CHAPTER 1
Introduction

Computational complexity theory, the study of the relative difficulty of computing functions, has ever since it's inception in the 1970's used the Turing Machine (TM) as a basic universal model of computation. This approach was justified by a series of theorems which showed all other suggested models as sufficiently equivalent to the Turing Machine. This fact is somewhat ironic, for in spite of its relatively late appearance, the theory utilizes a model that seems to disregard what is perhaps the most revolutionary theory of all modern time – Quantum Mechanics. Feynman in 1982, pointed out that the simulation of a quantum mechanical system on an ordinary computer is apparently plagued by exponential slow down, thereby suggesting that inherent in the Turing Machine are the limitations of classical physics [1]. Feynman's suggestion gave rise to a new model of computation known as the quantum Turing machine (QTM), whose execution follows the laws of quantum physics. This model was first formulated by Deutsch and later improved by Bernstein and Vazirani [2, 3].

The model introduced by Bernstein and Vazirani bears a great resemblance to the classical probabilistic Turing Machines. Both are models of computation involving forms of randomness which are both similar and different in many respects. The randomness in the probabilistic Turing Machine is based on the observable day to day phenomena of nature, whereas the quantum Turing Machine possesses randomness which occurs only at the microscopic level of interactions between atomic and subatomic particles. It is the special relationship between interactions of particles on the

microscopic scale, coupled with Feynman's observation, that motivates the definition of the quantum Turing Machine.

As defined by Bernstein and Vazirani [3], the quantum Turing Machine's distinguishing characteristic is it's phenomenon of interference. This effect which the model exhibits allows for a variety of interesting and powerful results to arise. Unfortunately, interference does not appear to lend itself to polynomial time simulation on a classical Turing Machine, and indeed, had such a simulation been feasible, a number of open questions in contemporary complexity theory would be answered. It has, however, been shown that the class of functions efficiently computable by a classical Turing Machine is a subset of those efficiently computable by a quantum Turing Machine [3]. This relationship proves extremely useful in determining the computational power of the quantum Turing Machine, as it is not at all clear from its definition that it indeed possesses any advantage over its classical predecessors.

The results currently under discussion in this relatively new area of complexity theory provide us with a much greater promise. Last year, Simon [4] showed that there exists a problem for which no apparent polynomial time solution on a classical probabilistic Turing Machine, yet has an elegant and efficient solution on a quantum Turing Machine. Simon's result sparked an interest in the model, and only a few months later polynomial time algorithms for factoring integers and solving the Discrete Log problem on a quantum Turing Machine were presented by Peter Shor. The latter result is of extreme significance as the solution to both those problems is conjectured to require super-polynomial time on any classical Turing Machine, and this conjecture is long standing.

In this final report I have attempted to consolidate the above definitions and results. As this is a new field in complexity theory, there is no published text on the topic to date. Papers in this area most often assume a strong background in mathematics, and thus do not lend themselves to an easy read. I have tried to produce a paper which would provide the reader with an introduction to the field, beginning with

basic definitions in classical complexity theory, and progressing towards the development of the quantum Turing Machine's definition. In addition, Simon's and Shor's results are brought here with their complete proofs. Finally, Chapter 8 includes a discussion of the current known limitations of the model. The main result presented was proposed by Bennett, Brassard, Bernstein and Vazirani in a paper that has not yet been published [5]. Understanding this paper consumed a large proportion of the time allotted to this project, and it appears as though the result presented here is better than the one in the original paper by a linear factor.

Before beginning the discussion of the model, it should be mentioned that while this is intended as an introductory paper to the subject, it does assume a basic knowledge of classical complexity theory, linear algebra, and group theory.

CHAPTER 2
Classical Turing Machines

A Turing Machine (TM) is commonly characterized by an alphabet, a one-sided infinite tape, a read-write tape head and a finite-state control which includes a halting state. For the purpose of facilitating the transition from Classical Turing Machines to Quantum Turing Machines we shall adopt a definition which is more general than normally introduced in the literature. We initially allow the tape to be infinite on both sides, and formally define,

DEFINITION 2.1. *A Turing Machine is a quadruple* (K, Σ, δ, s) *where,*
K *is a finite set of states, not containing the halt state* h.
Σ *is an alphabet containing a blank symbol, but not containing* L *and* R
$s \in K$ *is the initial state*
δ *is a function from* $K \times \Sigma \times (K \cup h) \times (\Sigma \cup \{L, R\})$ *to* $\mathbb{C}$. *(complex numbers)* [6]

The symbols L and R are excluded from Σ in definition 2.1 since they indicate a movement of the tape head, rather than a symbol that is written on the tape. Likewise, the state h is excluded from K, and is reserved to denote the halting state. For the classical Turing Machines, it can be shown that the extension of the Turing Machine from having a one-sided infinite tape to a two-sided infinite tape is insignificant and introduces nothing more than a constant speedup in the computation [6, 7] .

Classically, δ is defined as an everywhere defined relation from $K \times \Sigma$ to $(K \cup h) \times (\Sigma \cup \{L, R\})$. This definition is more intuitive as it relates every state-symbol pair at time t with the corresponding state-symbol pair at time $t+1$. We call δ for that reason the transition function, and its specification determines the type and operation of the Turing Machine. Different transition functions lead to different classes of Turing Machines. It is easy to see that by restricting the range of δ to $\{0, 1\}$ we effectively force δ to specify a relation between $K \times \Sigma$ and $(K \cup h) \times (\Sigma \cup \{L, R\})$ as in the classical sense. We will consider a TM to produce a state-symbol pair (k_2, σ_2) from a state-symbol pair (k_1, σ_1) if and only if $\delta(k_1, \sigma_2, k_2, \sigma_2) \neq 0$ [6].

The full description of a Turing Machine at a given time, known as the TM's configuration, consists of the current information stored on the tape, the current state of the machine, and position of the head on the tape. Formally,

DEFINITION 2.2. *A configuration of a Turing Machine $M = (K, \Sigma, \delta, s)$ is a member of*

$$(K \cup h) \times \Sigma^* \times \Sigma \times (\Sigma^*(\Sigma - \{\#\}) \cup e).$$

where e is the empty string.

Note that in a configuration (k, u, v, w), k represents the current state, v represents the current symbol under the head, u represents the contents of the tape to the left of the head excluding leading blanks and w represents the contents to the right excluding trailing blanks. For convenience, we will denote such a configuration by $(k, u\underline{v}w)$[6]

Given a particular Turing Machine M, M is said to *yield* configuration y from configuration x, if y can be produced from x by proceeding to a new state, and either writing one symbol on the tape or moving the tape head one step to the left or the right. Formally,

DEFINITION 2.3. *Given a Turing Machine $M = (K, \Sigma, \delta, s)$, M is said to yield configuration (k', u', v', w') from configuration (k, u, v, w) if one of the following three conditions hold:*

(1) $\delta(k, u, k', u') \neq 0$, $u' \in \Sigma$, $u' = u$ *and* $w' = w$

(2) $\delta(k, u, k', L) \neq 0$, $u'v' = u$, *and either*

  (a) $w' = vw$; *or,*

  (b) $w' = w = \lambda$ *and* $v' = \#$

(3) $\delta(k, u, k', R) \neq 0$, $u' = uv$, *and either*

  (a) $w = v'w'$; *or,*

  (b) $w = w' = \lambda$ *and* $v' = \#$

(Here λ is used to denote the empty string.) We will denote such a transition by $(k, u, v, w) \overset{M}{\Longrightarrow} (k', u', v', w')$ [6].

Note that while the number of configurations is infinite, (since the tape is infinite) the number of configurations that a single configuration can yield is finite. The TM's *initial configuration* is defined as the configuration of the TM before execution, and a *halted configuration* is defined as a configuration describing a machine which is currently in its halting state and is no longer running. Furthermore, a TM is said to compute a function f over an alphabet Σ if for every string $x \in \Sigma^*$, if the tape consists of nothing but x in the initial configuration, then the machine reaches a halting state and $f(x)$ is the only string written on the tape in the halted configuration. For the purpose of this discussion we will restrict the TMs under consideration to those which halt on every input. While this assumption is very restrictive in certain settings, it is inconsequential here. [6]

Thus, a computation of a function in a Turing Machine can be viewed as a finite sequence of configurations that starts with an initial configuration and ends with a halted configuration. In order to further clarify the distinction between the different types of Turing Machines, we introduce a graph $G = (V, E)$ known as the configuration graph in which every vertex is labeled by a configuration and an edge $(x, y) \in E$ if and only if configuration x can yield configuration y. Note that since the tape is infinite the configuration graph is as well infinite.

2.1. Deterministic Turing Machines

A deterministic Turing Machine (DTM) is one in which every non-halted configuration yields exactly one other configuration. This is the result of formally restricting the range of δ to $\{0,1\}$ and furthermore requiring that for every $k_1 \in K$ and $\sigma_1 \in \Sigma$ there exists exactly one pair $(k_2, \sigma_2) \in (K \cup h) \times (\Sigma \cup \{L, R\})$ such that $\delta(k_1, \sigma_1, k_2, \sigma_2) = 1$. In other words, we require this model to have a uniquely determined action to every state-symbol pair. This implies that if the machine is scanning a certain symbol and is in a particular state, it proceeds to a new state and either writes a new symbol on the tape or moves to the left or right. As a consequence, given a particular initial configuration x and a particular deterministic Turing Machine M, we can determine with certainty the final (halted) configuration.

EXAMPLE 2.1. In the following example we define a simple Turing Machine, and show it's computation on a particular string. In each step we will indicate which transition of delta it is that we are using.

Let $M = (K, \Sigma, \delta, s)$ be a deterministic Turing Machine where,

$K = \{p, q\}$, $\Sigma = \{0, 1, \#\}$, $s = p$, and

(1) $\delta(p, 1, q, \#) = 1$

(2) $\delta(p, 0, q, \#) = 1$

(3) $\delta(p, \#, h, \#) = 1$

(4) $\delta(q, \#, p, R) = 1$

(5) $\delta(q, 1, p, R) = 1$

(6) $\delta(q, 1, p, R) = 1$

(7) For all other 4-tuples x we define $\delta(x) = 0$.

We claim that this machine erases the input string from the tape. While not immediately obvious from the definition, this is best illustrated if we consider the computation of the machine on an initial configuration $(p, \#\underline{0}10\#)$ (notation defined

in definition 2.2). The computation is as follows:

$$
\begin{aligned}
(p, \#\underline{0}10\#) &\overset{M}{\Longrightarrow} (q, \#\#\underline{\#}10\#) && \text{by transition 2}\\
&\overset{M}{\Longrightarrow} (p, \#\#\underline{1}0\#) && \text{by transition 4}\\
&\overset{M}{\Longrightarrow} (q, \#\#\#\underline{0}\#) && \text{by transition 1}\\
&\overset{M}{\Longrightarrow} (p, \#\#\#\underline{0}\#) && \text{by transition 4}\\
&\overset{M}{\Longrightarrow} (q, \#\#\#\#\underline{\#}\#) && \text{by transition 2}\\
&\overset{M}{\Longrightarrow} (p, \#\#\#\#\underline{\#}) && \text{by transition 4}\\
&\overset{M}{\Longrightarrow} (h, \#\#\#\#\underline{\#}) && \text{by transition 3}
\end{aligned}
$$

A deterministic TM's computation on an initial configuration of x, can be viewed as a path in the configuration graph G starting at x and ending at some halted configuration. In fact, for a deterministic Turing Machine, the configuration graph G is composed of an infinite number of such paths, each of which is finite. The computation above is an example of such a path. [6]

2.2. Probabilistic Turing Machines

A probabilistic Turing Machine (PTM) is one in in which a probability is associated with every transition of the Machine. We therefore do not discuss whether a configuration x yields configuration y, but rather say that x yields y with some probability. This also implies that the operation of the machine is not completely determined by it's initial state, but rather by it's initial state and some sequence of random choices. In particular, we will allow an algorithm for a PTM to flip coins (not necessarily fair or double sided) and decide how to proceed based on the outcome. As a point of interest, it can be shown that a machine which is allowed to flip only double-sided fair coins can simulate any probabilistic Turing Machine with only a polynomial slow-down in time. That is to say, allowing the machine to flip unfair multi-sided coins does not decrease the time of the computation by more than a polynomial factor. [3, 4]

Embodying these concepts into a formal setting, we restrict the range of the transition function δ (from definition 2.1) to the interval $[0,1]$. Having done so, we consider $\delta(k_1, \sigma_1, k_2, \sigma_2) = \alpha$ to imply that if the Turing Machine is in state k_1, scanning the symbol σ_1, it will write σ_2 on the tape and go into state k_2 with probability α. The function delta thereby dictates the probability with which each transition of the Turing Machine shall occur.

To ensure that the machine conforms to the laws of probability we must require that given a configuration x, the sum of the probabilities of those configurations which x yields be 1. Formally this requires us to restrict δ in the following way:

For every fixed $k_1 \in K$ and $\sigma_1 \in \Sigma$,

$$(1) \qquad \sum_{k_2 \in K \cup \{h\}, \sigma_2 \in \Sigma \cup \{L,R\}} \delta(k_1, \sigma_1, k_2, \sigma_2) = 1$$

As a consequence, for every configuration x the sum of the probabilities of all reachable configurations from x must be one. [3, 4]

EXAMPLE 2.2. To illustrate the operation of a probabilistic Turing Machine we augment the Turing Machine in Example 2.1 so that erasure of a tape cell occurs with probability $\frac{1}{2}$. If the cell is not erased than a 1 is written in the cell. We define the Turing Machine $M = (K, \Sigma, \delta, s)$ such that $K = \{p, q\}$, $\Sigma = \{0, 1, \#\}$, $s = p$, and,

(1) $\delta(p, 1, q, \#) = \frac{1}{2}$

(2) $\delta(p, 0, q, \#) = \frac{1}{2}$

(3) $\delta(p, 1, q, 1) = \frac{1}{2}$

(4) $\delta(p, 0, q, 1) = \frac{1}{2}$

(5) $\delta(p, \#, h, \#) = 1$

(6) $\delta(q, \#, p, R) = 1$

(7) $\delta(q, 1, p, R) = 1$

(8) $\delta(q, 0, p, R) = 1$

(9) For all other 4-tuples x we define $\delta(x) = 0$.

We can now show the Turing Machine's computation when started in the initial configuration $(p, \#\underline{0}0\#)$.

In the first step, the computation will either write a blank or a one on the tape, each with probability $\frac{1}{2}$ (these are transitions 2 and 4). Thus with probability $\frac{1}{2}$ the machine takes one of two computational paths, both shown below. (The transitions used are shown in brackets.)

$$(p, \#\underline{0}0\#)$$

$\stackrel{M}{\Longrightarrow}$ $(q, \#\underline{\#}0\#)$ transition 2 $\qquad$ $\stackrel{M}{\Longrightarrow}$ $(q, \#\underline{1}0\#)$ transition 4

$\stackrel{M}{\Longrightarrow}$ $(p, \#\#\underline{0}\#)$ transition 6 $\qquad$ $\stackrel{M}{\Longrightarrow}$ $(p, \#1\underline{0}\#)$ transition 7

We have now reached step 3, and once again, the machine has a choice between transitions 2 and 4. Each will be chosen with probability $\frac{1}{2}$, and thus each path above splits into two paths again. Configuration $(p, \#\#\underline{0}\#)$ yields the following two computation paths:

$$(\mathrm{p}, \#\#\underline{0}\#)$$

$\stackrel{M}{\Longrightarrow}$ $(q, \#\#\underline{\#}\#)$ transition 2 $\qquad$ $\stackrel{M}{\Longrightarrow}$ $(q, \#\#\underline{1}\#)$ transition 4

$\stackrel{M}{\Longrightarrow}$ $(p, \#\#\#\underline{\#})$ transition 6 $\qquad$ $\stackrel{M}{\Longrightarrow}$ $(q, \#\#1\underline{\#})$ transition 7

$\stackrel{M}{\Longrightarrow}$ $(h, \#\#\#\underline{\#})$ transition 5 $\qquad$ $\stackrel{M}{\Longrightarrow}$ $(h, \#\#1\underline{\#})$ transition 5

The configuration $(h, \#\#\#\underline{\#})$ appears with probability $\frac{1}{4}$ since transition 2 must be chosen twice in order to get to it (each time with probability $\frac{1}{2}$). For the same reason configuration $(h, \#\#1\underline{\#})$ appears with probability $\frac{1}{4}$.

We now turn our attention to configuration $(p, \#1\underline{0}\#)$ which yields the following two computation paths:

$$(p, \#1\underline{0}\#)$$

$\stackrel{M}{\Longrightarrow}$ $(q, \#1\underline{\#}\#)$ transition 2 $\qquad$ $\stackrel{M}{\Longrightarrow}$ $(q, \#1\underline{1}\#)$ transition 4

$\stackrel{M}{\Longrightarrow}$ $(p, \#1\#\underline{\#})$ transition 6 $\qquad$ $\stackrel{M}{\Longrightarrow}$ $(q, \#11\underline{\#})$ transition 7

$\stackrel{M}{\Longrightarrow}$ $(h, \#1\#\underline{\#})$ transition 5 $\qquad$ $\stackrel{M}{\Longrightarrow}$ $(h, \#11\underline{\#})$ transition 5

As above, the two halted configurations appear with probability $\frac{1}{4}$ each. Note that when we take the sum of the probabilities of each possible halted configuration we get 1.

The computation of a PTM can also be viewed as a leveled tree whose construction is based on the configuration graph G as follows: the root is the initial configuration, and every other node is a child if the parent can yield it with non-zero probability. Thus each level of the tree represents one step in the computation. Since it is possible for a single configuration to yield more than one configuration, it is possible for a level of a tree to contain more than one node, each corresponding to a different configuration. In such a case we say that the machine is in a *superposition* of the configurations. Note that it is possible for a single configuration to be duplicated in the tree by being reached from a different parent node every time. The probability of reaching any particular node in the tree after k steps of the computation is the product of the probabilities along all the edges in the path leading to that node. Therefore the probability of reaching any particular configuration x after k steps is precisely the sum of the probabilities of reaching all those nodes in the k-th level that are labeled by x. Note that if a configuration appears at some level k of the tree then the probability of reaching that configuration after k steps, must be positive (i.e. non-zero). [4]

EXAMPLE 2.3. The configuration tree of the computation in Example 2.2 is illustrated in Figure 2.1.

We can also view the computation in a more mathematical context. We attempt to represent each superposition of configurations by a vector and represent the computation as applying a linear operator to the vector. A necessary and sufficient condition to guarantee that a a superposition can be succinctly represented by a vector, is that the set of all possible configurations must be countable. This indeed is the case, and thus we represent a superposition by a vector know as a *state vector*. [3, 4] Each entry

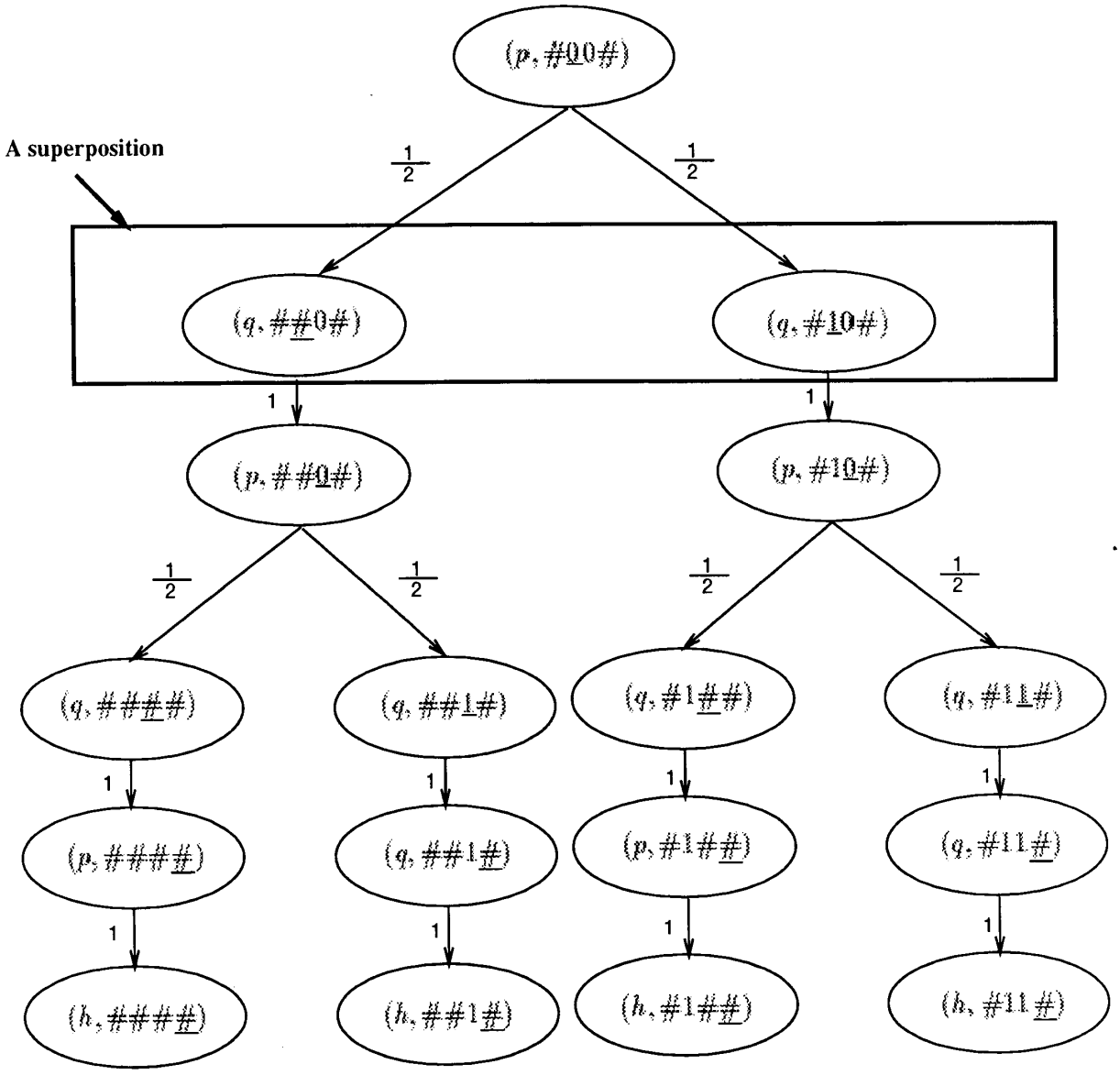

FIGURE 2.1. Leveled Tree of Configurations for the computation in Example 2.2

in the vector will correspond to a particular configuration. If the entry corresponding to a configuration c is p_c, then we will say that the probability of the machine being in state c is p_c. Since such a vector precisely indicates what the probability of each configuration of a superposition is, it in fact gives an accurate description of the state or current condition of the machine. For that reason it is called a *state vector.* Since each vector contains in its entries the probability of each and every possible configuration in the machine, the sum of the probabilities along a particular state vector must be 1, i.e. $\sum_c p_c = 1$. [3] This is due to the fact that the probability space is the set of all configurations, and the sum is across the entire space.

