April 2017

Quantum Algorithms from a Linear Algebra Perspective

Lauren Michelle Baker
Worcester Polytechnic Institute

Liam Michael Ogren
Worcester Polytechnic Institute

Quantum Algorithms from a Linear Algebra Perspective

A Major Qualifying Project
Submitted to the Faculty of Worcester Polytechnic Institute
in partial fulfillment of the requirements for the Degree in Bachelor of Science
in
Computer Science, Mathematical Sciences, and Physics
By

Lauren M. Baker

Liam M. Ogren

Date: April 27, 2017

Advisors:

Padmanabhan K. Aravind

William J. Martin

Abstract

The field of quantum computing has gained much attention in recent years due to further advances in the development of quantum computers and the recognition that this new paradigm will greatly endanger many modern encryption practices. In 1992, D. Deutsch and R. Jozsa published a quantum algorithm which was provably faster than the optimal classical counterparts. This challenge to the Church-Turing thesis sparked the interest in quantum algorithms to see if other problems could be solved faster using a quantum approach.

This paper gives analysis of some of these algorithms, notably Grover's database search algorithm, and Shor's factoring and discrete log algorithms, from the perspective of linear algebra. The consequences these have for modern cryptography are discussed, and a brief overview of the current state of the field is given. Without assuming a physics background, this paper aims to provide a self-contained and mathematically rigorous explanation of quantum algorithms for an undergraduate audience.

Contents

Chapter 1

Introduction

It is without question that the exchange of information through the internet is ubiquitous in our society. We trust this security when viewing a bank account balance or medical records online, as well as in more mundane tasks like entering a password for a social media account. In these situations, we trust that the messages to and from the website are kept secret, even in the presence of eavesdroppers. The techniques to accomplish this are known broadly as encryption.

Finding successful encryption techniques is an ever-present challenge as newer attacks are devised, and one of the latest threats has distinctly different capabilities. A purely theoretical concept at one point, quantum computers are likely to be a reality in the next few years by some estimates, and will be capable of defeating many widely used encryption algorithms.

There are two broad categories of cryptographic algorithms: symmetric-key and asymmetric-key. In symmetric-key cryptography, the two parties need to decide on a secret key (e.g., a number) to use ahead of time. This key is used by both parties to encrypt messages before sending, and decrypt received messages. Without the key, eavesdroppers will not be able to understand the conversation.

The other category is asymmetric-key or public key cryptography. In these systems, each individual generates a key pair, with a public key and a private key. Anyone can use the public key and the encryption algorithm to create a message for this individual, but only the individual has the private key needed to use the decryption algorithm and recover the message. For an asymmetric-key algorithm to be secure, it must be infeasible for an attacker to deduce the private key from the public key. This critical feature is often achieved by relying on mathematical problems where no efficient algorithm has been found, even after decades or centuries of effort.

For example, the RSA algorithm relies on the difficulty of factoring very large numbers with only two factors aside from one and themselves. These so-called "semiprime" numbers are found by multiplying two large primes. The public and private keys are derived from the specific semiprime number. The encryption and decryption operations both require only repeated multiplication, which computers can complete very quickly. To decrypt an intercepted message, however, requires the private key, and to do so using only the public

one in a reasonable time frame requires efficient factoring as far as we can tell.

According to the International Association for Cryptologic Research, a scientific non-profit organization, the largest "semiprime" number of the sort used in the RSA algorithm, to be factored is a number requiring 768 binary digits, or 232 decimal digits, to represent. (For context, this is a number larger than 7×10^{230}.) The factoring was a two-year effort, completed in 2009 by a group of academic researchers. The project was completed using many computers, but required the equivalent of almost two thousand years on a single-core 2.2 GHz AMD Opteron (a style manufactured from 2003-2006). The researchers noted that a 1024-bit number would be "about one thousand times harder to factor," and estimated that through a similar academic effort, it "may be possible... within the next decade" [10].

The importance of running time becomes apparent as key size increases. A 1024-bit number is about 1.3 times larger in size (in bits) than the 768-bit subject of the research effort, but it is roughly a thousand times more difficult to factor. A 2048-bit number meanwhile, would be over a trillion times more difficult to factor. The increase here is dramatic, while the storage requirement and the encryption and decryption operations remain very reasonable. Both of these operations, necessary to send and read messages, can be completed in less than time proportional to the key length (in bits) cubed. Decryption using a 2048-bit key takes roughly eight times as long as for a 1024-bit key, and the increase for encryption is slightly less. Both operations can be completed in well under a second on a typical laptop available today using 2048-bit keys.

Computers are capable of completing more operations per second with every passing year, but with the number field sieve as the best known method for factoring RSA numbers, increases in key length keep information secure. Consider the rough estimate that processor speeds double every 18 months. If that 768-bit number would take two thousand years of computing time, then in 18 years, this estimate would be under six months. However, for a 1024-bit number, the estimate would be almost 600 years, and for a 2048-bit number, over 694 billion years. As this example illustrates, there is a massive increase in the necessary time to break an encryption scheme based on the RSA algorithm from an increase in key size.

It is worth discussing the disparity between the time estimates for a single processor and expected results for multiple processors. In a factorization before RSA-768, some of the same researchers identified the two steps of the algorithm requiring the most effort. The first could be split over many processors working independently, while the second was essentially limited by the size of available computer clusters and formed the 'bottleneck' for the factoring algorithm [1]. Both factorizations noted that further challenges in terms of parallelization would appear for larger numbers, so reducing the time estimates given for a single processor at a rate to keep up with increases in key length is not as simple as acquiring more and faster computers.

Enter the quantum computer. A classical computer stores information in the form of bits, which may be modeled as taking the value 0 or 1 at any given time. A quantum computer, on the other hand, stores information using qubits, which can be modeled as an entity in a "superposition" of 0 and 1. At any given time, the qubit can be thought of as being 0 with

some probability, and 1 with the remaining probability. A sequence of 8 bits in a classical computer can store exactly one of 256 different values at a time, but the equivalent 8-qubit "quantum register" has some probability of being in each of those 256 values at any given time. Where a classical computer may change the value stored by these 8 bits, a quantum computer would work by changing the probability of the 8 qubits representing each of the 256 values.

In intuitive terms, a quantum computer is more powerful because it carries out calculations over a distribution of all possible values, instead of just one. It turns out that any calculation which is possible for a computer today can also be carried out using a quantum computer. While using a classical computer, the calculation is completed based on a single input, a quantum computer can complete the calculation for exponentially many inputs. This massive parallelism has a catch, however: we will only receive one output.

This catch adds a challenge to creating quantum algorithms. While a quantum computer manipulates superpositions of values, the result returned is only one value. Before a result can be found, the qubits must be measured, which causes the superposition of values to collapse into one of the values involved. This collapse happens based on the probability distribution, or superposition, however. The key to each quantum algorithm is the manipulation of these probabilities. An algorithm may start with a quantum register in an equal superposition of all possible values, but after the chosen sequence of operations, the probability of returning the value (or one of the values) corresponding to the correct answer is higher. If the algorithm is executed several times, and each result written down, the correct answer will show up more often. For many problems, including factoring, it is far easier to verify a proposed solution than to find the solution from scratch, so these probabilistic algorithms can be immensely useful.

Manipulating these probability distributions allows for more efficient algorithms than the best known classical algorithms for certain problems, including factoring. In 1994, Peter Shor published a paper with a factoring algorithm to run in time proportional to the key length cubed, the same efficiency seen for the encryption and decryption steps on a classical computer [20]. While a doubled key length saw a billion-fold increase in computing time needed to break the encryption on a classical computer, there would be only an eight-fold increase for a quantum computer. If a quantum computer large enough to carry out these computations is created, RSA encryption will not be considered secure. It is said that RSA encryption is not "quantum-resistant".

This problem is not unique to algorithms based on factoring. The Diffie-Hellman protocol, a key-agreement scheme, relies on the difficulty of the discrete log problem. The Diffie-Hellman protocol is used by two parties to establish a key for symmetric-key cryptography, while an eavesdropper able to hear the entire conversation is unable to deduce the key. A variation on Shor's algorithm for factoring can solve instances of the discrete log problem, also in probabilistic polynomial time.

There are other public-key cryptography approaches which are threatened by the existence of a quantum computer. Elliptic curve cryptography, a common alternative to RSA encryption, relies on a variation of the discrete log problem, and probabilistic polynomial

time quantum algorithms based on Shor's work have been devised.

As quantum algorithm attacks on RSA and other public-key schemes were published, interest has grown in finding quantum-resistant public-key schemes. Lattice-based schemes offer some promise of quantum-resistance, but their security has not been conclusively evaluated.

There are two mathematical problems based on the concept of lattices which are considered good candidates for public-key encryption schemes, the Shortest Vector Problem (SVP) and Closest Vector Problem (CVP). A number of lattice-based cryptographic algorithms have been proposed, which use these difficult problems in different ways. Based on the implementations chosen in each scheme, there is the possibility of breaking the encryption without solving the SVP or CVP. Some schemes have proofs to confirm that this possibility does not exist, but these, so far, have been less efficient to implement in practice. One of the more efficient schemes proposed is NTRU, based on the SVP, but there is no proof as of yet that recovering any key in this scheme is as hard as solving the SVP.

There have been a series of attempts to find a quantum algorithm to break a simpler lattice-based scheme, Soliloquy. The latest, published in 2016 by Biasse and Song, resolves some flaws in earlier attempts. The particular attack used, as the authors state, is also applicable to some other proposed lattice-based schemes. It is possible as the private key can be recovered from the public keys used in Soliloquy through solving a simpler problem than the SVP, and a polynomial time quantum algorithm for the simpler problem was devised. The more complex NTRU is not vulnerable to this algorithm, but it remains to be seen whether this attack can be extended further, or another devised, as there is no proof to the contrary for this scheme.

There is a common element between all of the mathematical problems considered thus far, factoring, the discrete log problem, SVP, and CVP. In complexity theory, the study of the hardness of particular problems to solve, the mathematical problems used in public-key cryptography systems are all known to exist in the class known as NP, but believed to be outside of one of its sub-classes, P. In short, a problem is in NP if a proposed instance and a solution (e.g. for factoring, a semiprime number and a claim of two factors,) can be verified in polynomial time, and a problem is in P if it can be solved in polynomial time. There is an even more specific category, NP-intermediate, inside of NP but outside P. For these problems, a proposed instance and claimed solution can also be 'debunked' in polynomial time. (Factoring is in this category, as multiplying the factors gives a clear yes or no response.) There are relatively few problems proposed to exist in this set, which includes the discrete log problem, the aforementioned factoring, and the Gap-SVP and Gap-CVP. These two problems are related to the SVP and CVP, which seem to be in a higher complexity set, and the questions of whether efficient quantum algorithms for these problems exist, and whether they might be used to solve instances of other lattice-based encryption schemes without security proofs, naturally arise.

Quantum computers may not be too far away. In MIT's 2017 "10 Breakthrough Technologies" list, quantum computers made the fourth spot, with predictions of systems with 30 to 100 qubits "stable enough to perform a wide range of computations for longer durations"

available in as little as two to five years. In addition to Google's team, IBM, Microsoft and Intel have made investments in the field [9]. With the development of these machines rendering widely-used public-key encryption schemes vulnerable, while the security of others remains to be seen, more work needs to be done in understanding the potential of quantum computation to ensure our future communications remain secure.

Chapter 2

Quantum Algorithms

To understand the unique capabilities of quantum computers as opposed to classical computers, we must begin with a mathematical definition of a qubit. We introduce the options for manipulating qubits, and illustrate the basic operations involved. We discuss the Deutsch-Jozsa algorithm, as a first example of a task that a quantum computer can perform more efficiently than a classical computer. The Deutsch-Jozsa problem is not one with important applications, but it demonstrated that quantum computers could complete at least one task more efficiently than any classical computer.

We close with the Quantum Fourier Transform (QFT), a key tool which is used in Shor's algorithms for factoring and solving the discrete log problem. It is worth noting that these two algorithms are almost entirely classical, but the Quantum Fourier Transform plays an important role in the novel portion of the algorithm. Understanding this transformation, then, is key to our discussion on Shor's work and its implications for modern cryptography.

2.1 Gates and Measurements

The goal of this section is to understand the components of a quantum computer, from a mathematical perspective. Indeed, to the present day, the "quantum computer" is an entirely mathematical concept. We begin by defining the qubit in purely mathematical terms, and the extension for considering a collection of multiple qubits. After establishing a representation for the qubit, we can describe the operations on qubits and sets of qubits used in quantum algorithms. We introduce the terminology and give a few examples before covering the possible operations in more depth.

The qubit serves the role of a bit in a classical computer, the most basic unit of information storage. While the state of a bit is either a 0 or 1 at all times, the state of a qubit is modeled as a linear combination of 0 and 1.

Vectors will most frequently be written using *Dirac* notation, which we take a moment to describe. Column vectors are represented as "ket" vectors, written $|x\rangle$, while the notation $\langle y|$ indicates a row vector, called a "bra" vector. In general for vectors x and y, we have the Hermitian inner product $\langle y|x\rangle = |y\rangle^\dagger |x\rangle = \sum \bar{y}_i x_i$ where † indicates the conjugate transpose

operation.[1]

We define the *length* of $|x\rangle$ as $\sqrt{\langle x|x\rangle}$ and denote it by $\| |x\rangle \|$. Throughout this report, each complex vector space is assumed to come equipped with a Hermitian inner product of this sort.

The quantum states we consider all reside in finite-dimensional spaces. For this section, we refer the reader to Postulate 1 in [15][Sec. 2.2.1].

Definition 2.1. A *qubit* is a two-dimensional complex vector space V with a distinguished unitary basis $\{|0\rangle, |1\rangle\}$. A qubit in *state* $|v\rangle$ is an ordered pair $(V, |v\rangle)$, where V is a qubit and $|v\rangle \in V$, and $\| |v\rangle \| = \langle v|v\rangle = 1$. Equivalently, there exist $\alpha, \beta \in \mathbb{C}$ so that $|v\rangle = \alpha |0\rangle + \beta |1\rangle$ with $\alpha\bar{\alpha} + \beta\bar{\beta} = 1$.

Independent of its physical realization, we model every qubit as a vector space and a specified unitary basis. The vectors $|0\rangle$ and $|1\rangle$ are the most common choice of basis. We identify each vector with its coordinate vector relative to the ordered basis $\{|0\rangle, |1\rangle\}$:

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

Whatever state a particular qubit is found in, this is represented as a linear combination of the unitary basis elements. As it is frequently necessary to work with multiple qubits, we next establish a representation for this scenario as well.

Definition 2.2. An *N-state quantum register* is an N-dimensional complex vector space V with distinguished basis $\{|a\rangle : 0 \leq a < N\}$. An N-state quantum register in state $|v\rangle$ is defined analogously to a qubit in a specified state.

The set $\{|a\rangle : 0 \leq a < N\}$ is called the *computational basis* and its elements are referred to as *basis states*. It is sometimes convenient to index the basis states by a set other than $\{0, 1, \ldots, N-1\}$. For example, as we will see, the index set $\mathbb{Z}_2^n = \{a_1 a_2 \ldots a_n : a_j \in \mathbb{Z}_2\}$ is quite natural.

During computations, qubits are acted upon by unitary matrices and measurements (Cf. Postulates 2 and 3 in [15][Sec. 2.2]).

Definition 2.3. A square matrix M with complex entries is *unitary* if

$$M^\dagger M = M M^\dagger = I,$$

where $M^\dagger$ is the conjugate transpose of M.

The rows, as well as the columns, of a unitary matrix M form a *unitary basis*. Vectors in a unitary basis have length one, and for distinct $|x\rangle$ and $|y\rangle$ in the basis, $\langle x|y\rangle = 0$.

We define $\mathcal{U}(n)$ to be the group of $n \times n$ unitary matrices. The unitary matrix $U \in \mathcal{U}(2)$ acting upon the qubit V in the state $|v\rangle$ is written as $|U|v\rangle$. The matrix U acts on the qubit V in state $|v\rangle$ mapping it to qubit V in state $|U|v\rangle$. [2]

[1] This inner product is linear in its second argument, and conjugate linear in its first argument.

[2] We treat $|H|v\rangle$ and $H|v\rangle$ as equal and use whichever notation is clearest in context.

Let us consider an example before continuing. We begin with a qubit in the state

$$|v\rangle = |0\rangle .$$

We choose to act upon the state of this qubit with the matrix

$$U = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

(It can be verified that this matrix is unitary by computing $UU^\dagger$.) If we apply this matrix to this state, we find

$$|U|v\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle)$$

This matrix U changed the state of the qubit from $|0\rangle$ to a linear combination of $|0\rangle$ and $|1\rangle$, with equal coefficients. We will see this matrix again when we discuss quantum gates. It is used frequently in quantum algorithms, starting with the Deutsch-Jozsa algorithm, to transform a state equal to a specific basis state to a combination of basis states. When we discuss measurement, we will see why a state with equal coefficients for all basis states is useful.

Returning to the definition of the qubit for a moment, note the requirement on the coefficients for the state. The condition that the squared moduli of the coefficients sum to one ensures that the state is *normalized*. Acting upon the state with any unitary matrix may change the coefficients, but this property will still hold. This can be verified for the state of the qubit in the previous example, before and after the application of U. We will revisit this property when discussing measurements.

We extend the definition for a single qubit to cover a collection of qubits. The ability to complete operations involving multiple qubits is crucial to the algorithms presented in this section.

Two qubits can be considered together by taking the direct product, or Kronecker product, of the vector spaces for the individual qubits. These two qubits are a four-dimensional complex vector space $V \cong \mathbb{C}^2 \otimes \mathbb{C}^2$. The Kronecker product is used again when representing the state of the two qubits. If both qubits are in the state $|0\rangle$, we represent this state as $|0\rangle \otimes |0\rangle$, which is abbreviated $|00\rangle$. Considering the two basis states for each qubit, we arrive at our unitary basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. An arbitrary state $|v\rangle \in V$ can be represented as

$$|v\rangle = \alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle .$$

To represent additional qubits, this process is repeated with further Kronecker products. We refer to these collections of qubits as *n-qubit quantum registers*, which are defined formally as follows.

Definition 2.4. An *n-qubit quantum register* is a 2^n-dimensional complex vector space V with a distinguished unitary basis which, unless otherwise specified, is denoted $\{|a\rangle : a \in \mathbb{Z}_2^n\}$.

An n-qubit quantum register in a state $|v\rangle$ is a pair $(V, |v\rangle)$ where V is as above and $|v\rangle \in V$ with $\langle v|v\rangle = 1$. Equivalently, there is a $\gamma_a \in \mathbb{C}$ for $a \in \mathbb{Z}_2^n$ with

$$|v\rangle = \sum_{a \in Z_2^n} \gamma_a |a\rangle , \qquad \sum_a |\gamma_a|^2 = 1. \tag{2.1}$$

The vector space isomorphism $\mathbb{C}^{2^n} \cong \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2$ allows us to view an n-qubit quantum register as a tensor product of n qubits. Informally, the register is viewed as simply consisting of n qubits. We will refer to an n-qubit quantum register as n qubits in further discussion, or, where the number of qubits has been established, as a register.

We pause to discuss a simple example of operating on two qubits.

We model the joint system by $\mathbb{C}^4$ with basis $\{|00\rangle , |01\rangle , |10\rangle , |11\rangle\}$ where each qubit is modeled by $\mathbb{C}^2$ with basis $\{|0\rangle , |1\rangle\}$.

The isomorphism $\mathbb{C}^4 \cong \mathbb{C}^2 \otimes \mathbb{C}^2$ is realized by

$$\begin{aligned} |00\rangle &\leftrightarrow |0\rangle \otimes |0\rangle , \\ |01\rangle &\leftrightarrow |0\rangle \otimes |1\rangle , \\ |10\rangle &\leftrightarrow |1\rangle \otimes |0\rangle , \\ \text{and } |11\rangle &\leftrightarrow |1\rangle \otimes |1\rangle . \end{aligned}$$

Operations on this register, referred to as *gates* or *operators*, may affect just the first qubit, just the second qubit, both together through a single operation, or both qubits independently.

A gate which acts on the state of a single qubit is represented by a 2×2 unitary matrix, and two of these may be combined to act on the state of a 2-qubit quantum register using the Kronecker product. Keeping in mind the algebraic properties of Kronecker products, consider the following examples.

The matrix representation of a gate which "flips" the state of the first qubit, while leaving the state of the second qubit unchanged, is

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} . \tag{2.2}$$

Note that when the identity matrix acts on the state of a qubit, the state is left unchanged. We might apply an arbitrary unitary matrix U to the state of the second qubit, while leaving the state of the first qubit unchanged, using

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} u_1 & u_2 \\ u_3 & u_4 \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & 0 & 0 \\ u_3 & u_4 & 0 & 0 \\ 0 & 0 & u_1 & u_2 \\ 0 & 0 & u_3 & u_4 \end{bmatrix} .$$

We can operate on both qubits simultaneously and independently as well, with Kronecker products of any two matrices in $\mathcal{U}(2)$, such as

$$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & -i & 0 \\ 0 & i & 0 & 0 \\ i & 0 & 0 & 0 \end{bmatrix}.$$

But not all unitary operations on this system are representable in this form. For example,

$$U = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

is not expressible as $u_1 \otimes u_2$ for any $u_1, u_2 \in \mathcal{U}(2)$. This is not difficult to see, as $U = u_1 \otimes u_2$ forces $\begin{bmatrix} 1 & -1 \\ -1 & i \end{bmatrix}$ to be a scalar multiple of $\begin{bmatrix} 1 & 1 \\ 1 & i \end{bmatrix}$ which is not possible. What we are seeing here is a glimpse of what is called entanglement, an important concept in quantum mechanics and quantum computing. We do not discuss this, but an account of this phenomenon can be found in [13][Chap. 4].

2.1.1 Measurements

When a quantum algorithm returns a result, it is through a measurement. While the state of a qubit is a linear combination of the elements of the unitary basis, the result of a measurement is a number. After the measurement, the state of the qubit is a certain state which corresponds to this number. This is not to say that a measurement is only used at the end of a quantum algorithm. A measurement may be used simply for its effect on the state of the qubit.

In this report, we consider only Von Neumann measurements. There are other, more general ways to define a measurement, but the algorithms we discuss only require these simple measurements. (Cf. Postulate 3. in [15][Sec. 2.2.3].)

Definition 2.5. A *measurement* of quantum system V is a set $\mathcal{M} = \{S_0, \ldots, S_k\}$ where $S_0, \ldots, S_k$ are subspaces of V such that $S_0 \perp \ldots \perp S_k = V$. If $|u\rangle \in S_i$ and $|v\rangle \in S_j$, and $\langle u|v\rangle \neq 0$, then $i = j$.

Now, we explain the behavior of the measurement $\mathcal{M}$. As the orthogonal direct sum of the subspaces S_0 to S_k is equal to the vector space V, and the intersection of any two of these subspaces is by definition the zero subspace, any vector in V can be uniquely expressed as a sum of one vector from each of the subspaces.

We choose to express these vectors such that each $|v_j\rangle$ is a unit vector and use scalars $\alpha_0, \ldots, \alpha_k \in \mathbb{C}$, so that $|v\rangle = \alpha_0 |v_0\rangle + \cdots + \alpha_k |v_k\rangle$ where $\sum \|\alpha_j\|^2 = 1$.

The process of taking a measurement is expressed through several steps.

STEP 0: Note that we can express the current state of the quantum system $|v\rangle$ in terms of the subspaces S_0 to S_k, as $|v\rangle = \alpha_0 |s_0\rangle + \alpha_1 |s_1\rangle + \cdots + \alpha_k |s_k\rangle$ where for $0 \leq j \leq k$, $|s_j\rangle \in S_j$, with $\langle s_j | s_j \rangle = 1$ is uniquely determined up to phase[3] when $\alpha_j \neq 0$. Assuming $|v\rangle$ is a normalized vector, we have $\sum_{j=0}^{k} \alpha_j \bar{\alpha_j} = 1$.

STEP 1: Choose one value[4] $j \in \{0, \ldots, k\}$, where the probability of choosing j is $\alpha_j \bar{\alpha_j}$.

STEP 2: The chosen j is returned as the outcome of the measurement. We denote the probability of the measurement returning j as $Pr[MR = j]$.

STEP 3: The state, now that the measurement is complete, is $|s_j\rangle$. If $|v\rangle$ is entirely in one of the subspaces, the corresponding j is returned with probability one, and $|v\rangle$ is unchanged.

We introduce some notation to denote the probability that this measurement returns any given value; we write $Pr[MR = j] = \alpha_j \bar{\alpha_j}$ for $0 \leq j \leq k$.

Let us consider an example measurement at this point. Take the 2-qubit quantum register with vector space $V \cong \mathbb{C}^2 \otimes \mathbb{C}^2$. Our unitary basis is $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$.

We choose $\mathcal{M} = \{S_0, S_1\}$, where $S_0 = \text{span}\{|00\rangle, |01\rangle\}$ and $S_1 = \text{span}\{|10\rangle, |11\rangle\}$.

This measurement acts on pure tensors $|\phi\rangle \otimes |\psi\rangle$ as

$$\mathcal{M} : |\phi\rangle \otimes |\psi\rangle \mapsto |b\rangle \otimes |\psi\rangle$$

where $b \in \{0, 1\}$.

As this second expression in particular suggests, this choice of $\mathcal{M}$ is akin to measuring the first qubit. The first subspace contains the basis vectors where the first qubit is in the state $|0\rangle$, and the second contains the basis vectors where the first qubit is in the state $|1\rangle$.

Suppose the qubit is in the state

$$|v\rangle = \frac{\sqrt{7}}{4} |00\rangle + \frac{1}{2} |01\rangle - \frac{1}{2} |10\rangle + \frac{i}{4} |11\rangle$$

initially. Note that this state meets the criteria on the coefficients. We can calculate $\langle v | v \rangle$ and verify the result is 1.

We first express $|v\rangle$ in the form $\alpha_0 |s_0\rangle + \alpha_1 |s_1\rangle$ where $|s_0\rangle \in S_0$, $|s_1\rangle \in S_1$, and $\langle s_0 | s_0 \rangle = \langle s_1 | s_1 \rangle = 1$. Using the coefficients for $|00\rangle$ and $|01\rangle$, $\alpha_0 \bar{\alpha_0} = (\frac{\sqrt{7}}{4})^2 + (\frac{1}{2})^2 = \frac{11}{16}$, and $|s_0\rangle = \sqrt{\frac{16}{11}} \left(\frac{\sqrt{7}}{4} |00\rangle + \frac{1}{2} |01\rangle \right)$. Similarly, $\alpha_1 \bar{\alpha_1} = \frac{5}{16}$ and $|s_1\rangle = \sqrt{\frac{16}{5}} \left(-\frac{1}{2} |10\rangle + \frac{i}{4} |11\rangle \right)$.

Note that $|s_0\rangle$ and $|s_1\rangle$ are unit vectors in S_0 and S_1 respectively, and $\alpha_0 \bar{\alpha_0} + \alpha_1 \bar{\alpha_1} = 1$.

Now we describe the outcomes of the measurement, with their respective probabilities. The result will be 0 with probability $\frac{11}{16}$, leaving the qubit in the state $|s_0\rangle$, and 1 with probability $\frac{5}{16}$, leaving the qubit in the state $|s_1\rangle$. These may be thought of as the outcome where the first qubit is in the state $|0\rangle$ and the state $|1\rangle$ after measurement, respectively.

