PROJECT REPORT

ON

Implementation of Shor's Algorithm in Factoring 21 Using Qubit Recycling Method and 51 Using Fermat Primes.

submitted by

Sauw Hritik Ramchandra
REG.NO:203102007

in partial fulfilment of the requirement
for the award of degree

MASTER OF SCIENCE

in

APPLIED MATHEMATICS AND COMPUTING

under the guidance of

Mr. Vinay Madhusudanan

Submitted to

Department Of Science
MIT Manipal

CERTIFICATE

This is to certify that the project entitled **Implementation of Shor's Algorithm in Factoring** 21 **Using Qubit Recycling Method and** 51 **Using Fermat Primes** is the work carried out by **Sauw Hritik Ramchandra** of M.Sc Applied Mathematics and Computing, Manipal Academy Of Higher Education, Manipal, during the year 2020-2022, in partial fulfillment of the requirements of the award of the degree of **Master of science** in **Applied Mathematics and Computing**.

Mr. Vinay Madhusudanan
Assistant Professor
Project Guide
Department of Mathematics
MIT Manipal

Dr. Sudhakara G
Professor and Head
Department of Mathematics
MIT Manipal

Place: Manipal
Date:

ACKNOWLEDGMENT

I take this opportunity with great pleasure to acknowledge the contribution of all those who were directly instrumental in leading to the development of this project to a phase in which it is visible to us today.
I wish to thank my project guide **Mr. Vinay Madhusudanan**, Assistant Professor,Department Of Mathematics, Manipal Academy Of Higher Education, for patiently helping me throughout the project.
I would like to express my gratitude to **Dr. Sudhakara G**, HOD, Department of Mathematics, for his support.

Sauw Hritik Ramchandra

Contents

Abstract

Shor's Algorithm is one of the algorithm which truly shows the supremacy of Quantum computers over its classical counterpart. In this project we show the speedup in factoring using Shor's Algorithm on a quantum computer. We construct a circuit to factor 21 using a compiled version of Shor's algorithm and successfully factor the number using very few resources compared to the standard approach. We also show another optimal method to factor a composite number 51 using Fermat prime. Both of these methods are one of the various methods which can be used to acheive factoring using very few resources on a Quantum Computer.

Chapter 1

Introduction to Linear Algebra and Quantum Mechanics

1.1 Linear Algbera

Linear algebra is the study of vector spaces and of linear operators on those vector spaces. A good understanding of quantum mechanics is based upon a solid grasp of elementary linear algebra.

The basic objects of linear algebra are *vector spaces*. The vector space of most interest to us is $\mathbb{C}^n$, the space of all n-tuples of complex numbers, $(z_1, z_2, \ldots, z_n)$. The elements of a vector space are called *vetors*, and we will sometimes use the column matrix notation

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} \tag{1.1}$$

to indicate a vector. In $\mathbb{C}^n$ the zero element is $(0, 0, \ldots, 0)$. A *vector subspace* of a vector space V is a subset W of V such that W is also a vector space, that is, W must be closed under scalar multiplication and addition.

The standard quantum mechanical notation for a vector in a vector space is the following

$$|\psi\rangle$$

$|\psi\rangle$ is a label for the vector. The $|\cdot\rangle$ notation is used to indicate that the object is a vector.

1.1.1 Bases and Linear Independence

A *spanning set* for a vector is a set of vectors $|v_1\rangle, |v_2\rangle, \ldots, |v_n\rangle$ such that any vector $|v\rangle$ in the vector space can be written as a linear combination $|v\rangle = \sum_i a_i |v_i\rangle$ of vectors in that set. For example, a spanning set for the vector space $\mathbb{C}^2$ is the set

$$|v_1\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix}; |v_2\rangle \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{1.2}$$

since any vector

$$|v\rangle \equiv \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{1.3}$$

in $\mathbb{C}^2$ can be written as a linear combination $|v\rangle = a_1 |v_1\rangle + a_2 |v_2\rangle$ of the vectors $|v_1\rangle$ and $|v_2\rangle$. We say that $|v_1\rangle$ and $|v_2\rangle$ *span* the vector space $\mathbb{C}^2$.

A set of non zero-vectors $|v_1\rangle, |v_2\rangle, \ldots, |v_n\rangle$ are *linearly dependent* if there exists a set of complex numbers $a_1, a_2, \ldots, a_n$ with $a_i \neq 0$ for at least one value of i, such that

$$a_1 |v_1\rangle + a_2 |v_2\rangle + \cdots + a_n |v_n\rangle = 0. \tag{1.4}$$

It can be shown that any two sets of linearly independent vectors which span a vector space V contain the same number of elements. We call such a set a *basis* for V. Furthermore, such a basis set always exists. The number of elements in the basis is defined to be the *dimension* of V.

1.1.2 Linear Operators and matrices

A *linear operator* between vector spaces V and W is defined to be any function $A\colon V \to W$ which is linear in its inputs,

$$A(\sum_i a_i |v_i\rangle) = \sum_i a_i A(|v_i\rangle) \tag{1.5}$$

When we say that a linear operator A is defined on a vector space, V, we mean that A is a linear opertaor from V to V. An important linear operator on any vector space V is the *identity operator*, I_v, defined by the equation $I_v |v\rangle \equiv |v\rangle$ for all vectors $|v\rangle$. The zero operator maps all vectors to the zero vector, $0 |v\rangle \equiv 0$.

Suppose V, W, and X are vector spaces, and $A\colon V \to W$ and $B\colon : W \to X$ are linear operatos. Then we use he notation BA to denote the composition of B with A, defined by $(BA)(|v\rangle) \equiv B(A(|v\rangle))$. Suppose $A\colon V \to W$ is a linear operator between vector spaces V and W. Suppose $|v_1\rangle, |v_2\rangle, \ldots, |v_m\rangle$ is a basis for V and

$|w_1\rangle, |w_2\rangle, \ldots, |w_n\rangle$ is a basis for W. Then for each j in the range $1, \ldots m$, there exist complex numbers A_{ij} through A_{nj} such that

$$A|v_j\rangle \equiv \sum_i A_{ij}|w_i\rangle. \tag{1.6}$$

The matrix whose entries are the values A_{ij} is said to form a *matix representation* of the opeartor A. Note that to make the connection between matrices and linear operators we must specify a set of input and output basis states for the input and output vector spaces of the linear operator.

1.1.3 Inner products

An *inner product* is a function which takes as input two vectors $|v\rangle$ and $|w\rangle$ from a vector space and produces a complex number as output. The standard quantum mechanical notation for the inner product $(|v\rangle, |w\rangle)$ is $\langle v|w\rangle$, where $|v\rangle$ and $|w\rangle$ are vectors in the inner product space, and the notation $\langle v|$ is used for the *dual vector* to the vector $|v\rangle$.

For example, $\mathbb{C}^n$ has an inner product defined by

$$((y_1, y_2, \ldots, y_n), (z_1, z_2, \ldots, z_n)) \equiv \sum_i y_i^* z_i = [y_1^*, \ldots, y_n^*] \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \tag{1.7}$$

We call a vector space equipped with an inner product an *inner product space*. In the finite dimensional complex vector spaces that come up in quantum computation and quantum information, a **Hilbert space** is exactly the same thing as an inner product.

Vectors $|w\rangle$ and $|v\rangle$ are *orthogonal* if their inner product is zero. We also say that $|v\rangle$ is *normalized* if $||\,|v\rangle\,|| = 1$. A set $|i\rangle$ of vectors with index i is *orthonormal* if each vector is a unit vector, and distinct vectors in the set are orthogonal. The inner product of two vectors is equal to the vector inner product between two matrix representation of those vectors, provided the representation are written with respect to the same orthonormal basis.

$$\langle v|w\rangle = (\sum_i v_i|i\rangle, \sum_j w_j|j\rangle) = \sum_{ij} v_i^* w_j \delta_{ij} = \sum_i v_i^* w_i = [v_1^* \ldots v_n^*] \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \tag{1.8}$$

The *outer product reprsentation* is defined as, suppose $|v\rangle$ is a vector in an inner product space V, and $|w\rangle$ is a vector in an inner product space W. Define $|w\rangle\langle v|$ to be the linear operator form V to W whose action is defined by

$$\langle v|w\rangle = (\sum_i v_i |i\rangle, \sum_j w_j |j\rangle) = \sum_{ij} v_i^* w_j \delta_{ij} = \sum_i v_i^* w_i = [v_1^* \dots v_n^*] \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \tag{1.9}$$

Let $|i\rangle$ be any orthonormal basis for the vector space V, so an arbitrary vector $|v\rangle$ can be written $|v\rangle = \sum_i v_i |i\rangle$ for some set of complex numbers v_i. Note that $\langle 1|v\rangle = v_i$ and therefore

$$(\sum_i |i\rangle\langle i|)|v\rangle = \sum_i |i\rangle\langle i|v\rangle = \sum_i v_i |i\rangle = |v\rangle \tag{1.10}$$

Since the last equation is true for all $|v\rangle$ it follows that

$$\sum_i |i\rangle\langle i| = I \tag{1.11}$$

This equation is known as the *completeness relation.*

1.1.4 Eigenvectors and Eigenvalues

An *eigenvector* of a linear operator A on a vector space is a non-zero vector $|v\rangle$ such that $A|v\rangle \equiv v|v\rangle$, where v is a complex number known as the *eigenvalue* of A corresponding to $|v\rangle$. The *characteristic function* is defined to be $c(\lambda) \equiv det|A - \lambda I|$, it can be shown that the characteristic function depends only upon the operator A, and not on the specific matrix representation used for A. The solutions of the characteristic equation $c(\lambda) = 0$ are the eigenvalues of the operator A. By the fundamental theorem of algebra, every polynomial has at least one complex roots, so every operator A has at least one eigenvalue, and a corresponding eigenvector. The *eigenspace* corresponding to an eigenvalue v is the set of vectors which have eigenvalue v.

A *diagonal representation* for an operator A on a vector space V is a representation $A = \sum_i \lambda_i |i\rangle\langle i|$, where the vectors $|i\rangle$ form an orthonormal set of eigenvalues for A, with corresponding eigenvalues λ_i. An operator is said to be *diagonalizable* if it has a diagonal representation.

Diagonal representations are sometimes also known as *orthonormal decompositions.* When an eigenvalue is more than one dimensional we say that it is *degenerate.* For example, the matrix A defined by

$$A \equiv \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{1.12}$$

has a two-dimensional eigenspace corresponding to the eigenvalue 2. The eigenvectors $(1,0,0)$ and $(0,1,0)$ are said to be degenerate because they are linearly independent eigenvectors of A with the same eigenvalue.

1.1.5 Adjoints and Hermitian operators

Suppose A is any linear operator on a Hilbert space , V. It turns out that there exists a unique linear operator $A^\dagger$ on V such that for all vectors $|v\rangle, |w\rangle \in V$,

$$(|v\rangle, A|w\rangle) = (A^\dagger |v\rangle, |w\rangle) \tag{1.13}$$

This linear operator is known as the *adjoint* or *Hermitian conjugate* of the operator A. From the definition it is easy to see that $(AB)^\dagger = B^\dagger A^\dagger$. By convention, if $|v\rangle$ is a vector, then we define $|v\rangle^\dagger \equiv \langle v|$. With this definition it is not difficult to see that $(A|v\rangle)^\dagger = \langle v| A^\dagger$.

An operator A whose adjoint is A is known as a *Hermitian* or *self-adjoint* operator. An important class of Hermitian operators is the *projectors*. Suppose W is a k-dimensional vector subspace of the d-dimensional vector space V. Using the Gram-Schmidt procedure it is possible to construct an orthonormal basis $|1\rangle, ..., |d\rangle$ for V such that $|1\rangle, \ldots, |k\rangle$ is an orthonormal basis for W. By definition,

$$P \equiv \sum_{i=1}^{k} |i\rangle \langle i| \tag{1.14}$$

is the projector onto the subspace W. From the definition it can be shown that $|v\rangle \langle v|$ is Hermitian for any vector $|v\rangle$, so P is Hermitian, $P^\dagger = P$. The *orthogonal complement* of P is the operator $Q \equiv I - P$. It is easy to see that Q is a projector onto the vector space spanned by $|k+1\rangle, \ldots, |d\rangle$, which we also refer to as the orthogonal complement of P, and may denote by Q.

An operator A is said to be *normal* if $AA^\dagger = A^\dagger A$. Clearly, an operator which is Hermitian is also normal. There is a remarkable representation theorem for normal operators known as the *spectral decomposition*, which states that an operator is a normal operator if and only if it is diagonalizable.

A matrix U is said to be *unitary* if $U^\dagger U = I = UU^\dagger$. It is easily checked that an operator is unitary if and only if each of its matrix representations is unitary. A unitary operator also satisfies $UU^\dagger = I$, and therefore U is normal and has a spectral decomposition. Geometrically, unitary operators are important because they preserve inner products between vectors. Then the inner products of $U|v\rangle$ and $U|w\rangle$ is the same as the inner product of $|v\rangle$ and $|w\rangle$,

$$(U|v\rangle, U|w\rangle) = \langle v|U^\dagger U|w\rangle = \langle v|I|w\rangle = \langle v|w\rangle . \tag{1.15}$$

A *positive operator* A is defined to be an operator such that for any vector $|v\rangle$, $(|v\rangle, A|v\rangle)$ is a real, non-negative number. If $|v\rangle$, $(|v\rangle, A|v\rangle)$ is trictly greater than zero for all $|v\rangle \neq 0$ then we say that A is *positive definite*.

1.1.6 Tensor products and Trace

The *tensor product* is a way of putting vector spaces together to form larger vector spaces. Suppose V and W are vector spaces of dimension m and n respectively. For convenience we also suppose that V and W are Hilbert spaces. The $V \otimes W$ is and mn dimensional vector space. The elements of $V \otimes W$ are linear combinations of 'tensor products' $|v\rangle \otimes |w\rangle$ of elements $|v\rangle$ of V and $|w\rangle$ of W. In particular, if $|i\rangle$ and $|j\rangle$ are orthonormal bases for the spaces V and W then $|i\rangle \otimes |j\rangle$ is a basis for $V \otimes W$.

By definition the tensor product satisfies the following basic properties:

1. For an arbitrary scalar z and elements $|v\rangle$ of V and $|w\rangle$ of W.

$$z(|v\rangle \otimes |w\rangle) = (z\,|v\rangle) \otimes |w\rangle = |v\rangle \otimes (z\,|w\rangle) \tag{1.16}$$

2. For arbitrary $|v_1\rangle$ and $|v_2\rangle$ in V and $|w\rangle$ in W,

$$(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle \tag{1.17}$$

3. For arbitrary $|v\rangle$ in V and $|w_1\rangle$ and $|w_2\rangle$ in W,

$$|v\rangle \otimes (|w_1\rangle + |w_2\rangle) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle \tag{1.18}$$

Suppose $|v\rangle$ and $|w\rangle$ are vectors in V and W, and A and B are linear operators on V and W, respectively. Then we can define a linear operator $A \otimes B$ on $V \otimes W$ by the equation

$$(A \otimes B)(|v\rangle \otimes |w\rangle) \equiv A\,|v\rangle \otimes B\,|w\rangle \tag{1.19}$$

The definition of $A \otimes B$ is then extended to all elements of $V \otimes W$ in the natural way to ensure linearity of $A \otimes B$, that is,

$$(A \otimes B)(\sum_i a_i\,|v_i\rangle) \equiv \sum_i a_i A\,|v_i\rangle \otimes B\,|w_i\rangle \tag{1.20}$$

It can be shown that $A \otimes B$ defined in this way is a well-defined linear operator on $V \otimes W$. This notion of the tensor product of two operators extends in the obvious way to the case where $A\colon V \to V'$ and $B\colon W \to W'$ map between different vector spaces. Indeed, an arbitrary linear operator C manpping $V \otimes W$ to $V' \otimes W'$,

$$C = \sum_i c_i A_i \otimes B_i, \tag{1.21}$$

where by definition

$$(\sum_i c_i A_i \otimes B_i)\,|v\rangle \otimes |w\rangle \equiv \sum_i c_i A_i\,|v\rangle \otimes B_i\,|w\rangle \tag{1.22}$$

The inner products on the space V and W can be used to define a natural inner product on $V \otimes W$. Define

$$\left(\sum_i a_i |v_i\rangle \otimes |w_i\rangle , \sum_i b_j |v_j'\rangle \otimes |w_j'\rangle \right) \equiv \sum_{ij} a_i^* b_j \langle v_i|v_j'\rangle \langle w_i|w_j'\rangle \tag{1.23}$$

The discussion above can be made much more concrete by moving to a convenient matrix representation known as the *Kronecker product*. Suppose A is an $m \times n$ matrix, and B is a $p \times q$ matrix. Then we have the matrix representation:

$$A \otimes B \equiv \begin{bmatrix} A_{11}B & A_{12}B & \ldots & A_{1n}B \\ A_{21}B & A_{22}B & \ldots & A_{2n}B \\ A_{m1}B & A_{m2}B & \ldots & A_{mn}B \end{bmatrix} \tag{1.24}$$

For example,

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \otimes \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \times 2 \\ 1 \times 3 \\ 2 \times 2 \\ 2 \times 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 6 \end{bmatrix} \tag{1.25}$$

The *trace of a matrix* A is defined to be the sum of its diagonal elements,

$$tr(A) \equiv \sum_i A_{ii} \tag{1.26}$$

The trace is easily seen to be *cyclic*, $tr(AB) = tr(BA)$, and linear, $tr(A + B) = tr(A) + tr(B), tr(zA) = ztr(A)$, where A and B are matrices, and z is a complex number. Furthermore, from the cyclic property it follows that the trace of a matrix is invariant under the *unitary similarity transformation* $A \rightarrow UAU^\dagger$, as $tr(UAU^\dagger) = tr(U^\dagger UA) = tr(A)$. In the light of this result, it makes sense to define the trace of an operator A to be the trace of any matrix representation of A.

Suppose $|\psi\rangle$ is a unit vector and A is an arbitrary operator. To evaluate $tr(A |\psi\rangle \langle\psi|)$ use the Gram-schmidt procedure to extend $|\psi\rangle$ to an orthonormal basis $|i\rangle$ which includes $|\psi\rangle$ as the first element. Then we have

$$tr(A |\psi\rangle \langle\psi|) = \sum_i \langle i|A|\psi\rangle \langle\psi|i\rangle = \langle\psi|A|\psi\rangle \tag{1.27}$$

1.2 Quantum Mechanics

Quantum mechanics is the most accurate and complete description of the world known. It is also the basis for an understanding of quantum computation and quantum information. Quantum mechanics is a mathematical framework for the development of physical theories. On its own quantum mechanics doesn't tell you what laws a physical system must obey, but it does provide a mathematical and conceptual framework for the development of such laws. The postulates of quantum mechanics were derived after a long process of trail and (mostly) error, which involved a considerable amount of guessing and fumbling by the originators of the theory.

1.2.1 State space

The first postulate of quantum mechanics sets up the arena in which quantum mechanics takes place, Hilbert space.

Postulate 1. *Associated to any isolated physical system is a complex vector space with the inner product (that is, a Hilbert space) known as the* state space *of the system. The system is completely described by its* state vector*, which is a unit vector in the system's state space.*

A **Qubit** has a two-dimensional state space. Suppose $|0\rangle$ and $|1\rangle$ form an orthonormal basis for that state space. Then an arbitrary state vector in the state space can be written

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,, \tag{1.28}$$

where a and b are complex numbers. The condition that $|\psi\rangle$ be a unit vector, $\langle\psi|\psi\rangle = 1$ is therefore equivalent to $|a|^2 + |b|^2 = 1$. The condition $\langle\psi|\psi\rangle = 1$ is often known as the *normalization condition* for state vectors.

We say that any linear combination $\sum_i \alpha_i\,|\psi_i\rangle$ is a superposition of the states $|\psi_i\rangle$ with the amplitude α_i for the state $|\psi_i\rangle$. So, for example, the state

$$\frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{1.29}$$

is a superposition of the states $|0\rangle$ and $|1\rangle$ with the amplitude $1/\sqrt{2}$ for the state $|0\rangle$, and amplitude $-1/\sqrt{2}$ for the state $|1\rangle$.

1.2.2 Evolution

How does the state, $|\psi\rangle$, of a quantum mechanical system change with time? The following postulate gives a prescription for the description of such state changes.

Postulate 2. *The evolution of a* closed *quantum system is described by a* unitary transformation*. That is, the state $|\psi\rangle$ of the system at time t_1 is related to the state*

$|\psi'\rangle$ of the system at the time t_2 by a unitary operator U which depends only on times t_1 and t_2,

$$|\psi'\rangle = U|\psi\rangle . \tag{1.30}$$

What unitary operators are natural to consider? In the case of single qubits, it turns out that any unitary operator at all can be realized in realistic systems.

The X and Z Pauli matrices are also sometimes referred to as the *bit flip* and *phase flip* matrices: the X matrix takes $|0\rangle$ to $|1\rangle$, and $|1\rangle$ to $|0\rangle$; and the Z matrix leaves $|0\rangle$ invariant, and takes $|1\rangle$ to $-|1\rangle$, with the extra factor of -1 added known as a *phase factor*. Another interesting unitary operator is the *Hadamard gate*, which we denote H. This has the action $H|0\rangle \equiv (|0\rangle + |1\rangle)/\sqrt{2}$, $H|1\rangle \equiv (|0\rangle - |1\rangle)/\sqrt{2}$ and the corresponding matrix representation

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{1.31}$$

Postulate 2 describes how the quantum states of a closed quantum system at two different times are related.