In order to facilitate a short description of the state vector we will adopt the convention of writing only those configurations which have non-zero probabilities. Each such configuration c will be denoted by $|c\rangle$ and a prefix coefficient α_c to indicate it's probability in the superposition. The entire state vector will then be written as the sum

$$\sum_c \alpha_c |c\rangle .$$

In order to simplify some of the analyses below we shall adopt another method of writing a particular configuration, different from the one introduced in definition 2.2. Since the set of configurations is countable, we can encode each configuration by some binary representation. In the discussion that follows, the binary encoding is often used in the representation of a configuration. Moreover, in our discussion of the quantum model, we resort to using the binary encoding in almost all cases, and in many cases where there is no fear of a contradiction we shall consider $|c\rangle$ to denote only the contents of the tape rather than the entire configuration.

EXAMPLE 2.4. Suppose that the state vector has $\frac{1}{2}$ in the entry corresponding to the configuration encoded by 00, $\frac{1}{2}$ in the entry corresponding to the configuration encoded by 11 and 0's everywhere else. Then we write the machine's superposition

as

$$\frac{1}{2}|00\rangle + \frac{1}{2}|11\rangle$$

and the probability of seeing each of the states $|00\rangle$ and $|01\rangle$ is $\frac{1}{2}$.

As the Turing Machine operates the state vector changes and in accordance with the transition function δ. From this perspective we see that δ effectively dictates a linear mapping in the space of superpositions; it maps one superposition to another, or one state vector to another with the corresponding probabilities. This mapping can be represented by a matrix whose rows and columns correspond to configurations enumerated in the same order as they are in the state vector. We require the enumeration to be in the same order so that the entry in the row corresponding to configuration c_1 and the column corresponding to configuration c_2 is the probability with which c_1 yields c_2. This probability is known, since it is the value of δ evaluated at the tuple which transforms c_1 into c_2. For a given PTM, $M = (K, \Sigma, \delta, s)$, we will denote the transition matrix by M_δ, and refer to M_δ as the *time evolution operator.*

EXAMPLE 2.5. From Example 2.2, the entry of M_δ at the column corresponding to configuration $(p, \#1\underline{0}\#)$ and the row corresponding to configuration $(q, \#1\underline{\#}\#)$ is $\frac{1}{2}$. It is exactly $\delta(p, 0, q, \#)$ (This is transition 2.)

Note that along a column corresponding to a particular configuration c in the time evolution operator we have the probabilities with which c yields each possible configuration. Since those configurations which c yields with non-zero probability are dictated by the state of c and the current symbol under the head in c, we conclude that the the sum along each column of the matrix is equal to 1 (by equation 1). [3]

For the purpose of this discussion we will assume that the probabilistic Turing Machine works in a black box, and that its operation is hidden from the user. In order to determine the outcome of a computation, or a partial result in the computation we must make an *observation.* It is easy to see that opening the black box and observing the current state of the machine will fix the machine in a particular state, eliminating

the uncertainty in the operation. Observing a particular bit in the machine's configuration will restrict the set of possible configurations to only those that agree with the outcome of the observation. This will in effect reset all those entries in the state vector that do not agree with the outcome to 0 and normalize the remaining entries to ensure their sum is one. If we view the computation as a tree then observing a bit at a particular level k will in fact eliminate all those nodes in level k and their corresponding children that do not agree with the outcome. Clearly, observing the machine's operation does not perturb the final output distribution. While this last fact is trivially true we mention it for contrast with our new computation model in which this does not hold. [3]

Both the probabilistic Turing Machine and the deterministic Turing Machine are considered classical Turing Machines. They behave in accordance with the laws of classical physics. The operation of both is intuitive to our understanding of the world in which we live. Both are logical consequences of two models of interactions: one which assumes a predetermined universe, in which every action is predictable, and another which presupposes an uncertainty factor to the extent that actions occur with certain probability, and thus a certain factor of non-determinism is involved. In the following chapter, we develop a model of computation which based on the model of interactions among atomic and sub-atomic particles. This model combines both deterministic and probabilistic aspects, and is based on the theory of Quantum Mechanics developed in the middle of this century.

CHAPTER 3
The Quantum Turing Machine

3.1. Quantum Phenomena

In the late nineteenth and early twentieth century a number of seemingly mysterious phenomena were observed. A series of experiments with small atomic and subatomic particles, seem to have produced results which contradicted all contemporary theories of particle interactions. A number of theories conjectured at the time attempted to explain the eccentric results of these experiments with little success. As a precursor and motivation to our discussion of the Quantum Turing Machine, we present one of these experiments that exhibits the prominent characteristics and effects of Quantum Mechanics – the double slit experiment.

Suppose that we were to place an electron gun in front of a wall which has a slit large enough for one electron to pass through. Behind the wall we place a backstop which absorbs the electrons and prevents them from going further. The backstop will also have a detection mechanism which records the distance from the centre of the backstop to the position where an electron was stopped and absorbed. We will also assume that our electron-gun is a poor one, and thus does not necessarily shoot the electrons directly, but rather scatters them across an angle. Since the electrons could hit the sides of the slit and bounce, they could essentially end up anywhere on the backstop; however, we expect most of the electrons to be located around the centre of the screen. This indeed happens as seen by the diagram in Figure 3.1, which shows

both the experiments and the number of electrons recorded at every distance, x, from the centre of the slit in a given period of time. [8]

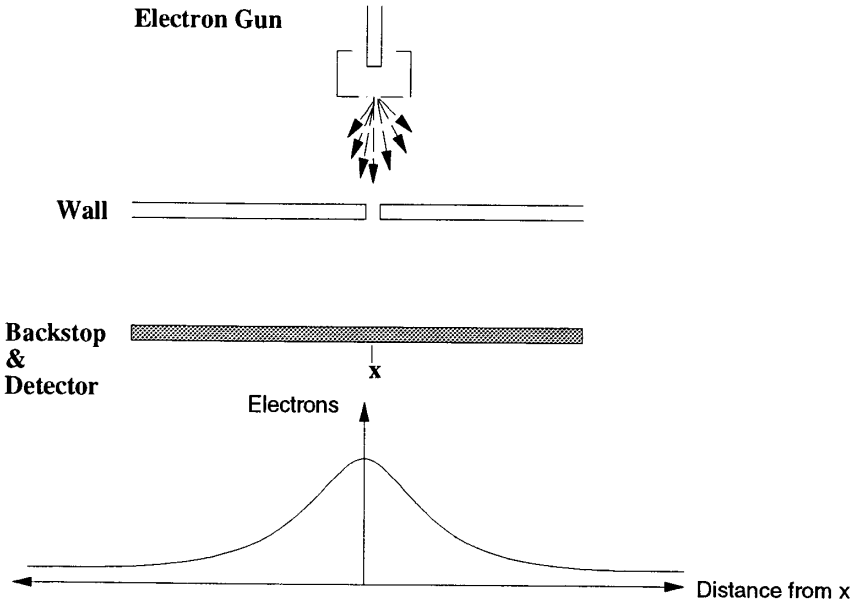

FIGURE 3.1. Single Slit Distribution of Electrons

We now consider what would happen if we were to have two slits in the wall. It would be reasonable for us to expect that the electrons from hole 1 would be distributed as shown by the graph of $P1$ and the electrons from hole 2 would be likewise distributed as in the graph of $P2$, both shown in Figure 3.2. The total scattering of electrons would thus be expected to be the sum of the two distributions as shown in the graph $P = P1 + P2$ in Figure 3.3. However the total scattering turns out to be drastically different and is shown in Figure 3.4.

It turns out that the resultant graph is a consequence of a phenomenon known as interference. The electrons going through the slits interfere with each other to exhibit the graph's strange behaviour. The points on the graph where the concentration of electrons is high are places where constructive interference took place, whereas the

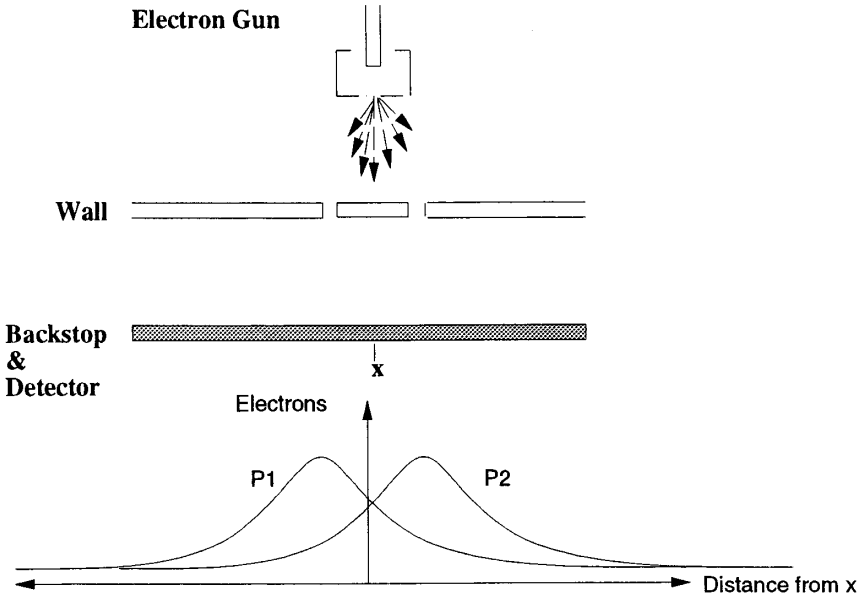

FIGURE 3.2. Double Slit Experiment and Expected Outcome

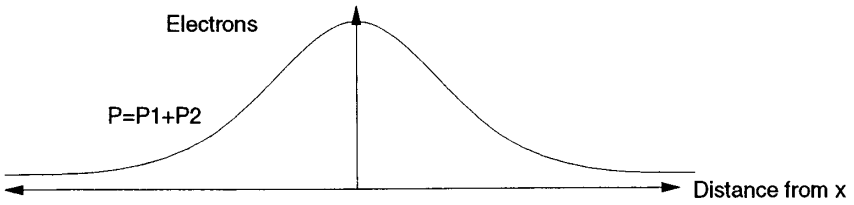

FIGURE 3.3. Expected Total Distribution of Electron Scattering

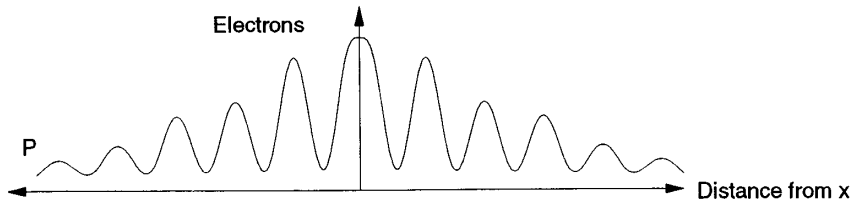

FIGURE 3.4. Actual Total Distribution of Electron Scattering

points of low concentration are points where destructive interference took place.[8] In algorithms used by our new model of computation, the effects of constructive and destructive interference are manipulated such that constructive interference occurs at places corresponding to desirable outputs, whereas destructive interference occurs at places corresponding to undesirable or incorrect outputs.

However, there is no free lunch. The effects of constructive and destructive interference are not handed to us without cost. Being able to observe the output distribution as shown above, is conditional upon our not knowing through which of the two holes a given electron passes. In fact, an interesting thing occurs when we try to ascertain this. If we place a light source next to the wall, so that we can detect a scattering of the light by the electron as it passes through one of the two holes, we will be able to determine which of the two holes the electron passes through. When we start shooting electrons out of the electron gun, we find that the distribution of electrons is perturbed, and now looks like the one in Figure 3.3. [8] The implication is of course that our observance of the electrons as they pass through the slit, actually perturbs their output distribution.

This second phenomenon of our observation of the experiment affecting the actual outcome is a consequence of Heisenberg's Uncertainty Principle. Applying this Principle to our experiment, it essentially states that we can not determine at the same time both the distribution of the electrons and the hole through which they passed. We can measure one or the other but not both. Until the advent of Quantum Mechanics both interference and Heisenberg's Uncertainty principles could not be explained. Theories which explained one phenomenon or the other existed, however, it was only Quantum Mechanics which managed to reconcile both effects, and incorporate them into one unifying theory.

The model we introduce, known as the quantum Turing Machine is based on the theory of Quantum Mechanics. It will thus be subject to both Heisenberg's uncertainty principle and interference. As we shall see this has interesting consequences,

and yields intriguing results.

3.2. The Model

A quantum Turing Machine (QTM) is one which is similar in many respects to the probabilistic Turing Machine. The main difference between the two models is that while in a probabilistic Turing Machine we associate with each transition a probability which can be any real number in the interval $[0, 1]$, in a quantum Turing Machine we have amplitudes which are any complex number. The entries of the state vector are therefore amplitudes that relate to the probability of seeing a given configuration in the following way: If the amplitude of a particular configuration c in some superposition is α, then the probability of seeing c when observing the machine in this superposition is $|\alpha|^2$ [3]. As before we will adopt the shorter summation notation in writing the state vector. For example if configurations $|00\rangle$ and $|11\rangle$ have amplitudes $\frac{1}{\sqrt{2}}$ and $\frac{-1}{\sqrt{2}}$ respectively and all other configurations have amplitude 0 then we write the state vector or superposition as:

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{-1}{\sqrt{2}}|11\rangle$$

Upon observation of this superposition, we see configuration $|00\rangle$ with probability *half*, and configuration $|11\rangle$ with probability $\frac{1}{2}$. As before we must require that the model conform to probabilistic axioms, and consequently we require that the sum of the squares of the magnitudes of the amplitudes, (i.e. the sum of the probabilities) in the state vector be 1. Now, consider a state vector whose entries are the amplitudes $\alpha_1, \alpha_2, \alpha_3, \ldots$. We require that $|\alpha_1|^2+|\alpha_2|^2+|\alpha_3|^2, \ldots = 1$. This is equivalent to insisting that $\sqrt{|\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2, \ldots} = 1$. However, this latter quantity is the Euclidean length of the state vector, and therefore requiring the sum of the probabilities to be 1 is equivalent to requiring the state-vector to preserve unit length. We will therefore say that a QTM is *well-formed* if it's operation preserves length. [3]

To ensure that the vector possesses unit length, we must insist that the Turing Machine's time evolution operator preserve length. As the following theorem by Bernstein and Vazirani shows this is equivalent to requiring the time evolution operator M_δ to be a unitary matrix (one such that $M_\delta^* M_\delta = M_\delta M_\delta^* = I$, where M_δ^* is the transpose conjugate of M_δ.)

THEOREM 3.1. *A QTM $M = (K, \Sigma, \delta, s)$ is well-formed if and only if it's time-evolution operator is unitary* [3]

PROOF For every $i \in N$, let $\vec{e_i}$ be the state vector with a 1 in the i-th entry and 0 everywhere else. Let M_δ be a unitary time-evolution operator and let $\vec{v}$ and $\vec{w}$ be state vectors. We write $\vec{v} = \sum_{i \in N} \alpha_i \vec{e_i}$ and $\vec{w} = \sum_{j \in N} \beta_j \vec{e_j}$, and compute the inner product,

$$\begin{aligned}
\langle M_\delta \vec{v}, M_\delta \vec{w} \rangle &= \left\langle M_\delta \sum_{i \in N} \alpha_i \vec{e_i}, M_\delta \sum_{j \in N} \beta_j \vec{e_j} \right\rangle \\
&= \left\langle \sum_{i \in N} \alpha_i (M_\delta \vec{e_i}), \sum_{j \in N} \beta_j (M_\delta \vec{e_j}) \right\rangle \\
&= \sum_{i \in N} \sum_{j \in N} \langle \alpha_i (M_\delta \vec{e_i}), \beta_j (M_\delta \vec{e_j}) \rangle \\
&= \sum_{i \in N} \sum_{j \in N} \alpha_i \bar{\beta}_j \langle M_\delta \vec{e_i}, M_\delta \vec{e_j} \rangle
\end{aligned}$$

(The bar denotes the complex conjugate.) However, multiplying the vector $\vec{e_i}$ by the matrix M_δ is equivalent to selecting the i-th column in M_δ which we will denote by $\vec{c_i}$. Furthermore, note that M_δ's unitary property implies that the inner product

$\langle \vec{c_i}, \vec{c_j} \rangle = 0$ if $i \neq j$ and $\langle \vec{c_i}, \vec{c_j} \rangle = 1$ if $i = j$. Thus the above reduces to:

$$\begin{aligned} &= \sum_{i \in N} \sum_{j \in N} \alpha_i \bar{\beta}_j \langle \vec{c_i}, \vec{c_j} \rangle \\ &= \sum_{i \in N} \alpha_i \bar{\beta}_i \langle \vec{c_i}, \vec{c_i} \rangle \\ &= \sum_{i \in N} \alpha_i \bar{\beta}_i = \langle \vec{v}, \vec{w} \rangle \end{aligned}$$

Whence, M_δ unitary implies that M_δ preserves length.

Conversely, suppose that M_δ preserves length. We first show that $M_\delta^* M_\delta = I$. To that end, observe that the entry in the i-th row and j-th column of the matrix $M_\delta^* M_\delta$ is the inner product $\langle \vec{c_j}, \vec{c_i} \rangle$, due to the fact that the columns of a unitary matrix are an orthonormal set. Since M_δ preserves length,

$$\langle \vec{e_j}, \vec{e_i} \rangle = \langle M_\delta \vec{e_j}, M_\delta \vec{e_i} \rangle = \langle \vec{c_j}, \vec{c_i} \rangle$$

and $\langle \vec{e_j}, \vec{e_i} \rangle = 0$ if $i \neq j$ and 1 otherwise. Consequently, we conclude that the entry in the i-th row and j-th column of the matrix $M_\delta^* M_\delta$ is 1 if $i = j$, and 0 otherwise, which is precisely the description of I. So $M_\delta^* M_\delta = I$. If M_δ is onto, then it has an inverse in which case

$$\begin{aligned} M_\delta M_\delta^* &= M_\delta M_\delta^*(I) \\ &= M_\delta M_\delta^*(M_\delta M_\delta^{-1}) \\ &= M_\delta (M_\delta^* M_\delta) M_\delta^{-1} \\ &= M_\delta I M_\delta^{-1} \\ &= M_\delta M_\delta^{-1} = I \end{aligned}$$

So suppose M_δ is not onto. Then there must be some configuration c which can not be reached from any other configuration. However, this implies that any configuration that looks locally like c can not be reached. We say that a configuration looks locally like c if it is one that in in the same state as c, is scanning the same symbol as c and has the same symbols written to the immediate left and right of it's tape head.

Such a configuration can not be reached since any way to reach it would also imply a way to reach c. (The transition function is defined independently of c.) Now consider any n consecutive cells, and let S_n be the set of configurations that have their tape head inside those cells and whose tapes are blank everywhere else. The number of possible such configurations is $|S_n| = n|K||\Sigma|^n$, since n is the number of possible head positions, $|\Sigma|^n$ is the number of possible distinct contents of n cells, and $|K|$ is the number of possible states. It is clear that the set of configurations reachable from configurations in S_n are those that are in S_n already in addition to the configurations that are reached by moving out of the n cells. The number of configurations reachable from S_n by moving out of the n cells is at most $2|\Sigma|^n|K|$ since the n cells could still have $|\Sigma|^n$ possible contents in conjunction with $|K|$ different states, however there are now only 2 possible head positions, namely those corresponding to the left and right ends of the n cells. In total therefore we have $(n+2)|K||\Sigma|^n$ possible configurations that can be reached from configurations in S_n. However, the number of configurations in S_n that look locally like c is $(n-2)|\Sigma|^{n-3}$ because we are not restricted to the contents of $n-3$ cells, and the three contiguous cells around the head (that give the configuration the look of c) can appear in any one of $n-2$ different positions. This leaves us with only $(n+2)|K||\Sigma|^n - (n-2)|\Sigma|^{n-3}$ possible reachable configurations, which is less than $n|K||\Sigma|^n$ if we choose $n > |K||\Sigma|^3 + 2$.

