BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING
FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF MATHEMATICS
ÚSTAV MATEMATIKY

SHOR'S ALGORITHM IN QUANTUM CRYPTOGRAPHY
SHORŮV ALGORITMUS V KVANTOVÉ KRYPTOGRAFII

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **B.Tech. Martyns Nwaokocha**
AUTOR PRÁCE

SUPERVISOR — **doc. Mgr. Jaroslav Hrdina, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Mechanical Engineering

MASTER'S THESIS

Brno, 2020 B.Tech. Martyns Nwaokocha

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF MECHANICAL ENGINEERING

Assignment Master's Thesis

Institut:	Institute of Mathematics
Student:	**B.Tech. Martyns Nwaokocha**
Degree programm:	Applied Sciences in Engineering
Branch:	Mathematical Engineering
Supervisor:	**doc. Mgr. Jaroslav Hrdina, Ph.D.**
Academic year:	2020/21

As provided for by the Act No. 111/98 Coll. on higher education institutions and the BUT Study and Examination Regulations, the director of the Institute hereby assigns the following topic of Master's Thesis:

Shor's algorithm in Quantum Cryptography

Brief Description:

A fundamental problem in computacional number theory is the search of proper divisors of a big positive integer N. The quantum framework permits a solution whose complexity is polynomial. It is hepfull for a new cryptosystems that are secure from quantum computers, collectively called post–quantum cryptography.

Master's Thesis goals:

Learning the basics of quantum computing. Understendig the Shor's algorithm and its applications in cryptography. Programing the cryptography scheme in the simulation software and discusing its complexity with respect to classical non–quantum ones.

Recommended bibliography:

[GC] DE LIMA MARQUEZINO, Franklin, et al.: A Primer on Quantum Computing, SpringerBriefs in Computer Science, 2019.

[QN] RUE, Juanjo, XAMBO, Sebastian. Mathematical essentials of quantum computing, Lecture notes UPC, https://web.mat.upc.edu/sebastia.xambo/QC/qc.pdf

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2020/21

In Brno,

L. S.

prof. RNDr. Josef Šlapal, CSc. Director of the Institute	doc. Ing. Jaroslav Katolický, Ph.D. FME dean

Abstract

Cryptography is a very important aspect of our daily lives as it gives the theoretical foundation of information security. Quantum computation and information is also becoming a very important field of science because of its many application areas including cryptology and more specifically in public key cryptography.

The difficulty of factoring numbers into its prime factors is the basis of some important public key cryptosystems of which the RSA is an important example. Shor's Quantum factoring algorithm leverages most especially the quantum interference effect of quantum computing to factor semi-prime numbers in polynomial time on a quantum computer. Though the capacity of current quantum computers to execute the Shor's Algorithm is very limited, there are many extensive foundational scientific research on various techniques of optimizing the algorithm in terms of factors such as number of qubits, depth of the circuit and number of gates.

In this thesis, two variants of the Shor's factoring algorithm and quantum circuits are discussed. Next, these variants are simulated on the QASM simulator in the IBM Quantum Experience platform using the Python Programming language and the Qiskit software development kit. Afterwards, the simulation results are analysed and compared in terms of their complexity and success rate.

The organization of the thesis is as follow: Chapter 1 discusses some key historical result in quantum cryptography, states the problem discussed in this thesis and presents the objectives to be achieved. Chapter 2 summarizes the mathematical background in quantum computing and public key cryptography as well as describing the notation used throughout the thesis. This also explains how a realizable order-finding or factoring algorithm can be used to break the RSA cryptosystem. Chapter 3 presents the building blocks of Shor's algorithm including the Quantum Fourier Transform, Quantum Phase Estimation, Modular Exponentiation and Shor's algorithm in detail. Different optimization variants of the quantum circuits are also presented and compared here. Chapter 4 presents the results of the simulations of the various versions of the Shor's algorithm. In Chapter 5, we discuss the achievement of thesis goals, summarize the results of the research and outline possible future research directions.

Summary

In computational number theory, a fundamental problem is the ability to factor a large positive integer N. This fundamental problem is the strength of some key public key cryptosystems being used today and key to note is the RSA cryptosystem. Shor's quantum factoring algorithm offers a solution with polynomial complexity when compared to the exponential complexity obtainable in existing classical algorithms. In this thesis, we have discussed Shor's quantum factoring algorithm in cryptography, its building blocks and how an efficient factoring algorithm such as Shor's factoring algorithm could be used in breaking the RSA system. We also discussed and simulated two implementation variants using the standard Quantum Fourier Transform (QFT) and the semiclassical QFT.

From our simulation, we have confirmed previous theoretical results that the variant using semiclassical QFT optimizes the number of qubits by reducing the number of qubits from from $2n$ to 1 qubit in the factorization of an n-bit integer N. However, this comes at the cost of more quantum operations and measurements thereby resulting in a slightly lower success rate of finding the right order and hence the right factorization. Another drawback is that compared to the standard implementation of the QFT, it takes more time to execute the algorithm including the amount of time it takes for the measurement outcomes gotten from the simulation to be classically post-processed using the continued fraction algorithm. Hence when the amount of available qubits is very limited and its optimization is of higher priority, the implementation of the Shor's factoring algorithm using semiclassical is better suited.

Souhrn

V algoritmické teorii čísel je jedním ze zásadních problémů schopnost faktorizovat velká kladná celé číslo N. Na schopnosti faktorizovat velká kladná čísla je založena výpočetní síla některých klíčových kryptosystémů využívající veřejného klíče, které se dnes používají. Jedná se především o kryptosystém RSA. Například algoritmus kvantového faktorizování SHORA nabízí řešení s polynomiální složitostí ve srovnání s exponenciální složitostí dosažitelnou ve stávajících klasických algoritmech. V této práci jsme diskutovali Shorův kvantový algoritmus v kryptografii, jeho stavební kameny a to, jak lze použít efektivně při prolomení systému RSA. Rovněž jsme diskutovali a simulovali dvě varianty implementace pomocí standardní kvantové Fourierovy transformace (QFT) a semiklasické QFT.

Pomocí simulace jsme potvrdili předchozí teoretické výsledky. Především, že varianta využívající semiklasický QFT optimalizuje počet qubitů snížením jejich počtu z 2n na 1 qubit při faktorizaci n-bitového celého čísla N. To však přichází za cenu více kvantových operací a měření, což má za následek mírně nižší úspěšnost nalezení správného řádu, a tedy správné faktorizace. Další nevýhodou je, že ve srovnání se standardní implementací QFT trvá provedení algoritmu více času, včetně doby, kterou trvá, než budou výsledky měření získané ze simulace klasicky dodatečně zpracovány pomocí algoritmu pokračujícího zlomku. Pokud je tedy počet dostupných qubitů velmi omezený a jeho optimalizace má vyšší prioritu, je vhodnější implementace Shorova factoringového algoritmu pomocí semiklasického QFT.

Keywords
Cryptography, Quantum Computing, Quantum Algorithm, Shor Algorithm, Fourier transformation, Factorization, RSA.

Klíčová slova
Kryptografie, Kvantové počítání, Kvantové algoritmy, Shorův algoritmus, Fourierova transformace, Faktorizace, RSA

NWAOKOCHA, M.*Shorův algoritmus v kvantové kryptografii.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2021. 76 s. Vedoucí doc. Mgr. Jaroslav Hrdina, Ph.D.

I acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team.

Declaration

I declare that I have written my Master's thesis on the theme "Shor's Algorithm in Quantum Cryptography" independently, under the guidance of my Master's thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis. As the author of the Master's thesis I furthermore declare that, as regards the creation of this Master's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation S 11 and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

Brno ....

B.Tech. Martyns Nwaokocha

Acknowledgement

I would like to express the deepest appreciation to my supervisor Mgr. Jaroslav Hrdina, Ph.D for his valuable advice, support, discussions and suggestions. I am grateful to the Intermaths Consortium and Erasmus$^+$ mobility for the mobility grant I have received while studying abroad.
To my wonderful family, especially my parents Mr. and Mrs. Patrick and Regina Nwaokocha, thank you so much for your relentless prayers, support and encouragement all the way. Also, a big thanks to the family of Mr. and Mrs. Abashe Shehu for your prayers. My appreciation also to Reverend Father Anyanwu Chibueze Cajethan for all your support and prayers to me.
To almighty God for his grace and blessings, I'm utmost grateful.

.....

B.Tech. Martyns Nwaokocha

Contents

1. Introduction

This chapter introduces the thesis scope which is Shor's algorithm in quantum cryptography. It describes the motivation of the thesis, its objective and the outline of the problem that it solves. Section 1.1 discusses the history of quantum mechanics especially as it relates to computer science. In section 1.2, the historical development as well as some key results in quantum cryptography are highlighted. Section 1.3 summarizes the problem to be solved in this thesis and in section 1.4, the objectives of the thesis is presented. Lastly, section 1.5 presents the outline of the following chapters.

1.1. History of Quantum Mechanics

Quantum mechanics is a fundamental theory in physics that gives a description of the physical properties of nature at the scale of atoms and subatomic particles [1]. It differs from classical mechanics given that energy, momentum and other quantities of a bound system are quantized. Also in quantum mechanics, objects exhibit wave-particle duality and the uncertainty principle.

Quantum mechanics began from scientists formulating theories trying to explain observations which could not be explained with classical physics. Such problems included the black-body radiation problem which Max Planck solved in 1900 and the relationship between energy and frequency which Albert Einstein in 1905 explained in his paper on photoeletric effect. Since then, there has been significant progress in the theory where new branches such as quantum chemistry, quantum field theory, quantum technology and quantum information science have arisen from.

After the discovery of elementary particles, it was noticed that they could carry information. In 1980, Paul Benioff observed that quantum mechanics could be used to perform computations [2]. Later in 1982, Richard Feynman had the thought of a "quantum computer" which was to be a computer that used the effects of quantum mechanics such as superposition and interference to its advantage [3]. Several physicists and scientists have been working on this idea and in 2009, Aaron D. O'Connell invented the first quantum machine, by the application of quantum mechanics to a macroscopic object just large enough to be seen by the naked eye and which is able to vibrate a small amount and large amount simultaneously[4].

Lately, the key to improving computer performance has been the reduction of size in the transistors used in modern processors [5]. However, this continual reduction in size cannot continue for too long as if the transistors become too small, the effects of quantum mechanics will begin to hinder their performance. Over the years, many ideas for exploiting quantum mechanics in computer science have been proposed. Consequently, the field Quantum Computer Science was developed and currently could be split into Quantum Computation and Quantum Communication.

1.2. History of Quantum Computing and Quantum Cryptography

Quantum computing is the use of quantum phenomena such as superposition, interference and entanglement to perform computation. It is a subfield of quantum information science and began in the early 1980s, when Paul Benioff proposed a quantum mechanical model of the Turing machine. Later, Yuri Manin in 1980 and Richard Feynman in 1982 suggested that a quantum computer could simulate things that a classical computer could not do [6, 7].
Quantum cryptography relies on the foundations of quantum mechanics. This is unlike traditional cryptography which relies on the computational difficulty of mathematical problems. Quantum cryptography was first proposed in 1968 by Stephen Wiesner who also introduced the concept of quantum conjugate coding and quantum money. In his paper titled "Conjugate Coding" published in 1983, he showed how to store or transmit two messages by encoding them in two "conjugate observables" [8]. These included linear and circular polarization of light thereby ensuring that either but not both, may be received and decoded. A decade later, Charles H. Bennett and Gilles Brassard proposed a method for secure communication based on Wiesner's conjugate observables [9]. In 1991, Artur Ekert developed a different approach to quantum key distribution based on special quantum correlations called quantum entanglement [10]. As traditional public key cryptography is being threatened by the development of quantum algorithms and quantum computing for breaking cryptosystems, quantum cryptography has received more attention since the 1990s.