1.2.3 Quantum measurement

We introduce Postulate 3, which provides a means for describing the effects of measurements on quantum systems.

Postulate 3. *Quantum measurements are described by a collection $\{M_m\}$ of* measurement operator*. These are operators acting on the state space of the sustem being measured. the index m refers to the measurement outcomes that may occur in the experiment. If the state of the quantum system is $|\psi\rangle$ immediately before the measurement then the probability that result m occurs is given by*

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle \tag{1.32}$$

and the state of the system after the measurement is

$$\frac{M_m\psi}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}} \tag{1.33}$$

The measurement operators satisfy the *completeness equation*,

$$\sum_m M_m^\dagger M_m = I \tag{1.34}$$

The completeness equation expresses the fact that probabilities sum to one:

$$1 = \sum_m p(m) = \sum_m \langle\psi|M_m^\dagger M_m|\psi\rangle \tag{1.35}$$

A simple but important example of a measurement is the *measurement of a qubit in the computational basis*. This is a measurement on a single qubit with two outcomes defined by the two meausrement operators,= $M_o = |0\rangle\langle 0|$, $M_1 = |1\rangle\langle 1|$. Observe that each measurement operator is Hermitian, and that $M_o^2 = M_o$, $M_1^2 = M_1$. Thus the completeness relation is obeyed, $I = M_o^\dagger M_o + M_1^\dagger M_1 = M_o + M_1$. Suppose the state being measured is $|\psi\rangle = a|0\rangle + b|1\rangle$. Then the probability of obtaining measurement outcome 0 is

$$p(0) = \langle\psi|M_0^\dagger M_0|\psi\rangle = \langle\psi|M_0|\psi\rangle = |a|^2 \tag{1.36}$$

Similarly, the probability of obtaining the measurement outcome 1 is $p(1) = |b|^2$.

1.2.4 Phase

'Phase' is a commonly used term in quantum mechanics, with several different meanings dependent upon context. Consider, for example, the state $e^{i\theta}|\psi\rangle$, where $|\psi\rangle$ is a state vector, and θ is a real number. We say that the state $e^{i\theta}|\psi\rangle$ is equal to $|\psi\rangle$, up to the *global phase factor* $e^{i\theta}$. It is interesting to note that the *statistics of measurement* predicted for these two states are the same. Therefore, from an observational point of view these two states are identical. For this reason we may ignore global phase factors as being irrelevant to the observed properties of the physical system.

There is another kind of phase known as the *relative phase*, which has quite a different meaning. Consider the states,

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad \textit{and} \quad \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{1.37}$$

In the first case the amplitude of $|1\rangle$ is $1/\sqrt{2}$. For the second state the amplitude is $-1/\sqrt{2}$. In each case the *magnitude* of the amplitudes is the same, but they differ in sign. More generally, we say that two amplitudes, a and b, *differ by a relative phase* if there is a real θ such that $a = exp(i\theta)b$. More generally still, two states are said to *differ by a relative phase* in some basis if each of the amplitudes in that basis is related by such a phase factor.

The difference between relative phase factors and global phase factors is that for relative phase the phase factors may vary from amplitude to amplitude. This makes the relative phase a basis-dependent concept like global phase.

1.2.5 Composite systems

The following postulate describes how the state space of a composite system is built up from the state spaces of the component systems.

Postulate 4. *The state space of a composite physical system is the tensor product of the state space of the component physical systems. Moreover, if we have systems numbered 1 through n, and system number* i *is prepared in the state* $|\psi_i\rangle$*, then the joint state of the total system is* $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \ldots \otimes |\psi_n\rangle$

The *superposition principle of quantum mechanics* states, that if $|x\rangle$ and $|y\rangle$ are two states of a quantum system, then any superposition $\alpha |x\rangle + \beta |y\rangle$ should also be an allowed state of a quantum system, where $|\alpha|^2 + |\beta|^2 = 1$.

Postulate 4 also enables us to define one of the most interesting and puzzling ideas associated with composite quantum systems - *entanglement.* Consider the two qubit state

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{1.38}$$

This state has the remarkable property that there are no single qubit states $|a\rangle$ and $|b\rangle$ such that $|\psi\rangle = |a\rangle |b\rangle$. We say that a state of a composite system having this property is an *entangled state.*

Chapter 2

Introduction to Quantum Computation

2.1 Introduction to Computer Science

The modern incarnation of computer science was announced by the great mathematician Alan Turing in a remarkable 1936 paper. Turig showed that there is a *Universal Turing Machine* that can be used to simulate any other Turing machine. Furthermore, he claimed that the Universal Turing Machine completely captures what it means to perform a task by algorithmic means. That is, if an algorithm can be performed on any piece of hardware, then there is an equivalent algorithm for a Universal Turing Machine which performs exactly the same task as the algorithm running on the personal computer. This assertion, known as the *Church-Turing thesis* in honor of Turing and Alonxo Church, asserts the equivalence between the physical concept of what class of algorithms can be performed on *some physical device* with the rigorous mathematical concept of a Universal Turing Machine.

John von Neumann developed a simple theoretical model for how to put together in a practical fashion all the components necessary for a computer to be fully as capable as a Universal Turing Machine. Hardware development truly took off, though, in 1947, when John Bardeen, Walter Brattain and Will Shockley developed the transistor. Computer hardware has grown in power at an amazing pace ever since, so much so that the growth was codified by Gordon Moore in 1965 in what has come to be known as *Moore's Law*, which states that computer power will double for constant cost roughly once every two years. Nevertheless, most observers expect that this dream run will end some time during the first two decades of the 21^{st} century. Conventional aprroaches to the fabrication of computer technology are beginning to run up against fundamental difficulties of size. Quantum effects are beginning to interfere in the functioning of electronic devices as they are made smaller and smaller.

One possible solution to the problem posed by the eventual failure of Moore's law is to move to a different computing paradigm. One such paradigm is provided by

the theory of quantum computation, which is based on the idea of using quantum mechanics to perform computations, instead of classical physics.

2.1.1 Analog Computation and Randomized Algorithm

What do we mean by 'efficient' versus 'inefficient' simulations of a quantum computer? Roughly speaking, an efficient algorithm is one which runs in time polynomial in the size of the problem solved. In contrast, an inefficent algorithm requires super-polynomial time. Around 1960s a strengthened version of the Church-Turing thesis was developed:

Any algorithmic process can be simulated efficiently using a Turing machine.

One class of challenges to the strong Church-Turing thesis comes from the field of *analog computation.* Unfortunately for analog computation, it turns out that when realistic assumptions about the presence of noise in analog computation are made, their power disappears in all known instances; they cannot efficiently solve problems which are not efficiently solvable on a Turing machine. This lesson - that the effects of realistic noise must be taken into account in evaluating the efficiency of a computational model - was one of the great early challenges of quantum computation and quantum information, a challenge successfully met by the development of a theory of *quantum error-correcting codes* and *fault-tolerant quantum computation.* Thus, unlike analog computation, quantum computation can in principle tolerate a finite amount of noise and still retain its computational advantages.

The first major challenge to the strong Church-Turing thesis arose in the mid 1970s when Robert Solovay and Volker Strassen showed that it is possible to test whether an integer is prime or composite using a *randomized algorithm.* The algorithm did not determine whether a given integer was prime or composite with certainty. Instead, the algorithm could determine that a number was *probably* prime or else composite *with certainty.* This was solved by a simple modification of the strong Church-Turing thesis;

Any algorithmic process can be simulated efficiently using a probabilistic Turing machine.

In 1985 David Deutsch asked whether the laws of physics could be use to derive an even stronger version of the Church-Turing thesis. In particular, Deutsch attempted to define a computational device that would be capable of efficiently simulating an *arbitrary* physical system. Because the laws of physics are ultimately quantum mechanical, Deutsch was naturally led to consider computing devices based upon the principles of quantum mechanics.

What Deutsch's model of a quantum computer did enable was a challenge to the strong form of the Church–Turing thesis. Deutsch asked whether it is possible for

a quantum computer to efficiently solve computational problems which have no efficient solution on a classical computer, even a probabilistic Turing machine. This remarkable first step taken by Deutsch was improved in the subsequent decade by many people, culminating in Peter Shor's 1994 demonstration that two enormously important problems – the problem of finding the prime factors of an integer, and the so-called 'discrete logarithm' problem – could be solved efficiently on a quantum computer. Shor's results are a powerful indication that quantum computers are more powerful than Turing machines, even probabilistic Turing machines. Further evidence for the power of quantum computers came in 1995 when Lov Grover showed that another important problem – the problem of conducting a search through some unstructured search space – could also be sped up on a quantum computer

2.2 Need of Quantum Computers

Quantum Computation and quanutm information is the study of the information processing tasks that can be accomplished using quantum mechanical systems.

Small quantum computers, capable of doing dozens of operations on a few quantum bits (or qubits) represent the state of the art in quantum computation. Experimental prototypes for doing quantum *quantum cryptography* - a way of communicating in secret across long distances - have been demonstrated, and are even at the level where they may be useful for some real-world applications. It is likely that one of the major applications of quantum computers in the future will be performing simulations of quantum mechanical systems too difficult to simulate on a classical computer, a problem with profound scientific and technological implications.

Algorithm design for quantum computers is hard because designers face two difficult problems not faced in the construction of algorithms for classical computers. First, our human intuition is rooted in the classical world. If we use that intuition as an aid to the construction of algorithms, then the algorithmic ideas we come up with will be classical ideas. Second, to be truly interesting it is not enough to design an algorithm that is merely quantum mechanical. The algorithm must be better than any existing classical algorithm!

At the same time computer science was exploding in the 1940s, another revolution was taking place in our understanding of communication. In 1948 Claude Shannon published a remarkable pair of papers laying the foundations for the modern theory of information and communication. Perhaps the key step taken by Shannon was to *mathematically define the concept of information.* Shannon was interested in two key questions related to the communication of information over a communications channel. First, what resources are required to send information over a communications channel? Second, can information be transmitted in such a way that it is protected against noise in the communications channel?

Shannon answered these two questions by proving the two fundamental theorems of information theory. The first, Shannon's *noiseless channel coding theorem*, quan-

tifies the physical resources required to store the output from an information source. Shannon's second fundamental theorem, the *noisy channel coding theorem*, quantifies how much information it is possible to reliably transmit through a noisy communications channel. To achieve reliable transmission in the presence of noise, Shannon showed that error-correcting codes could be used to protect the information being sent.

Quantum information theory has followed with similar developments. In 1995, Ben Schumacher provided an analogue to Shannon's noiseless coding theorem, and in the process defined the 'quantum bit' or 'qubit' as a tangible physical resource.

2.2.1 Cryptography

Cryptography is the problem of doing *communication* or *computation* involving two or more parties *who may not trust one another*. The best known cryptographic problem is the transmission of secret messages. The most important distinction in cryptosystem is between *private key cryptosystems* and *public key cryptosystems*.

The way a private key cryptosystem works is that two parties, 'Alice' and 'Bob', wish to communicate by sharing a *private key*, which only they know. The point is that this key is used by Alice to encrypt the information she wishes to send to Bob. After Alice encrypts she sends the encrypted information to Bob, who must now recover the original information. Exactly how Alice encrypts the message *depends upon the private key*, so that to recover the original message Bob needs to know the private key, in order to undo the transformation Alice applied. Unfortunately, private key cryptosystems have some severe problems in many contexts. The most basic problem is how to distribute the keys? In many ways, the key distribution problem is just as difficult as the original problem of communicating in private – a malevolent third party may be eavesdropping on the key distribution, and then use the intercepted key to decrypt some of the message transmission.

One of the earliest discoveries in quantum computation and quantum information was that quantum mechanics can be used to do key distribution in such a way that Alice and Bob's security can not be compromised. This procedure is known as *quantum cryptography* or *quantum key distribution*. The basic idea is to exploit the quantum mechanical principle that observation in general disturbs the system being observed. Thus, if there is an eavesdropper listening in as Alice and Bob attempt to transmit their key, the presence of the eavesdropper will be visible as a disturbance of the communications channel Alice and Bob are using to establish the key. Alice and Bob can then throw out the key bits established while the eavesdropper was listening in, and start over.

The second major type of cryptosystem is the *public key cryptosystem*. Public key cryptosystems don't rely on Alice and Bob sharing a secret key in advance. Instead, Bob simply publishes a 'public key', *which is made available to the general public*. Alice can make use of this public key to encrypt a message which she sends to Bob.

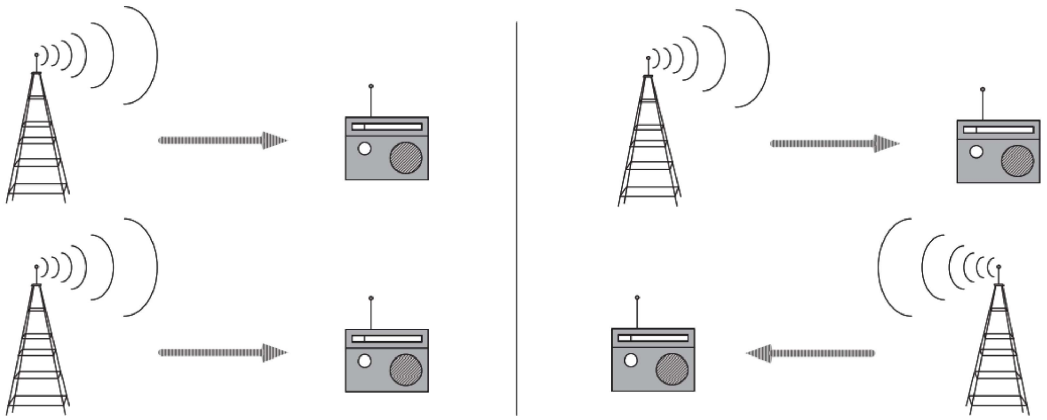


Figure 2.1: Classically, if we have two very noisy channels of zero capacity running side by side, then the combined channel has zero capacity to send information. If we reverse the direction of one of the channels, we still have zero capacity to send information. Quantum mechanically, reversing one of the zero capacity channels can actually allow us to send information!

What is interesting is that a third party cannot use Bob's public key to decrypt the message! Strictly speaking, we shouldn't say cannot. Rather, the encryption transformation is chosen in a very clever and non-trivial way so that it is extremely difficult (though not impossible) to invert, given only knowledge of the public key. To make inversion easy, Bob has a *secret key* matched to his public key, which together enable him to easily perform the decryption. This secret key is not known to anybody other than Bob, who can therefore be confident that only he can read the contents of Alice's transmission, to the extent that it is unlikely that anybody else has the computational power to invert the encryption, given only the public key. Public key cryptosystems solve the key distribution problem by making it unnecessary for Alice and Bob to share a private key before communicating.

Rather remarkably, public key cryptography did not achieve widespread use until the mid-1970s, when it was proposed independently by Whitfield Diffie and Martin Hellman, and by Ralph Merkle, revolutionizing the field of cryptography. A little later, Ronald Rivest, Adi Shamir, and Leonard Adleman developed the RSA cryptosystem, which at the time of writing is the most widely deployed public key cryptosystem, believed to offer a fine balance of security and practical usability.

The key to the security of public key cryptosystems is that it should be difficult to invert the encryption stage if only the public key is available. For example, it turns out that inverting the encryption stage of RSA is a problem closely related to factoring. Much of the presumed security of RSA comes from the belief that factoring is a problem hard to solve on a classical computer. However, Shor's fast algorithm for factoring on a quantum computer could be used to break RSA! This practical

application of quantum computers to the breaking of cryptographic codes has excited much of the interest in quantum computation and quantum information.

2.2.2 Quantum Tunneling

As per Moore's law, the number of transistors must double every year. A transistor is just an electric switch which can allow or block the passage of electrons moving from moving in one direction. Today a typical scale for transistor is 4nm. As transistors are shrinking to the size of only a few atoms, electrons may just transfer themselves to the other side of a blocked passage via a process called *Quantum Tunneling* and thereby making classical computers useless.

(a) Transistor as a closed switch

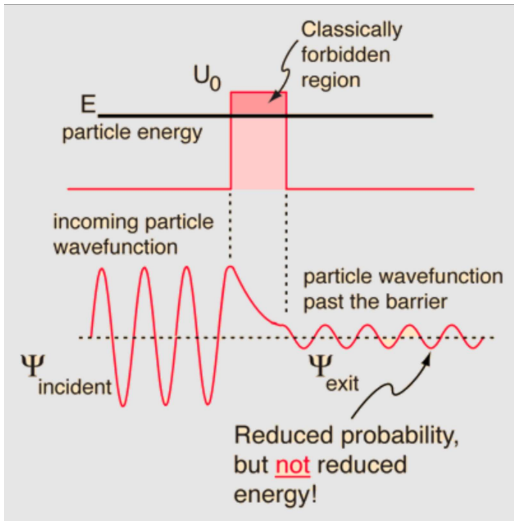
(b) Transistor as an open switch

Figure 2.2: Transistor as an electric switch with electrons passing through it.

In the quantum realm, physics work quite differently from the predictable ways we're used to, and traditional computers stop making sense. Quantum Tunneling is responsible for many important phenomenon such as the alpha decay of atomic nuclei and the operation of certain types of diodes. Yet, quantum tunneling is completely contrary to our intuition about how the universe is supposed to work. This is because quantum tunneling has absolutely no analogy in classical physics. To understand quantum tunneling, we must understand that particles do not have a defined position until they are observed. Instead all particles are observed by what we call a *'wave function'*. The probability of a particle being observed at a particular location is given by the square of the amplitude of the wave function at that location.

Suppose that the particle bounces off a barrier, where the energy of the barrier is greater than the energy of the particle. This is represented by the wave function reflecting at the boundary.Let's consider the behaviour of wave function inside the barrier. As the distance into the barrier increases, the amplitude of the wave function decreases exponentially. But the wave function does not actually reach an amplitude of zero. But if the barrier is shorter in length, the amplitude of the wave function will still decay inside the barrier, but since the wave function does not reach an amplitude of zero, the wave function can exit the barrier on the other

side. Once the wave function exits the barrier, its amplitude no longer decays. A portion of the wave function passes through each of the two boundaries and a portion of the wave function also reflects at each of the two boundaries. This means that there is a certain probability that the particle will pass through the barrier to the other side, and a certain probability that the particle will bounce off the barrier. If we send a number of these particles towards the barrier, some of them will pass through, and some of them will bounce off. Now suppose the barrier length is even shorter than above scenario. In this case, the wave function does not have as much distance to decay inside the barrier, and we therefore have a larger amplitude for the portion of the wave function that exits the barrier. This means with this shorter barrier, the particle has a greater probability of passing through. It also means that the particle has a lower probability of bouncing off the barrier, which is represented by a smaller amplitude for the reflected wave.



As we increase the length of the barrier, the particle is less likley to pass through, and more likely to bounce off. But even if the probability of each individual particle passing through a barrier is small, if we have very large number of particles, we can have a large probability that at least some of them will pass through. From the perspective of classical physics, if the energy of the barrier is greater than the energy of the incoming particles, then there is no possibility that any of the particles will make it past the boundary. But from the perspective of quantum mechanics, it is a very different world, and hence what we call *quantum tunneling*.

The phenomenon of quantum tunnelling can be explained through the macroscopic analogy of a rock rolling up a steep hill. In this scenario, we know that the ball can only go over the hill if its kinetic energy is higher than the gravitational potential of the hill itself. Otherwise, the ball will simply roll back down the hill. In order to pass over the hill, the ball will either need to surpass the gravitational potential of the hill through additional energy (e.g. an individual rolling the ball up the hill), or use quantum tunnelling to simply pass through the barrier as a wave.

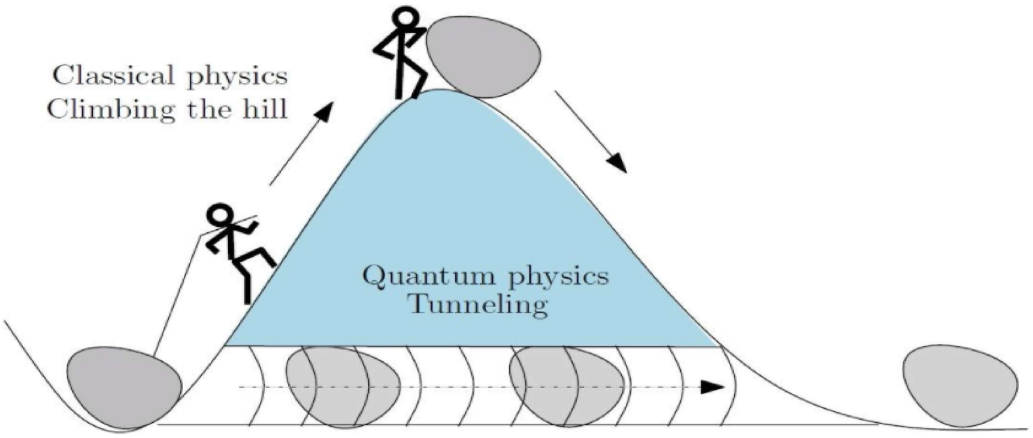


Figure 2.3: Quantum tunnelling explained through rock rolling up a steep hill

2.3 Quantum Computation

2.3.1 Quantum bits

The bit is the fundamental concept of classical computation and classical information.