This last fact shows that, in total, the configurations in S_n can reach at most r configurations where $r < |S_n|$, thereby showing that in the every column corresponding to a configuration in S_n, there are at most r non-zero entries, and each of those entries appears along some set of r rows. Consequently, this implies that the columns corresponding to the elements in S_n can not all be mutually orthogonal, contradicting the fact that $M_\delta^* M_\delta = I$. So M_δ must be unitary. Note that the proof was dependent on the fact that the machine has a two way infinite tape, and that the theorem would not hold had the tape been infinite in only one direction. □[3]

The theorem, while suggesting a very rigid condition to ensure a machine is well

formed is in practice difficult to apply as it would mean verifying that an infinite dimensional matrix is unitary. However, if we restrict the number of bits encoding the configuration to some finite number n (Note that this will of course limit the number of tape cells we use) then we are only required to check that the matrix operating on those n bits is unitary, as all other configurations essentially play no part in the computation and we can thus disregard them. In general this matrix will be of size $2^n \times 2^n$. Furthermore, if the operation is concerned with only a certain number of bits k then it is sufficient for us to specify a unitary matrix operating on those k bits, as this will not change the larger matrix. [3, 4] Perhaps an example is at this point in order.

EXAMPLE 3.1. Consider a Turing machine whose operation can be fully described by configurations of 3 bits, i.e. a configuration can be encoded as one of 000, 001, 010, 011, 100, 101, 110 and 111. Then applying the unitary matrix

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

to the last bit in the configuration should have the effect of flipping a fair coin on the last bit. This is equivalent to applying the matrix

$$\begin{array}{c} \\ 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array} \begin{array}{c} \begin{array}{cccccccc} 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \end{array} \\ \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix} \end{array}$$

to all three bits. The numbering on the top and to the side of the matrix, indicates the

ordering used in the state-vector and the matrix. If we observe the column indexed by 010 we notice that it has two non-zero entries, each containing the amplitude $\frac{1}{\sqrt{2}}$ – one corresponding to the row 010 and the other corresponding to the row 011. This means that the configuration 010 would yield configurations 011 and 010 each with amplitude $\frac{1}{\sqrt{2}}$. In other words all bits stay the same with the exception of the last whose bit is flipped with amplitude $\frac{1}{\sqrt{2}}$. More generally, applying an operator M $(2^k \times 2^k)$ to the last k bits in an n-bit configuration scheme is equivalent to applying the Kronecker product $M \otimes I$ to all n bits where I is the $2^{(n-k)} \times 2^{(n-k)}$ identity matrix. (The Kronecker product of two $n \times n$ matrices $A = (a_{ij})$ and $B = (b_{ij})$ is defined to be $A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & & \vdots \\ a_{n1}B & \dots & a_{nn}B \end{bmatrix}$ in block form.) It is clear that such an operation preserves the unitarity of the computation and we will thus not specify the entire matrix but rather only that portion which pertains to the bits we are interested in operating on. [4, 9]

This suggests that for the Turing Machine to be well-formed there may be some characterizations which are easier to verify given finite means. Towards that end the following theorem was proved by Bernstein and Vazirani:

THEOREM 3.2. *A QTM $M = (K, \Sigma, \delta, s)$ is well-formed if and only if the following three conditions hold:* [3]

(1) *The superposition of state-symbol pairs (i.e. resulting configurations) leaving any particular state-symbol pair has unit length, i.e. for all $(p, \sigma) \in K \times \Sigma$,*

$$\sum_{q,\tau} |\delta(p, \sigma, q, \tau)|^2 = 1$$

(2) *The superpositions of symbol and new state leaving any two different state-symbol pair are orthogonal, i.e. for all $(p_1, \sigma_1) \neq (p_2, \sigma_2) \in K \times \Sigma$ we have*

$$\sum_{q,\tau} \delta(p_1, \sigma_1, q, \tau)\delta^*(p_2, \sigma_2, q, \tau) = 0$$

(3) *The space of all linear superpositions of states,* $\mathbb{C}^{|K|}$*, consists of three mutually orthogonal subspaces* Q_L*,* Q_R *and* Q_S*, the first two corresponding to a direction of movement, and the last corresponding to transitions which write a symbol under the head, such that if we fix all of* δ*'s parameter except for the new state so that it dictates a head movement to either the left or right or a writing of a symbol, then the resulting superposition will lie in the subspace corresponding to the movement of the tape head, or the writing of a symbol. Formally,* $\exists Q_L, Q_R, Q_S \subseteq \mathbb{C}^{|K|}$ *mutually orthogonal subspaces, such that the following hold:*

(a) *For all* $(p, \sigma) \in K \times \Sigma$,

$$\sum_{q \in K} \delta(p, \sigma, q, R) \, |q\rangle \in Q_R$$

(b) *For all* $(p, \sigma) \in K \times \Sigma$,

$$\sum_{q \in K} \delta(p, \sigma, q, L) \, |q\rangle \in Q_L$$

(c) *For all* $(p, \sigma, \tau) \in K \times \Sigma \times \Sigma$,

$$\sum_{q \in K} \delta(p, \sigma, q, \tau) \, |q\rangle \in Q_S$$

PROOF By Theorem 3.1, for M to be well-formed, the transition function M_δ must be unitary, and M_δ is unitary if and only if the columns of M_δ are orthonormal. Hence, in order to prove the theorem it suffices to show that the above three conditions are equivalent to stating that the columns of M_δ are orthonormal. It is clear that the first condition states that each column has unit length, for it claims that the sum of all possible amplitudes leaving a particular state-symbol pair, and hence a particular configuration, is 1. It remains to show that the last two conditions are equivalent to the columns of M_δ being orthogonal. Towards that end we consider two columns of M_δ labeled by configurations a and b. Clearly, if configurations a and b differ by a cell that is not under the tape head in configuration a, and is not under the

tape head in configuration b, then the set of possible configurations which a yields is disjoint from the set of possible configurations that b yields. This is because the machine can not change cells that are not under the head in one step. Consequently, there will be no entry which is non-zero in both columns a and b of M_δ, and thus the inner-product of the two columns will be zero. Furthermore, this argument extends to those configurations whose tape head is more than 2 cells apart. So it is only those configurations whose heads are within two cells of each other that can possibly yield similar configurations. Now, the second condition takes care of all those cases in which the state and symbol are distinct under the two tape heads, making sure that such columns are orthogonal. Consequently, two configurations can not have their heads on the same cell, be reading two distinct symbols, and yielding two non-orthogonal superpositions. It remains to ensure that those configurations whose tapes are the same in content, and are one or two cells apart in their head position, yield two orthogonal superpositions. The third condition does that adequately since it ensures that movements to the left, to the right and the writing of a symbol remain mutually orthogonal. A careful exhaustion of all possible ways in which two configurations that are one or two cells apart shows that indeed this suffices. Thus the above three conditions are equivalent to column orthonormality in M_δ and hence to the well-formedness of M_δ. □[3]

If we observe the conditions in Theorem 3.2 it is easy to see that they are independent of the time evolution operator in the sense that for a given machine M, each one can be verified without referring to the operator. In fact, each one of the conditions can be checked and ascertained to be true by finite means. We only require knowledge of the definition of the Turing Machine. The above conditions for that reason are known as locality constraints, since they are concerned with the operation of the machine on a single tape cell rather than on the entire tape. These locality constraints allow us to specify a local unitary matrix as part of a computation (or algorithm) on a Quantum Turing Machine. In practice, we already assumed this property in

Example 3.1.

3.3. Interference

Just like a Probabilistic Turing Machine, the quantum Turing machine's computation can also be viewed as a leveled tree, with each level representing a step in the computation. The amplitude of reaching any particular node is the product of the amplitudes of all the edges in the path to the node, and as with probability in the PTM, the amplitude of reaching a particular configuration x after k steps in the computation is precisely the sum of the amplitudes of reaching all those nodes in the k-th level that are labeled by x. [4]

While it appears as though the QTM's computation is similar to the PTM's, the fact that the value of the amplitude function can be any complex number leads to curious effects. Consider for example a configuration x, that is duplicated exactly twice at the k-th level of the tree at nodes 1 and 2. Suppose that the amplitude of reaching node 1 is a_1 and the amplitude of reaching node 2 is a_2. Then, by definition the amplitude of reaching configuration x is $a_1 + a_2$. The probability of reaching that configuration is therefore $(a_1 + a_2)^2$. However, if $a_1 = -a_2$, then the probability of reaching configuration x is 0, notwithstanding it's appearance in the tree. This has interesting consequences, for it means that the computation of a function on a quantum Turing Machine is affected by all other computations paths. Sounds familiar? This is precisely the phenomenon of interference from section 3.1 in our quantum Turing Machine. This effect is very heavily used in the algorithms below. [3, 4]

With interference, enters the issue of Heisenberg's Uncertainty Principle. Applied to our machine it states that the observation of the machine's operation actually affects the distribution of the outcomes. As an example, consider a machine in the configuration $|0\rangle$. We apply what is considered to be the one bit quantum coin flip

unitary operator,

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

to the bit in the configuration and arrive at the superposition

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Suppose that we decide to observe the bit at this stage of the computation. We will see $|0\rangle$ with probability $\left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$ and $|1\rangle$ with probability $\left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$. We now apply U to the outcome and arrive at either

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

or

$$\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

depending on whether the outcome from the first trial was $|0\rangle$ or $|1\rangle$ respectively. In either case we see $|0\rangle$ with probability $\frac{1}{2}$ and $|1\rangle$ with probability $\frac{1}{2}$. This makes sense in a classical setting since flipping a coin twice should not affect the output distribution of the second coin flip.

However, consider what would have happened had we not observed the outcome after the first step. We would then apply U to the superposition

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

and arrive at the superposition

$$\frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) + \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = |0\rangle$$

The output is undistributed, and we have effectively unflipped the coin. The only difference between the first experiment and the second is that in the first case we observed the outcome after the first coin flip, whereas in the second we flipped the bit twice without observing it midway through the computation. This means that the our act of observing the machine as it operates actually has an effect on the outcome.

The implications and questions this raises are numerous.

The first question that must be asked is, If we can't observe the machine, how will we ever know that it has halted? This turns out to be nothing more than a technical detail. To overcome this difficulty we alter the machine so that it contains a cell called the *acceptance cell* whose contents will indicate whether the machine has halted or not. We observe this cell after every step in the computation to determine whether the machine has or hasn't halted. It is clear that if the machine has not halted, our observation of that particular cell does not perturb the distribution of the configurations, as all possible configurations have not halted. Moreover, our observation becomes inconsequential once the machine has halted since at that point we are interested in observing all the cells and not just the acceptance cell. [3]

Having surpassed the technical difficulties, we come to the deeper, more profound questions. Is the interference phenomenon an advantage or a disadvantage of the quantum Model? Assuming it is not a disadvantage, how big an advantage is it? Can the quantum model solve problems more efficiently than the classical models? Moreover, is it possible that interference allows us to compute non-recursive functions on the quantum Turing Machine, thereby allowing us to solve classically undecidable problems such as the halting problem? If we can not compute non-recursive functions, then what limitations does the quantum Turing Machine have? The next theorem is an attempt to answer the first of these questions. The next few theorems and the remainder of this paper attempt to provide partial responses to some of these questions. Admittedly, not all these questions are adequately answered, and furthermore we find that it is unlikely for us to solve some of these problems as their solution settles other problems which have been open for a long time.

CHAPTER 4
Efficient Computation

In order to be able to answer questions regarding the relative efficiency between the classical Turing Machines and the quantum Turing Machine, we must be able to have some measure of what we call an efficient computation. Towards that end we must define how we measure the amount of time a computation takes. In the case of a Turing Machine the length of the computation, or the number of transitions shall be considered the amount of time required for the computation. Clearly the length of time that a machine requires to compute any reasonable function is dependent upon the length of the input. Firstly, a longer input will take longer to read, and secondly, the function itself may have properties that require an increase in the computation time for longer inputs. In classical complexity theory it is generally accepted that for most problems an efficient solution is one which takes time that is no longer than some fixed polynomial in the length of the input. The class of languages that can be decided in polynomial time by a deterministic Turing Machine, is the well known class P. An analogous class, BPP, exists for the probabilistic Turing Machine. For reference a definition of both is given.

DEFINITION 4.1. *A language L is in the class P if there exists a deterministic Turing machine M with a distinguished acceptance cell y and a polynomial p such that given any string x the contents of the acceptance cell y at time $p(|x|)$ correctly classifies x with respect to L.* [7]

DEFINITION 4.2. *A language L is in the class BPP if there exists a probabilistic Turing Machine M with a distinguished acceptance cell y and a polynomial p such that given any string x the contents of the acceptance cell y at time $p(|x|)$ correctly classifies x with respect to L with error at most 1/3. That is to say,*

if $x \in L$ then y indicates that $x \in L$ with at least $\frac{2}{3}$ probability; and
if $x \notin L$ then y indicates that $x \notin L$ with at least $\frac{2}{3}$ probability.

For a complete discussion of the above two classes see *Computational Complexity* [7].

We now introduce two new classes – EQP and BPP. EQP is the class of languages that can be decided in polynomial time by a quantum Turing Machine with certainty, while BQP is the class of languages decidable on a quantum Turing Machine in polynomial time with non-negligible probability.

DEFINITION 4.3. *A language L is in the class EQP (exact or error-free quantum polynomial time) if there exists a quantum Turing Machine with a distinguished acceptance cell and a polynomial p, such that given any string x as input, observing the acceptance cell at time $p(|x|)$ correctly classifies x with respect to L. Furthermore, L is in the class BQP (bounded-error quantum polynomial time) if we allow the classification to be correct within a two-sided error of at most 1/3.* [3]

At this point we have the tools necessary to start comparing the efficiency of the classical Turing Machine to the Quantum Turing Machine. The first observation we make is that the Quantum Turing Machine possesses no less computational power from a complexity theoretic standpoint than its classical predecessors. As evidenced by the following theorem, anything which can be computed in polynomial time on a classical Turing Machine can also be computed in polynomial time on a quantum Turing machine:

THEOREM 4.1. *$P \subseteq EQP$ and $BPP \subseteq BQP$.* [3]

PROOF To show both statements it would suffice to prove that anything which can be efficiently computed by a classical Turing Machine can be efficiently computed by a quantum Turing Machine. We begin by showing that the randomness in a PTM can be efficiently simulated on an QTM. This is due to the fact that it suffices for us to simulate a one bit fair coin flip on a quantum Turing Machine, which can be easily done by the unitary transformation shown before. Should we need to flip a coin again, we simply do so on a different part of the tape, thereby ensuring that no interference takes place. Thus, it remains to show that any deterministic computation can be efficiently simulated on a quantum Turing Machine. Towards that end we consider the class of reversible deterministic Turing Machines: a Turing Machine M is said to be reversible if reversing the arrows in the configuration graph gives a deterministic mapping of configurations M' that reverses the operation of M, i.e. if M transforms configuration c to d, then M' transforms configuration d back to c. It is clear that not all Turing Machines are reversible. In particular consider a Turing Machine which computes a function f on input x such that only $f(x)$ is written on the tape when the machine has halted. Then if f is not one-to one it is clear that the machine can not be reversible. However, an important result by Bennett [10] shows that for any Turing Machine M computing a function f(x) on the tape, there exists a reversible Turing machine $\hat{M}$ which computes $g(x) = (x, f(x))$ with only a constant slow down in time. In other words, a function can be computed reversibly provided that the input is not erased from the tape. Therefore, to prove the theorem it suffices to show that any classical reversible Turing Machine is also a quantum Turing Machine. However this is almost trivial, since if we observe the time evolution operator M of a reversible deterministic Turing machine, it has nothing but 0 and 1 entries in all columns. Since it's reversible, there is no row or column with two or more 1's (otherwise it computes a function that is not one-to-one). Furthermore, since M is reversible it must be onto, and thus each row and column contain exactly one entry that is 1. A simple observation reveals that the conjugate transpose of such a mapping is it's inverse

mapping, which implies that M is unitary. Applying Theorem 3.1 we conclude that M is a well-formed QTM. In light of our previous observations, this suffices to show that $P \subseteq EQP$ and $BPP \subseteq BQP$ □[3]

In the above proof we used Bennett's result, for reference, we state Bennett's result in the following theorem:

THEOREM 4.2. *Let* $f : \{0,1\}^n \rightarrow \{0,1\}^m$ *be some function, that is computable in polynomial time, p, by some classical deterministic Turing machine. Then, there exists a reversible deterministic Turing Machine M (and hence a quantum Turing machine), such that given* $|x, y\rangle$ *on the tape, where* $x \in \{0,1\}^n$ *and* $y \in \{0,1\}^m$*, (we omit the state and head position here) M computes* $|x, y \oplus f(x)\rangle$ *in time at most cp, for some constant c.* [10]

Theorem 4.1 proves essentially that anything which is efficiently computable by a classical Turing Machine is also efficiently computable by a quantum Turing Machine. At the very least, this suppresses any concerns that the QTM's well-formedness constraint imposes a disadvantage on the model. However as we shall now see there appears to be evidence that the quantum Turing Machine can efficiently compute functions that are not easily computable by any classical Turing Machine.

CHAPTER 5

The XOR Mask Problem and its Quantum Solution

The first evidence that there may be languages in BQP which are not in BPP, came to light in the form of a promise problem. In 1994, Daniel Simon presented a problem for which there appears to be no classical solution in polynomial time, yet has a very elegant solution on a quantum Turing Machine. While Simon's problem is somewhat concocted with no immediately apparent applications, it does serve to highlight the potential promise in the quantum Turing Machine. The problem, known as the XOR Mask problem is presented below: [4]:

PROBLEM 5.1. *XOR Mask. Given a deterministic Turing Machine that computes a 2 to 1 function* $f : \{0,1\}^n \to \{0,1\}^{n-1}$ *such that for all* $x, y \in \{0,1\}^n$, $f(x) = f(y)$ *iff* $x = y \oplus s$ *for some mask* $s \in \{0,1\}^n$. *Find* s. [4]

In this problem, a description of a Turing Machine is given. It is promised that this Turing machine computes some 2 to 1 function from $\{0,1\}^n$ to $\{0,1\}^{n-1}$. Moreover, this function is also promised to have the property that there exists some string $s \in \{0,1\}^n$ such that whenever two string $x, y \in \{0,1\}^n$ are mapped to the same value in $\{0,1\}^{n-1}$, their XOR value, i.e. $x \oplus y$ is s. We give an example of such a function:

EXAMPLE 5.1. Let $n = 3$. Then, we would like to describe a function that computes a 2 to 1 function from $\{000, 001, 010, 011, 100, 101, 110, 111\}$ to $\{00, 01, 10, 11\}$. If we denote the three bits as (x_1, x_2, x_3), then the function $g(x_1, x_2, x_3) = (x_1, x_2)$ is

a 2-1 function such that $g(x) = g(y)$ iff $x \oplus y = s$ for $s = 001$. To see this we observe that $g(x_1, x_2, x_3) = g(y_1, y_2, y_3)$ iff $(x_1, x_2) = (y_1, y_2)$. Hence the function will produce the following mapping:

$$g(x) = \begin{cases} 00 & \text{if } x = 000 \text{ or } x = 001 \\ 01 & \text{if } x = 010 \text{ or } x = 011 \\ 10 & \text{if } x = 100 \text{ or } x = 101 \\ 11 & \text{if } x = 110 \text{ or } x = 111 \end{cases}$$

Note that $000 \oplus 001 = 010 \oplus 011 = 100 \oplus 101 = 110 \oplus 111 = 001$. Hence, g is an example of a promise function described in problem 5.1.

A careful examination of the above example reveals that the function could have been constructed in at least 7 different ways, each corresponding to a different mask. The only mask which could not be used in this example is 000, since such a mask forces the function to be 1-1 rather than 2-1. (Clearly there is no 1-1 function from $\{0,1\}^n$ to $\{0,1\}^{n-1}$.) In general, the mask used by the function in problem 5.1 can take on $2^n - 1$ distinct values, i.e. any value from the set $\{0,1\}^n \setminus \{0^n\}$. The enormous number of possible values for s is the source of the problem's difficulty in the classical sense.

In the following theorem we show that if the Turing Machine M from problem 5.1 is given as an oracle, than there is no probabilistic Turing Machine that can find the mask s in polynomial time with non-negligible probability for any M. In particular what we show is that there exists a 2-1 function f such that if M is an oracle for a 2-1 function which is randomly chosen from a subset of all 2-1 function satisfying the condition in problem 5.1 then the chances of a probabilistic Turing Machine identifying s is small.

THEOREM 5.1. *For any Turing Machine M using at most k steps, there exist a 2-1 function* $f : \{0,1\}^n \to \{0,1\}^{n-1}$, *such that* $f(x) = f(y)$ *iff* $x = y \oplus s$, *and an oracle O*

computing f, such that M can not correctly output s with probability greater than $\frac{k^2}{2^n-1}$.

Note that if k is a polynomial, than the probability that the answer is correct is exponentially small.

PROOF We first construct a set of functions F of size $2^n - 1$ such that each function $f_i \in F$ is 2 to 1 from $\{0,1\}^n$ to $\{0,1\}^{n-1}$ and $f(x) = f(y)$ iff $x = y \oplus s_i$ where the s_i's are distinct. Thus, each function in F satisfies the promise in problem 5.1 and each such function has a distinct mask s. To that end, let $s_i = i$ for all $i \in \{0,1\}^n \setminus \{0^n\}$. Then let

$$D_i = \{(x,y) | x \in \{0,1\}^n; y \in \{0,1\}^n; x \oplus y = s_i\}$$

Observe that the size of D_i is 2^{n-1}, hence there exist approximately $(n-1)!$ one-to-one and onto functions $g : D_i \to \{0,1\}^{n-1}$. We randomly choose g, and define $f_i : \{0,1\}^n \to \{0,1\}^{n-1}$ as follows:

$$f_i(x) = g((x, x \oplus s_i))$$

Then define

$$F = \{f_i | i \in \{0,1\}^n \setminus \{0^n\}\}.$$

It's clear that F is indeed as required.