In 1994, Peter Shor developed a quantum algorithm for factoring integers which could be applied in the decryption of cipher-text encrypted with RSA [11]. Very recently in December 2019, Craig Gidney and Martin Ekerå [12] showed how to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits by combining techniques from Shor 1994 [11], Griffiths-Niu 1996 [13] , Zalka 2006 [14], Fowler 2012 [15], Ekerå-Håstad 2017 [16] , Ekerå 2017 [17] , Ekerå 2018 [18] , Gidney-Fowler 2019 [19] and Gidney 2019 [20].

There are several models of quantum computing systems, including the quantum circuit model, quantum Turing machine, adiabatic quantum computer, one-way quantum computer, and different quantum cellular automata. Amongst these, the quantum circuit model is the most used and they are based on the quantum bit (qubit) which is analogous to the bit in classical computation.

There have been significant progress in building a physical quantum computer however there are a number of significant obstacles impeding the construction of useful quantum computers. In particular, it is difficult to maintain the quantum states of qubits as they suffer from quantum decoherence and state fidelity. Hence, Quantum computers require error correction solutions [21, 22].

Quantum computers obey the Church–Turing thesis which means that any computational problem that can be solved by a classical computer can also be solved by a quantum computer and conversely, any problem that can be solved by a quantum computer can also

be solved by a classical computer, at least in principle given enough time. Hence while quantum computers offer no additional advantages over classical computers in terms of computability, quantum algorithms for certain problems have significantly lower time complexities than corresponding known classical algorithms. Notably, quantum computers are believed to be able to quickly solve certain problems that no classical computer could solve in any feasible amount of time and this phenomenon is called "quantum supremacy".

A major problem in the execution of a quantum algorithm is the number of qubits required for the algorithm. In Shor's algorithm, a major subroutine is the modular exponentiation and many researchers have focused on optimizing the circuit that realises modular exponentiation operation. In particular, some work have been done in reducing the number of qubits in implementing the circuit for factorization especially the modular exponentiation sub-circuit. Vedral, Barenco and Ekert in [23] showed that $7n + 1$ qubits and $O(n^3)$ elementary gates are needed for modular exponentiation. This number could be reduced to $5n + 2$ qubits with some basic optimization and reduced further to $4n + 3$ if unbounded Toffoli gates are available. As at the time of this writing in IBM Quantum [24] for example, the maximum Toffoli gates existing is the doubly controlled-NOT. Beckman, Chan, Devabhaktoni and Preskill in [25] performed an extended analysis of modular exponentiation with a circuit of $5n + 1$ qubits using elementary gates and $4n + 1$ if unbounded Toffoli gates are available. Zalka in [26] also explained a factorization method with $3n + O(logn)$ qubits using only elementary gates. As shown in [27], if the type and size of all the quantum gates are not restricted in any way, the order-finding can be done directly using controlled multiplication gates with just $n + 1$ qubits. However, with the current physical implementation of quantum computer, having no restrictions in the type and size of the quantum gates is not existing. The work of Stephane [28] showed that using the semi-classical QFT, the order finding circuit could be performed with $2n + 3$ qubits, $O(n^3 log(n))$ gates and a depth of $O(n^3)$. This thesis is focused on implementing the Shor's algorithm and simulating the exponential speedup it offers when solving period-finding problems as theoretically deduced from the work of Stephane [28].

1.3. Problem Outline

Quantum computing is becoming more diverse and practical in several scientific areas. Currently, there are many quantum computers available such as the one from IBM available to the public. However, their capacity is still limited especially on the number of qubits they possess and possible gate operations they can directly perform.

Peter Shor in 1994 discovered an algorithm which have been named after him that showed how a Quantum computer could factor semi-prime numbers into their prime factors in polynomial time [11]. In comparison with classical factoring algorithms, they do so in exponential time when performed in a generic way. This computational complexity is the basis for cryptosystems such as the RSA which is widely used in information security. Shor's factoring algorithm when implemented on a scalable quantum computer could lead to breaking the RSA cryptosystem. This has also encouraged more research in this area to explore other different ways quantum mechanics can be leveraged in cryptography applications. Despite the current technological limitation however, much work has been

done to derive the most efficient quantum circuit implementation with optimism regarding better architectures of quantum computers in the future.

A quantum bit (qubit) is a basic unit of information in quantum computers and is often implemented with elementary particles. These qubits are fragile and interferes with the environment despite the great amount of care in isolating them. This forms a fundamental obstacle in the construction of a scalable quantum computer, hence spurring the need of optimizing quantum algorithms in terms of the number of qubits used as well as gate operations in the circuit.

Shor's algorithm is an example of a complex algorithm with multiple parts can be implemented in various ways usually based on the objective of whether the number of qubits, or gate operations or other metrics is to be optimized. In this thesis, we try to describe two implementation variants and compare them with emphasis on the optimization of the number of qubits used. Though the technology to execute Shor's algorithm is limited, quantum computer simulators that can execute the algorithm on classical computers have been developed. These simulators are great tools to test and analyse quantum algorithms however as shown in [7], these simulators cannot execute quantum algorithms efficiently. In this thesis, two variants of the Shor's factoring algorithm are discussed, simulated and analysed and compared.

1.4. Goals of the Thesis

The main objective of this thesis is to review and simulate Shor's algorithm in quantum cryptography as well as discuss its complexity with respect to classical non-quantum schemes. This has been fulfilled by achieving the following goals:

Review of Shor's Algorithm

The Shor's algorithm should be described in full details. The initial steps of the algorithm with respect to classical pre-processing, the quantum order finding and classical post-processing should be discussed. Different approaches to implement the quantum circuits should be described and its comparison in terms of execution time, success rate and circuit metrics such as the required number of qubits, number of gates and circuit depth should be summarized.

Simulation of Shor's Algorithm

The most significant variants of the quantum circuits should be implemented in a quantum computer simulator. These should be tested and ensured to be working correctly.

Analysis of simulation results

The simulation results should be presented and analyzed. Comparisons should be made with respect to execution time and the success rate of the different implementation variants.

1.5. Thesis Outline

The organization of the thesis is as follows:

Chapter 2 summarizes the mathematical background in quantum computing and public key cryptography. Here we present the notations used in quantum mechanics and basics of complex linear algebra. Next we discuss the quantum bits (qubits), operations on qubits and quantum circuits. At the end of the chapter, we summarize integer factorization in cryptography with a focus on how a realizable order-finding or factoring algorithm could be used to break the RSA cryptosystem.

Chapter 3 presents the building blocks of Shor's algorithm. We begin by discussing the Quantum Fourier Transform (QFT), the Quantum Phase Estimation (QPE) and how it is used as a building block for developing the Shor's algorithm. Next, we discuss in details the sub-circuits within the Shor's factoring quantum algorithm from the quantum adder to the modular exponentiation. This is followed by discussing a different optimization variant of the Shor's algorithm using Semi-classical QFT and the chapter ends with the complexity analysis of the Shor's algorithm including the complexity of the subroutines that make up the entire circuit.

Chapter 4 presents the manual calculation steps of a simple example and a discussion of the computing steps taken to simulate two variants of the Shor's algorithm including a simplified variant that uses a constant optimized circuit. Next, we present the results of the simulations of the two variants of the Shor's algorithm on the IBM QASM simulator. We conclude the chapter by analysing and comparing the complexity and success rate of these implementation variants of the algorithm.

In Chapter 5, we discuss the achievement of the thesis goals, summarize the results of the research simulations and outline possible future research directions.

2. Mathematical Background in Quantum Cryptography

In this chapter, we introduce key concepts in quantum computing and integer factorization in cryptography. Only the basic concepts which are necessary to understand the thesis are presented and more comprehensive description can be found in [29, 30]. In section 2.1, we summarize the concepts of single and multi-qubit as well as gate operations on them and measurements. At the end of that section, quantum entangled states and quantum circuits are introduced. In section 2.2, we summarize integer factorization in cryptography listing some classical integer factorization schemes and their complexities.

2.1. Basics of Quantum computing

2.1.1. Complex linear algebra

Vector space

A vector space V over the scalar field $\mathbb{C}$ is a set, the elements of which are called vectors $|v\rangle$, on which the following two operations are defined

- $V \times V \ni (|x\rangle, |y\rangle) \mapsto |x\rangle + |y\rangle \in V$
- $\mathbb{C} \times V \ni (\lambda, |x\rangle) \mapsto \lambda|x\rangle \in V$

with the following properties:

1. $\forall\ |x\rangle, |y\rangle, |z\rangle \in V$,
   (a) $|x\rangle + |y\rangle = |y\rangle + |x\rangle$,
   (b) $|x\rangle + (|y\rangle + |z\rangle) = (|x\rangle + |y\rangle) + |z\rangle$.
2. $\forall\ |x\rangle \in V, \exists\ 0 \in V : 0 + |x\rangle = |x\rangle$,
3. $\forall\ |x\rangle \in V,\ \exists!\ -|x\rangle \in V :\ |x\rangle + (-|x\rangle) = (-|x\rangle) + |x\rangle = 0$,
4. $\forall\ |x\rangle,\ |y\rangle \in V$ and $\lambda,\ \mu \in \mathbb{C}$,
   (a) $1|x\rangle = |x\rangle$,
   (b) $(\lambda + \mu)|x\rangle = \lambda|x\rangle + \mu|x\rangle$,
   (c) $\lambda(\mu|x\rangle) = (\lambda\mu)|x\rangle$,
   (d) $\lambda(|x\rangle + |y\rangle) = \lambda|x\rangle + \lambda|y\rangle$.

Normed spaces, Inner product and Hilbert Spaces

A norm on a linear space V is a function $\|.\| : V \to \mathbb{R}$ with the following properties:

1. $\forall\ |\varphi\rangle \in V, \||\varphi\rangle\| \geq 0$,
2. $\forall\ |\varphi\rangle \in V$ and $\lambda \in \mathbb{C}, \|\lambda|\varphi\rangle\| = |\lambda|\||\varphi\rangle\|$,

3. $\forall\ |\varphi\rangle$ and $|\psi\rangle \in V, \||\varphi\rangle + |\psi\rangle\| \leq \||\varphi\rangle\| + \||\psi\rangle\|$,
4. $\||\varphi\rangle\| = 0 \implies |\varphi\rangle = 0.$

A normed linear space $(V, \|\cdot\|)$ is a linear space V equipped with a norm $\|\cdot\|$. An inner product on V is a map $\langle .|.\rangle\ : V \times V \to \mathbb{C}$ such that, $\forall\ |x\rangle, |y\rangle, |z\rangle \in V$ and $\alpha, \beta \in \mathbb{C}$:

1. $\langle x, \lambda y + \mu z\rangle = \lambda\langle x, y\rangle + \mu\langle x, z\rangle,$
2. $\langle y, x\rangle = \overline{\langle x, y\rangle},$
3. $\langle x, x\rangle \geq 0,$
4. $\langle x, x\rangle = 0 \iff x = 0.$

A linear space with an inner product is called a pre-Hilbert (inner product) space and a Hilbert space is a complete pre-Hilbert space. In quantum mechanics we work with the Hilbert space $\mathbb{C}^n$ and if $\langle x| = |x\rangle^\dagger$ denotes the conjugate transpose of $|x\rangle$, the standard inner product here is defined as:

$$\langle x|y\rangle\ = \langle x||y\rangle\ = |x\rangle^\dagger|y\rangle\ = \sum_{j=1}^{n} \bar{x}_j y_j$$

where $|x\rangle\ = (x_1, ..., x_n)^T,\ |y\rangle\ = (y_1, ..., y_n)^T \in \mathbb{C}^n$ with $x_i, y_i\ \in \mathbb{C}$.

For an element $\langle\psi|$ of Hilbert space H representing a quantum system, the normalization condition given below must hold.

$$\forall\langle\psi| \in\ H, \quad \langle\psi|\psi\rangle\ =\ 1$$

Two important types of matrices in quantum computing are **Hermitian** ($H\ =\ H^\dagger$) and **Unitary** ($U^{-1}\ =\ U^\dagger$) matrices, where the † symbol denotes its conjugate transpose. Unitary matrices are important in quantum computation because they preserve the inner product.

