{35} & \omega^{42} & \omega^{49} \end{bmatrix}$$

Now, we apply this transformation to the initial product state vector, to get a resultant product state vector. We can then break this back down to the individual state representation, to see how the QFT affected each of the three input qubits:

$$\frac{1}{\sqrt{2^3}} \left(|0\rangle + \omega_{2^3}^{[\psi_1\psi_2\psi_3]}|1\rangle\right) \otimes \left(|0\rangle + \omega_{2^2}^{[\psi_2\psi_3]}|1\rangle\right) \otimes \left(|0\rangle + \omega_{2^1}^{[x_3]}|1\rangle\right) \tag{33}$$

This individual state notation matches what we would expect as the output if we compare it to the outputs given in Figure 19.

4.3 Cracking Modern Encryption

We can now get back to the problem of breaking the RSA encryption scheme by factoring the very large number *N*. Where we last left off in this problem, we had a set of potential solutions in the form of $g^{\frac{p}{2}} \pm 1 = mN$ (where m is simply a positive integer, and thus *mN* is some multiple of *N*), but did not have a way to find the correct values of *p* to use in this equation.

While a classical computer would have to check for each value of *p* separately, we can set up our quantum circuit so that it can check for all values of *p* in a single query. We need it have the input to be a superposition of all possible *p* values. As we've seen before, putting the inputs into an exact superposition state (giving all outcomes equal probability) can be done by applying the Hadamard gate to each of them. Let's call this input $|\psi\rangle$.

We then take our guess g, and raise it to the power of ψ. We then check if $g^{\psi} > N$. If this is true, we use the modulus operation to find r, the remainder: $g^{\psi} \pmod{N} \Rightarrow g^{\psi} - mN \Rightarrow r$. Figure 20 illustrates this process. Here I have represented the operations we are carrying out on the qubits as black box functions, instead of showing their full inner workings, to reduce complexity.

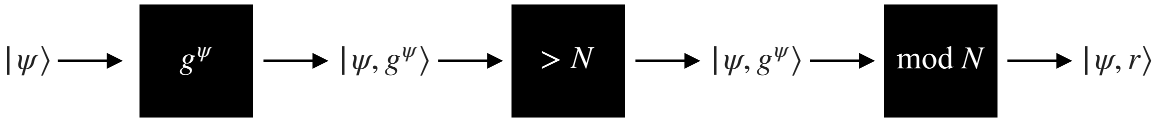

Figure 20: A simplified black box diagram for a circuit that finds the remainder r to help identify the solutions to the power p. [16]

Since $|\psi\rangle$ is a superposition testing out many values, the resulting $|\psi, r\rangle$ is also a superposition, containing all possible values of r. We now need to search through all of these answers in superposition to find the one where $|\psi, r\rangle = |p, 1\rangle$.

The next detail we need to consider is that if:

$$g^{p} = mN + 1 \ \text{ and } \ g^{\psi} = m_1 N + r \tag{34}$$

then we can rewrite this as:

$$g^{\psi + qp} = m_2 N + r \tag{35}$$

where m_1, and m_2 are also positive integers. q is an integer that can be positive or negative. This equation means that p has a repeating property. Regardless of what the value of ψ is in the above equation, if we add or subtract a multiple q of p to it in the exponent of g, we will get the same remainder r. The only thing that changes as we change the term in the exponent, is the multiple m_2.

To summarise, the repeating property of p means that all the values of $\psi + qp$ result in the same remainder r. If p and ψ are taken as constants, and we check for various values of q, all solutions will have a difference of p from one another.

Thus, the final output in Figure 20 provides us with a superposition of solutions that repeat periodically with a period of p. This means that the frequency of the wave function that makes up the output qubit would simply be:

$$f = \frac{1}{p} \tag{36}$$

If we can find the frequency, we can reciprocate it to find the value of p. We've already discussed the tool we can use to check all values of the frequency to give us the appropriate f for our problem: the Quantum Fourier Transform. If we follow the steps laid out in section 4.2, and then reciprocate the results, we get answers for p. We can now plug those into the expression $g^{\frac{p}{2}} \pm 1$ that we found in Section 4.1, and finally factorise N and break the encryption.

There are some bottlenecks in this algorithm [18], such as in checking if $g^{\psi} > mN$, and then returning the modulus to find the remainder r. However, even with these hurdles, it would only take a few minutes for the quantum computer to factor a 1024 digit RSA encryption key. Internet privacy as we know it has been destroyed.

5. Sorting Algorithms

Sorting an unstructured list has countless applications in computer science. From something as simple as putting folders in alphabetical order, to something as complex as building a search engine, sorting through data is critical. Unfortunately, it can also be incredibly time consuming for classical computers.

In the previous section, we saw that quantum computers can drastically improve compute times in some scenarios. However, this is not always true, and scientists are still trying to come up with an algorithm that takes advantage of the properties of quantum computers to drastically speed up sorting times [19]. To demonstrate this improvement, I will first highlight some sorting methods in classical computing, and compare how efficient they would be on large random unstructured lists of data, using Big-O notation. I will then describe one potential solution for a quantum search algorithm.