This is one choice for $\mathcal{M}$ given a 2-qubit quantum register, but there are other valid options. For instance, $\{S_0, S_1, S_2, S_3\}$ where $S_0 = \text{span}\{|00\rangle\}$, and so on, up to $S_3 = \text{span}\{|11\rangle\}$.

[3]Two vectors $|v\rangle$ and $|w\rangle$ with $|v\rangle = |w\rangle e^{i\theta}$ for some real θ are said to differ only in their phase.

[4]Note that if $\alpha_j = 0$, j will never be returned.

The four basis states could also be divided based on the second qubit, so that $\mathcal{M} = \{S_0, S_1\}$ where $S_0 = \text{span}\{|00\rangle, |10\rangle\}$ and $S_1 = \text{span}\{|01\rangle, |11\rangle\}$.

Consider the Kronecker product V of two registers W_1 and W_2, so that $V = W_1 \otimes W_2$. Let W_1 be an N-state quantum register with the computational basis $\{|j\rangle : 0 \leq j < N\}$. To take a measurement of V which could be said to "measure the first register", choose $\mathcal{M} = \{S_j \otimes W_2 : 0 \leq j < N\}$ where $S_j = \text{span}\{|j\rangle\}$. The result of this measurement would be a j indicating a basis state of W_1. The state of V would be the Kronecker product of $|j\rangle$ with the original state of W_2. In the special case where this measurement is applied to a pure tensor $|\phi\rangle \otimes |\psi\rangle$, $|\phi\rangle \in W_1$, $|\psi\rangle \in W_2$, the information stored in W_2 is not lost or modified as a result of this measurement[5].

Many authors express measurements of quantum systems using the language of Hermitian operators. Informally, an operator is applied to a quantum state, an eigenvalue of that operator is returned, and the state collapses to an eigenvector in the corresponding eigenspace. We now show how, in the finite-dimensional case, this is exactly what we have described above. Consider a measurement $\mathcal{M} = \{S_0, \ldots, S_k\}$ for the N-state quantum register V. Denote by $d_j = \dim S_j$ and let

$$\{|u_{j,r}\rangle : 1 \leq r \leq d_j\} \tag{2.3}$$

denote a unitary basis for S_j. Note that $\langle u_{j,r}|u_{j,s}\rangle = \delta_{r,s}$. The linear transformation $\pi_j : V \to S_j$ effecting orthogonal projection onto S_j is defined by

$$\begin{array}{c} \pi_j : |v\rangle \mapsto |v\rangle \text{ for } |v\rangle \in S_j \\ \pi_j : |v\rangle \mapsto \mathbf{0} \text{ for } |v\rangle \perp S_j \end{array}, \tag{2.4}$$

where $\mathbf{0} \in V$ is the zero vector.

It is well-known that the matrix representing π_j is

$$P_j = \sum_{r=1}^{d_j} |u_{j,r}\rangle \langle u_{j,r}| . \tag{2.5}$$

Choose any real eigenvalues $\theta_0, \ldots, \theta_k$ for our operator: for simplicity, select $\theta_j = j$. Then the matrix

$$A = \theta_0 P_0 + \cdots + \theta_k P_k \tag{2.6}$$

is Hermitian with eigenvalue θ_j belonging to eigenspace S_j. We can then state

$$A = \sum_{j=0}^{k} \theta_j \sum_{r=1}^{d_j} |u_{j,r}\rangle \langle u_{j,r}| . \tag{2.7}$$

In the Hermitian operator interpretation of our measurement, we have

STEP 0: The current state $|v\rangle$ is uniquely expressible as a sum of eigenvectors $|s_0\rangle, |s_1\rangle, \ldots, |s_k\rangle$ with $|s_j\rangle \in S_j$.

[5]When a state is expressible as $|\phi\rangle \otimes |\psi\rangle$, we will sometimes abbreviate this as $|\phi\rangle |\psi\rangle$ when it improves readability.

STEP 1: An eigenvalue is selected at random with the probability of θ_j being chosen equal to $\langle s_j|s_j\rangle$. (Note that $\langle v|v\rangle = 1$ forces $\sum_{j=0}^{k} \langle s_j|s_j\rangle = 1$.)

STEP 2: The classical information returned by the measurement is the eigenvalue θ_j.

STEP 3: As a result of the measurement, the state collapses to the corresponding eigenvector: A moves $|v\rangle$ to $\frac{1}{\||s_j\rangle\|}|s_j\rangle$.

The reader may compare this to our first description of measurement and see that they are equivalent.

2.2 Quantum Gates

When a classical computer executes a computation, it is through a circuit with a sequence of gates representing logical operations, referred to as logic gates. The actions of these gates are often represented using truth tables, which list the possible inputs and the corresponding outputs. The input and output may be represented using T or F for "True" or "False", or by using 1 and 0 where these are understood to mean true and false, respectively.

The logic gates in a circuit correspond to operations in Boolean logic. Without delving too deeply into the topic, the operations AND, OR, and NOT may be combined to form any Boolean formula with n input bits, the building blocks for algorithms on a classical computer. In fact, the gate NAND, which takes two inputs and computes AND and then NOT, is by itself enough, because AND, OR, and NOT can be constructed using the NAND gate. Because any classical computation can be completed using only the NAND gate, it is said that the NAND gate is *universal* [25][Sec. 2.1.3].

a	b	$\neg(a \wedge b)$
0	0	1
0	1	1
1	0	1
1	1	0

Table 2.1: The truth table corresponding to the NAND operation.

Notice that the NAND operation has two inputs and a single output. From the truth table for NAND, we see that three of the four input possibilities map to the output 1. From only the output 1, it is not possible to determine the input a and b. As there are multiple inputs mapped to the same output, we say that NAND is an *irreversible* computation.

Irreversible computations are commonly used in classical computing, but with the quantum gates we now introduce, we will insist that all operations are reversible.

Quantum gates, and more generally, unitary operators, act on the state of a qubit or a register. While the effect of a classical gate is best summarized by a truth table, each quantum gate, as a linear transformation, is naturally represented as a matrix. We find

ourselves multiplying matrices and vectors to model gates acting on qubits or registers. A fundamental postulate of quantum mechanics tells us that every quantum gate corresponds to a unitary matrix; we will discuss this later. The question as to which unitary matrices should be categorized as gates is an ongoing discussion.

Throughout the report, we may refer to a gate acting on a qubit or several qubits as shorthand for the corresponding transformation acting on the vector space representing those qubits. We also extend this language to registers.

Quantum circuits, combinations of gates and measurements acting on qubits, are represented using *wire diagrams*. In these diagrams, each horizontal line represents a qubit, and gates are represented as labeled rectangles. If a gate acts on the state of a particular qubit, the line representing this qubit intersects with the rectangle. Each such gate acts on the entire quantum register via taking a tensor product with identity matrices of appropriate size. Identity components in such an operator are suppressed in wire diagrams and the corresponding "wires" are displayed as untouched by the gate. A rectangle with a small meter symbol indicates a measurement only of the qubits represented by the wires passing through that rectangle. In wire diagrams, time passes with movement from the left to the right. When we refer to the qubits represented individually, the uppermost wire is the 'first' qubit.

In the example wire diagram in Figure 2.1, there are two qubits. First, a NOT gate $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ acts on the second qubit, and then a Hadamard gate $H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ acts on each of the qubits. A gate denoted by some unitary transformation U_f acts on both qubits, followed by a Hadamard gate on the first qubit. Finally, the state of the first qubit is measured. We will define the NOT and Hadamard gates in this section. If this wire diagram were used in context, U_f would be defined as a 4×4 matrix.

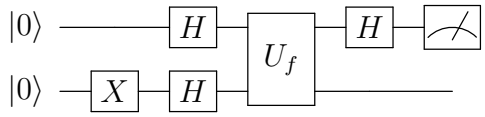


Figure 2.1: An example of a wire diagram.

Let us now pick apart the diagram in Figure 2.1. The state is first acted upon by $I \otimes X$, then by $H \otimes H$, and so on. Just before the measurement, the state of the system is

$$(H \otimes I)U_f(H \otimes H)(I \otimes X)\left|00\right\rangle.$$

It is important to note that matrix multiplication occurs in reverse order from the order displayed in the diagram. Our measurement measures the first qubit only, so it is

$$\mathcal{M} = \{\text{span}\{\left|00\right\rangle, \left|01\right\rangle\}, \text{span}\{\left|10\right\rangle, \left|11\right\rangle\}\}.$$

2.2.1 Single-Qubit Gates

In this subsection, we enumerate a selection of commonly used quantum gates, such as X and Hd. We begin with a discussion of gates acting on a single qubit. Each gate is

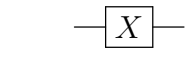
defined by a unitary matrix, and where it is most helpful we include the effect of the gate on each basis state from the computational basis. Each gate has a representation for use in wire diagrams. We use wire diagram representations, as well as definitions of gates, from [15] unless otherwise noted. The wire diagram representations of the single-qubit gates we describe may be seen in Figure 2.2.

We begin with the Pauli Gates, which are a basic set of quantum gates operating on single qubits. They are represented by the matrices

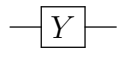
$$P_X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad P_Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \qquad \text{and} \qquad P_Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

It is simple to see that these matrices are unitary, as $P_X^\dagger P_X = P_Y^\dagger P_Y = P_Z^\dagger P_Z = 1$.

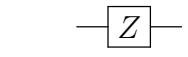
The Pauli-X gate acts as a NOT gate for quantum circuits. We referenced this operation in the example above. Its function when acting on a state is to interchange the coefficients of the two basis states.



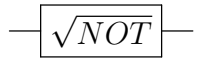
(a) Pauli-X gate, or NOT gate



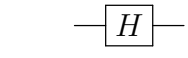
(b) Pauli-Y gate



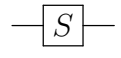
(c) Pauli-Z Gate



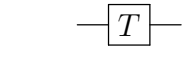
(d) $\sqrt{NOT}$ gate



(e) Hadamard gate



(f) Phase gate



(g) $\pi/8$ gate

Figure 2.2: A selection of single-qubit gates.

The $\sqrt{NOT}$ gate is a linear operator which, when applied twice to the state of a qubit, has the same result as a NOT gate, as defined by [25].

The matrix representation of this gate is

$$\sqrt{NOT} = \frac{1}{2}\begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}. \tag{2.8}$$

One easily verifies that $\sqrt{NOT}\sqrt{NOT} = NOT$, by matrix multiplication.

The Hadamard gate also acts on a single qubit. It is a staple in quantum algorithms, as it is able to create a uniform superposition of the two states in the computational basis from the state $|0\rangle$. We will see in later algorithms that it is used to create a uniform superposition of all states in the computational basis for an n-qubit quantum register.

The Hadamard gate performs the transformation on the state

$$\alpha_0 |0\rangle + \alpha_1 |1\rangle \mapsto \frac{\alpha_0 + \alpha_1}{\sqrt{2}} |0\rangle + \frac{\alpha_0 - \alpha_1}{\sqrt{2}} |1\rangle. \tag{2.9}$$

This gate is represented by the matrix

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.10}$$

Since $H^\dagger H = I$, this matrix is unitary and the Hadamard gate is a valid quantum gate.

We also define the phase gate, and $\pi/8$ gates, with the matrices S and T, respectively. We will use these gates in a universal set of quantum gates, established at the conclusion of this section.

The phase and $\pi/8$ gates correspond to unitary matrices

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \qquad \text{and} \qquad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\pi i/4} \end{bmatrix}. \tag{2.11}$$

2.2.2 Multiple-Qubit Gates

We now describe gates which act on a register with two or more qubits. As alluded to in the first section of this chapter, some of these gates representable as Kronecker products of single-qubit gates, while others are not. We will address this more in the next subsection.

As many of these gates are represented by permutation matrices, we often make use of the following theorem.

Theorem 2.1. *All permutation matrices are unitary.*

The SWAP gate acts on a two-qubit quantum register. The effect of this gate on the state is a swapping of the coefficients of $|01\rangle$ and $|10\rangle$. Thinking of the register informally as two qubits, the basis states of the two qubits are swapped.

The SWAP gate completes the transformation on the states in the computational basis denoted by

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |10\rangle \\ |10\rangle &\mapsto |01\rangle \\ |11\rangle &\mapsto |11\rangle . \end{aligned}$$

The matrix to represent this transformation is

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.12}$$

Because this matrix is a permutation matrix, it is unitary by Theorem 2.1, and SWAP is a quantum gate.

Figure 2.3: SWAP Gate

In a wire diagram, the SWAP gate is represented as in Figure 2.3.

The controlled NOT gate acts on a two-qubit quantum register. At the level of basis states, the second qubit is subjected to the NOT gate if and only if the first qubit is currently $|1\rangle$. In terms of the computational basis, the effect on each basis state is

$$\begin{aligned} |00\rangle &\mapsto |00\rangle \\ |01\rangle &\mapsto |01\rangle \\ |10\rangle &\mapsto |11\rangle \\ |11\rangle &\mapsto |10\rangle . \end{aligned}$$

The matrix to perform this operation is

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.13}$$

The CNOT matrix is unitary by Theorem 2.1. It is interesting to note that this gate makes a change only if the first qubit has a specified value. Gates of this form are referred to as *controlled operations* [15].

In a wire diagram, the CNOT gate is represented as seen in Figure 2.4. The symbol on the first wire denotes this qubit as a *control qubit*, and the XOR symbol on the second wire indicates the *target qubit* which is replaced by the mod 2 sum of the two values indexing the basis state. In a controlled operation, the target qubit is affected if the control qubit is $|1\rangle$. In controlled gates other than controlled-NOT, the XOR symbol is replaced by the appropriate gate.

Figure 2.4: CNOT Gate

The Toffoli gate is also known as the CCNOT gate, an abbreviation of the controlled controlled-NOT gate. This gate acts on a three-qubit quantum register. At the level of basis states, the third qubit is subjected to a NOT gate if and only if the first two qubits are both $|1\rangle$.

The transformation on the basis states can be represented as

$$\begin{aligned} |110\rangle &\mapsto |111\rangle \\ |111\rangle &\mapsto |110\rangle \\ \text{and } |b_1b_2b_3\rangle &\mapsto |b_1b_2b_3\rangle \text{ otherwise.} \end{aligned}$$

Six of the basis states are represented in the last line of the above transformation, as the Toffoli gate leaves them unchanged.

The matrix to represent this gate is

$$TOFFOLI = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \tag{2.14}$$

where, as we will henceforth do, we order the basis states in lexicographic order: 000, 001, 010, 011, 100, 101, 110, 111.

In a wire diagram, the Toffoli gate is represented as seen in Figure 2.5. The first and second qubits are both control qubits, meaning that both must be $|1\rangle$ for the third qubit to be affected by the NOT gate.

Figure 2.5: Toffoli Gate

As TOFFOLI is a permutation matrix, it is unitary, confirming that it corresponds to a quantum gate.

The Fredkin gate is a controlled SWAP gate, also called the CSWAP gate. This gate acts on a three-qubit quantum register. At the level of basis states, the second and third qubits are subjected to a SWAP gate if and only if the first qubit is $|1\rangle$.

The gate may be represented using the transformation

$$\begin{aligned} |101\rangle &\mapsto |110\rangle \\ |110\rangle &\mapsto |101\rangle \\ \text{and } |b_1b_2b_3\rangle &\mapsto |b_1b_2b_3\rangle \text{ otherwise.} \end{aligned}$$

The remaining six basis states are not affected by the Fredkin gate. Informally, if the first qubit is $|0\rangle$, then no swap takes place. If the second and third qubits are equal, then the swap has no effect.

The matrix to represent this transformation is

$$CSWAP = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.15}$$

In Figure 2.6, we see the Fredkin gate. The first qubit is the control qubit, and the remaining two are target qubits. The target qubits may or may not be affected by the SWAP gate, depending on the control qubit.

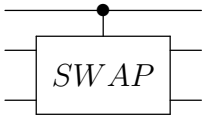


Figure 2.6: Fredkin Gate

We can define a general category of controlled operations, with arbitrarily many control and target qubits [15].

Definition 2.6. For an $(n+k)$-qubit quantum register, and a k-qubit unitary operator U, the controlled operation $C^n(U)$ is defined as

$$C^n(U)\left|b_1 b_2 \ldots b_n\right\rangle\left|\psi\right\rangle = \left|b_1 b_2 \ldots b_n\right\rangle U^{b_1 b_2 \ldots b_n}\left|\psi\right\rangle. \tag{2.16}$$

The unitary matrix U is applied to the state $|\psi\rangle$ in the second register consisting of the last k qubits if the product $b_1 b_2 \ldots b_n$ of all indices in the first register is 1. Otherwise, the basis state is left unchanged.

Now we have an arsenal of gates, but is this enough for any quantum computation? We wish to discuss universal quantum gates, but let us explain first why everything we have shown so far is reversible.

2.2.3 Reversible Computation

The postulates of quantum mechanics require that all operations, aside from measurements, are carried out by unitary transformations. (Cf. Postulate 2 in Section 2.2.2 of [15].) In particular, all operations carried out by unitary transformations are reversible. If we wish to emulate classical logic circuits, we must restrict our attention to reversible logic. The question then arises, what sorts of classical computation can be done by reversible logic circuits?

Fortunately, this question was answered long ago for different reasons. In 1961, Rolf Landauer tied heat generation to information loss in classical computing. Landauer's work led others to study reversible circuits. From this work, we know that every classical logic circuit may be emulated by a reversible circuit with a negligible loss of efficiency. We will discuss this further in Section 2.4.

The output of a reversible logic gate is a permutation of the inputs. As a linear transformation, such a reversible operation is represented by a permutation matrix. Every permutation matrix is a unitary matrix, and so at least in principle, can be implemented as the evolution of a quantum system. This is why all of our quantum gates are unitary, and also why we have not compromised our flexibility in emulating classical computation with this restriction. It remains to decide which quantum gates are the best building blocks.

2.2.4 Universal Quantum Gates

As we did for classical gates, we discuss universal sets of quantum gates. One approach is to express an arbitrary unitary operator as a composition of arbitrary single qubit gates and CNOT gates.

Theorem 2.2. *Any unitary operation on n qubits may be implemented using CNOT gates on pairs of qubits and single qubit gates [15].*

One should be reminded that there are physical issues around constructing an arbitrary single qubit gate with perfect precision, as discussed in [15]. For example, a time-dependent operator that acts as $\begin{bmatrix} 1 & 0 \\ 0 & e^{it} \end{bmatrix}$ at time t may be difficult to apply up to exactly a desired time t_0. Another complexity arises when one's implementation only allows CNOT gates on spatially proximal qubits.

There is another universal set of quantum gates containing only four gates, instead of arbitrary single qubit gates, to avoid the challenges of implementing general single qubit gates to arbitrary precision. This is achieved by establishing that we can approximate any unitary operation using these gates, and we can bound the number of gates needed in terms of the number in the original circuit and the acceptable error.

To consider approximations of operations, a measure of error is necessary. The error may be thought of as the difference between applying the original operation and applying the approximation, in the worst case.

Definition 2.7. The *error* $E(U_a, U_b)$ between $N \times N$ unitary matrices U_a and U_b is defined as

$$E(U_a, U_b) = \max_{|\psi\rangle \in V} \| (U_a - U_b) |\psi\rangle \|, \tag{2.17}$$

where V is an N-state quantum register and $\| |\psi\rangle \| = 1$.

We now state a result derived from [15, Sec. 4.5] on approximating any unitary operator.

Theorem 2.3. *Given a circuit U_a comprised of m gates which are either CNOT or correspond to matrices from $\mathcal{U}(2)$, we may create a unitary operator U_b such that $E(U_a, U_b) < \epsilon$,*

for any $\epsilon > 0$, where all gates in U_b come from the set $\{CNOT, H, S, T\}$. Furthermore, U_b can be expressed using $\mathcal{O}\left((m \log^c(m/\epsilon))\right)$ gates, where c is a constant and $c \approx 2$.

From this result, the Hadamard, phase, $\pi/8$, and CNOT gates can be employed as building blocks to approximate any unitary operator with arbitrary precision.[6] The number of gates needed increases, but the bound is considered by [15][Sec. 4.5] to be "acceptable for virtually all implementations." We use the results from this section to justify the use of various unitary matrices in the algorithms explored in later chapters. For example, any time our algorithms require computations which may be completed on a classical computer, we do not hesitate to encode the corresponding gates as a "black box."

2.3 Initial State of a Register

The quantum algorithms we consider will begin with all qubits initialized to the same state, represented by $|0\rangle$ in the computational basis. Preparation of qubits in this state is considered a requirement for a physical implementation of a quantum computer.

Excluding measurements, transformations on the state of the qubits after initialization must be reversible, as dictated by the postulates of quantum mechanics, but the initial state of the qubits is a physical consideration. For a detailed consideration of the requirements for a physical quantum computer, see [6].

2.4 Employing Classical Algorithms

In our discussion of quantum gates, we started with a brief overview of classical gates. We now briefly consider classical reversible computing, to continue the discussion started in Section 2.2.3.

There is a requirement for each reversible logic gate that every input map to a unique output, so the AND, OR, and NAND gates cannot be used. The NOT gate, however, is clearly reversible. The input 0 is mapped to the output 1, and vice versa. A second NOT gate reverses the effect of the first.

Other classical reversible gates include SWAP, CNOT, TOFFOLI, and FREDKIN. It is important to note that the TOFFOLI gate is a universal gate for classical reversible computation, as is the FREDKIN gate [25][Sec. 2.1.5]. We use this assertion to establish that any computation which may be completed using a classical computer may also be completed using a quantum computer, by using the TOFFOLI or FREDKIN gates.

But there remains the question of efficiency. If a certain computation could be completed in polynomially many irreversible gates, it would make reversible computation impractical if exponentially many gates were needed for the same task.

Fortunately, we know this is not the case. Using the NOT, CNOT, and TOFFOLI gates, we can recreate any operation possible through irreversible gates with a polynomial increase

[6] Given the $\pi/8$ gate, the phase gate is not needed for universality, but is needed for fault-tolerance, as explained in [15][Sec. 4.5]

in the number of gates and space. We quote a result directly from [25][Sec. 2.2] on this topic. Note that ancillae refers to "workspace" qubits, whose count is not included in either the size of the input or the size of the output.

Theorem 2.4. *Any irreversible computation (in the synchronous form) having t gates, depth d, and width w, can be simulated by a reversible circuit having $\mathcal{O}(t^{2.58})$ gates, and at most $(w+1)\log d + \mathcal{O}(1)$ ancillae.*

We take a moment to explain the terminology used in the theorem. The circuit is considered as an acyclic directed graph whose vertices are partitioned into a series of levels. Vertices at the top level, level 0, have in-degree zero and are labeled by input variables. All other vertices are labeled by gates from a universal set, together with an identity gate, where the identity gate is a way of representing the null operator NOP. This allows the statement that all gates at level m rely on results from level $m-1$. The depth denotes the number of levels, and the maximum number of non-identity gates in a single level is the width. The term "synchronous" is defined by [25][Sec. 2.2] as the requirement that for any gate, all inputs to the gate have the same path length from the start of the computation. The path length is easily seen to be the number of gates along any path from an input node to the current one. With the idea of identity gates, this is the equal to the level number in which the gate resides.

Implementations for classical computations are often given using irreversible gates. From this result, it is understood that a reversible version may be constructed with at most a polynomial increase in the number of gates and necessary workspace.

Given a reversible classical gate construction, we can recreate the same circuit for a quantum computer. The NOT, CNOT, and TOFFOLI gates, a universal set of gates for classical reversible computation, have equivalent quantum gates. The same circuit can be constructed using the quantum NOT, CNOT, and TOFFOLI gates, and the original classical computation will be carried out when we restrict all inputs to basis states in the quantum circuit.

Implementing classical computations with polynomial efficiency allows for the construction of "oracle" gates, which we will see in Grover's search algorithm. We will also see an oracle gate in this chapter, with the Deutsch-Jozsa algorithm. The certainty that quantum computers can recreate classical algorithms justifies these "oracle" or "black box" subroutines, and we will see they are used in algorithms which are more efficient than the optimal classical solutions.

The linearity of unitary operators allows us to apply such a "black box" to a superposition of exponentially many basis states at the same cost of applying the corresponding algorithm to a single input. One should not immediately infer, however, that this affords us an exponential speedup in every case: the resulting state of the quantum register must be measured in order for us to learn any classical information and as we have seen, such measurement loses information.

2.5 The Deutsch-Jozsa Algorithm

There is strong interest in quantum information and quantum computing today due to the development of algorithms where the best known classical solutions have poor efficiency. We will explore examples with implications on modern cryptography in Chapters 3 and 4, but now we consider a simple example.

The first quantum algorithm we consider is the Deutsch-Jozsa algorithm, an extension of the Deutsch algorithm[7]. This algorithm does not solve a problem with particularly useful applications, but it is notable as an example of a problem that a quantum computer can solve in fewer steps than any classical computer.

Consider a function $f : \{0,1\}^n \mapsto \{0,1\}$. Suppose we are promised, in advance, that function f has one of two very special forms. Given that f will either map every input to 0, map every input to 1 (f is *constant*), or map exactly half of the inputs to 0 and the other half to 1 (f is *balanced*), the task is to determine whether f is constant or balanced.

For example, if $f(x)$ is the mod 2 sum of the bits in x, then one easily checks that f is balanced. But we only assume that f is given by a black box.

Using a classical computer, and given a black box implementation of f, it would take at least $2^{n-1}+1$ evaluations of f to determine with absolute certainty whether it is constant or balanced. Notice that this is one more than half the number of possible input values. Consider choosing one n-bit binary string at a time, computing f with the black box implementation, and noting the result. If at any point, two strings produce a different result, there is no reason to continue as f is known to be balanced. Otherwise, for the first 2^{n-1} input choices, it is possible (albeit unlikely) that 2^{n-1} of the remaining input choices produce a different result. Once more than half of the input values have been evaluated, it is possible to say with certainty that f is constant.

On the other hand, given a quantum computer and a gate implementation of f, an *oracle*, only one use of the oracle is required. We first explain how f is implemented in a unitary transformation U_f, and then give a description of the Deutsch-Jozsa algorithm.

As part of the problem statement, we are given the function f which maps every n-bit binary string to either 0 or 1. Our first task is to implement this in a reversible way, consistent with the laws of quantum mechanics. We use the approach set forth by [25][Sec. 1.7] to arrive at a unitary transformation.

As we addressed the implementation of any classical computation using a quantum computer in the previous section, we assume the ability to arrive at the value $f(x)$ with knowledge of an input x. We focus on the issue of embedding this action in a unitary transformation.

To create a unitary transformation, we must have a bijection between the states before and after the transformation. Thinking of this as a "reversibility" requirement, it is reasonable to want the n-bit input string represented in the initial state to also be included in the state after the transformation takes place. However, the transformation must include the output of the function f. The next reasonable step is to include an ancillary qubit, creating extra space to store the result of f. One could imagine the transformation

[7]This algorithm happens to appear in Figure 2.1.

$|x\rangle\,|b\rangle \mapsto |x\rangle\,|f(x)\rangle$, where $x \in \mathbb{Z}_2^n$, regardless of b.