Let M be a Probabilistic Turing Machine using k steps. To show the existence of a function as in the statement of the theorem we use a probabilistic technique. We show that if a function f_i is randomly chosen from F then the probability that M guesses s_i correctly given f_i as an oracle, O, is negligible. To that end, let f_i be chosen randomly uniformly from F. Then, since the number of steps the Turing Machine uses is k, it can make at most k queries to the oracle. We will call a set of k distinct queries to the oracle $\{m_1, m_2, m_3, ..., m_k\}$ good if there exist $p, q \in \{1,2,3,..k\}$, $p \neq q$ such that the oracle returns the same answer on both m_p and m_q. For any particular m_p, querying the oracle returns no information other than $O(m_p)$, except to the extent

that $O(m_p)$ can be the same or different from $O(m_q)$, because the function g above was randomly chosen. For that reason, we may consider the choices of the m_j's to be independent of each other. Now, the only way that the machine would be able to find s is if it had a good set of queries. The probability that a set of k queries is good is,

$$\begin{aligned} & \Pr[O(m_p) = O(m_q) \text{ for some } p, q \in \{1, 2, 3, ..k\},\ p \neq q] \\ = \ & \Pr[m_p \oplus m_q = s_i \text{ for some } p, q \in \{1, 2, 3, ..k\},\ p \neq q] \\ \leq \ & \sum_{\substack{p,q \in \{1,2,3,..k\} \\ p \neq q}} \Pr[m_p \oplus m_q = s_i] \end{aligned}$$

We can now show that the $\Pr[m_p \oplus m_q = s_i]$ is small since that is the same as the probability with which s_i equals some particular string. Since f_i was chosen uniformly and randomly from F and since s_i is distinct for every f_i, the probability that s_i is chosen is the same as the probability of choosing f_i. Hence we have that

$$\Pr[m_p \oplus m_q = s_i] = \Pr[\text{choosing } f_i] = \frac{1}{2^n - 1}.$$

The sum above now reduces to

$$\begin{aligned} & \textstyle\sum_{p,q \in \{1,2,3,..k\}, p \neq q} \frac{1}{2^n - 1} \\ = \ & \tfrac{1}{2^n - 1} \textstyle\sum_{p,q \in \{1,2,3,..k\}, p \neq q} 1 \\ < \ & \tfrac{k^2}{2^n - 1}. \end{aligned}$$

What we have shown is that if f is a random function from F than the probability that M correctly guesses s is small. Hence there must exist some function for which this is true, proving the theorem. [4] $\square$

The above theorem gives some evidence to the conjecture that the XOR Mask problem is hard. It is important to note that this is only evidence rather than proof. It may very well be that a probabilistic or deterministic Turing Machine can indeed solve the problem by analysing the description of the Turing Machine computing the function rather than simply using it as an oracle. However, the next theorem shows that the XOR-Mask problem does have a polynomial time solution on a quantum

Turing Machine, given that the input Turing Machine computes the function f in polynomial time.

THEOREM 5.2. *There exists a Quantum Turing Machine M such that given a Turing Machine N computing a 2 to 1 function* $f : \{0,1\}^n \to \{0,1\}^{n-1}$ *such that for all* $x, y \in \{0,1\}^n$, $f(x) = f(y)$ *iff* $x = y \oplus s$ *for some mask* $s \in \{0,1\}^n$, *M finds s in time polynomial in n and t(n), where t(n) is the time it takes N to compute* $f(x)$ *on input* $x \in \{0,1\}^n$. [4]

PROOF Consider what would happen if we were to successively apply the unitary transformation

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

to each of n-bits on a tape containing the n-bit string $|\alpha_1, \alpha_2, ..., \alpha_n\rangle$. Applying it to α_1 would yield the superposition:

$$\frac{1}{\sqrt{2}}(-1)^{\alpha_1 \cdot 0} |0, \alpha_2, ..., \alpha_n\rangle + \frac{1}{\sqrt{2}}(-1)^{\alpha_1 \cdot 1} |1, \alpha_2, ..., \alpha_n\rangle$$

applying U to α_2 would then yield the superposition

$$\frac{1}{\sqrt{2}^2}(-1)^{\alpha_1 \cdot 0 + \alpha_2 \cdot 0} |0, 0, ..., \alpha_n\rangle + \frac{1}{\sqrt{2}^2}(-1)^{\alpha_1 \cdot 0 + \alpha_2 \cdot 1} |0, 1, ..., \alpha_n\rangle +$$
$$\frac{1}{\sqrt{2}^2}(-1)^{\alpha_1 \cdot 1 + \alpha_2 \cdot 0} |1, 0, ..., \alpha_n\rangle + \frac{1}{\sqrt{2}^2}(-1)^{\alpha_1 \cdot 1 + \alpha_2 \cdot 1} |1, 1, ..., \alpha_n\rangle$$

Therefore, after n iterations we will arrive at the superposition:

$$\sum_{\beta_1, \beta_2, ..., \beta_n \in \{0,1\}} \frac{1}{\sqrt{2^n}}(-1)^{\alpha_1\beta_1 + \alpha_2\beta_2 + ... \alpha_n\beta_n} |\beta_1, \beta_2, ..., \beta_n\rangle$$

or if we write $\beta = (\beta_1, \beta_2, ..., \beta_n)$ and $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ we could write this as

$$\sum_{\beta \in \{0,1\}^n} \frac{1}{\sqrt{2^n}}(-1)^{\alpha \cdot \beta} |\beta\rangle$$

where by $\alpha \cdot \beta$ we mean the dot product of the two vectors mod 2. This is the result of applying the transformation $F = \underbrace{U \otimes U \otimes .... \otimes U}_{n}$ (The Kronecker product) to all

n-bits.

Given this result, using a QTM that initially contains on its tape k copies of a string of n zeros, we perform the following algorithm k times – once on each set of n zeros. In other we begin with a TM that initially contains on its tape $\left|\underbrace{0^n, 0^n, ..., 0^n}_{n-times}\right\rangle$. and apply the following algorithm to each of the 0^n strings:

(1) Apply the transformation F above to all n 0 bits to arrive at the superposition

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

(2) Reversibly compute $|x, f(x)\rangle$ on the tape, using Theorem 4.1.

(3) Apply F to all n bits of x to arrive at the superposition

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

When we make an observation we will see some configuration $|y, f(x)\rangle$ on the tape, with probability equal to the square of the sum of the amplitudes of all those paths leading to $|y, f(x)\rangle$. However we note that there are only two such paths, namely, the path that wrote $|y, f(x)\rangle$ and the path that wrote $|y, f(x \oplus s)\rangle$. The probability is thus,

$$\begin{aligned}
\frac{1}{2^n}((-1)^{x\cdot y} + (-1)^{(x\oplus s)\cdot y}) &= \frac{1}{2^n}((-1)^{x\cdot y} + (-1)^{(x\cdot y)\oplus(s\cdot y)}) \\
&= \frac{1}{2^n}((-1)^{x\cdot y} + (-1)^{(x\cdot y)\oplus 1}) \qquad \text{if } s\cdot y \neq 0 \\
&= 0
\end{aligned}$$

This shows that the only possible outcomes for y are those such that $y \cdot s = 0$. A closer observation will reveal that y is uniformly distributed over the set $\{d|d\cdot s = 0\}$. But this means that over the vector space $\mathbb{Z}_2{}^n$ we consistently get vectors y that are certain to be orthogonal to s. Clearly, if we are trying to find s, it is sufficient for us to collect $n-1$ linearly independent such vectors, since then we would then have enough information to construct a system of equations whose solution is either s or

0. It turns out to be sufficient to collect $k = O(n)$ such vectors in order to ensure that n of them are linearly independent with high probability. To solve such a system would require us to be able to solve an $n \times n$ system of equations over the field $\mathbb{Z}_2$. This however, is easily done in time that is polynomial in n. [4] $\square$

As an immediate corollary from Theorems 5.1 and 5.2 we get the following:

THEOREM 5.3. *There exists an oracle O relative to which $BQP \not\subseteq BPP$.* [4]

CHAPTER 6
The Discrete Log

The most substantial evidence to date regarding the power of the quantum Turing Machine was offered by Shor. In a paper published last year he presents an algorithm for the Discrete Log and Integer Factorization problems, which runs in polynomial time on a quantum Turing Machine. Although there is no proof that one can't solve these problems in polynomial time on a classical Turing machine, it has been a long standing conjecture. While integer factorization is a well-known problem, the discrete-log problem is somewhat less famous. We present it here along with an explanation and its quantum solution.

6.1. Background

Since the problem is an algebraic problem, we will require a number of definitions before it can be properly introduced. To that end we bring a number of definitions from abstract algebra.

DEFINITION 6.1. *A set G with a binary operation $\circ$ is called a group if it satisfies the following four axioms:*

(1) *G is closed under the binary operation $\circ$. That is to say, if $g, h \in G$ then $g \circ h \in G$.*

(2) *The operation is associative, so $(a \circ b) \circ c = a \circ (b \circ c)$.*

(3) *G has a unity, meaning there exists an element* $e \in G$ *such that for all* $g \in G$,

$$e \circ g = g = g \circ e$$

(4) *Every element of G has an inverse in G, so that for all* $g \in G$ *there exists* $g^{-1} \in G$ *such that*

$$g^{-1} \circ g = e = g \circ g^{-1}$$

Furthermore, G will be an abelian (commutative) group if it satisfies axiom 5.

(5) *For all* $g, h \in G$, $g \circ h = h \circ g$

For convenience we shall hereinafter denote $g \circ h$ *by* gh, $g \circ g$ *will be denoted by* g^2 *and the unity e will be denoted by 1.* [11]

DEFINITION 6.2. *A generator of a group G is element* $g \in G$, *such that for all* $x \in G$,

$$g^k = x \text{ for some } k \in \mathbb{Z}$$

A group G is cyclic if it is has one generator.

DEFINITION 6.3. *The order of a group G is the number of elements in it* $|G|$, *and the order of an element* $g \in G$ *is the smallest integer k such that* $g^k = 1$. [11]

EXAMPLE 6.1. For any positive integer n, the set $U_n = \left\{e^{\frac{2\pi ik}{n}} | k \in \mathbb{Z}\right\}$ is a cyclic abelian group closed under multiplication. It's unity is $1 = e^{\frac{2\pi i0}{n}}$. It is easy to see that $e^{\frac{2\pi i}{n}}$ is a generator of the group. In fact, U_n is generated by any element $e^{\frac{2\pi ik}{n}}$ such that $gcd(k, n) = 1$. Note that U_n is the set of all n-th roots of unity, evenly spaced around the unit circle.

EXAMPLE 6.2. For any prime p, the set $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ is a cyclic abelian group closed under multiplication.

PROOF Proposition II.1.2 in *A Course in Number Theory and Cryptography* by Neal Koblitz [12] □

EXAMPLE 6.3. For any positive integer m, let $\mathbb{Z}_m^*$ denote all those elements in $\mathbb{Z}_m$ for which property 4 of definition 6.1 holds under the operation of integer multiplication. Then $\mathbb{Z}_m^*$ is as well a group. The elements in $\mathbb{Z}_m^*$ will be all those elements $r \in \mathbb{Z}_m$, such that $\gcd(r, m) = 1$.

A thorough discussion of the above concepts and definitions is beyond the scope of this paper. For a more extensive treatment of the subject the reader is referred to Nicholson's excellent book, *An Introduction to Abstract Algebra* [11].

6.2. The Fourier Transform

In order to solve the discrete log problem using Shor's method we are required to perform a particular unitary transformation using our quantum Turing Machine. This transformation is known as the discrete Fourier Transform.

DEFINITION 6.4. *Given a vector $\vec{x} = (x_0, x_1, ..., x_{n-1})$ in an n-dimensional vector space V over the field $\mathbb{C}$, the discrete Fourier Transform F_n, is defined as:*

$$F_n(\vec{x}) = \left(\frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \omega^0, \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \omega^j, \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \omega^{2j}, ..., \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \omega^{(n-1)j} \right).$$

where $\omega = e^{\frac{2\pi i}{n}}$, the n-th root of unity.

Note that the above transformation can also be represented by the following matrix:

$$F_n = \frac{1}{\sqrt{n}} \begin{bmatrix} \omega^{0\times0} & \omega^{0\times1} & \omega^{0\times2} & \dots & \omega^{0\times(n-1)} \\ \omega^{1\times0} & \omega^{1\times1} & \omega^{1\times2} & \dots & \omega^{1\times(n-1)} \\ \omega^{2\times0} & \omega^{2\times1} & \omega^{2\times2} & \dots & \omega^{2\times(n-1)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \\ \omega^{(n-1)\times0} & \omega^{(n-1)\times1} & \omega^{(n-1)\times2} & \dots & \omega^{(n-1)\times(n-1)} \end{bmatrix}. \tag{2}$$

In particular, row α, column β is the entry $(F_n)_{\alpha\beta} = \frac{1}{\sqrt{n}} e^{\frac{2\pi i \alpha\beta}{n}}$.

EXAMPLE 6.4. If V is a 3-dimensional vector space, then since the 3-rd root of unity is $e^{\frac{2\pi i}{3}} = \omega$, the Fourier Transform on V will be the matrix:

$$f_3 = \frac{1}{\sqrt{3}} \begin{bmatrix} \omega^{0\times 0} & \omega^{0\times 1} & \omega^{0\times 2} \\ \omega^{1\times 0} & \omega^{1\times 1} & \omega^{1\times 2} \\ \omega^{2\times 0} & \omega^{2\times 1} & \omega^{2\times 2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 \\ 1 & \omega^2 & \omega^1 \end{bmatrix}$$

We are interested in applying the transformation F_n to a configuration in our quantum Turing Machine. Given the discussion in Chapter 3, we are required to show that F_n is a well-formed quantum step before we can use it in a quantum Turing Machine. Towards that end we prove the following lemma

LEMMA 6.1. *For all positive integers n and $m \not\equiv 0 \pmod{n}$, the sum*

$$\sum_{k=0}^{n-1} e^{\frac{2\pi i k m}{n}} = 0$$

PROOF Let n and m be positive integers. Write $d = \gcd(m, n)$, and let $m' = \frac{m}{d}$ and $n' = \frac{n}{d}$. Then,

$$\sum_{k=0}^{n-1} e^{\frac{2\pi i k m}{n}} = \sum_{k=0}^{n-1} e^{\frac{2\pi i k m'}{n'}}.$$

We now index the sum using two indices j and r such that $k = jn' + r$ to get

$$= \sum_{j=0}^{d-1} \sum_{r=0}^{n'-1} e^{\frac{2\pi i (jn'+r) m'}{n'}}$$

$$= \sum_{j=0}^{d-1} \sum_{r=0}^{n'-1} e^{\frac{2\pi i r m'}{n'}}$$

Now if $e^{\frac{2\pi i r_1}{n'}} = e^{\frac{2\pi i r_2}{n'}}$, then $r_1 \equiv r_2 \pmod{n'}$. This means though, that if $r_1 \not\equiv r_2 \pmod{n'}$ then $e^{\frac{2\pi i r_1}{n'}} = e^{\frac{2\pi i r_2}{n'}}$. But this implies that $\sum_{r=0}^{n'-1} e^{\frac{2\pi i r m'}{n'}}$, is a sum over n' distinct n'-th roots of unity, i.e. this is the sum over all the n'-th roots of unity evenly spaced around the unit circle. This is zero for all but the first root of unity. Since $m \not\equiv 0 \pmod{n}$, we have $d \neq n$ and thus $n' \neq 1$. Therefore $\sum_{r=0}^{n'-1} e^{\frac{2\pi i r m'}{n'}} = 0$. □

The next theorem shows that F_n is a well-formed quantum step:

THEOREM 6.1. *F_n is a unitary operator.*

PROOF It suffices to show that the columns of F_n form an orthonormal set in $\mathbb{C}^n$. Let $\omega = e^{\frac{2\pi i}{n}}$, and consider rows a and b in F_n, where $a \leq b$ without loss of generality. We take the inner product of the two rows. Remember that since the inner product is in $\mathbb{C}^n$, it is defined as the dot product of the entries in row a with the complex conjugates of the entries in row b (The complex conjugate is denoted by a bar). So,

$$\begin{aligned} < Row_a, Row_b > &= \sum_{k=0}^{n-1} (F_n)_{ak} \overline{(F_n)_{bk}} \\ &= \sum_{k=0}^{n-1} \frac{1}{\sqrt{n}} \frac{1}{\sqrt{n}} \omega^{ak} \overline{\omega^{bk}} \\ &= \frac{1}{n} \sum_{k=0}^{n-1} e^{\frac{2\pi i a k}{n}} \overline{e^{\frac{2\pi i b k}{n}}} \end{aligned}$$

It is easily verified that $\overline{e^{\frac{2\pi i b k}{n}}} = e^{\frac{2\pi i (n-bk)}{n}}$. Therefore the above reduces to

$$= \frac{1}{n} \sum_{k=0}^{n-1} e^{\frac{2\pi i a k}{n}} e^{\frac{2\pi i (n-bk)}{n}} \tag{3}$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} e^{\frac{2\pi i (n+(a-b)k)}{n}} \tag{4}$$

$$= \frac{1}{n} \sum_{k=0}^{n-1} e^{\frac{2\pi i (a-b)k}{n}} \tag{5}$$

Now, if $a = b$ then the above is just the length of row a, which is

$$\begin{aligned} &= \frac{1}{n} \sum_{k=0}^{n-1} e^{\frac{2\pi i (a-a)k}{n}} \\ &= \frac{1}{n} \sum_{k=0}^{n-1} e^{\frac{2\pi i 0}{n}} \\ &= \frac{1}{n} \sum_{k=0}^{n-1} 1 \\ &= 1. \end{aligned}$$

So each row is indeed a normal vector. If $a \neq b$ then write $t = a - b$, and since $a \leq b$,

t is positive. Applying Lemma 6.1, we have 3 reducing to is:

$$\begin{aligned} &= \frac{1}{n}\sum_{k=0}^{n-1} e^{\frac{2\pi itk}{n}} \\ &= 0. \end{aligned}$$

Therefore the rows of F_n form an orthonormal set, whence F_n is unitary. □

Hence, F_n is a unitary matrix, and we can use it in a quantum computation. This being our goal, it is useful to describe the action of F_n on a particular state vector of configurations. Translating definition 6.4 to our notation of superpositions, we find that the action of F_n on a configuration $|x\rangle$ where x is an element of $\mathbb{Z}_n$ (encoded in binary representation) results in the superposition

$$\frac{1}{\sqrt{n}}\sum_{y\in\mathbb{Z}_n} e^{\frac{2\pi iyx}{n}}|y\rangle.$$

Note that this last sum is column x of F_n in its matrix representation from 2. Furthermore, an observation of this superposition would yield $y \in \mathbb{Z}_n$ with probability $\frac{1}{n}$, and as a result we can view the Fourier Transform as a form of randomly and uniformly choosing an element in $\mathbb{Z}_n$. (Although, this is a very simplistic approach.)

6.3. The Problem

Before presenting the problem, and in order to facilitate an understanding of the Discrete Log problem we give the more popular continuous logarithm problem. Note that the two are similar in many respects, and in fact Discrete Log is the the continuous log problem in a finite, cyclic, abelian group.

PROBLEM 6.1. *Continuous Log. Given two numbers $g, x \in \mathbb{R}$, we would like to find a number $r \in \mathbb{R}$ such that $g^r = x$. Note that the number $r = \log_g(x)$.*

Given the above definitions we are now ready to present the problem. Contrast and compare it to problem 6.1.

PROBLEM 6.2. *Discrete Log. Given three numbers, a prime p, an element* $g \in \mathbb{Z}_p^*$ *such that g is a generator of* $\mathbb{Z}_p^*$*, and* $x \in \mathbb{Z}_p^*$*, we would like to find a number* $r \in \mathbb{Z}_{p-1}$ *such that* $g^r = x$.

EXAMPLE 6.5. Given $p = 5$, $g = 2$ and $x = 1$, the correct solution to problem 6.2 would be 0 since $2^0 = 1$. Note that 2 generates $\mathbb{Z}_5^*$ since $2^0 = 1 \pmod 5$), $2^1 = 2 \pmod 5$, $2^2 = 4 \pmod 5$ and $2^3 = 3 \pmod 5$.

EXAMPLE 6.6. Given $p = 7$, $g = 3$ and $x = 5$, the correct solution to problem 6.2 would be 5 since $3^5 = 243 = 5 \pmod 5$.

We begin by treating an easy case of problem 6.2. In this simplification of the problem we shall restrict the groups that we are analysing to those groups $\mathbb{Z}_p^*$ such that $p-1$ has a particular property. In order to facilitate this property's definition, we fix some positive real number c which will remain constant throughout this discussion. Then we define the following:

DEFINITION 6.5. *A number* $n = p_1 p_2 p_3 \dots p_k$ *is smooth if all the* p_i*'s are distinct primes such that for all* $i \in \{1, 2 \dots k\}$, $p_i < clog(n)$. *That is to say, n has no prime factor that is larger than a logarithm in n.*

6.4. A Solution of Discrete Log in a Restricted Setting

We first solve the Discrete Log problem over all groups $\mathbb{Z}_p^*$ such that $p-1$ is smooth. This is the easy case of the algorithm that Shor presents in his paper.