5.1 Classical Sorting and Big-O

Big-O notation is a simple way of representing how many computations an algorithm needs in order to sort a list containing n items, when n is an arbitrary large integer. For example, if an algorithm takes n^2 computations to sort out a list of n items, we write it as having $O(n^2)$. When woking with Big-O notation, we usually consider the worst case scenario for an algorithm, to define an upper bound for the number of required computations.

We usually simplify the function that describes the behaviour of an algorithm to the term with the highest growth rate, since as n approaches infinity, this term tends to dominate the efficiency of the sorting algorithm. If the most dominant term is multiplied by a constant, we also omit the constant for simplicity. For example, if an algorithm takes $4n^2 + 3n + 6$ computations to sort out a list of n items, we would once again denote it as having $O(n^2)$.

Perhaps the "simplest" way to sort something it to just randomly shuffle it with the hope that the result is a sorted list. This sorting algorithm is called Bogo sort [20]. In order to actually sort out the list, it would need to try out every single possible shuffling of the list, until it gets it right. For a list with n items, this would require $n!$ attempts, making it incredibly inefficient.

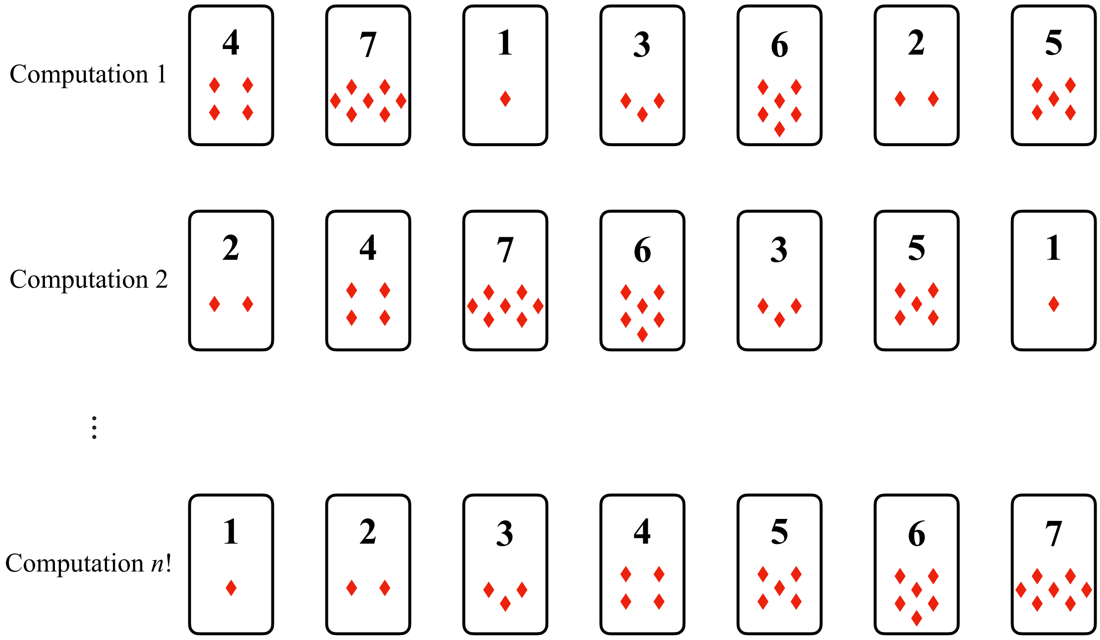

Figure 21: Here, cards are used to represent items in the list. Bogo sort keeps randomly shuffling the cards until they are sorted. [20]

Bogo sort relies on chance to arrive at the right answer. For the simple example in Figure 21, with seven cards, it would take anything between 1 query and ∞ queries to solve the sorting problem. We might get lucky and shuffle to a sorted list on our first computation, or we might keep shuffling until every possible order has been tried. Assuming we do not repeat any order of the cards that we have already tried, the random shuffling sorts the list in $n!$ computations, so the Bogo sort algorithm has an efficiency of $O(n!)$. This method is obviously not very practical for any real world applications, but it serves as a good starting point to think about sorting speeds.

If we apply some simple math, we can greatly improve the sorting time. Let's now look at Bubble sort [21]. Here, we take two neighbouring items with indices i and $i+1$ and then compare them. If $i > i+1$, swap the two elements, else leave them unchanged. Repeat this over and over again until there are no two items for which the inequality $i > i+1$ holds true. At this point, the list is sorted.

When implementing Bubble sort, we would have to effectively check each element on the list multiple times to put it in the correct place. If each element is out of order, Bubble sort needs n^2 computations to sort out the list in a worst case scenario.

Just like in Bogo sort, we could get lucky and start off with a sorted list, which would mean we'd only need to check each item once to see if it satisfies $i > i+1$. Here, we get a best case scenario where the algorithm needs just n computations to solve the list.