There is one problem with the above suggestion. We must map every possible basis state to a unique result, and there is no way to distinguish between the two initial states $|x\rangle\,|0\rangle$ and $|x\rangle\,|1\rangle$. To create a unitary transformation, these two states must be mapped to different resulting states. One simple way to achieve this is with the transformation $|x\rangle\,|y\rangle \mapsto |x\rangle\,|y \oplus f(x)\rangle$, where $y \in \mathbb{Z}_2$ and $\oplus$ represents the exclusive-or (XOR) operation. With this modification, regardless of the value of $f(x)$, $0 \oplus f(x)$ is not equal to $1 \oplus f(x)$. We now have a unitary transformation defined by its action on basis states:

$$U_f : |x\rangle\,|y\rangle \mapsto |x\rangle\,|y \oplus f(x)\rangle\,. \tag{2.18}$$

Now that we have established the existence of a unitary U_f, we know this may be implemented using either of the universal sets of quantum gates mentioned in Section 2.2. We proceed with the method of determining whether f is constant or balanced. These steps are presented as a wire diagram in Figure 2.7.

Deutsch-Jozsa Algorithm

Input: A function $f : \{0,1\}^n \mapsto \{0,1\}$ which is constant or balanced.

Output: A 0 or a positive value to indicate whether f is constant or balanced, respectively.

Step 1: Begin with an $n+1$-qubit quantum register, with all qubits initialized to $|0\rangle$. The state of the register may be represented as

$$\bigotimes_{i=1}^{n+1} |0\rangle\,. \tag{2.19}$$

Step 2: Perform a NOT operation on the last qubit. The state of the register is now

$$\left(\bigotimes_{i=1}^{n} |0\rangle\right) \otimes |1\rangle \tag{2.20}$$

Step 3: Perform a Hadamard gate operation on each qubit. Transforming the state from the previous step using $H^{\otimes(n+1)}$, or the Kronecker product of $n+1$ Hadamard matrices, the new state is

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{Z}_2^n} |x\rangle \otimes (|0\rangle - |1\rangle)\,. \tag{2.21}$$

Step 4: Use the gate U_f to complete the transformation $|x\rangle\,|y\rangle \mapsto |x\rangle\,|y \oplus f(x)\rangle$, as defined above. Applying this operator to the superposition (2.21) yields

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{Z}_2^n} (-1)^{f(x)}\,|x\rangle \otimes (|0\rangle - |1\rangle)\,, \tag{2.22}$$

since $f(x) = 1$ means $|x\rangle \otimes (|0\rangle - |1\rangle)$ is mapped to $|x\rangle \otimes (|1\rangle - |0\rangle)$.

STEP 5: Perform a Hadamard gate operation on the first n qubits. We use the effect of this operation on a single basis state $|x\rangle$, which is explained after Step 6. We find that $H^{\otimes n}$ acts on $|x\rangle$ to create the mapping

$$|x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{z \in \mathbb{Z}_2^n} (-1)^{x \cdot z} |z\rangle , \tag{2.23}$$

where $x \cdot z = \sum_{j=1}^{n} x_j z_j$. Our actual transformation on the $(n+1)$-qubit register is $H^{\otimes n} \otimes I$ so that the last qubit is unaffected.

Applying this transformation to every $|x\rangle$ in (2.22), we find the resulting state is

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{x \in \mathbb{Z}_2^n} (-1)^{f(x)} \left(\frac{1}{\sqrt{2^n}} \sum_{z \in \mathbb{Z}_2^n} (-1)^{x \cdot z} |z\rangle \right) \otimes (|0\rangle - |1\rangle) . \tag{2.24}$$

After some rearranging, we can express the state as

$$\frac{1}{2^n \sqrt{2}} \sum_{z \in \mathbb{Z}_2^n} \left(\sum_{x \in \mathbb{Z}_2^n} (-1)^{f(x) + x \cdot z} \right) |z\rangle \otimes (|0\rangle - |1\rangle) . \tag{2.25}$$

The state now involves two sums over all elements of $\mathbb{Z}_2^n$. Ignoring the Kronecker product with the last qubit, the nested sum has 2^{2n} terms. There are 2^n terms contributing to the coefficient for each basis state $|z\rangle$, one for each value of $x \in \mathbb{Z}_2^n$. In the next step, we consider one of these coefficients.

STEP 6: Measure the first n qubits. We measure the first n qubits with $\mathcal{M} = \{S_j : 0 \leq j < 2^n\}$. The subspaces S_j each correspond to the span of a vector from the computational basis. We can write $S_j = \text{span}\{|j\rangle \otimes |0\rangle , |j\rangle \otimes |1\rangle\}$.

It turns out we only need to consider one possible output. Consider the coefficient of $|0\rangle \otimes (\frac{|0\rangle - |1\rangle}{\sqrt{2}})$ which is given by

$$2^{-n} \sum_{x \in \mathbb{Z}_2^n} (-1)^{f(x)}. \tag{2.26}$$

If $f(x)$ is balanced, then the above sum includes equally many 1's and -1's. In this case, the probability amplitude is zero, and the measurement returned can never be zero by our definition of a measurement.

On the other hand, if $f(x)$ is constant, then all terms in the sum are equal, and the above sum has absolute value 1. As a direct consequence, the state belongs to S_0, and the measurement returned can only be 0.

Based on the result of this measurement, we have determined the behavior of $f(x)$. This algorithm only requires one application of the gate U_f, our implementation of $f(x)$, which, from our earlier discussion, has the same computational cost as computing $f(x)$ for

a single input. By contrast, the classical version requires $2^{n-1}+1$ calls in the worst case, a number which grows exponentially in n. The Deutsch algorithm, essentially the case of this algorithm where $n = 1$, was one of the early quantum algorithms, and presented a task where a quantum computer could outperform the optimal classical alternative.

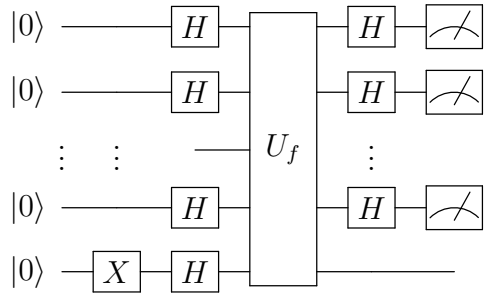


Figure 2.7: A wire diagram for the Deutsch-Jozsa algorithm.

We look in detail at the operation of applying a gate constructed through the Kronecker product of n Hadamard gates to an n-qubit quantum register in a computational basis state $|x\rangle$, using the indexing $x \in \mathbb{Z}_2^n$.

We start by considering the product of the n Hadamard gates with the state $|x\rangle$. Assume $|x\rangle = |x_1\rangle \otimes \cdots \otimes |x_n\rangle$ belongs to the computational basis. From the basic properties of the Kronecker product, we can distribute to find

$$H^{\otimes n} |x\rangle = \left(\bigotimes_{j=1}^{n} H \right) |x\rangle = \bigotimes_{j=1}^{n} \left(H |x_j\rangle \right). \tag{2.27}$$

Next, we evaluate the products formed by each Hadamard gate acting on one of the qubits in the Kronecker product $|x\rangle$. Depending on whether the qubit is a $|1\rangle$ or $|0\rangle$, the result of the Hadamard gate changes as seen in

$$\bigotimes_{j=1}^{n} \left(H |x_j\rangle \right) = \frac{1}{\sqrt{2^n}} \left(\bigotimes_{j=1}^{n} (|0\rangle + (-1)^{x_j} |1\rangle) \right). \tag{2.28}$$

We express the state using a sum of Kronecker products, instead of a Kronecker product of sums. The result involves summing over all possible basis states $z \in \mathbb{Z}_2^n$. We find the coefficient for each z by, informally, considering one qubit of z at a time, with the corresponding qubit of x. If the qubit of $|z\rangle$ is $|0\rangle$, then the coefficient is multiplied by one. If the qubit of $|z\rangle$ is a $|1\rangle$, then the effect on the coefficient depends on the corresponding qubit in $|x\rangle$. The state may be represented as

$$\frac{1}{\sqrt{2^n}} \sum_{z \in \mathbb{Z}_2^n} \left(\bigotimes_{j=1}^{n} (-1)^{x_j z_j} |z_j\rangle \right). \tag{2.29}$$

This may be represented more concisely as

$$\frac{1}{\sqrt{2^n}} \sum_{z \in \mathbb{Z}_2^n} (-1)^{x \cdot z} |z\rangle , \tag{2.30}$$

where

$$x \cdot z = \sum_{j=1}^{n} x_j z_j. \tag{2.31}$$

We have arrived at the transformation used in Step 5 of the Deutsch-Jozsa algorithm, where Hadamard gates are applied to the first n qubits, after the gate U_f. The Hadamard gates are applied to an equal superposition of states, where the result is not as evident as the Hadamard gates applied to the single state $|0\rangle$ previously in the algorithm.

2.6 The Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is a key part of the algorithms we discuss in Chapter 4, for factoring and the discrete log problem. The purpose of this section is to give a construction of the QFT from gates discussed earlier in this chapter.

The transformation on the basis state $|j\rangle$ of an N-state quantum register as a result of the QFT is

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N} |k\rangle . \tag{2.32}$$

The goal of this section is to give an efficient implementation in the special case where $N = 2^n$ for some integer n.

By restating the result of applying the QFT, we can see a connection between the statement and a gate construction. This alternate form is called the *product representation* by [15][Sec. 5.1], and the same construction is present in [25][Sec. 3.4.4].

For clarity, we define $|Q\rangle$ to be the result of applying the QFT to $|j\rangle$, so that

$$|Q\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N} |k\rangle . \tag{2.33}$$

Our first step is to express k as a sum of powers of 2. We choose to write the binary representation[8] of k as $k_1 k_2 \ldots k_n$, defined by

$$k = \sum_{m=1}^{n} k_m 2^{n-m} = k_1 2^{n-1} + k_2 2^{n-2} + \cdots + k_n 2^0.$$

From the previous equation, we can write

$$\frac{k}{N} = \frac{1}{N} \sum_{m=1}^{n} k_m 2^{n-m} = \sum_{m=1}^{n} k_m 2^{-m},$$

[8]Note the different choice of indices for the binary digits of k in this section.

so that

$$\exp(2\pi ijk/N) = \exp\left(\frac{2\pi ij}{N}\sum_{m=1}^{n} k_m 2^{n-m}\right)$$
$$= \prod_{m=1}^{n} \exp\left(2\pi ijk_m 2^{-m}\right).$$

Now, express the sum over all basis states $k = 0, \ldots, N-1$ as a sum - over all m - of sums over $k_m \in \{0, 1\}$. We write

$$|Q\rangle = \frac{1}{\sqrt{N}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \exp(2\pi ij \sum_{m=1}^{n} k_m 2^{-m}) \, |k_1 \ldots k_n\rangle$$
$$= \frac{1}{\sqrt{N}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \prod_{m=1}^{n} \exp(2\pi ijk_m 2^{-m}) \, |k_1 \ldots k_n\rangle .$$

Since $|k_1 \ldots k_n\rangle = |k_1\rangle \otimes \cdots \otimes |k_n\rangle$, we can express $|k\rangle$ using an n-fold Kronecker product. By using this instead of the product in the previous equation, we find

$$|Q\rangle = \frac{1}{\sqrt{N}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{m=1}^{n} \exp(2\pi ijk_m 2^{-m}) \, |k_m\rangle .$$

Because the Kronecker product distributes over addition, we can express $|Q\rangle$ as

$$|Q\rangle = \frac{1}{\sqrt{N}} \left(\sum_{k_1=0}^{1} \exp(2\pi ijk_1 2^{-1}) \, |k_1\rangle\right) \bigotimes \left(\sum_{k_2=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{m=2}^{n} \exp(2\pi ijk_m 2^{-m}) \, |k_m\rangle\right)$$
$$= \frac{1}{\sqrt{N}} \left(\sum_{k_1=0}^{1} \exp(2\pi ijk_1 2^{-1}) \, |k_1\rangle\right) \bigotimes \cdots \bigotimes \left(\sum_{k_n=0}^{1} \exp(2\pi ijk_n 2^{-n}) \, |k_n\rangle\right)$$
$$= \frac{1}{\sqrt{N}} \bigotimes_{m=1}^{n} \sum_{k_m=0}^{1} \exp(2\pi ijk_m 2^{-m}) \, |k_m\rangle .$$

In this way, we have regrouped terms to turn a sum of Kronecker products into a Kronecker product of sums. We now express each inner sum as two terms, instead of using a summation, as it is more illustrative. At this point,

$$|Q\rangle = \frac{1}{\sqrt{N}} \bigotimes_{m=1}^{n} \left(|0\rangle + \exp(2\pi ij2^{-m}) \, |1\rangle\right). \tag{2.34}$$

Next, we expand j as a sum of powers of 2. The expression $j2^{-m}$ becomes $\sum_{l=1}^{n} j_l 2^{n-l} 2^{-m}$. Notice that only the last m terms in this sum will be negative powers of 2. This is relevant as $\exp(2\pi iL)$, for any nonnegative integer L, is equal to 1. We therefore have

$$\exp(2\pi ij2^{-m}) = \exp(2\pi i(j_1 2^{n-m-1} + \cdots + j_n 2^{-m})) = \exp(2\pi i(j_{n-m+1} 2^{-1} + \cdots + j_n 2^{-m})). \tag{2.35}$$

There is a more convenient notation to use for these sums of negative powers of 2, the binary fraction.

Definition 2.8. The *binary fraction* representation of the sum $a_1 2^{-1} + a_2 2^{-2} + \cdots + a_l 2^{-l}$ is $0.a_1 a_2 \ldots a_l$.

Making use of binary fractions, we can write $|Q\rangle$ as

$$|Q\rangle = \frac{1}{\sqrt{N}} \left(|0\rangle + \exp(2\pi i 0.j_n)\,|1\rangle\right) \otimes \left(|0\rangle + \exp(2\pi i 0.j_{n-1} j_n)\,|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + \exp(2\pi i 0.j_1 \ldots j_n)\,|1\rangle\right). \tag{2.36}$$

From this representation, we can arrive at a gate construction for the QFT. In this product, the term $|0\rangle + \exp(2\pi i 0.j_n)\,|1\rangle$ depends only on the state of the n-th qubit of the basis state $|j\rangle$. The second term depends on the state of the n-th and $(n-1)$-st qubits. This continues, until the last term $|0\rangle + \exp(2\pi i 0.j_1 \ldots j_n)\,|1\rangle$ depends on the state of every qubit in the register. In this construction, we will use many controlled-phase gates.

The m-th qubit in the register is the control qubit for $m-1$ controlled phase gates. It is the target qubit in $n-m$ such gates, as well as a Hadamard gate. The instances where it is a control qubit will come before the instances where it is a target qubit, as we need the state of the qubit before it is changed.

The only term in the product representation of $|Q\rangle$ to depend on $|j_1\rangle$ is the last term. We begin the gate construction with a Hadamard gate, and all controlled-phase gates where $|j_1\rangle$ is the target qubit. The only terms to depend on $|j_2\rangle$ are the last and second to last. As all gates where $|j_1\rangle$ is the target qubit have been added, we next add all gates which act on the state of $|j_2\rangle$. This pattern continues; the terms which depend on the initial state of $|j_m\rangle$ are the m-th through n-th terms. Based on this, we start with the n-th term and proceed in reverse order to the first term.

We consider the action on the m-th qubit of a Hadamard gate followed by $n-m$ controlled phase gates, depending on qubits $m+1$ to n. The term in the product representation of $|Q\rangle$ is the $n-m+1$-th term, and consists of

$$|0\rangle + \exp(2\pi i 0.j_m \ldots j_n)\,|1\rangle.$$

We claim that the product $HR_2 \cdots R_{n-m+1}$, when applied to the m-th qubit, will result in this term.

Working as usual in terms of basis states, the result of applying a Hadamard gate to the m-th qubit $|j_m\rangle$ is $\frac{1}{\sqrt{2}}\left(|0\rangle + \exp(2\pi i j_m/2)\,|1\rangle\right)$, where $\exp(2\pi i j_m/2) = \exp(2\pi i 0.j_m)$. The coefficient of $|1\rangle$ depends on j_m. The unitary matrix representing the controlled-phase gate applied to $|j_m\rangle$ with control qubit $|j_t\rangle$, $t \geq m+1$, is

$$R_{t-m+1} \equiv \begin{bmatrix} 1 & 0 \\ 0 & \exp(2\pi i j_t/2^{t-m+1}) \end{bmatrix}.$$

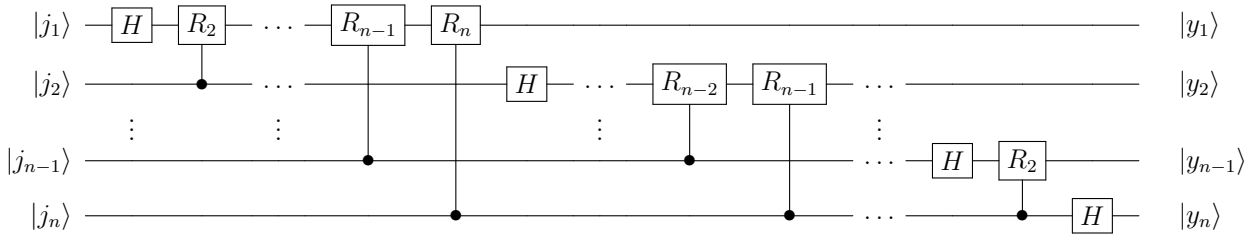


Figure 2.8: A wire diagram illustrating the construction of the QFT.

Controlled-phase gates are applied to $|j_m\rangle$ with control qubit $|j_t\rangle$ for t in the range $[m+1, n]$, so we need the product, including the Hadamard gate,

$$\frac{1}{\sqrt{2}} \prod_{t=m}^{n} \begin{bmatrix} 1 & 0 \\ 0 & \exp(2\pi i j_t / 2^{t-m+1}) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \exp(2\pi i (j_m 2^{-1} + \cdots + j_n 2^{-(n-m+1)})) \end{bmatrix} \tag{2.37}$$

$$= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 0 & \exp(2\pi i 0.j_m \ldots j_n) \end{bmatrix}. \tag{2.38}$$

The state of the qubit $|j_m\rangle$ after this matrix is applied is now

$$\frac{1}{\sqrt{2}} \left(|0\rangle + \exp(2\pi i 0.j_m \ldots j_n) \right),$$

which is the m-th term in the product representation of $|Q\rangle$. This proves our claim.

Overall, the state $|j\rangle$ is mapped to

$$\prod_{m=1}^{n} \frac{1}{\sqrt{2}} \left(|0\rangle + \exp(2\pi i 0.j_m \ldots j_n) |1\rangle \right) =$$
$$\frac{1}{\sqrt{N}} \left[\left(|0\rangle + \exp(2\pi i 0.j_1 \ldots j_n) |1\rangle \right) \cdots \left(|0\rangle + \exp(2\pi i 0.j_n) |1\rangle \right) \right],$$

which is equal to $|Q\rangle$ as expressed in (2.34) above.

We show the sequence of gates, $HR_2 \ldots R_n$ applied to the first qubit, followed by $HR_2 \ldots R_{n-1}$ applied to the second qubit, and so on, up to H applied to the n-th qubit in a wire diagram in Figure 2.8. To make the figure more readable, the labels on the right side are abbreviated to $|y_1\rangle$ through $|y_n\rangle$, from top to bottom, where

$$|y_s\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + \exp(2\pi i 0.j_s \ldots j_n) |1\rangle \right).$$

These labels should go from $|y_n\rangle$ to $|y_1\rangle$, from the top wire to the bottom, to match the ordering in the product representation of $|Q\rangle$. After the last Hadamard gate, SWAP gates are used to correct this reversal. While the diagram only describes the action of this operator on basis states, its action on superpositions is inferred by linearity.

The number of gates needed to construct the QFT on n qubits is proportional to n^2. Taking a sum over the number of Hadamard and controlled-phase gates needed, we find

$$\sum_{m=1}^{n} m = \frac{n(n+1)}{2}. \tag{2.39}$$

We need to swap the order of the results after these gates. To reorder n qubits, using SWAP gates on adjacent qubits, requires at most $\sum_{m=1}^{n} m - 1 = \frac{(n-1)(n)}{2}$ gates.

We give a brief explanation for this. If we wish to swap the state of the qubit in the m-th position in the register to the last qubit in the register, this requires at most $n-1$ swap gates. Swapping the state of some qubit with the second to last qubit in the register requires at most $n-2$ swap gates. This continues down to the first qubit, at which point no further swap gates are needed.

In the QFT construction as a whole, the total number of single-qubit and two-qubit controlled operations is $\Theta(n^2)$. (Cf. [15][Sec. 5.1].)

It remains to establish that the QFT is unitary. We have constructed the QFT from a sequence of 1- and 2-qubit operations known to be unitary. Each of these is extended to an $N \times N$ matrix via tensor products, with identity matrices of appropriate size. Therefore, the result is also unitary. This gate construction can also be represented as a product of unitary matrices, and the product of unitary matrices is likewise unitary.

Chapter 3

Grover's Database Search

In this chapter, we explore a quantum algorithm for searching an unordered list of numbers. We assume we are given a function with which to test each number. This function will return '0' for all but one of the numbers, and in this case it returns '1'. We assume the function is given as an oracle; we have only black box access to it. With a classical computer, the only way to complete this task is to try evaluating the function for each number, until the result is found. As the numbers are unordered, it is expected that we will need to try half of the numbers before the specific number is identified. We will need to evaluate the function $\mathcal{O}(N)$ times in the average case, if there are N numbers in total.

However, using this quantum algorithm, we only need to evaluate a quantum implementation of this function $\mathcal{O}\left(\sqrt{N}\right)$ times. With a small quantity of numbers to search, this speedup might seem unremarkable. However, consider using this method to search a table with 2^{100} possible keys for a symmetric encryption scheme in time proportional to 2^{50}. This situation occurs in a *known-plaintext attack*: an adversary has a plaintext message and the corresponding ciphertext. (Cf. [14][Sec. 1.13.1].) Testing every possible key for a given symmetric encryption scheme, to see whether the plaintext produces the ciphertext, is partially made difficult by the size of the key space. With an algorithm to search the key space in time proportional to the square root of the number of keys, however, these attacks require less resources to use, potentially making the difference between an intractable search problem and a tractable one.

3.0.1 Problem Statement

We first give a formal problem statement and our assumptions.

We have a set of N values. We assume N is a power of 2, so that $N = 2^n$. In the case that N is not a power of 2, the set of values can be padded with extra entries until the next power of 2 is reached. We designate a bijection between the set of given values, and the numbers $0, \ldots, N-1$, for the purposes of indexing basis states using the computational basis.

Let the function $f : \{0,1\}^n \to \{0,1\}$. There is a unique value $x^* \in \{0,1\}^n$ for which $f(x^*) = 1$, and for $x \in \{0,1\} \backslash x^*$, $f(x) = 0$. After analyzing this case, where the proverbial

"needle" in the haystack is unique, we consider a case with multiple needles, where the number has a specific upper bound.

3.1 Grover's Algorithm

The key ingredients to Grover's algorithm are an oracle gate and a diffusion gate. In this section, we describe these in turn before giving the algorithm itself in Section 3.1.3.

3.1.1 The Oracle Gate

We introduced a strategy for implementing classical algorithms in Section 2.4. Further, in Section 2.5, we saw how to ensure that the implementation of the algorithm is unitary.

Recall that for a function $f : \{0,1\}^n \to \{0,1\}$, we can construct a unitary transformation of the form

$$U_f : |x\rangle |b\rangle \mapsto |x\rangle |b \oplus f(x)\rangle , \tag{3.1}$$

for $x \in \{0,1\}^n$ and $b \in \{0,1\}$.

From this unitary transformation, we construct an equivalent version making use of an ancillary qubit, so that we may consider the input and output to be n qubits with one qubit of workspace.

Our goal is to use U_f to construct a transformation

$$S_f |x\rangle = \begin{cases} -|x\rangle & \text{if } x = x^* \\ |x\rangle & \text{otherwise} \end{cases} . \tag{3.2}$$

We can say that $S_f |x\rangle = (-1)^{f(x)} |x\rangle$. We construct this transformation by using an ancillary qubit in the state $\frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)$, the result of applying a NOT gate and a Hadamard gate to the state $|0\rangle$.

We now apply the transformation given by U_f to the state $|\psi\rangle = |x\rangle \otimes \left(\frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \right) = \frac{1}{\sqrt{2}} (|x\rangle |0\rangle - |x\rangle |1\rangle)$.

If $x \in \{0,1\}^n \backslash x^*$, then $U_f |\psi\rangle = |x\rangle \otimes \left(\frac{1}{\sqrt{2}} (|0 \oplus 0\rangle - |1 \oplus 0\rangle) \right)$. We can see that $U_f |\psi\rangle = |\psi\rangle$.

In the other case, where $x = x^*$, $U_f |\psi\rangle = |x\rangle \otimes \left(\frac{1}{\sqrt{2}} (|0 \oplus 1\rangle - |1 \oplus 1\rangle) \right)$. This time, the state after applying U_f is $-|\psi\rangle$.

By analyzing these two cases for x given by the definition of S_f, we have verified that U_f applied to $|x\rangle$ with an ancillary qubit as described will complete the desired transformation S_f. As is true for U_f, the transformation S_f is its own inverse. The state of the ancillary qubit is the same before and after the gate, and the sign of the coefficient associated with $|x\rangle$ is either changed or not changed. By inspection, applying S_f a second time will yield the state before the first application.

In addition to being reversible, S_f maps every basis vector $|y\rangle$ for $y \in \{0,1\}^n$ to another vector of modulus one. From this information, we may conclude that S_f is unitary.

3.1.2 The Diffusion Gate

Let the operation D be defined as

$$D = -WS_0W, \tag{3.3}$$

where $W = H^{\otimes n}$ and

$$S_0 |x\rangle = \begin{cases} -|x\rangle & \text{if } x = 0 \\ |x\rangle & \text{otherwise} \end{cases}. \tag{3.4}$$

Note that S_0 is an implementation of S_f for a specific function f.

We first establish a construction of S_0 from familiar matrices.

Lemma 3.1. *For S_0 as defined above, $S_0 = -2 |0\rangle^{\otimes n} \langle 0|^{\otimes n} + I$, where I is the $N \times N$ identity matrix.*

Proof. We establish this lemma by considering two cases for $|x\rangle$. First, let $x = 0$. We denote this computational basis state as $|0\rangle^{\otimes n}$. By setting up the following multiplication and distributing, we find that the first case for S_0 is satisfied:

$$\begin{aligned} \left(-2 |0\rangle^{\otimes n} \langle 0|^{\otimes n} + I\right) |0\rangle^{\otimes n} &= -2 |0\rangle^{\otimes n} \langle 0|^{\otimes n} |0\rangle^{\otimes n} + |0\rangle^{\otimes n} \\ &= -2 |0\rangle^{\otimes n} \langle 0|0\rangle^{\otimes n} + |0\rangle^{\otimes n} \\ &= -2 |0\rangle^{\otimes n} + |0\rangle^{\otimes n} \\ &= -|0\rangle^{\otimes n}. \end{aligned}$$

In this simplification, we use the fact that $\langle 0|^{\otimes n} |0\rangle^{\otimes n} = \langle 0|0\rangle^{\otimes n} = \| |0\rangle^{\otimes n} \|^2 = 1$. The dot product between a vector and its conjugate transpose results in the squared modulus of the vector, which is 1 in this case.