THEOREM 6.2. *There exists a quantum Turing Machine M and a polynomial P, such that given an n bit prime p where* $p - 1$ *is smooth, a generator* $g \in G$ *and a number x, M returns a number* $r \in \mathbb{Z}_{p-1}$ *such that* $g^r = x$, *in time* $P(n)$ *with error at most 1/3.* [9]

PROOF Initially the Turing Machine has p, g and x written on the tape, and is in some state q. Since p, g, and x all remain constant throughout the computation and do not change, we will not write them when writing a configuration. Furthermore,

since the machine will never be found in a superposition of states we will not write the state of the machine in the configuration. In other words, the machine moves from one state to the next in a deterministic fashion, never making a probabilistic choice between two states. Those superpositions that are formed are made by a choice of symbols to write on the tape, rather than a choice of states to enter. We will therefore mention that portion of the tape, whose contents change. Given this, the tape will operate on a tape of all zero's once the input p, g and x has been read.

Initially we let the machine choose two random values a and b in $\mathbb{Z}_{p-1}$. These two values can be chosen in a classical probabilistic way, and thus by the application of Theorem 4.1 can be chosen by M. If these values are selected in a way which is random and uniform, each one appears with probability $\frac{1}{p-1}$ and hence with amplitude $\frac{1}{\sqrt{p-1}}$. After choosing the first number a, our machine will be in superposition

$$\frac{1}{\sqrt{p-1}} \sum_{a=0}^{p-2} |a\rangle.$$

Note that the sum is over 0 to $p-2$ since $p-2$ is the largest element of $\mathbb{Z}_{p-1}$. Choosing b randomly, then places us in the superposition:

$$\begin{aligned} & \frac{1}{\sqrt{p-1}} \frac{1}{\sqrt{p-1}} \sum_{b=0}^{p-2} \sum_{a=0}^{p-2} |a, b\rangle. \\ = \; & \frac{1}{p-1} \sum_{b=0}^{p-2} \sum_{a=0}^{p-2} |a, b\rangle. \end{aligned}$$

We now deterministically compute $g^a x^{-b}$ on the tape. This computes a particular element in $\mathbb{Z}_p^*$. Since a and b are not erased from the tape, we can do this reversibly and thus such a computation is feasible using M. Once we have completed this, our machine should be in the superposition:

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

Now, since a is in $\mathbb{Z}_{p-1}$we can apply the Fourier Transform to it to get the super-

position:

$$\frac{1}{p-1}\sum_{c=0}^{p-2}\sum_{b=0}^{p-2}\sum_{a=0}^{p-2}\frac{1}{\sqrt{p-1}}e^{\frac{2\pi ica}{p-1}}|c,b,g^a x^{-b}\rangle.$$

Likewise, since b is in $\mathbb{Z}_{p-1}$we apply the Fourier Transform to it and get:

$$\begin{aligned}&\frac{1}{p-1}\sum_{d=0}^{p-2}\sum_{c=0}^{p-2}\sum_{b=0}^{p-2}\sum_{a=0}^{p-2}\frac{1}{\sqrt{p-1}}e^{\frac{2\pi ica}{p-1}}\frac{1}{\sqrt{p-1}}e^{\frac{2\pi idb}{p-1}}|c,d,g^a x^{-b}\rangle\\ =\ &\frac{1}{(p-1)^2}\sum_{d=0}^{p-2}\sum_{c=0}^{p-2}\sum_{b=0}^{p-2}\sum_{a=0}^{p-2}e^{\frac{2\pi i(ca+db)}{p-1}}|c,d,g^a x^{-b}\rangle.\end{aligned}$$

Since g is a generator of $\mathbb{Z}_p^*$ there exists a solution r such that $x = g^r$. Therefore, the above superposition can be written as

$$\begin{aligned}&\frac{1}{(p-1)^2}\sum_{d=0}^{p-2}\sum_{c=0}^{p-2}\sum_{b=0}^{p-2}\sum_{a=0}^{p-2}e^{\frac{2\pi i(ca+db)}{p-1}}|c,d,g^a g^{-rb}\rangle\\ =\ &\frac{1}{(p-1)^2}\sum_{d=0}^{p-2}\sum_{c=0}^{p-2}\sum_{b=0}^{p-2}\sum_{a=0}^{p-2}e^{\frac{2\pi i(ca+db)}{p-1}}|c,d,g^{a-rb}\rangle.\end{aligned}$$

When we observe the machine we will see some configuration $|c,d,g^k\rangle$ for some $k \in \mathbb{Z}_{p-1}$. Note that k will depend on a,b and r, and in fact $k \equiv a - rb \pmod{p-1}$. Now consider the probability with which we see such a state. This is the square of the absolute value of the amplitude of being in configuration $|c,d,g^k\rangle$. The amplitude of observing this particular state is the sum of the amplitudes of each path that could lead us to such a configuration. Those paths arise from all possible values of a and b such that $k \equiv a - rb \pmod{p-1}$. Therefore the probability of arriving at this state is:

$$\left|\frac{1}{(p-1)^2}\sum_{\substack{a,b\in\mathbb{Z}_{p-1}\\ a-rb=k \pmod{p-1}}}e^{\frac{2\pi i(ac+bd)}{p-1}}\right|^2$$

Now substituting for a in the amplitude we get:

$$\begin{aligned}
&\left|\frac{1}{(p-1)^2}\sum_{b=0}^{p-2} e^{\frac{2\pi i(k+rb)c+bd}{p-1}}\right|^2 \\
= &\left|\frac{1}{(p-1)^2}\sum_{b=0}^{p-2} e^{\frac{2\pi i(kc+b(d+rc))}{p-1}}\right|^2 \\
= &\left|\frac{e^{\frac{2\pi ikc}{p-1}}}{(p-1)^2}\sum_{b=0}^{p-2} e^{\frac{2\pi ib(d+rc)}{p-1}}\right|^2 \\
= &\left|e^{\frac{2\pi ikc}{p-1}}\right|^2\left|\frac{1}{(p-1)^2}\sum_{b=0}^{p-2} e^{\frac{2\pi ib(d+rc)}{p-1}}\right|^2
\end{aligned}$$

Since $e^{\frac{2\pi ikc}{p-1}}$ is just a $p-1$-th root of unity, it is situated on the unit circle and its absolute value is therefore 1. Thus the probability reduces to:

$$\left|\frac{1}{(p-1)^2}\sum_{b=0}^{p-2} e^{\frac{2\pi ib(d+rc)}{p-1}}\right|^2$$

Now, suppose that $d+rc \neq 0 \pmod{p-1}$, then applying Lemma 6.1 we have the probability being 0. Therefore, the pairs of d and c that we do get must have the property that $d = -rc \pmod{p-1}$. Let $c' = (-c) \pmod{p-1}$. Then if c' and $p-1$ are relatively prime then $\gcd(c', p-1) = 1$, and we can use the extended Euclidean algorithm to find two integers s and v such that $1 = sc' + v(p-1)$. But then $1 = sc' \pmod{p-1}$ which means that s is the inverse of c' in ${\mathbb{Z}_{p-1}}^*$. So we can find r simply by computing ds. It remains to show then that indeed c' and $p-1$ are relatively prime with high probability. Since c is essentially chosen randomly and uniformly from $\mathbb{Z}_{p-1}$, so is c' random and uniform. The probability that c' is relatively prime

to $p-1$ is therefore:

$$\begin{aligned}\frac{|\{k \in \mathbb{Z}_{p-1} | \gcd(k, p-1) = 1\}|}{p-1} &\geq \frac{|\{k \in \mathbb{Z}_{p-1} | k \text{ is prime and } k \not| p-1\}|}{p-1} \\ &\geq \frac{|\{k \in \mathbb{Z}_{p-1} | k \text{ is prime}\}| - |\{k \in \mathbb{Z}_{p-1} | k | p-1\}|}{p-1}\end{aligned}$$

(The notation $k \not| p-1$ is used to denote k does not divide $p-1$.) The prime number theorem states that the number of primes not exceeding a number x is in $\Theta(\frac{x}{\log(x)})$, and $|\{k \in \mathbb{Z}_{p-1} | k | p-1\}| \leq \log(p-1)$ is trivial. So there exists some constant ρ such that:

$$\begin{aligned}\frac{|\{k \in \mathbb{Z}_{p-1} | \gcd(k, p-1) = 1\}|}{p-1} &\geq \frac{\frac{\rho(p-1)}{\log(p-1)} - \log(p-1)}{p-1} \\ &\geq \frac{\rho}{\log(p-1)} - \frac{\log(p-1)}{p-1} \\ &\geq \frac{\gamma}{\log(p-1)}\end{aligned}$$

for some constant γ. If we perform this algorithm $t = \frac{\log(p-1)\ln(3)}{\gamma}$ times, we will get a collection of independently chosen $c'_1, c'_2, c'_3, ..., c'_t$. Now, the probability that this algorithm fails to find one c_i which is relatively prime to $p-1$ is

$$\begin{aligned}\Pr[\text{for all } i \in \{1,2,..,t\}, \gcd(c'_i, p-1) \neq 1] &= \prod_{i=1}^{t} \Pr[\gcd(c'_i, p-1) \neq 1] \\ &= \prod_{i=1}^{t} (1 - \Pr[\gcd(c'_i, p-1) = 1] \\ &\leq \prod_{i=1}^{t} \left(1 - \frac{\gamma}{\log(p-1)}\right) \\ &= \left(1 - \frac{\gamma}{\log(p-1)}\right)^t \\ &\leq \left(e^{-\frac{\gamma}{\log(p-1)}}\right)^t \\ &\leq \frac{1}{3}.\end{aligned}$$

We have thus exhibited a quantum computation which is correct within an error

bound of $\frac{1}{3}$. It remains to show that this computation is feasible in polynomial time. A quick observation shows that with the exception of the computation of F_n on a and b above, all other steps can be performed in polynomial time. The discussion in the next section and its culmination in Lemma 6.3 completes the proof. [9] □

6.5. Efficient Computation of the Fourier Transform

In order to justify Shor's algorithm in Theorem 6.2 we must show that the Fourier Transform can indeed be computed in time that is polynomial in the length of the binary representation of the input. Since the output of the Fourier Transform on a given configuration $|x\rangle$, where $x \in \mathbb{Z}_n$, is the superposition

$$\frac{1}{\sqrt{n}} \sum_{y \in \mathbb{Z}_n} e^{\frac{2\pi i y x}{n}} |y\rangle.$$

it is not at all clear that F_n is an operator computable in polynomial time. In fact, since the binary representation of $x \in \mathbb{Z}_n$ has length $b = \lceil log(n) \rceil$, the number of possible resulting configurations is n, which is greater than 2^b (an exponential in the length of the representation.) Furthermore, it is not known how to compute F_n in general for any n in polynomial time. However, if n is a smooth number, there is an elegant construction which allows us to compute F_n. In fact, there are a number of other restrictions that one can place on the Fourier Transform problem to ensure that it can be computed in polynomial time. For example, Coppersmith [13] shows how to compute F_n in polynomial time for cases where n is a power of 2.

In order to show that this construction is polynomial in nature we will require the following lemma:

LEMMA 6.2. *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a function such that both f and $f^{-1} : \{0,1\}^m \to \{0,1\}^n$ can be computed in polynomial time by a classical deterministic Turing Machine. Then there exists a quantum Turing machine M which given an input $|x\rangle$ where $x \in \{0,1\}^n$ computes $|f(x)\rangle$ on the tape and erases the input x.*

PROOF Since f is polynomial time computable, we appeal to Bennett's result (Theorem 4.2) and compute $|x, f(x)\rangle$ on the tape. We then appeal to Bennett's result again and compute f^{-1} on $f(x)$ to get $|f^{-1}(f(x)) \oplus x, f(x)\rangle$. (Note that Theorem 4.2 is applied in reverse order of input and output.) However,

$$|f^{-1}(f(x)) \oplus x, f(x)\rangle = |x \oplus x, f(x)\rangle = |x \oplus x, f(x)\rangle = |f(x)\rangle \quad \square$$

The next theorem completes the justification of Theorem 6.2 and provides us with a method for computing the Fourier Transform on vector spaces of dimension n where n is a smooth number. The proof of the theorem was proposed by Wayne Eberly. [14]

THEOREM 6.3. *There exists a quantum Turing Machine M and a polynomial P, such that given a smooth number n, M computes the n-bit Fourier Transform F_n on an input $|x\rangle$ where $x \in \mathbb{Z}_n$ in time polynomial in* $\log(n)$. [9]

PROOF Let n be some smooth positive number. If n is a smooth number then its prime factorization $p_1 p_2 p_3 .. p_k$ can be computed in polynomial time, since a simple trial and error which attempts to divide n by all numbers up to $clog(n)$ would reveal all the primes. (c is the constant which we fixed above.) Since each $p_i \geq 2$ for all i, we have that $k \leq log(n)$. Such a computation is clearly polynomial on a deterministic Turing Machine, and since we are not restricted to erasing the input n, it can be done in polynomial time on a quantum Turing Machine. The algorithm will do the following:

(1) Given that $n = p_1 p_2 p_3 \cdots p_k$, the algorithm divides n into $q = p_1$ and $r = p_2 \cdots p_k$.

(2) Since n is smooth, $\gcd(q, r) = 1$, and thus we can use the Extended Euclidean Algorithm to find two elements $s, t \in \mathbb{Z}$ such that $sq + tr = 1$.

(3) Given $|x\rangle$ reversibly compute $|v, w\rangle$ where $v = (tx \mod q)$ and $w = (sx \mod r)$.

(4) Since $v \in \mathbb{Z}_q$, and q is guaranteed to be logarithmic in n, we can compute F_q on v in polynomial time.

(5) Since $w \in \mathbb{Z}_r$, we can compute F_r recursively on w using our algorithm, unless of course r = 1, at which point we stop.

(6) The computation of steps 4 and 5 yields some configuration $|v', w'\rangle$ where $v' \in \mathbb{Z}_q$ and $w' \in \mathbb{Z}_r$. Given these as inputs, we reversibly compute a $y \in \mathbb{Z}_n$ such that $y \equiv v' \pmod{q}$ and $y \equiv w' \pmod{r}$.

In order to show that this algorithm does indeed compute F_n in polynomial time, we must show that all six steps of the algorithm can be computed in polynomial time, and that the algorithm is indeed correct. We begin by showing that each step is a polynomial time well-formed quantum computation. We have already argued that step one is well-formed and polynomial time computable when n is smooth. The Extended Euclidean Algorithm is a polynomial time algorithm and it does not require us to erase the inputs. Appealing to Bennett's result, step 2 is a polynomial time well-formed quantum step. In order to show that the same is true of step 3, we prove two facts. Firstly, we show that the function $f : \mathbb{Z}_n \to \mathbb{Z}_q \times \mathbb{Z}_r$, $f(x) = (tx \pmod{q}, sx \pmod{r})$ where $sq + tr = 1$, is a polynomial time computable function on a deterministic Turing Machine, and secondly we show that $f^{-1} : \mathbb{Z}_q \times \mathbb{Z}_r \to \mathbb{Z}_n$ is computable in polynomial time on a deterministic Turing machine. We then apply Lemma 4.2.

It is obvious that f can be computed in polynomial time on a deterministic Turing Machine, since the entire computation involves two multiplications and two divisions. The inverse is a bit trickier. We are given $(v, w) \in \mathbb{Z}_q \times \mathbb{Z}_r$ and we are interested in getting the element $x \in \mathbb{Z}_n$ such that $v \equiv tx \pmod{q}$ and $w \equiv sx \pmod{r}$. Since $sq + tr = 1$, we have that $tr \equiv 1 \pmod{q}$ and $sq \equiv 1 \pmod{r}$. Therefore in $\mathbb{Z}_q$, $t^{-1} = r$, and in $\mathbb{Z}_r$, $s^{-1} = q$. Thus if we compute $rv = r(tx) = (t^{-1}t)x = x \pmod{q}$ and $qw = q(sx) = (s^{-1}s)x = x \pmod{r}$, we will have $(x \mod r)$ and $(x \mod q)$. With the use of Chinese Remaindering (Theorem 3.28 in *Abstract Algebra* [11]), we

can now find x in $\mathbb{Z}_n$. So f^{-1} is as well computable in polynomial time, and hence, step 3 is a well-formed polynomial time quantum computation. A similar but easier argument applies to step 6 in the computation.

Step 4 can be done trivially in polynomial time, and step 5 can be computed in polynomial time by induction. It therefore remains to show that this computation is indeed the computation of F_n.

Starting with input $|x\rangle$, we compute $|v, w\rangle$ where $v = (tx \mod q)$ and $w = (sx \mod r)$. We then compute F_q on v and arrive at the superposition:

$$\sum_{v'=0}^{q-1} \frac{1}{\sqrt{q}} e^{\frac{2\pi i v v'}{q}} |v', w\rangle$$

In step 5 we compute F_r on w, to get the superposition:

$$\begin{aligned} & \sum_{v'=0}^{q-1} \sum_{w'=0}^{r-1} \frac{1}{\sqrt{q}} \frac{1}{\sqrt{r}} e^{\frac{2\pi i v v'}{q}} e^{\frac{2\pi i w w'}{r}} |v', w'\rangle \\ = & \frac{1}{\sqrt{qr}} \sum_{v'=0}^{q-1} \sum_{w'=0}^{r-1} e^{\frac{2\pi i (rvv' + qww')}{qr}} |v', w'\rangle \end{aligned}$$

Now, back substituting $n = qr$,$v = (tx \pmod q)$ and $w = (sx \mod r)$ we get:

$$\frac{1}{\sqrt{n}} \sum_{v'=0}^{q-1} \sum_{w'=0}^{r-1} e^{\frac{2\pi i (r(tx \pmod q)v' + q(sx \pmod r)w')}{n}} |v', w'\rangle$$

In order to simplify the notation we let $\omega = e^{\frac{2\pi i}{n}}$ and so the above is:

$$\frac{1}{\sqrt{n}} \sum_{v'=0}^{q-1} \sum_{w'=0}^{r-1} \omega^{(r(tx \pmod q)v' + q(sx \pmod r)w')} |v', w'\rangle$$

As the last step we compute y such that $y = v' \pmod q$, and $y = w' \pmod r$ to arrive at the superposition:

$$\frac{1}{\sqrt{n}} \sum_{y \in \mathbb{Z}_n} \omega^{(r(tx \pmod q)(y \mod q) + q(sx \pmod r)(y \mod r)} |y\rangle$$

We now want to show that,

$$[r(tx \pmod q)(y \mod q)] \mod q = [rtxy] \mod q.$$

But, we know that for some integers $\alpha_1, \alpha_2 \in \mathbb{Z}$, $y = (z_1 + \alpha_1 q)$ and $tx = (z_2 + \alpha_2 q)$. So,

$$\begin{aligned} rtxy &= r(z_2 + \alpha_2 q)(z_1 + \alpha_1 q) \\ &= rz_2 z_1 + rq(z_2\alpha_1 + z_1\alpha_2 + \alpha_2\alpha_1 q) \\ &= rz_2 z_1 + n(z_2\alpha_1 + z_1\alpha_2 + \alpha_2\alpha_1 q) \end{aligned}$$

Taking $(\text{mod } n)$ on both sides we get:

$$rtxy \equiv rz_2 z_1 \ (\text{mod } n)$$

Observe that $z_2 = (y \mod n)$ and $z_1 = (tx \mod n)$. So indeed

$$[r(tx \pmod q)(y \mod q)] \equiv [rtxy] \pmod q.$$

In the same way we show that

$$[q(sx \pmod r)(y \mod r)] \equiv [qsxy] \pmod q.$$

This means that the above superposition can be written as:

$$\begin{aligned} & \frac{1}{\sqrt{n}} \sum_{y \in \mathbb{Z}_n} \omega^{(rtxy + qsxy)} |y\rangle \\ = & \frac{1}{\sqrt{n}} \sum_{y \in \mathbb{Z}_n} \omega^{xy(rt+qs)} |y\rangle \\ = & \frac{1}{\sqrt{n}} \sum_{y \in \mathbb{Z}_n} \omega^{xy} |y\rangle. \end{aligned}$$

Note that this is precisely the action of the Fourier Transform on input $|x\rangle$, and we can therefore conclude that the algorithm is correct and will compute F_n on $|x\rangle$.[14] □

6.6. The General Solution

Prior to exhibiting the algorithm for the general, unrestricted case of the Discrete Log problem as it is presented in problem 6.2, we require a number of intermediate results which are algebraic and number theoretic in nature. Their proofs are some-

what cumbrous and the reader is therefore advised that it may behoove their general understanding to skip the proofs of Lemmas 6.3, 6.4 and 6.5 on an initial reading. Nevertheless, for the benefit of interested readers the proofs are presented in their logical order.

6.6.1. Producing smooth numbers. The general solution to the Discrete-Log problem is analogous to the solution of the smooth case. The main problem is that we are unable to compute the Fourier Transform F_n in polynomial time as we did in the previous section. It is unknown how to compute F_n in polynomial time for a general number n, and we are therefore required to find a way to circumvent this problem. In Theorem 6.2 we restricted the solution of the problem to those groups $\mathbb{Z}_p^*$ where $p-1$ was a smooth number. This is a solution for such groups where the order of the group is a smooth number itself. We will show that it is suffices to find a smooth number that is close to the order of the group in order to find a solution that is satisfactory within an error bound of $\frac{1}{3}$.

Thus, we will require our choice for a smooth number q to be within a factor of 2 away from p, i.e. q will be smooth such that $p < q \leq 2p$. Before we use this, we must first show that there exists such a q and that it can be found in polynomial time.

LEMMA 6.3. *There exists a quantum Turing machine M and a polynomial P, such that given a number p, M finds a smooth number q such that $p < q \leq 2p$ in time $P(|p|)$ where p is encoded in binary.* [9]

PROOF Since we are not required to erase the input p from the tape it is sufficient for us to exhibit a classical deterministic algorithm and apply Bennett's result. The algorithm is quite simple.