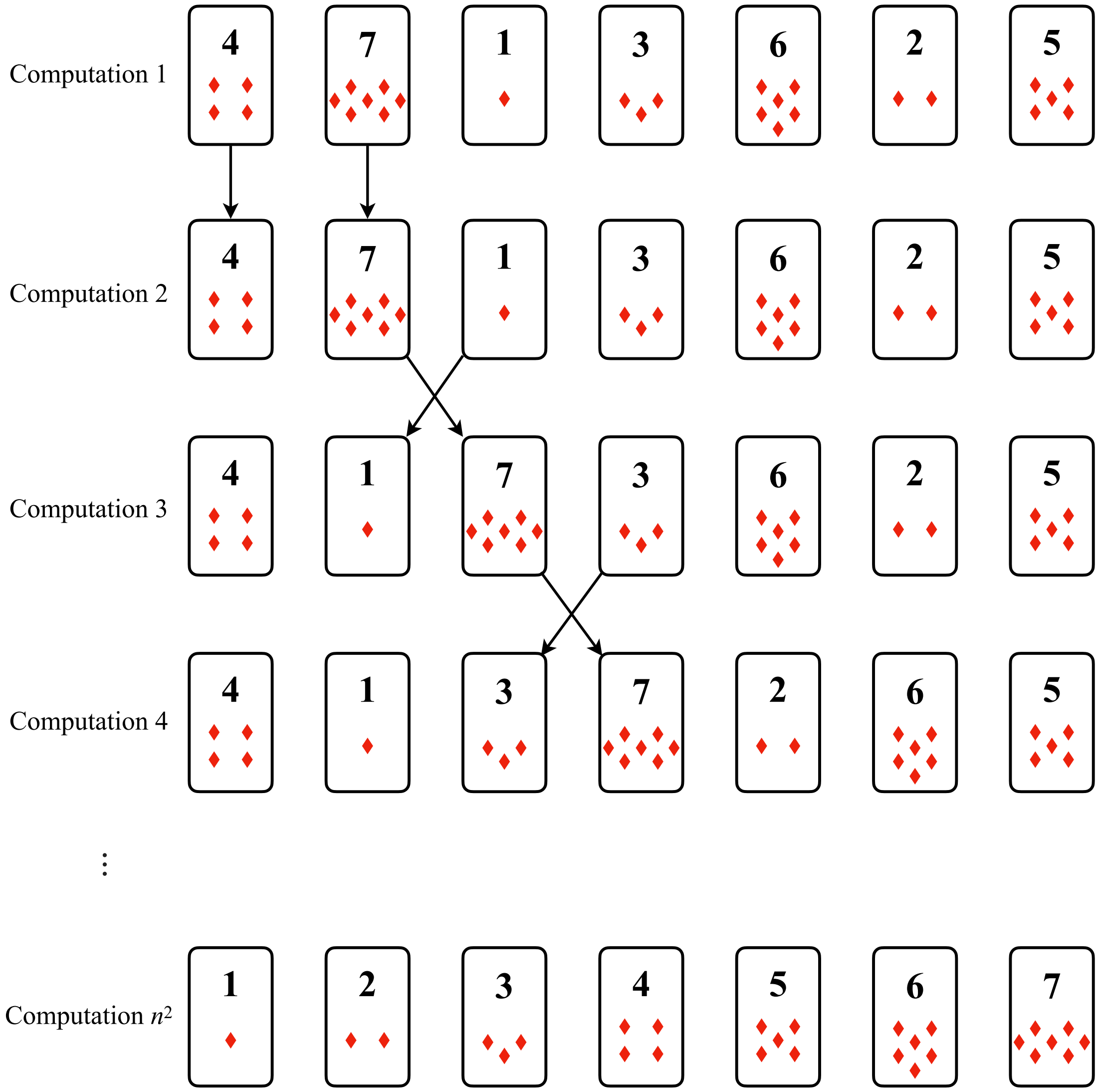

Figure 22: A representation of Bubble sort, attempting to sort cards in ascending order. [21]

Figure 22 provides us with a more intuitive understanding of what is happening here. In computation 1, we see that 4 and 7 are already in the correct order, and so they are left alone. Next, 7 and 1 are compared. Since 7 is greater than 1, they are swapped. This process is repeated over and over again until the entire list is sorted.

In bubble sort, each element is checked multiple times, making it inefficient when n becomes big. One way to speed up the sorting process is to split the list into smaller pieces, and then tackle them separately. One popular method to implement something like this is called Merge sort [22]. This needs just $n\log(n)$ computations when sorting through a list with length n. This is considerably faster than Bubble sort on larger lists, and is pushing the boundaries of what is possible on a classical computer.

Merge sort works by first splitting the list in half over and over again, until each element in the list is placed in its own separate list. Each pair of adjacent lists is then compared and “merged” to form a bigger sorted list. Again, this process repeats, until all the elements are back together in a single sorted out list.