Outer Products and Tensor Products

For two vectors $|a\rangle$ and $|b\rangle$ in a vector space H, the outer product is defined as

$$|a\rangle|b\rangle^\dagger = |a\rangle\ \langle b|\ =\ \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \begin{bmatrix} \bar{b}_1 & \bar{b}_2 & \cdots & \bar{b}_n \end{bmatrix}\ =\ \begin{bmatrix} a_1\bar{b}_1 & a_1\bar{b}_2 & \cdots & a_1\bar{b}_n \\ a_2\bar{b}_1 & a_2\bar{b}_2 & & \vdots \\ \vdots & & \ddots & \vdots \\ a_n\bar{b}_1 & \cdots & \cdots & a_n\bar{b}_n \end{bmatrix}$$

The outer product is a specific example of the more general **tensor product** (denoted $|a\rangle \otimes |b\rangle$ or $|ab\rangle$) used to multiply vector spaces together and defined on vectors as

$$|a\rangle \otimes |b\rangle\ =\ \begin{bmatrix} a_1 \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \\ a_2 \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \end{bmatrix}\ =\ \begin{bmatrix} a_1b_1 \\ a_1b_2 \\ a_2b_1 \\ a_2b_2 \end{bmatrix}$$

For an operator acting on the tensor product of $|a\rangle$ and $|b\rangle$, we take the tensor product of the operators acting on them. The tensor product of matrices A and B equals:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

Matrix Exponentials

We note that unitary transformations U are of the form

$$U = e^{i\gamma H} = \sum_{n=0}^{\infty} \frac{(i\gamma H)^n}{n!}$$

where H is an Hermitian matrix and $\gamma \in \mathbb{R}$ since $UU^\dagger = e^{i\gamma H}e^{-i\gamma H} = e^0 = \mathbb{I}$.

An **involutory matrix** B is such that $B^2 = \mathbb{I}$ where $\mathbb{I}$ is an identity matrix and for such involutory matrix B, we have

$$e^{i\gamma B} = \cos(\gamma)\mathbb{I} + i\sin(\gamma)B$$

The Pauli gates introduced in section 2.1.4 are examples of involutory matrices and is a useful property as these kind of gates are their own inverses.

Lastly, given some matrix M, with eigenvectors $|v\rangle$ and corresponding eigenvalues λ, we have:

$$e^M|v\rangle = \left(\sum_{n=0}^{\infty} \frac{M^n}{n!}\right)|v\rangle = \sum_{n=0}^{\infty} \frac{M^n|v\rangle}{n!} = \sum_{n=0}^{\infty} \frac{\lambda^n|v\rangle}{n!} = \left(\sum_{n=0}^{\infty} \frac{\lambda^n}{n!}\right)|v\rangle = e^\lambda|v\rangle$$

2.1.2. Single Qubit

A **quantum bit (qubit)** is a bit that obeys the rules of quantum mechanics and is the basic variable of quantum computers. The state of a qubit is a vector in a two-dimensional complex vector space. The special states $|0\rangle$ and $|1\rangle$ are called the computational basis states, and together, form an orthonormal basis for this vector space. **Statevectors** are used to describe the state of a system in quantum physics and helps in keeping track of quantum systems. For example

$$|x\rangle = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{array}{c} \\ \\ \\ \leftarrow \text{an object being at} \\ \text{position } k \\ \\ \\ \end{array} \text{Probability of}$$

Qubit Notation

The states $|0\rangle$ and $|1\rangle$ form an orthonormal basis of $\mathbb{C}^2$, so any 2-dimensional vector $|q_0\rangle$ can be represented by a linear combination of these two states. For example:

$$|q_0\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} \end{bmatrix} \text{ can be written as } |q_0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{i}{\sqrt{2}}|1\rangle, \text{ where}$$

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$|q_0\rangle$ is called the qubit's statevector and is in a state of superposition (linear combination) between $|0\rangle$ and $|1\rangle$.

Measurements

The probability of measuring a state $|\psi\rangle$ in the state $|x\rangle$ is given by:

$$p(|x\rangle) = |\langle x|\psi\rangle|^2$$

The first implication of this rule is that amplitudes are related to probabilities. For the probabilities to be valid,

$$\langle\psi|\psi\rangle = 1.$$

So if $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \;\Rightarrow\; \sqrt{|\alpha|^2 + |\beta|^2} = 1.$

Secondly, this form of measurement is just one of an infinite number of possible ways to measure a qubit. For any orthogonal pair of states, an alternative measurement can be defined that would make a qubit choose between the two states.

In general, if $\gamma \in \mathbb{C}$ is any overall factor on a state and $|\gamma| = 1$, it is called a **global phase**. States differing only by a global phase are physically indistinguishable.

$$|\langle x|(\gamma|a\rangle)|^2 = |\gamma\langle x|a\rangle|^2 = |\langle x|a\rangle|^2$$

2.1.3. Representation of single qubits on the Bloch sphere

Given the general state of a qubit ($|q\rangle$):

$$|q\rangle = \tilde{\alpha}|0\rangle + \tilde{\beta}|1\rangle \text{ for } \tilde{\alpha}, \tilde{\beta} \in \mathbb{C}$$

Since the global phase cannot be measured, only the difference in phase between the states $|0\rangle$ and $|1\rangle$ is measured so it becomes:

$$|q\rangle = \alpha|0\rangle + e^{i\phi}\beta|1\rangle, \text{ for } \alpha, \beta, \phi \in \mathbb{R}.$$

Then, since the qubit state must also be normalised so that:

$$1 = \||q\rangle\| = \alpha\tilde{\alpha} + e^{i\phi}\beta e^{-i\phi}\tilde{\beta} = \alpha\tilde{\alpha} + \beta\tilde{\beta} = \alpha^2 + \beta^2$$

using the trigonometric identity $\sin^2 x + \cos^2 x = 1$ gives

$$\alpha = \cos\tfrac{\theta}{2}, \quad \beta = \sin\tfrac{\theta}{2} \text{ and hence, } |q\rangle = \cos\tfrac{\theta}{2}|0\rangle + e^{i\phi}\sin\tfrac{\theta}{2}|1\rangle$$

Interpreting θ and $\phi \in \mathbb{R}$ as spherical co-ordinates, any qubit state can be plotted on the surface of a sphere called the Bloch sphere. As an example, we plot in figure 2.1 the qubit in the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ which is a superposition of states $|0\rangle$ and $|1\rangle$ corresponding to $\theta = \pi/2$ and $\phi = 0$.

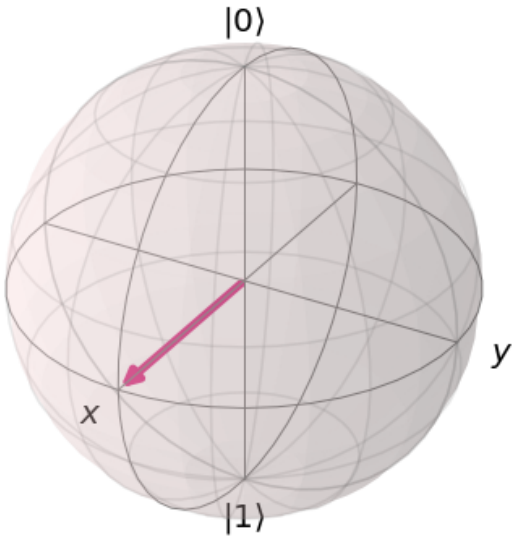


Figure 2.1: Bloch sphere representation of the qubit $|+\rangle$ ($\theta = \pi/2$ and $\phi = 0$)

2.1.4. Single Qubit Gates

Gates are the operations that change a qubit between states. In $\mathbb{C}^2$, qubits are limited to the form:

$$|q\rangle = \cos\left(\tfrac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\tfrac{\theta}{2}|1\rangle.$$

All gates in quantum computing, (except measurement and reset operations), can be represented by unitary matrices and hence are always reversible. Common single qubit gates include:

A. The Pauli Gates

The Pauli matrices can represent some very commonly used quantum gates namely the X, Y and Z gates.

The X-Gate

The X-gate is represented by the Pauli-X matrix:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0|.$$

The X-gate switches the amplitudes of the states $|0\rangle$ and $|1\rangle$, for example:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle.$$

The X-gate is also called a NOT-gate and can be seen as a rotation by π radians around the **x-axis** of the Bloch sphere. Figure 2.2 shows an example of the action of the X-gate on a qubit $|0\rangle$ on the Bloch sphere.

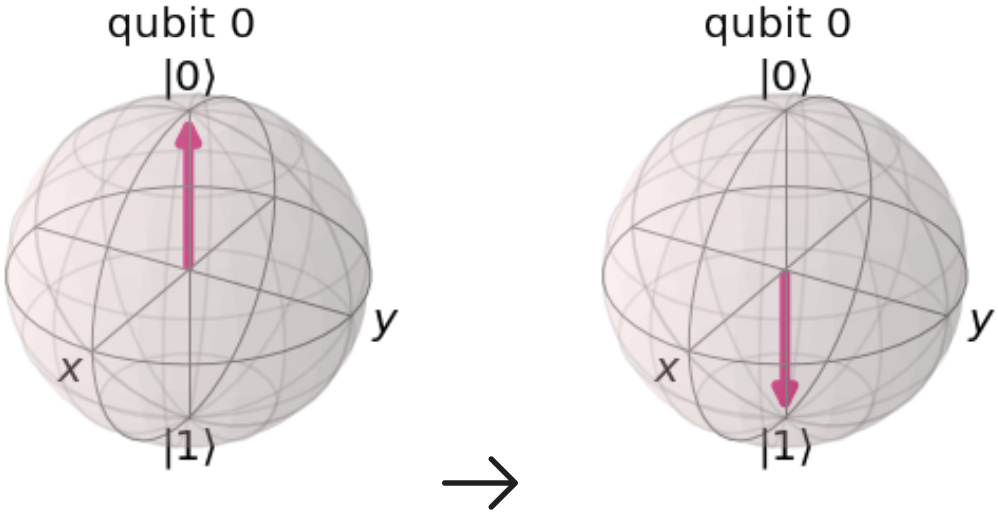


Figure 2.2: Transformation of $|0\rangle$ to $|1\rangle$ using the X-gate

The Y and Z Gates

The Y and Z Pauli matrices also act as the Y and Z-gates in quantum circuits and defined as follows:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$Y = -i|0\rangle\langle 1| + i|1\rangle\langle 0| \qquad Z = |0\rangle\langle 0| - |1\rangle\langle 1|$$

Also, they perform rotations by π radians around the y and z-axis respectively of the Bloch sphere. Figure 2.3 shows an example of the action of the Y-gate on a qubit $|0\rangle$ on the Bloch sphere.

Figure 2.3: Transformation of $|0\rangle$ to $|1\rangle$ using the Y-gate

The Z-Gate has no effect on the qubit in the computational basis states $|0\rangle$ or $|1\rangle$ because they are the two *eigenstates* of the Z-gate. Another popular basis is the X-basis, formed by the eigenstates of the X-gate and are called $|+\rangle$ and $|-\rangle$:

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

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

In fact, there are an infinite number of bases and to form one, just two orthogonal vectors are needed.

B. The Hadamard Gate

The Hadamard gate (H-gate) is a fundamental quantum gate that allows qubits move away from the poles of the Bloch sphere and create a superposition of $|0\rangle$ and $|1\rangle$. It is represented by the matrix:

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

and performs the transformations below:

$$H|0\rangle = |+\rangle,\ H|1\rangle = |-\rangle$$

Figure 2.4 shows an example of the action of the Hadamard gate on a qubit $|0\rangle$ on the Bloch sphere.