(1) $\hat{q} \leftarrow 1$

(2) $k \leftarrow 2$

(3) while $(\hat{q} \leq p)$ do

(4) if (k is prime) then

(5) $q \leftarrow q \times k$

(6) $k \leftarrow k + 1$

(7) while($\hat{q} > 2p$)

(8) if (k is a prime) then

(9) if ($\frac{\hat{q}}{k}$) then

(10) $\hat{q} \leftarrow \frac{\hat{q}}{k}$

(11) $k \leftarrow k - 1$

(12) $q \leftarrow \hat{q}$

This algorithm multiplies the primes $\hat{q} = 2 \cdot 3 \cdot 5 \cdot 7 \cdot p_r$, until the product is larger than p. If the product is larger than $2p$ it divides by the largest number that keeps the product larger than p. So it is guaranteed that $p < q \leq 2p$. We must argue, however, that this algorithm is indeed polynomial. There are known randomized algorithms to perform the primality testing in lines 4 and 8 in polynomial time. [12, 15] So it remains to show that k never exceeds a polynomial in $\log(p)$. Furthermore, we must show that the resulting q is indeed smooth, and so p_r can not exceed $c\log(p)$ for some fixed constant c. The final value of k, however, is the last prime that is multiplied say p_r. So $k \leq p_r$, and both facts are shown if we prove that p_r is less than $clog(p)$. The Prime Number Theorem states that the number of primes less then or equal to x is in $\Theta(\frac{x}{\ln(x)})$. Let p_m be the m-th prime number, and use the fact that the number of primes less than p_m is exactly m. So for some constants α and β,

$$\frac{\alpha p_m}{\ln(p_m)} \leq m \leq \frac{\beta p_m}{\ln(p_m)}$$

Therefore, for all $m \geq 1$,

$$\frac{1}{\beta} m \ln(p_m) \leq p_m \leq \frac{1}{\alpha} m \ln(p_m).$$

Applying this inequality again to p_m we get,

$$\frac{1}{\beta} m \ln(\frac{1}{\beta} m \ln(p_m)) \leq p_m \leq \frac{1}{\alpha} m \ln(\frac{1}{\alpha} m \ln(p_m)).$$

Which implies that,

$$\frac{1}{\beta}m[\ln(\frac{1}{\beta}m) + \ln(\ln(p_m))] \leq p_m \leq \frac{1}{\alpha}m[\ln(\frac{1}{\alpha}m) + \ln(\ln(p_m))]$$

But its clear that $\ln(p_m) \leq m$. (It is an easy induction given Theorem 418 in *An Introduction to the Theory of Numbers* [16] which states that for all positive integers z there exists a prime between z and $2z$.) Therefore, it's clear from the above that there exist constants α' and β' such that:

$$\alpha' m \ln(m) \leq p_m \leq \beta' m \ln(m)$$

The number q is the product of the first r primes with the exception of 1, and since we are interested in a lower bound for this number, we may assume that it is the product of the first $r-1$ primes. Moreover, this product is guaranteed to be less than $2p$. Thus,

$$2p \geq \prod_{i=1}^{r-1} p_i$$

Using the inequality in 6, we have,

$$\begin{aligned} &\geq \prod_{i=1}^{r-1} \alpha' i \ln(i) \\ &\geq (r-1)! \\ &\geq \left(\frac{(r-1)}{e}\right)^{(r-1)} \end{aligned}$$

The last step is an immediate result of Stirling's approximation of $r!$. [17].

Taking the natural logarithm of both sides we arrive at

$$\ln(2p) \geq (r-1)\ln(r-1) - (r-1)$$

For r sufficiently large, there exists a constant δ such that $r\ln(r) \leq \delta(r-1)\ln(r-1)$. Therefore,

$$r\ln(r) \leq \delta(\ln(2p) + (r-1))$$

Through the use of equation 6 we deduce that,

$$p_r \leq \beta' r \ln(r)$$

Combining the last two facts two we get,

$$p_r \leq \beta' \delta(\ln(2p) + r - 1)$$

Since $r \leq \ln(2p)$ trivially, we conclude that

$$p_r \leq 2\beta' \delta(\ln(2p))$$

which shows that p_r is indeed in $O(\log(p))$.[9] $\square$

6.6.2. A Solution to a set of Equations. Unlike the solution for the smooth case of the Discrete Log problem, where we are guaranteed to ultimately arrive at a congruence of the form $ar \equiv b(\mod p-1)$, where r is the discrete logarithm, the general case does not have this property. We will therefore require the next two lemmas. The proof of the first was proposed by Wayne Eberly [14]

LEMMA 6.4. *There exists a quantum Turing Machine M and a polynomial P, such that given a number $n = p_1^{k_1} p_2^{k_2} p_3^{k_3} \cdots p_l^{k_l}$, and the t simultaneous equations:*

$$\begin{aligned} a_1 r &\equiv d_1 \pmod{n} \\ a_2 r &\equiv d_2 \pmod{n} \\ &\vdots \\ a_t r &\equiv d_t \pmod{n} \end{aligned}$$

with the promise that a solution r exists, and that for every p_i, there exists a j such that $p_i \not| a_j$ (p_i does not divide a_j,) M returns r in time $P(u)$ where u is the length of n and the congruences in binary representation. Note that n is given to the machine without its prime factorization.

PROOF Since we are not required to erase the inputs, we once again appeal to Bennett's result (Theorem 4.2) and thus exhibit a polynomial time deterministic algorithm solving the problem.

Given the above equations, we begin by computing $g_i = \gcd(a_i, n)$ for all $i \in \{1, 2, ..., t\}$, and thus $g_i | a_i$ and $g_i | n$. Since $a_i r \equiv d \pmod{n}$, we have that $g_i | d_i$ (for otherwise there is no solution.) But now we can divide the equation through by g_i to get a new equation:

$$\left(\frac{a_i}{g_i}\right) r \equiv \left(\frac{d_i}{g_i}\right) \pmod{\frac{n}{g_i}}$$

Let $r_i = r \mod \frac{n}{g_i}$. So r_i is a solution to the new equation. Since $\gcd(\frac{a_i}{g_i}, \frac{n}{g_i}) = 1$, it is the unique solution, and furthermore, it is easily found.

So we now have a new set of congruences:

$$\begin{aligned} r &\equiv r_1 \pmod{\frac{n}{g_1}} \\ r &\equiv r_2 \pmod{\frac{n}{g_2}} \\ &\vdots \\ r &\equiv r_t \pmod{\frac{n}{g_t}} \end{aligned}$$

Let $m_i = \frac{n}{g_i}$, and note that for any positive integer h, if $h | m_i$, then

$$r \equiv r_i \pmod{\frac{m_i}{h}}.$$

We desire a set of equations with relatively prime moduli say $m'_1, m'_2, ..., m'_t$. If we can find equations for r in relatively prime moduli $m'_1, m'_2, ..., m'_s$ then, Chinese remaindering and the fact that $\mathbb{Z}_{m'_1 m'_2 \cdots m'_s} \cong \mathbb{Z}_{m'_1} \times \mathbb{Z}_{m'_2} \times \cdots \mathbb{Z}_{m'_s}$, (Theorem 3.28 in *Abstract Algebra* [11]) will provide a solution $r \in \mathbb{Z}_{m'_1 m'_2 \cdots m'_s}$. It can easily be shown that $n | \text{lcm}(m_1, m_2, ..., m_t)$. Thus, if we can further construct $m'_1, m'_2, ..., m'_s$ such that that $\text{lcm}(m'_1, m'_2, ..., m'_s) = \text{lcm}(m_1, m_2, ..., m_t)$, we will easily be able to find a solution for r in $\mathbb{Z}_n$.

We derive the new set of moduli by initially letting $m'_1 = m_1$. We then process each m_i, $i \in \{2,..,t\}$ in turn, each time constructing a new set of relatively prime moduli m'_j's, $j \in \{2,..,s_i\}$ where the s_i's are simply an increasing sequence whose values will be determined by the algorithm as it progresses. The final number in this sequence s_t will be equal to s. We will also require that after processing m_i the new set of m'_j's, $j \in \{2,..,s_i\}$ will obey $\text{lcm}(m_1, m_2, ..., m_i) = \text{lcm}(m'_1, m'_2, ..., m'_{s_i})$. The construction of this new set will be based on the old set of moduli $m'_1, m'_2, ..., m'_{s_{i-1}}$ and on m_i. In particular, we would like to extract all the new information which m_i possesses, i.e., those factors which are not found in any of $m'_1, m'_2, ..., m'_{s_{i-1}}$. To that end, for each $j \in \{2,..,s_i\}$ we let $u_j = \frac{m'_j}{\gcd(m'_j, m_i)}$ and we let $v_j = \frac{m_i}{gcd(m'_j, m_i)}$. A careful observation will reveal that u_j consists of only and all those primes whose power in m_i is strictly smaller than their power in m'_j. Likewise, v_j has as factors only and all those primes whose power in m_i is strictly larger than their power in m'_j. Let $w = \log(\max(m_1, m_2, ..., m_t))$, where w represents some polynomial upper bound on the largest power of any prime factor in any of the moduli $m_1, m_2, ..., m_t$. Set $\alpha = \gcd(u_j^w, m_j)$, $\beta = \gcd(v_j^w, m_i)$, $\gamma = \gcd(u_j^w, m_i)$ and $\delta = \frac{m_i}{\beta\gamma}$. Thus α is composed of all those primes whose power in m'_j is strictly larger than their power in m_i, and, moreover, the power of those primes in α is precisely equal to the power of those primes in m'_j. γ has the same prime divisors as α except the power of those primes in γ is equal to their power in m_i. Likewise, β is composed of all those primes whose power in m'_j is strictly smaller then their power in m_i, and the power of those primes is equal to their power in m_i. Thus, δ is composed of exactly those primes that appear with equal powers in both m_i and m'_j. Note that α and δ are relatively prime and hence we replace the modulus m_j by the two moduli α and δ. This process seperates the new prime powers introduced by m_i from prime powers which have already been incorporated. The only new information brought about by m_i is now contained in β and thus we replace the modulus m_i with β. It is quite evident that this procedure will yield a new set of relatively prime moduli whose least common multiple is the

same as $\text{lcm}(m_1, m_2, ..., m_i)$. This completes the proof.

LEMMA 6.5. *There exists a quantum Turing Machine M and a polynomial P, such that given a number $n = p^k$, (p prime) and the congruence:*

$$ar \equiv d \pmod{n}$$

with the promise that a solution r exists, M returns r in time $P(u)$ where u is the length of n and the congruence in binary representation. In this case n is given to the machine with its prime factorization.

PROOF Once again we don't need to erase the input, so we simply exhibit a deterministic algorithm for the problem. Now, if $\gcd(a, n) = 1$ we simply find a^{-1} and we are done. Otherwise, we can assume that $p^l = \gcd(a, n)$, where $l \leq k$, since p is prime. Then, $p^l|a$,$p^l|p^k$ and $p^l|d$ (otherwise the equation has no solution). This implies that we can divide the equation through by p^l, and observe that

$$ra \equiv d \pmod{n} \text{ if and only if } \left(\frac{a}{p^l}\right) a \equiv \left(\frac{d}{p^l}\right) \pmod{\frac{p^k}{p^l}}.$$

However, in the latter equation $\left(\frac{a}{p^l}\right)$ is relatively prime to $\left(\frac{p^k}{p^l}\right)$, thus we can invert a and solve for r. □

6.6.3. The Solution. Given the above results, we are now able to devise an algorithm for solving problem 6.2. Shor's major result is the following theorem, solving the Discrete Log problem in quantum polynomial time.

THEOREM 6.4. *There exists a quantum Turing Machine M and a polynomial P, such that given an n bit prime p, a generator $g \in \mathbb{Z}_p^*$ and a number $x \in \mathbb{Z}_p^*$, M returns a number $r \in \mathbb{Z}_{p-1}$ such that $g^r = x$ in time $P(u)$ with error at most $\frac{1}{3}$, where u is the length of p, g,and x in their binary representation.* [9]

PROOF We begin the proof by presenting an algorithm similar to the one used in the proof of Theorem 6.2. We then define what we consider to be good outputs from the algorithm, and argue that a polynomial sample of good outputs will enable us to

deduce the discrete logarithm r. Finally, we show that the algorithm gives a good output with constant probability.

Just as in the proof of Theorem 6.2, we will not mention in the description of any particular configuration any of those values that are never erased (such as the inputs). So given a prime p, a generator $g \in \mathbb{Z}_p^*$ and a number $x \in \mathbb{Z}_p^*$. The algorithm will be as follows:

(1) Use Lemma 6.3 to find a smooth number q such that $p < q \leq 2p$.

(2) Choose two elements a, b from $\mathbb{Z}_{p-1}$ at random, to get $|a, b\rangle$ on the tape.

(3) We compute $g^a x^{-b}$ on the tape to get $|a, b, g^a x^{-b}\rangle$.

(4) Compute F_q on a and F_q on b.

A quick observation and appeal to Lemma 6.3 and Theorem 4.1 reveals that each one of these steps can be done in polynomial time. The only step which requires some justification is step 4, in which we compute F_q on a and b. In this case we compute F_q on a and b despite of the fact that both elements are in $\mathbb{Z}_{p-1}$ rather then $\mathbb{Z}_q$. However since $q > p$, we can view $\mathbb{Z}_p$ as a subset of $\mathbb{Z}_q$. Since q is smooth, we can compute F_q in polynomial time by appealing to Lemma 6.3.

Now consider the state of the machine after step 2. It is found to be in a superposition of all possible choices for a and b as follows:

$$\frac{1}{p-1} \sum_{b=0}^{p-2} \sum_{a=0}^{p-2} |a, b\rangle.$$

Step 3 will place the machine in the superposition:

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

Breaking down step 4, we first compute F_q on a to get the superposition:

$$\frac{1}{p-1} \sum_{c=0}^{q-1} \sum_{b=0}^{p-2} \sum_{a=0}^{p-2} \frac{1}{\sqrt{q-1}} e^{\frac{2\pi i a c}{q}} |c, b, g^a x^{-b}\rangle$$

We then compute F_q on b to get the superposition:

$$\frac{1}{p-1}\sum_{d=0}^{q-1}\sum_{c=0}^{q-1}\sum_{b=0}^{p-2}\sum_{a=0}^{p-2}\frac{1}{\sqrt{q-1}}e^{\frac{2\pi iac}{q}}\frac{1}{\sqrt{q-1}}e^{\frac{2\pi ibd}{q}}|c,d,g^a x^{-b}\rangle$$

$$= \frac{1}{p-1}\frac{1}{q}\sum_{d=0}^{q-1}\sum_{c=0}^{q-1}\sum_{b=0}^{p-2}\sum_{a=0}^{p-2}e^{\frac{2\pi iac+bd}{q}}|c,d,g^a x^{-b}\rangle$$

Observing the machine upon termination of the algorithm, we will find it in a state $|c,d,y\rangle$ where $y = g_k$ for some $k \in \mathbb{Z}_{p-1}$. We will define a *good* output to be a state $|c,d,y\rangle$ that bears the following two properties:

(1) $\left|rc + d - \frac{r[c(p-1) \mod q]}{p-1}\right| \leq \frac{1}{2}\{\mod q\}$

(2) $[c(p-1) \mod q] \leq \frac{q}{20}$

By $\{\mod q\}$ we mean the remainder after division of a real number by q, (e.g. $5.5\{\mod 3\} = 2.5$.) We intend to show that given a polynomial size sample of good final states we can deduce r. To that end, we will show how to extract an equation of the form $ar = b \pmod{p-1}$ from property 1, and then using a polynomial sample of these with property 2 will indeed be enough to solve for r.

Property 1 states that:

$$\left|rc + d - \frac{r[c(p-1) \mod q]}{p-1}\right| \leq \frac{1}{2}\{\mod q\}$$

Dividing through by q we get:

$$\left|\frac{rc}{q} + \frac{d}{q} - \frac{r[c(p-1) \mod q]}{(p-1)q}\right| \leq \frac{1}{2}\{\mod 1\}$$

Rearranging terms and factoring r we arrive at:

$$\left|\frac{d}{q} + r\left(\frac{c}{q} - \frac{r[c(p-1) \mod q]}{(p-1)q}\right)\right| \leq \frac{1}{2q}\{\mod 1\}. \tag{6}$$

We let,

$$\alpha = \frac{c(p-1) - [c(p-1) \mod q]}{q}.$$

and rewrite 6 as

$$\left|\frac{d}{q} + r\left(\frac{\alpha}{(p-1)q}\right)\right| \leq \frac{1}{2q}\{\mod 1\}.$$

Note that α is an integer, since $q|(c(p-1) - [c(p-1) \mod q])$. Now, by simply rounding off the number $\frac{d}{q}$ to the *nearest* multiple of $\frac{1}{p-1}$, say $\frac{d'}{p-1}$, we can find d' such that

$$\left|\frac{d'}{p-1} - \frac{d}{q}\right| \leq \frac{1}{2(p-1)}.$$

We now consider $\left|\frac{d'+r\alpha}{p-1}\right| \{\mathrm{mod}\ 1\}$, as follows:

$$\begin{aligned}
\left|\tfrac{d'+r\alpha}{p-1}\right| &= \left|\tfrac{d'}{p-1} - \tfrac{d}{q} + \tfrac{d}{q} + \tfrac{r\alpha}{p-1}\right| \quad \{\mathrm{mod}\ 1\} \\
&\leq \left|\tfrac{d'}{p-1} - \tfrac{d}{q}\right| + \left|\tfrac{d}{q} + \tfrac{r\alpha}{p-1}\right| \quad \{\mathrm{mod}\ 1\} \qquad \text{by triangle inequality} \\
&\leq \tfrac{1}{2(p-1)} + \tfrac{1}{2q} \qquad\qquad \{\mathrm{mod}\ 1\}.
\end{aligned}$$

Multiplying both sides by $p-1$ we get,

$$\begin{aligned}
|d' + r\alpha| &\leq \frac{p-1}{2(p-1)} + \frac{p-1}{2q} \\
&< \frac{1}{2} + \frac{p-1}{2(p-1)} \\
&= 1 \{\mathrm{mod}\ p-1\}
\end{aligned}$$

Hence, $|d' + r\alpha| < 1 \{\mathrm{mod}\ p-1\}$, but $d' + r\alpha$ is an integer, whence $d' + r\alpha \equiv 0 \pmod{p-1}$. At this point we once again encounter the problem of being unable to invert $\alpha' = -\alpha$ unless it is relatively prime to $p-1$. We claim though that while in general α' won't be relatively prime to $p-1$, a polynomial sample of such equations will indeed allow us to deduce r.

We write $p-1 = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$, where the p_i's are the prime factors of $p-1$ in increasing order. Using the fact that for any number $w = w_1 w_2$, if $\gcd(w_1, w_2) = 1$, then $\mathbb{Z}_w \cong \mathbb{Z}_{w_1} \times \mathbb{Z}_{w_2}$, we decompose

$$\mathbb{Z}_{p-1} \cong \mathbb{Z}_{p_1^{k_1}} \times \mathbb{Z}_{p_2^{k_2}} \times \cdots \times \mathbb{Z}_{p_s^{k_s}} \times \mathbb{Z}_{p_{s+1}^{k_{s+1}} p_{s+2}^{k_{s+2}} \cdots p_m^{k_m}}$$

where $1 \leq i \leq s$ implies that $p_i < 40$. This decomposition can be done in polynomial time since we are only required to find the factors of $p-1$ that are less than forty. Since the two groups are isomorphic, it is suffices for us to solve the equation $d' = r\alpha'$

in each one of the groups $\mathbb{Z}_{p_1^{k_1}}, \mathbb{Z}_{p_2^{k_2}}, ..., \mathbb{Z}_{p_s^{k_s}}, \mathbb{Z}_{p_{s+1}^{k_{s+1}} p_{s+2}^{k_{s+2}} ... p_m^{k_m}}$ from the decomposition. The answer in $\mathbb{Z}_{p-1}$can then be recovered using Chinese remaindering. Now, for each $1 \leq \imath \leq s$, we can solve the equation in $\mathbb{Z}_{p_\imath^{k_\imath}}$ using Lemma 6.5.

We now observe that if $p_i > 40$ is a prime such that $p_i | p-1$ then $\frac{20}{p_i}$ is an upper bound on the probability that $p_i | \alpha'$. From the algorithm we can generate a set of t, α_i''s uniformly and randomly satisfying $\alpha_i' r = d_i \pmod{p-1}$. The probability of any particular prime $p_i > 40$ dividing all of the t, a_i's is thus at most $\left(\frac{20}{p_i}\right)^t$. Now,

$$\begin{aligned}
& \Pr\left[\text{there exists a } p_i | p-1 \text{ such that } p_i \text{ divides all } t \text{ } \alpha'\text{s}\right] \\
\leq & \sum_{\substack{p_i > 40 \\ p_i | p-1}} \Pr[p_i \text{ divides all } t \text{ } \alpha'\text{s}] \\
\leq & \sum_{\substack{p_i > 40 \\ p_i | p-1}} \left(\frac{20}{p_i}\right)^t \\
\leq & \sum_{n=40}^{\infty} \left(\frac{20}{n}\right)^t \\
\leq & \varepsilon. \qquad\qquad \text{if } t \geq 2
\end{aligned}$$

If t is large enough then ε can be made small enough to ensure that for every prime $p_i | p$, $p_i > 40$, there exists some α_i' such that $\alpha_i' r = d_i \pmod{p-1}$ for some d_i and $p_i \not| \alpha_i$ (Pr_i does not divide α_i. We now apply Lemma 6.4 to solve these equations. To summarize, we have thus far shown that if we can could obtain a polynomial number of good states then we would be able to solve for r.