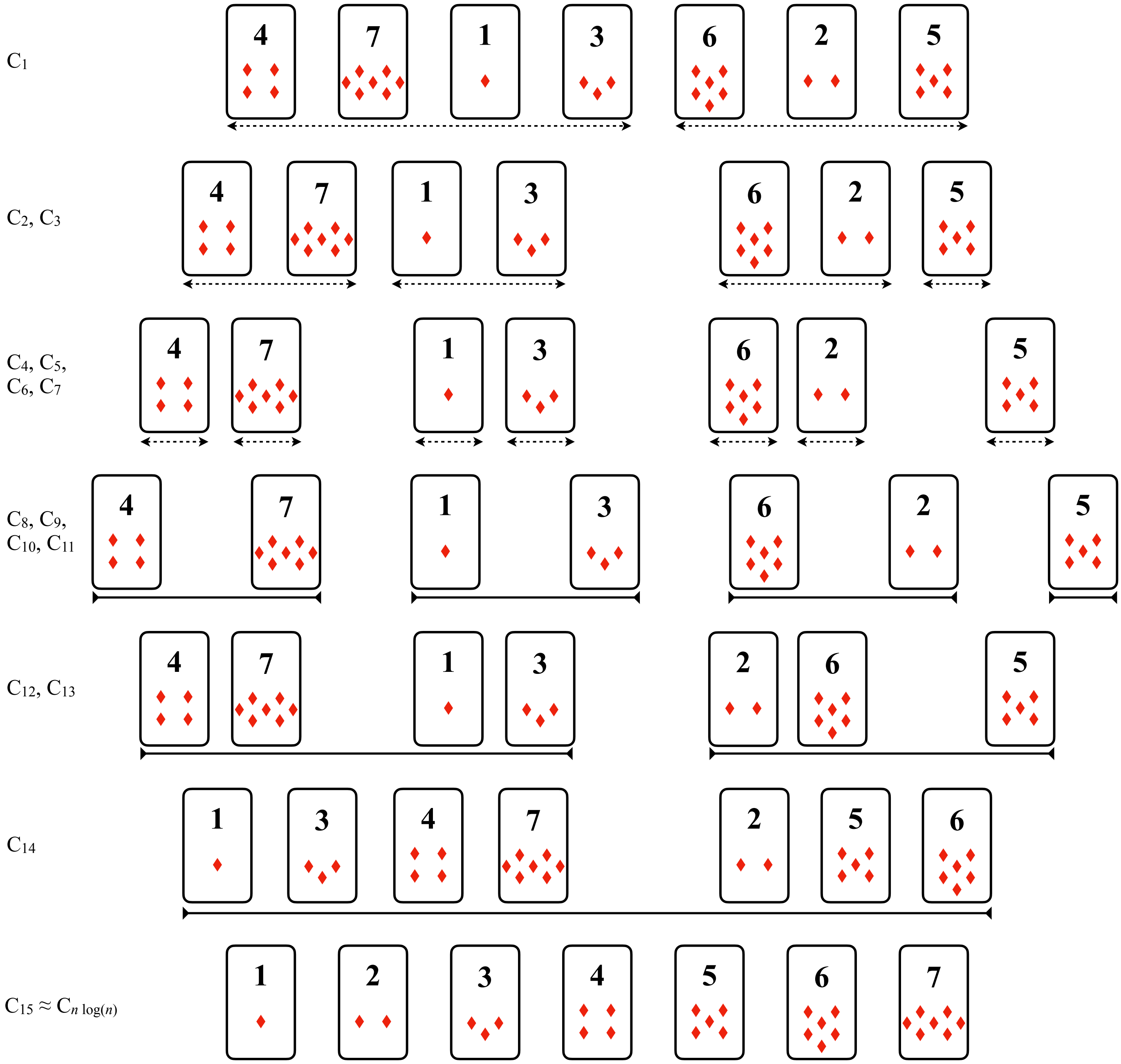

Figure 23: A representation of merge sort, attempting to put cards in ascending order. C_n represents the nth computation. [22]

Figure 23 illustrates how merge sort goes through the list. This method requires so many fewer computations than Bogo sort and Bubble sort that the entire process could be depicted on a single page. It is one of the fastest algorithms for sorting in classical computing, with an efficiency of $O(n\ \log(n))$.

Understanding the mechanics behind these classical algorithms can help build some intuition when discussing quantum algorithms. Additionally, they serve as great points for comparison when trying to understand how much of a difference in efficiency a quantum algorithm could make.

5.2 Quantum Sorting

The creation of a quantum sorting algorithm is still being researched. Some known quantum computing algorithms can outperform classical computers when it comes to the amount of memory space needed to implement the algorithm [23], but they still take $O(n\log(n))$ computations to sort the list, just like merge sort. Since quantum computers are incredibly expensive compared to classical ones, it is impractical to use them in these scenarios.

However, progress is being made, and perhaps one day a quantum algorithm can be found that massively outperforms classical computers. A 2013 paper by Odeh et. al. [28] proposed one such algorithm, which managed to attain a time efficiency of:

$$O\left((n-2)\log(n-2)\right) \tag{37}$$

While this is certainly an improvement, as n gets large Equation 37 simply approaches $O(n\log(n))$, which is the same time efficiency as merge sort. This is not a very exciting result.