The second case to consider is $|x\rangle$ where $x \neq 0$. By completing a multiplication analogous to the one above, we see that

$$\begin{aligned} \left(-2 |0\rangle^{\otimes n} \langle 0|^{\otimes n} + I\right) |x\rangle &= -2 |0\rangle^{\otimes n} \langle 0|^{\otimes n} |x\rangle + |x\rangle \\ &= |x\rangle. \end{aligned}$$

The transition from the first line to the second line comes from the equation $\langle 0|^{\otimes n} |x\rangle = 0$. Since $|0\rangle^{\otimes n}$ and $|x\rangle$ are distinct vectors from the same unitary basis, they are orthogonal. Therefore, their dot product is zero.

We have therefore shown $S_0 |x\rangle = \left(-2 |0\rangle^{\otimes n} \langle 0|^{\otimes n} + I\right) |x\rangle$ for every basis state $|x\rangle$.

□

We can now express D in the following two equivalent ways:

$$-WS_0W = H^{\otimes n} \left(2 |0\rangle^{\otimes n} \langle 0|^{\otimes n} - I\right) H^{\otimes n}. \tag{3.5}$$

Taking the right side of this equation, we can distribute to find

$$D = 2H^{\otimes n} |0\rangle^{\otimes n} \langle 0|^{\otimes n} H^{\otimes n} - H^{\otimes n} H^{\otimes n}. \tag{3.6}$$

Using $H^\dagger = H$, $|0\rangle^\dagger = \langle 0|$, and basic properties, we have $\langle 0|^{\otimes n} H^{\otimes n} = (H^{\otimes n} |0\rangle^{\otimes n})^\dagger$, $H^{\otimes n} |0\rangle^{\otimes n} = (H |0\rangle)^{\otimes n}$, and $H^{\otimes n} H^{\otimes n} = (HH)^{\otimes n} = I^{\otimes n}$. We use these three equations to express D as

$$D = 2(H |0\rangle)^{\otimes n} (H |0\rangle)^{\otimes n \dagger} - I. \tag{3.7}$$

To see the effect D has on the state of a qubit, we consider an arbitrary state

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle . \tag{3.8}$$

We see that

$$D |\psi\rangle = 2(H |0\rangle)^{\otimes n} (H |0\rangle)^{\otimes n \dagger} |\psi\rangle - |\psi\rangle . \tag{3.9}$$

Next, we complete several computations with the goal of expressing $D |\psi\rangle$ in a simplified form.

First, consider the expression $(H |0\rangle)^{\otimes n}$. By evaluating the matrix product, we see that

$$(H |0\rangle)^{\otimes n} = \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^{\otimes n} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}^{\otimes n}$$

by factoring out the scalar $\frac{1}{\sqrt{2}}$.

Note that the result of the Kronecker product is equal to the sum of all basis vectors in the computational basis:

$$(H |0\rangle)^{\otimes n} = \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n} |x\rangle . \tag{3.10}$$

Taking the conjugate transpose,

$$(H |0\rangle)^{\otimes n \dagger} = \frac{1}{\sqrt{N}} \left(\begin{bmatrix} 1 & 1 \end{bmatrix} \right)^{\otimes n} \tag{3.11}$$

$$= \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n} \langle x| . \tag{3.12}$$

By using the simplification in the previous equation, we rewrite the expression $(H |0\rangle)^{\otimes n \dagger} |\psi\rangle$ as

$$(H |0\rangle)^{\otimes n \dagger} |\psi\rangle = \left(\frac{1}{\sqrt{N}} \sum_{y \in \{0,1\}^n} \langle y| \right) \left(\sum_x \alpha_x |x\rangle \right) \tag{3.13}$$

$$= \frac{1}{\sqrt{N}} \sum_{x,y \in \{0,1\}^n} \alpha_x \langle y|x\rangle . \tag{3.14}$$

Since x and y are both computational basis vectors, $\langle y|x\rangle = \delta_{xy}$. The only nonzero terms in this sum are those where $x = y$, and in that case $\langle x|y\rangle = 1$. Taking this into account,

$$(H|0\rangle)^{\otimes n\dagger}|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x\in\{0,1\}^n} \alpha_x. \tag{3.15}$$

Define

$$\mu = \frac{1}{N} \sum_{x\in\{0,1\}^n} \alpha_x \tag{3.16}$$

so that

$$(H|0\rangle)^{\otimes n\dagger}|\psi\rangle = \mu\sqrt{N}. \tag{3.17}$$

By substituting the results (3.10) and (3.17) into (3.9), we find $D|\psi\rangle$ can be expressed as

$$\begin{aligned} D|\psi\rangle &= 2\left(\frac{1}{\sqrt{N}} \sum_{x\in\{0,1\}^n} |x\rangle\right)\mu\sqrt{N} - \sum_{x\in\{0,1\}^n} \alpha_x|x\rangle \\ &= \sum_{x\in\{0,1\}^n} (2\mu - \alpha_x)|x\rangle. \end{aligned} \tag{3.18}$$

We take a moment to interpret this result geometrically. We have defined μ so that it is the average of all coefficients α_x for the state ψ. The distance from μ to α_x is $\alpha_x - \mu$. To 'reflect' the value α_x across μ, the expression would be $\mu - (\alpha_x - \mu) = 2\mu - \alpha_x$. Informally, D reflects each coefficient of $|\psi\rangle$ across the mean of the coefficients.

3.1.3 Steps to Grover's Algorithm

We provide the steps to Grover's algorithm in this subsection. This is followed by a discussion of the efficiency of the algorithm, in terms of the number of repetitions of Step 2 needed.

Grover's Database Search Algorithm

Input: A function $f : \{0,1\}^n \to \{0,1\}$, where $f(x) = 0$ for all but one integer x in the range $[0, 2^n - 1]$.

Output: The unique integer $x^* \in [0, 2^n - 1]$ such that $f(x) = 1$.

Step 0: Prepare an n-qubit quantum register in the state $|0\rangle$.

Step 1: Apply $H^{\otimes n}$ to the register, resulting in the state

$$\frac{1}{\sqrt{N}} \sum_{x\in\{0,1\}^n} |x\rangle.$$

STEP 2: Apply the transformation DS_f to the state $|\psi\rangle$. In other words, apply one oracle gate, followed by one diffusion gate.

STEP 3: Repeat the previous step until $\left\lceil \frac{\pi}{8}\sqrt{N} \right\rceil$ iterations have taken place.

STEP 4: Measure the register, and check if the result y satisfies $f(y) = 1$. If yes, return y. If no, then the algorithm must be repeated.

3.1.4 Changes in State After Gate Applications

Grover's algorithm consists of alternating applications of the gates S_f, which we refer to as the oracle gate, and D, which we refer to as the diffusion gate.

We define several quantities to aid in the upcoming efficiency analysis.

Definition 3.1. Let $\alpha^{(t)}$ be the coefficient of the state $|x^*\rangle$ in $(DS_f)^t |\psi_0\rangle$, where $|\psi_0\rangle$ represents the initial state.

Let $\beta^{(t)}$ be the coefficient of any other state $|x\rangle$, $x \neq x^*$, in the same expression. (We will prove by induction that $\beta^{(t)}$ is well-defined.)

Finally, let $\mu^{(t)}$ be the average of all coefficients in the state $|\psi_t\rangle$, where $|\psi_t\rangle = S_f (DS_f) |\psi_0\rangle$.

We will show that

$$\alpha^{(t+1)} = 2\mu^{(t)} + \alpha^{(t)} \tag{3.19}$$

$$\beta^{(t+1)} = 2\mu^{(t)} - \beta^{(t)} \tag{3.20}$$

$$\mu^{(t+1)} = -\frac{1}{N}\alpha^{(t+1)} + \frac{N-1}{N}\beta^{(t+1)}. \tag{3.21}$$

At the start of the algorithm, all basis states are in an equal superposition. We therefore have

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle, \tag{3.22}$$

so

$$\alpha^{(0)} = \frac{1}{\sqrt{N}}$$

$$\beta^{(0)} = \frac{1}{\sqrt{N}}.$$

Since $S_f |\psi_0\rangle = -\frac{1}{\sqrt{N}} |x^*\rangle + \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n x^*} |x\rangle$, we can calculate that

$$\begin{aligned} \mu^{(0)} &= \frac{1}{N}\left(-\frac{1}{\sqrt{N}}\right) + \frac{N-1}{N}\left(\frac{1}{\sqrt{N}}\right) \\ &= \frac{N-2}{N\sqrt{N}}. \end{aligned}$$

So far, we can confirm that (3.21) is satisfied in this case.

Applying a diffusion gate to $S_f |\psi_0\rangle$, we see that

$$\begin{aligned} DS_f |\psi_0\rangle &= \left(2\mu - (-\alpha^{(0)})\right) |x^*\rangle + \sum_{x \neq x^*} \left(2\mu - \beta^{(0)}\right) |x\rangle \\ &= \left(2\frac{N-2}{N\sqrt{N}} + \frac{1}{\sqrt{N}}\right) |x^*\rangle + \sum_{x \neq x^*} \left(2\frac{N-2}{N\sqrt{N}} - \frac{1}{\sqrt{N}}\right) |x\rangle \\ &= \frac{3N-4}{N\sqrt{N}} |x^*\rangle + \sum_{x \neq x^*} \frac{N-4}{N\sqrt{N}} |x\rangle . \end{aligned}$$

From the last line of this equation, we read off $\alpha^{(1)}$ and $\beta^{(1)}$ as

$$\begin{aligned} \alpha^{(1)} &= \frac{3N-4}{N\sqrt{N}} \\ \beta^{(1)} &= \frac{N-4}{N\sqrt{N}}. \end{aligned}$$

We take a moment to informally consider the results of these two gates. Notice that, while the mean of the coefficients decreased only slightly from $\frac{1}{\sqrt{N}}$, $\alpha^{(1)}$ is close to $\frac{3}{\sqrt{N}}$, almost a three-fold increase. On the other hand, $\beta^{(1)}$ is only slightly less than $\beta^{(0)}$. This larger relative change in $\alpha^{(t)}$ over a number of iterations allows us to establish that we have a certain constant chance of measuring x^*. The number of iterations to reach this point will depend on N.

Let $|\psi\rangle = (DS_f)^t |\psi_0\rangle$. We can express $|\psi\rangle$ using $\alpha^{(t)}$ and $\beta^{(t)}$, as

$$|\psi\rangle = \alpha^{(t)} |x^*\rangle + \sum_{x \neq x^*} \beta^{(t)} |x\rangle .$$

First, applying S_f,

$$S_f |\psi\rangle = -\alpha^{(t)} |x^*\rangle + \sum_{x \neq x^*} \beta^{(t)} |x\rangle .$$

Taking the average of the coefficients of $S_f |\psi\rangle$, we can confirm, as claimed by (3.21),

$$\mu^{(t)} = \frac{1}{N}\left(-\alpha^{(t)}\right) + \frac{N-1}{N}\beta^{(t)}.$$

By applying D, we see that

$$DS_f |\psi\rangle = (2\mu^{(t)} - (-\alpha^{(t)}) |x\rangle + \sum_{x \neq x^*} \left(2\mu^{(t)} - \beta^{(t)}\right) |x\rangle .$$

We can read off $\alpha^{(t+1)}$ and $\beta^{(t+1)}$ from this equation in terms of $\alpha^{(t)}$, $\beta^{(t)}$, and $\mu^{(t)}$. We see that

$$\begin{aligned} \alpha^{(t+1)} &= 2\mu^{(t)} + \alpha^{(t)} \\ \beta^{(t+1)} &= 2\mu^{(t)} - \beta^{(t)}, \end{aligned}$$

confirming the equations (3.19) and (3.20).

Next, we consider how many times to apply the transformation GS_f, and the corresponding probability of measuring x^*.

3.2 Efficiency of Grover's Algorithm

First, we will give a closed form representation of our coefficients as found in [2] to simplify analysis. We define an angle θ, such that

$$\sin^2\theta = \frac{1}{N}$$

and $0 < \theta \leq \pi/2$. Consideration of the right triangle formed by such an angle with a unit length hypotenuse will allow us to find that $\cos^2\theta = \frac{N-1}{N}$. Inspection of these values should indicate a connection between the sine term and the α coefficient and the cosine term and the β coefficient. Now we wish to eliminate the $\mu^{(t)}$ dependence in (3.19) and (3.20) by substituting in (3.21). The result of this is

$$\alpha^{(t+1)} = \frac{N-2}{N}\alpha^{(t)} + \frac{2N-2}{N}\beta^{(t)} \tag{3.23}$$

$$\beta^{(t+1)} = -\frac{2}{N}\alpha^{(t)} + \frac{N-2}{N}\beta^{(t)}. \tag{3.24}$$

Lemma 3.2. *The coefficients of the states, $\alpha^{(t)}$ and $\beta^{(t)}$ can be represented in the closed form*

$$\alpha^{(t)} = \sin((2t+1)\theta) \qquad \beta^{(t)} = \frac{1}{\sqrt{N-1}}\cos((2t+1)\theta) \tag{3.25}$$

Proof. We will proceed by induction. When $t = 0$, $\alpha^{(0)} = \sin(\theta) = \frac{1}{\sqrt{N}}$, and $\beta^{(0)} = \frac{1}{\sqrt{N-1}}\cos(\theta) = \frac{1}{\sqrt{N}}$, agreeing with the previously determined initial values. The base case is satisfied.

Assume $\alpha^{(t)} = \sin((2t+1)\theta)$ and $\beta^{(t)} = \frac{1}{\sqrt{N-1}}\cos((2t+1)\theta)$ for some nonnegative integer t. Starting with the new closed form for $\alpha^{(t+1)}$, we state that

$$\begin{aligned}\alpha^{(t+1)} &= \frac{N-2}{N}\sin((2t+1)\theta) + \frac{2(N+1)}{N}\frac{1}{\sqrt{N-1}}\cos((2t+1)\theta) \\ &= \left(1 - \frac{2}{N}\right)\sin((2t+1)\theta) + \frac{2\sqrt{N-1}}{N}\cos((2t+1)\theta).\end{aligned}$$

By making the substitutions $\frac{2}{N} = 2\sin^2\theta$ and $\cos\theta = \frac{\sqrt{N-1}}{\sqrt{N}}$, we may arrive at the result

$$\begin{aligned}\alpha^{(t+1)} &= (1 - 2\sin^2\theta)\sin((2t+1)\theta) + \cos\left((2t+1)\theta\right)2\sin\theta\cos\theta \\ &= \cos(2\theta)\sin((2t+1)\theta) + \cos\left((2t+1)\theta\right)\sin(2\theta) \\ &= \sin\left((2t+1)\theta + 2\theta\right) \\ &= \sin\left((2t+3)\theta\right),\end{aligned}$$

by making use of the double angle identities for sine and cosine, as well as the angle sum identity for sine.

For $\beta^{(t+1)}$, we follow a similar process, starting with (3.24). We can say that

$$\begin{aligned}
\beta^{(t+1)} &= \frac{N-2}{N}\frac{1}{\sqrt{N-1}}\cos\left((2t+1)\theta\right) - \frac{2}{N}\sin\left((2t+1)\theta\right) \\
&= \frac{1}{\sqrt{N-1}}\left[\left(\frac{2N-2}{N}-1\right)\cos\left((2t+1)\theta\right) - 2\frac{\sqrt{N-1}}{N}\sin\left((2t+1)\theta\right)\right] \\
&= \frac{1}{\sqrt{N-1}}\left[\left(2\cos^2\theta - 1\right)\cos\left((2t+1)\theta\right) - 2\sin(\theta)\cos(\theta)\sin\left((2t+1)\theta\right)\right] \\
&= \frac{1}{\sqrt{N-1}}\left[\cos(2\theta)\cos\left((2t+1)\theta\right) - \sin(2\theta)\sin\left((2t+1)\theta\right)\right] \\
&= \frac{1}{\sqrt{N-1}}\cos\left((2t+1)\theta + 2\theta\right) \\
&= \frac{1}{\sqrt{N-1}}\cos\left((2t+3)\theta\right)
\end{aligned}$$

The steps to this argument are similar to the steps of the previous one. The double angle identities are used again, as well as the angle sum identity for cosine. □

With these equations, we can determine the amplitude of the unique value x^* after a given number of iterations. From (3.25), the maximum value attainable for $\alpha^{(t)}$ is 1, which occurs at $t = (\pi - 2\theta)/4\theta$, solving for t. We also need to take the floor of this value, as the number of iterations must be an integer.

However, the amplitude of $\alpha^{(t)}$ decreases again after $t = (\pi - 2\theta)/4\theta$, so we wish to terminate the algorithm before this occurs. One option is to measure after the number of iterations when $\alpha^{(t)}$ is near $\frac{1}{\sqrt{2}}$, such that the probability of measuring the state $|x^*\rangle$ is near $\frac{1}{2}$. It is possible to measure the desired value with fifty percent certainty if we iterate to an integer near $\frac{\pi}{8}\sqrt{N}$. Substituting this value in for t in (3.25) will yield

$$\alpha^{(t)} = \sin(\pi/4 + \theta),$$

which has squared modulus near $\frac{1}{2}$. Therefore, if we terminate the algorithm at $\left\lceil \frac{\pi}{8}\sqrt{N} \right\rceil$ steps, we will observe the state $|x^*\rangle$ with probability near 1/2. Based on the number of iterations, our efficiency is $\mathcal{O}\left(\sqrt{N}\right)$. With a half probability of measuring the desired value, after k completions of the algorithm, the probability of success is $1 - \frac{1}{2^k}$. In practice, the measurement result might be checked after each completion of the algorithm to see if x^* was found, as verifying a result can be done classically.

We can see that after a few completions of the algorithm, we will find x^* with very high probability. After 10 completions, the probability of measuring x^* is approximately 99.902%.

Chapter 4

Shor and Order Finding

In this chapter, we address two mathematical problems, their uses in cryptography, and quantum algorithms to solve them. These two problems, factoring and computing discrete logarithms, both relate to another problem, order finding. No polynomial time algorithms have been found to solve either of these problems, so they are generally considered to be computationally intractable, and encryption systems have been created which rely on their difficulty for system security. However, Peter Shor found quantum algorithms for both of these problems which run in probabilistic polynomial time.

One of the most widely used encryption algorithms, the RSA algorithm, is based on factoring, and we begin with an overview of this problem and precisely how messages can be decrypted using factoring. We present Shor's algorithm for factoring, split into the components relying on quantum mechanics and the auxiliary computations which may be completed using a classical computer. The third section introduces the discrete log problem, and the quantum algorithm to solve this problem comprises the final section.

4.1 The RSA Cryptosystem

The RSA cryptosystem is an example of a public-key cryptosystem. In public-key encryption, or asymmetric encryption, a key pair is constructed for each user, with one key used for encryption and the other used for decryption. The *public key* is made available for any sender to use, while the corresponding *private key* is kept secret and used only by the recipient.

Before defining the RSA cryptosystem, we establish the necessary background terminology [14][Sec. 1.4]. An *alphabet of definition* is a finite set $\mathcal{A}$. The *message space* $\mathcal{M}$ is a set containing strings of symbols from the alphabet of definition. A message is an element $m \in \mathcal{M}$, which may be referred to as a plaintext message. The set $\mathcal{C}$ is the *ciphertext space*. Typically, $\mathcal{C}$ contains strings of symbols from an alphabet of definition, not necessarily the same alphabet as the message space. An element $c \in \mathcal{C}$ may be referred to as a ciphertext.

The set $\mathcal{K}$ denotes the *key space*. For each $e \in \mathcal{K}$, there is a unique bijection $E_e : \mathcal{M} \to \mathcal{C}$, called an *encryption transformation*. Similarly, for each $d \in \mathcal{K}$, there is a unique bijection

$D_d : \mathcal{C} \to \mathcal{M}$, a *decryption transformation.*

Finally, an *encryption scheme* consists of a chosen $\mathcal{M}$, $\mathcal{C}$, $\mathcal{K}$, $\{E_e : e \in \mathcal{K}\}$, and $\{D_d : d \in \mathcal{K}\}$, where for each $e \in \mathcal{K}$, there is a unique $d \in \mathcal{K}$ such that $D_d(E_e(m)) = m$ for all $m \in \mathcal{M}$. A key e with the corresponding key d may be referred to as a *key pair* and denoted (e, d).

As defined by [14][Sec. 1.8], a *public-key encryption scheme* is an encryption scheme where for each key pair (e, d), the *public key* e is made available to message senders, where the *private key* d is kept private. It must be computationally infeasible to compute d from e for the system to be secure[1].

For one party, traditionally named Alice, to send a message to another party, traditionally named Bob, Bob must have a key pair (e, d) and make e available publicly. Alice may then choose a message $m \in \mathcal{M}$ and encrypt it by using the transformation E_e. Alice sends the ciphertext $c = E_e(m)$ to Bob. When Bob receives c, he uses his private key d and the decryption transformation to find $m = D_d(c)$.

We are now prepared to discuss the RSA cryptosystem in particular. First, a key pair is generated through the process given by the following steps.

RSA Key Generation

Step 1: Choose two distinct prime numbers p and q.

Step 2: Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$.

Step 3: Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.

Step 4: Find the unique integer d where $1 < d < \phi(n)$ and $ed \equiv 1 \pmod{\phi(n)}$.

Step 5: The public key is (n, e) and the private key is d.

From these steps, we see that during key generation, e is chosen randomly, and d is computed from e. Given $\phi(n)$ however, computing d from e is possible in polynomial time using the extended Euclidean algorithm. For an attacker, without p and q, $\phi(n)$ cannot be easily found[2]. Computing d from (n, e) is *computationally equivalent* to factoring n. In other words, given a process to either factor n or compute d from (n, e) in polynomial time, the other task is possible in polynomial time as well. As the steps above require polynomial time, one side of this statement is clear, and the other side is given in [14][Sec. 8.2].

To encrypt a message m, it must be represented as an integer such that $0 \leq m \leq n-1$. The ciphertext is $c = m^e \pmod{n}$. To decrypt a message, $m = c^d \pmod{n}$.

The specific intractable problem the RSA algorithm relies on is the following problem when p and q are large primes [14][Sec. 3.3].

Definition 4.1. Given $n \in \mathbb{N}$ where n is a product of two distinct primes p and q, $e \in \mathbb{N}$ where $\gcd(e, (p-1)(q-1)) = 1$, and $c \in \mathbb{Z}$, assume there exists $m \in \mathbb{Z}$ such that $m^e \equiv c \pmod{n}$. The task of finding m, given n, e, and c, is called the *RSA problem* (RSAP).

[1]What is considered "secure" is an issue for cryptography experts, and specific criteria vary by situation. There is an overview in [14][Sec. 1.2].

[2]This is not to say that there are no other ways for an attacker to decrypt a message. There is a discussion of other possible attacks on the RSA encryption scheme in [14][Sec 8.2].

If factoring an integer n can be achieved in polynomial time, then the RSAP can be solved in polynomial time. This may be done using the steps given for RSA key generation and message encryption, both possible in polynomial time. It is believed that integer factorization reduces to the RSAP as well, but this has not been proven. Informally, it might be possible to solve the RSAP by some means which is in some sense "easier" than factoring, but this is considered unlikely.

The current best known running times for classical factoring algorithms come from the number field sieve. The general number field sieve has running time

$$\mathcal{O}\left(\exp\left(\frac{1}{3}+o(1)\right)\ln(n)^{(32/9)^{1/3}}\ln(\ln(n))^{1-(32/9)^{1/3}}\right), \tag{4.1}$$

according to [14][Sec. 3.2.7].

4.2 Shor's Factoring Algorithm

Given that breaking the RSA cryptosystem is considered likely to depend on factoring, the development of a quantum algorithm to factor integers in a polynomial number of gates is an important result in cryptography. In this section, we elaborate on the algorithm as given by [20].

We present the algorithm in two parts. The quantum computation is presented first. The measurement does not immediately provide the factors of n, but the output can be used with high probability to do so. We give the additional steps, which may be completed with a classical computer in polynomial time, as a separate algorithm.

Shor's Factoring Algorithm - Quantum Computation

Input: An integer n, a product of two distinct primes p_1 and p_2.

Output: Integers x and c, such that $g(c) = r$ satisfies $x^r \equiv 1 \pmod{n}$ with some probability[3].

Step 0: Let $q = 2^l$ where $l \in \mathbb{N}$ and $n^2 < q < 2n^2$. Prepare two l-qubit quantum registers in the state $|0\rangle\,|0\rangle$.

Step 1: Apply $H^{\otimes l}$ to the first register. This may be represented as the multiplication

$$\left(H^{\otimes l}\otimes I^{\otimes l}\right)(|0\rangle\otimes|0\rangle) = \frac{1}{\sqrt{2^l}}\sum_{a=0}^{q-1}|a\rangle\otimes|0\rangle\,. \tag{4.2}$$

Step 2: Choose a random $x \in \mathbb{N}$ where $1 \le x < n$, and apply the transformation

$$|a\rangle\,|0\rangle \to |a\rangle\,|x^a \pmod{n}\rangle\,. \tag{4.3}$$

[3]The probability is discussed in this section, see (4.10).

We provide a classical polynomial time algorithm to compute $x^a \pmod n$, and from the results of Chapter 2, it may be implemented as a quantum circuit in a polynomial number of gates.

The state of the two registers after this transformation is

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle \, |x^a \pmod n\rangle \, . \tag{4.4}$$

STEP 3: Apply the QFT to the first register. Recall this transformation acts on a basis state as

$$|a\rangle \to \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} \exp(2\pi i ac/q) \, |c\rangle \, , \tag{4.5}$$

so the state of the two registers is

$$\frac{1}{q} \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} \exp(2\pi i ac/q) \, |c\rangle \, |x^a \pmod n\rangle \, . \tag{4.6}$$

STEP 4: Measure the state of the two registers with the measurement $\mathcal{M} = \{S_{ab} : a, b \in [0, q-1]\}$ where $S_{ab} = \text{span}\{|a\rangle \, |b\rangle\}$. Informally, both registers are measured so that the state after the measurement for each will be a vector from the computational basis.

After measurement, the state of the system is

$$|c\rangle \, |x^a \pmod n\rangle \tag{4.7}$$

for some c and some $x^a \pmod n$.

The result of the measurement will be two numbers, c and $x^a \pmod n$. The value used in the following classical steps is c.

From the result of the measurement of the first register, we hope to identify the order of x, or the smallest integer $r \in \mathbb{N}$ with $0 \leq r < n$ so that $x^r \equiv 1 \pmod n$. The result returned by the measurement will not always yield r, so we will later discuss the probability of success at this step.

Next, we hope to use the order of x to find the two nontrivial factors of n. Again, there is a probability of failure, so we analyze the possible outcomes.

We explain how the order r is found from c, as well as how the factors of n are determined from this result. A summary of the steps is as follows.

SHOR'S FACTORING ALGORITHM - ADDITIONAL COMPUTATION [CLASSICAL PART]

INPUT: An integer c where $0 \leq c < q$, and an integer x where $1 \leq x < n$.

OUTPUT: Two factors of n, or a result indicating failure.

STEP 1: Find the closest approximation $\frac{d}{r}$, with $d, r \in \mathbb{Z}$, to $\frac{c}{q}$, where it must be that $r < n$. Let the function g map each c to the corresponding r.