It remains to show that we get a good output with constant probability. We begin by analyzing the probability of arriving at a particular state $|c, d, y\rangle$ where $y = g^k$. Since after step four of the algorithm the machine is the superposition

$$\frac{1}{q(p-1)} \sum_{d=0}^{q-1} \sum_{c=0}^{q-1} \sum_{b=0}^{p-2} \sum_{a=0}^{p-2} e^{\frac{2\pi i ac+bd}{q}} |c, d, g^{a-rb}\rangle$$

the desired probability is:

$$\left| \frac{1}{(p-1)q} \sum_{\substack{a,b \in \mathbb{Z}_{p-1} \\ a - rb = k \pmod{p-1}}} e^{\frac{2\pi i(ac+bd)}{q}} \right|^2.$$

Observe that because $br+k = (br+k \pmod{p-1}) + (p-1)\left\lfloor \frac{br+k}{p-1} \right\rfloor$, we can substitute

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

in the equation above to get

$$\left| \frac{1}{(p-1)q} \sum_{b=0}^{p-2} e^{\frac{2\pi i}{q}((br+k-(p-1)\lfloor \frac{br+k}{p-1} \rfloor)c+bd)} \right|^2.$$

We eliminate a constant factor of $e^{\frac{2\pi ikc}{q}}$ and arrive at,

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

We then add and subtract a term of $\frac{br}{p-1}[c(p-1) \mod q]$ which yields,

$$\left| \frac{1}{(p-1)q} \sum_{b=0}^{p-2} e^{\frac{2\pi ib}{q}(rc+d-\frac{br}{p-1}[c(p-1) mod q])} e^{\frac{2\pi i}{q}(\frac{br}{p-1} - \lfloor \frac{br+k}{p-1} \rfloor)[c(p-1) \mod q]} \right|^2.$$

We want to analyse the magnitude of this quantity in the case of a good output. So we begin by first considering the somewhat simpler sum, which does not include the second exponential:

$$\frac{1}{(p-1)q} \sum_{b=0}^{p-2} e^{\frac{2\pi ib}{q}(rc+d-\frac{br}{p-1}[c(p-1) mod q])}$$

We rewrite it as:

$$\frac{1}{q}\left[\frac{\sum_{b=0}^{p-2} e^{\frac{2\pi ib}{q}(rc+d-\frac{br}{p-1}[c(p-1) mod q])}}{p-1} \right].$$

The first thing to observe is that the sum is a sum of vectors in the complex plane. This is because each one of these vectors is simply some q-th root of unity situated on the unit circle. We can view the quantity in the brackets as taking the "average" of all these vectors. We are interested in lower bounding this so called average for

those cases where the output is good. Recall that in a good output:

$$\delta = \left| rc + d - \frac{r}{p-1}[c(p-1) \mod q] \right| \leq \frac{1}{2}\{\mod q\}$$

So in fact as b ranges from 0 to $p-2$ the angles in the sum range from 0 to $\frac{2\pi(p-2)\delta}{q} \leq \pi$. Hence, all the vectors in the sum are in the top half of the unit circle. Figure 6.1 depicts this situation.

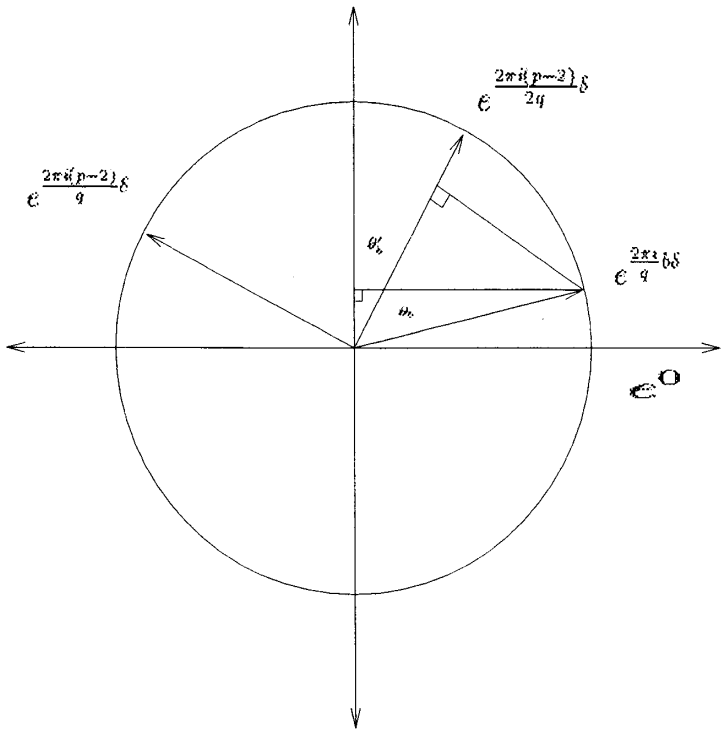

FIGURE 6.1. Vectors Around the Unit Circle

Now, if we want to lower bound the average of the vectors, we can simply choose a direction and lower bound the average of the vectors by the average of the projections of all the vectors in that direction. We choose the direction of the vector $e^{\frac{2\pi i}{q}(\frac{p-2}{2})\delta}$. This direction is chosen specifically since it is expected to be close to the average

direction of all the vectors (note that the angle formed by this vector and the first vector bisects the one formed by the last and first vectors.) The projection of an arbitrary vector $e^{\frac{2\pi i}{q}b\delta}$, for $b \leq \frac{p-2}{2}$ is $\cos(\theta_b)$ where θ_b is the angle between $e^{\frac{2\pi i}{q}b\delta}$ and $e^{\frac{2\pi i}{q}(\frac{p-2}{2})\delta}$. However, as seen in Figure 6.1, if θ'_b is the angle between $e^{\frac{2\pi i}{q}b\delta}$ and the y-axis then

$$\cos(\theta_b) \geq cos(\theta'_b) = \sin(\frac{\pi}{2} - \theta'_b) = \sin(\frac{2\pi}{q}b\delta) \geq \sin(\frac{\pi b}{2p}).$$

Therefore,

$$\begin{aligned} \frac{1}{q}\left[\sum_{b=0}^{p-2} e^{\frac{2\pi i b\delta}{q}} p - 1\right] &\geq \frac{2}{q}\sum_{b=0}^{\frac{p-2}{2}} \sin(\frac{\pi b}{2p}) \\ &\approx \frac{2}{q\pi}\int_0^{\frac{\pi}{2}} sinb\, db \\ &\in \Omega(\frac{1}{q}) \end{aligned}$$

However, the sum we that we are actually interested in lower bounding contains the same angles shifted by at most, $\frac{\pi}{10}$ (this is the only effect that the second exponential has.). If we apply this shift in a way that minimizes the possible resulting projection we get,

$$\frac{2}{q\pi}\int_{\frac{-\pi}{10}}^{\frac{4\pi}{10}} sinb\, db \in \Omega(\frac{1}{q})$$

Therefore the probability of each good state is in $\Omega(\frac{1}{q^2})$. Now, for every c there is exactly one d such that condition 1 holds, and there are approximately $\frac{q}{20}$ c's such that condition 2 holds. For each pair c and d there are $p-1$ possible values for y. Thus, there are $\frac{(p-1)q}{20}$ possible good states. The probability of obtaining some good state is therefore in $\Omega\left(\frac{(p-1)q}{q^2}\right) = \Omega(\frac{1}{2})$, and hence it is constant. This completes the proof. [9] $\square$

CHAPTER 7
Factoring

Perhaps the most striking result in the quantum computation theory is Shor's result regarding the factoring of integers. Factoring of an integer into it's prime factors is considered a very difficult problem on a classical Turing Machine, and one for which no known polynomial time algorithm exists. In his paper Shor proves that a quantum Turing machine can compute the prime factors of an integer in polynomial time. In order to prove Shor's result, we will require some background in the elegant theory of continued fractions.

7.1. Continued Fractions

A finite continued fraction is a fraction of the form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots \cfrac{\vdots}{a_{n-1} + \cfrac{1}{a_n}}}}}$$

where each of the a_i's are integers.

Every rational number can be expressed as a finite continued fraction, and the sequence of integers $a_1, a_2, a_3, ...a_n$ is called the continued fraction expansion. In order to find the continued fraction expansion of a particular rational number x,

we let $a_0 = \lfloor x \rfloor$, and $x_0 = x - \lfloor x \rfloor$. Then, for each $i \geq 1$, let $a_i = \left\lfloor \frac{1}{x_{i-1}} \right\rfloor$ and $x_i = \frac{1}{x_{i-1}} - a_i$. When $x_i = 0$ we stop and set $n = i$. The claim is that this algorithm produces $a_0, a_1, ..., a_n$ such that

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots \cfrac{\vdots}{a_{n-1} + \cfrac{1}{a_n}}}}}$$

This can be seen by observing that if $x_i = \frac{1}{x_{i-1}} - a_i$ then $x_{i-1} = \frac{1}{x_i + a_i}$. Recursively applying this equation we get:

$$x = a_0 + x_0 = a_0 + \frac{1}{a_1 + x_1} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + x_2}} = \ldots$$

When $x_i = 0$ this process terminates. It is easy to show that the $x_i's$ are fractions with decreasing denominators, and thus this process will indeed terminate if x is rational. Furthermore, we can define the sequences b_i and c_i relative to a given continued fraction expansion a_i to be:

$$\frac{b_i}{c_i} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots \cfrac{\vdots}{a_{i-1} + \cfrac{1}{a_i}}}}}.$$

We call the sequence of fractions $\frac{b_i}{c_i}$ the convergents of x, and present the following algorithm to find them.

Let $\frac{b_0}{c_0} = \frac{a_0}{1}$ and $\frac{b_1}{c_1} = \frac{a_0 a_1 + 1}{a_1}$. Then for every $i \geq 2$, let $\frac{b_i}{c_i} = \frac{a_i b_{i-1} + b_{i-2}}{a_i c_{i-1} + c_{i-2}}$. The correctness of this algorithm is easily proven by induction (we omit the proof here). Moreover, it can be shown that $gcd(b_i, c_i) = 1$ for all $i \geq 1$. In lieu of a proof we provide an example, which will illustrate the method behind continued fraction expansion.

EXAMPLE 7.1. Consider the fraction $\frac{377}{562}$. Then

$$
\begin{array}{lll}
a_0 = \left\lfloor \frac{377}{562} \right\rfloor = 0 & x_0 = \frac{377}{562} - 0 = \frac{377}{562} & \frac{b_0}{c_0} = \frac{0}{1}. \\
a_1 = \left\lfloor \frac{562}{377} \right\rfloor = 1 & x_1 = \left\lfloor \frac{562}{377} \right\rfloor - 1 = \frac{185}{377} & \frac{b_1}{c_1} = \frac{1}{1}. \\
a_2 = \left\lfloor \frac{377}{185} \right\rfloor = 2 & x_2 = \left\lfloor \frac{377}{185} \right\rfloor - 2 = \frac{7}{185} & \frac{b_2}{c_2} = \frac{2\times1+0}{2\times1+1} = \frac{2}{3}. \\
a_3 = \left\lfloor \frac{185}{7} \right\rfloor = 26 & x_3 = \left\lfloor \frac{185}{7} \right\rfloor - 26 = \frac{3}{7} & \frac{b_3}{c_3} = \frac{26\times2+1}{26\times3+1} = \frac{53}{79}. \\
a_4 = \left\lfloor \frac{7}{3} \right\rfloor = 2 & x_4 = \left\lfloor \frac{7}{3} \right\rfloor - 2 = \frac{1}{3} & \frac{b_4}{c_4} = \frac{2\times53+2}{2\times79+3} = \frac{108}{161}. \\
a_5 = \lfloor 3 \rfloor = 3 & x_5 = \lfloor 3 \rfloor - 3 = 0 & \frac{b_5}{c_5} = \frac{3\times108+53}{3\times161+79} = \frac{377}{562}.
\end{array}
$$

Note that the algorithm is simply performs an efficient top down evaluation of the continued fraction expansion of

$$
\frac{377}{562} = \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{26 + \cfrac{1}{2 + \frac{1}{3}}}}}.
$$

This algorithm runs in time polynomial in $\log(x)$ on a deterministic Turing Machine. This is easily seen from the fact that the algorithm is essentially Euclid's greatest common divisor algorithm applied to fractions. We can use it to find all of the convergents to any rational number x. Since this algorithm is not required at any stage to erase the rational number input, we have effectively proven the following lemma.

LEMMA 7.1. *There exists a Quantum Turing Machine M and a polynomial P, such that given a rational number* $\frac{a}{b}$, *M finds all the convergents to* $\frac{a}{b}$ *in time* $P(\log(a) + \log(b))$, *i.e. polynomial in their binary representation.*

7.2. The order of an element in $\mathbb{Z}_n^*$

The Integer Factorization problem is the following:

PROBLEM 7.1. *Integer Factorization. Given a composite number n, we would like to find* $p \geq 2$ *and* $q \geq 2$ *such that* $n = pq$.

The algorithm proposed by Shor to solve Integer Factorization attempts to find the order of an element in a multiplicative group $\mathbb{Z}_n^*$. There exists a BPP reduction proposed by Miller [15] from factoring into determining the order of an element. Thus, given an algorithm to find the order of an element in $\mathbb{Z}_n^*$ for any n, we can use this reduction to factor any integer. The idea in the reduction is that given a number n to factor, we randomly and uniformly choose an element $x \in \mathbb{Z}_n^*$, and then compute its order r_x. Using Euclid's algorithm we find $d = \gcd(x^{\frac{r_x}{2}}, n)$. It can be shown, that with high probability d is neither 1 nor n and thus d is a non-trivial divisor of n.

In this section we present Shor's algorithm for the order of an element together with it's analysis. For clarity we define the Order of an element problem:

PROBLEM 7.2. *Order of an element. Given a positive integer n and an element* $x \in \mathbb{Z}_n^*$ *we would like to find the least positive integer r such that* $x^r = 1 \pmod{n}$.

The algorithm and the analysis is somewhat easier than the one for the general case of the Discrete Log problem. Just as in Discrete Log, we define which of the algorithm's outputs are considered good for our purposes, and then show that such outputs are expected with high probability. The idea being that a collection of good outputs will allow us through the use of continued fractions to deduce some fraction $\frac{d}{r}$ (for which we will know both the numerator and denominator), and such that r is the order of the element x, in $\mathbb{Z}_n^*$. This will prove the following theorem:

THEOREM 7.1. *There exists a quantum Turing Machine M and a polynomial P, such that given a positive integer n and an element $x \in \mathbb{Z}_n^*$ M finds the order of $x \in \mathbb{Z}_n^*$ in time $P(\log(n) + \log(x))$.*

PROOF Given n, and $x \in \mathbb{Z}_n^*$, the quantum algorithm begins by finding a number q such that $2n^2 < q \leq 4n^2$. This step can be done in polynomial time using Lemma 6.3.

Initially we start the machine in the state $|0\rangle$. As before the inputs are not mentioned since they are never erased. Using Lemma 6.3 we will compute the Fourier Transform F_q on the tape to get,

$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle$$

(Note that this is equivalent to choosing an arbitrary element in $\mathbb{Z}_q$). We now reversibly compute $x^a \mod n$ on the tape to arrive a the superposition:

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

At this point the machine is in q distinct states. However, the second component of the state, is some element in $\mathbb{Z}_n^*$ since $x \in \mathbb{Z}_n^*$. Furthermore, there are exactly r possible values for the second component, since r is the order of the element. Because $q > 2n^2 > 2r^2$, some of these states must have the same element in the second component.

At this point we compute F_q again on the first component of the tape, and the machine is then found in the superposition:

$$\frac{1}{q} \sum_{c=0}^{q-1} \sum_{a=0}^{q-1} |c, x^a \mod n\rangle.$$

When the algorithm stops we observe the outcome, and find the machine not in a superposition of states but rather in some particular state $|c, x^a \mod n\rangle$. We call such a state a good state if it satisfies:

$$\left| [rc \hat{\mod} n] \right| \leq \frac{r}{2} \tag{7}$$

The wedge above the mod is to indicate that $[rc \hat{\mod} n]$ is in the range $\frac{-rc}{q}$ to $\frac{rc}{q}$, i.e,

$$\frac{-rc}{q} \leq [rc \hat{\mod} n] \leq \frac{rc}{q},$$

as opposed to $[rc \mod q]$ which is in the range 0 to $q-1$. Given the above, let us see that for a good state there exists exactly one rational number with a denominator r, which is within a $\frac{1}{2q}$ neighborhood of $\frac{c}{q}$, i.e. it satisfies:

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

To that end suppose that the state is good; then, by the Division Theorem, there exists $d \geq 0$, $d \in \mathbb{Z}$, such that $rc = dq + [rc \hat{\mod} n]$. (Note that $d = \left\lfloor \frac{rc}{q} \right\rfloor$, but we can't find it since we lack knowledge of r.) Together with the condition 7 we have

$$|rc - dq| \leq \frac{r}{2}$$

Dividing by r and q we get,

$$\left|\frac{c}{q} - \frac{q}{r}\right| \leq \frac{1}{2q}.$$

So the fraction exists. Now suppose that there are two such fractions $\frac{d'}{r}$ and $\frac{d}{r}$. Then,

$$\begin{aligned}
\left|\frac{d'}{r} - \frac{d}{r}\right| &= \left|\frac{d'}{r} - \frac{c}{q} + \frac{c}{q} - \frac{d}{r}\right| \\
&\leq \left|\frac{d'}{r} - \frac{c}{q}\right| + \left|\frac{c}{q} - \frac{d}{r}\right| \\
&\leq \frac{1}{2q} + \frac{1}{2q} \leq \frac{1}{q} \leq \frac{1}{2n^2} \leq \frac{1}{2r^2}
\end{aligned}$$

Hence, $|d' - d| \leq \frac{1}{2}$, with d' and d integers, which implies that $d' = d$. So indeed there is exactly one fraction with denominator r satisfying the above condition for any particular good state. Since we are dealing with a good state we have,

$$\left|\frac{c}{q} - \frac{q}{r}\right| \leq \frac{1}{2q} \leq \frac{1}{4n^2} \leq \frac{1}{2r^2}.$$

We now appeal to Theorem 185 in *An Introduction to the Theory of Numbers* [16],

which states that for any integers α and β, and any rational number x, if $\left|\frac{\alpha}{\beta} - x\right| < \frac{1}{2\beta^2}$, then $\frac{\alpha}{\beta}$ is a convergent of x. In our case, since $\left|\frac{c}{q} - \frac{q}{r}\right| \leq \frac{1}{2r^2}$, we must have that $\frac{d}{r}$ is a convergent of $\frac{c}{q}$ and thus we can find it using Lemma 7.1. We will set the fraction $\frac{d}{r}$ to be the first convergent we find with denominator less than n.

The only problem which may arise in this scheme is that d may not be relatively prime to r and thus the fraction $\frac{d}{r}$ which we desire may not be in it's lowest terms (which would imply that Lemma 7.1 won't find it.) However, consider all possible numbers $d \leq r$ such that d is relatively prime to r. Suppose that there are α such numbers. Then, every one of the α fractions $\frac{d}{r}$ is close to some multiple c of the fraction $\frac{1}{q}$, where by close we mean,

$$\left|\frac{c}{q} - \frac{q}{r}\right| \leq \frac{1}{2q}.$$

This is due to the fact that there is always some multiple of $\frac{1}{q}$ that lies in the $\frac{1}{2q}$ neighborhood of any real number. Hence, every d relatively prime to r has a good state associated with it, and that state is unique with respect to c. Therefore, the number of c's such that $\gcd(d, r) = 1$ is α. Now, in each state we have $|c, x^a\rangle$, and for each c there are exactly r possible values which x^a can take. Thus in total, the number of good states $|c, x^a\rangle$ that have d relatively prime to r is $r\alpha$. But the number of elements relatively prime to r can be shown to be in $\Omega(\frac{r}{\log(r)})$ (this was done in the proof of Theorem 6.2.) So there are $\Omega(\frac{r^2}{\log(r)})$, possible good states from which we can deduce d.