What then is a qubit? Just as a classical bit has a state – either 0 or 1 – a qubit also has a state. Two possible states for a qubit are the states $|0\rangle$ and $|1\rangle$, which correspond to the states 0 and 1 for a classical bit. The difference between bits and qubits is that a qubit can be in a state other than $|0\rangle$ or $|1\rangle$. It is also possible to form *linear combination* of states, often called *superpositions*:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

The numbers α and β are complex numbers, although for many purposes not much is lost by thinking of them as real numbers. Put another way, the state of a qubit is a vector in a two-dimensional complex vector space. The special states $|0\rangle$ and $|1\rangle$ are known as *computational basis states*, and form an orthonormal basis for this vector space.

When we measure a qubit we get either the result 0, with probability $|\alpha|^2$, or the result 1, with probability $|\beta|^2$. Naturally, $|\alpha|^2 + |\beta|^2 = 1$, since the probabilities must sum to one. Geometrically, we can interpret this as the condition that the qubit's state be normalized to length 1. Thus, in general a qubit's state is a unit vector in a two-dimensional complex vector space.

By contrast, a qubit can exist in a continuum of states between $|0\rangle$ and $|1\rangle$ - until it is observed. For example, a qubit can be in the state

$$\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle \tag{2.2}$$

which, when measured, gives the result 0 fifty percent $\left|\frac{1}{\sqrt{2}}\right|^2$, and the result 1 fifty percent of the time. The above state is denoted by $|+\rangle$.

2.3.2 Bloch Sphere

A qubit can be realised as the two different polarizations of a photon; as the alignment of a nuclear spin in a uniform magnetic field; as two states of an electron orbiting a single atom. In the atom model, the electron can exist in either the so-called 'ground' or 'excited' states, which we'll call $|0\rangle$ and $|1\rangle$, respectively. By shining light on the atom, with appropriate energy and for an appropriate length of time, it is possible to move the electron from the $|0\rangle$ state to the $|1\rangle$ state and vice versa. But more interestingly, by reducing the time we shine the light, an electron initially in the state $|0\rangle$ can be moved 'halfway' between $|0\rangle$ and $|1\rangle$, into the $|+\rangle$ state.

Because $|\alpha|^2 + |\beta|^2 = 1$, we may write $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ as

$$|\psi\rangle = e^{i\gamma}(cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}sin\frac{\theta}{2}\,|1\rangle) \tag{2.3}$$

where θ, φ, ψ are real numbers. We can ignore the factor of $e^{i\gamma}$ out the front, because

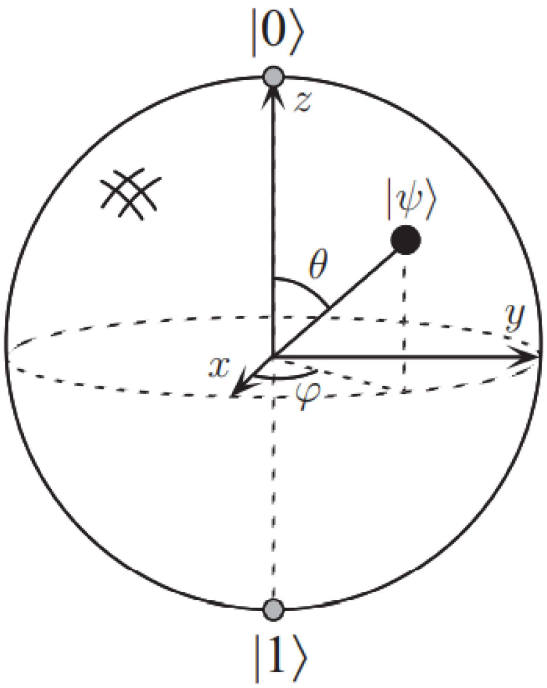


Figure 2.4: Bloch sphere representation of a qubit

it has no *observable effects*, and for that reason we can effectively write

$$|\psi\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\phi}sin\frac{\theta}{2}|1\rangle \tag{2.4}$$

However, it must be kept in mind that this intuition is limited because there is no simple generalization of the Bloch sphere known for multiple qubits.

2.3.3 Measurement of a qubit

How much information is represented by a qubit? Paradoxically, there are an infinite number of points on the unit sphere, so that in principle one could store an entire text of Shakespeare in the infinite binary expansion of θ. However, this conclusion turns out to be misleading, because of the behavior of a qubit when observed. Measurement of a qubit will only give either 0 or 1. Furthermore, measurement *changes* the state of a qubit, collapsing it from its superposition of $|0\rangle$ and $|1\rangle$ to the specific state consistent with the measurement result. For example, if measurement of $|+\rangle$ gives 0, then the post-measurement state of the qubit will be $|0\rangle$. This behavior is simply one of the *fundamental postulates* of quantum mechanics. From a single measurement one obtains only a single bit of information about the state of the qubit, thus resolving the apparent paradox. It turns out that only if infinitely many identically prepared qubits were measured, one would be able to determine α and β for a qubit in the state

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle . \tag{2.5}$$

2.3.4 Two qubit system and Bell state

A two qubit system has four *computational basis states* denoted $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. A pair of qubits can also exist in superpositions of these four states, so the quantum state of two qubits involves associating a complex coefficient - sometimes called an *amplitude* - with each computational basis state, such that the state vector describing the two qubits is

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{2.6}$$

The condition that probabilities sum to one is therefore expressed by the *normalization* condition that $\sum_{x\in\{0,1\}^2}|\alpha_x|^2 = 1$, where the notation $\{0,1\}^2$ means 'the set of strings of length two with each letter being either zero or one'. For a two qubit system, we could measure just a subset of the qubits, say the first qubit, and you can probably guess how this works: measuring the first qubit alone gives 0 with probability $|\alpha_{00}|^2 + |\alpha_{01}|^2$, leaving the post-measurement state

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} \tag{2.7}$$

Note how the post-measurement state is *re-normalized* by the factor $|\alpha_{00}|^2 + |\alpha_{01}|^2$ so that it still satisfies the normalization condition.

An important two qubit state is the *Bell state or EPR pair*,

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.8}$$

The Bell state has the property that upon measuring the first qubit, one obtains two possible results: 0 with probability $\frac{1}{2}$, leaving the post-measurement state $|\varphi'\rangle = |00\rangle$, and 1 with probability $\frac{1}{2}$, leaving $|\varphi'\rangle = |11\rangle$. As a result, a measurement of the second qubit always gives the same result as the measurement of the first qubit. That is, the measurement outcomes are correlated. Indeed, it turns out that other types of measurements can be performed on the Bell state, by first applying some operations to the first or second qubit, and that interesting correlations still exist between the result of a measurement on the first and second qubit.

More generally, we may consider a system of n qubits. The compuational basis states of this system are of the form $|x_1 x_2 \dots x_n\rangle$, and so a quantum state of such a system is specified by 2^n amplitudes. For n = 500 this number is larger than the estimated number of atoms in the Universe!

Chapter 3

Quantum gates and Quantum circuit

Analogous to the way a classical computer is built from an electrical circuit containing wires and logic gates, a quantum computer is built from a quantum circuit containing wires and elementary quantum gates to carry around and manipulate the quantum information.

3.1 Quantum Gates

3.1.1 Single qubit gates

The quantum NOT gate acts *linearly*, that is, it takes the state

$$\alpha |0\rangle + \beta |1\rangle \tag{3.1}$$

to the corresponding state in which the role of $|0\rangle$ and $|1\rangle$ have been interchanged,

$$\alpha |1\rangle + \beta |0\rangle \tag{3.2}$$

It turns out that this linear behavior is a general property of quantum mechanics, and very well motivated empirically; moreover, nonlinear behavior can lead to apparent paradoxes such as time travel, faster than-light communication, and violations of the second laws of thermodynamic.

There is a convenient way of representing the quantum NOT gate in matrix form, which follows directly from the linearity of quantum gates. We define a matrix X to represent the quantum NOT gate as follows:

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{3.3}$$

If the quantum state $\alpha|0\rangle + \beta|1\rangle$ is written in a vector notation as

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}, \tag{3.4}$$

with the top entry corresponding to the amplitude of $|0\rangle$ and the bottom entry the amplitude for $|1\rangle$, then the corresponding output from the quantum NOT gate is

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \tag{3.5}$$

Notice that the action of the NOT gate is to take the state $|0\rangle$ and replace it by the state corresponding to the first column of the matrix X. Similarly, the state $|1\rangle$ is replaced by the state corresponding to the second column of the matrix X.

The appropriate condition on the matrix representing the gate is that the matrix U describing the single qubit gate be *unitary*, that is $U^\dagger U = I$, where $U^\dagger$ is the *adjoint* of U, and I is the two by two identity matrix. The NOT gate is unitary, that is, $X^\dagger X = I$

Amazingly, this unitarity constraint is the only constraint on quantum gates. Any unitary matrix specifies a valid quantum gate! Consider the Z gate

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3.6}$$

which leaves $|0\rangle$ unchanged, and flips the sign of $|1\rangle$ to give $-|1\rangle$. The *Hadamard* gate,

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.7}$$

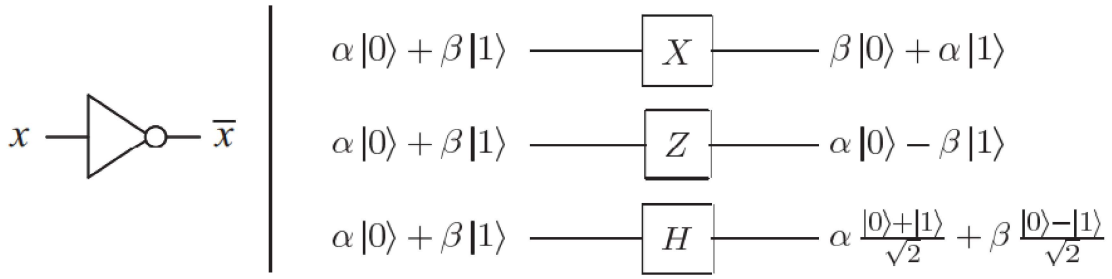


Figure 3.1: Action of X, H and Z gate

turns $|0\rangle$ into $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and turns $|1\rangle$ into $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. This gate is sometimes described as 'square-root' of NOT gate. Note, however, that H^2 is not a NOT gate, as $H^2 = I$ and applying H twice to a state does nothing to it.

An arbitrary quantum computation on any number of qubits can be generated by a finite set of gates that is said to be *universal* for quantum computation.

3.1.2 Multiple qubit gates

The prototypical multi-qubit quantum logic gate is the *controlled-NOT* or *CNOT* gate. This gate has two input qubits, known as the *control* qubit and the *target* qubit, respectively. The circuit representation for the is shown below:

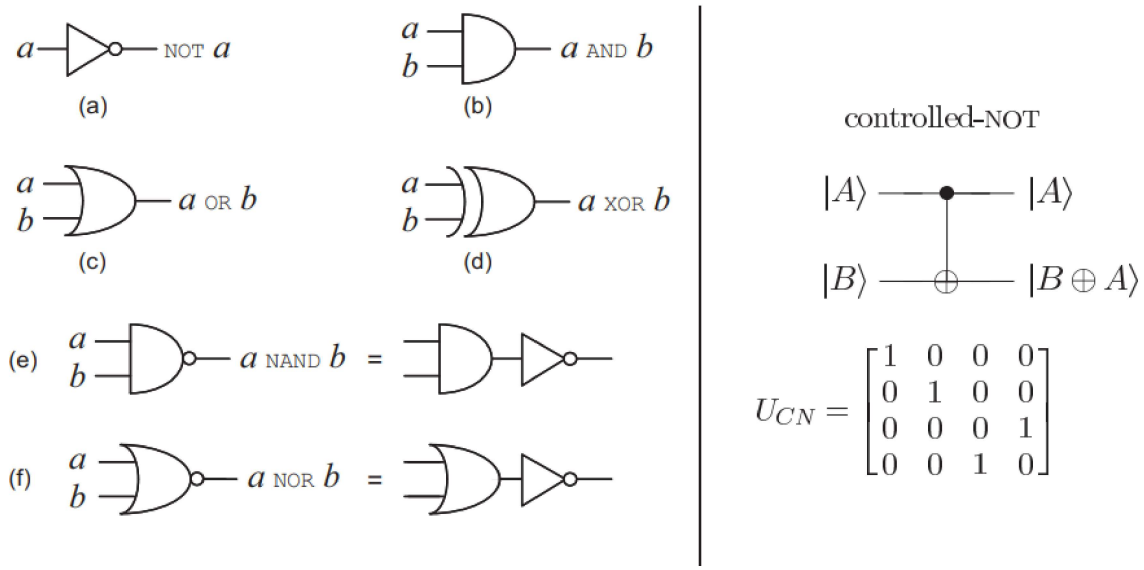


Figure 3.2: On the left are some standard single and multiple bit gates, while on the right is the prototypical multiple qubit gate, the controlled-NOT. The matrix representation of the controlled-NOT, U_{CN}, is written with respect to the amplitudes for $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$, in that order.

The top line represents the control qubit, while the bottom line represents the target qubit. The action of the gate may be described as follows. If the control qubit is set to 0, then the target qubit is left alone. If the control qubit is set to 1, then the target qubit is flipped. In equations:

$$|00\rangle \rightarrow |00\rangle\,; |01\rangle \rightarrow |01\rangle\,; |10\rangle \rightarrow |11\rangle\,; |11\rangle \rightarrow |10\rangle \tag{3.8}$$

Another way of describing the is as a generalization of the classical gate, since the action of the gate may be summarized as $|A, B\rangle \rightarrow |A, A \oplus B\rangle$, where $\oplus$ is addition modulo two, which is exactly what the gate does. That is, the control qubit and the

target qubit are XORed and stored in the target qubit. U_{CN} is a unitary matirx, that is, $U_{CN}^{\dagger}U_{CN} = I$.

Given the output $A \oplus B$ from and XOR gate, it is not possible to determine what the inputs A and B were; there is an irretrievable *loss of information* associated with the irreversible action of XOR gate. On the other hand, unitary quantum gates are *always* invertible, since the inverse of a unitary matrix is also a unitary matrix, and thus a quantum gate can be inverted by another quantum gate.

3.1.3 The R_ϕ, S and T gate

The R_ϕ gate is parametrized, that is, it needs a number ϕ to tell it exactly what to do. The R_ϕ gate performs a rotation of ϕ around the z-axis direction.

$$R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} \tag{3.9}$$

The Z-gate is a special case of the R_ϕ gate, with $\phi = \pi$.

The S gate is a R_ϕ gate with $\phi = \pi/2$. It does a quarter turn around the Bloch sphere. It is important to note that S gate is **not** it's own inverse. The $S^\dagger$ gate is clearly a R_ϕ gate with $\phi = -\pi/2$.

$$S = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{bmatrix} \; ; \quad S^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{2}} \end{bmatrix} \tag{3.10}$$

The S gate is also called as the $\sqrt{Z}$ gate due to the fact that two successively applied S gate has the same effect as a Z gate:

$$SS\left|q\right\rangle = Z\left|q\right\rangle. \tag{3.11}$$

The T gate is a very commonly used gate, it is an R_ϕ gate with $\phi = \pi/4$:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix} \; ; \quad T^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{4}} \end{bmatrix} \tag{3.12}$$

As with the S-gate, the T-gate is sometimes also known as the $\sqrt{S}$ gate.

3.1.4 The Swap, Toffoli and Fredkin gate

Sometimes we need to move information around in a quantum computer. For some qubit implementations, this could be done by physically moving them. Another option

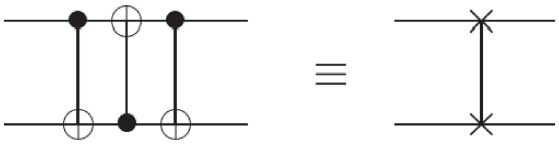


Figure 3.3: Circuit swappping two qubits, and an equivalent schematic symbol notation for this circuit.

is simply to move the state between two qubits. This is done by the $SWAP$ gate. The sequence of gates has the following sequence of effects on a computational basis state $|a, b\rangle$,

$$\begin{aligned} |a, b\rangle &\rightarrow |a, a \oplus b\rangle \\ &\rightarrow |a \oplus (a \oplus b), a \oplus b\rangle = |b, a \oplus b\rangle \\ &\rightarrow |b, (a \oplus b) \oplus b\rangle = |b, a\rangle \, . \end{aligned}$$

Any classical circuit can be replaced by an equivalent circuit containing only *reversible* elements, by making use of a reversible gate known as the *Toffoli gate*. The Toffoli gate has three input bits and three output bits. Two of the bits are *control bits* that are unaffected by the action of the Toffoli gate. The third bit is a *target bit* that is flipped if both control bits are set to 1, and otherwise is left alone. Note that applying the Toffoli gate twice to a set of bits has the effect $(a, b, c) \rightarrow (a, b, c \oplus ab) \rightarrow (a, b, c)$ and thus the Toffoli gate is a reversible gate, since it has an inverse – itself.

Inputs			Outputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

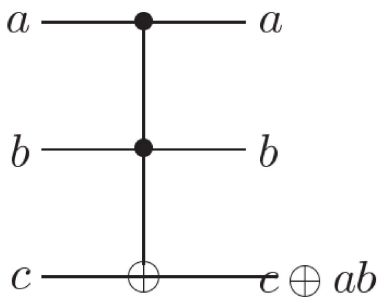


Figure 3.4: Truth table for the Toffoli gate, and its circuit representation

The *Fredkin gate* has three input bits and three output bits, which we refer to as a, b, c and $a^{'}, b^{'}, c^{'}$ respectively. The bit c is a *controlbit*, whose value is not changed by the action of the Fredkin gate, that is, $c^{'} = c$. The reason c is called the control

bit is because it controls what happens to the other two bits, a and b. If c is set to 0 then a and b are left alone, $a^{'} = a, b^{'} = b$. If c is set to 1, a and b are swapped, $a^{'} = b, b^{'} = a$.

Inputs			Outputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	1	1

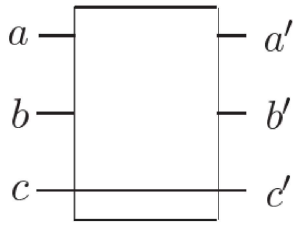
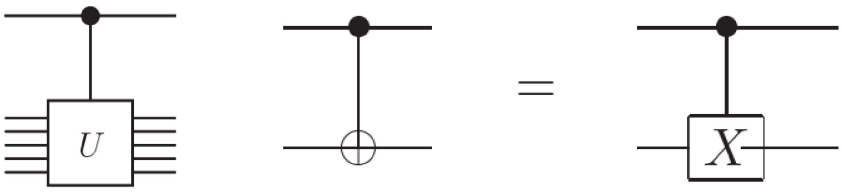


Figure 3.5: Fredkin gate truth table and circuit representation. The bits a and b are swapped if the control bit c is set 1, and otherwise are left alone

3.1.5 The Controlled-U gate

Suppose U is any unitary matrix acting on some number n of qubits, so U can be regarded as a quantum gate on those qubits. Then we can define a $controlled - U$ gate which is a natural extension of the controlled-NOT gate. Such a gate has a single *controlqubit*, indicated by the line with the black dot, and *ntargetqubits*, indicated by the boxed U. If the control qubit is set to 0 then nothing happens to the target qubits. If the control qubit is set to 1 then the gate U is applied to the target qubits. The prototypical example of the controlled-U gate is the controlled-NOT gate, which is a controlled-U gate with $U = X$.



(a) (b)

Figure 3.6: (a) Controlled-U gate; (b) Controlled-U with U=X

3.2 Quantum Circuits and measurement

3.2.1 Quantum Circuits

A *quantum circuit* is a computational routine consisting of *coherent quantum operations on quantum data*, such as *qubits*, and *concurrent real-time classical computation.* It is an ordered sequence of *quantum gates, measurements and resets*, all of which may be conditioned on and use data from the real-time classical computation.

A set of quantum gates is said to be universal if any unitary transformation of the quantum data can be efficiently approximated arbitrarily well as a sequence of gates in the set. Any quantum program can be represented by a sequence of quantum circuits and non-concurrent classical computation.

Each line in the circuit represents a *wire* in the quantum circuit. This wire does not necessarily correspond to a physical wire; it may correspond instead to the passage of time, or perhaps to a physical particle such as a photon – a particle of light – moving from one location to another through space. It is conventional to assume that the state input to the circuit is a computational basis state, usually the state consisting of all $|0\rangle$s.

Note:

1. We don't allow 'loops', that is, feedback from one part of the quantum circuit to another; we say the circuit is *acyclic.*

2. Classical circuits allow wires to be 'joined' together, an operation known as *FANIN*, with the resulting single wire containing the bitwise *OR* of the inputs. Obviously this operation is not reversible and therefore not unitary, so we don't allow in our quantum circuits.

3. Third, the inverse noperation,*FANOUT*, whereby several copies of a bit are produced is also not allowed in quantum circuits. In fact, it turns out that quantum mechanics forbids the copying of a qubit, making the *FANOUT* operation impossible!

While a universal quantum computer can do anything any classical computer can, we often add classical parts to our quantum circuits because quantum states are fragile.

When we measure the qubit, we collapse its state and destroy a lot of the information. Since all measurement does is destroy information, we can in theory always measure last and lose no computational advantage. In reality, measuring early offers many practical advantages. The advantage is that classical channels are very stable, while we don't really have a way of sending quantum information to other people since the channels are so difficult to create.

To use the results of our quantum computation in our classical, everyday world, we need to measure and interpret these states at the end of our computation.

3.2.2 N-Controlled Operations

Earlier we saw the implemetation of controlled-U opeartion for arbitrary single qubit U, using only single quibit operations and the $CNOT$ gate.

A $controlled - U$ operation is a two qubit operation, with a control and a target qubit. If the control qubit is set then U is applied to the target qubit, otherwise the target qubit is left alone, that is, $|c\rangle |t\rangle \rightarrow |c\rangle U^c |t\rangle$.