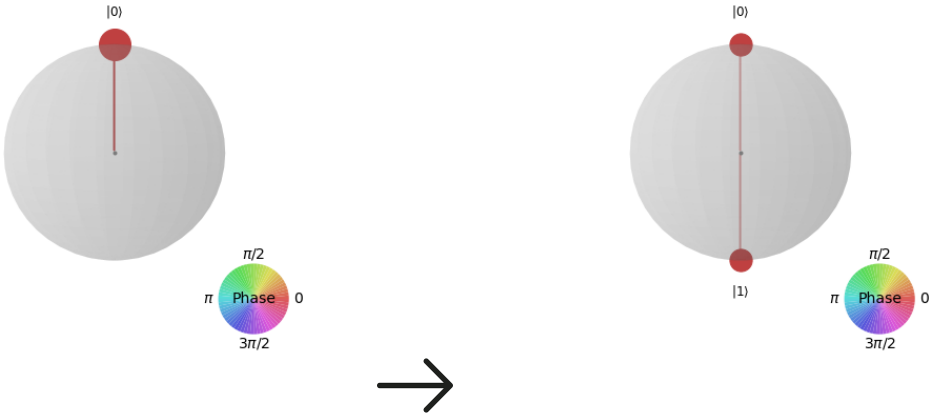


Figure 2.4: Transformation of $|0\rangle$ to $|+\rangle$ using the H-gate

C. The R_ϕ-gate

The R_ϕ-gate is parameterised by (ϕ) which specifies its action. The R_ϕ-gate performs a rotation of ϕ around the Z-axis direction and has the matrix:

$$R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix} \text{ where } \phi \in \mathbb{R}$$

The Z-gate is a special case of the R_ϕ-gate, with $\phi = \pi$. In fact there are three more commonly used gates (I, S, T) all of which are special cases of the R_ϕ-gate.

D. The I, S and T-gates

The I-gate

The I-gate ('Id-gate' or 'Identity gate') is a gate that has no effect on the qubit state and has the identity matrix as its representation:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The S-gates

The S-gate is an R_ϕ-gate with $\phi = \pi/2$ performing a quarter-turn around the Bloch sphere. Unlike the X,Y,Z and H gates, the S-gate is **not** its own inverse having as inverse $S^\dagger$-gate. The $S^\dagger$-gate is also an R_ϕ-gate with $\phi = -\pi/2$:

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

We observe that $SS|q\rangle = Z|q\rangle$ and hence the S-gate is also called "$\sqrt{Z}$-gate" since two successively applied S-gates has the same effect as one Z-gate.

The T-gate
The T-gate is an R_ϕ-gate with $\phi = \pi/4$ having the matrix representation

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

Similar to the S-gate, the T-gate is also called the $\sqrt[4]{Z}$-gate.

E. General U-gates

The U_3-gate is the most general of all single-qubit quantum gates and is a parameterised gate of the form:

$$U_3(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i\lambda+i\phi}\cos(\theta/2) \end{bmatrix}$$

We will use IBM Quantum Experience platform [24] which is a platform for simulating and running quantum programs so let us remark that it only provides the U_2 and U_1 gates which are specific cases of the U_3 gate where $\theta = \frac{\pi}{2}$, and $\theta = \phi = 0$ respectively:

$$U_3(\tfrac{\pi}{2}, \phi, \lambda) = U_2 = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i\lambda+i\phi} \end{bmatrix} \qquad U_3(0, 0, \lambda) = U_1 = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix}$$

Before running on IBM quantum hardware, all single-qubit operations are compiled down to combinations of U_1, U_2 and U_3 gates and hence these U gates are also called the physical gates.

2.1.5. Measuring in Different Bases

An X-gate can be created by sandwiching the Z-gate between two H-gates as follows:

$$X = HZH$$

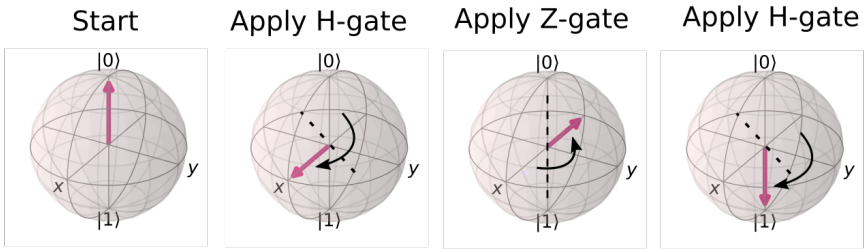


Figure 2.5: Creation of the X-gate from the Z-gate

Source: IBM Qiskit

This is verified below:

$$HZH = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = X$$

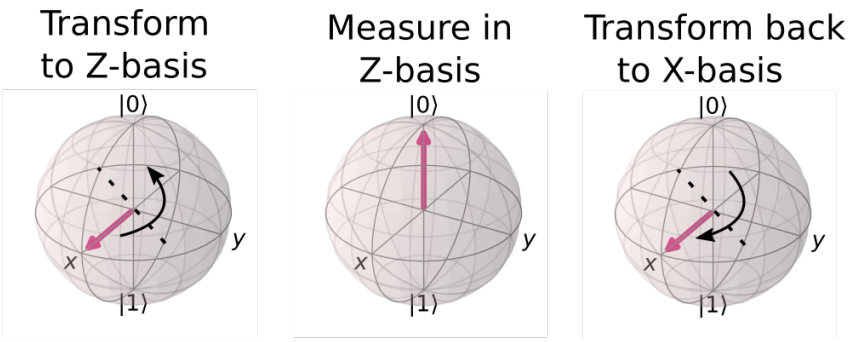


Figure 2.6: Creation of the X-measurement from the Z-measurement [65]

Similarly, an X-measurement can be created by sandwiching the Z-measurement between two H-gates.

2.1.6. Multi-Qubits

The state of two (2) qubits could be described with four (4) complex amplitudes stored in a $4-$dimensional complex vector. Its representation is:

$$|a\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix}$$

The measurement for two qubits is defined in a similar way as in the case of a single qubit, for example the probability of measuring a state $|a\rangle$ in the state $|00\rangle$ is given by:

$$p(|00\rangle) = |\langle 00|a\rangle|^2 = |a_{00}|^2$$

Also, the normalisation condition holds in a similar way as in the case of a single qubit. We have that the normalisation condition on $|a\rangle$ represented above implies that the following equation holds:

$$|a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1$$

The collective state of two separated qubits can be described using the tensor product given below:

$$|a\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}, \quad |b\rangle = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$$

$$|ba\rangle = |b\rangle \otimes |a\rangle = \begin{bmatrix} b_0 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \\ b_1 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} b_0 a_0 \\ b_0 a_1 \\ b_1 a_0 \\ b_1 a_1 \end{bmatrix}$$

Following similar rules as in the case of two qubits, for three qubits we have that :

$$|cba\rangle = \begin{bmatrix} c_0b_0a_0 \\ c_0b_0a_1 \\ c_0b_1a_0 \\ c_0b_1a_1 \\ c_1b_0a_0 \\ c_1b_0a_1 \\ c_1b_1a_0 \\ c_1b_1a_1 \end{bmatrix}$$

Where $|b\rangle = b_{00}|00\rangle + b_{01}|01\rangle + b_{10}|10\rangle + b_{11}|11\rangle$ and $|c\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$.

Hence n qubits require 2^n complex amplitudes.

Single Qubit Gates on Multi-Qubit Statevectors

For a single qubit gate acting on a qubit in a multi-qubit vector, the tensor product is used to calculate the multi-qubit statevector and furthermore, the tensor product is again used to calculate the matrices that act on these statevectors.

Given the two-qubit $|q_1q_0\rangle$, the simultaneous operations of two single qubit gates (for example H and X) can be represented using their tensor product given as:

$$X|q_1\rangle \otimes H|q_0\rangle = (X \otimes H)|q_1q_0\rangle$$

with the operation of these single qubit gates given by:

$$X \otimes H = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & 1 \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ 1 \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & 0 \times \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

which is then applied to the 4-dimensional statevector of $|q_1q_0\rangle$. Another more compact notation of $X \otimes H$ is:

$$X \otimes H = \begin{bmatrix} 0 & H \\ H & 0 \end{bmatrix}$$

To apply a gate to only one qubit at a time, the tensor product is done with the identity matrix for example $X \otimes I$

2.1.7. Multi-Qubit Gates

The CNOT-Gate is a conditional gate that performs an X-gate on the target qubit, if the state of the control qubit is $|1\rangle$. When the qubits are not in a superposition of the $|0\rangle$ or $|1\rangle$ states, the CNOT-Gate have the following truth table representation given below:

Input $\|target, control\rangle$	Output $\|target, control\rangle$
$\|00\rangle$	$\|00\rangle$
$\|01\rangle$	$\|11\rangle$
$\|10\rangle$	$\|10\rangle$
$\|11\rangle$	$\|01\rangle$

The CNOT-Gate acting on a 4-dimensional statevector has one of the following two matrices representation using $|q_1 q_0\rangle$ notation:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \text{ if } q_0 \text{ is the control and } \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \text{ if } q_0 \text{ is the target}$$

Hence for

$$|a\rangle = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix}, \quad \text{CNOT}|a\rangle = \begin{bmatrix} a_{00} \\ a_{11} \\ a_{10} \\ a_{01} \end{bmatrix} \begin{matrix} \\ \leftarrow \\ \\ \leftarrow \end{matrix}$$

Where the qubits in the positions indicated by the arrow have been flipped.

2.1.8. Entangled States

The state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is called a Bell state. This state has 50% probability of being measured in the state $|00\rangle$ and state $|11\rangle$ respectively hence no possibility of being measured in the states $|01\rangle$ or $|10\rangle$. This combined state cannot be written as two separate qubit states and although the qubits are in a superposition state, measuring one will tell the state of the other and collapse its superposition.

For two qubits, there are four Bell states and each have the maximal value of $2\sqrt{2}$ and hence are called maximally entangled Bell states. These four Bell states form the Bell basis of the four-dimensional complex vector space V for two qubits and are given below:

$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right)$$
$$\left|\Psi^+\right\rangle = \frac{1}{\sqrt{2}}\left(|01\rangle + |10\rangle\right)$$
$$\left|\Phi^-\right\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle - |11\rangle\right)$$
$$\left|\Psi^-\right\rangle = \frac{1}{\sqrt{2}}\left(|01\rangle - |10\rangle\right)$$

An easy way to create an entangled Bell state is by taking a computational basis as input and then applying a H gate on the first (or second) qubit and then a CNOT gate having control as the first (or second) qubit and target as the second (or first) qubit.

The measurement result of a single qubit in a Bell state is random but once the first qubit is measured in the computational basis, the measurement result of the second qubit is certain to be same (for Φ Bell states) or the negation (for Ψ Bell states) as that of the first qubit measurement.

2.1.9. 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. Any quantum program can be represented by a sequence of quantum circuits and non-concurrent classical computation [31].

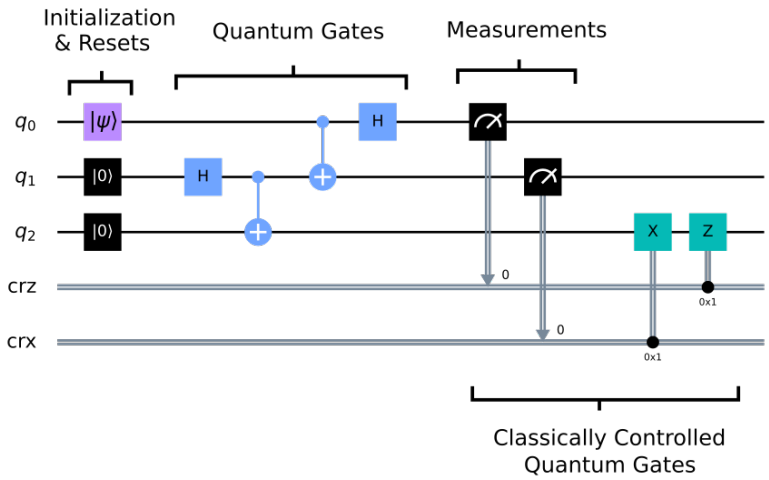


Figure 2.7: Components of a quantum circuit [65]

The quantum circuit in figure 2.7 uses three qubits (q_0, q_1 and q_2) and two classical bits.