We now show that each such state is seen with probability $\Omega(\frac{1}{r^2})$. This completes the proof for it means that the probability of seeing a good state from which we can deduce r is in $\Omega(\frac{1}{r^2})$. To see this we need to approximate the probability of seeing any particular state $|c, x^k\rangle$. If we observe the final superposition, we find that there is more than one computation path leading to every such state. In particular, all those states $|c, x^a\rangle$, such that $x^a \equiv x^k \pmod{n}$) are in the same computation path and interfere with each other. If we take the square of the absolute value of the sum

of the amplitudes we get

$$\left| \frac{1}{q} \sum_{\substack{a=0 \\ x^a \equiv x^k}}^{q-1} e^{\frac{2\pi i a c}{q}} \right|^2$$

Now, if $x^a \equiv x^k \pmod{n}$, then $x^{a-k} \equiv 1 \pmod{n}$, which implies that the order, r, of x divides $a-k$. Conversely, if $r|(a-k)$, then $x^a \equiv x^k \pmod{n}$. Therefore $a-k = rb$ where $0 \leq b \leq \left\lfloor \frac{q-k-1}{r} \right\rfloor$. Substituting for a, the probability of any given state is:

$$\left| \frac{1}{q} \sum_{b=0}^{\left\lfloor \frac{q-k-1}{r} \right\rfloor} e^{\frac{2\pi i (br+k)c}{q}} \right|^2$$

$$= \left| \frac{1}{q} \sum_{b=0}^{\left\lfloor \frac{q-k-1}{r} \right\rfloor} e^{\frac{2\pi i b r c}{q}} e^{\frac{2\pi i k c}{q}} \right|^2$$

Since $e^{\frac{2\pi i r c}{q}}$ is constant with respect to b, we can take it out of the sum. Furthermore, since it's magnitude is 1, we ignore it completely, and the above reduces to:

$$\left| \frac{1}{q} \sum_{b=0}^{\left\lfloor \frac{q-k-1}{r} \right\rfloor} e^{\frac{2\pi i b r c}{q}} \right|^2$$

We replace rc by $[rc \mod n]$, in the sum (since all multiples of q can be factored out). If the state is a good state, we know that

$$[rc \mod n] \leq \frac{r}{2}.$$

Therefore the angles of the complex vectors in the above sum range from 0 to:

$$\frac{2\pi \left\lfloor \frac{q-k-1}{r} \right\rfloor [rc \mod n]}{q} \leq \frac{2\pi \left\lfloor \frac{q-k-1}{r} \right\rfloor \frac{r}{2}}{q} \leq \pi.$$

Hence all the vectors lie in the first half of the unit circle. As we did in the general case of the Discrete-Log problem, we lower bound this sum, by the projection of each of the complex numbers on the y-axis, i.e. we consider only each vector's imaginary component's contribution to the sum. Thus, the quantity above is bounded below

by:

$$\left|\frac{1}{q}\sum_{b=0}^{\lfloor\frac{q-k-1}{r}\rfloor}\sin\left(2\pi\left(\frac{b}{q}\right)[rc \hat{\mod} n]\right)\right|^2.$$

We approximate the value of the sin function by its integral to get a lower bound:

$$\left|\frac{1}{q}\int_0^{\lfloor\frac{q-k-1}{r}\rfloor}\sin\left(2\pi\left(\frac{b}{q}\right)[rc \hat{\mod} n]\right)db\right|^2.$$

We use the substitution $y=\frac{b}{q}$, so $dy=\frac{1}{q}db$, and we have,

$$\begin{aligned}
&\left|\int_0^{\frac{1}{q}\lfloor\frac{q-k-1}{r}\rfloor}\sin\left(2\pi[rc \hat{\mod} n]y\right)dy\right|^2\\
=&\left|\frac{-\cos\left(2\pi[rc \hat{\mod} n]y\right)}{2\pi[rc \hat{\mod} n]}\Bigg|_0^{\frac{1}{q}\lfloor\frac{q-k-1}{r}\rfloor}\right|^2\\
=&\left|\frac{1}{2\pi[rc \hat{\mod} n]}-\frac{\cos\left(2\pi[rc \hat{\mod} n]\frac{1}{q}\left\lfloor\frac{q-k-1}{r}\right\rfloor\right)}{2\pi[rc \hat{\mod} n]}\right|^2
\end{aligned}$$

But $\cos\left(2\pi[rc \hat{\mod} n]\frac{1}{q}\left\lfloor\frac{q-k-1}{r}\right\rfloor\right)\leq\frac{1}{\sqrt{2}}$, if $[rc \hat{\mod} n]\geq\frac{r}{8}$, since the last angle is then larger than $\frac{\pi}{4}$. Under this condition the quantity above is,

$$\begin{aligned}
&\geq\left|\frac{1-\frac{1}{\sqrt{2}}}{2\pi[rc \hat{\mod} n]}\right|^2\\
&\geq\left|\frac{1}{6\pi\sqrt{2}[rc \hat{\mod} n]}\right|^2
\end{aligned}$$

Since this is a good state, $[rc \hat{\mod} n]\leq\frac{r}{2}$, thus we can bound the quantity above below by

$$\left|\frac{2}{6\pi\sqrt{2}r}\right|^2\geq\left|\frac{1}{6\pi r}\right|^2=\frac{1}{36\pi^2r^2}.$$

This last quantity is clearly in $\Omega\left(\frac{1}{r^2}\right)$. A similar argument works when $[rcmodq]\leq\frac{r}{8}$. In that case we approximate the sum of the vectors by their projection on the real

number line rather than the imaginary. The proof is complete.

CHAPTER 8
Limitations of the Model

Shor's and Simon's result suggest that the quantum Turing Machine has a tremendous potential. Thus far they have provided us with three problems which are not known to be efficiently solvable on a classical Turing Machine, yet have efficient solutions on a probabilistic Turing Machine. All three problems are in NP, and none are known to be in P. Unfortunately, complexity theory has not yet managed to classify the Discrete-Log problem or the Integer Factorization problem as NP-complete, and that does not seem to be a likely result in the near future. There is therefore a gap in knowledge regarding the relative complexity of the above problems to the NP-complete problems, and we are tempted to ask whether is it possible to solve one of the NP-complete problems in polynomial time on a quantum Turing Machine?

Unfortunately, the answer seems to be as before - we don't know. However, the following theorem suggests that the Quantum Turing Machine exhibits the same difficulty as all other models to date.

THEOREM 8.1. *Relative to a random oracle R, with probability 1, NP^R includes languages not in BQP^R* [5]

This theorem suggests that in order to have any hope of deciding a random language in NP in polynomial time on a QTM we must explore the description of the oracle beyond using the oracle as a verifier. This, however, is also the state of the art in the continuing efforts to solve the $P = NP$ problem in the classical setting.

In order to prove Theorem 8.1, we must first define the notion of indistinguishable final superpositions. The definition is introduced in order for us to discuss limitations on the decidability of languages. In order to motivate the definition, we consider a quantum Turing Machine M deciding some language $L \in BQP$. By definition of BQP, we know that the Turing Machine will classify an input x with respect to L with an error bound of at most $\frac{1}{3}$. Now, suppose that another quantum Turing Machine does the same with error bound of at most $\frac{5}{12}$; in that case, both machines will classify x with respect to L correctly up to a constant difference in the error bound. It is easy to see that running the latter machine 7 times on the same input and computing the majority function on the results decreases it's probability of failure to less than $\frac{1}{3}$. Hence, the two machines are equivalent within a constant factor in the running time.

DEFINITION 8.1. *Given an quantum Turing Machine M, an input x, and two quantum oracles O, and O', let $M^O[x]$ and $M^{O'}[x]$ be the the final state vectors of the quantum Turing machine operating on input x with oracles O' and O respectively. We will say that the Turing Machine does not distinguish between the oracles, if the probability of associated with any subset of configurations in $M^O[x]$ is different from the probability of associated with the same subset of configuration in $M^{O'}[x]$, by at most $\frac{1}{12}$.*

Given this, we are able to introduce the following theorem:

THEOREM 8.2. *Let M be a BQP machine that runs in time $p(n)$, O be an oracle computing $g : \{0,1\}^n \to \{0,1\}^n$, and $x \in \{0,1\}^n$. Then for any subset $S \subseteq \{0,1\}^n$ of size greater than $20p(n)$, there exist two strings $s_1, s_2 \in S$, $s_1 \neq s_2$, such that if an*

oracle O' computes the function $g' : \{0,1\}^n \to \{0,1\}^n$ where,

$$g'(t) = \begin{cases} g(t) & \text{if } t \neq s_1 \text{ and } t \neq s_2, \\ g(s_2) & \text{if } t = s_1, \\ g(s_1) & \text{if } t = s_2. \end{cases}$$

then, $M^O[x]$ is indistinguishable from $M^{O'}[x]$.

PROOF Let M be a BQP machine that runs in time $p(n)$,O be an oracle computing $g : \{0,1\}^n \to \{0,1\}^n$, and $x \in \{0,1\}^n$. Let $S \subseteq \{0,1\}^n$ be such that $|S| = k$. We consider the computation of a BQP machine running in time $p(n)$. Since the number of tape cells used in any polynomial time computation is a polynomial in n, we can consider the computation to be a sequence of applications of finite unitary matrices on the initial state vector. The composition of these unitary matrices is

$$U_r O U_{r-1} O U_{r-2} \cdots O U_0.$$

where each the U_i's correspond to a computation which does not query the oracle, and O corresponds to an oracle query. We will reserve a certain portion of the machine's tape for oracle queries, and thus the representation of any configuration will require 2^{n+m} entries in the state vector, for some m. (The proof itself is independent of m.) Let $\vec{x}$ be the initial state vector. Then $\vec{x}$ has a 1 in the entry corresponding to input x, and 0's everywhere else. Let

$$\vec{x_0} = U_0(\vec{x}),$$

and

$$\vec{x_t} = U_t O(\vec{x_{t-1}}).$$

Then, it is easy to see that the final superposition vector is $\vec{x_r}$. We consider the superposition of states in the state vector $\vec{x_i}$. It has 2^{m+n} configurations that are about to query O. (Although only some of them have amplitudes of magnitude greater than 0.) However, only $k2^m$ of those configurations are about to query strings

in S. Define Q to be the set of configurations that are about to query strings in S. Let x_{tq} denote the entry in $\vec{x_t}$ corresponding to configuration q; then,

$$\sum_{q \in Q} |x_{tq}|^2 \leq 1.$$

This is simply because each of the x_{tq}'s is an entry in the state vector and the sum of the squares of the amplitudes in the state vector is always 1. Therefore,

$$\sum_{i=0}^{r} \sum_{q \in Q} |x_{tq}|^2 \leq r.$$

However, since we chose to reserve n bits on the tape for oracle queries each configuration q can be indexed by $js = q$, where $j \in \{0,1\}^m$ and $s \in S \subseteq \{0,1\}^n$. So we write,

$$\sum_{i=0}^{r} \sum_{s \in S} \sum_{j \in \{0,1\}^m} |x_{t(js)}|^2 \leq r. \tag{8}$$

By the pigeon hole principle, there exists $s_1 \in S$ such that

$$\sum_{i=0}^{r} \sum_{j \in \{0,1\}^m} |x_{t(js_1)}|^2 \leq \frac{r}{|S|} \leq \frac{r}{|S|-1}. \tag{9}$$

However, from 8 we can also deduce that

$$\sum_{i=0}^{r} \sum_{s \in S \setminus \{s_1\}} \sum_{j \in \{0,1\}^m} |x_{t(js)}|^2 \leq r.$$

and applying the pigeon hole principle again we get,

$$\sum_{i=0}^{r} \sum_{j \in \{0,1\}^m} |x_{t(js_2)}|^2 \leq \frac{r}{|S|-1}. \tag{10}$$

We now construct the oracle O' as in the statement of the theorem, and consider the magnitude of the difference in the computation of M using O and the computation of M using O'. This difference is:

$$|U_r O' U_{r-1} \cdots U_1 O' U_0(\vec{x}) - U_r O U_{r-1} \cdots U_1 O U_0(\vec{x})|.$$

We would like to show that this distance is small. Towards that end, we write

each vector $\vec{x_t} = \vec{v_t} + \vec{w_t}$, where $\vec{v_t}$ is the vector which has non-zero entries only in components corresponding to configurations querying s_1 or s_2, and $\vec{w_t}$ has non-zero entries only in components corresponding to configurations which do not query either s_1 or s_2. In the same way, we introduce a new sequence of vectors $\vec{x'_t}$, $\vec{v'_t}$ and $\vec{w'_t}$, such that $\vec{x'_t} = \vec{v'_t} + \vec{w'_t}$, together corresponding to the computation of M using O'. Now,

$$\begin{aligned} \vec{x_1} &= U_1 O(\vec{x_0}) \\ &= U_1 O(\vec{v_0} + \vec{w_0}) \\ &= U_1 O(\vec{v_0}) + U_1 O(\vec{w_0}) \end{aligned}$$

Also, consider $\vec{x'_1}$

$$\begin{aligned} \vec{x'_1} &= U_1 O'(\vec{x_0}) \\ &= U_1 O'(\vec{v_0} + \vec{w_0}) \\ &= U_1 O'(\vec{v_0}) + U_1 O'(\vec{w_0}) \end{aligned}$$

Since O operates the same way as O' does on the vector $\vec{w_0}$, we have

$$\begin{aligned} \vec{x'_1} &= U_1 O'(\vec{v_0}) + U_1 O(\vec{w_0}) \\ &= U_1 O'(\vec{v_0}) + U_1 O(\vec{x_0}) - U_1 O(\vec{v_0}) \\ &= \vec{x_1} + U_1 O'(\vec{v_0}) - U_1 O(\vec{v_0}) \end{aligned}$$

We have established in the last step a relationship between $\vec{x'_1}$ and $\vec{x_1}$, and we shall now attempt to find such a relationship in general. To that end, we do the same with

x'_2 as follows:

$$\begin{aligned}
\vec{x'_2} &= U_2O'(\vec{x'_1}) \\
&= U_2O'(\vec{x_1} + U_1O'(\vec{v_0}) - U_1O(\vec{v_0})) \\
&= U_2O'(\vec{x_1}) + U_2O'U_1O'(\vec{v_0}) - U_2O'U_1O(\vec{v_0}) \\
&= U_2O'(\vec{v_1} + \vec{w_1}) + U_2O'U_1O'(\vec{v_0}) - U_2O'U_1O(\vec{v_0}) \\
&= U_2O'(\vec{v_1}) + U_2O'(\vec{w_1}) + U_2O'U_1O'(\vec{v_0}) - U_2O'U_1O(\vec{v_0}) \\
&= U_2O'(\vec{v_1}) + U_2O(\vec{w_1}) + U_2O'U_1O'(\vec{v_0}) - U_2O'U_1O(\vec{v_0}) \\
&= U_2O'(\vec{v_1}) + U_2O(\vec{x_1}) - U_2O(\vec{v_1}) + U_2O'U_1O'(\vec{v_0}) - U_2O'U_1O(\vec{v_0}) \\
&= \vec{x_2} + U_2O'(\vec{v_1}) - U_2O(\vec{v_1}) + U_2O'U_1O'(\vec{v_0}) - U_2O'U_1O(\vec{v_0})
\end{aligned}$$

In general,

$$\begin{aligned}
&\vec{x'_t} = \\
&\quad \vec{x_t} + U_tO'(\vec{v_{t-1}}) - U_tO(\vec{v_{t-1}}) + U_tO'U_{t-1}O'(\vec{v_{t-2}}) - U_tO'U_{t-1}O(\vec{v_{t-2}}) \\
&\quad + \cdots + U_tO'U_{t-1}O' \cdots U_1O'(\vec{v_0}) - U_tO'U_{t-1}O' \cdots U_1O(\vec{v_0}).
\end{aligned}$$

Therefore the magnitude in the difference between the two computations:

$$\begin{aligned}
|\vec{x'_r} - \vec{x_r}| \;\leq\; & |U_r(O' - O)(\vec{v_{r-1}})| + |U_rO'U_{r-1}(O' - O)(\vec{v_{r-2}})| \\
& + \cdots + |U_rO'U_{r-1}O' \cdots U_1(O' - O)(\vec{v_0})|
\end{aligned}$$

However, these matrices are unitary, and they therefore preserve length. Hence the above is,

$$\begin{aligned}
&= |(O' - O)(\vec{v_{r-1}})| + |(O' - O)(\vec{v_{r-2}})| + \cdots + |(O' - O)(\vec{v_0})| \\
&\leq 2|\vec{v_{r-1}}| + 2|\vec{v_{r-2}}| + \cdots + |\vec{v_0}| \\
&= 2\sum_{i=0}^{r-1} |\vec{v_i}|.
\end{aligned}$$

We now note that $|\vec{v_i}| = \sum_{j\in\{0,1\}^m} |x_{i(js_1)}|^2 + |x_{i(js_2)}|^2$, and using equations 9 and 10 we have,

$$\begin{aligned} |\vec{x_r'} - \vec{x_r}| &\leq 2\sum_{i=0}^{r}\sum_{j\in\{0,1\}^m} |x_{i(js_1)}|^2 + |x_{i(js_2)}|^2 \\ &\leq 2\left(\frac{2r}{|S|-1}\right) \end{aligned}$$

Since $r \leq p(n)$, we set $|S| - 1 \geq 20p(n)$. So that,

$$|\vec{x_r'} - \vec{x_r}| \leq \frac{1}{5}$$

We will now show that the difference in the probabilities between $\vec{x_r'}$ and $\vec{x_r}$ over any subset of configurations is less then $\frac{1}{12}$. Let $T \subseteq \{0,1\}^n + m$, so T is a subset of all possible configurations. Recall that the Cauchy-Schwartz inequality implies that,

$$\left|\sum c\vec{x_{rc}}\vec{x_{rc}'}\right| \leq \left(\sum_c |\vec{x_{rc}}|^2\right)\left(\sum_c |\vec{x_{rc}'}|^2\right)$$

$$\begin{aligned} \sum_{c\in T} |x_{rc}'|^2 - \sum_{c\in T} |x_{rc}|^2 &\leq |\vec{x_r'} - \vec{x_r}|^2|\vec{x_r'} + \vec{x_r}|^2 \\ &\leq |\vec{x_r'} - \vec{x_r}|^2(|\vec{x_r'}|^2 + |\vec{x_r}|)^2 \\ &\leq 2|\vec{x_r'} - \vec{x_r}|^2 \leq 2\left(\frac{1}{5}\right)^2 \leq \frac{1}{12} \end{aligned}$$

Hence the two final outcomes are indistinguishable. □

Theorem 8.2 essentially states that a quantum Turing Machine can not be sensitive to more than a polynomial number of queries. Theorem 8.1 is a corollary to that, since a non-deterministic Turing Machine is clearly not plagued by such a limitation. It should be mentioned that the authors of *Strengths and Weakness of Quantum Computation* claim that Theorem 8.2 shows that given a random permutation oracle O, there exist no quantum Turing Machine which inverts O in polynomial time with non-negligible probability. This result is not presented in their paper, and after having

spent some time contemplating this it appears to be a true statement which is by no means an obvious corollary to Theorem 8.2.

On a final note, in reference to the question posed in Chapter 3, regarding whether or not it is possible for a non-recursive function to be computed on a quantum Turing Machine, the answer is clearly no. It is obvious from the model's definition that it can be simulated by a deterministic Turing Machine. In fact, Bernstein and Vazirani prove that any BQP machine can be simulated by a deterministic Turing Machine using at most a polynomial number of tape cells, which puts the question to rest.

CHAPTER 9
Conclusion

In this paper we have attempted to show the potential promise that the quantum Turing Machine possesses. This is undoubtably done most adequately by Shor's result regarding the factoring of integers and Discrete Log. His results have multiple implications. If we suppose for a moment that Integer Factoring and Discrete Log are indeed not in P, then by showing that Integer Factoring and Discrete Log are computable in polynomial time on a quantum Turing Machine, Shor has effectively shown that $BQP \neq P$. Such a result is indeed likely as evidenced by Simon's result providing the oracle separation between the two classes. Let us contemplate the question of $P \stackrel{?}{=} BQP$ for a moment. While there is no definite answer, we do know that a positive answer is unlikely, as $P = BQP$ would also imply $P = BPP$, and the latter is a long standing open problem. Thus, there is relatively compelling evidence to suggest that the quantum Turing Machine is more powerful than its classical predecessors.

Unfortunately, current technology does not allow for the building of a quantum computer. There are significant physical obstacles which must be overcome before such a computer can be built. However, the construction of any such computer would change the field of computer science quite drastically. Perhaps the most immediate application of a quantum computer is in the area of cryptography. Given the existence of a quantum computer, the well known RSA cryptosystem would effectively become obsolete, as it is based on the conjecture that factoring is hard. Likewise,

any cryptosystem based on Discrete Log would as well be rendered obsolete, and it may be that the only solution left for cryptography would be quantum cryptosystems. These are as well a field of research today. [18]

Finally, I sincerely hope that this paper can be used as an introduction to the field by those who are interested and have no knowledge in the area.

Bibliography

[1] Richard P. Feynman, "Simulating physics with computers", *in International Journal of Theoretical Physics*, vol. 21, pp. 467–488, 1982.

[2] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer", *in Proc. R. Soc. Lond.*, vol. A400, pp. 73–90, 1985.

[3] Ethan Bernstein and Umesh Vazirani, "Quantum complexity theory", *in Proc. 25th ACM Symp. on the Theory of Computation*, pp. 11–20, 1993.

[4] Daniel R. Simon, "On the power of quantum computations", *in Proc. of 35th Annual Symp. on Foundations of Computer Science*. IEEE Press, November 1994.

[5] Ethan Bernstein Charles H. Bennett, Gilles Brassard and Umesh V. Vazirani, "Strengths and weaknesses of quantum computing", Manuscript, May 1994.

[6] Harry R. Lewis and Christos H. Papadimitriou, *Elements of the Theory of Computation*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[7] Christos H. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Company, 1994.

[8] Richard P. Feynman, *The Feynman Lectures on Physics*, Addison-Wesley Publishing Company, 1965.

[9] Peter W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring", *in Proc. of 35th Ann. Symp. on Foundations of Computer Science*. IEEE Press, November 1994.

[10] Charles H. Bennett, "Logical reversibiity of computation", *IBM Journal of Research and Development*, vol. 17, pp. 525–532, 1973.

[11] W. Keith Nicholson, *Abstract Algebra*, PWS-KENT, Boston, 1993.

[12] Neal Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, New York, 1994.

[13] Don Coppersmith, "An approximate fourier transform useful in quantum factoring", Technical report, IBM Research Division, Yorktown Heights, New York, July 1994.

[14] Wayne Eberly, "Personal communication", April 1995.

[15] Gary L. Miller, "Reimann's hypothesis and tests for primality", *Journal of Computer and System Sciences*, vol. 13, pp. 300–317, 1976.

[16] G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 1962.

[17] Charles E. Leiserson Thomas H. Cormen and Ronald L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.

[18] Gilles Brassard, "Cryptology column - quantum computing: The end of classical cryptography?", *Sigact News*, October 1994.