STEP 2: Check if $x^r \equiv 1 \pmod{n}$, and if r is even. If either of these conditions is not satisfied, indicate a failure.

STEP 3: Let $y = x^{\frac{r}{2}} \mod n$. Compute $\gcd(y-1, n)$ and $\gcd(y+1, n)$ to find two factors of n.

We begin with the result of the measurement, c, and the value q as defined during the quantum computation. In Step 1, we find the closest rational approximation to $\frac{c}{q}$ with denominator less than n. This is possible through use of continued fractions. As discussed in Section 4.2.2, this technique yields "convergents" to $\frac{c}{q}$, in lowest terms. Using continued fractions, the first convergent of $\frac{c}{q}$ is $\frac{0}{1}$. This is a result of the inequality $c < q$. The denominators of the convergents are strictly increasing, so there will be a unique convergent with a maximal denominator which is less than n. We let $\frac{d'}{r'}$ equal this convergent.

The initial steps in calculation the probability of measuring a c is given in Appendix B. In this appendix, we prove that if, for a given c, there exists an integer d to satisfy

$$\left|\frac{c}{q} - \frac{d}{r}\right| \leq \frac{1}{2q}, \tag{4.8}$$

then the probability of an outcome of the measurement with c in the first register is at least $\frac{1}{3r^2}$.

If d and r are coprime, then $\frac{d}{r}$ is in lowest terms. When we find the convergents to $\frac{c}{q}$, the last convergent with a denominator smaller than n, denoted $\frac{d'}{r'}$ will be $\frac{d}{r}$. However, if d and r are not coprime, then the fraction $\frac{d'}{r'}$ is equal to $\frac{d}{r}$, but the numerator and denominator will be different. I.e., there is some $k \in \mathbb{N}$, $k > 1$, such that $kd = d'$ and $kr = r'$. In this case, we will have recovered r' where $r' \neq r$.

We have determined that to correctly identify the order of x, d and r in (4.8) must be coprime. Given that $d < r$, the number of possible values for d such that d and r are coprime is captured by Euler's totient function, $\phi(r)$.

Definition 4.2. The *Euler totient function* $\phi(n)$, defined for positive integers n, is the number of integers in the interval $[1, n]$ which are coprime to n.

However, there are r outcomes of the measurement which yield a given c, because there are states $|c\rangle |x \pmod{n}\rangle, |c\rangle |x^2 \pmod{n}\rangle, \ldots, |c\rangle |x^r \pmod{n}\rangle$, each a distinct outcome of the measurement.

The total number of outcomes where there is a c such that r may be identified is $r\phi(r)$, and the probability of each of these outcomes is at least $\frac{1}{3r^2}$. Therefore, the probability that one such c is chosen is at least

$$\frac{\phi(r)}{3r}. \tag{4.9}$$

Using the lower bound on $\phi(r)$ from [14][Fact 2.102, Sec. 2.4], if $r \geq 5$, $\frac{\phi(r)}{r} \geq \frac{1}{6 \ln \ln r}$. Then

$$\frac{\phi(r)}{3r} \geq \frac{1}{18 \ln \ln r}. \tag{4.10}$$

Because the probability of success is bound below by $\frac{1}{18 \ln \ln r}$, we expect to find an r' equal to r after $\mathcal{O}(\ln \ln r)$ completions of the algorithm, assuming independence.

For the next step, we need two additional conditions on r, which are provided by the following theorem.

Theorem 4.1. *Let N be a natural number with prime factorization pq, where p and q are distinct primes. Then for an integer x chosen at random in the interval $[1, N-1]$, with $gcd(N, x) = 1$,*

$$Pr[r \text{ is even and } x^{r/2} \not\equiv -1 \ (mod\ N)\ ] \geq \frac{1}{2}, \tag{4.11}$$

where r is the order of x $(mod\ N)$.

The proof of this theorem is included in Appendix A.

This theorem requires $\gcd(n, x) = 1$, but this is not an issue. We can check $\gcd(n, x)$ after choosing x at the start of the algorithm. If $gcd(n, x) > 1$, then we have stumbled across a nontrivial factor of n and there is no need to proceed. Given that the security of the RSA algorithm depends on factoring n being computationally intractable, the probability of finding a factor by chance is extremely small, and this case need not be considered further.

Otherwise, we have identified r such that $x^r \equiv 1 \pmod{n}$. Rearranging this, $x^r - 1 \equiv 0 \pmod{n}$. If r is divisible by two, we can factor the left side to find

$$(x^{r/2} + 1)(x^{r/2} - 1) \equiv 0 \pmod{n}.$$

Since $n \nmid (x^{r/2} + 1)$ and $n \nmid (x^{r/2-1})$, but $n | (x^{r/2} + 1)(x^{r/2} - 1)$, we have that $\gcd(n, x^{r/2} + 1) > 1$ and $\gcd(n, x^{r/2} - 1) > 1$. In this scenario, we have found the desired factors of n.

4.2.1 Modular Exponentiation

This subsection provides a polynomial time algorithm for modular exponentiation, as needed for what we refer to as Step 2 of Shor's factoring algorithm, in the quantum section. As there is a polynomial time algorithm for a classical computer, the same process may be implemented as a quantum circuit in a polynomial number of gates.

We take for granted that multiplication and division are possible with a classical computer in complexity[4] $\mathcal{O}((\lg n)^2)$, for $a, b \in \mathbb{N}$ and $n = \max\{a, b\}$. In this case, the division operation returns $a = qb + r$, with $q, r \in \mathbb{Z}$. These are given in [14][Sec. 2.4.2].

From these operations, we can construct an algorithm for modular exponentiation, known as *square-and-multiply*, as given in [14][Sec. 2.4.2].

The goal is to compute $a^k \pmod{n}$, for $a, k \in \mathbb{N}$, $0 \leq a < n$ and $0 \leq k < n$ We represent k as a binary string, $k_t k_{t-1} \ldots k_1 k_0$, so that $k = \sum_{j=0}^{t} k_j 2^j$.

To compute this exponent, we represent a^k using this alternate representation for k, so that

$$a^k = \prod_{j=0}^{t} a^{k_j 2^j}. \tag{4.12}$$

[4]The notation $\lg n$ is defined as $\log_2 n$.

For this product, we will need a raised to various powers of 2. Instead of finding a^2, a^4, a^8 from a each time, the algorithm completes the steps in a certain order which allows each power to be computed when needed by squaring the previous one. This process may be seen in Step 3 of the algorithm below. For each higher power, there is only one multiplication step, leading to t multiplications to find all necessary powers of a.

As multiplication requires more resources for larger numbers, and the result of the algorithm is a number $\pmod n$, we avoid multiplying numbers larger than n by using the properties of modular arithmetic. Specifically, $ab \equiv cd \pmod n$ if $a \equiv c \pmod n$ and $b \equiv d \pmod n$. In our case, if $a^2 \equiv b \pmod n$, then $a^4 \equiv b^2 \pmod n$. Also, as we compute the product in equation (4.12), we take the remainder modulo n after each multiplication step using the same justification.

Since we only multiply numbers smaller than n, all multiplications have complexity[5] $\mathcal{O}\left((\lg n)^2\right)$.

The square-and-multiply algorithm may be stated as follows.

Square-And-Multiply Algorithm

Input: Integers a and k where $0 \leq a < n$ and $0 \leq k < n$, and k is represented as a binary string with $k_j \in \{0, 1\}$, $0 \leq j \leq t$, and $t = \lceil \lg k \rceil$.

Output: The integer $a^k \pmod n$.

Step 1: Let $b = 1$. If $k = 0$, return b.

Step 2: Let $A = a$. If $k_0 = 1$, then set $b \leftarrow a$.

Step 3: Repeat the following steps with each value of j, from 1 to t.

1. Set $A \leftarrow A^2 \pmod n$.
2. If $k_j = 1$, then set $b \leftarrow Ab \pmod n$.

Step 4: Return b.

We confirm that Square-And-Multiply has complexity $\mathcal{O}\left((\lg n)^3\right)$. Representing k in binary should be trivial as it is stored in this format, but this would require at most $\lg n$ uses of integer division, for a requirement of at worst $\mathcal{O}\left((\lg n)^3\right)$. Steps 1 and 2 require only condition checks and assignments.

Step 3 requires at most $\lg n$ repetitions of a loop. This loop contains at most two multiplications of numbers smaller than n, so each iteration has complexity $\mathcal{O}\left((\lg n)^2\right)$. Step 3 has overall complexity $\mathcal{O}\left((\lg n)^3\right)$. As the algorithm's complexity is representable as the sum of the complexity of each step, the overall complexity is $\mathcal{O}\left((\lg n)^3\right)$.

Example 4.1. In this example, $4^{19} \pmod{31}$ is calculated. Let $a = 4$, $x = 19$, and $n = 31$. The first row of Table 4.1 contains the initial value of A, $A = a = 4$. Because $k_0 = 1$, $b \leftarrow 1 \cdot 4 \equiv 4 \pmod{31}$. In the second row, $A \leftarrow a^2 \equiv 16 \pmod{31}$. As $k_1 = 1$, $b \leftarrow 2 \equiv 16 \cdot 4 \pmod{31}$. This repeats until $t = 4$, and $b = 8$ is the answer returned.

[5] As is done by [14], we approximate the more precise $\lceil \lg n \rceil$ count of the maximum number of bits needed to represent each number by $\lg n$.

A	k	b
4	1	4
16	1	2
8	0	2
2	0	2
4	1	8

Table 4.1: Modular Exponentiation Example

4.2.2 Continued Fractions

This subsection provides an explanation of the procedure used in what we denote as Step 1 of the classical section of Shor's factoring algorithm. The continued fractions method may be used to find approximations of fractions where the numerator and denominator are natural numbers[6]. We present it using the Euclidean algorithm as a reference, as the two are closely related.

Consider $a, b \in \mathbb{N}$, where $b \geq 1$. The Euclidean algorithm may be used to find $\gcd(a, b)$. We examine the steps now.

By the division algorithm for integers [14][Sec. 2.4.1], there are unique integers q and r_0, referred to as the quotient and remainder, such that

$$a = qb + r_0, \quad \text{where} \quad 0 \leq r_0 < b. \tag{4.13}$$

We repeat this step on $r_{-1} = b$ and r_0. We find

$$b = q_1 r_0 + r_1, \quad \text{where} \quad 0 \leq r_1 < r_0. \tag{4.14}$$

These steps are repeated until for some $t \in \mathbb{N}$, r_t divides r_{t-1}. By substituting from $r_{t-2} = q_t r_{t-1} + r_t$ into $r_{t-3} = q_{t-1} r_{t-2} + r_{t-1}$ and so on, it is possible to show that $\gcd(a, b) = r_t$.

At this point, we can see the above algorithm does terminate. At the first step, we have $r_0 < b$, and at each subsequent step k, we have $r_{k-1} < r_{k-2}$. Because all of the remainders are nonnegative integers, and they are strictly decreasing from the initial nonnegative integer b, there are at most b iterations before the remainder is zero. We will improve this bound later, but we establish at this point that the algorithm will terminate.

A rational number represented as a continued fraction takes the form

$$\frac{a}{b} = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots + \cfrac{1}{q_{t+1}}}}}, \tag{4.15}$$

where $q_j \in \mathbb{Z}$, for $j \in \mathbb{N}$ with $0 \leq j \leq t + 1$.

[6]There are more general uses of continued fractions to approximate irrational numbers, but we do not discuss these as they are not relevant.

We construct this representation by a modified version of the Euclidean algorithm. With $a, b \in \mathbb{N}$, and $b \geq 1$, we found (4.13). We rearrange this equation to express $\frac{a}{b}$ as

$$\frac{a}{b} = q_0 + \frac{r_0}{b}. \tag{4.16}$$

Referring to (4.14),

$$\frac{b}{r_0} = q_1 + \frac{r_1}{r_0}. \tag{4.17}$$

As we continue this process, the continued fraction representation of $\frac{a}{b}$ may be built. From the two equations thus far,

$$\frac{a}{b} = q_0 + \frac{1}{q_1 + \frac{r_1}{r_0}}. \tag{4.18}$$

We continue using the division algorithm on pairs of remainders, r_{k-1} and r_k, until we find r_t which divides r_{t-1}. After $t+1$ iterations, the last equation is

$$r_{t-2} = q_t r_{t-1} + r_t. \tag{4.19}$$

Because r_t divides r_{t-1}, we have $r_{t-1} = q_{t+1} r_t$. Dividing by r_t, we substitute $\frac{r_{t-1}}{r_t} = q_{t+1}$ into the equation

$$\frac{a}{b} = q_0 + \cfrac{1}{q_1 + \cfrac{1}{\ddots + \frac{r_t}{r_{t-1}}}}, \tag{4.20}$$

to arrive at (4.15).

For example,

$$\frac{8}{3} = 2 + \frac{1}{1 + \frac{1}{2}}.$$

We will use this representation to find approximations of $\frac{a}{b}$, but first we explore the complexity of this algorithm. We established that there are at most b iterations of the division algorithm needed, so that we may find the gcd, or for our purposes, arrive at a continued fractions representation with a finite number of fractions, but we can find a better bound.

Let $n \in \mathbb{N}$ be a number greater than or equal to a and b. We will establish that the number of applications of the division algorithm, referred to as iterations, is bounded by $\mathcal{O}(\lg n)$.

From (4.13), $r_0 < b$. If we continue from (4.14), the third use of the division algorithm yields

$$r_0 = q_2 r_1 + r_2, \quad \text{where} \quad r_2 < r_1. \tag{4.21}$$

By substitution, $r_2 < \frac{b}{2}$. From the next three applications of the division algorithm, $r_1 = q_3 r_2 + r_3$, $r_2 = q_4 r_3 + r_4$, and $r_3 = q_5 r_4 + r_5$. Using the first and third of these equations, we see that $r_5 < \frac{r_2}{2}$. From the results so far,

$$r_5 < \frac{r_2}{2} < \frac{b}{4}. \tag{4.22}$$

In fact, for $j \leq t$, $r_j < \frac{r_{j-3}}{2}$. This is evident from the equations $r_{j-4} = q_{j-2}r_{j-3} + r_{j-2}$, $r_{j-3} = q_{j-1}r_{j-2} + r_{j-1}$, and $r_{j-2} = q_j r_{j-1} + r_j$, using the same argument.

Choose $l \in \mathbb{N}$ such that $2^{l-1} \leq b < 2^l$. Then $2^{l-2} \leq r_2 < 2^{l-1}$, and more generally $2^{l-k} \leq r_{3k-4} < 2^{l-k+1}$ for $2 \leq k \leq l$. From these inequalities, the remainder at a given step will be less than half the remainder from three steps ago. This determines an upper bound on the number of iterations, $3l$.

We can think of this another way. Considering b as an l-bit binary string, after three iterations, the remainder r_0 will require at most $l-1$ bits to represent. This pattern continues every three iterations, and is another way to interpret the upper bound of $3l$ iterations.

As we chose n greater than a and b, we can state the upper bound on applications of the division algorithm as $3 \lg n$. From [14][Sec. 2.4.2], integer division has complexity $\mathcal{O}\left((\lg n)^2\right)$, so we can establish an upper bound of $\mathcal{O}\left(\lg n^3\right)$ for computing continued fractions expressions for rational numbers.

Now that we are able to compute continued fraction representations of rational numbers in time polynomial in the size of the input, we address the way it is used to generate approximations of rational numbers.

Using Continued Fractions to Find Approximations

The method of continued fractions can be used to find approximations of the fraction $\frac{a}{b}$, where a and b are as defined previously, with denominator less than a given value. In Shor's factoring algorithm, we look for approximations of $\frac{c}{q}$ with denominator less than n. This takes place in what we denote as Step 1 of the classical part of Shor's algorithm.

For the fraction $\frac{a}{b}$, we encode the continued fraction from (4.15) as

$$\frac{a}{b} = [q_0, q_1, \ldots, q_{t+1}], \tag{4.23}$$

where q_0 is a nonnegative integer, and $q_1, \ldots, q_n$ are positive integers.

We define the *convergents* of $\frac{a}{b}$ as

$$c_j = [q_0, \ldots, q_j], \quad \text{where} \quad 0 \leq j \leq q+1. \tag{4.24}$$

For example, $c_0 = q_0$, $c_1 = q_0 + \frac{1}{q_1} = \frac{q_0 q_1 + 1}{q_1}$, and $c_{q+1} = \frac{a}{b}$.

These convergents may be expressed in a more compact form, as fractions with integer numerators and denominators according to a theorem from [17][Chap. 1].

Theorem 4.2. *The j-th convergent of the continued fraction $[q_0, q_1, \ldots, q_{t+1}]$ may be expressed as a fraction with numerator p_j and denominator s_j, where*

$$p_j = q_j p_{j-1} + p_{j-2}, \tag{4.25}$$

$$\textit{and} \quad s_j = q_j s_{j-1} + s_{j-2}, \tag{4.26}$$

for $2 \leq j \leq t+1$, where $j = 0$ and $j = 1$ have numerators and denominators

$$p_0 = q_0, \qquad p_1 = q_1 q_0 + 1, \tag{4.27}$$

$$s_0 = 1, \qquad s_1 = q_1. \tag{4.28}$$

These values p_j and s_j are the result of making substitutions in $[q_0, \ldots, q_j]$. We will establish that these convergents are fractions in lowest terms using a second theorem from the same source, [17][Chap. 1].

Theorem 4.3. *For p_j and s_j as previously defined, with the additional values*

$$p_{-1} = 1, \qquad p_{-2} = 0, \tag{4.29}$$

$$s_{-1} = 0, \qquad s_{-2} = 1, \tag{4.30}$$

it follows that

$$p_j s_{j-1} - p_{j-1} s_j = (-1)^j, \tag{4.31}$$

for $j \geq 0$.

From this theorem, we prove a corollary from [17]. For convergent $c_j = \frac{p_j}{s_j}$, c_j is represented in lowest terms. Any factor of p_j and s_j divides the left side of (4.31), so it must also divide the right side. However, the only integers which divide the right side are 1 and -1. Therefore, p_j and s_j only have the common factors 1 and -1, and the fraction $\frac{p_j}{s_j}$ is in lowest terms.

From Theorem 4.2, the denominators of the convergents, expressed in the way given by the theorem, are strictly increasing. Since these representations are also in lowest terms, we can consider the convergents in order when searching for an approximation of a fraction where the denominator must be below a certain value.

Example 4.2. As an example, we find the continued fraction representation of $\frac{27}{256}$.

From one application of the division algorithm with $a = 27$ and $b = 256$, $27 = 0 \cdot 256 + 27$. Therefore, $q_0 = 0$, and $r_0 = 27$.

Dividing again, we find $256 = 9 \cdot 27 + 13$, so $q_1 = 9$ and $r_1 = 13$. Repeating this process, $27 = 2 \cdot 13 + 1$, with $q_2 = 2$ and $r_2 = 1$. At this point, we notice r_2 divides r_1. The final equation is $13 = 13 \cdot 1 + 0$, with $q_3 = 13$ and $r_3 = 0$.

We can express these steps, and the final expression, as

$$\begin{aligned} \frac{27}{256} &= \frac{1}{\frac{256}{27}} \\ &= \frac{1}{9 + \frac{13}{27}} \\ &= \frac{1}{9 + \frac{1}{\frac{27}{13}}} \\ &= \frac{1}{9 + \frac{1}{2 + \frac{1}{13}}}. \end{aligned} \tag{4.32}$$

Now, we consider the convergents. From Theorem 4.2, $c_0 = \frac{q_0}{1} = 0$, and $c_1 = \frac{q_1 q_0 + 1}{q_1} = \frac{1}{9}$. We use the general equations to find $p_2 = q_2 p_1 + p_0 = 2$ and $s_2 = q_2 q_1 + q_0 = 19$. Combining these, $c_2 = \frac{2}{19}$.

For the next approximation, $p_3 = q_3 p_2 + p_1 = 13 \cdot 2 + 1 = 27$ and $s_3 = q_3 s_2 + s_1 = 13 \cdot 19 + 9 = 256$. As expected, $c_3 = \frac{27}{256}$. The last convergent is equal to the original fraction.

We can verify the results stated about the convergents. In this example, $c_0 = \frac{0}{1}$, $c_1 = \frac{1}{9}$, $c_2 = \frac{2}{19}$, and $c_3 = \frac{27}{256}$. All of these are fractions in their lowest terms, and the denominators are strictly increasing.

4.3 The Discrete Log Problem

Like factoring, the discrete logarithm problem may be solved through order finding. A similar quantum algorithm to the one presented in the previous section can solve instances in probabilistic polynomial time.

The computational intractability of the discrete log problem is relied upon for exchanging secret, randomly-chosen integers during Diffie-Hellman key agreement. Key agreement algorithms are not for exchanging arbitrary messages, but are used by two parties to establish a shared secret key over a public communication channel. This allows parties with no previous contact to use symmetric-key encryption schemes. There are variants of the Diffie-Hellman key agreement method, including ElGamal key agreement, which are vulnerable to similar quantum algorithms.

For context, we provide the steps of Diffie-Hellman key agreement. (C.f. [14][Sec. 12.6.1].)

Diffie-Hellman key agreement

Initial Conditions: Two parties A and B who may communicate over a public channel.

Result: A and B have a shared secret key K. I.e., they both know a number K which an observer who intercepts all their communications cannot know.

Step 1: The two parties agree on a prime p and an integer g, where $2 \leq g \leq p-2$. The chosen p and g are public knowledge.

Step 2: A chooses a random integer a, such that $1 \leq a \leq p-2$, and sends the value $g^a \pmod{p}$ to B.

B likewise chooses an integer b, where $1 \leq b \leq p-2$ and sends $g^b \pmod{p}$ to A.

The values a and b are known only by A and B, respectively.

Step 3: A has received $g^b \pmod{p}$ from B, and computes $(g^b)^a \pmod{p}$. B has $g^a \pmod{p}$, and computes $(g^a)^b \pmod{p}$.

Both parties have arrived at the secret key

$$K \equiv g^{ab} \pmod{p}. \tag{4.33}$$

Any party with access to the channel used by A and B has p, g, $g^a \pmod{p}$, and $g^b \pmod{p}$. Without knowing a or b, it is difficult to find $g^{ab} \pmod{p}$. The fastest method to solve $g^a \pmod{p}$ for a, or likewise find b, is the General Number Field Sieve, discussed previously in Section 4.1 as it is also used for factoring [24].

4.4 Shor's Discrete Log Algorithm

We begin with the algorithm given by [20] for solving instances of the discrete logarithm problem. It is similar to the algorithm given for factoring, in that both rely on the QFT. As we will see, this algorithm has an additional register, and the QFT is applied to the first two registers.

Discrete Log Algorithm

Input: Integers p, g and x, where p is prime and g is a generator of $\mathbb{Z}_p^*$.

Output: The integer r, where $0 \leq r < p-1$, such that $g^r \equiv x \pmod{p}$.

Step 0: Let $q = 2^l$ where $l \in \mathbb{N}$ and $p < q < 2p$. Prepare three l-qubit quantum registers in the state $|0\rangle |0\rangle |0\rangle$.

Step 1: In the first two registers, create a uniform superposition of the states $|0\rangle$ to $|p-2\rangle$.

Step 2: Apply the transformation

$$|a\rangle |b\rangle |0\rangle \rightarrow |a\rangle |b\rangle |g^a x^{-b} \pmod{p}\rangle , \tag{4.34}$$

resulting in the new state

$$\frac{1}{p-1} \sum_{a=0}^{p-2} \sum_{b=0}^{p-2} |a\rangle |b\rangle |g^a x^{-b} \pmod{p}\rangle . \tag{4.35}$$

Step 3: Apply the QFT to the first and second registers. The transformation on a basis state $|a\rangle |b\rangle$ is

$$|a\rangle |b\rangle \rightarrow \frac{1}{q} \sum_{c=0}^{q-1} \sum_{d=0}^{q-1} \exp\left(\frac{2\pi i}{q}(ac+bd)\right) |c\rangle |d\rangle . \tag{4.36}$$

When applied to the state from the previous step, the new state is

$$\frac{1}{(p-1)q} \sum_{a,b=0}^{p-2} \sum_{c,d=0}^{q-1} \exp\left(\frac{2\pi i}{q}(ac+bd)\right) |c\rangle |d\rangle |g^a x^{-b} \pmod{p}\rangle . \tag{4.37}$$

Step 4: Measure each register using the computational basis.

After the measurement takes place, we receive three integers c, d, and y. The integers returned by measuring the first and second registers are in the range $[0, q-1]$, while the result of measuring the third register is in the range $[0, p-1]$.

We will first consider the outcome of measuring the third register. Our goal is to find the probability that this register returns some given y.

Since g is a generator for the group $\mathbb{Z}_p^*$, we know

$$g^a x^{-b} \equiv g^k \pmod{p} \tag{4.38}$$

for some k, where $1 \leq k \leq p-1$. Since we are looking for x to satisfy $g^r \equiv x \pmod{p}$, we can rewrite (4.38) as $g^a g^{-rb} \equiv g^k \pmod{p}$. Therefore, we require

$$a - rb \equiv k \pmod{p-1}. \tag{4.39}$$

Finally, we let

$$y := g^k \pmod{p}.$$

We consider the probability amplitude of a given state $|c\rangle\,|d\rangle\,|y\rangle$. We must consider terms in the sum (4.37) where c and d are as specified, and $a - rb \equiv k \pmod{p-1}$. The probability amplitude of $|c\rangle\,|d\rangle\,|y\rangle$ is

$$Y = \frac{1}{(p-1)q} \sum_{\substack{a,b \\ a-rb \equiv k}} \exp\left(\frac{2\pi i}{q}(ac+bd)\right). \tag{4.40}$$

To eliminate the sum taken over a, we rewrite (4.40) using the relationship between a,b,k, and p. By rearranging (4.39), we can say that

$$a \equiv k + rb \pmod{p-1}. \tag{4.41}$$

In other words, our sum is over all integers a of the form

$$a = k + rb + l(p-1). \tag{4.42}$$

Since a,b,k, and r are all positive and less than p, $a < k + rb + (p-1)$. As a result, l must be zero or negative, so we can write

$$l = -\left\lfloor \frac{k+rb}{p-1} \right\rfloor. \tag{4.43}$$

Using this expression for l, we see that

$$a = k + rb - (p-1)\left\lfloor \frac{k+rb}{p-1} \right\rfloor. \tag{4.44}$$

This expression allows us to rewrite Y as defined by (4.40), expressing a in terms k, b, and r. We find that

$$Y = \frac{1}{(p-1)q} \sum_{b=0}^{p-2} \exp\left(\frac{2\pi i}{q}\left(brc + kc + bd - c(p-1)\left\lfloor \frac{br+k}{p-1} \right\rfloor\right)\right). \tag{4.45}$$

Now only one summation is needed, but we still wish to express this amplitude in a more compact format. Let T and V be defined as follows:

$$T = rc + d - \frac{r}{p-1}\{c(p-1)\}_q \tag{4.46}$$

$$V = \left(\frac{br}{p-1} - \left\lfloor \frac{br+k}{p-1} \right\rfloor\right)\{c(p-1)\}_q. \tag{4.47}$$

We use the notation $\{m\}_q$ to denote the value m' where $m' \equiv m \pmod q$ and $-\frac{q}{2} < m' < \frac{q}{2}$. In other words, m' is the integer equivalent to $m \pmod q$ with the smallest absolute value.