Improvements are already underway, and in 2016 an updated sorting algorithm was proposed by Odeh and Addelfattah [24], with which we could attain a sorting efficiency of:

$$O\left(\frac{n-2}{2}\log\left(\frac{n-2}{2}\right)\right) \tag{38}$$

This is a significant improvement over classical sorting. As as n gets large, the sorting time of the proposed algorithm is less than half the sorting time of merge sort. The full details of the algorithm are not detailed in this paper for the sake of brevity, however I highly recommend taking a look at the 2016 paper [24] as it contains a several flowcharts and diagrams that aid ones understanding of the sorting algorithm.

To actually implement either of the proposed algorithms in a quantum circuit, we'd have to make use of quantum entanglement (see Section 6), and another new type of gate called a Toffoli gate [25].

The Toffoli gate is effectively a three-qubit version of the CNOT gate that was discussed in Section 2.1 (which is why it is also sometimes called the CCNOT gate). Here, there is still one target qubit, but its state is influenced by two independent control qubits. The Toffoli gate is a universal gate, which means that we can create any other Boolean quantum operator using various combinations of Toffoli gates.

I'm not going to detail the circuit construction for this sorting algorithm, as it would take up several pages without adding much value to the tutorial. Instead, I will just show the efficiency of this algorithm compared to the classical algorithms discussed in Section 5.1, on Figure 24:

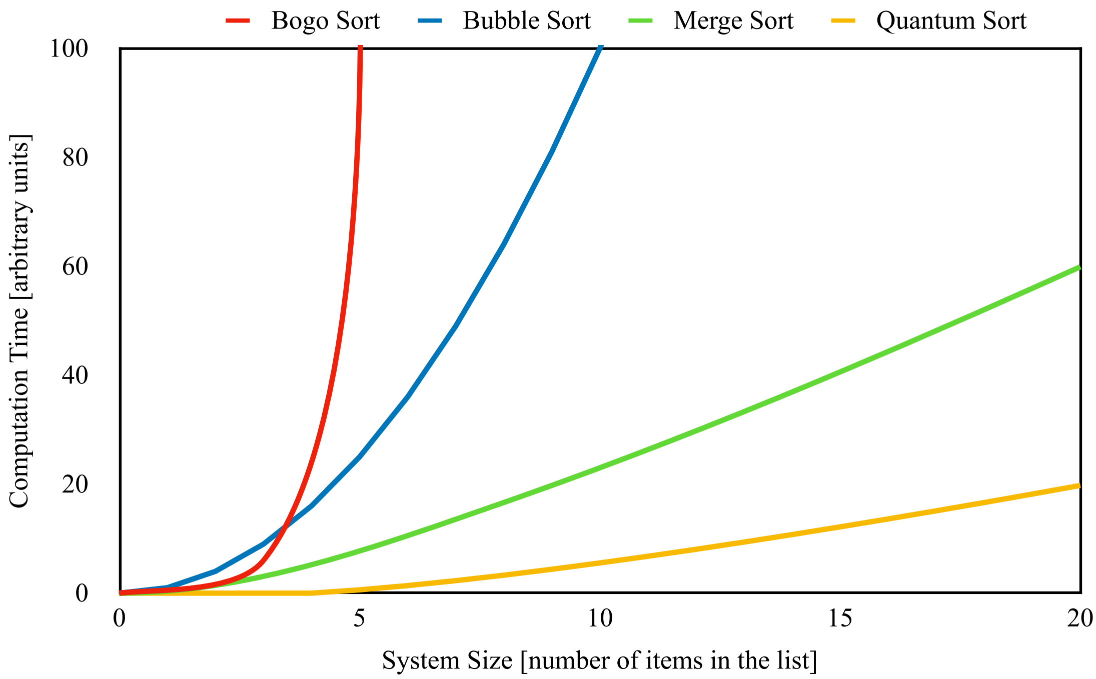

Figure 24: Comparing the classical sorting algorithms to quantum sorting for small *n*. [24]

In Figure 24 we see that the quantum sorting algorithm proposed by Odeh and Addelfattah [24] is clearly a lot faster than conventional algorithms, but I do not believe that this will be enough to warrant its adoption in the near future. Building a quantum computer capable of holding enough bits to execute this algorithm will be extremely costly and difficult.

Unlike with Shor's algorithm, which provides an exponential increase in speed over classical computers, the existing quantum sorting algorithms don't seem to do much better than classical ones. Perhaps, someone will discover a better way to sort lists using quantum computers in the future, but until then, we will have to be satisfied with algorithms like merge sort.

6. Entanglement and Teleportation

So far, we've looked at algorithms and operations in Quantum computers that make use of the fact that the qubit exists in a superposition instead of a single value. This has amazing implications, and we've seen it applied in many different ways to solve all sorts of problems.