There are four main components in a quantum circuit namely Initialization and reset, Quantum gates, Measurements and Classically conditioned quantum gates. These are illustrated in Figure 2.7.

2.2. Integer Factorization in Cryptography

In this section, we will discuss the application of integer factorization in the *RSA* cryptosystem and conclude with a list of some classical factorization algorithms together with their complexity.

2.2.1. Rivest–Shamir–Adleman (RSA) Cryptosystem

Public key cryptography (PKC) are cryptosystems that do not require a secured channel for the transmission of the encryption key. Hence the security of PKC is based on the computational hardness of deriving the private key given the public key. Some of such computational hardness include Discrete logarithm, Shortest vector problem in a lattice (Lattice-based cryptography) and Integer factorization. Examples of cryptosystems depending on each of these computational hard problems are:

1. **Discrete logarithm:** Cramer–Shoup, Diffie-Hellman key exchange, Curve25519, ElGamal and Elliptic Curve Cryptography.
2. **Lattice-based cryptography:** NTRUEncrypt and GGH encryption scheme
3. **Integer factorization:** Cayley–Purser, Schmidt–Samoa, Paillier, Rabin and *RSA* cryptosystems

In particular, we note that *RSA* is used with key lengths of 1024 to 4096 bits long and we will now discuss an example of how it could be used.

If Alice wishes to communicate securely with Bob, Alice needs to generate her private and public key pair and make her public key known to Bob. She performs the following steps to generate her public and private key pair:

1. Choose 2 large prime numbers, p and q.
2. Compute the product $n = pq$.
3. Choose a random small odd integer, e, such that $gcd(e, \varphi(n)) = 1$ where $\varphi(n) = (p-1)(q-1)$ is the Euler-phi function of n.
4. Calculate $d = e^{-1} \bmod \varphi(n)$.
5. Hence, Alice public key is the pair $P = (e, n)$ while her private key is the pair $S = (d, n)$

If Bob wishes to send a message (M) to Alice, he performs the following:

1. Convert the message M to a number m (say L bits long)
2. If $L > \lfloor \log n \rfloor$, break m into blocks of at most $\lfloor \log n \rfloor$ bits and encrypt the blocks separately.
3. For each block, compute $E(m) = m^e (\mathrm{mod} n)$ where $E(m)$ is the encrypted message
4. Bob sends $E(m)$ to Alice

Alice deciphers $E(m)$ using her private key $S = (d, n)$ by performing the following:

$$\begin{aligned} D(E(m)) &= E(m)^d (\mathrm{mod} n) \\ &= m^{ed} (\mathrm{mod} n) \\ &= m^{1+k\varphi(n)} (\mathrm{mod} n) \text{ [since } ed = 1 (\mathrm{mod} \varphi(n)) \implies ed = 1 + k\varphi(n) \text{ for some } k \in \mathbb{Z}] \\ &= m \cdot m^{k\varphi(n)} (\mathrm{mod} n) \\ &= m (\mathrm{mod} n) \text{ [since by Euler's theorem, } m^{\varphi(n)} = 1 (\mathrm{mod} n)] \end{aligned}$$

This works both in the case when $gcd(m, n) = 1$ and $gcd(m, n) > 1$. When $gcd(m, n) = 1$, then the justification is as shown above.

If $gcd(m, n) > 1$, then either p or q or both divide m. For example, if $p \mid m$ and $q \nmid m$, $p \mid m$ implies that $m = 0 \bmod p$ and so $m^{ed} = 0 = m \bmod p$. Then by Fermat's little theorem, $q \nmid m$ implies that $m^{\varphi(n)} = 1 \bmod q$ since $\varphi(n) = (p-1)(q-1)$. So from $ed = 1 + k\varphi(n)$, $m^{ed} = m \bmod q$ and from the Chinese Remainder Theorem (CRT), we have that $m^{ed} = m \bmod n$ so that Alice can also recover the message (m) even when $gcd(m, n) > 1$.

RSA can be broken through order-finding and factoring. If an attacker receives the ciphertext $m^e \bmod n$ and has the public key (e, n), then he can find the order (r) of the encrypted message so tht $(m^e)^r = 1 \bmod n$. Supposing r exists, then $gcd(m^e, n) = 1$. If r does not exist, then $m^e \bmod n$ and n will have a common factor which can be computed by using the Euclid's algorithm hence enabling the attacker to break the RSA using the method based on factoring. From the CRT, $r \mid \varphi(n)$ and since $gcd(e, \varphi(n)) = 1$, $gcd(e, r) = 1$ hence $e^{-1} \bmod r$ exists and we denote it by d'. This means $ed' = 1 + kr$ for some $k \in \mathbb{Z}$ and the attacker can recover m by computing:

$$\begin{aligned} (m^e)^{d'} (\mathrm{mod} n) &= m^{1+kr} (\mathrm{mod} n) \\ &= m \cdot m^{kr} (\mathrm{mod} n) \\ &= m (\mathrm{mod} n) \end{aligned}$$

Hence we see that with an efficient order-finding algorithm, even without the attacker knowing the private (d, n), he can break and recover the original message m.

The second method based on factoring allows the attacker to completely recover the message m and the private key (d, n). If the attacker could factor $n = pq$, then he can compute $\varphi(n) = (p-1)(q-1)$. Hence, given $\varphi(n)$ and e, he can compute $d = e^{-1} \bmod \varphi(n)$

and completely recover the private key (d, n) of which computing the original message m is now a trivial operation.

2.2.2. Classical Factorization Schemes

In this section, we will summarise some classical factorization schemes. This will closely follow the sequence as presented in [32].

Factorization schemes are generally divided into two. The first is referred to as Dark Age methods which include algorithms such as Trial division, $p-1$ method, $p+1$ method and Pollard rho method etc. The second is referred to as modern methods and include algorithms such as Continued Fraction Method, Quadratic Sieve, Elliptic Curve Method and Number Field Sieve (NFS) amongst others.

The complexity of modern factoring algorithms lies between polynomial and exponential time, in an area referred to as sub-exponential time. This complexity of modern factoring algorithms are measured by the function

$$L_N(\alpha, \beta) = \exp\left((\beta + o(1))(\log N)^{\alpha}(\log\log N)^{1-\alpha}\right)$$

Where we see that $L_N(0, \beta) = (\log N)^{\beta+o(1)}$ is polynomial time and $L_N(1, \beta) = N^{\beta+o(1)}$ is exponential time [32].
The prime factorization of an n-bit integer using the General Number Field Sieve (GNFS) algorithm (which is the best known classical algorithm) requires $e^{\Theta\left(n^{1/3}\log^{2/3} n\right)}$ operations [29] and as at the time of writing, its theoretical asymptotic running time [33] is:

$$e^{\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}\right)} \quad \approx \quad \mathcal{O}(e^{c.n^{\frac{1}{3}}(\log(n))^{\frac{2}{3}}}) \quad \text{where c is a constant}$$

Other factoring algorithms with their complexities include:

1. Sieve of Erathostenes $[O(n\log\log n)]$ [34],
2. Trial Division factoring method $\left[O\left(\frac{2^{n/2}}{(\frac{n}{2})\ln 2}\right)\right]$,
3. Pollard's (p–1)-method $[O(Bx\log Bx\log^2 n)]$ where B is a smoothness bound,
4. Lenstra elliptic-curve factorization method $\left[e^{(\sqrt{2}+o(1))\sqrt{\ln p\ln\ln p}}\right]$ where p is the smallest factor of n[35],
5. Self-initializing Quadratic Sieve (QS) $\left[\mathrm{O}(e^{(\sqrt{(}\log n\log\log n))})\right]$,
6. Index Calculus Method $[k^{O(\log k)}]$ [36].

With Shor's algorithm, this complexity reduces to $\mathcal{O}(n^2\log(n)\log(\log(n)))$ which is a little faster than $\mathcal{O}(n^3)$.

3. Building blocks for the Shor's algorithm

This chapter discusses the Shor's algorithm. In section 3.1, we introduce the Standard Quantum Fourier Transform and the Semi classical Quantum Fourier Transform. In section 3.2, we discuss the Quantum Phase Estimation. The Shor's algorithm is discussed in section 3.3 and section 3.4. In section 3.5, we give 3 theorems that gives a reliability of getting a solution from Shor's algorithm even though it's probabilistic. Detailed discussion of the Modular exponentiation operation based on the Quantum adder is discussed in section 3.6. We end the chapter with a discussion of some implementation variants of the Shor's algorithm in section 3.7 and the complexity analysis of the Shor's algorithm including its subroutines in section 3.8.

3.1. Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is the quantum implementation of the Discrete Fourier Transform (DFT) over the amplitudes of a wavefunction. DFT acts on a vector $(x_0, ..., x_{N-1})$ and maps it to the vector $(y_0, ..., y_{N-1})$ according to the formula

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk} \ , \qquad \text{where} \qquad \omega_N^{jk} = e^{2\pi i \frac{jk}{N}}. \tag{3.1}$$

Similarly, the QFT acts on a quantum state $\sum_{i=0}^{N-1} x_i |i\rangle$ and maps it to the quantum state $\sum_{i=0}^{N-1} y_i |i\rangle$ according to the formula 3.1 where $|i\rangle$ is in decimal digits representation so that as an example, for 2-qubits, $|2\rangle = |10\rangle$.

The QFT can also be expressed as the map:

$$|x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{xy} |y\rangle$$

Or the unitary matrix:

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \omega_N^{xy} |y\rangle\langle x|$$

QFT transforms between the computational (Z) basis, and the Fourier basis for example, the H-gate implements the QFT for single-qubit systems.

$$|\text{State in the Computational Basis}\rangle \xrightarrow[QFT]{} |\text{State in the Fourier Basis}\rangle$$

Let $N = 2^n$ and let QFT_N act on the state $|x\rangle = |x_1 \dots x_n\rangle$. Then we have that,

$$
\begin{aligned}
QFT_N|x\rangle &= \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}\omega_N^{xy}|y\rangle \\
&= \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}e^{2\pi ixy/2^n}|y\rangle \qquad \text{since } \omega_N^{xy} = e^{2\pi i\frac{xy}{N}} \text{ and } N = 2^n \\
&= \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}e^{2\pi i(\sum_{k=1}^{n} y_k/2^k)x}|y_1\dots y_n\rangle \qquad \left[\, y/2^n = \sum_{k=1}^{n} y_k/2^k \text{ in fractional binary notation} \,\right] \\
&= \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}\prod_{k=1}^{n}e^{2\pi ixy_k/2^k}|y_1\dots y_n\rangle \\
&= \frac{1}{\sqrt{N}}\bigotimes_{k=1}^{n}\left(|0\rangle + e^{2\pi ix/2^k}|1\rangle\right) \qquad \left[\text{by expanding } \sum_{y=0}^{N-1} = \sum_{y_1=0}^{1}\sum_{y_2=0}^{1}\dots\sum_{y_n=0}^{1}\right] \\
&= \frac{1}{\sqrt{N}}\left(|0\rangle + e^{\frac{2\pi i}{2}x}|1\rangle\right)\otimes\left(|0\rangle + e^{\frac{2\pi i}{2^2}x}|1\rangle\right)\otimes\dots\otimes\left(|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}|1\rangle\right)\otimes\left(|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle\right)
\end{aligned}
\tag{3.2}
$$

3.1.1. Standard QFT circuit

The circuit implementing the standard QFT uses two gates. The first gate is a single-qubit Hadamard gate H, and its action on the single-qubit state $|x_k\rangle$ is