To see how T and V are substituted into (4.45), consider the equivalent expression

$$\frac{1}{(p-1)q}\sum_{b=0}^{p-2}\exp\left(\frac{2\pi i}{q}kc\right)\exp\left(\frac{2\pi i}{q}\left([brc+bd] + \left[-c(p-1)\left\lfloor\frac{br+k}{p-1}\right\rfloor\right]\right)\right). \tag{4.48}$$

We take a moment to rewrite the term $-c(p-1)\left\lfloor\frac{br+k}{p-1}\right\rfloor$. We can write $c(p-1)$ using its residue $\pmod q$, by choosing an integer R such that

$$c(p-1) = Rq + \{c(p-1)\}_q. \tag{4.49}$$

The term Rq may be ignored, because $\exp(2\pi i Rq/q)$ is equal to one.

If we also factor the term $\exp\left(\frac{2\pi i}{q}kc\right)$ out of the sum, we are left with

$$\frac{\exp\left(\frac{2\pi i}{q}kc\right)}{(p-1)q}\sum_{b=0}^{p-2}\exp\left(\frac{2\pi i}{q}\left([brc+bd] + \left[-\left\lfloor\frac{br+k}{p-1}\right\rfloor\{c(p-1)\}_q\right]\right)\right). \tag{4.50}$$

Because we are interested in the complex modulus of this expression, we may remove the coefficient $\exp\left(\frac{2\pi i}{q}kc\right)$. The complex modulus of this coefficient is one.

By adding $\frac{br}{p-1}\{c(p-1)\}_q$ to the second expression in brackets, and subtracting it from the first, we can then substitute in V and T, yielding

$$Y = \frac{1}{(p-1)q}\sum_{b=0}^{p-2}\exp\left(\frac{2\pi i}{q}bT\right)\exp\left(\frac{2\pi i}{q}V\right). \tag{4.51}$$

We define $W = \frac{p-2}{q}\{T\}_q$, and observe that we may write W as

$$W = \frac{p-2}{q}\left(rc + d - \frac{r}{p-1}\{c(p-1)\}_q - jq\right), \tag{4.52}$$

where the closest integer to T/q is defined to be j. By comparison with (4.46), we see that the expression in parenthesis is indeed $\{T\}_q$, so $W = \frac{p-2}{q}\{T\}_q$.

We will focus on measurement result pairs c and d which satisfy two conditions. These are defined below.

Definition 4.3. A measurement result (c, d) observed in the first two registers is *suitable* if the following two conditions are satisfied:

1.
$$|\{T\}_q| = \left| rc + d - \frac{r}{p-1}\{c(p-1)\}_q - jq \right| \leq \frac{1}{2} \tag{4.53}$$

2.
$$|\{c(p-1)\}_q| \leq \frac{q}{12} \tag{4.54}$$

These inequalities are referenced as **Condition 1** and **Condition 2**, respectively.

In the next lemma, we consider equivalence of real numbers modulo 2π. We say that $a \equiv b \pmod{2\pi}$ if $\exp(2\pi i a) = \exp(2\pi i b)$. Also, when referring to a quantity "$a \mod 2\pi$", this is the value in the range $(-\pi, \pi]$ which is equivalent to $a \pmod{2\pi}$. At the conclusion of the lemma, we compare values modulo 2π according to this choice.

Definition 4.4. The *phase* of a complex number expressed as $ze^{i\theta}$, for $z, \theta \in \mathbb{R}$ is the value in the range $(-\pi, \pi]$, which is equivalent to θ modulo 2π.

Lemma 4.4. *If Condition 1 holds, then as b takes integer values over the interval $[0, p-2]$, the phase of $\exp\left(\frac{2\pi i}{q} bT\right)$ varies from 0 to $2\pi W$.*

Proof. Substituting from (4.46),

$$\frac{2\pi bT}{q} = \frac{2\pi b}{q}\left(rc + d - \frac{r}{p-1}\{c(p-1)\}_q\right). \tag{4.55}$$

Add and subtract $2jb\pi$ from the right side to find that

$$\frac{2\pi bT}{q} = \frac{2\pi b}{q}\left(rc + d - \frac{r}{p-1}\{c(p-1)\}_q - jq\right) + 2jb\pi. \tag{4.56}$$

Since $2jp\pi$ is a multiple of 2π, we have

$$\frac{2\pi bT}{q} \equiv \frac{p-2}{p-2}\frac{2\pi b}{q}\left(rc + d - \frac{r}{p-1}\{c(p-1)\}_q - jq\right) \pmod{2\pi}. \tag{4.57}$$

Substitute in W from (4.52) to see that

$$\frac{2\pi bT}{q} \equiv \left(\frac{b}{p-2}\right) 2\pi W \pmod{2\pi}. \tag{4.58}$$

Since $W = \frac{p-2}{q}\{T\}_q$, by Condition 1, $|W| \leq \frac{p-2}{2q}$. Therefore,

$$\left|\frac{b}{p-2}2\pi W\right| \leq |2\pi W| \leq \left|\frac{p-2}{q}\pi\right| < \pi. \tag{4.59}$$

As a result, $\frac{b}{p-2}2\pi W$ is equal to $\frac{b}{p-2}2\pi W$ modulo 2π, and $2\pi W$ is equal to $2\pi W$ modulo 2π.

From (4.58), we know that $\frac{2\pi bT}{q}$ modulo 2π is equal to $\frac{b}{p-2}2\pi W$ modulo 2π. We conclude that $\frac{2\pi bT}{q}$ modulo 2π is smaller in magnitude than $|2\pi W|$. Therefore, the phase of $\exp\left(\frac{2\pi i}{q}bT\right)$ varies from 0 to $2\pi W$. □

Lemma 4.5. *The component of* $\exp\left(\frac{2\pi i}{q}bT\right)$ *in the direction of* $\exp(\pi i W)$ *is equal to* $\cos\left(2\pi\left|\frac{W}{2} - \frac{Wb}{p-2}\right|\right)$.

Proof. The component of $\exp\left(\frac{2\pi i}{q}bT\right)$ in the direction of $\exp(\pi i W)$ is equal to the cosine of the angle θ between the two complex numbers, as $\exp\left(\frac{2\pi i}{q}bT\right)$ is a unit vector. We find θ by considering the difference in phases $\left|W\pi - \frac{2\pi}{q}bT\right|$.

Since $\frac{2\pi}{q}bT \equiv \frac{2\pi}{q}b\{T\}_q \pmod{2\pi}$,

$$\left|W\pi - \frac{2\pi}{q}bT\right| \equiv \left|W\pi - \frac{2\pi}{q}b\{T\}_q\right| \pmod{2\pi} \tag{4.60}$$

Factoring out 2π and substituting in W, we find that

$$\left|W\pi - \frac{2\pi}{q}b\{T\}_q\right| = 2\pi\left|\frac{W}{2} - \frac{bW}{p-2}\right| \tag{4.61}$$

Since $\left|W\pi - \frac{2\pi}{q}bT\right| \equiv 2\pi\left|\frac{W}{2} - \frac{bW}{p-2}\right| \pmod{2\pi}$, $\theta = 2\pi\left|\frac{W}{2} - \frac{bW}{p-2}\right|$, and the lemma is proven. □

Lemma 4.6. *If Condition 2 holds, then* $\left|\frac{2\pi V}{q}\right| < \frac{\pi}{6}$.

Proof. From (4.47), $V = \left(\frac{br}{p-1} - \left\lfloor\frac{br+k}{p-1}\right\rfloor\right)\{c(p-1)\}_q$. Since $|\{c(p-1)\}_q| \leq \frac{q}{12}$, it remains to show that $\left|\frac{br}{p-1} - \left\lfloor\frac{br+k}{p-1}\right\rfloor\right| < 1$ to find $|V| < \frac{q}{12}$.

Since k is defined in $a - br \equiv k \pmod{p-1}$, k takes integer values in the range $[0, p-2]$. From the definition of a, a takes integer values in the same range. Since $a \equiv br+k \pmod{p-1}$, it follows that

$$a + \left\lfloor\frac{br+k}{p-1}\right\rfloor(p-1) = br + k. \tag{4.62}$$

Rearrange terms to find

$$\frac{a-k}{p-1} = \frac{br}{p-1} - \left\lfloor \frac{br+k}{p-1} \right\rfloor. \tag{4.63}$$

Since $|a-k| < p-1$, we have $\left|\frac{a-k}{p-1}\right| < 1$. That is,

$$\left| \frac{br}{p-1} - \left\lfloor \frac{br+k}{p-1} \right\rfloor \right| < 1. \tag{4.64}$$

Then $|V| < \frac{q}{12}$, and $\left|\frac{2\pi V}{q}\right| < \frac{\pi}{6}$. □

We use the lemmas established so far to find the minimum probability of measuring a given suitable measurement result.

Lemma 4.7. *If a measurement result (c, d) is suitable, then its amplitude has the lower bound*

$$\frac{1}{(p-1)q} \sum_{b=0}^{p-2} \cos\left(2\pi \left| \frac{W}{2} - \frac{Wb}{p-2} \right| + \frac{\pi}{6}\right).$$

Proof. Let the amplitude be expressed as in (4.51):

$$Y = \frac{1}{(p-1)q} \sum_{b=0}^{p-2} \psi_b,$$

where

$$\psi_b = \exp\left(\frac{2\pi i}{q} bT\right) \exp\left(\frac{2\pi i V}{q}\right).$$

We can express ψ_b as a real linear combination of unit complex numbers, $\exp(\pi i W)$ and $\exp(\pi i W + \frac{\pi i}{2})$ for some α_b and β_b. Specifically,

$$\psi_b = \alpha_b \exp(\pi i W) + \beta_b \exp(\pi i W + \frac{\pi i}{2}).$$

Then Y can be expressed as

$$Y = \frac{1}{(p-1)q} \sum_{b=0}^{p-2} \left(\alpha_b \exp(\pi i W) + \beta_b \exp(\pi i W + \frac{\pi i}{2}) \right). \tag{4.65}$$

As Y is now expressed using the sums of orthogonal components, $|Y|$ is bounded below by the complex moduli of these components.

Using Lemma 4.5, the complex modulus of the component $\exp\left(\frac{2\pi i}{q} bT\right)$ in the direction $\exp(\pi i W)$ is $\cos\left(2\pi \left| \frac{W}{2} - \frac{Wb}{p-2} \right|\right)$.

One easily observes that Condition 1 and $W = \frac{p-2}{q}\{T\}_q$ imply $|W| < \frac{1}{2}$.

The angle $2\pi\left|\frac{W}{2}-\frac{Wb}{p-2}\right|$ can be expressed equivalently as $2\pi|W|\left|\frac{1}{2}-\frac{b}{p-2}\right|$. Since $|W|$ is less than $\frac{1}{2}$, this angle is strictly less than $\pi\left|\frac{1}{2}-\frac{b}{p-2}\right|$. Given that $0\leq b\leq p-2$, $|\frac{1}{2}-\frac{b}{p-2}|\leq\frac{1}{2}$. We can then state that

$$2\pi\left|\frac{W}{2}-\frac{Wb}{p-2}\right|<\frac{\pi}{2}.$$

Since cosine is decreasing from 0 to π radians,

$$\cos\left(2\pi\left|\frac{W}{2}-\frac{Wb}{p-2}\right|\right)\geq\cos\left(2\pi\left|\frac{W}{2}-\frac{Wb}{p-2}\right|+\frac{\pi}{6}\right).$$

As the right-hand side of this inequality is a lower bound for the component of ψ_b in the direction $\exp(\pi iW)$, we can say that $\alpha_b\geq\cos\left(2\pi\left|\frac{W}{2}-\frac{Wb}{p-2}\right|+\frac{\pi}{6}\right)$. The amplitude of Y will be greater than or equal to the magnitude of its component in the direction $\exp(\pi iW)$. We use the previous inequality, a lower bound on α_b, to compute the amplitude of the aforementioned component of Y. We can state that

$$|Y|\geq\frac{1}{(p-1)q}\sum_{b=0}^{p-2}\cos\left(2\pi\left|\frac{W}{2}-\frac{Wb}{p-2}\right|+\frac{\pi}{6}\right). \tag{4.66}$$

If the result of a measurement is suitable, we can use the above equation for $|Y|$ to find the minimum probability of measuring it. □

We now wish to transform the result from Lemma 4.7 into an integral. Discussion of the associated error term can be found in Appendix C. Let $u=\frac{1}{2}-\frac{b}{p-2}$, such that $u|_{b=0}=1/2$ and $u|_{b=p-2}=-1/2$. We can then rewrite the sum as an integral with an error term

$$|Y|\geq\frac{1}{(p-1)q}\int_{1/2}^{-1/2}\cos\left(2\pi|W|\,|u|+\frac{\pi}{6}\right)(-du)+\mathcal{O}\left(\frac{W}{pq}\right).$$

It follows from the properties of integrals and that u is symmetric about 0 that

$$|Y|\geq\frac{2}{q}\int_{0}^{1/2}\cos\left(2\pi|W|u+\frac{\pi}{6}\right)du+\mathcal{O}\left(\frac{W}{pq}\right). \tag{4.67}$$

It is easily checked that the integrand is minimized when $|W|<1/2$. Using this and another change of variables defined as $u'=\pi u+\frac{\pi}{6}$, we obtain the desired lower bound for $|Y|$,

$$|Y|\geq\frac{2}{q\pi}\int_{\pi/6}^{2\pi/3}\cos u'du'+\mathcal{O}\left(\frac{W}{pq}\right). \tag{4.68}$$

Lemma 4.8. *If integers c',d' in the range $[0,q-1]$ satisfy Conditions 1 and 2, then the probability of measuring the quantum register and finding a result (c,d,y) such that $c=c'$ and $d=d'$ is at least $\frac{1}{40q}$.*

Proof. We find that $|Y| > \frac{1}{20q^2}$, implying that for a given pair c', d', where both values are integers in the range $[0, q-1]$ and satisfy the two conditions, the probability of measuring a state where $c = c'$ and $d = d'$ is at least $\frac{1}{20q^2}$. When the register is measured, the result is of the form c, d, y where y is an integer in the range $[0, p-2]$, so there are $p-1$ possible measurement outcomes of the form c', d', y. As each state of this form has a probability of at least $\frac{1}{20q^2}$ of being measured, the probability of finding $c = c'$ and $d = d'$ after a measurement is at least $\frac{p-1}{20q^2}$. As q was chosen such that $p < q < 2p$, $q < 2p$ which implies $\frac{1}{2} \leq \frac{p-1}{q}$. Therefore, $\frac{p-1}{20q^2} \geq \frac{1}{40q}$. □

Lemma 4.9. *The integer c may take at least $\frac{q}{12}$ of the values in the range $[0, q-1]$ and satisfy Condition 2, $|\{c(p-1)\}_q| \leq \frac{q}{12}$.*

Proof. Consider the additive cyclic group $\mathbb{Z}_q$. The elements in this group are the values c assumes, the integers $0, \ldots, q-1$.

The product $c(p-1)$, for each c, is mapped to an element of $\mathbb{Z}_q$ by modular reduction. The elements in $\mathbb{Z}_q$ which are equal to the modulus of $c(p-1)$ for some c form a subgroup of $\mathbb{Z}_q$. This subgroup is generated by $p-1$.

By Lagrange's theorem for finite abelian groups, the order of a subgroup must divide the order of a group [16][P. 63]. Using this property, the order of $\mathbb{Z}_q$ is q, a power of 2, so the order of any subgroup must also be a power of 2. Let $q = 2^l$, and the order of the subgroup generated by $p-1$ be $|H| = 2^j$ where $0 \leq j \leq l$.

By a result from [16][P. 52], any subgroup of $\mathbb{Z}_q$ has as a generator $\bar{g}$ where $\bar{g} = \frac{n}{|H|}$. As $|H| = 2^j$, $\mathbb{Z}_q$ has a generator $\bar{g} = 2^{l-j}$.

As c takes each integer value from 0 to $q-1$, each element of H appears $\frac{q}{|H|}$ times as $c(p-1) \mod q$ is computed. The result of $\{c(p-1)\}_q$ is equivalent modulo q.

To find at least $\frac{q}{12}$ of the possible values for c satisfy Condition 2, it is sufficient to find at least $\frac{q}{8}$ values satisfy this requirement. To show a value of c satisfies Condition 2, it is sufficient to show that $|\{c(p-1)\}_q| \leq \frac{q}{16}$. We will show 2^{l-3} possible values for c satisfy the condition $|\{c(p-1)\}_q| \leq 2^{l-4}$.

We split this proof into two cases: $2^j < 12$ and $2^j > 12$. The equality case will never occur, as 12 is not a power of 2.

In the first case, $2^j < 12$, meaning $j \leq 3$. As 2^j is the order of H, we can say H contains 8 or fewer elements. Note that $0 \in H$. We have one element out of at most 8 which satisfies Condition 2. We can then say that at least $\frac{q}{|H|} = \frac{q}{2^j} \geq 2^{l-3}$ of possible values for c satisfy Condition 2.

In the second case, $2^j > 12$, meaning $j \geq 4$.

If we express all elements of H using $\bar{g}$ we find

$$H = \left\{0, \bar{g}, 2\bar{g}, \ldots, \left(2^j - 1\right)\bar{g}\right\}. \tag{4.69}$$

The elements of H can be expressed as $2^{l-j}t$, for $t \in \mathbb{Z}$ and $0 \leq t \leq 2^j - 1$. To find elements less than or equal to 2^{l-4}, we choose t so that $1 \leq t \leq 2^{j-4}$. As this range contains 2^{j-4} values of t, there are an equal number of elements in H shown to satisfy Condition 2 thus far.

The largest 2^{j-4} elements of H also satisfy Condition 2. As t is in the range $2^j - 2^{j-4} \leq t \leq 2^j - 1$, the value equal $\pmod q$ with smallest absolute value, the result of the $\{\cdot\}_q$ operation, is in the range $[-2^{l-4}, -2^{l-j}]$. We have identified another 2^{j-4} elements which satisfy Condition 2.

The value 0 also satisfies Condition 2, so we have a total of $2^{j-3} + 1$ elements from H. As each element of H appears for $\frac{q}{|H|} = 2^{l-j}$ possible values of q, this is a total of $2^{l-j}(2^{j-3} + 1) > 2^{l-3}$ values of c satisfying Condition 2.

We have established that for any nonnegative j, 2^{l-3} possible values of c satisfy the condition $|\{c(p-1)\}_q| \leq 2^{l-4}$. □

Combining the results from Lemmas 4.8 and 4.9, we find the probability that any good c is found to be at least $\frac{1}{40q} \cdot \frac{q}{12} = \frac{1}{480}$.

We now need to recover r from our output. To begin this process, consider the following rearrangement of Condition 1,

$$-\frac{1}{2} \leq d + r\left(c - \frac{\{c(p-1)\}_q}{p-1}\right) - jq \leq \frac{1}{2}. \tag{4.70}$$

Then by combining the term in the parenthesis and dividing by q, we find

$$-\frac{1}{2q} \leq \frac{d}{q} + r\left(\frac{c(p-1) - \{c(p-1)\}_q}{(p-1)q}\right) - j \leq \frac{1}{2q}. \tag{4.71}$$

Now define

$$c' := \frac{c(p-1) - \{c(p-1)\}_q}{q}. \tag{4.72}$$

It is clear that c' is an integer because $\{c(p-1)\}_q$ is the residue of $c(p-1) \pmod q$. Additionally, we note that

$$\frac{c'}{p-1} = \frac{c}{q} - \frac{\{c(p-1)\}_q}{(p-1)q}, \tag{4.73}$$

so Condition 2 tells us that $\frac{c}{q}$ is within $\frac{1}{12(p-1)}$ of $\frac{c'}{p-1}$ for suitable c, and thus every suitable c is uniquely mapped to a c'.

With c' defined, we can express (4.71) as

$$-\frac{1}{2q} \leq \frac{d}{q} + \frac{rc'}{(p-1)} - j \leq \frac{1}{2q}. \tag{4.74}$$

We can rewrite this in a form reminiscent of (4.8) by returning to the absolute value representation,

$$\left|\frac{d}{q} + \frac{rc'}{(p-1)} - j\right| \leq \frac{1}{2q}. \tag{4.75}$$

To recover r, we will want to round $\frac{d}{q}$ to the closest fraction of the form $\frac{d'}{p-1}$ and then divide by the integer $c' \pmod{(p-1)}$. We need c' coprime to $(p-1)$ to avoid the second

term simplifying, which would result in needing to round $\frac{d}{q}$ to a different fraction. The result of this is a guess at r of $r \equiv d'c'^{-1} \mod p-1$, where c'^{-1} is the inverse of c' modulo $(p-1)$. We cannot be certain that the resultant r is the order we are looking for as we are not guaranteed that c' is coprime to $(p-1)$, so we must use the Chinese Remainder Theorem (CRT).

Theorem 4.10 (Chinese Remainder Theorem, [14])**.** *Given k pairwise relatively prime integers $n_1, n_2, \ldots, n_k$, the system*

$$\begin{aligned} x &\equiv a_1 \ (mod\ n_1) \\ x &\equiv a_2 \ (mod\ n_2) \\ &\vdots \\ x &\equiv a_k \ (mod\ n_k) \end{aligned}$$

has a unique solution modulo $n = n_1 n_2 \cdots n_k$.

The solution can be found using the method presented in [14] to be $\sum_{i=1}^{k} a_i N_i M_i \pmod{n}$, where $N_i = \frac{n}{n_i}$ and M_i is the inverse of $N_i \pmod{n_i}$.

To apply this theorem, we need to consider the probability that c' is coprime to $(p-1)$ to determine how many times we must run the algorithms before obtaining enough distinct c' values to construct the equivalence relation over prime powers of $(p-1)$. Because a suitable c yields exactly one c', we will consider such c' as suitable as well. Because at least $1/12$ of the c are suitable, it must also be true that at least $1/12$ of the c' are also suitable. If c' is to be coprime to $(p-1)$, it must be true that for any prime power $p_i^{\alpha_i}$ which divides $(p-1)$, $p_i^{\alpha_i}$ does not divide c'. The probability that some integer κ divides a random integer γ is known to be $\frac{1}{\kappa}$. Using this and the frequency of good c', we can bound the probability of some suitable c' being divisible by $p_i^{\alpha_i}$ above by $12/p_i^{\alpha_i}$ provided we restrict ourselves to only consider the prime powers greater than 18. The choice of this bound will be discussed later. There are only 11 prime powers less than 18, so this restriction will result in only a large constant increase in run time as we would need to check all possible residues from these values individually.

With this probability, we can find the probability a suitable c' will have one of the $p_i^{\alpha_i}$ divide it to be

$$\sum_{18<p_i^{\alpha_i}|(p-1)} \frac{12}{p_i^{\alpha_i}}, \tag{4.76}$$

the sum of the probabilities of each prime power. We can generalize this to find the probability that a set of t suitable c' share some $p_i^{\alpha_i}$ factor to be

$$\sum_{18<p_i^{\alpha_i}|(p-1)} \left(\frac{12}{p_i^{\alpha_i}}\right)^t. \tag{4.77}$$

The exponent arises because the probability of measuring some c across multiple iterations of the algorithm is disjoint, so specific c' are as well. Because they are disjoint, the probability is found by considering the product of all of our c', of which we have t.

The importance of the restriction on only considering prime powers greater than 18 is now apparent. The ratio we are considering is restricted by it to be less than $2/3$, and for a constant $t \geq 2$, the fraction will be less than $1/2$. As we consider a larger set of c', we can guarantee that this probability is less than $1/2$, as for each increase in t the ratio is scaled by a factor less than $2/3$.

After determining a sufficient t, we can determine how many experiments must be done to, with high probability, obtain this many distinct c'. The probability measuring a suitable c was determined to be at least $1/480$, so we can expect that we would need at least $480t$ experiments to likely have t distinct c', as suitable c' occur with equal probability. We can then find some c' in the set of size t such that some $p_i^{\alpha_i}$ is coprime to the c' for all $p_i^{\alpha_i}$. For these sets of coprime c's and $p_i^{\alpha_i}$s, we will define $r_i := d_i' c_i'^{-1}$, where the index i denotes a pairing with $p_i^{\alpha_i}$ and $c_i'^{-1}$. We can then set up an equivalence relation $r \equiv r_i \pmod{p_i^{\alpha_i}}$ for all i. We now have a set of equivalence relations over the prime powers of $(p-1)$, and can apply the CRT to find r.

Chapter 5

Computational Complexity

In this chapter, our goal is to assess the potential of quantum computers in terms of complexity theory. Up to this point, we have considered certain efficient algorithms which have been developed. We now consider, in a sense, the sets of problems for which efficient algorithms can be found.

In this chapter, we explain decision problems, and give a notion of what a Turing machine is. With this background, we define several important classical complexity classes, and then introduce some quantum complexity classes. Our objective is to explore how these classes fit into the classical complexity class hierarchy.

We begin with the notion of a decision problem, and its relationship to languages. We define an *alphabet* as some set of symbols Σ. The set Σ^* is all finite sequences of symbols from Σ. (Cf. [21][Sec. 2.1].)

Definition 5.1. Given an alphabet Σ, a *language* is a subset of Σ^*.

As a brief example, let $\Sigma = \{0, 1\}$. In this case, Σ^* may be thought of as all finite-length binary strings. One possible language we could consider is the set of all finite-length binary strings with an even number of 1's. Another possible language is the the set of finite-length binary strings containing the string '01011'.

These examples hint at the relationship between a decision problem and a language. Given a binary string, one could ask, "Does this string contain an even number of 1's?". If the answer is yes, then the language defined to include all binary strings with an even number of 1's would include this string. This language includes precisely the (finite-length) binary strings for which the answer to the aforementioned question is "yes". Keeping this idea in mind, we define a decision problem, as done by [21][P. 343].

Definition 5.2. A *decision problem* P is a set of related questions, where each question has a yes or no answer. A question from the set P is an *instance* of the decision problem.

One example of a decision problem is whether a natural number is a multiple of 7. Each natural number corresponds to an instance of this decision problem. We can denote by p_0 the question "Is 0 a multiple of 7?", and p_1 the question "Is 1 a multiple of 7?", and so on.

We say that a *solution* to a decision problem is an algorithm to correctly determine the answer to every instance of the problem. Likewise, we can refer to a decision problem as *solvable*.

We consider Turing machines because decision problems are classified based on, informally, whether and how they are answered by a Turing machine. We use this theoretical construct to simulate what any computing device is able to accomplish, for reasons explained by the Church-Turing Thesis. A complete discussion of this topic is far outside what can be explained in a chapter of this report, but the interested reader may consult the treatment by [21][Chap. 11] for an approachable explanation.

A Turing machine may be thought of as a head capable of reading and writing symbols to an infinite tape. The head will, at any given time, be in one of a finite number of states. At the start of execution, there will be some input written on the tape. In each movement, the head will print a symbol onto the tape, optionally move one symbol-width to the left or right, and change state. The machine will halt if it reaches one of two special states, called the accept and reject states. Depending on which of these states was reached, the given input is either accepted or rejected. For some inputs, the Turing machine may never halt.

Inputs to a Turing machine are expressed as a string from a specified alphabet that the machine is designed to consider. A language is said to be *accepted* by a Turing machine if the set of words in the language is equal to the set of words for which the Turing machine halts in the accept state.