Superposition isn't the only difference between a classical bit and a qubit. Qubits can also exist in an entangled state, where the values of multiple qubits cannot be separated from one another. Quantum mechanics also allows for teleportation of a qubit. These are both very strange properties, and Einstein even famously described them as "spooky action at a distance".

These properties of qubits may allow us to accomplish some things with quantum computers that we couldn't even dream of with classical computers. In this section, I will briefly touch upon both entanglement and teleportation, and how we can make use of them in quantum circuitry.

Let's begin by defining what it means for two qubits to be entangles mathematically: if the product state of the two qubits cannot be factored out into individual states, they are said to be entangled. Take the example:

$$\begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix} \Rightarrow \begin{pmatrix} \frac{\sqrt{2}}{2} \\ 0 \\ 0 \\ \frac{\sqrt{2}}{2} \end{pmatrix} \tag{39}$$

If we try to solve for a, b, c, d, we find that the system has no solutions. Thus, there is a 50% chance that it will collapse to $|00\rangle$ when measured, and a 50% chance that it will collapse to $|11\rangle$ when measured.

We can put two qubits into entanglement with the following circuit:

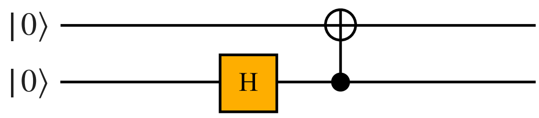

Figure 25: A quantum circuit that is entangling two qubits together. [27]

Here, we are putting the MSB into a superposition state using the Hadamard gate, and then using it to control the LSB (least significant bit — see Section 2.1) in a CNOT gate. The resultant qubits from this circuit are in the form of the above example product state.

The implications of being able to entangle two qubits are massive. If you measure one qubit, the other collapses into the exact same value. This happens across any distance, and it seems to happen instantaneously. The phenomena is referred to as quantum teleportation.

Quantum teleportation is a process wherein the state of one qubit is transferred onto another qubit — which is potentially extremely far away — instantly. This can be achieved by transferring the state of an input qubit onto a qubit in a pair of entangled qubits. This would instantly paste the state of the input qubit onto the second entangled qubit as well.

In 2018, the Chinese Space Agency demonstrated this by putting one entangled qubit on a satellite, and the second entangled qubit in the pair was kept on Earth. When one qubit was measured, the second collapsed to the same state instantly [26].

This is rather curious: the particles are seemingly communicating faster than the speed of light. However, on further inspection, we realise that even though the particles are coordinating their state faster than light, communication is not taking place, and no information is being transmitted. Even though the entangled qubit is receiving the state of the first qubit, we still are not breaking the speed limit of light, since we need to send two classical bits of information to help the final qubit collapse into the correct state.

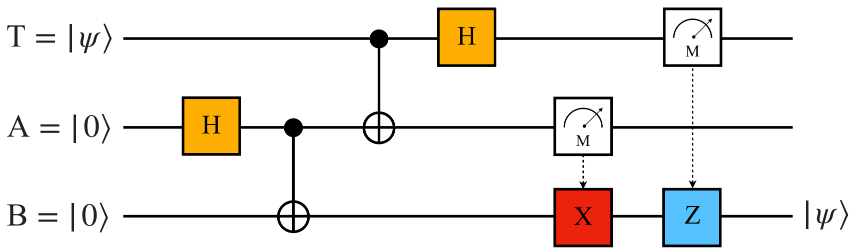

Figure 26: A quantum circuit that is teleporting the state of T to B. [27]

In Figure 26, T is the qubit we want to teleport, and A and B are initialised to $|0\rangle$. The first step in this circuit is to entangle A and B with one another. This is done using the same Hadamard gate, CNOT gate combination seen in Figure 25. Next, T and A are entangled with one another. This effectively gives us a three-qubit entangled system. We can verify this mathematically; if we apply the CNOT and Hadamard operations on all three input qubits, we see that the product state of the three qubits cannot be factored back into the individual states.

Moving on, T is put through a Hadamard gate, making it deterministic instead of probabilistic when it is measured. Thus, measuring both A and T results in two classical bits. These classical bits are then used to control operations on B. If A is measured as $|1\rangle$, a negation operation is performed on B, and if T is measured as $|1\rangle$, a Pauli-Z operation is performed on B.

The Pauli-Z shift gate is just a special case of the phase shift gate we saw when describing the QFT in Section 4.2. We can write out the gate as:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{40}$$

Referring to the Bloch sphere in Figure 4, the Pauli-Z shift represents a 90 degree rotation along the axis of the sphere. Doing the math, we see that if the negation and the Pauli-Z are applied as per the measured results of A and T, then B displays the exact same value as $|\psi\rangle$ regardless of the distance.

7. Conclusion

This brings us to the end of our discussion of quantum computing algorithms. We've covered a lot of ground in this tutorial, so let's briefly summarise everything we've spoken about.