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

The second gate is a two-qubit controlled rotation $CROT_k$ given in block-diagonal form as

$$CROT_k = \begin{bmatrix} I & 0 \\ 0 & UROT_k \end{bmatrix} \text{ where } UROT_k = \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\frac{2\pi i}{2^k}\right) \end{bmatrix}$$

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

$$CROT_k|0x_j\rangle = |0x_j\rangle \text{ and } CROT_k|1x_j\rangle = e^{\left(\frac{2\pi i}{2^k}x_j\right)}|1x_j\rangle$$

Using these gates, a circuit that implements an n-qubit QFT is given by figure 3.1
Starting with an n-qubit input state $|x_1x_2\dots x_n\rangle$, steps to implement the circuit are as follows:

1. Apply H-gate to qubit 1 transforming the input state to

$$H_1|x_1x_2\dots x_n\rangle = \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2}x_1\right)|1\rangle\right]\otimes|x_2x_3\dots x_n\rangle$$

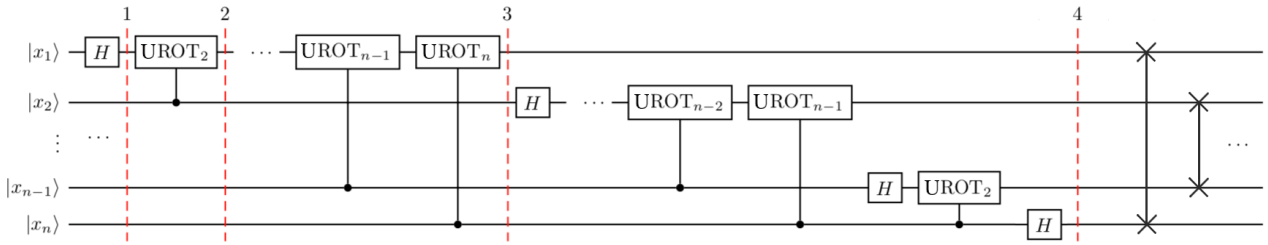


Figure 3.1: The Quantum Fourier Transform Circuit for $N = 2^n$ [65]

2. For i = 2 to n, apply the $UROT_i$ gate on qubit 1 controlled by qubit i. for i = 2, the transformed state is now

$$\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1\right)|1\rangle\right] \otimes |x_2x_3\ldots x_n\rangle$$

3. After the last $UROT_n$ gate (i.e $i = n$) is applied on qubit 1 controlled by qubit n, the state becomes

$$\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^n}x_n + \frac{2\pi i}{2^{n-1}}x_{n-1} + \ldots + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1\right)|1\rangle\right] \otimes |x_2x_3\ldots x_n\rangle$$

which is equivalent to

$$\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^n}x\right)|1\rangle\right] \otimes |x_2x_3\ldots x_n\rangle$$

since

$$x = 2^{n-1}x_1 + 2^{n-2}x_2 + \ldots + 2^1x_{n-1} + 2^0x_n$$

4. After applying a similar sequence of gates for qubits $2, \ldots, n$, the final state is:

$$\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^n}x\right)|1\rangle\right] \otimes \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^{n-1}}x\right)|1\rangle\right] \otimes \ldots$$
$$\otimes \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^2}x\right)|1\rangle\right] \otimes \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^1}x\right)|1\rangle\right]$$

This is the QFT of the input state as shown in equation 3.2 but with the qubits reversed in the output state. Hence to get the desired QFT, we apply swap operations to reverse the order of the output qubits. As the QFT circuit becomes larger, an increasing amount of time is spent doing increasingly slight rotations. However with **approximate** QFT, we can ignore rotations below a certain threshold and still get good results. This is also important in physical implementations, as reducing the number of operations can greatly reduce decoherence and potential gate errors. However, to further reduce the gate errors, techniques such as Quantum Error Correction using repetition codes is used. More on this can be found in [29].

In IBM Qiskit [24], the implementation of the $CROT$ gate is a controlled phase rotation gate and defined as

$$CP(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{bmatrix}$$

Hence, the mapping from the $CROT_k$ gate used above into the $CP(\theta)$ gate is

$$\theta = 2\pi/2^k = \pi/2^{k-1}$$

This implementation of the QFT on n qubits requires $O(n^2)$ operations [29] however, in the physical implementations, there are thresholds for the precision of the gates. This is because many phase shifts will be very small and almost negligible that we could ignore the phase shifts with $k > k_{max}$ where k_{max} is a given threshold. Coppersmith [37] showed that the error introduced by ignoring all gates with $k > k_{max}$ is proportional to $n2^{-k_{max}}$. Hence we can choose $k_{max} \in O(log(\frac{n}{\epsilon}))$.

The implementation of this approximate version of the QFT on n qubits requires $O(nlog(n))$ gates. The depth of either the exact or approximate QFT can be reduced below $O(n)$ but it requires using extra qubits in parellelization techniques as shown in [37]. Hence the depth of the standard QFT on $n+1$ qubits is $O(n)$.

3.1.2. Semiclassical QFT ('One controlling qubit')

Robert Griffiths and Chi-Sheng Niu in 1995 presented an alternative way to perform the standard QFT [39]. Their idea was from the assumption that two-qubit gates could be difficult to construct than single-qubit gates in their physical implementation. Hence they proposed a circuit which used only one-qubit gates that were classically controlled.

If we interchange the roles of the target and control qubits of the $UROT_i$ gates in figure 3.1, we observe that after the application of the H gate on the i^{th} qubit, no other gate operates on it (except the swap operation) till it is measured. Hence it is possible not to delay the measurements and instead measure the i^{th} qubit just after the application of the H gate. We can then use the measurement result which is a classical bit value to classically control the $UROT_i$ gates. As we will see in section 3.7, using the semiclassical QFT can be used to optimize the number of qubits used in the Shor's factoring algorithm.

3.2. Quantum Phase Estimation (QPE)

Given a unitary operator U, the aim of the Quantum Phase Estimation (QPE) algorithm is to estimate the phase θ of U in the relation $U|\psi\rangle = e^{\mathbf{2\pi i\theta}}|\psi\rangle$. Here $|\psi\rangle$ is an eigenvector of U and $e^{\mathbf{2\pi i\theta}}$ is its corresponding eigenvalue. The general quantum circuit for QPE is shown in figure 3.2.

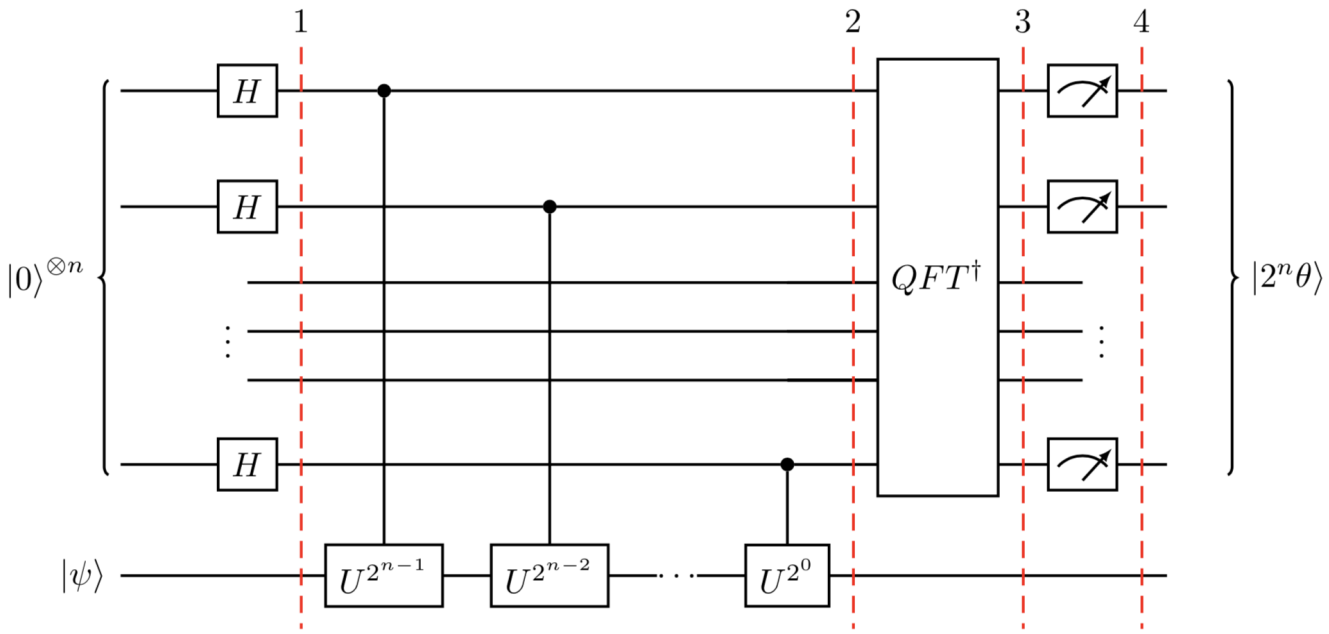


Figure 3.2: The Quantum Phase Estimation Circuit for $N = 2^n$ [65]

The top register (also referred to as counting register) contains n qubits, and the bottom register contains qubits in the state of the eigenvector $|\psi\rangle$. QPE algorithm leverages the effect of phase kickback to write the phase of U to the n qubits in the counting register. This phase of U denoted by θ is in the Fourier basis and $QFT^\dagger$ is then used to translate this from the Fourier basis into the computational basis, which is finally measured.

Due to phase kickback, when a qubit is used to control the U-gate, the qubit will rotate proportionally to the phase $e^{2i\pi\theta}$. Hence, successive CU-gates can be used to repeat this rotation the right amount of times until (θ) has been encoded within the interval $0 \leq \theta \leq 2^n$ in the Fourier basis. Lastly, the inverse QFT is applied which transforms this to the computational basis. Steps to implement the circuit are as follows:

1. **Circuit setup**: We setup two registers as follows. The first register (counting register) will contain n qubits with which it will store the value $2^n\theta$. The second register will store $|\psi\rangle$ so that after the circuit setup, we will have:

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

2. **Put qubits in counting register in a state of superposition**: For the n qubits in the counting register, we will apply $H^{\otimes n}$ to get:

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

3. **Application of controlled unitary operations $(C-U)$**: The $C-U$ operations applies U on the target register when its control bit is in the $|1\rangle$ state. Given an eigenvector $|\psi\rangle$ of U, we have that $U|\psi\rangle = e^{\mathbf{2\pi i\theta}}|\psi\rangle$ and since U is unitary, it implies that

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

Hence after the application of all the n $C{-}U^{2^j}$ operations where $0 \leq j \leq n-1$, we use the relation $|0\rangle \otimes |\psi\rangle + |1\rangle \otimes e^{2\pi i\theta}|\psi\rangle = \left(|0\rangle + e^{2\pi i\theta}|1\rangle\right) \otimes |\psi\rangle$ to get:

$$\begin{aligned}\psi_2 &= \frac{1}{2^{\frac{n}{2}}}\left(|0\rangle + e^{\mathbf{2\pi i\theta 2^{n-1}}}|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{\mathbf{2\pi i\theta 2^1}}|1\rangle\right) \otimes \left(|0\rangle + e^{\mathbf{2\pi i\theta 2^0}}|1\rangle\right) \otimes |\psi\rangle \\ &= \frac{1}{2^{\frac{n}{2}}}\sum_{k=0}^{2^n-1} e^{\mathbf{2\pi i\theta k}}|k\rangle \otimes |\psi\rangle\end{aligned} \tag{3.3}$$

where k is the representation of n-bit binary numbers in base 10.