Suppose we have $n+k$ qubits, and U is a k qubit unitary operator. Then we define the controlled operation $C^n(U)$ by the equation,

$$C^n(U) |x_1 x_2 \ldots x_n\rangle |\psi\rangle = |x_1 x_2 \ldots x_n\rangle U^{x_1 x_2 \ldots x_n} |\psi\rangle , \tag{3.13}$$

where $x_1 x_2 \ldots x_n$ in the exponent of U means the product of the bits $x_1, x_2, \ldots, x_n$. That is, the operator U is applied to the last k qubits if the first n qubits are all equal to one, and otherwise, nothing is done.

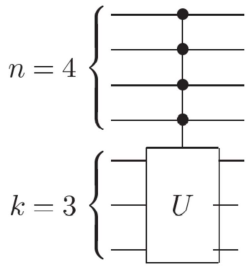


Figure 3.7: Sample circuit representation for the $C^n(U)$

The implementation of $C^n(U)$ gate is shown below. The circuit divides up into three stages, and makes use of a small number $(n-1)$ of working qubits, which all start and end in the state $|0\rangle$. Suppose the control qubits are in the computational basis state $|c_1 c_2 \ldots c_n\rangle$. The first stage of the circuit is to reversibly AND all the control bits $c_1 c_2 \ldots c_n$ together to produce the product $c_1 \cdot c_2 \ldots c_n$. To do this, the first gate in the circuit $ANDs$ c_1 and c_2 together, using a Toffoli gate, changing the state of the first work qubit to $|c_1 \cdot c_2\rangle$. The next Toffoli gate $ANDs$ c_3 with the product $c_1 \cdot c_2$, changing the state of the second work qubit to $|c_1 \cdot c_2 \cdot c_3\rangle$. We continue applying Toffoli gates in this fashion, until the final work qubit is in the state $|c_1 \cdot c_2 \ldots c_n\rangle$. Next, a U operation on the target qubit is performed, conditional on the final work qubit being set to one. That is, U is applied if and only if all of c_1 through c_n are set. Finally, the last part of the circuit just reverses the steps of the first stage, returning all the work qubits to their initial state, $|0\rangle$. The combined result, therefore, is to apply the unitary operator U to the target qubit, if and only if all the control bits c_1 through c_n are set, as desired.

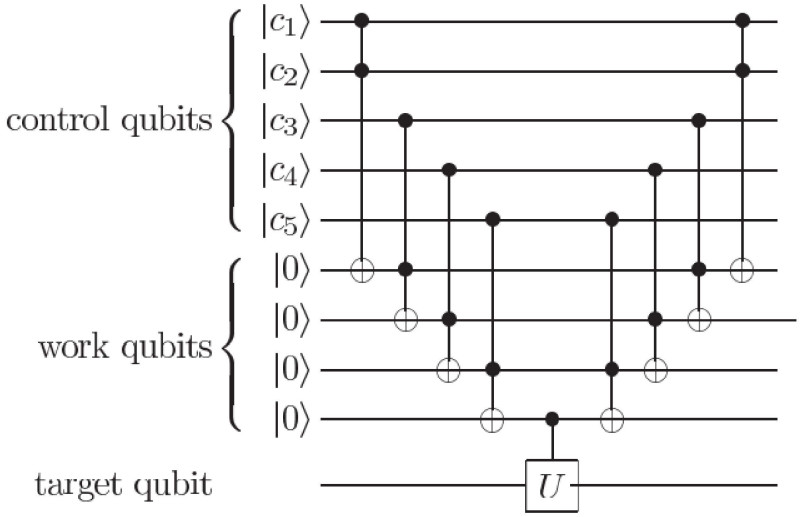


Figure 3.8: Network implementing $C^n(U)$ operation, for the case $n = 5$.

3.2.3 Measurement

A final element used in quantum circuits, almost implicitly sometimes, is measurement. In our circuits, we shall denote a projective measurement in the computational basis using a 'meter' symbol.

Figure 3.9: Symbol for projective measurement on a single qubit. In this circuit nothing further is done with the measurement result, but in more general quantum circuits it is possible to change later parts of the quantum circuit, conditional on measurement outcomes in earlier parts of the circuit

There are two important principles that it is worth bearing in mind about quantum circuits. The first principle is that classically conditioned operations can be replaced by quantum conditioned operations:

Principle of deferred measurement. *Measurements can always be moved from an intermediate stage of a quantum circuit to the end of the circuit; if the measurement results are used at any stage of the circuit then the classically controlled operations can be replaced by conditional quantum operations.*

Often, quantum measurements are performed as an intermediate step in a quantum

circuit, and the measurement results are used to conditionally control subsequent quantum gates. The second principle is even more obvious - and suprisingly useful!

Principle of implicit measurement. *Without loss of generality, any unterminated quantum wires (qubits which are not measured) at the end of a quantum circuit may be assumed to be measured.*

It is important to keep in mind that the role of measurements in quantum circuits is an interface between the quantum and classical worlds, measurement is generally considered to be an irreversible operation, destroying quantum information and replacing it with classical information.

Chapter 4

Quantum Algorithms

The spectacular promise of quantum computers is to enable new algorithms which render feasible problems requiring exorbitant resources for their solution on a classical computer. At the time of writing, two broad classes of quantum algorithms are known which fulfill this promise. The first class of algorithms is based upon Shor's *quantum Fourier transform*, and includes remarkable algorithms for solving the factoring and discrete logarithm problems, providing a striking *exponential* speedup over the best known classical algorithms. The second class of algorithms is based upon Grover's algorithm for performing *quantum searching*.

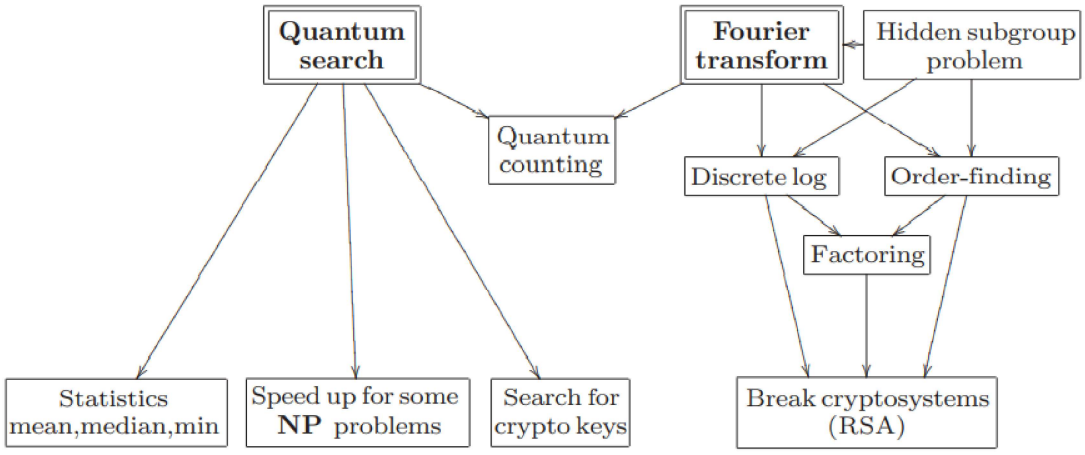


Figure 4.1: The main quantum algorithms and their relationships, including some notable applications.

The quantum Fourier transform also has many interesting applications. It can be used to solve the discrete logarithm and factoring problems. These results, in turn, enable a quantum computer to break many of the most popular cryptosystems now in use, including the RSA cryptosystem. The Fourier transform also turns out to be

closely related to an important problem in mathematics, finding a hidden subgroup (a generalization of finding the period of a periodic function).

Why are there so few quantum algorithms known which are better than their classical counterparts? The answer is that coming up with good quantum algorithms seems to be a difficult problem. There are at least two reasons for this. First, algorithm design, be it classical or quantum, is not an easy business! The history of algorithms shows us that considerable ingenuity is often required to come up with near optimal algorithms, even for apparently very simple problems, like the multiplication of two numbers. Finding good quantum algorithms is made doubly difficult because of the additional constraint that we want our quantum algorithms to be better than the best known classical algorithms. A second reason for the difficulty of finding good quantum algorithms is that our intuitions are much better adapted to the classical world than they are to the quantum world. If we think about problems using our native intuition, then the algorithms which we come up with are going to be classical algorithms. It takes special insights and special tricks to come up with good quantum algorithms.

4.1 The Quantum Fourier Transform

One of the most useful ways of solving a problem in mathematics or computer science is to *transform* it into some other problem for which a solution is known. A great discovery of quantum computation has been that some such transformations can be computed much faster on a quantum computer than on a classical computer, a discovery which has enabled the construction of fast algorithms for quantum computers.

One such transformation is the discrete Fourier transform. In the usual mathematical notation, the *discrete Fourier transform* takes as input a vector of complex numbers, $x_0, \ldots, x_{N-1}$ where the length N of the vector is a fixed parameter. It outputs the transformed data, a vector of complex numbers $y_0, \ldots, y_{N-1}$, defined by

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k/N} \tag{4.1}$$

The *quantum Fourier transform* is exactly the same transformation. The quantum Fourier transform on an orthonormal basis $|0\rangle, \ldots, |N-1\rangle$ acts on a quantum state $|X\rangle = \sum_{j=0}^{N-1} x_j |j\rangle$ and maps it to the quantum state

$$|Y\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k/N} |k\rangle \tag{4.2}$$

Equivalently, the action of QFT on an arbitrary state may be written as,

$$\sum_{j=0}^{N-1} x_j |j\rangle \rightarrow \sum_{k=0}^{N-1} y_k |k\rangle \tag{4.3}$$

It is easy to show that the Quantum Fourier Transform is unitary.

4.1.1 Counting in the Fourier Basis

The QFT transforms between two bases, the computational basis (z-basis), and the Fourier basis.

$$|\textit{State in computational basis}\rangle \rightarrow |\textit{State in Fourier basis}\rangle$$

In the computational basis, we store numbers in binary using the states $|0\rangle$ and $|1\rangle$. In the Fourier basis, we store numbers using different rotations around the Z-axis. The number we want to store dictates the angle at which each qubit is rotated around the Z-axis. In the state $|\bar{0}\rangle$, all qubits are in the state $|+\rangle$.

To encode $|\bar{x}\rangle$ on n qubit, we rotate the leftmost qubit by $\frac{x}{2^n}$ full turns. The next qubit is turned double this, which is doubled again for the next qubit and so on.

4.1.2 QFT on 2^n qubits

Below is the QFT on $N = 2^n$ qubits acting on the state $|x\rangle = |x_1 \dots x_n\rangle$ where x_1 is the most significant bit.

$$QFT_N |x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x y/2^n} |y\rangle \tag{4.4}$$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i (\sum_{k=1}^{n} y_k/2^k) x} |y_1 \dots y_n\rangle \tag{4.5}$$

$$= \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \prod_{k=1}^{n} e^{2\pi i x y_k/2^k} |y_1 \dots y_n\rangle \tag{4.6}$$

$$= \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} (|0\rangle + e^{2\pi i x/2^k} |1\rangle) \tag{4.7}$$

$$= \frac{1}{\sqrt{2^n}} \left((|0\rangle + e^{\frac{2i\pi}{2} x} |1\rangle) \otimes (|0\rangle + e^{\frac{2i\pi}{2^2} x} |1\rangle) \otimes \cdots \otimes (|0\rangle + e^{\frac{2i\pi}{2^n} x} |1\rangle) \right) \tag{4.8}$$

In equation 4.4, we substitue $N = 2^n$ and then in equation 4.5 we rewrite y in binary notation, that is, $y = y_1 \dots y_n$ and $y = \sum_{k=1}^{n} y_k/2^k$. Following this we expand the exponential of a sum to a product of exponentials and later rearranging the sum and products, and expanding $\sum_{y=0}^{N-1} = \sum_{y_1=0}^{1} \sum_{y_2=0}^{1} \cdots \sum_{y_n=0}^{1}$ we get equation 4.8.

4.1.3 The circuit that implements QFT and Time Complexity

The circuit that implements QFT makes use of two gates. The first one is a single-qubit Hadamard gate H, which is considered as a single-qubit QFT. The action of H on the single-qubit state $|x_k\rangle$ is

$$H|x_k\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + exp\left(\frac{2i\pi}{2}x_k\right)|1\rangle\right) \tag{4.9}$$

The second is a two-qubit controlled rotation $CROT_k$ defined as

$$CROT_k = \begin{bmatrix} I & 0 \\ 0 & UROT_k \end{bmatrix} \tag{4.10}$$

where

$$UROT_k = \begin{bmatrix} 1 & 0 \\ 0 & exp\left(\frac{2\pi i}{2}\right) \end{bmatrix} \tag{4.11}$$

The action of $CROT_k$ on a two-qubit state $|x_l x_j\rangle$ where the first qubit is the control and the second is the target qubit given by

$$CROT_k|0x_j\rangle = |0x_j\rangle \tag{4.12}$$

and

$$CROT_k|1x_j\rangle = exp\left(\frac{2\pi i}{2}x_j\right)|1x_j\rangle \tag{4.13}$$

Given these two gates, a circuit that implements an n-qubit QFT is shown below.

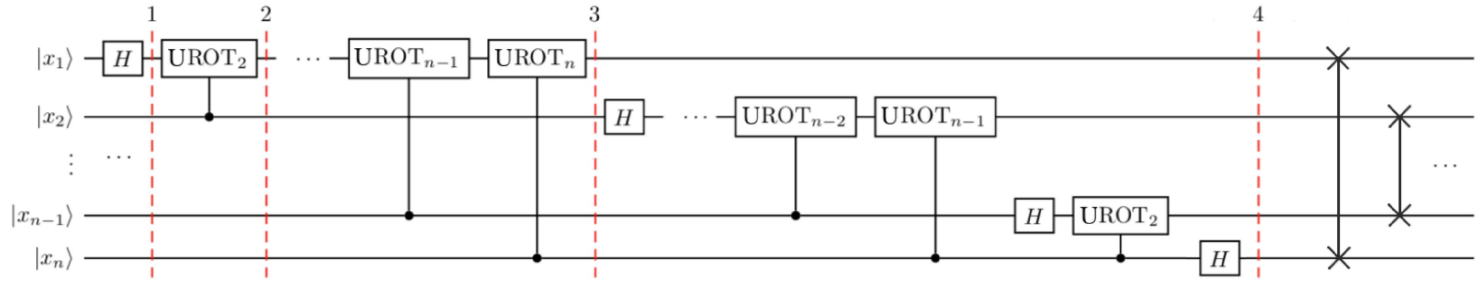


Figure 4.2: Efficient circuit for Quantum Fourier Transform with swap gates at the end

How many gates does this circuit use? We start by doing a Hadamard gate and $n-1$ conditional rotations on the first qubit – a total of n gates. This is followed

by a Hadamard gate and $n-2$ conditional rotations on the second qubit, for a total of $n + (n-1)$ gates. Continuing in this way, we see that $n + (n-1) + \cdots + 1 = n(n+1)/2$ gates are required, plus the gates involved in the swaps. At most $n/2$ swaps are required, and each swap can be accomplished using three controlled-NOT gates. Therefore, this circuit provides a $\Theta(n^2)$ algorithm for performing the Quantum Fourier Transform.

In contrast, the best classical algorithms for computing the discrete Fourier transform on 2^n elements are algorithms such as the $FastFourierTransform(FFT)$, which compute the discrete Fourier transform using $\Theta(n2^n)$ gates. That is, it requires exponentially more operations to compute the Fourier transform on a classical computer than it does to implement the quantum Fourier transform on a quantum computer.

Note: As the QFT circuit becomes large, an increasing amount of time is spent doing increasingly slight rotations. It turns out that we can ignore rotations below a certain threshold and still get decent results, this is known as the *approximate QFT*. This is also important in physical implementations, as reducing the number of operations can greatly reduce decoherence and potential gate errors.

4.2 Quantum Phase Estimation

The Fourier transform is the key to a general procedure known as *phase estimation*, which is one of the most important subroutines in quantum computation. It serves as a central building block for many quantum algorithms. The objective of the algorithm is the following:

Given a unitary operator U, the algorithm estimates θ in

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle \tag{4.14}$$

Here $|\psi\rangle$ is an eigenvector and $e^{2\pi i\theta}$ is the corresponding eigenvalue. Since U is unitary, all of its eigenvalues have a norm of 1.

To perform the estimation we assume that we have available *black boxes* (sometimes known as *oracles*) capable of preparing the state $|u\rangle$ and performing the controlled-U^{2^j} operation, for suitable non-negative integers j.

4.2.1 Quantum Phase Estimation Circuit

The quantum phase estimation procedure uses two registers. The first register contains t *'counting qubits'* initially in the state $|0\rangle$. How we choose t depends on two things: the number of digits of accuracy we wish to have in our estimate for φ, and with what probability we wish the phase estimation procedure to be successful. The dependence of t on these quantities emerges naturally from the following analysis.

The second register begins in the state $|u\rangle$, and contains as many qubits as is necessary to store $|u\rangle$.

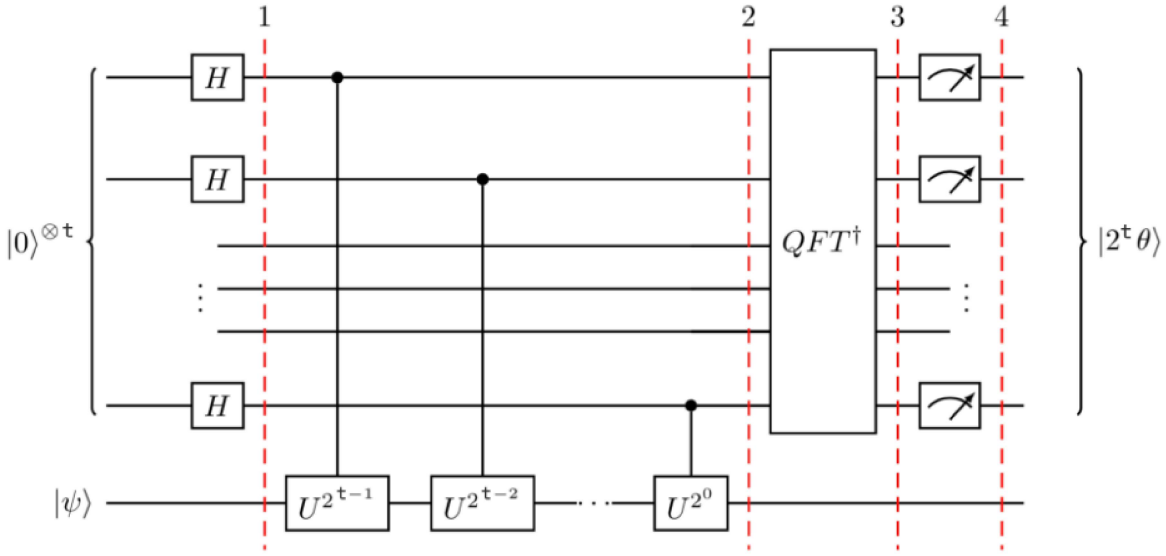


Figure 4.3: Schematic of the overall phase estimation procedure.

The quantum phase estimation algorithm uses phase kickback to write the phase of U (in the Fourier basis) to the t qubits in the counting register. We then use the inverse QFT to translate this from the Fourier basis into the computational basis, which we can measure.

When we use a qubit to control the U-gate, the qubit will turn (due to kickback) proportionally to the phase $e^{2\pi i\theta}$. We can use successive CU-gates to repeat this rotation an appropriate number of times until we have encoded the phase θ as a number between 0 and 2^t in the Fourier basis. Then we simply use $QFT^\dagger$ to convert this into the computational basis.

4.2.2 Mathematical Foundation

The circuit operates in the following steps:

1. **Setup:** $|\psi\rangle$ is in one set of qubit registers. An additional set of n qubits form the counting register on which we will store the value $2^n\theta$.

$$|\psi_0\rangle = |0\rangle^{\otimes n} |\psi\rangle \tag{4.15}$$

2. **Superposition:** Apply a n-bit Hadamard gate operation $H^{\otimes n}$ on the counting register:

$$|\psi_1\rangle = \frac{1}{2^{\frac{n}{2}}} \left(|0\rangle + |1\rangle\right)^{\otimes n} |\psi\rangle \tag{4.16}$$

3. **Controlled-Unitary Operations:** We need to introduce the controlled unitary CU that applies the unitary operator U on the target register only if its corresponding control bit is $|1\rangle$. Since U is a unitary operator with eigenvector $|\psi\rangle$ such that $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, this means:

$$U^{2^j}|\psi\rangle = U^{2^{j-1}}U|\psi\rangle = U^{2^{j-1}}e^{2i\pi\theta}|\psi\rangle = \cdots = e^{2i\pi\theta 2^j}|\psi\rangle \tag{4.17}$$

Applying all the n controlled operations CU^{2^j} with $0 \leq j \leq n-1$, and using the relation $|0\rangle \otimes |\psi\rangle + |1\rangle \otimes e^{2i\pi\theta}|\psi\rangle = \left(|0\rangle + |1\rangle e^{2i\pi\theta}\right) \otimes |\psi\rangle$:

$$|\psi_2\rangle = \frac{1}{2^{\frac{n}{2}}}\left(|0\rangle + e^{2i\pi\theta 2^{n-1}}|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{2i\pi\theta 2^1}|1\rangle\right) \otimes \left(|0\rangle + e^{2i\pi\theta 2^0}|1\rangle\right) \otimes |\psi\rangle \tag{4.18}$$

$$= \frac{1}{2^{\frac{n}{2}}}\sum_{k=0}^{2^n-1} e^{2i\pi\theta k}|k\rangle \otimes |\psi\rangle \tag{4.19}$$

where k denotes the integer representation of n-bit binary numbers.