We started by looking at how classical bits and qubits can be represented mathematically, using vectors. We then saw how basic one bit operations — identity, negation, Constant-0, and Constant-1 — can be represented as matrix operations and how they transform bit vectors from one value to another. We carried on to see how multiple bits can be represented, and learned about the Hadamard gate and the CNOT gate, which are both integral to any quantum circuitry.

Next, we saw how all the possible values of a qubit could be represented by a unit sphere, called a Bloch sphere. If we made the assumption that the qubits we are working with were purely real, the Bloch sphere simplified into the unit state machine, which was a good visual representation to see how the Hadamard gate and the CNOT gate changed the values of qubits.

We used the unit state machine to discover how operations could be built using circuitry in a quantum computer, and then solved our first real problem: how to tell what type of function is inside a black box. This was solved using the Deutsch algorithm. The Deutsch algorithm could be generalised so as to work on multiple bits instead of a single bit, and this was called the Deutsch-Jozsa Algorithm.

When solving for the Deutsch-Jozsa Algorithm, we discovered a neat trick that quantum computers can perform: they can superpose multiple wave functions of multiple qubits within them, and can check to see if these wave functions interfere constructively, or destructively. This ability allows quantum computers to check multiple solutions to a problem at the same time, yielding an exponential speed-up over a classical computer.

The incredible power of quantum computers can be used to break modern encryption methods, such as the RSA encryption. This is done by factoring out an incredibly large number, and then using the list of factors as a key to break the encryption. We used some number theory to simplify the problem into something where we simplify needed to find the periodicity of a possible solution. Then, we applied a quantum Fourier transform to help identify the key to breaking the RSA encryption scheme.

We then explored some sorting algorithms. Unlike in the cases of checking a black box for its function type, or factorising large numbers, quantum computers do not result in an exponential speed-up here. While the quantum algorithm I described was far more efficient than Bogo sort, Bubble sort, and Merge sort (and in fact faster than any classical algorithm), the increase in computing speed was not large enough to justify the massive costs of building a quantum computer.

Finally, we looked at quantum teleportation and quantum tunnelling. Both these concepts are incredibly exciting, and they seemingly allow qubits to communicate faster than the speed of light. However, on closer inspection we saw that they were some caveats to this, and even though the entanglement results in instantaneous coordination between qubits, no information is actually communicated until two classical bits are sent over to the teleported qubit. Thus, the speed limit of light is not broken.

Overall, I think the field of quantum computing could potentially to change the way we build process large data sets, run simulations, and build encryptions. Current quantum computers are still very technically challenging to build and maintain, not to mention prohibitively expensive for anyone other than select organisations and governments. However, I do believe that over the course of the next few years we will see advances in the field. Google and IBM both have functioning quantum computers, which have demonstrated quantum supremacy over classical computers, and both these tech giants are continuing in their research efforts. Perhaps, one day in the near future, we will see some of the fascinating algorithms described in this tutorial being applied to solve real world problems.

8. Acknowledgements

This tutorial is based on a project for an introductory quantum mechanics class at Brown University, taught by Professor Dmitri Feldman. He was vital in helping me take the project further, and his encouragement, feedback, and support was invaluable. Suffice to say, this paper would not exist without him. Thank you so much Professor Dima, you are truly an inspiration.

The language and formatting of this tutorial was edited by Aditi Marshan, an undergraduate English student. She ensured that the manuscript was grammatically correct, free of typos, and coherent. I will never forget her perpetual excitement when editing this, and her disapproval of my exclamation marks! I am in awe of the fact that she was able to drastically improve the quality of this paper without an extensive background in physics.

Research papers and talks by Professor Christopher Rose also helped me learn more about efficiency in communication and data storage. While I did not be directly reference his work in this paper, his work on molecular data storage has been influential in my interest in novel computing technologies.

Finally, a talk given by Andrew Helwer at Microsoft [27] was greatly helpful for my background research. It speaks mostly about the basics of quantum computing, and qubits, which helped structure this tutorial. It also briefly touches upon some algorithms, and I would highly recommend watching the recording.

9. References

[1] ICT Results. "How Small Can Computers Get? Computing In A Molecule." *ScienceDaily*. ScienceDaily, 30 December 2008. www.sciencedaily.com/releases/2008/12/081222113532.htm.

[2] Seabaugh, Alan. "The Tunneling Transistor." *IEEE Spectrum*, 30 Sept. 2013, spectrum.ieee.org/semiconductors/devices/the-tunneling-transistor.

[3] Brown, David Scott. "Why Not Ternary Computers?" *Techopedia*, 2 Aug. 2019, www.techopedia.com/why-not-ternary-computers/2/32427.

[4] Schumacher, Benjamin. "Quantum Coding." *Physical Review A*, vol. 51, no. 4, 1995, pp. 2738–2747., doi:10.1103/physreva.51.2738.

[5] Nielsen, Michael A., and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2010.

[6] Arute, Frank, et al. "Quantum Supremacy Using a Programmable Superconducting Processor." *Nature*, vol. 574, no. 7779, 2019, pp. 505–510., doi:10.1038/s41586-019-1666-5.