To connect this to our earlier examples, a Turing machine with *input alphabet* $\{0, 1\}$ might be designed to accept the language of binary strings containing an even number of 1's. For this question, we could specify a machine such that, if the input word met this criteria, then the machine would eventually halt and do so in the accept state. For a string with an odd number of 1's, it would halt in a reject state. We can say that this machine accepts the language of binary strings with an even number of 1's.

To find the answer to instances of a decision problem using a Turing machine, instances of the problem must be represented using the input language of the Turing machine. For our 'multiple of 7' decision problem, a reasonable choice would be to design a Turing machine with the input alphabet $\{0, 1\}$, and use the binary representation of each natural number to represent that instance of the decision problem for the Turing machine.

We need to consider a few types of Turing machines to understand the complexity classes in the next section. A deterministic Turing machine (DTM), given that it is in a certain state and reads a certain symbol from the tape, has only one possible action to take. On the other hand, a nondeterministic Turing machine (NTM) may have more than one possible action. For a certain state, and a certain symbol on the tape, a nondeterministic Turing machine might choose between printing a symbol and moving right, or leaving the symbol as-is and moving left, for example. A probabilistic Turing machine (PTM) is a nondeterministic Turing machine where there is a given probability distribution for each situation with multiple options.

5.1 Classical Complexity Classes

We begin with a general explanation of complexity, an abridged version of the treatment by [21][Sec. 14.1]. Complexity is a measurement of the resources used when solving a problem. The resource might be the number of instructions executed, arithmetic operations performed, amount of memory used, and so on. There are three components we might consider when analyzing the complexity of an algorithm: the resource we are measuring, a partition of input instances by their complexity, and a function to map input complexity to resource usage.

In our 'multiples of 7' decision problem, we could measure the number of arithmetic operations performed, partition the input instances by the length of the binary representation of the number, and determine a function to relate the length of the binary representation to the number of arithmetic operations needed. This is an example of time complexity, intuitively a measure of the 'work' involved in a computation.

For a Turing machine, we can measure time complexity as the number of transitions, essentially steps, carried out by the machine.

Definition 5.3. Let M be a standard Turing machine. The *time complexity* of M is defined by the function $tc_M : \mathbb{N} \to \mathbb{N}$, where $tc_m(n)$ is the maximum number of transitions processed by a computation of M, when n is the length of the input string [21][P. 443].

We will consider several commonly referenced classical complexity classes, almost all of which are measures of time complexity. The class **P** is the set of decision problems which are solvable by a deterministic Turing machine in time complexity expressible as a polynomial function. **P** is contained in every other complexity class we will discuss. As a shorthand, we say that a problem is solvable in polynomial time if there is a Turing machine which solves the relevant problem with time complexity expressible as a polynomial function.

Probabilistic Turing machines lead to algorithms which are capable of giving correct responses, as well as incorrect ones. For example, if a problem is in the class **RP**, which we do not focus on but mention as it illustrates this circumstance, then there is a PTM for which the following conditions hold:

1. Instances where the answer is "yes" are accepted with probability $\geq \frac{1}{2}$.
2. Instances where the answer is "no" are always rejected.

As an aside, the class **ZPP** in the table below is the intersection of **RP** and **co-RP**. The class has 'average-case' polynomial time, instead of simply 'polynomial time,' because it may take multiple attempts to receive an answer with certainty for a given instance. For problems in **RP**, an answer of "yes" means that the correct answer for the decision problem has been found. With an answer of "no", however, the correct response is still unknown. For problems in **co-RP**, answers of "no" are guaranteed to be correct, while answers of "yes" have this uncertainty. With one PTM for each of these sets of characteristics, the correct response may be found with certainty, but the number of attempts is unknown. However, the probability that, for **RP**, answers of "yes" are correct at least half the time, and analogously for "no" in **co-RP**, allows for reasoning about the number of attempts needed to reach a solution with

certainty. For problems in **ZPP**, the average-case time complexity is a polynomial function, where the phrase *average-case* indicates an average taken over all possible inputs.

The definitions of the relevant complexity classes are included in the following table.

Complexity Class	Attributes of the set of decision problems
P	Polynomial Time: Solvable by a DTM in polynomial time.
ZPP	Zero-Error Probabilistic Polynomial Time: Solvable by a PTM in average-case polynomial time.
BPP	Bounded-Error Probabilistic Polynomial Time: There is a PTM in polynomial time with probability $> \frac{2}{3}$ of a correct response.
NP	Nondeterministic Polynomial Time: If the answer to an instance is "yes", then given the instance and some input representing the solution, the answer may be verified by a DTM in polynomial time.
Co-NP	Complement of NP: If the answer to an instance is "no", then given the instance and solution, the answer may be verified by a DTM in polynomial time.
PP	Probabilistic Polynomial Time: There is a PTM which will, in polynomial time, accept instances where the answer is "yes" with probability $> \frac{1}{2}$, and accept instances where the answer is "no" with probability $\leq \frac{1}{2}$.
PSPACE	Solvable by a DTM with space complexity expressible as a polynomial function.

The relationships between these classes are not entirely known. Whether or not **P=NP** is a famous unsolved problem in computer science. We can say that

$$\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \mathbf{BPP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}.$$

As for **NP** and **Co-NP**, the intersection of these two sets contains **ZPP**. Also, **NP** and **Co-NP** are contained in **PP**. Figure 5.1 includes some of these classes, as well as the most important quantum complexity class discussed in the next section.

5.2 Quantum Complexity Classes

The study of quantum Turing machines began with a paper from David Deutsch, on the capabilities of a quantum computer. Deutsch proposed a quantum Turing machine (QTM), and stated that such a machine would have "remarkable properties not reproducible by any Turing machine" [5]. Referring back to our intuitive notion of a Turing machine presented in the previous section, Deutsch's quantum Turing machine differs in that the symbols on the tape may exist in superpositions, analogous to our discussion in Chapter 2. Deutsch pointed out the massive parallelism afforded by these models, and explored the implications on the Church-Turing Thesis. We discuss the progress made to date in settling the question

of whether these additional capabilities enable a quantum Turing machine to exceed the capabilities of a classical one.

There are quantum analogues to certain classical complexity classes. For example, the class **P** of problems solvable by a DTM in polynomial time, has the counterpart **QP** of problems which can be solved with certainty on a QTM in polynomial time. Since a quantum Turing machine may be used to "perfectly" simulate a classical Turing machine, as realized by Deutsch, the class **P** is contained in the class **QP**. In other words, if an algorithm may be completed by a classical Turing machine in polynomial time, a quantum Turing machine can do the same. We discussed this in a more concrete sense in Chapter 2.

The following table defines some further quantum complexity classes [25][Sec. 4.4]

Complexity Class	Attributes of the set of decision problems
QP	Quantum Polynomial Time: Solvable by a QTM, with certainty, in polynomial time.
ZQP	Zero-Error Probabilistic Polynomial Time: Solvable by a QTM with error probability zero in average-case polynomial time.
BQP	Bounded-Error Quantum Polynomial Time: Solvable by a QTM in polynomial time with probability $> \frac{2}{3}$ of a correct response.

There are some known relationships between these classes and classical complexity classes. To start, **P**$\subset$**QP**. This is a strict inequality, established by Berthiaume and Brassard in 1992. Likewise, **ZPP**$\subset$**ZQP** is known to be a strict inclusion.

An important open question is where exactly **BQP** fits in with classical complexity classes. This class is essentially the problems which may be efficiently solved by a quantum computer [23]. It includes the algorithms for factoring and the discrete log problem discussed in Chapter 4, as well as finding solutions for Pell's equation, a problem with connections to algebraic number theory. It is known that **BPP**$\subseteq$**BQP**. However, it is not known whether this is a strict inclusion. Integer factorization is in **BQP**, as established by Shor, but whether there is an integer factorization algorithm in the class **BPP** is unknown.

The strongest statement to be made, in terms of a classical complexity class containing **BQP**, is that **BQP**$\subseteq$**PP**, according to the authors of [23] at time of writing.

Further refinements of the inclusion relationships between **BQP** and commonly used classical complexity classes would tell us what quantum computers are capable of achieving in polynomial time.

We have the following sequence of inclusion relationships established for **BQP**:

$$\mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PP}.$$

This relationship is expressed by Figure 5.1.

It is not known if **BPP** is equal to **BQP**, or whether **BQP** contains **NP** or vice versa. The answers to these questions would provide insight into the advantage quantum computers have over classical computers. If we find that **BQP**$\neq$**BPP**, this would mean that there are tasks which a quantum computer can accomplish in polynomial time with a bounded error probability, which cannot be accomplished by a classical computer with the same efficiency.

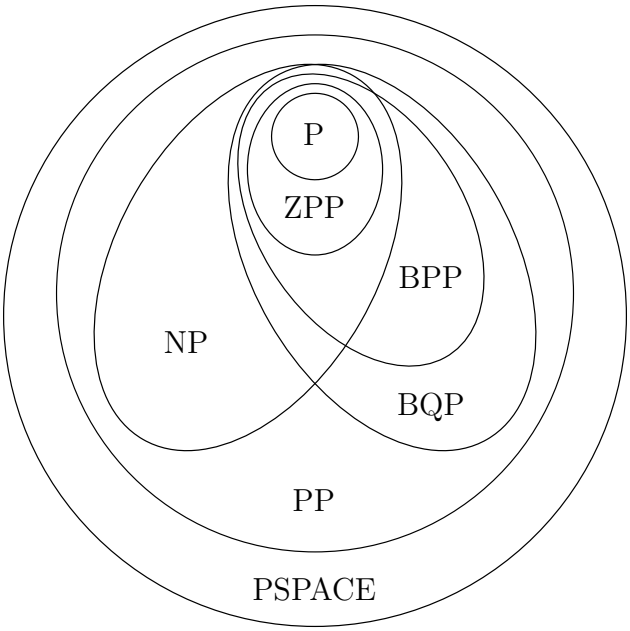


Figure 5.1: A representation of the inclusion relationships between complexity classes. It is not known whether **NP** is contained in **BQP**.

If **BQP** were found to contain **NP**, then many problems we currently consider hard to solve yet easy to verify could theoretically be handled efficiently by a quantum computer. To return to the questions posed in the introduction, problems which are in **NP** but not suspected to be in **P** are used in many asymmetric-key encryption schemes. These are open questions in computer science, and the outcomes are of great interest in finding the potential of quantum computers.

Chapter 6

Physical Implementations of Quantum Computers

The algorithms we have presented represent the theoretical groundwork for solving some of the most important problems in cryptography by means of a quantum computer. The existence of these algorithms motivates the effort to build hardware which can implement quantum computers. At present, there is still no publicly known functioning quantum computer, but the effort to build one represents a major field of research. The following will discuss the basic considerations required for creation of a quantum computer and the current state of development of some of the more promising approaches.

We gave earlier a mathematical definition of a qubit as some two dimensional vector space, $\mathbb{C}^2$. In the physical sense, a qubit is some quantum system always in some state expressed as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|0\rangle$ represents one basis state of the quantum system and $|1\rangle$ another, and coefficients α and β obey the normalization condition $\alpha\bar{\alpha}+\beta\bar{\beta} = |\alpha|^2+|\beta|^2 = 1$. If a measurement is made on state $|\psi\rangle$, the probability of finding it in state $|0\rangle$ is $|\alpha|^2$ and in state $|1\rangle$ is $|\beta|^2$. This probability of measuring a qubit in one of two potential states is the primary difference between classical bits and qubits. A classical bit will deterministically be either 0 or 1, and will not change unless acted upon. That is to say, if single bits are prepared from the same starting point in the same way, every time one of these bits are measured the result will be the same. On the other hand, qubits can exist in a superposition of both states until the state is measured, at which point the state collapses to one state or the other. This means that if we prepared multiple identical quantum systems in the same state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, every initial measurement could return either 0 or 1, even though the states and measurements are the same.

The quantum systems of interest in quantum computing are mostly finite quantum systems whose states exist in a finite-dimensional Hilbert space. The simplest of these systems is a two-state quantum system, or qubit, whose Hilbert space can be spanned by just two independent states. There exist many physical realizations of a qubit such as the spin of an electron, the polarization of a photon, or a pair of energy levels of an atom. For quantum computing, two state systems are of primary interest. Two state systems provide a convenient analog to classical computing as it means that measurement of the system will result

in only one of two possible results, analogous to a 0 or 1 in classical computing.

6.1 Requirements for a Quantum Computer

The issue of creating a physical quantum computer has problems that must be addressed and resolved. Five requirements for a physically realizable quantum computer are as follows: 1) scalability and well defined qubits, 2) ability to initialize the quantum register, 3) long decoherence time compared to switching time, 4) the ability to construct a universal set of quantum gates for the qubits used, and 5) the ability to measure the state of the qubits [6].

In order for a quantum computer to perform any nontrivial computation, the method by which it is implemented must allow it to be scalable. While there are numerous two level quantum systems that could be used to construct a functional qubit, not all of them are equally capable of being scaled up to a size that is useful for a nontrivial task. If a method for qubit creation requires large laboratory setups to implement a single qubit, it would be impractical to create a register of 2048 qubits to perform computations.

It is also necessary that any quantum computer be able to initialize to a starting state, typically $|\psi\rangle = |0\rangle \otimes |0\rangle \otimes \ldots$. Initialization is needed to allow definitive knowledge of the starting state of the quantum system. Without information of the starting state of the quantum register, any computation and measurement of the register would provide computationally useless results.

The issues of decoherence and switching times are of great importance and warrant further discussion over the previously mentioned requirements. Such discussion will follow this section.

The fourth requirement is the ability to perform the necessary processing operations on the system. In classical computing, operations are performed on bits by gates. Similarly, quantum gates are needed to manipulate qubits in a quantum computer. While classical computers allow the existence of irreversible computation at the cost of heat production, the laws of quantum mechanics require that all computations done by a quantum computer be reversible, as mentioned earlier in the paper. It turns out that any quantum gate can be approximated by combining a finite subset of gates. These gates consist of a set of building blocks for the set of all single qubit gates and a single type of two-qubit gate known as the quantum CNOT or XOR gate [6]. The ability to implement these gates reliably is crucial to the success of a quantum computer.

The final requirement is that the state of the machine (i.e. of all the qubits in it) must be capable of being reliably measured, so that the results of computations can be obtained. In practice, the ability to perform reliable measurements is not always guaranteed, giving rise to efficiency of measurements and measurement fidelity. While perfect efficiency and fidelity would be desirable, much less is actually needed to perform a reliable computation.

6.1.1 Decoherence and Switching Times

Two of the issues that must be addressed in creating a quantum computer are those of decoherence and switching times.

In classical computers, the state of a bit is typically realized by varying voltage or current levels. While errors can occur, it is unlikely that the bit will change its state as a result of small perturbation from its environment. Unfortunately, the quantum systems used for qubits are far more fragile. The fragility of the quantum system is described by the phenomenon of decoherence, by which exposure to environmental factors, such as ambient temperature or a background electromagnetic field, can degrade the quantum behavior of the system. The advantage that qubits have over classical bits is their ability to exist in a superposition of basis states. Decoherence undermines this by collapsing the state of the qubit to one of the basis states, causing it to behave like a classical bit. The time it takes a qubit to degrade from an initial superposition state to one of the distinct basis states is called the decoherence time.

A second useful metric for comparison of qubits is the switching time. In order to perform any manipulations on a qubit or quantum register, a gate must be applied to the system. The transformation of the quantum state due to the gate is not instantaneous but requires a non-zero time to change the state of the system. The time required to apply a gate operation to a qubit is the switching time. Switching times are dependent upon the physical system being used.

While the decoherence and switching times are two useful metrics for comparing the performance of different types of qubits, their ratio is even more significant. That is, one would like the time it takes to perform an operation on the system to be short compared to the time it takes for the quantum system to be destroyed as a result of decoherence. In practice, a good relationship between decoherence and switch times is that decoherence of the system should be 10^4 to 10^5 times the switch time [6], so that about 10^4 gates could be applied to the qubit before it has decohered.

6.2 Quantum Error Correction

A second type of error that can occur in a quantum computer arises from the imprecise action of its gates. Any gate applied to the qubits in a quantum register must be applied for some time t to fully complete its operation. This time cannot be precisely achieved in every application, this deviation from the ideal time t is one source of error. Dealing with these errors, and all other possible errors, is a task for the field of *quantum error correction*.

In general, we can state five criteria for a quantum computation to be considered fault tolerant, meaning it can be rescued successfully by quantum error correction. First, no qubit should be used in multiple computations. This will prevent an error in a single qubit from being propagated throughout the entire computation. It also illustrates the need for multiple ancillary qubits to be included in a register for every qubit used in the computation. Second, measurement of ancillary qubits should be done in such a way that the error is measured

without corrupting the information stored in the primary qubits. The third, fourth, and fifth criteria are concerned with the actual encoding of the information in the qubits. The third criterion is that known quantum states should be verified once they have been encoded, if possible. The fourth is that operations should be repeated, especially error checking operations. The final criterion is that the correct code be used for the computation. A more detailed discussion of these criteria and quantum error correction in general is given in [18].

We can take away from these criteria some important considerations related to physically realizing a quantum computer. The first criterion recognizes the need for more qubits than just those required for basic computation. Ancillary qubits must be included for helping with computation and for correcting errors. This, combined with the second criterion, makes the task of preparing a desired state in a quantum computer a difficult one. Because we need to use new qubits whenever we wish to check for errors, we need to be able to provide the quantum computer with a steady supply of initialized qubits. This is also the reason why we need a long decoherence time relative to switching time. If we cannot initialize a register for error correction fast enough to catch the errors being produced, we risk losing the state before the next gate is applied. The qubits will eventually be unusable due to decoherence.

6.3 Current Methods of Qubit Creation

There are numerous methods that are currently being explored for the creation of physically viable qubits. The following is a survey of some of these methods and a brief comparison of some of their important characteristics.

6.3.1 Trapped Ion

One common method used is the trapped ion method. This entails arranging some ions in a straight line along the axis of the trap. Lasers are situated in such a way as to create standing waves with nodes at the rest position of the ions. By applying laser pulses of suitable intensity and duration, it is possible to put each of the ions in its ground state $|0\rangle$ or first excited state $|1\rangle$ or an arbitrary superposition of these two states, which is equivalent to implementing an arbitrary single qubit gate. Using the ground state of an ion is referred to as using cold ions, and this method enjoys a relatively long decoherence time of around 6 seconds, depending upon the ion used, and a switching time of 35 milliseconds [3]. An alternate form of the ion trap uses "hot" ions for qubits, meaning states above the ground state are utilized for the states of the qubits. Using hot ions eliminates the need to maintain the low energy ground state of the ion through the application of gates and measurements. Scalability is an issue for the ion trap. Every qubit requires a designated pair of lasers for gate implementation, meaning a computer with n qubits would require $2n$ lasers. A potential solution to this has been suggested [12] that involves trapping the ions in silicon and using long-wavelength radiation and local magnetic fields for qubit manipulation. Successful implementation of this system would allow the fabrication of a scalable quantum computer [12].

6.3.2 Quantum Dot

Quantum dots are another popular source of qubits. A quantum dot is a semiconducting nanostructure comprised, for this application, of three semiconducting disks separated by two thin barriers through which an electron could tunnel. There are several options for qubits using this structure. One uses the two lowest energy levels of one such quantum dot with one electron acting as a qubit, with a third energy level used for conditional rotations of the qubit's state. Control of the spacing between the energy levels is achieved by application of a variable voltage between the top and bottom surfaces of the quantum dot. Multiple quantum dots are stored in a microcavity with a fixed wavelength laser at one end, whose wavelength is necessarily different from the resonance wavelength of the cavity. In order to manipulate the state of the quantum dot qubit, and thus implement gates, the voltage is manipulated to vary the energy needed to excite the electron in the quantum dot. The transition energies between the excited states are brought into and out of resonance with the laser and cavity frequencies and the sum of these frequencies, allowing manipulation of the state of the electron, and hence qubit manipulation [19]. A second method for creating qubits in quantum dots is to use the spin of the electron in the quantum dot. The exchange coupling in the quantum dot is manipulated to perform operations. Through simulations, the second method has been found to be scalable and can reach the target of 10^4 gate operations before an error occurs [8].

6.3.3 Nuclear Magnetic Resonance

A different approach to qubit creation, and even constructing a quantum computer, uses techniques of nuclear magnetic resonance and expectation values. This method is based on the use of a large number identical particles. The state of the nuclear spin is manipulated through short pulses of radio frequency radiation over the entire sample. These pulses can be tuned so they only affect specific nuclei in each particle, while affecting all particles in the same manner. This allows implementation of qubit specific gates. A measurement of the sample is achieved by measuring the magnetic moment of the sample, which equates to the expectation value of the average state of the qubits in the sample instead of the state of any one qubit in it. Scalability is a significant problem for this method. It is predicted that this method would only be able to solve problems up to 70 bits in size, and in order for that many qubits to be realized, a molecule with that many atoms with distinguishable spins must be utilized. If a liquid is used as the sample, decoherence times can be on the order of seconds, and the expected number of operations possible before decoherence is predicted to be on the order of 10^3 [4]. These drawbacks limit the viability of an NMR quantum computer being practical for large scale problems.

6.3.4 Spin-Resonance Transistor

The final method we will consider uses the spin of an electron trapped in a hetero-structure comprised of two different semiconducting alloys. The electron of a doped ion surrounded by

two different alloys is excited by an electric field, pulling the wave function of the electron into the two alloys and the difference in spin in the two alloys is used as the qubit. A measurement of the state is accomplished by reading the charge of the system. Manipulating the state of the system can be done for a single qubit by varying the electric field relative to a background field, allowing implementation of single qubit gates. In order to achieve a two qubit gate with these qubits, the voltage applied to both electrons must be increased so as to overlap their wave functions. When this has occurred, the two qubit gate can be implemented by the same method as before. Further application of electric field on neighboring systems results in two qubit systems. This method has a predicted operation time of up to 10^{-9} seconds for one operation and decoherence time of 10^{-3} seconds, sufficiently long for error correction [22].

Qubit	Control	Decoherence Time (seconds)	Switching Time (seconds)	Ratio
Trapped Ion	Laser Pulses	6	$3.5 \cdot 10^{-2}$	171
Quantum Dot	Variable Voltage and Lasers	-	-	10^4
NMR	Radio Frequency Radiation	-	-	10^3
SRT	Variable Electric Field	10^{-9}	10^{-3}	10^6

Table 6.1: Table showing basic information for comparison of various qubit creation techniques. Ratio indicates the ratio of switching time to decoherence time.

Appendix A

A Proof of Theorem 4.1

This section contains the proof of Theorem 4.1, restated below for reference.

Theorem A.1. *Let N be a natural number with prime factorization pq, where p and q are distinct primes. Then for an integer x chosen at random in the interval $[1, n-1]$, with $gcd(N, x) = 1$,*

$$Pr[r \text{ is even and } x^{r/2} \not\equiv -1 \ (mod\ N)\ ] \geq \frac{1}{2}, \tag{A.1}$$

where r is the order of x $(mod\ N)$.

The proof of this theorem is given for an odd N with any prime factorization in [7]. We provide this proof with expansions, and for the specific case of two unique prime factors. To establish necessary definitions, we begin the proof here, before proving several necessary lemmas. We conclude this appendix with the remainder of the proof.

Since r is by definition the least positive integer such that $x^r \equiv 1 \pmod N$, $x^{r/2} \not\equiv 1 \pmod N$. The condition "$r$ is even and $x^{r/2} \not\equiv \pm 1 \pmod N$" is equivalent to "$r$ is even and $x^{r/2} \not\equiv -1 \pmod N$". We find the probability of the negation of this condition, which is "r is odd or $x^{r/2} \equiv -1 \pmod N$."

To prove the inequality in Theorem 4.1, we will establish

$$Pr(r \text{ is odd or } x^{r/2} \not\equiv -1 \pmod N) \leq \frac{1}{2}. \tag{A.2}$$

Since $\gcd(x, N) \equiv 1$, we can state that for any $n \in \mathbb{N}$, if $n|x$ and $n|N$, then $n|1$. The only natural numbers which divide N are 1, p, and q. If $p|x$, this would imply $p|1$, and similarly for q. As both of these are greater than one, $p \nmid x$ and $q \nmid x$. Since 1 and p are the only natural numbers to divide p, and $p \nmid x$, $\gcd(p, x) = 1$. Similarly, $\gcd(q, x) = 1$.

Let r_p denote the order of $x \pmod p$, and r_q denote the order of $x \pmod q$. Since $x^r \equiv 1 \pmod N$, we can state equivalently that $N|(x^r - 1)$. As $p|N$, $p|(x^r - 1)$, so $x^r \equiv 1 \pmod p$. It follows that $r_p|r$, as stated in [14][Sec. 2.129]. By the same logic, $r_q|r$.

Since r is the least nonnegative integer such that $x^r \equiv 1 \pmod N$,

$$r = \text{lcm}(r_p, r_q). \tag{A.3}$$

We can factor r_p and r_q, as

$$r_p = s_p 2^{t_p}$$
$$r_q = s_q 2^{t_q},$$

where s_p and s_q are odd, and $s_p, s_q, t_p, t_q \in \mathbb{N}$.

Since r is the least common multiple of r_p and r_q, r must be a multiple of $2^{\max\{t_p,t_q\}}$. We let

$$t = \max\{t_p, t_q\}.$$

Likewise, r must be a multiple of s_p and s_q. We let

$$s = \operatorname{lcm}(s_p, s_q).$$

As s_p and s_q are both odd, s is also odd by Lemma A.4, and therefore 2^t and s are coprime. Then

$$r = \operatorname{lcm}(s, 2^t) = s2^t. \tag{A.4}$$

At this point, we pause to prove the necessary lemmas to complete the proof.

Lemma A.2. *For an even r, where r is the order of $x \pmod N$ with $N = pq$ where p and q are distinct odd primes,*

$$x^{r/2} \equiv -1 \pmod N \text{ if and only if } x^{r/2} \equiv -1 \pmod p \text{ and } x^{r/2} \equiv -1 \pmod q. \tag{A.5}$$

Proof. Let $x^{r/2} \equiv -1 \pmod N$. Equivalently, $N|(x^{r/2}+1)$. Since $p|N$, $p|(x^{r/2}+1)$, so $x^{r/2} \equiv -1 \pmod p$. The same is true for q.

Let $x^{r/2} \equiv -1 \pmod p$ and $x^{r/2} \equiv -1 \pmod q$. By the Chinese Remainder Theorem, since p and q are relatively prime (as they are both prime), the system of congruences

$$x \equiv -1 \pmod p$$
$$x \equiv -1 \pmod q$$

has a unique solution $\pmod N$, keeping in mind $N = pq$.

By inspecting the system, $x = -1$ is a solution. Since $x^{r/2}$ is a solution, and the solution $\pmod N$ is unique, $x^{r/2} \equiv -1 \pmod N$. □

Lemma A.3. *If $x^{r/2} \equiv -1 \pmod p$, then $t_p = t$. The same is true for t_q.*

Proof. Assume that $x^{r/2} \equiv -1 \pmod p$, but $t_p < t$. Using (A.4) to express r as $r = s2^t$, we see that $2 \nmid s$, and $2^{t_p}|2^t$. If $t_p < t$, then $2^{t_p}|2^{t-1}$, and $r_p|s2^{t-1}$. We can then write $kr_p = \frac{r}{2}$ for some $k \in \mathbb{N}$.