4. **Inverse Fourier Transform:**Recall that QFT maps an n-qubit input state $|x\rangle$ into an output as

$$QFT|x\rangle = \frac{1}{2^{\frac{n}{2}}}\left(|0\rangle + e^{\frac{2i\pi}{2}x}|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2i\pi}{2^2}x}|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{\frac{2i\pi}{2^{n-1}}x}|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2i\pi}{2^n}x}|1\rangle\right) \tag{4.20}$$

Replacing x by $2^n\theta$ in the above expression gives exactly the expression derived in step 2 above. Therefore, to recover the state $|2^n\theta\rangle$, apply an inverse Fourier transform on the auxiliary register. Doing so, we find

$$|\psi_3\rangle = \frac{1}{2^{\frac{n}{2}}}\sum_{k=0}^{2^n-1} e^{2i\pi\theta k}|k\rangle \otimes |\psi\rangle \xrightarrow{QFT_n^{-1}} \frac{1}{2^n}\sum_{x=0}^{2^n-1}\sum_{k=0}^{2^n-1} e^{-\frac{2i\pi k}{2^n}(x-2^n\theta)}|x\rangle \otimes |\psi\rangle \tag{4.21}$$

5. **Measurement:** The above expression peaks near $x = 2^n\theta$. For the case when $2^n\theta$ is an integer, measuring in the computational basis gives the phase in the auxiliary register with high probability:

$$|\psi_4\rangle = |2^n\theta\rangle \otimes |\psi\rangle \tag{4.22}$$

For the case when $2^n\theta$ is not an integer, it can be shown that the above expression still peaks near $x = 2^n\theta$ with probability better than $4/\pi^2 \approx 40\%$.

Algorithm: Quantum Phase Estimation

Inputs: (1) A black box which performs a controlled-U^j operation, for integer j,
(2) an eigenstate $|u\rangle$ of U with eigenvalue $e^{2\pi i\varphi_u}$, and
(3) $t = n + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right)\right\rceil$ qubits initialized to $|0\rangle$.

Outputs: An n-bit approximation $\overline{\varphi}_u$ to φ_u

Runtime: $O(t^2)$ operations and one call to controlled-U^j black box. Succeeds with probability at least $1 - \epsilon$.

Procedure:

1. $|0\rangle\,|u\rangle$ — initial state
2. $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle\,|u\rangle$ — create superposition
3. $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle\,U^j\,|u\rangle$ — apply black box
4. $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{2\pi i j\varphi_u}\,|j\rangle\,|u\rangle$ — result of black box
5. $\rightarrow |\bar{\varphi_u}\rangle\;|u\rangle$ — apply inverse Fourier transform
6. $\rightarrow \bar{\varphi_u}$ — measure first register

4.2.3 Performance and requirements

Let b be the integer in the range 0 to $2^t - 1$ such that $b/2^t = 0.b_1 \ldots b_t$ is the best t bit approximation to φ which is less than φ. That is, the difference $\delta \equiv \varphi - b/2^t$ between φ and $b/2^t$ satisfies $0 \leq \delta \leq 2^{-t}$. Here it is shown that the observation at the end of the phase estimation procedure produces a result which is close to b, and thus enables to estimate φ accurately, with high probability. Applying the inverse quantum Fourier transform to Equation (4.19) produces the state

$$\frac{1}{2^t} \sum_{k,l=0}^{2^t-1} e^{\frac{-2\pi i k l}{2^t}} e^{2\pi i\varphi k}\,|l\rangle \tag{4.23}$$

Let α_l be the amplitude of $|(b+l)(\mod 2^t)\rangle$,

$$\alpha_1 \equiv \frac{1}{2^t} \sum_{k=0}^{2^t-1} \left(e^{2\pi i(\varphi-(b+l))/2^t}\right)^k \tag{4.24}$$

This is the sum of a geometric series, so

$$\alpha_1 \equiv \frac{1}{2^t}\left(\frac{1-e^{2\pi i(2^t\varphi-(b+l))}}{1-e^{2\pi i(\varphi-(b+l)/2^t)}}\right) \tag{4.25}$$

$$= \frac{1}{2^t}\left(\frac{1-e^{2\pi i(2^t\delta-l)}}{1-e^{2\pi i(\delta-l/2^t)}}\right) \tag{4.26}$$

Suppose the outcome of the final measurement is m. We aim to bound the probability of obtaining a value of m such that $|m-b|>e$, where e is a positive integer characterizing our desired tolerance to error. The probability of observing such an m is given by

$$p(|m-b|>e) = \sum_{-2^{t-1}<l\leq-(e+1)} |\alpha_l|^2 + \sum_{e+1\leq l\leq 2^{t-1}} |\alpha_l|^2 \tag{4.27}$$

But for any real θ, $\left|1-exp(i\theta)\right| \leq 2$, so

$$|\alpha_l| \leq \frac{2}{2^t\left|1-e^{2\pi i(\delta-l/2^t)}\right|} \tag{4.28}$$

By elementary geometry or calculus $\left|1-exp(i\theta)\right| \geq 2|\theta|/\pi$ whenever $-\pi\leq\theta\leq\pi$. But when $-2^{t-1}<l\leq 2^{t-1}$ we have $-\pi\leq 2\pi(\delta-l/2^t)\leq\pi$. Thus

$$|\alpha_l| \leq \frac{1}{2^{t+1}(\delta-l/2^t)} \tag{4.29}$$

Combining (4.27) and (4.29) gives

$$p(|m-b|>e) \leq \frac{1}{4}\left[\sum_{l=-2^{t-1}+1}^{-(e+1)} \frac{1}{(l-2^t\delta)^2} + \sum_{l=e+1}^{2^{t-1}} \frac{1}{(l-2^t\delta)^2}\right] \tag{4.30}$$

Recalling that $0\leq 2^t\delta\leq 1$, we obtain

$$p(|m-b|>e) \leq \frac{1}{4}\left[\sum_{l=-2^{t-1}+1}^{-(e+1)} \frac{1}{l^2} + \sum_{l=e+1}^{2^{t-1}} \frac{1}{(l-1)^2}\right] \tag{4.31}$$

$$\leq \frac{1}{2}\sum_{l=e}^{2^{t-1}-1} \frac{1}{l^2} \tag{4.32}$$

$$\leq \frac{1}{2}\int_{e-1}^{2^{t-1}-1} dl\frac{1}{l^2} \tag{4.33}$$

$$= \frac{1}{2(e-1)} \tag{4.34}$$

Suppose we wish to approximate φ to an accuracy 2^{-n}, that is, we choose $e = 2^{t-n} - 1$. By making use of $t = n+p$ qubits in the phase estimation algorithm we see from (4.34) that the probability of obtaining an approximaiton correct to this accuracy is at least $1 - 1/2(2^p - 2)$. Thus to successfully obtain φ accurate to n bits with probability of success at least $1 - \epsilon$ we choose

$$t = n + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right) \right\rceil \tag{4.35}$$

4.3 Order Finding

For positive integers x and N, with no common factors, the *order* of x modulo N is defined to be the least positive integer, r, such that $x^r = 1(\mod N)$. The order-finding problem is to determine the order for some specific x and N. Order-finding is believed to be a hard problem on a classical computer, in the sense that no algorithm is known to solve the problem using resources polynomial in the $O(L)$ bits needed to specify the problem, where $L \equiv \lceil log(N) \rceil$ is the number of bits needed to specify N.

The quantum algorithm for order-finding is just the phase estimation algorithm applied to the unitary operator

$$U|y\rangle \equiv |xy(\mod N)\rangle \tag{4.36}$$

with $y \in \{0,1\}^L$. A simple calculation shows that the states defined by

$$|u_s\rangle \equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} exp\left[\frac{-2\pi isk}{r}\right] |x^k mod N\rangle\,, \tag{4.37}$$

for integer $0 \leq s \leq r-1$ are eigenstates of U, since

$$U|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} exp\left[\frac{-2\pi isk}{r}\right] |x^{k+1} mod N\rangle \tag{4.38}$$

$$= exp\left[\frac{2\pi is}{r}\right] |u_s\rangle \tag{4.39}$$

Using the phase estimation procedure allows us to obtain, with high accuracy, the corresponding eigenvalues $exp(2\pi is/r)$, from which we can obtain the order r with a little bit more work.

There are two important requirements to be able to use the phase estimation procedure: we must have efficient procedures to implement a controlled U^{2^j} operation for any integer j, and we must be able to efficiently prepare an eigenstate $|u_s\rangle$ with a non-trivial eigenvalue, or at least a superposition of such eigenstates. The first requirement is satisfied by using a procedure known as *modular exponentiation*, with

which we can implement the entire sequence of controlled-U^{2^j} operations applied by the phase estimation procedure using $O(L^3)$ gates.

The second requirement is: preparing $|u_s\rangle$ requires that we know r, so this is out of the question. Fortunately, there is a clever observation which allows us to circumvent the problem of preparing $|u_s\rangle$ which is that

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle \,. \tag{4.40}$$

In performing the phase estimation procedure, from equation (4.35), if we use

$$t = 2L + 1 + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right) \right\rceil \tag{4.41}$$

qubits in the first register, and prepare the second register in the state $|1\rangle$ - which is trivial to construct - it follows that for each s in the range 0 through $r-1$, we will obtain an estimate of the phase $\varphi \approx s/r$ accurate to $2L+1$ bits, with probability at least $(1-\epsilon)/r$.

4.3.1 Modular Exponentiation

The sequence of controlled-U^{2^j} operations used in phase estimation is equivalent to multiplying the contents of the second register by the modular exponential $x^z (\mathrm{mod}\ N)$, where z is the contents of the first register.

$$\begin{aligned} |z\rangle\,|y\rangle &\rightarrow |z\rangle\, U^{z_t 2^{t-1}} \ldots U^{z_1 2^0}\,|y\rangle && (4.42)\\ &= |z\rangle\,|x^{z_t 2^{t-1}} \times \cdots \times x^{z_1 2^0} y (\mathrm{mod} N)\rangle && (4.43)\\ &= |z\rangle\,|x^z y (\mathrm{mod} N)\rangle && (4.44) \end{aligned}$$

The basic idea is to reversibly compute the function $x^z (\bmod N)$ of z in a third register, and then to reversibly multiply the contents of the second register by $x^z (\mathrm{mod} N)$, using the trick of uncomputation to erase the contents of the third register upon completion.

The algorithm for computing the modular exponentiation has two stages. The first stage uses modular multiplication to compute $x^2 (\mathrm{mod} N)$, by squaring x modulo N, then computes $x^4 (\mathrm{mod} N)$ by squaring $x^2 (\mathrm{mod} N)$, and continues in this way, computing $x^{2^j} (\mathrm{mod} N)$ for all j up to $t-1$. We use $t = 2L + 1 + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right) \right\rceil = O(L)$, so a total of $t - 1 = O(L)$ squaring operations is performed at a cost of $O(L^2)$ each, for a total cost of $O(L^3)$ for the first stage. The second stage of the algorithm is based upon the observation,

$$x^z (\mathrm{mod} N) = \left(x^{z_t 2^{t-1}} (\mathrm{mod} N)\right) \left(x^{z_{t-1} 2^{t-2}} (\mathrm{mod} N)\right) \ldots \left(x^{z_1 2^0} (\mathrm{mod} N)\right) \tag{4.45}$$

Performing $t-1$ modular multiplications with a cost $O(L^2)$ each, we see that this product can be computed using $O(L^3)$ gates.

It is now straightforward to construct a reversible circuit with a t bit register and an L bit register which, when started in the state (z, y) outputs $(z, x^z y(\mathrm{mod}\, N))$, using $O(L^3)$ gates, which can be translated into a quantum circuit using $0(L^3)$ gates computing the transformation $|z\rangle\,|y\rangle \to |z\rangle\,|x^z y(\mathrm{mod} N)\rangle$.

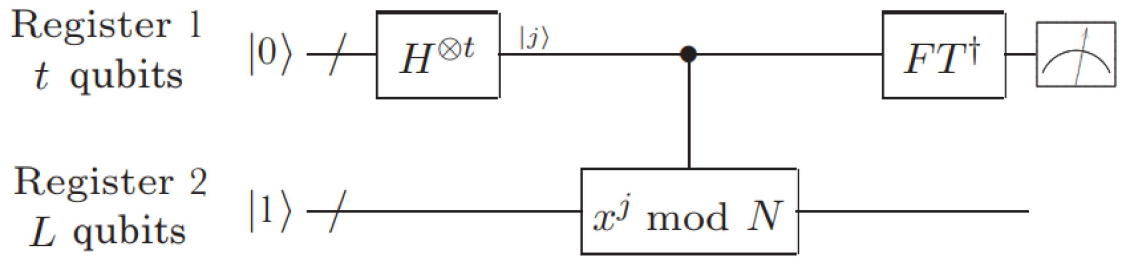


Figure 4.4: Quantum circuit for the order-finding algorithm

4.3.2 The continued fractions algorithm

The idea of the continued fractions algorithm is to describe real numbers in terms of integers alone, using expressions of the form

$$[a_0, \ldots, a_M] \equiv a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\cdots + \cfrac{1}{a_M}}}} \tag{4.46}$$

where $a_0, \ldots, a_M$ are positive integers. The *continued fractions algorithm* is a method for determining the continued fraction expansion of an arbitrary real number.

Suppose we are trying to decompose 31/13 as a continued fraction. The first step of the continued fractions algorithm is to split 31/13 into its integer and fractional part,

$$\frac{31}{13} = 2 + \frac{5}{13} \tag{4.47}$$

Next we invert the fractional part, obtaining

$$\frac{31}{13} = 2 + \frac{1}{\frac{13}{5}} \tag{4.48}$$

These steps - split then invert - are now applied to 13/5, giving

$$\frac{31}{13} = 2 + \frac{1}{2 + \frac{3}{5}} = 2 + \frac{1}{2 + \frac{1}{\frac{5}{3}}} \tag{4.49}$$

Next we split and invert 5/3:

$$\frac{31}{13} = 2 + \frac{1}{2 + \frac{1}{1+\frac{2}{3}}} = 2 + \frac{1}{2 + \frac{1}{1+\frac{1}{\frac{3}{2}}}} \tag{4.50}$$

The decomposition into a continued fraction now terminates, since

$$\frac{3}{2} = 1 + \frac{1}{2} \tag{4.51}$$

may be written with a 1 in the numerator without any need to invert, giving a final continued fraction representation of 31/13 as

$$\frac{31}{13} = 2 + \frac{1}{2 + \frac{1}{1+\frac{2}{3}}} = 2 + \frac{1}{2 + \frac{1}{1+\frac{1}{\frac{1}{1+\frac{1}{2}}}}} \tag{4.52}$$

It's clear that the continued fractions algorithm terminates after a finite number of 'split and invert' steps for any rational number, since the numerators which appear $(31, 5, 3, 2, 1$ in the example) are strictly decreasing. How quickly does this termination occur? It turns out that if $\varphi = s/r$ is a rational number, and s and r are L bit integers, then the continued fraction expansion for φ can be computed using $O(L^3)$ operations - $O(L)$ 'split and invert' steps, each using $O(L^2)$ gates for elementary arithmetic.

Algorithm: Quantum order-finding

Inputs: (1) A black box $U_{x,N}$ which performs the transformation $|j\rangle|k\rangle \rightarrow |j\rangle|x^j k (\mathrm{mod} N)\rangle$, for x co-prime to the L-bit number N,
(2) $t = 2L + 1 + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right)\right\rceil$ qubits intialized to $|0\rangle$, an
(3) L qubits initialized to the state $|1\rangle$.

Outputs: The least integer $r > 0$ such that $x^r = 1 (\mathrm{mod} N)$.

Runtime: $O(L^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. $|0\rangle|1\rangle$ — initial state
2. $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$ — create superposition
3. $\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j (\mathrm{mod} N)\rangle$ — apply $U_{x,N}$

4. $\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi isj/r} |j\rangle |u_s\rangle$

5. $\rightarrow \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\overline{s/r}\rangle |u_s\rangle$ apply inverse Fourier transform to first register

6. $\rightarrow \overline{s/r}$ measure first register

7. $\rightarrow r$ apply continued fractions algorithm

4.3.3 Performance

How can the order-finding algorithm fail? There are two possibilities. First, the phase estimation procedure might produce a bad estimate to s/r. This occurs with probability at most ϵ, and can be made small with a negligible increase in the size of the circuit. More seriously, it might be that s and r have a common factor, in which case the number r' returned by the continued fractions algorithm be a factor of r, and not r itself. Fortunately, there are at least three ways around this problem.

The most straightforward way is to note that for randomly chosen s in the range 0 through $r-1$, it's actually pretty likely that s and r are co-prime, in which case the continued fractions algorithm must return r. Note that the number of prime numbers less than r is at least $r/2\log r$, and thus the chance that s is prime (and therefore, co-prime to r) is at least $1/2\log(r) > 1/2\log(N)$. Thus, repeating the algorithm $2\log(N)$ times we will, with high probability, observe a phase s/r such that s and r are co-prime, and therefore the continued fractions algorithm produces r, as desired.

A second method is to note that if $r' \neq r$, then r' is guaranteed to be a factor of r, unless $s = 0$, which possibility occurs with probability $1/r \leq 1/2$, and which can be discounted further by a few repetitions. Suppose we replace a by $a' \equiv a^{r'} \pmod N$. Then the order of a' is r/r'. We can now repeat the algorithm, and try to compute the order of a, which, if we succeed, allows us to compute the order of a, since $r = r'r/r'$. If we fail, then we obtain r'' which is a factor of r/r', and we now try to compute the order of $a'' \equiv (a')^{r''} \pmod N$. We iterate this procedure until we determine the order of a. At most $\log(r) = O(L)$ iterations are required, since each repetition reduces the order of the current candidate a'' by a factor of at least two.

The third method is better than the first two methods, in that it requires only a constant number of trials, rather than $O(L)$ repetitions. The idea is to repeat the phase estimation-continued fractions procedure twice, obtaining r_1', s_1' the first time, and r_2', s_2' the second time. Provided s_1', s_2' have no common factors, r may be extracted by taking the least common multiple of r_1 and r_2. The probability that s_1' and s_2' have no common factors is given by

$$1 - \sum_q p(q|s_1')p(q|s_2'), \tag{4.53}$$

where the sum is over all prime numbers q, and $p(x|y)$ here means the probability of x dividing y. If q divides s_1' then it must also divide the true value of s, s_1, on the

first iteration, so to upper bound $p(q|s_1^{'})$ it suffices to upper bound $p(q|s_1)$, where s_1 is chosen uniformly at random from 0 through $r-1$. It is easy to see that $p(q|s_1) \leq 1/q$, and thus $p(q|s_1^{'}) \leq 1/q$. Similarly,$p(q|s_2^{'}) \leq 1/q$, and thus the probability that $s_1^{'}$ and $s_2^{'}$ have no common factors satisfies

$$1 - \sum_q p(q|s_1^{'})p(q|s_2^{'}) \geq 1 - \sum_q \frac{1}{q^2} \tag{4.54}$$

The right hand side can be upper bounded in a number of way, a simple technique gives,

$$1 - \sum_q p(q|s_1^{'})p(q|s_2^{'}) \geq \frac{1}{4} \tag{4.55}$$

and thus the probability of obtaining the correct r is at least $1/4$.

What resource requirements does this algorithm consume? The Hadamard transform requires $O(L)$ gates, and the inverse Fourier transform requires $O(L^2)$ gates. The major cost in the quantum circuit proper actually comes from the modular exponentiation, which uses $O(L^3)$ gates, for a total of $O(L^3)$ gates in the quantum circuit proper. The continued fractions algorithm adds $O(L^3)$ more gates, for a total of $O(L^3)$ gates to obtain $r^{'}$. Using the third method for obtaining r from $r^{'}$ we need only repeat this procedure a constant number of times to obtain the order, r, for a total cost of $O(L^3)$.

4.4 Shor's Algorithm

Shor's algorithm is famous for factoring integers in polynomial time. Since the best-known classical algorithm requires superpolynomial time to factor the product of two primes, the widely used cryptosystem, RSA, relies on factoring being impossible for large enough integers.

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission. The acronym "RSA" comes from the surnames of Ron Rivest, Adi Shamir and Leonard Adleman, who publicly described the algorithm in 1977. The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem". The way RSA works is, each user first chooses two extremely large prime numbers p and q, and sets $n = pq$. Knowing the factorization of n, it is easy to compute $\varphi(n) = (p-1)(q-1) = n + 1 - p - q$. Next, the user randomly chooses an integer e between 1 and $\varphi_(n)$, which is prime to $\varphi_(n)$

4.4.1 Reduction to Order-finding

Shor gave an algorithm to find the prime factors of a large number N. Theis *factoring problem* turns out to be equivalent to the order-finding problem discussed above, in the

sense that a fast algorithm for order-finding can easily be turned into a fast algorithm for factoring.