[7] Monroe, C., et al. "Demonstration of a Fundamental Quantum Logic Gate." *Physical Review Letters*, vol. 75, no. 25, 1995, pp. 4714–4717., doi:10.1103/physrevlett.75.4714.

[8] Mano, M. Morris, et al. *Logic and Computer Design Fundamentals.* Pearson, 2016.

[9] Krumm, Marius, and Markus P. Müller. "Quantum Computation Is the Unique Reversible Circuit Model for Which Bits Are Balls." *Npj Quantum Information*, vol. 5, no. 1, 2019, doi:10.1038/s41534-018-0123-x.

[10] Thornton, M.a., et al. "Chrestenson Spectrum Computation Using Cayley Color Graphs." *Proceedings 32nd IEEE International Symposium on Multiple- Valued Logic*, doi:10.1109/ismvl.2002.1011079.

[11] Charemza, Michal. "Examples of Quantum Circuit Diagrams." Warwick University, Apr. 2006, warwick.ac.uk/fac/sci/physics/research/cfsa/people/pastmembers/charemzam/pastprojects/mcharemza_quant_circ.pdf.

[12] Bloch, F. "Nuclear Induction." *Physical Review*, vol. 70, no. 7-8, 1946, pp. 460–474., doi:10.1103/physrev.70.460.

[13] Deutsch, David, and Jozsa, Richard. “Rapid Solution of Problems by Quantum Computation.” *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, 1992, pp. 553–558., doi:10.1098/rspa.1992.0167.

[14] Cleve, R., et al. “Quantum Algorithms Revisited.” *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 454, no. 1969, 1998, pp. 339–354., doi:10.1098/rspa.1998.0164.

[15] Thakkar, Jay. “Types of Encryption: 5 Encryption Algorithms & How to Choose the Right One.” The SSL Store, 16 Sept. 2020, www.thesslstore.com/blog/types-of-encryption-encryption-algorithms-how-to-choose-the-right-one/.

[16] Shor, P.W. “Algorithms for Quantum Computation: Discrete Logarithms and Factoring.” *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, doi:10.1109/sfcs.1994.365700.

[17] Coppersmith, D. “An Approximate Fourier Transform Useful in Quantum Factoring.” *IBM Internal Report*, 1994.

[18] Markov, Igor, and Mehdi Saeedi. “Constant-Optimized Quantum Circuits for Modular Multiplication and Exponentiation.” *Quantum Information and Computation.* May 2012.

[19] Høyer, Peter, et al. “Introduction to Recent Quantum Algorithms.” *Mathematical Foundations of Computer Science 2001 Lecture Notes in Computer Science*, Sept. 2001, pp. 62–74., doi:10.1007/3-540-44683-4_7.

[20] Gruber, Hermann, et al. “Sorting the Slow Way: An Analysis of Perversely Awful Randomized Sorting Algorithms.” *Lecture Notes in Computer Science Fun with Algorithms*, 2007, pp. 183–197., doi:10.1007/978-3-540-72914-3_17.

[21] Donald Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching, Second Edition. Addison-Wesley, 1998. ISBN 0-201-89685-0. Pages 106–110 of section 5.2.2: Sorting by Exchanging.

[22] Katajainen, Jyrki, and Jesper Larsson Träff. “A Meticulous Analysis of Mergesort Programs.” *Lecture Notes in Computer Science Algorithms and Complexity*, Mar. 1997, pp. 217–228., doi:10.1007/3-540-62592-5_74.

[23] Klauck, Hartmut. “Quantum Time-Space Tradeoffs for Sorting.” *Proceedings of the Thirty-Fifth ACM Symposium on Theory of Computing - STOC '03*, 2003, doi:10.1145/780542.780553.

[24] A. Odeh and E. Abdelfattah, "Quantum sort algorithm based on entanglement qubits {00, 11}," *2016 IEEE Long Island Systems*, Applications and Technology Conference (LISAT), Farmingdale, NY, 2016

[25] Toffoli, Tommaso. "Reversible Computing." Automata, *Languages and Programming Lecture Notes in Computer Science*, 1980, pp. 632–644., doi:10.1007/3-540-10003-2_104.

[26] Popkin, Gabriel. "China's Quantum Satellite Achieves 'Spooky Action' at Record Distance." *Science*, 27 Dec. 2018, www.sciencemag.org/news/2017/06/china-s-quantum-satellite-achieves-spooky-action-record-distance.

[27] Helwer, Andrew. "Quantum Computing for Computer Scientists." *Microsoft*, 2018, www.microsoft.com/en-us/research/uploads/prod/ 2018/05/40655.compressed.pdf.

[28] A. Odeh, K. Elleithy, M. Almasri and A. Alajlan, "Sorting N Elements Using Quantum Entanglement Sets", *Innovative Computing Technology (INTECH) 2013 Third International Conference on*, pp. 213-216, 2013.

The graphics and artwork in this tutorial are original, and were produced by Advay Mansingka. The information they convey is based on existing knowledge, and all the figures have been cited as such.