4. **Application of $QFT^\dagger$**: From the result in equation 3.3 and equation 3.2, $x = 2^n\theta$. Hence, an inverse Fourier transform is applied on the counting register to retrieve the state $|2^n\theta\rangle$ and hence the output is given below

$$|\psi_3\rangle = \frac{1}{2^{\frac{n}{2}}}\sum_{k=0}^{2^n-1} e^{\mathbf{2\pi i\theta k}}|k\rangle \otimes |\psi\rangle \xrightarrow[\mathcal{QFT}_n^{-1}]{} \frac{1}{2^n}\sum_{x=0}^{2^n-1}\sum_{k=0}^{2^n-1} e^{-\frac{2\pi ik}{2^n}(x-2^n\theta)}|x\rangle \otimes |\psi\rangle \tag{3.4}$$

5. **Take measurements of the counting register**: Equation 3.4 peaks near $x = 2^n\theta$. If $2^n\theta$ is an integer, then with high probability, measurement of the counting register in the computational basis gives the phase:

$$|\psi_4\rangle = |2^n\theta\rangle \otimes |\psi\rangle$$

If $2^n\theta$ is not an integer, equation 3.4 still peaks near $x = 2^n\theta$ with probability better than $4/\pi^2 \approx 40\%$ as given in [29].

3.3. Shor's Algorithm

Quantum factorization generally consists of classical pre-processing, quantum algorithm for order-finding and classical post-processing [29, 41, 42]. The only use of quantum computation in Shor's algorithm is in order finding (indicated by the black arrow in figure 3.3) which entails finding the order r of a modulo N, where N is an n -bit integer to be factored.

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

The factoring algorithm for N as given in [29] is:

1. If N is even, return the factor 2.
2. Determine classically if $N = p^q$ for $p \geq 1$ and $q \geq 2$ and if so return the factor p.

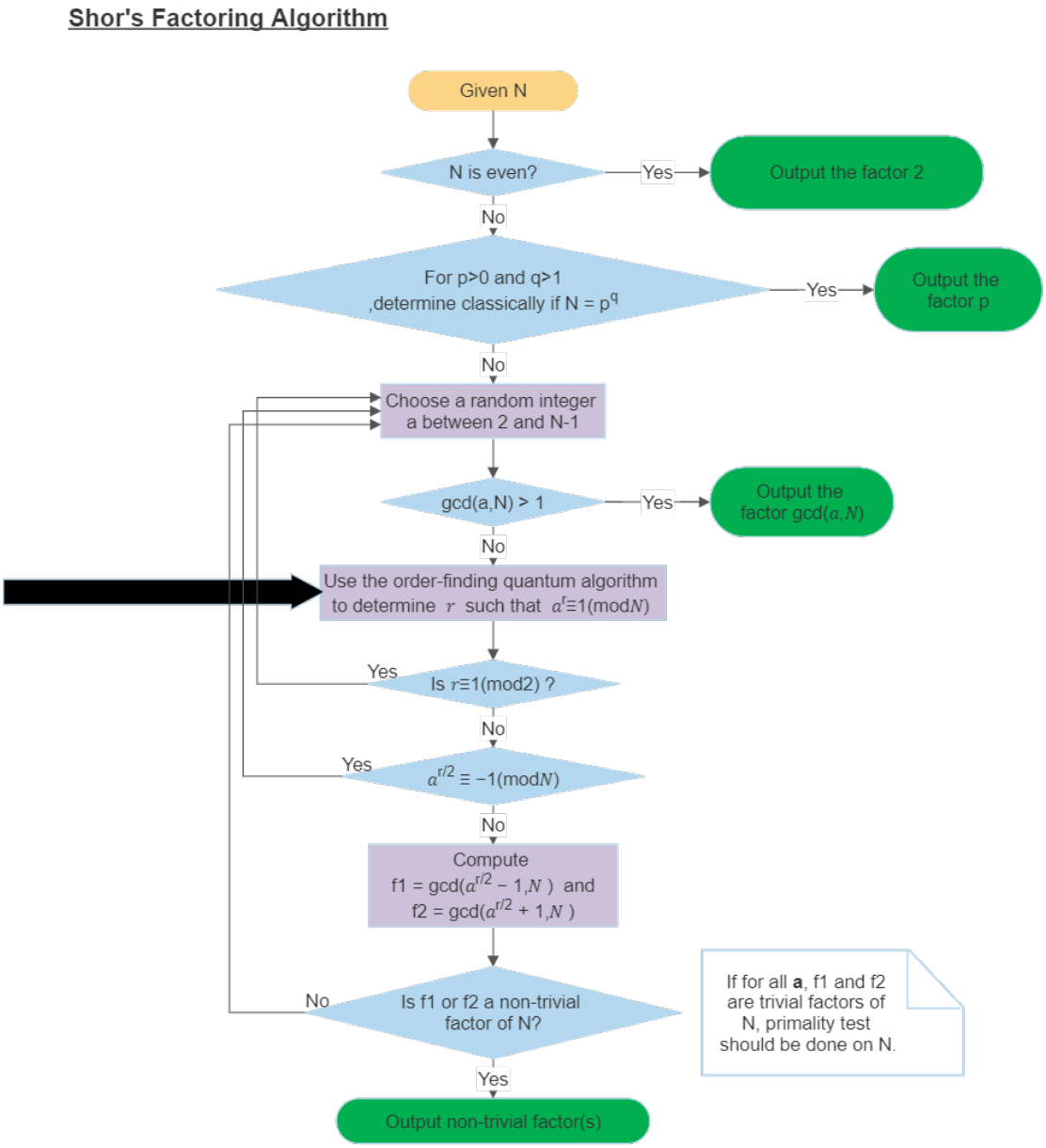


Figure 3.3: Overview of Shor's factoring algorithm

3. Choose a random number a such that $1 < a \leq N - 1$. Using Euclid's algorithm, determine if $\gcd(a, N) > 1$ and if so, return the factor $\gcd(a, N)$

4. Use the order-finding quantum algorithm to find the order r of a modulo N.

5. If r is odd or r is even but $a^{r/2} = -1(\mathrm{mod}N)$, then go to step (3). Else, compute $\gcd\left(a^{r/2} - 1, N\right)$ and $\gcd\left(a^{r/2} + 1, N\right)$. Test to see if one of these is a non-trivial factor of N, and if so, return the factor.

With at least 50% probability, r will be even and $a^{r/2} \neq -1(\mathrm{mod}N)$ [29, 42]. The quantum part of the algorithm (i.e. step 4) is computable in polynomial time on a quantum computer and we can build the order-finding circuit using a polynomial number of elementary gates and a linear number of qubits [42]. While the best-known classical algorithm requires super-polynomial time to factor the product of two primes, Shor's algorithm does

this in polynomial time. Here, our focus will be on the quantum part of Shor's algorithm, which solves the problem of period finding. Since, a factoring problem can be turned into a period finding problem in polynomial time, an efficient period finding algorithm can be used to factor integers efficiently too.

3.4. Period finding

Consider the periodic function:

$$f(x) = a^x \bmod N$$

where a and N are positive integers, $a < N$, and $gcd(a, N) = 1$. The period, or order (r), is the smallest (non-zero) integer such that

$$a^r \bmod N = 1$$

Shor's solution was to use QPE on U defined as:

$$U|y\rangle \equiv |ay \bmod N\rangle$$

Starting in the state $|1\rangle$, each successive application of U will multiply the state of our register by $a \pmod N$, and after r applications, we will have the state $|1\rangle$ again.
So a superposition of the states in this cycle ($|u_0\rangle$) would be an eigenstate of U:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |a^k \bmod N\rangle$$

An interesting eigenstate could be one in which the phase of the kth state is proportional to k hence the phase is different for each of these computational basis states:

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i k}{r}} |a^k \bmod N\rangle$$

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

Hence we see that the eigenvalue contains r which ensures that the phase differences between the r computational basis states are equal. However, there are other eigenstates with this behaviour and so to generalise, the phase difference is multiplied by an integer s and which then appears in the eigenvalue as given in equation 3.5:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} |a^k \bmod N\rangle$$

$$U|u_s\rangle = e^{\frac{2\pi i s}{r}} |u_s\rangle \tag{3.5}$$

From equation 3.5, we have a unique eigenstate for each integer value of s where

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

Summing up all these eigenstates, the different phases cancel out all computational basis states except $|1\rangle$ to imply that $|1\rangle$ is a superposition of the eigenstates of U:

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

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

$$\phi = \frac{s}{r} \qquad \text{where } s \text{ is a random integer satisfying } 0 \leq s \leq r-1$$

Finally to obtain the period r, continued fractions algorithm on ϕ is used. The circuit diagram for the Shor's period finding algorithm is shown below

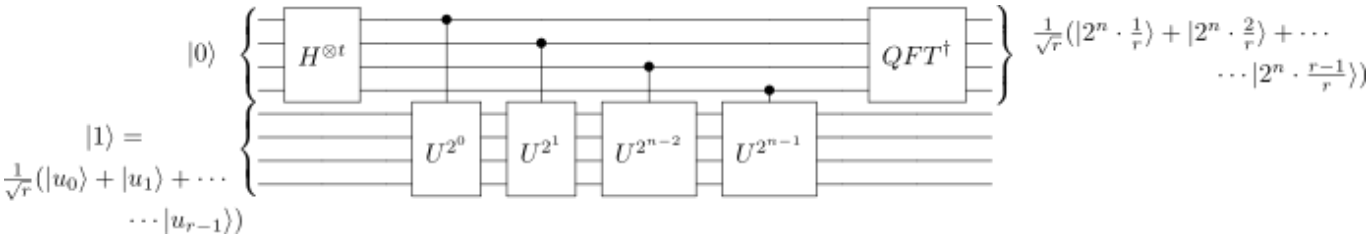


Figure 3.4: Shor's period finding algorithm for U^{2^j} [65]

As shown in fig 3.4, creating the U^{2^j} gates by repeating U grows exponentially (rather than polynomially) with j. Hence we consider this operator:

$$U^{2^j}|y\rangle = |a^{2^j} y \bmod N\rangle$$

This grows polynomially with j and so the calculation $a^{2^j} \bmod N$ is efficiently possible. The repeated squaring algorithm usable in classical computers can be used to calculate an exponential and this algorithm is even simpler in the case of exponentials in the form 2^j. However, even though this algorithm scales polynomially with j, modular exponentiation circuits are more involved and is one of the constraints in Shor's algorithm. A simplified version of the implementation is seen in reference [28] and discussed in section 3.6.

3.5. Reliability of getting a solution from Shor's algorithm

After taking measurements in the Shor's period finding algorithm, we get the phase ϕ and by the continued fraction algorithm, we are guaranteed for a precision level to get a rational number sufficiently close as given in the following theorem in [29].

Theorem 1. *Suppose s/r is a rational number such that*

$$\left|\frac{s}{r} - \varphi\right| \leq \frac{1}{2r^2}$$

Then s/r is a convergent of the continued fraction for φ, and thus can be computed in $O(n^3)$ operations using the continued fractions algorithm.

The claim about step 5 in Shor's algorithm described in section 3.3 is confirmed by the following theorem [29]:

Theorem 2. *Suppose N is an n bit composite number, 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 $O(n^3)$ operations.*

By repeatedly choosing a random number a as specified in step 3 of section 3.3, we are almost guaranteed that we won't have to do so many trials as the probability approaches 1 as the number of trials increases. This is summarized in the following theorem as given in [29]:

Theorem 3. *Suppose $N = p_1^{\alpha_1} \dots p_m^{\alpha_m}$ is the prime factorization of an odd composite positive integer. Let x be an integer chosen uniformly at random, subject to the requirements that $1 \leq x \leq N-1$ and x is co-prime to N. Let r be the order of x modulo N. Then $p\left(r \text{ is even and } x^{r/2} \neq -1(\mathrm{mod}N)\right) \geq 1 - \frac{1}{2^m}$*

3.6. Modular exponentiation in Shor's algorithm

As shown in figure 3.4, the order finding algorithm requires several smaller quantum circuits from the primitive Quantum addition up to the more involved modular exponentiation circuits. All the subroutines that makes up the Shor's algorithm are discussed in this section and closely follows the idea presented in [28].