Theorem 1. *Suppose N is a composite number L bits, and x is a non-trivial solution to the equation $x^2 = 1(\mathrm{mod}N)$ in the range $1 \leq x \leq N$, that is, neither $x = 1(\mathrm{mod}\ N)$ nor $x = N-1 = -1(\mathrm{mod}\ N)$. Then at least one of $\gcd(x-1, N)$ and $\gcd(x+1, N)$ is a non-trivial factor of N that can be computed using $O(L^3)$ operations.*

Proof. Since $x^2 = 1(\mathrm{mod}N)$, it must be that N divides $x^2 - 1 = (x+1)(x-1)$, and thus N must have a common factor with one or the other of $(x+1)$ and $(x-1)$. But $1 < x < N-1$ by assumption, so $x - 1 < x + 1 < N$, from which we see that common factor cannot be N itself. Using Euclid's Algorithm we may compute $\gcd(x-1, N)$ and $\gcd(x+1, N)$ and thus obtain a non-trivial factor of N, using $O(L^3)$ operations. □

Theorem 2. *Let p be an odd prime. Let 2^d be the largest power of 2 dividing $\varphi(p^\alpha)$. Then with probability eactly one-half 2^d divides the order modulo p^α of a randomly chisen element of $Z^*_{p^\alpha}$*

Theorem 3. *Suppose $N = p_1^{\alpha_1} \ldots p_m^{\alpha_m}$ is the prime factorization of an odd composite positive integer. Let x be chosen uniformly at random from Z^*_N, and let r be the order of x, modulo N. Then*

$$p(r \text{ is even and } x^{r/2} \neq -1(\mathrm{mod}N)) \geq 1 - \frac{1}{2^m} \tag{4.56}$$

Proof. We show that

$$p(r \text{ is odd or } x^{r/2} = -1(\mathrm{mod}N)) \leq 1 - \frac{1}{2^m} \tag{4.57}$$

By the Chinese remainder theorem, choosing x uniformly at random from Z^*_N is equivalent to choosing x_j independently and uniformly at random from $Z^*_{p_j^{\alpha_j}}$ and requiring that $x = x_j \mod p_j^{\alpha_j})$ for each j. Let r_j be the order of x_j modulo $p_j^{\alpha_j}$. Let 2^{d_j} be the largets power of 2 that divides r_j and 2^d be the largest power of 2 that divides r. We will show that to have r odd or $x^{r/2} = -1(\mathrm{mod}N)$ it is necessary that d_j takes the same value for all values of j. The result then follows, as from Theorem 2, the probability of this occurring is at most $1/2^m$.

The first case we consider is when r is odd. It is easy to see that $r_j | r$ for each j, and therefore r_j is odd, so $d_j = 0$ for all $i = 1, \ldots, k$. The second and final case is when r is even and $x^{r/2} = -1(\mathrm{mod}N)$. Then $x^{r/2} = -1(\mathrm{mod}p_j^{\alpha_j})$, so $r_j \nmid (r/2)$. Since $r_j | r$ we must have $d_j = d$ for all j. □

The reduction of factoring to order-finding proceeds in two steps. The first step is to show that we can compute a fcator of N if we can find a npn-trivial solution $x \neq \pm 1(\mathrm{mod}N)$ to the equation $x^2 = 1(\mathrm{mod}N)$. The second step is to show that a randomly chosen y co-prime to N is quite likely to have an order r which is even, and such that $y^{r/2} \neq \pm 1(\mathrm{mod}N)$, and thus $x \equiv y^{r/2}(\mathrm{mod}N)$ is a non-trivial solution to $x^2 = 1(\mathrm{mod}N)$. These two steps are embodied in the three theorems above.

4.4.2 The Algorithm

Given an integer N, the goal is to find its prime factors p and q. We first take a guess number say a, then we proceed with the following possibilities:

Case 1. *If a is a factor of N, then we can find other factor using N/a, i.e,*

$$p = a \text{ and } q = \frac{N}{a} \tag{4.58}$$

Case 2. *Suppose a is not a factor of N, but a shares a factor with N. Then in this case we find the* $\gcd(N, a)$ *which is one of the factor, and the other factor is obtained by dividing N by* $\gcd(N, a)$

$$p = \gcd(N, a) \text{ and } q = \frac{N}{\gcd(N, a)} \tag{4.59}$$

Case 3. *If a and N have no common factor, then we find r such that*

$$a^r \equiv 1(\mathrm{mod} N) \tag{4.60}$$

then

$$(a^r - 1) \equiv 0(\mathrm{mod} N) \tag{4.61}$$

from which we get

$$(a^r - 1) = (a^{r/2} + 1)(a^{r/2} - 1) \tag{4.62}$$

which gives the two prime factors of N.

The question is how to find $'r'$? Shor's solution was to use quantum phase estimation on the unitary operator:

$$U |y\rangle \equiv |ay \ (\mathrm{mod} N)\rangle \tag{4.63}$$

To see how this is helpful, let's work out what an eigenstate of U might look like. If we started in the state $|1\rangle$, we can see that each successive application of U will multiply the state of our register by $a(\mathrm{mod} N)$, and after r applications we will arrive at the state $|1\rangle$ again. For example with $a = 3$ and $N = 35$:

$$\begin{aligned} U |1\rangle &= |3\rangle \\ U^2 |1\rangle &= |9\rangle \\ U^3 |1\rangle &= |27\rangle \\ &\vdots \\ U^{r-1} |1\rangle &= |12\rangle \\ U^r |1\rangle &= |1\rangle \end{aligned}$$

So a superposition of the states in this cycle ($|u_0\rangle$) would be an eigenstate of U:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |a^k \ (\mathrm{mod} N)\rangle \tag{4.64}$$

This eigenstate has an eigenvalue of 1, which isn't very interesting. A more interesting eigenstate could be one in which the phase is different for each of these computational basis states. Specifically, if we look at the case in which the phase of the k-th state is proportional to k:

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i k}{r}} |a^k \ (\mathrm{mod} N)\rangle \tag{4.65}$$

$$U|u_1\rangle = e^{\frac{2\pi i}{r}} |u_1\rangle \tag{4.66}$$

This is a particularly interesting eigenvalue as it contains r. In fact, r has to be included to make sure the phase differences between the r computational basis states are equal. This is not the only eigenstate with this behaviour; to generalise this further, we can multiply an integer, s, to this phase difference, which will show up in our eigenvalue:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} |a^k \ (\mathrm{mod} N)\rangle \tag{4.67}$$

$$U|u_s\rangle = e^{\frac{2\pi i s}{r}} |u_s\rangle \tag{4.68}$$

We now have a unique eigenstate for each integer value of s where

$$0 \leq s \leq r-1$$

Very conveniently, if we sum up all these eigenstates, the different phases cancel out all computational basis states except $|1\rangle$:

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle \tag{4.69}$$

Since the computational basis state $|1\rangle$ is a superposition of these eigenstates, which means if we do QPE on U using the state $|1\rangle$, we will measure a phase:

$$\phi = \frac{s}{r} \tag{4.70}$$

Where s is a random integer between 0 and $r-1$. We finally use the continued fractions algorithm on ϕ to find r.

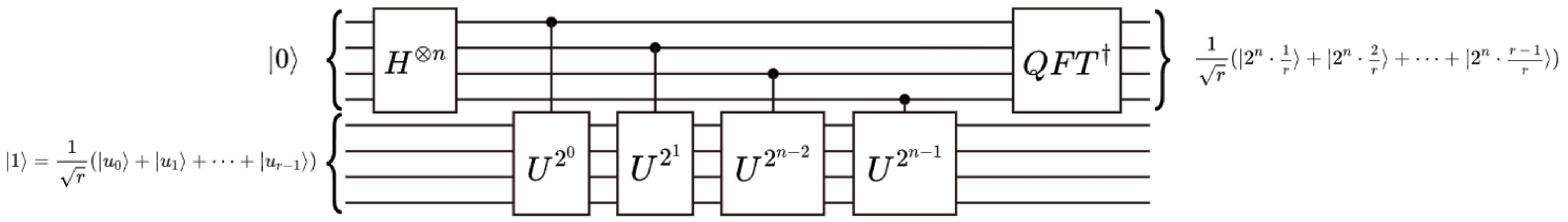


Figure 4.5: Circuit diagram for factoring using Shor's Algorithm

Algorithm: Reduction of factoring to order-finding

Inputs: A composite number N.

Outputs: A non-trivial factor of N.

Runtime: $O((\log N)^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. If N is even, return the factor 2.
2. Determine whether $N = a^b$ for integers $a \geq 1$ and $b \geq 2$, and if so return the factor a
3. Randomly choose x in the range 1 to $N - 1$. If $\gcd(x, N) > 1$ then return the factor $\gcd(x, N)$.
4. Use the order-finding subroutine to find the order r of x modulo N.
5. If r is even and $x^{r/2} \neq -1 \mod N)$ then compute $\gcd(x^(r/2) - 1, N)$ and $\gcd(x^(r/2) + 1, N)$ and test to see if one of these is a non-trivial factor, returning that factor if so. Otheriwse, the algortihm fails.

The above procedure is the general approach for Shor's Algorithm. For mathematically solving we are going to use the method described in the paper [4], Secton 3 and [5], Section 7. These methods are identical to the general method described above, but they are easier to understand when mathematically factoring a number N

1. We consider $t = \lceil \log_2 N \rceil$ as the number of qubits in the first register. We must choose t such that 2^t is always between N^2 and $2N^2$.
2. The size of the second register $M = \lfloor \log_2 N \rfloor + 1$.
3. Initially we start in the state
$$|0\rangle |0\rangle \tag{4.71}$$
4. Applying Hadamard gate on the first register we get,
$$\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |0\rangle \tag{4.72}$$
Let $q = 2^t$, then we have
$$\frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle |0\rangle \tag{4.73}$$
5. Applying $U_{x,N}$ we get
$$\frac{1}{\sqrt{q}} \sum_{j=0}^{q-1} |j\rangle |x^j (\mathrm{mod} N)\rangle \tag{4.74}$$
6. Measure register 2, then we will be left in superpositions of that number. The state of $|j\rangle$ will collapse into the reimage of $f(j_0)$. As f is periodic, the reimage of $f(j_0)$ is $\left\{j_0,\ j_0 + P,\ j_0 + 2P, \ldots, \left(\frac{n}{r} - 1\right) P\right\}$
$$\sqrt{\frac{p}{q}} \sum_{s=0}^{q-1} |j_0 + sp\rangle |f(j_0)\rangle \tag{4.75}$$
7. Applying QFT Inverse to register 1 we get,
$$\frac{1}{\sqrt{p}} \sum_{s=0}^{q-1} |s\frac{q}{p}\rangle \phi_i \tag{4.76}$$
where ϕ_i is some unimportant phase associated with each term due to linear shift j_0
8. Measuring the first register gives us $s\frac{q}{p}$, where s main contribution(peak value in probability graph) is from 0 to $p - 1$. From which we can find p and hence factorize the number.

4.4.3 Example: Factoring N=15

Suppose we want to find prime factors m *and* p of $N = mp = 15$. We will proceed with the method described in the box above.

1. Let us guess a number, say g, which lies betweeen 1 and $N - 1$.
2. The possible choices for g are $\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$.
3. Consider $g \in \{3, 5\}$

$$\textit{Suppose } g = m = 3, \textit{ then } p = \frac{N}{m} = \frac{15}{3} = 5$$

4. Consider $g \in \{6, 9, 10, 12\}$ which share a factor with N

$$\textit{Suppose } g = 9, \textit{ then } m = \gcd(N, g) = \gcd(15, 9) = 3$$

$$p = \frac{N}{m} = \frac{15}{3} = 5$$

   Simillarly, we can show the same for other elements too.

5. Consider $g \in \{2, 4, 7, 8, 11, 13, 14\}$, which are all co-prime to N.

   Suppose $g = 7$, then we proceed with Shor's Algorithm.

   (a) We must choose t such that 2^t is between N^2 and $2N^2$. Since $N^2 = 225$ and $2N^2 = 450$, the only number betweeen then, which is a power of 2, is 256. Since $q = 2^t$, we have $q = 256 = 2^8$.

   (b) We start in the state

$$|\psi_0\rangle = |0\rangle\,|0\rangle$$

   (c) Applying Hadamard gate on the first register.

$$\begin{aligned} |\psi_1\rangle &\rightarrow \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle\,|0\rangle \\ &\rightarrow \frac{1}{\sqrt{256}} \sum_{j=0}^{255} |j\rangle\,|0\rangle \end{aligned}$$

   (d) Next we apply $U_{x,N}$ on the second register, controlled by first register;

$$|\psi_2\rangle \rightarrow \frac{1}{\sqrt{256}} \sum_{j=0}^{255} |j\rangle\,|x^j \mod 15\rangle$$

$$|\psi_2\rangle = \frac{1}{\sqrt{256}}\Big[\,|0\rangle\,|7^0 \mod 15\rangle + |1\rangle\,|7^1 \mod 15\rangle + |2\rangle\,|7^2 \mod 15\rangle + |3\rangle\,|7^3 \mod 15\rangle + |4\rangle\,|7^4 \mod 15\rangle + |5\rangle\,|7^5 \mod 15\rangle + |6\rangle\,|7^6 \mod 15\rangle + |7\rangle\,|7^7 \mod 15\rangle + |8\rangle\,|7^8 \mod 15\rangle + |9\rangle\,|7^9 \mod 15\rangle + |10\rangle\,|7^{10} \mod 15\rangle + |11\rangle\,|7^{11} \mod 15\rangle + |12\rangle\,|7^{12} \mod 15\rangle + |13\rangle\,|7^{13} \mod 15\rangle + |14\rangle\,|7^{14} \mod 15\rangle + |15\rangle\,|7^{15} \mod 15\rangle + \cdots\Big]$$

$$|\psi_2\rangle = \frac{1}{\sqrt{256}}\Big[\,|0\rangle\,|1\rangle + |1\rangle\,|7\rangle + |2\rangle\,|4\rangle + |3\rangle\,|13\rangle + |4\rangle\,|1\rangle + |5\rangle\,|7\rangle + |6\rangle\,|4\rangle + |7\rangle\,|13\rangle + |8\rangle\,|1\rangle + |9\rangle\,|7\rangle + |10\rangle\,|4\rangle + |11\rangle\,|13\rangle + |12\rangle\,|1\rangle + |13\rangle\,|7\rangle + |14\rangle\,|4\rangle + |15\rangle\,|13\rangle + \cdots\Big]$$

$$|\psi_2\rangle = \frac{1}{\sqrt{256}}\begin{bmatrix} \big(|0\rangle + |4\rangle + |8\rangle + |12\rangle + \cdots\big)\,|1\rangle + \\ \big(|1\rangle + |5\rangle + |9\rangle + |13\rangle + \cdots\big)\,|7\rangle + \\ \big(|2\rangle + |6\rangle + |10\rangle + |14\rangle + \cdots\big)\,|4\rangle + \\ \big(|3\rangle + |7\rangle + |11\rangle + |15\rangle + \cdots\big)\,|13\rangle \end{bmatrix}$$

(e) Now we measure the second register and we will get one result from 1, 4, 7, 13. Supppose we get 4 as the output, so we will be left in superposition of just those states having 4 in the second register.

$$|\psi_3\rangle = \sqrt{\frac{4}{256}}\big[\,\big(|2\rangle + |6\rangle + |10\rangle + |14\rangle + \ldots\big)\,\big]$$

$$|\psi_3\rangle = \sqrt{\frac{4}{256}}\sum_{a=0}^{255}|4a+2\rangle$$

(f) Now if we compare $|\psi_3\rangle$ obtained above with Equation(4.75) we can observe we get the desired form to apply Inverse QFT.

(g) Now when we apply Inverse QFT on register 1, we get the following (refer to Equation(4.76))

$$|\psi_4\rangle = \frac{1}{\sqrt{4}}\sum_{s=0}^{255}|s\frac{256}{4}\rangle\,\phi_i$$

(h) The major contribution of s in $\frac{256s}{4}$ will be for value of s between 0 and $r-1$. That is we have the peak value as $0, 64, 128, 192$.

(i) Suppose we get 192 with highest probability, then we apply continued fraction on

$$\begin{aligned}\frac{192}{256} &= 0 + \frac{3}{4} \\ &= 0 + \frac{1}{4/3} \\ &= 0 + \frac{1}{1 + \frac{1}{3}} \\ &= \frac{1}{1 + \frac{1}{3}}\end{aligned}$$

From which we get $p = 4$. Simillarly, with other values we obtain $p = 4$.

(j) So the numbers are $7^{4/2} \pm 1$. The numbers are $7^2 + 1 = 50$ and $7^2 - 1 = 48$. Both the numbers are not factors of 15, but they share a factor with 15.

$$\gcd(50, 15) = 5$$

$$\gcd(48, 15) = 3$$

(k) So the prime factors are 3 and 5.

Chapter 5

Factoring $N = 21$ using qubit recycling method

The first goal of this project is to factorize 21 using Qubit recycling method. Our demonstration uses a compiled version of the Quantum Phase Estimation, using a configuration of approximate Tofoli gates with residual phase shifts, which preserves the functional correctness and allows us to achieve a complete factoring of $N = 21$. We have presented the factorization build on the idea provided in the paper [6], [7], [8] respectively.

We build on the order-finding routine of [7] and implement a version of Shor's Algorithm for factoring 21 using 5 qubits - 2 qubits in work register and 3 qubits in control register, each providing 1-bit of accuracy in the resolution of the peaks in the output probability distribution used to find the order. This approach is in contrast with the iterative approach, which employs a single qubit that is recycled through measurement and feed-forward, giving 1-bit of accuracy each time it is recycled. The advantage of using iterative method is that, through mid-circuit measurement and realtime conditional feed-forward operations, the total number of qubits required by the algorithm is signifcantly reduced. At the time of writing, IBM's quantum processors do not yet support real-time conditionals necessary for the implementation of the iterative approach, so we use 3 qubits for the control register, one for each effective iteration.

The number of controlled-NOT gates exceeds more than 40 in factoring 21 using the standard algorithm. The improved version uses relative Toffoli gates to reduce the count of CX gates by half while leaving the overall operation of the circuit unchanged.

5.1 Compiled version of Shor's Algorithm

To factorize a number N using standard approach of Shor's algorithm, we need $72L^3$ quantum gates and $5L+1$ qubits to prepare the quantum circuit, where $L = \lfloor \log_2(N) \rfloor$. That is, to factorize 21 we need 9000 elementary quantum gates and 26 qubits, which

is not possible on today's hardware. So we use compilation techniques to make the quantum circuit scalable of today's quantum hardware. The quantum circuits are tailored around properties of the number to be factorized.

We take our guess number a as 4, which is co-prime to 21, to factorize 21. We can easily observe

$$\begin{aligned} U\left|4\right\rangle &= \left|4\right\rangle (\mathrm{mod} 21) \\ U^2\left|4\right\rangle &= \left|16\right\rangle (\mathrm{mod} 21) \\ U^3\left|4\right\rangle &= \left|64\right\rangle = \left|1\right\rangle (\mathrm{mod} 21) \\ U^4\left|4\right\rangle &= \left|4\right\rangle (\mathrm{mod} 21) \\ U^5\left|4\right\rangle &= \left|16\right\rangle (\mathrm{mod} 21) \\ U^6\left|4\right\rangle &= \left|1\right\rangle (\mathrm{mod} 21) \\ &\vdots \end{aligned} \tag{5.1}$$

that $|1\rangle$, $|4\rangle$ and $|16\rangle$ are the only possible levels are implementation of U.

Hence in Ref[6], the compilation is demonstarted in mapping the three levels $|1\rangle$, $|4\rangle$ and $|16\rangle$ accessed by the possible $2^L = 2^5$ levels of the work-register to only a single qutrit system. Since IBM processors consist of qubits, we represent the the work register by three basis states from a two-qubit system and discard the fourth basis state as a null state. The states encoding the three possible levels of the work register; $|1\rangle$, $|4\rangle$ and $|16\rangle$ are mapped to $|q_0 q_1\rangle$ according to

$$\begin{aligned} \left|1\right\rangle &\rightarrow \left|\log_4 1\right\rangle = \left|00\right\rangle \\ \left|4\right\rangle &\rightarrow \left|\log_4 4\right\rangle = \left|01\right\rangle \\ \left|16\right\rangle &\rightarrow \left|\log_4 16\right\rangle = \left|10\right\rangle \end{aligned} \tag{5.2}$$

Here instead of evaluating 4^x mod 21 (i.e a^x mod N), as we do in Shor's Algorithm, we instead evaluate $\log_4[4^x \bmod 21]$ in its place for $x = 0, 1 \ldots 2^3 - 1$. This reduces the size of the work register to 2 qubits, instead of 5 qubits required in the standard construction. The ordering of quantum bits in the work register is $|q\rangle = |q_0\rangle\,|q_1\rangle$, where the rightmost qubit is associated with the least signifcant bit. Similarly, with the control register we have $|c\rangle = |c_0\rangle\,|c_1\rangle\,|c_2\rangle$. In total the algorithm requires 5 qubits: 3 for the control register and 2 for the work register.