Since $x^{r_p} \equiv 1 \pmod p$, we can raise both sides to the power k and find $x^{r/2} \equiv 1 \pmod p$. However, we assumed that $x^{r/2} \equiv -1 \pmod p$, so it must be that $t_p = t$. □

Lemma A.4. *Let $r = \operatorname{lcm}(r_p, r_q)$. Then r is odd if and only if r_p and r_q are odd.*

Lemma A.5. *For any $j \in \mathbb{N}$, $Pr(t_p = j) \leq \frac{1}{2}$, and the same is true for t_q.*

Proof. We consider the multiplicative group $\mathbb{Z}_p^*$. Using the Euler totient function, $|\mathbb{Z}_p^*| = \phi(p)$. Since p is prime, $\phi(p) = p - 1$.

Let g be a generator of $|\mathbb{Z}_p^*|$. The order of g is $\phi(p)$.

Let $\phi(p) = \sigma 2^\tau$, where σ is odd.

Let $x \in \mathbb{Z}_p^*$ be represented using the generator g as $g^b \pmod p$, where b is an integer in the range $[1, p-1]$. Represent the order of $x = g^b$ as $\tilde{\sigma} 2^{\tilde{\tau}}$, with odd $\tilde{\sigma}$. From the definition of order,

$$g^{b\tilde{\sigma}2^{\tilde{\tau}}} \equiv 1 \pmod p. \tag{A.6}$$

Since the order of an element is unique, and the order of g was previously defined, it must be that

$$\sigma 2^\tau = b\tilde{\sigma}2^{\tilde{\tau}}. \tag{A.7}$$

There are two cases to consider, depending on the parity of b. If b is even, then $\tilde{\tau} < \tau$, keeping in mind that σ and $\tilde{\sigma}$ are defined as odd. If b is odd, then $\tau = \tilde{\tau}$. We can conclude that $\bar{\tau}$ is always less than or equal to τ, so if $j > \tau$, $Pr(t_p = j) = 0$.

Since each element of $\mathbb{Z}_p^*$ is equal to g raised to a unique exponent b in $[1, p-1]$, half of the possible choices for x lead to $\tilde{\tau} = \tau$ and the other half lead to $\tilde{\tau} < \tau$. Then at most half of the elements in $\mathbb{Z}_p^*$ will satisfy the requirement that $\tilde{\tau} = j$ for any specified j. Therefore,

$$Pr(\tilde{\tau} = j) \leq \frac{1}{2}. \tag{A.8}$$

□

The rest of the proof of Theorem 4.1 is as follows.

Proof. We now explore the two conditions given in (A.2), to find that if either one is true, then $t_p = t_q$.

If r is odd, then r_p and r_q are odd by Lemma A.4. If r_p is odd, then from the definition $r_p = s_p 2^{t_p}$, $t_p = 0$. The same is true for r_q, so $t_p = t_q = 0$.

If $x^{r/2} \equiv -1 \pmod N$, then $x^{r/2} \equiv -1 \pmod p$, and the same is true modulo q, by Lemma A.2. Applying Lemma A.3, $t_p = t_q = t$.

We have confirmed that if either of the conditions in (A.2) is satisfied, then $t_p = t_q$. Therefore,

$$Pr(r \text{ is odd or } x^{r/2} \not\equiv -1 \pmod N) \leq Pr(t_p = t_q). \tag{A.9}$$

Using Lemma A.5,

$$\begin{aligned} Pr(t_p = t_q) &= \sum_{j \in \mathbb{N}} Pr(t_p = j) Pr(t_q = j) \\ &\leq \sum_{j \in \mathbb{N}} Pr(t_p = j) \frac{1}{2} \\ &\leq \frac{1}{2}. \end{aligned} \tag{A.10}$$

The second line leads to the third since $\sum_{j\in\mathbb{N}} Pr(t_p = j) = 1$, as t_p must be equal to some nonnegative integer.

Combining (A.9) and $Pr(t_p = t_q) \leq \frac{1}{2}$, we arrive at (A.2). Since the negation of the condition in (A.2) is that r is even and $x^{r/2} \equiv -1 \pmod{N}$, we can state that

$$\begin{aligned} Pr(r \text{ is even and } x^{r/2} \equiv -1 \pmod{N}\ ) &= 1 - Pr(r \text{ is odd or } x^{r/2} \not\equiv -1 \pmod{N}) \\ &\geq \frac{1}{2}. \end{aligned} \tag{A.11}$$

□

Appendix B

Probability Integral for Factoring

We want to calculate the probability of observing the state $|c\rangle\,|x^k \pmod n\rangle$. This probability is given by

$$Pr[MR = c, x^k \pmod n] = \left| \frac{1}{q} \sum_{x^a \equiv x^k} \exp(2\pi i a c/q) \right|^2 . \tag{B.1}$$

This is arrived at by the process of measurement as defined earlier. The right hand side of (B.1) is the square of the complex modulus of the coefficients for the desired states, which we distinguish with the condition $a \equiv k \pmod r$. Letting $\omega = e^{2\pi i/q}$, we can rewrite this as

$$Pr[MR = c, x^k \pmod n] = \left| \frac{1}{q} \sum_{x^a \equiv x^k} \omega^{ac} \right|^2 = \frac{1}{q} \sum_{b=0}^{\lfloor \frac{q-k-1}{r} \rfloor} \omega^{(br+k)c}. \tag{B.2}$$

Because a and k are congruent $\pmod r$, we can write $a = br + k$. Keeping in mind the highest b such that $br + k < q$ satisfies $br + k \leq q - 1$, we find $b \leq \lfloor \frac{q-k-1}{r} \rfloor$. Now, after substitution and simplifications,

$$\begin{aligned} Pr[MR = c, x^a \pmod n] &= \left| \frac{1}{q} \sum_{b=0}^{\lfloor \frac{q-k-1}{r} \rfloor} \omega^{(br+k)c} \right|^2 \\ &= \left| \frac{1}{q} \sum_{b=0}^{\lfloor \frac{q-k-1}{r} \rfloor} \omega^{b\{rc\}_q} \right|^2 , \end{aligned} \tag{B.3}$$

where $rc = dq + t$ for $d \in \mathbb{Z}$, $t \in (-\frac{q}{2}, \frac{q}{2}]$ and $t = \{rc\}_q$ is the residue of rc modulo q. It is of note that ω^{kc} is independent of b so it can be factored out of the sum and ignored as $|\omega^{kc}| = 1$.

Having this series representation for the probability of measuring the state $|c\rangle\,|x^k \pmod n\rangle$, the following is claimed by Shor:

$$\frac{1}{q}\sum_{b=0}^{\lfloor (q-k-1)/r\rfloor}\exp(2\pi ib\{rc\}_q/q) = \frac{1}{q}\int_0^{\lfloor (q-k-1)/r\rfloor}\exp(2\pi ib\{rc\}_q/q)db + \mathcal{O}\left(\frac{\lfloor (q-k-1)/r\rfloor}{q}(\exp(2\pi ib\{rc\}_q/q)-1)\right). \tag{B.4}$$

We define a few quantities for convenience, as follows:

$$L = \lfloor (q-k-1)/r\rfloor$$
$$f(x) = \exp(2\pi ix\{rc\}_q/q).$$

Note that $f(x) \in \mathbb{C}$ with $|f(x)| = 1$ for every real x.

Our goal is to establish an upper bound for

$$\left|\frac{1}{q}\int_0^L f(b)db - \frac{1}{q}\sum_{b=0}^{L} f(b)\right|. \tag{B.5}$$

If the sum is thought of as an approximation to a Riemann integral, there are rectangles with left endpoints at integers from 0 to L. However, the integral ends at L. We make these two quantities easier to compare by stating that

$$\begin{aligned}\left|\frac{1}{q}\int_0^L f(b)db - \frac{1}{q}\sum_{b=0}^{L} f(b)\right| &= \frac{1}{q}\left|\int_0^L f(b)db - \sum_{b=0}^{L-1} f(b) - f(L)\right| \\ &\leq \frac{1}{q}\left|\int_0^L f(b)db - \sum_{b=0}^{L-1} f(b)\right| + \frac{1}{q}\left|f(L)\right|,\end{aligned} \tag{B.6}$$

where additionally $|f(L)| = 1$.

We can state that

$$\int_0^L f(b)db = \sum_{t=0}^{L-1}\left(\int_t^{t+1} f(b)db\right),$$

and consider the summation as well to find

$$\int_0^L f(b)db - \sum_{t=0}^{L-1} f(t) = \sum_{t=0}^{L-1}\left(\int_t^{t+1} f(b)db - f(t)\right).$$

By the Triangle Inequality,

$$\left|\int_0^L f(b)db - \sum_{t=0}^{L-1} f(t)\right| \leq \sum_{t=0}^{L-1}\left|\int_t^{t+1} f(b)db - f(t)\right|. \tag{B.7}$$

We observe that $f(b+x) = f(b)f(x)$, so

$$\int_t^{t+1} f(b)db = \int_0^1 f(b+t)db = f(t)\int_0^1 f(b)db.$$

By substituting back into (B.7),

$$\begin{aligned}\left|\int_0^L f(b)db - \sum_{t=0}^{L-1} f(t)\right| &\leq \sum_{t=0}^{L-1}\left|f(t)\int_0^1 f(b)db - f(t)\right| \\ &\leq \sum_{t=0}^{L-1}|f(t)|\left|\int_0^1 (f(b)-1)db\right|.\end{aligned}$$

Since $|f(t)| = 1$, we have, using the fact that $|\int h(z)dz| \leq \int |h(z)|dz$,

$$\left|\int_0^L f(b)db - \sum_{t=0}^{L-1} f(t)\right| \leq \sum_{t=0}^{L-1}\int_0^1 |f(b)-1|\,db.$$

We can create an upper bound on the above integrand,

$$|f(b)-1| \leq |f(1)-1|.$$

Substituting this into our inequality yields

$$\left|\int_0^L f(b)db - \sum_{t=0}^{L-1} f(t)\right| \leq \sum_{t=0}^{L-1}\int_0^1 |f(1)-1|\,db,$$

and then

$$\left|\int_0^L f(b)db - \sum_{t=0}^{L-1} f(t)\right| \leq \sum_{t=0}^{L-1}|f(1)-1|. \tag{B.8}$$

The summand on the right is now independent of t, so we can evaluate to find

$$\left|\int_0^L f(b)db - \sum_{t=0}^{L-1} f(t)\right| \leq L\,|f(1)-1|.$$

Before substituting back in the values for L and $f(x)$, we should recall that we are still neglecting the last term in the series. Bringing this term back yields

$$\left|\int_0^L f(b)db - \sum_{t=0}^{L-1} f(t) - f(L)\right| \leq L\,|f(1)-1| + |f(L)|.$$

As previously mentioned, $|f(L)| = 1$, so making this substitution and bringing the $1/q$ factor back on both sides results in

$$\frac{1}{q}\left|\int_0^L f(b)db - \sum_{t=0}^{L-1} f(t) + f(L)\right| \leq \frac{\lfloor (q-k-1)/r \rfloor}{q} \left|\exp(2\pi i \{rc\}_q/q) - 1\right| + \frac{1}{q}. \tag{B.9}$$

Clearly, the extra $1/q$ term can be absorbed into the leading term when we express this error in Landau notation

$$\mathcal{O}\left(\frac{\lfloor (q-k-1)/r \rfloor}{q} \left|\exp(2\pi i \{rc\}_q/q) - 1\right|\right). \tag{B.10}$$

We now have

$$\begin{aligned} Pr[MR = c, x^a \pmod n] = \Bigg| &\frac{1}{q}\int_0^{\lfloor (q-k-1)/r \rfloor} \exp(2\pi i b \{rc\}_q/q) db \\ &+ \mathcal{O}\left(\frac{\lfloor (q-k-1)/r \rfloor}{q}(\exp(2\pi i b \{rc\}_q/q) - 1)\right)\Bigg|^2. \end{aligned} \tag{B.11}$$

We can make a final simplification to this integral term by making the substitution $u = rb/q$. The result of this substitution is

$$\begin{aligned} Pr[MR = c, x^a \pmod n] = \Bigg| &\frac{1}{r}\int_0^{\frac{r}{q}\lfloor (q-k-1)/r \rfloor} \exp(2\pi i u \{rc\}_q/r) du \\ &+ \mathcal{O}\left(\frac{\lfloor (q-k-1)/r \rfloor}{q}(\exp(2\pi i b \{rc\}_q/q) - 1)\right)\Bigg|^2, \end{aligned} \tag{B.12}$$

allowing us to approximate the upper limit of the integral by 1 with only an error of size $\mathcal{O}\left(\frac{1}{q}\right)$.

If we were to evaluate the integral, we would have a leading coefficient $\frac{r}{\{rc\}_q 2\pi}$. Clearly this is undefined if the ratio $\frac{\{rc\}_q}{r}$ is zero. Furthermore, it can be observed that this coefficient's value will decrease as this ratio increases. Therefore, we want to take the endpoints of the interval over which $\frac{\{rc\}_q}{r}$ varies, i.e. $\pm\frac{1}{2}$, to minimize the integral. Taking this value and evaluating the integral yields a lower bound on the probability of $\frac{1}{3r^2}$.

Using this restriction on $\{rc\}_q$,

$$-\frac{r}{2} \leq \{rc\}_q \leq \frac{r}{2}.$$

For some integer d, $rc = dq + \{rc\}_q$. Therefore, we have that

$$-\frac{r}{2} \leq rc - dq \leq \frac{r}{2}.$$

After division by rq, we find

$$-\frac{1}{2q} \leq \frac{c}{q} - \frac{d}{r} \leq \frac{1}{2q}.$$

Equivalently, as used in Chapter 4, $\left|\frac{c}{q} - \frac{d}{r}\right| \leq \frac{1}{2}$. Furthermore, if d exists to satisfy this inequality, it is unique. By subtracting $\frac{c}{q}$ from each term in the inequality, we find

$$-\left(\frac{1+2c}{2q}\right) \leq -\frac{d}{r} \leq \frac{1-2c}{2q}.$$

After multiplying by -1,

$$-\left(\frac{1-2c}{2q}\right) \leq \frac{d}{r} \leq \frac{1+2c}{2q}. \tag{B.13}$$

Let $\frac{d}{r}$ be the smallest element of the form $\frac{x}{r}$ to satisfy (B.13). All $\frac{x}{r}$ satisfying (B.13) must be within a range of size $\frac{1+2c}{2q} + \frac{1-2c}{2q} = \frac{1}{q}$. Since $r < n < n^2 < q$, $\frac{1}{q} < \frac{1}{r}$ and $\frac{d+1}{r}$ does not satisfy (B.13), there is at most one $\frac{x}{r}$ satisfying (B.13).

Appendix C

Probability Integral for Discrete Log

The following discussion was provided by Professor William Martin.

Turning a Sum into an Integral

Explication of (6.16) in Peter Shor's famous paper

William J. Martin, WPI

Task: On page 22 of his preprint "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer"[1], Peter Shor is deriving a lower bound on a probability amplitude of a certain output of an important quantum measurement. He states "The absolute value of the amplitude ... is at least"

$$S := \frac{1}{(p-1)q} \sum_{b=0}^{p-2} \cos\left(2\pi \left|\frac{W}{2} - \frac{Wb}{p-2}\right| + \frac{\pi}{6}\right).$$

He then writes "Replacing this sum with an integral, we get that the absolute value of this amplitude is at least"

$$\frac{2}{q}\int_0^{1/2} \cos\left(\frac{\pi}{6} + 2\pi|W|u\right)\, du \;+\; \mathcal{O}\left(\frac{W}{pq}\right). \tag{C.1}$$

He then uses the fact that $|W| \leq 1/2$ to replace the error term with $\mathcal{O}\,(1/pq)$.

Detailed Justification: First re-write the sum as

$$S := \frac{1}{(p-1)q} \sum_{b=0}^{p-2} \cos\left(\frac{\pi}{6} + 2\pi|W|\left|\frac{1}{2} - \frac{b}{p-2}\right|\right).$$

We introduce $v_b = \frac{b}{p-2}$ and note that, for $\bar{b} = (p-3)/2$, we have $v_{\bar{b}} < 1/2 < v_{\bar{b}+1}$. Next, we give the integrand a name: let

$$g(v) = \cos\left(\frac{\pi}{6} + 2\pi|W|\left|\frac{1}{2} - v\right|\right). \tag{C.2}$$

[1] arXiv:quant-ph/9508027v2

and observe that $g\left(\frac{1}{2}+x\right) = g\left(\frac{1}{2}-x\right)$ so that

$$J := \frac{1}{q}\int_0^1 g(u)\,du = \frac{2}{q}\int_0^{1/2} \cos\left(\frac{\pi}{6} + 2\pi|W|u\right)\,du\ .$$

Using this notation, we may likewise abbreviate the sum as

$$S := \frac{1}{(p-1)q}\sum_{b=0}^{p-2} g\left(v_b\right).$$

Our goal is to approximate $J' := \int_0^1 g(u)\,du$ by a Riemann sum with $p-2$ rectangles of width $1/(p-2)$ and height $g(v_b)$ (left-hand endpoint rule) for the b^{th} rectangle. Since the sine function ranges between -1 and 1, the area of any rectangle in this Riemann sum has absolute value at most $\frac{1}{p-2}$, so we can throw away any constant number of these and absorb the difference into the error term. Let's ignore the last term in the sum and scale by q to define

$$S' := \sum_{b=0}^{p-3} \frac{g\left(v_b\right)}{p-1}$$

so that $qS \approx S'$. Next, we replace $p-1$ in the denominator by $p-2$ (which is of course very close to $p-1$ when p is large) and define

$$S'' := \sum_{b=0}^{p-3} \frac{g\left(v_b\right)}{p-2},$$

which we view as our Riemann sum for the integral $J' = \int_0^1 g(u)\,du$.

To compute the error in this estimate, we split up the integral as follows:

$$J' = \sum_{b=0}^{p-3} \int_{I_b} g(u)\,du$$

where I_b denotes the closed interval $[v_b, v_{b+1}]$ $(0 \le b < p-2)$ of width $\frac{1}{p-2}$. We have

$$J' - S'' = \sum_{b=0}^{p-3} \left(\int_{I_b} g(u)\,du - \frac{g(v_b)}{p-2}\right).$$

Now we bound the absolute value using the Triangle Inequality

$$|J' - S''| \le \sum_{b=0}^{p-3} \left|\int_{I_b} g(u)\,du - \frac{g(v_b)}{p-2}\right|. \tag{C.3}$$

Observe that $g(x)$ is increasing on $[0, 1/2]$ (e.g., on I_b for $0 \le b < \frac{p-3}{2}$) and, being symmetric about $1/2$, is decreasing on $[1/2, 1]$. So, except when $1/2 \in I_b$ (where we switch from

increasing to decreasing), the minimum and maximum of $g(v)$ on I_b are the two endpoints v_b and v_{b+1}. So the difference between the integral on I_b and the corresponding rectangle is at most, in absolute value, the area of the rectangle with corners

$$(v_b, g(v_b)),\ (v_{b+1}, g(v_b)),\ (v_{b+1}, g(v_{b+1})),\ (v_b, g(v_{b+1})).$$

This rectangle has width $\frac{1}{p-2}$ and height $|g(v_{b+1}) - g(v_b)|$. Since g is continuous on I_b and differentiable on its interior, the Mean Value Theorem tells us that there exists some $c_b \in I_b$ with

$$g(v_{b+1}) - g(v_b) = \frac{1}{p-2} g'(c_b).$$

But g is simply a sine function and we easily compute, from Equation (C.2),

$$g'(v) = -\sin\left(\frac{\pi}{6} + 2\pi|W|\left|\frac{1}{2} - v\right|\right)(\pm 2\pi|W|)$$

($v \neq \frac{1}{2}$). So we have $|g'(c_b)| \leq 2\pi|W|$ and we find

$$|g(v_{b+1}) - g(v_b)| \leq \frac{2\pi|W|}{p-2}.$$

Substituting this into our estimate for the b^{th} term of the error, we have

$$\left|\int_{I_b} g(u)\,du - \frac{g(v_b)}{p-2}\right| \leq \frac{2\pi|W|}{(p-2)^2}$$

for $b \neq \bar{b}$. So, Inequality (C.3) can be extended to

$$|J' - S''| \leq \sum_{b=0}^{p-3}\left|\int_{I_b} g(u)\,du - \frac{g(v_b)}{p-2}\right| \leq (p-3)\frac{2\pi|W|}{(p-2)^2} + \left|\int_{I_{\bar{b}}} g(u)\,du - \frac{g(v_{\bar{b}})}{p-2}\right|.$$

Since the interval $I_{\bar{b}}$ has width only $\frac{1}{p-2}$, we can use the maximum value $\cos\left(\frac{\pi}{6}\right)$ to obtain the upper bound

$$\left|\int_{I_{\bar{b}}} g(u)\,du - \frac{g(v_{\bar{b}})}{p-2}\right| \leq \frac{\cos\left(\frac{\pi}{6}\right)}{p-2}.$$

Folding this into the above inequality, we find

$$|J' - S''| \leq \frac{2\pi|W| + \cos\left(\frac{\pi}{6}\right)}{p-2}.$$

Now we have to retrace our steps and replace J' by J and S'' by S in this error bound. First note that

$$S' = S'' - \frac{1}{p-1} S''$$

so that

$$|S' - S''| \leq \frac{1}{p-1}|S''| \leq \frac{1}{p-1}$$

since each of the $p-2$ terms in our definition of S'' has absolute value at most $\frac{1}{p-2}$. By the Triangle Inequality and using $1/(p-2) \leq 2/(p-1)$, we have

$$|J' - S'| \leq |J' - S''| + |S'' - S'| \leq \frac{4\pi|W| + 2\cos\left(\frac{\pi}{6}\right)}{p-1} + \frac{1}{p-1} = \frac{4\pi|W| + 1 + 2\cos\left(\frac{\pi}{6}\right)}{p-1}.$$

Next we account for the deleted term "$b = p-2$" in our definition of S by adding on $1/(p-1)$ to the right-hand side to get

$$\left| J' - \frac{1}{p-1}\sum_{b=0}^{p-2} g\left(v_b\right) \right| \leq \frac{4\pi|W| + 2 + 2\cos\left(\frac{\pi}{6}\right)}{p-1}.$$

Finally, we scale both sides by $1/q$ to find our error is bounded by

$$|J - S| \leq \frac{4\pi|W| + 2 + 2\cos\left(\frac{\pi}{6}\right)}{(p-1)q}.$$

Since $1/(p-1) \leq 2/p$ we can bound the error by

$$\frac{8\pi|W| + 4 + 4\cos\left(\frac{\pi}{6}\right)}{pq}$$

which clearly belongs to $\mathcal{O}\left(\frac{W}{pq}\right)$. $\square$

Bibliography

[1] Aoki, K., Franke, J., Kleinjung, T., Lenstra, A. K., and Osvik, D. A. (2007). A kilobit special number field sieve factorization. In *Advances in cryptology—ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Comput. Sci.*, pages 1–12. Springer, Berlin.

[2] Boyer, M., Brassard, G., Høyer, P., and Tapp, A. (1996). Tight bounds on quantum searching. *arXiv preprint quant-ph/9605034*.

[3] Cirac, J. I. and Zoller, P. (1995). Quantum computations with cold trapped ions. *Phys. Rev. Lett.*, 74:4091–4094.

[4] Cory, D. G., Fahmy, A. F., and Havel, T. F. (1997). Ensemble quantum computing by nmr spectroscopy. *Proceedings of the National Academy of Sciences of the United States of America*, 94(5):1634–1639.

[5] Deutsch, D. (1985). Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. Roy. Soc. London Ser. A*, 400(1818):97–117.

[6] DiVincenzo, D. P. (2000). The physical implementation of quantum computation. *Fortschritte der Physik*, 48(9-11):771–783.

[7] Ekert, A. and Jozsa, R. (1996). Quantum computation and Shor's factoring algorithm. *Rev. Modern Phys.*, 68(3):733–753.

[8] Friesen, M., Rugheimer, P., Savage, D. E., Lagally, M. G., van der Weide, D. W., Joynt, R., and Eriksson, M. A. (2003). Practical design and simulation of silicon-based quantum-dot qubits. *Physical Review B*, 67(12):121301.

[9] Juskalian, R. (2017). Practical quantum computers. *MIT Technology Review*.

[10] Kleinjung, T., Aoki, K., Franke, J., Lenstra, A. K., Thomé, E., Bos, J. W., Gaudry, P., Kruppa, A., Montgomery, P. L., Osvik, D. A., te Riele, H., Timofeev, A., and Zimmermann, P. (2010). Factorization of a 768-bit RSA modulus. In *Advances in cryptology—CRYPTO 2010*, volume 6223 of *Lecture Notes in Comput. Sci.*, pages 333–350. Springer, Berlin.

[11] Laflamme, R., Knill, E., Cory, D. G., Fortunato, E. M., Havel, T., Miquel, C., Martinez, R., Negrevergne, C., Ortiz, G., Pravia, M. A., et al. (2002). Introduction to nmr quantum information processing. *arXiv preprint quant-ph/0207172*.

[12] Lekitsch, B., Weidt, S., Fowler, A. G., Mølmer, K., Devitt, S. J., Wunderlich, C., and Hensinger, W. K. (2017). Blueprint for a microwave trapped ion quantum computer. *Science Advances*, 3(2).

[13] McIntyre, D., Manogue, C., Tate, J., and University, O. S. (2012). *Quantum Mechanics: A Paradigms Approach*. Pearson.

[14] Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. (1997). *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, Boca Raton, FL.

[15] Nielsen, M. A. and Chuang, I. L. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge.

[16] Norman, C. (2012). *Finitely Generated Abelian Groups and Similarity of Matrices over a Field*. Springer-Verlag, London.

[17] Olds, C. D. (2014). *Anneli Lax New Mathematical Library : Continued Fractions*. Mathematical Association of America.

[18] Preskill, J. (1998). Reliable quantum computers. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):385–410.

[19] Sherwin, M. S., Imamoglu, A., and Montroy, T. (1999). Quantum computation with quantum dots and terahertz cavity quantum electrodynamics. *Phys. Rev. A*, 60:3508–3514.

[20] Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509.

[21] Sudkamp, T. (2006). *Languages and Machines: An Introduction to the Theory of Computer Science*. Pearson international edition. Pearson Addison-Wesley.

[22] Vrijen, R., Yablonovitch, E., Wang, K., Jiang, H. W., Balandin, A., Roychowdhury, V., Mor, T., and DiVincenzo, D. (2000). Electron-spin-resonance transistors for quantum computing in silicon-germanium heterostructures. *Phys. Rev. A*, 62:012306.

[23] Watrous, J. (2012). Quantum computational complexity. In *Computational complexity. Vols. 1–6*, pages 2361–2387. Springer, New York.

[24] Weber, D. (1996). Computing discrete logarithms with the general number field sieve. In *Algorithmic number theory (Talence, 1996)*, volume 1122 of *Lecture Notes in Comput. Sci.*, pages 391–403. Springer, Berlin.

[25] Williams, C. P. (2011). *Explorations in Quantum Computing*. Texts in Computer Science. Springer-Verlag, London.