Implementing the controlled unitaries U^x that perform the modular exponentiation $|x\rangle\,|y\rangle \rightarrow |x\rangle\,U^x\,|y\rangle = |x\rangle\,|a^x y \bmod N\rangle$ reduces to effectively swapping around the states $|1\rangle$, $|4\rangle$ and $|16\rangle$ in the work register controlled by the corresponding bit of the integer x in the control register, which is given by $x = c_2 2^0 + c_1 2^1 + c_0 2^2$. In other words, $U^x = U^{c_0 2^2} U^{c_1 2^1} U^{c_2 2^0}$. Thus depending on the control qubit c_i one of the following maps is applied:

$$\begin{aligned} U^1 &: \left\{\left|1\right\rangle \rightarrow \left|4\right\rangle, \left|4\right\rangle \rightarrow \left|16\right\rangle, \left|16\right\rangle \rightarrow \left|1\right\rangle\right\} \\ U^2 &: \left\{\left|1\right\rangle \rightarrow \left|16\right\rangle, \left|4\right\rangle \rightarrow \left|1\right\rangle, \left|16\right\rangle \rightarrow \left|4\right\rangle\right\} \\ U^4 &: \left\{\left|1\right\rangle \rightarrow \left|4\right\rangle, \left|4\right\rangle \rightarrow \left|16\right\rangle, \left|16\right\rangle \rightarrow \left|1\right\rangle\right\} \end{aligned} \tag{5.3}$$

The above mappings are obtained through $|y\rangle \rightarrow |a^x y \bmod N\rangle$. **Consider for U^1 where $x = 1$.**
Now if $y = 1$,

$$|1\rangle \rightarrow |4^1 \times 1 \bmod 21\rangle \rightarrow |4\rangle\,. \tag{5.4}$$

If $y = 4$

$$|4\rangle \rightarrow |4^1 \times 4 \bmod 21\rangle \rightarrow |16\rangle\,. \tag{5.5}$$

If $y = 16$

$$|16\rangle \rightarrow |4^1 \times 16 \bmod 21\rangle \rightarrow |64\rangle \rightarrow |1 \bmod 21\rangle \rightarrow |1\rangle\,. \tag{5.6}$$

Now consider for U^2 where $x = 2$.
If $y = 1$

$$|1\rangle \rightarrow |4^2 \times 1 \bmod 21\rangle \rightarrow |16\rangle\,. \tag{5.7}$$

If $y = 4$

$$|4\rangle \rightarrow |4^2 \times 4 \bmod 21\rangle \rightarrow |64\rangle \rightarrow |1\rangle\,. \tag{5.8}$$

If $y = 16$

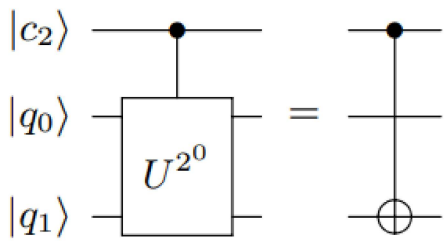
$$|16\rangle \rightarrow |4^2 \times 16 \bmod 21\rangle \rightarrow |256\rangle \rightarrow |4\rangle\,. \tag{5.9}$$

Simillarly we can show the mapping for U^4 too.

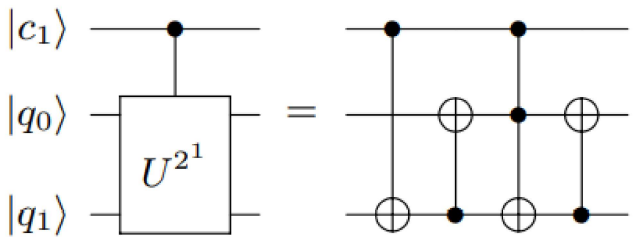
The next simplifcation step comes from the fact that these operations on the work register need not be controlled SWAP(Fredkin) gates, they can be as simple as CX gates, as we show next.

5.2 Building the Modular exponentiation circuit

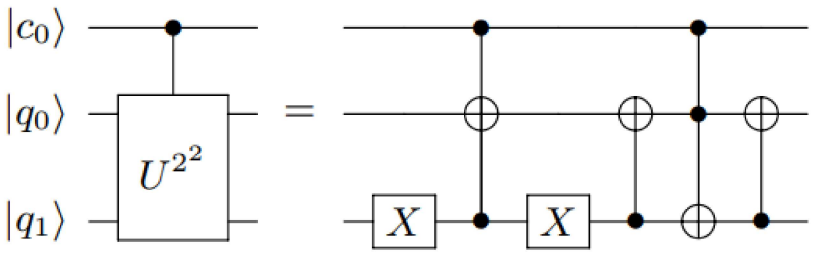
Implementing U^1 on the two-qubit work register is simplifed considerably by noting that the states $|4\rangle$ and $|16\rangle$ initially have zero amplitude, and thus the operation $|1\rangle \rightarrow |4\rangle$ alone is sufficent. This operation can realized with a CX gate controlled by $|c_2\rangle$ targeting the second work qubit $|q_1\rangle$.



Simillarly, the implementation of U^2 can be simplifed by noting that the states $|1\rangle$ and $|4\rangle$ are the only non-zero amplitude states in the work register after applying U^1, thus we consider only $|1\rangle \rightarrow |4\rangle$ and $|4\rangle \rightarrow |16\rangle$. A CX gate controlled by $|c_1\rangle$ targeting $|q_1\rangle$ followed by a Fredkin gate, swapping $|q_0\rangle$ and $|q_1\rangle$ realizes this simplied U^2.



The subsequent implementation of U^4 admits no simplifications as all the possible states in the work register may have non-zero amplitude at this point. Tis operation is implemented with a Tofoli and a Fredkin gate with single-qubit X gate



The complete circuit diagram is shown below

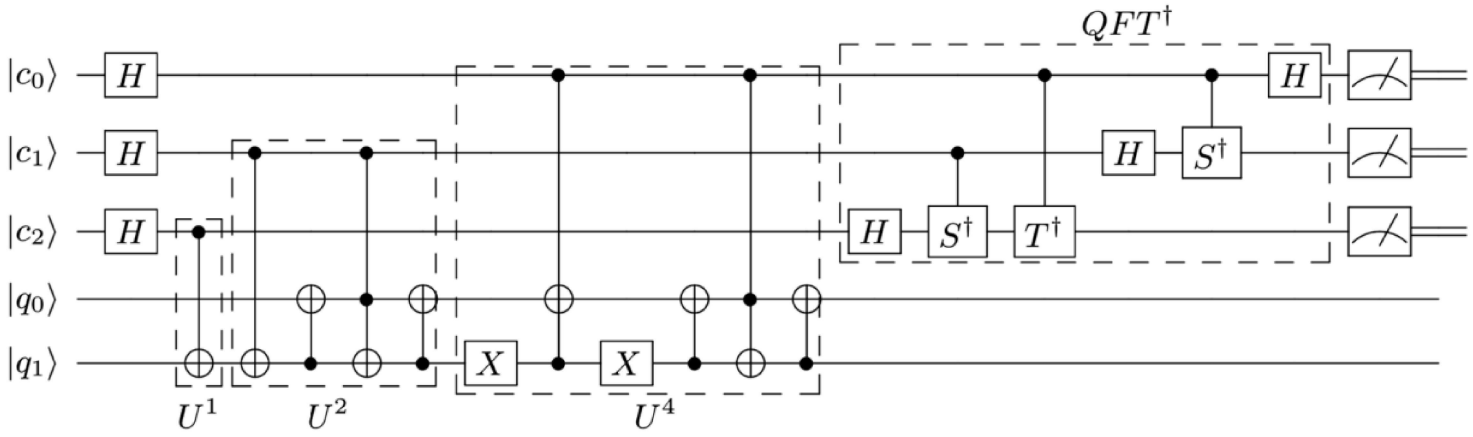


Figure 5.1: Compiled quantum order-fnding routine for $N = 21$ and $a = 4$. This circuit uses fve qubits in total; 3 for the control register and 2 for the work register. The above circuit determines $2^n s/r$ to three bits of accuracy, from which the order can be extracted.

5.3 Implementation on Qiskit

We have the circuit for factorizing 21 using Qubit recycling method. Now we will implement and execute on the circuit on a quantum simulator as well as on a real IBM quantum computer. We will also compare the output received from both and see to what accuracy the real quantum computer produces the output.

Importing libraries

The following are the necessary libraries required to execute the code.

```
  import matplotlib.pyplot as plt
  import numpy as np
  from qiskit import IBMQ, QuantumCircuit, Aer, transpile, assemble
  from qiskit.visualization import plot_histogram
  from math import gcd
  from numpy.random import randint
  import pandas as pd
  from qiskit.providers.ibmq import least_busy
  from fractions import Fraction
```

Inverse Quantum Fourier Transform

```
# building the Inverse QFT
def qft_inv(n):
    qc = QuantumCircuit(n)
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT_INV"
    return qc
```

The circuit

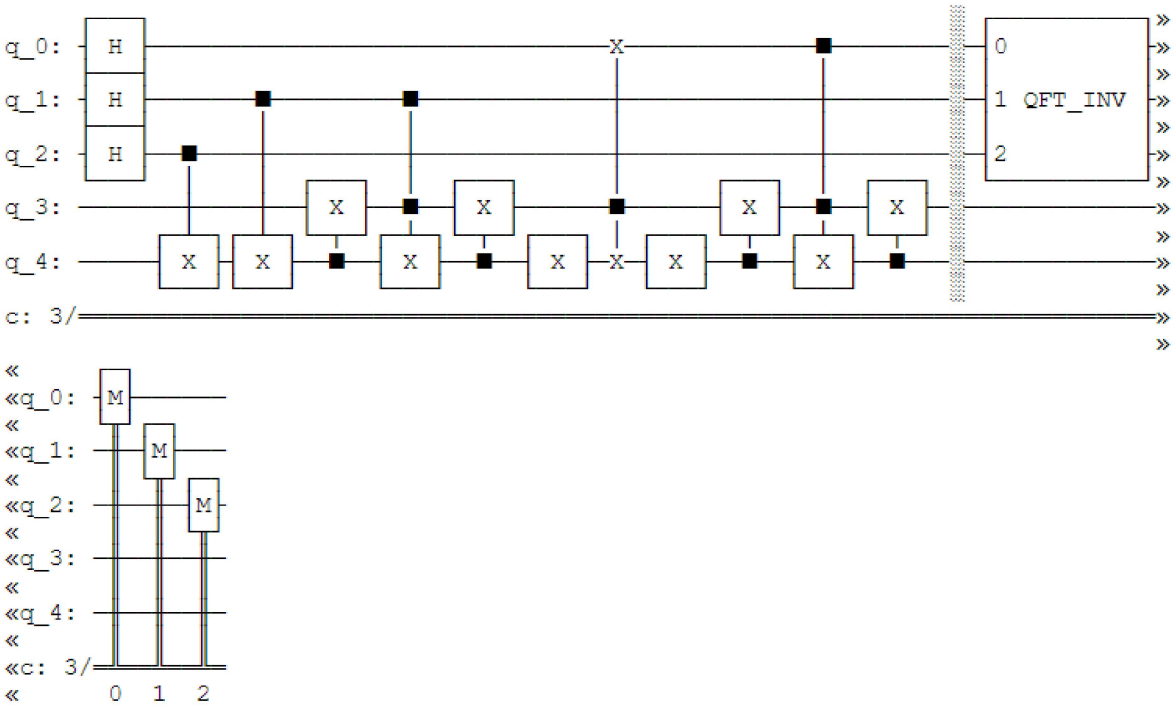
```
n_count = 3 #3 classical bit to measure the ouput
n = 5 #5 qubits as mentioned above
qc = QuantumCircuit(n_count+2, n_count)
# applying Hadamard gate to each control register
for q in range(n_count):
    qc.h(q)

#building the modular exponentiation circuit as mentioned above
qc.cx(2,4)

```

```
qc.cx(1,4)
qc.cx(4,3)
qc.ccx(1,3,4)
qc.cx(4,3)

qc.x(4)
qc.cswap(3,0,4)
qc.x(4)
qc.cx(4,3)
qc.ccx(0,3,4)
qc.cx(4,3)
qc.barrier()
# applying the inverse QFT
qc.append(qft_inv(n_count), range(n_count))
# measuring the ouput on the classical bits
qc.measure(range(n_count), range(n_count))
#drawing the circuit
qc.draw()
```

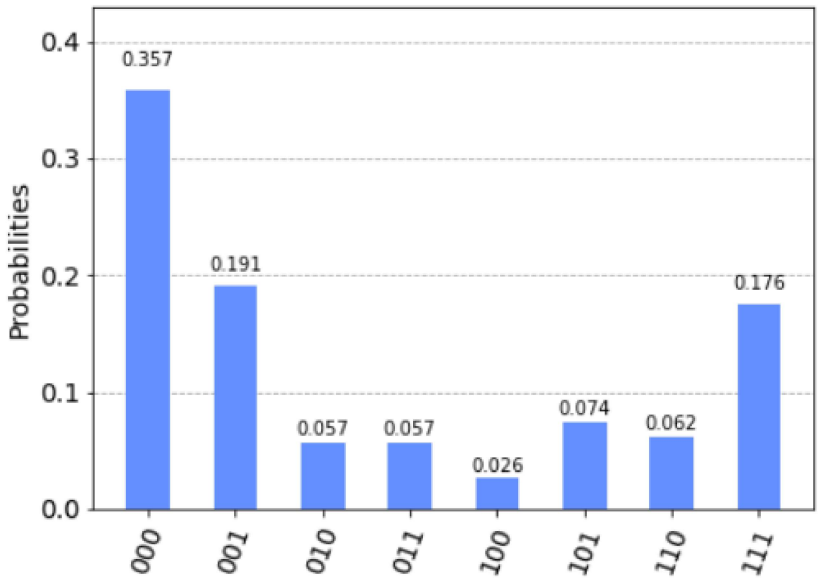


5.3.1 Running on a simulator

We have prepared our quantum circuit using compiled Shor's Algorithm to factor 21. Firstly, we will run our circuit on a simulator and calculate the phase from the probability of each state. Applying continued fraction algortihm on these phases we will calculate the period r which we help us factor the number 21.

We will be using **'Aer simulator'** for this purpose.

```
aer_sim = Aer.get_backend('aer_simulator')
t_qc = transpile(qc, aer_sim)
qobj = assemble(t_qc)
results = aer_sim.run(qobj).result()
counts = results.get_counts()
plot_histogram(counts)
```



Calculating Phase

```
rows, measured_phases = [], []
for output in counts:
    decimal = int(output, 2)  # Convert (base 2) string to decimal
    phase = decimal/(2**n_count)  # Find corresponding eigenvalue
    measured_phases.append(phase)
    # Add these values to the rows in our table:
    rows.append([f"{output}(bin) = {decimal:>3}(dec)",
                 f"{decimal}/{2**n_count} = {phase:.2f}"])
# Print the rows in a table
headers=["Register Output", "Phase"]
df = pd.DataFrame(rows, columns=headers)
print(df)
```

```
       Register Output       Phase
0  101(bin) =   5(dec)  5/8 = 0.62
1  111(bin) =   7(dec)  7/8 = 0.88
2  000(bin) =   0(dec)  0/8 = 0.00
3  011(bin) =   3(dec)  3/8 = 0.38
4  001(bin) =   1(dec)  1/8 = 0.12
5  110(bin) =   6(dec)  6/8 = 0.75
6  010(bin) =   2(dec)  2/8 = 0.25
7  100(bin) =   4(dec)  4/8 = 0.50
```

Calculating r

```python
rows = []
for phase in measured_phases:
    frac = Fraction(phase).limit_denominator(15)
    rows.append([phase, f"{frac.numerator}/{frac.denominator}", frac.
   denominator])
# Print as a table
headers=["Phase", "Fraction", "Guess for r"]
df = pd.DataFrame(rows, columns=headers)
print(df)
```

```
   Phase Fraction  Guess for r
0  0.625      5/8            8
1  0.875      7/8            8
2  0.000      0/1            1
3  0.375      3/8            8
4  0.125      1/8            8
5  0.750      3/4            4
6  0.250      1/4            4
7  0.500      1/2            2
```

Finding the factors

We have obtained different values for r. Out of these, we only consider those which are even and find the factors using $a^{r/2} \pm 1$. Here we have $a = 4$ and $N = 21$.

Consider $r = 2$

$$4^{2/2} + 1 = 4^1 + 1 = 5 \tag{5.10}$$

$$4^{2/2} - 1 = 4^1 - 1 = 3 \tag{5.11}$$

We can observe that 5 is not a factor of 21 and $\gcd(5, 21) = 1$, so we discard it. But 3, a prime, is a factor of 21 and $\frac{21}{3} = 7$ which is also a prime. Hence the prime factors are 3 and 7.

Consider $r = 4$

$$4^{4/2} + 1 = 4^2 + 1 = 17 \tag{5.12}$$

$$4^{4/2} - 1 = 4^2 - 1 = 15 \tag{5.13}$$

Here we observe 17 is not a factor of 21 and also $\gcd(17, 21) = 1$. Although, 15 is not a factor of 21, but $\gcd(15, 21) = 3$ which is a prime factor of 21 and from above discussion we get 7 as the other prime factor.

Now consider $r = 8$

$$4^{8/2} + 1 = 4^4 + 1 = 257 \tag{5.14}$$

$$4^{8/2} - 1 = 4^4 - 1 = 255 \tag{5.15}$$

Here both 257 and 255 are not factors of 21 but $\gcd(255, 21) = 3$, a prime factor of 21. Accordingly, we get the other factor as 7.

Hence, in all cases of r(being even), we get the prime factors of 21 as 3 and 7.

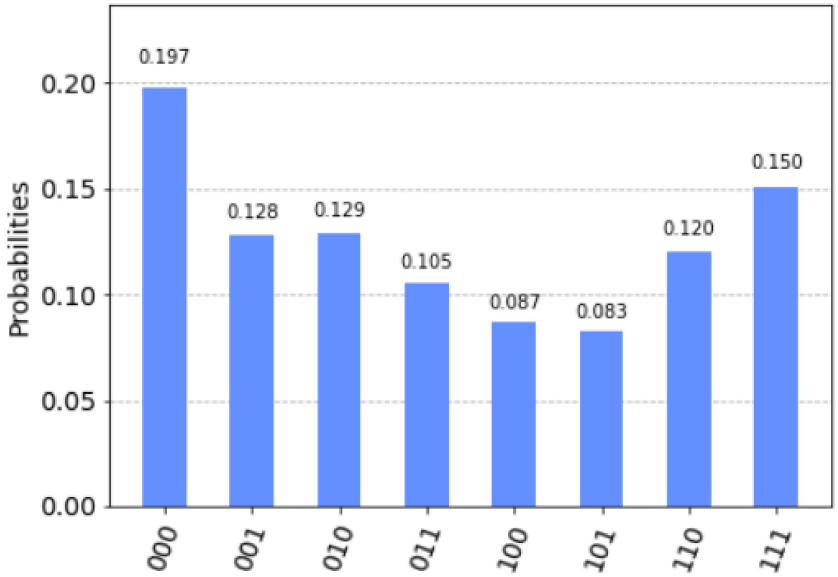
5.3.2 Running on IBM quantum computer

We repeat the procedure but this time run it on a real quantum computer.

```
# Load our saved IBMQ accounts and get the least busy backend
 device with less than or equal to 5 qubits
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x: x.
 configuration().n_qubits >= n and
                not x.configuration().simulator and x.status().
 operational==True))
print("least busy backend: ", backend)

# Execute and monitor the job
from qiskit.tools.monitor import job_monitor
shots = 1024
transpiled_simon_circuit = transpile(qc, backend,
 optimization_level=3)
qobj = assemble(transpiled_simon_circuit, shots=shots)
job = backend.run(qobj)
job_monitor(job, interval=2)
```

```
# Get results and plot counts
device_counts = job.result().get_counts()
plot_histogram(device_counts)
```



Calculating Phase

```
rows, measured_phases = [], []
for output in device_counts:
    decimal = int(output, 2)  # Convert (base 2) string to decimal
    phase = decimal/(2**n_count)  # Find corresponding eigenvalue
    measured_phases.append(phase)
# Add these values to the rows in our table:
    rows.append([f"{output}(bin) = {decimal:>3}(dec)",
                 f"{decimal}/{2**n_count} = {phase:.2f}"])
# Print the rows in a table
headers=["Register Output", "Phase"]
df = pd.DataFrame(rows, columns=headers)
print(df)
```

```
      Register Output       Phase
0  000(bin) =   0(dec)  0/8 = 0.00
1  001(bin) =   1(dec)  1/8 = 0.12
2  010(bin) =   2(dec)  2/8 = 0.25
3  011(bin) =   3(dec)  3/8 = 0.38
4  100(bin) =   4(dec)  4/8 = 0.50
5  101(bin) =   5(dec)  5/8 = 0.62
6  110(bin) =   6(dec)  6/8 = 0.75
7  111(bin) =   7(dec)  7/8 = 0.88
```

Calculating r

```
rows = []
for phase in measured_phases:
    frac = Fraction(phase).limit_denominator(15)
    rows.append([phase, f"{frac.numerator}/{frac.denominator}", frac.
   denominator])
# Print as a table
headers=["Phase", "Fraction", "Guess for r"]
df = pd.DataFrame(rows, columns=headers)
print(df)
```

```
   Phase Fraction  Guess for r
0  0.000      0/1            1
1  0.125      1/8            8
2  0.250      1/4            4
3  0.375      3/8            8
4  0.500      1/2            2
5  0.625      5/8            8
6  0.750      3/4            4
7  0.875      7/8            8
```

Finding the factors

From the above we can easily see that for all possible values of r(being even), the prime factors of 21 are 3 and 7.

Hence from both simulator and real quantum computer we get the prime factorization of 21 which are 3 and 7.

Chapter 6

Factoring $N = 51$ using Fermat primes

Earlier we showed factorization of 21 using qubit recycling, an optimal method for factorization based on Shor's algorithm. In this section, we will go through another optimal method for factoring composites N given by product of Fermat primes. We derive the circuit based on the method presented in Ref[9] and observe that factorization of 51 can be easily achieved with only 8 qubits and a modular exponentiation circuit consisting of no more than four CNOT gates.

6.1 General approach for factoring using Fermat primes

In this section we consider the application of Shor's algorithm to products of special primes of the form

$$p_k \equiv 2^{2^k} + 1 \qquad with\ k = 0, 1, 2, 3, 4. \tag{6.1}$$

Explicitly,

$$p = 3, 5, 17, 257,\ \ and\ \ 65537 \tag{6.2}$$

Fermat proposed that numbers of the form $2^{2^k} + 1$ for any $k = 0, 1, 2, \ldots$ (called Fermat numbers) are prime; however it is now know that the Fermat numbers with $5 \leq k \leq 32$ are not prime.

Products of the form

$$N = p_k p_{k'} = (2^{2^k} + 1)(2^{2^k} + 1),\ \ with\ k, k^{'} \in \{0, 1, 2, 3, 4\}\, and\ k\ \neq k^{'} \tag{6.3}$$

$$\tag{6.4}$$

$$= 15, 51, 85, 771, 1285, 4369, 196611, 327685, 1114129,\ \ and\ 16843009 \tag{6.5}$$

have the special property that the order of a mod N for every base a coprime to N is a power of 2. This follows from Euler's theorem,

$$a^{\phi(y)} \bmod y = 1, \tag{6.6}$$

where y is a positive integer, $\phi(y)$ is the number of positive integers less than y that are coprime to y, and $\gcd(a, y) = 1$. When p and $p^{'}$ are odd primes, all $pp^{'} - 1$ positive integers less than $pp^{'}$ are coprime to $pp^{'}$ except for the $p - 1$ multiples of $p^{'}$ and the $p^{'} - 1$ multiples of p, and these exceptions are distinct, so

$$\phi(pp^{'}) = pp^{'} - 1 - (p - 1) - (p^{'} - 1) = (p - 1)(p^{'} - 1) \tag{6.7}$$

Thus,

$$a^{(p-1)(p^{'}-1)} \bmod pp^{'} = 1 \tag{6.8}$$

Since order r of a mod N is the smallest positive integer x satisfying $a^x \bmod N = 1$; therefore

$$\phi(N) = (p_k - 1)(p_{k'} - 1) = 2^{2^k + 2^{k^{'}}} \tag{6.9}$$

must be a multiple of r. Because r must be an integer, we conclude that for any $1 < a < N$ with $\gcd(a, N) = 1$, r is a power of 2 as well.

6.2 Building the circuit

As discussed earlier, the general order-finding circuit for factoring contains n qubits in the first register and m qubits in the second register. After application of modular exponentiation, we apply the Inverse QFT and later measurement of the first register is done in the diagonal basis. The probability to observe the value

$$x \in \{0, 1, \ldots, 2^n - 1\} \tag{6.10}$$

is

$$prob(x) = \frac{sin^2(\pi r x A / 2^n)}{2^n A sin^2(\pi r x / 2^n)} \tag{6.11}$$

where r is the order and A is the number of distinct values of x such that $a^x \bmod N$ has the same value.As discussed in [], this probability distribution has peaks at integer values of x near

$$j \times \frac{2^n}{r} \; with \; j = 0, 1, \ldots, r - 1. \tag{6.12}$$

The number of qubits n in the first register is chosen to enable reliable extraction of the value ofrin, which depends on whether or not r is a power of 2. In actual applications of Shor's algorithm this will not be known, of course, as the point of the quantum algorithm is to determine r. In this usual situation, measurement will yield an x satisfying

$$\left| \frac{x}{2^n} - \frac{j}{r} \right| \leq \frac{1}{2^n} \; with \; j \in \{0, 1, \ldots, r - 1\} \tag{6.13}$$

By choosing $n = 2b$ qubits in the first register, where $b \equiv \lceil log_2 N \rceil$, we are guaranteed that j/r will be a convergent of $x/2^n$. However for the family of composites $N = (2^{2^k} + 1)(2^{2^{k'}} + 1)$ considered here, all bases have orders

$$r = 2^l \; with \; l \in \{1, 2, 3, \ldots, l_{max}\} \tag{6.14}$$

When N is a product of distinct odd primes, r can be as large as $\phi(N)/2$, so for an N of the form (6.3) we have the bound

$$l_{max} \leq 2^k + 2^{k'} - 1 \tag{6.15}$$

For example, in the case of $N = 51 (k = 0, k' = 2)$, the largest order is $2^4 = 16$, and the upper bound is realized.

The second register stores the values of

$$a^x \bmod N \in \{0, 1, \cdots, N-1\} \tag{6.16}$$

and therefore normally requires b qubits. However, for a given a, only r of these values are distinct. Thus we can use fewer than b qubits. The reduction amounts to computing a table of values of $a^x \bmod N$ classically for a given base a, constructing a corresponding quantum circuit, and ignoring or eliminating unused qubits in the second register.

We will adopt an equivalent—but perhaps more systematic and transparent—modular exponentiation circuit construction: We follow the output of ax mod N by a second transformation

$$\begin{aligned} 1 &\rightarrow 0 \\ a \bmod N &\rightarrow 1 \\ a^2 \bmod N &\rightarrow 2, \\ &\vdots \\ a^{r-1} \bmod N &\rightarrow r-1 \end{aligned} \tag{6.17}$$

which maps the r distinct values of $a^x \bmod N$ to $0, 1, \ldots, r-1$. We refer to this classical pre-processing of $a^x \bmod N$ as *compression*. Compression does not adversely affect the operation of the order-finding circuit, but reduces m from b to l_{max} in a systematic manner.

Any set of r distinct non-negative integers—in any order—could be used for the output of the compression map (6.17). However the choice employed here, and indicated in (6.17), is especially simple because it can be compactly written as

$$a^x \bmod N \rightarrow x \bmod r(a). \tag{6.18}$$

. Then, after changing the intial state of the second register from $|00\cdots1\rangle$ to $|00\cdots0\rangle$, we have the compressed modular exponentiation operation

$$|x\rangle \otimes |0\cdots0\rangle \rightarrow |x\rangle \otimes |x \bmod r\rangle \,. \tag{6.19}$$

The operation (6.19) without the modulo r is just the bit-wise COPY and the effect of the modulo r is to only copy the $log_2 r$ least significant bits.

In conclusion, we require l_{max} qubits in each register, for a total of $2l_{max}$ qubits. l_{max} can either be computed classically or the bound (6.15) can be used.

Here we show explicitly the case for $N = 51$. In this case $l_{max} = 4(thelargestorderis16)$, so we require $n = 4$ qubits in the first register and $m = 4$ in the second, for a total of 8 qubits. After the compression discussed above, only four different circuits are needed to cover all $N = 51$ cases, because there are four possible orders.

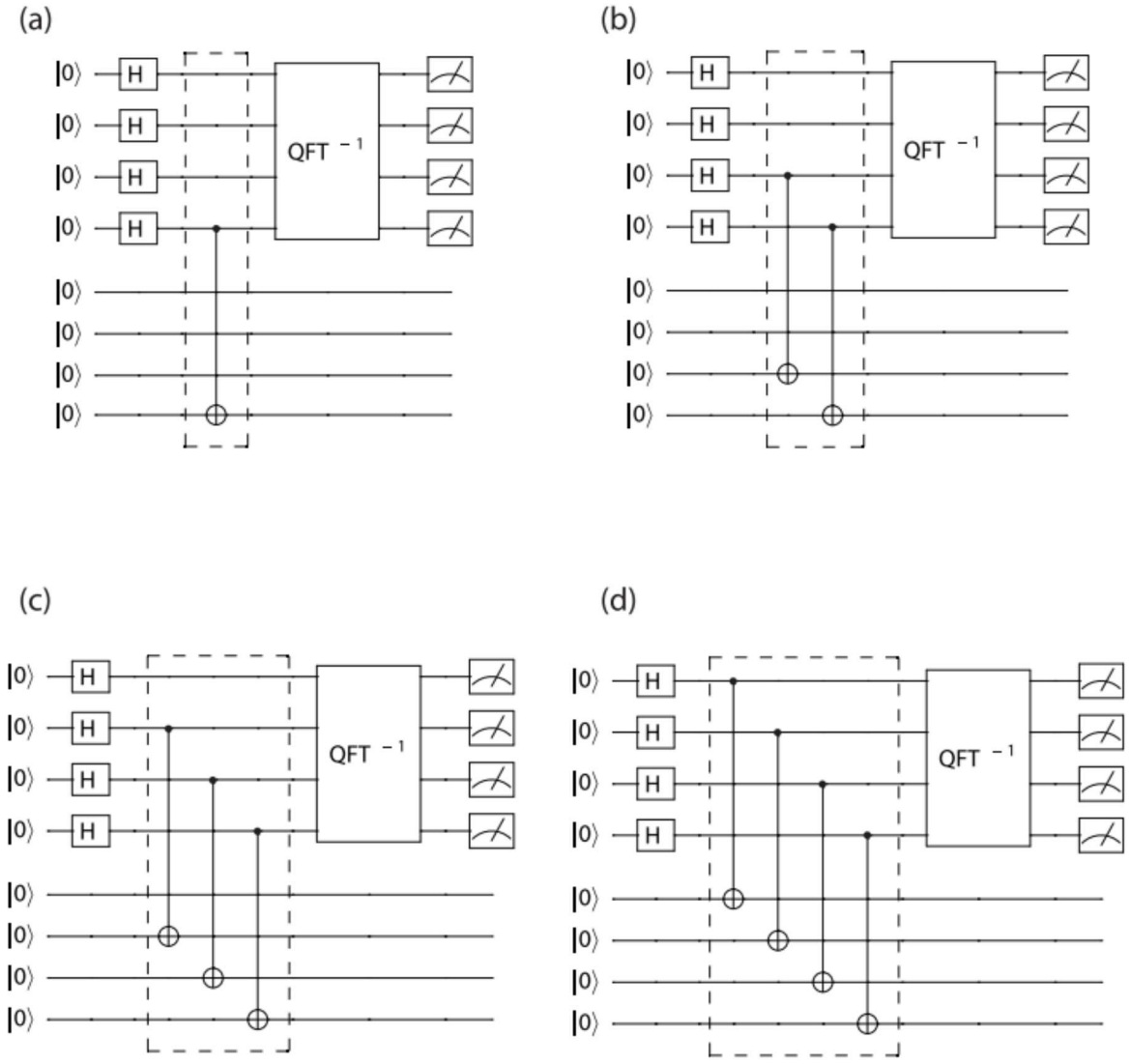


Figure 6.1: Quantum circuit for factoring 51. The circuit is selected based on the guess value a. All possible assignments of a is given in the table below.

All possible assignments of a.	
base a	circuit
16, 35, 50*	Fig 6.1(a)
4, 13, 38, 47	Fig 6.1(b)
2, 8, 19, 25, 26, 32, 43, 49	Fig 6.1(c)
5, 7, 10, 11, 14, 20, 22, 23, 28, 29, 31, 37, 40, 41, 44, 46	Fig 6.1(d)

Table 6.1: The table consists of all possible values of a which are co-prime to 51, for $N = 51$ quantum circuits. The base marked by an asterisk satisfies $a^{r/2} = -1 \bmod N$ and will result in a factorization failure in the classical post-processing analysis.

6.3 Implementation on Qiskit

Now let's implement the circuit on Qiskit. Note that here we will implement it only a simulator as we don't have access to real quantum computer with more than 5 qubits. Here we are considering $a = 10$ and hence implement the circuit provided in Fig 6.1(d).

Importing Libraries

```
import matplotlib.pyplot as plt
import numpy as np
from qiskit import IBMQ, QuantumCircuit, Aer, transpile, assemble
from qiskit.visualization import plot_histogram
from math import gcd
from numpy.random import randint
import pandas as pd
from qiskit.providers.ibmq import least_busy
from fractions import Fraction
```

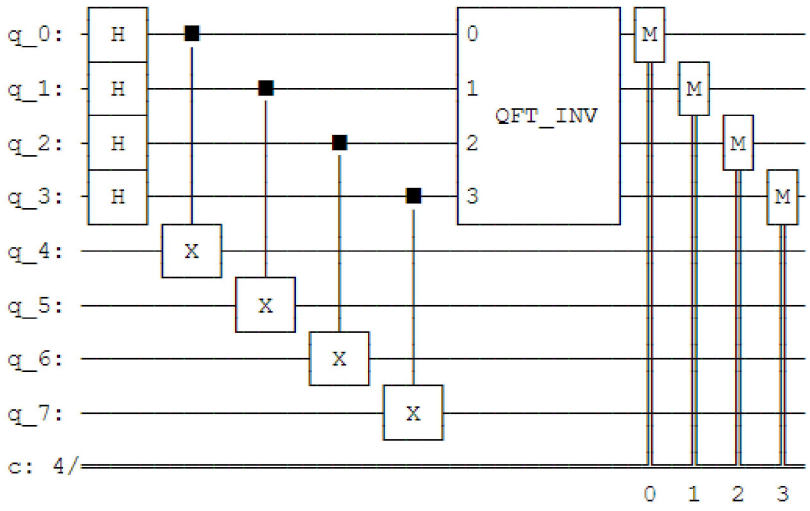
The Inverse QFT

```
def qft_inv(n):
    qc = QuantumCircuit(n)
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "QFT_INV"
    return qc
```

The Circuit

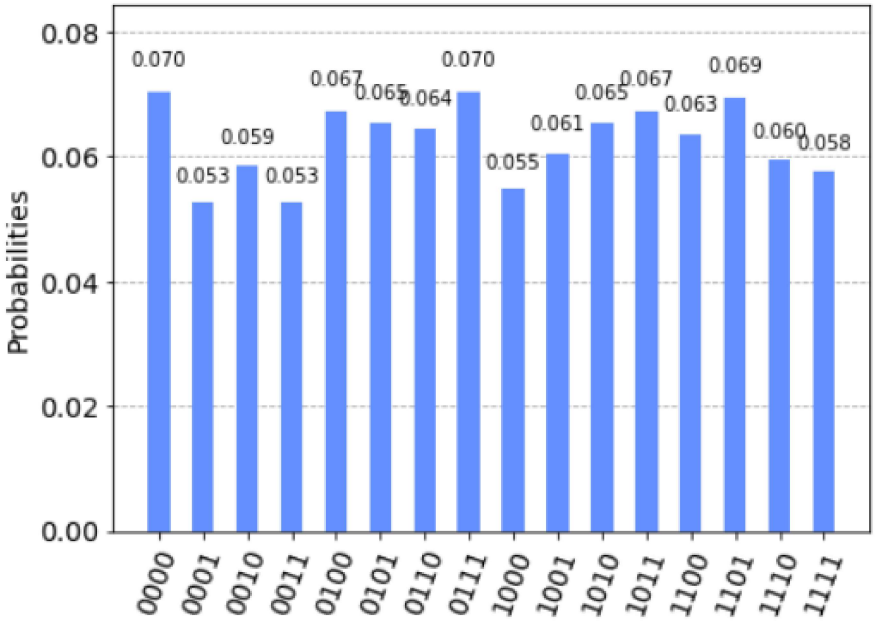
```
n_count = 4
n = 5
qc = QuantumCircuit(n_count+4, n_count)
for q in range(n_count):
    qc.h(q)

#building the modular exponentiation circuit
qc.cx(0,4)
qc.cx(1,5)
qc.cx(2,6)
qc.cx(3,7)
#applying inverse QFT
qc.append(qft_inv(n_count), range(n_count))
#measuring the qubit
qc.measure(range(n_count), range(n_count))
qc.draw()
```



6.3.1 Running on a simulator

```
aer_sim = Aer.get_backend('aer_simulator')
t_qc = transpile(qc, aer_sim)
qobj = assemble(t_qc)
results = aer_sim.run(qobj).result()
counts = results.get_counts()
plot_histogram(counts)
```



Calculating the Phase

```
rows, measured_phases = [], []
for output in device_counts:
    decimal = int(output, 2)  # Convert (base 2) string to decimal
    phase = decimal/(2**n_count)  # Find corresponding eigenvalue
    measured_phases.append(phase)
    # Add these values to the rows in our table:
    rows.append([f"{output}(bin) = {decimal:>3}(dec)",
                 f"{decimal}/{2**n_count} = {phase:.2f}"])
# Print the rows in a table
headers=["Register Output", "Phase"]
df = pd.DataFrame(rows, columns=headers)
print(df)
```

Calculating r

```
rows = []
for phase in measured_phases:
    frac = Fraction(phase).limit_denominator(15)
    rows.append([phase, f"{frac.numerator}/{frac.denominator}",
 frac.denominator])
# Print as a table
headers=["Phase", "Fraction", "Guess for r"]
df = pd.DataFrame(rows, columns=headers)
print(df)
```

	Register Output	Phase
0	0110(bin) = 6(dec)	6/16 = 0.38
1	0000(bin) = 0(dec)	0/16 = 0.00
2	1000(bin) = 8(dec)	8/16 = 0.50
3	1100(bin) = 12(dec)	12/16 = 0.75
4	1011(bin) = 11(dec)	11/16 = 0.69
5	1101(bin) = 13(dec)	13/16 = 0.81
6	0001(bin) = 1(dec)	1/16 = 0.06
7	1001(bin) = 9(dec)	9/16 = 0.56
8	0010(bin) = 2(dec)	2/16 = 0.12
9	1110(bin) = 14(dec)	14/16 = 0.88
10	1010(bin) = 10(dec)	10/16 = 0.62
11	0101(bin) = 5(dec)	5/16 = 0.31
12	0111(bin) = 7(dec)	7/16 = 0.44
13	0011(bin) = 3(dec)	3/16 = 0.19
14	0100(bin) = 4(dec)	4/16 = 0.25
15	1111(bin) = 15(dec)	15/16 = 0.94

	Phase	Fraction	Guess for r
0	0.3750	3/8	8
1	0.0000	0/1	1
2	0.5000	1/2	2
3	0.7500	3/4	4
4	0.6875	9/13	13
5	0.8125	9/11	11
6	0.0625	1/15	15
7	0.5625	5/9	9
8	0.1250	1/8	8
9	0.8750	7/8	8
10	0.6250	5/8	8
11	0.3125	4/13	13
12	0.4375	4/9	9
13	0.1875	2/11	11
14	0.2500	1/4	4
15	0.9375	14/15	15

Finding the factors

As discussed earlier we only consider those r which are even. Here the choices for $r = 8, 2, 4$. We have $a = 10$ and $N = 51$.

If $r = 2$

$$10^{2/2} + 1 = 10^1 + 1 = 11 \tag{6.20}$$

$$10^{2/2} - 1 = 10^1 - 1 = 9 \tag{6.21}$$

We observe that 11 and 9 both are not factors of 51, but $\gcd(9, 51) = 3$, which is a prime and $\frac{51}{3} = 17$ which is also a prime. So the prime factors of 51 are 3 and 7.

Consider $r = 4$

$$10^{4/2} + 1 = 10^2 + 1 = 101 \tag{6.22}$$

$$10^{4/2} - 1 = 10^2 - 1 = 99 \tag{6.23}$$

Here too, 101 and 99 are not factors of 51 but $\gcd(99, 51) = 3$ and using this we get other factor as 17.

Now consider $r = 8$

$$10^{8/2} + 1 = 10^4 + 1 = 10001 \tag{6.24}$$

$$10^{8/2} - 1 = 10^4 - 1 = 9999 \tag{6.25}$$

Procedding in similar way, we get the factors as 3 and 17.

So for all even r, we get the prime factors of 51 as 3 and 17.

Concluding Remarks

In summary, we started with the requirement of Quantum Computing, then later we discussed how gates and circuits are defined and how we perform measurements on them. Later, we described Quantum Fourier transform, one of the key concepts in Quantum computing and using the foundation of QFT we constructed the circuits for Quantum Phase Estimation and Order-Finding Algorithm which we later use to factorize a given composite number N.

We approached Shor's Algorithm by mathematically factoring 15. Moving forward we implemented a compiled version of Shor's Algorithm on IBM's quantum computer and quantum simulator for the prime factorization of 21. Here we took our guess a as 4 and compiled it on a circuit using only 5 qubits. The circuit consisted only of elementary gates like Fredkin, Toffoli, CNOT, H, S and X and we could factor the number 21 using very limited resources.

Next we described another optimal method to factor composite number N into it's prime factors using Fermat primes. Here we considered a series of numbers that have all orders equal to a power of two and can be factored with fewer resources than that of other products with the same number of bits. We demonstrated an example by factoring 51 using only 8 qubits and elementary gates like H, CNOT and X.

Different compiled versions of Shor's Algorithm have been demostrated using very few resources and this project builds upon that idea. Moving forward one can use such techniques to factor large number efficiently on a quantum computer.

References

[1] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010.

[2] Quantum Computers Explained – Limits of Human Technology

[3] Quantum Tunneling-Physics Videos by Eugene Khutoryansky

[4] Poornima Ardhyamath1 , N. M. Naghabhushana2 , Rohitha Ujjinimatad. *Quantum Factorization of Integers 21 and 91 using Shor's Algorithm*

[5] C. Lavor , L.R.U. Manssur , and R. Portugal. *Shor's Algorithm for Factoring Large Integers*

[6] Unathi Skosana and MarkTame. *Demonstration of Shor's factoring algorithm for N = 21 on IBM quantum processors*

[7] Enrique Martin-Lopez, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou, Jeremy L. O'Brien. *Experimental realisation of Shor's quantum factoring algorithm using qubit recycling*

[8] S. Parker and M.B. Plenio. *Efficient factorization with a single pure qubit and* $\log N$ *mixed qubits*

[9] Michael R. Geller and Zhongyuan Zhou. *Factoring 51 and 85 with 8 qubits*

[10] The science behind quantum computers: Tunnelling and Entanglement

[11] IBM Qiskit Textbook

[12] Neal Koblitz. *A Course in Number Theory and Cryptography*

[13] How Quantum Computers Break Encryption-minutephysics