Quantum addition

There are two major variants of the addition circuit that could be used in Modular exponentiation namely Plain adder and Quantum Adder. Many variants of the quantum addition circuits have been proposed and for the modular exponentiation circuit explored in this thesis, we will use the version proposed by Draper [43] and shown in figure 3.5. This circuit comprises of 2 registers namely A and B. Register A takes n qubits as input

representing a number a, and register B takes n qubits containing the QFT of another number b denoted as $\Phi(b)$. After the addition operation, register A remains the same but register B now contains the QFT of $(a+b)$ mod 2^n which we denote as $\Phi(a+b)$. Since we want to find the period of the function a^x mod N where $a < N$ is a classical random variable, it's sufficient to only add a classical value to the quantum register. Hence, the qubits representing a can be changed to classical bits and so the gates are classically controlled.

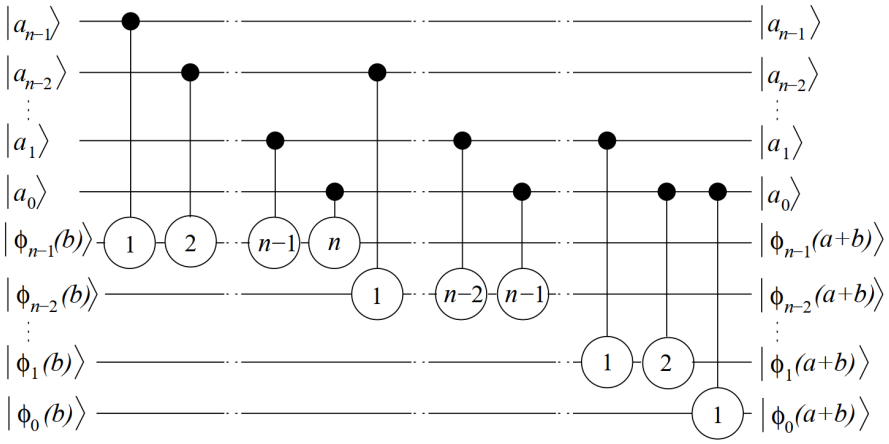


Figure 3.5: Quantum addition as given in [28]

QFT Adder gate

Next, since the addition is performed in the Fourier space, the circuit can be denoted as the $\Phi ADD(a)$ gate. This gate is shown in figure 3.6 with a thick black bar on the right of the gate symbol to differentiate it from its unitary inverse. To prevent addition overflow, $n+1$ qubits are needed in register B so that $\Phi(b)$ denotes the QFT of $(n+1)$ qubit register containing a n-bit number.

Applying the unitary inverse of $\Phi ADD(a)$ (denoted $\Phi^{-1}ADD(a)$) with input $\Phi(b)$, either $\phi\left(2^{n+1}-(a-b)\right)$ if $b < a$ or $\phi(b-a)$ if $b \geq a$ is gotten. This $\Phi^{-1}ADD(a)$ gate is used for subtraction and comparing. This gate is shown in figure 3.7 with a thick black bar on the left of the gate symbol.

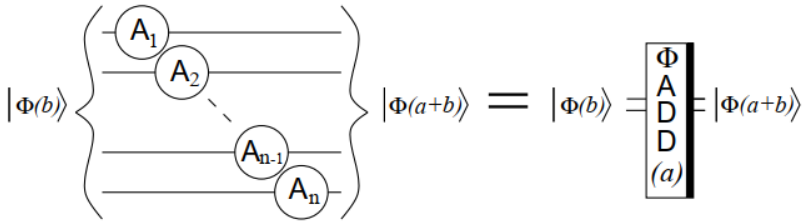


Figure 3.6: Adder gate [28]

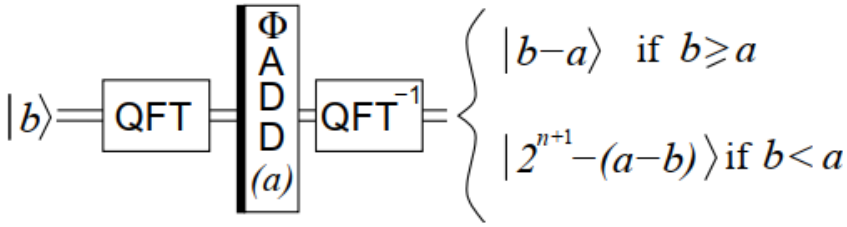


Figure 3.7: Adder gate inverse [28]

Modular adder gate

The $\Phi ADD(a)$ gate, is used in building a modular adder gate (figure 3.8) and we denote it as $\Phi ADD(a)$ mod N. For this gate, $a+b$ is first computed and N subtracted from it if $a+b \geq N$. Its input are $\Phi(b)$ with $b < N$ and a classical number a also satisfying $a < N$. To create the $\Phi ADD(a)$ mod N gate, we do the following:

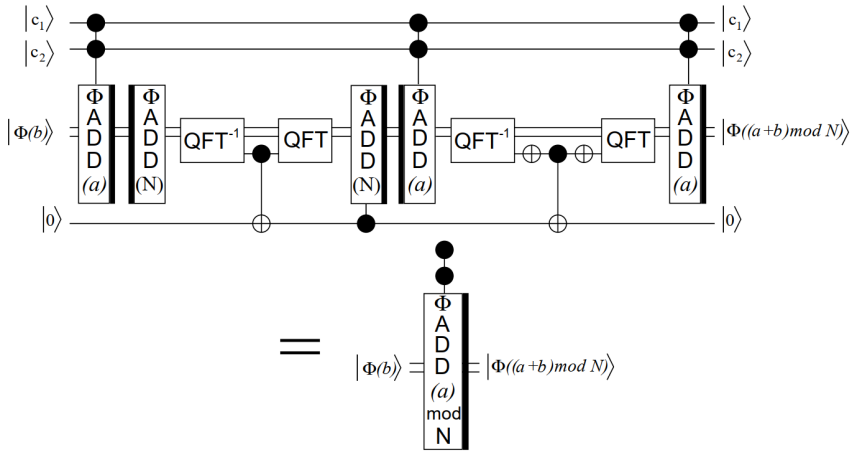


Figure 3.8: Modular adder gate [28]

1. Apply $\Phi ADD(a)$ gate to register $\Phi(b)$ to contain $\Phi(a+b)$ with no overflow
2. Apply $\Phi^{-1}ADD(N)$ to get $\Phi(a+b-N)$
3. Apply $QFT^{\dagger}$ on the whole register to access the Most Significant Bit (MSB)
4. Use the qubit from step 3 as the controlling qubit of a C-NOT gate acting on an ancillary qubit
5. Reapply QFT and use the ancillary qubit in step 4 as the control qubit for a $\Phi ADD(N)$ controlled gate. This gives $\Phi((a+b) \bmod N)$ in the register.
6. To restore the ancilla to $|0\rangle$, the identity 3.6 below is used:

$$(a+b) \bmod N \geq a \Leftrightarrow a+b < N \tag{3.6}$$

7. To get the most significant qubit of $(a+b) \bmod (N-a)$ so as to compare $(a+b) \bmod N$ with a, we apply $\Phi^{-1}ADD(a)$ and then $QFT^{\dagger}$

8. Apply a NOT gate on this most significant qubit and use it as the controlling qubit of a $C-NOT$ with the ancilla as the target.

9. Apply another NOT gate on the most significant qubit

10. Apply QFT and $\Phi ADD(a)$ gates on the register B. This gives the computation of $\Phi((a+b) \bmod N)$ with clean ancilla.

In the Shor's algorithm, we use a doubly controlled $\Phi ADD(a) \bmod N$. To do this with low complexity of the circuit, only the $\Phi ADD(a)$ gates are doubly controlled and two control qubits ($|c_1\rangle$ and $|c_2\rangle$) are added. By close inspection, we observe that if the $\Phi ADD(a)$ gates are switched off, then the entire circuit implements the identity gate on all qubits since $b < N$.

Controlled-SWAP

The controlled-$SWAP$ gate is composed of two controlled-NOT and one Toffoli gates as depicted in figure 3.9.

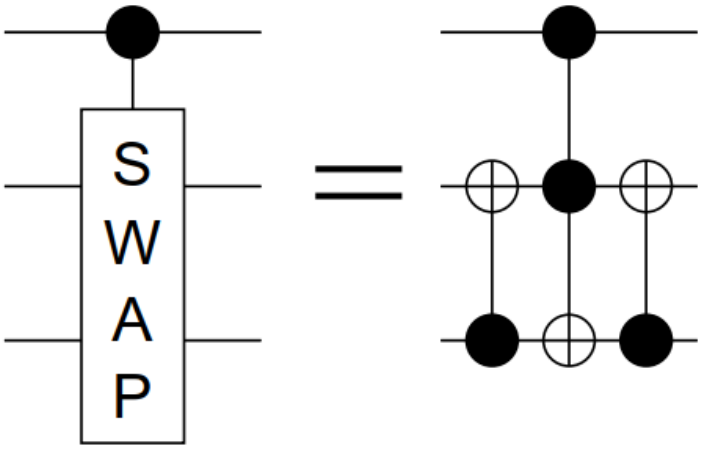


Figure 3.9: Controlled SWAP gate [28]

This performs a SWAP operation on two qubits controlled by a third qubit and needs $O(n)$ of the controlled-$SWAP$ gates to SWAP n qubits in a controlled manner.

Controlled multiplier gate

The doubly controlled $\Phi ADD(a) \bmod N$ gate is used in building a controlled multiplier gate denoted $CMULT(A) \bmod N$ as shown in figure 3.10. Its inputs are $|c\rangle|x\rangle|b\rangle$ and its output is dependent on the value of the qubit $|c\rangle$ as follows:

1. If $|c\rangle = |1\rangle$, its output is $|c\rangle|x\rangle|b + (ax) \pmod N\rangle$

2. If $|c\rangle = |0\rangle$, its output remains as $|c\rangle|x\rangle|b\rangle$

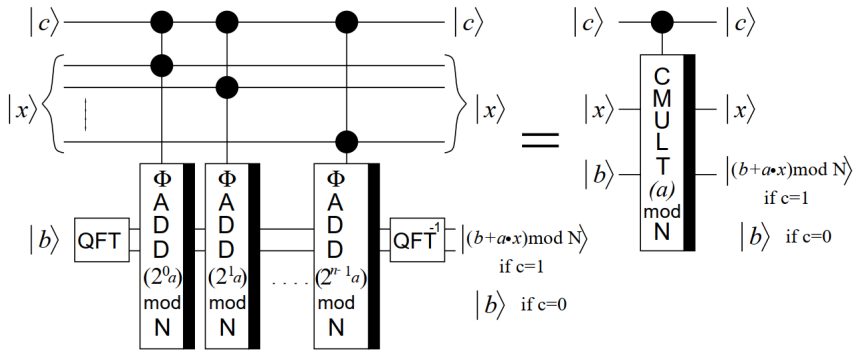


Figure 3.10: Controlled multiplier gate [28]

To implement the $CMULT(A)$ mod N gate, the following identity 3.7 is used:

$$(ax) \bmod N = \left(\ldots\left(\left(2^0 ax_0\right) \bmod N + 2^1 ax_1\right) \bmod N + \ldots + 2^{n-1} ax_{n-1}\right) \bmod N \quad (3.7)$$

From identity 3.7, to create the $CMULT(A)$ mod N gate, only n successive doubly controlled $\Phi ADD(a)$ mod N gates are needed with each adding a different value $(2^i a)$ mod N for $0 \leq i < n$. After these n operations, the output is $|x\rangle|b\rangle \longrightarrow |x\rangle|b + (ax) \bmod N\rangle$ however, our objective is to compute $|x\rangle \longrightarrow |(ax) \bmod N\rangle$. To do this, two controlled multiplication and one swap gate is used as shown in figure 3.11 and steps are outlined as follows:

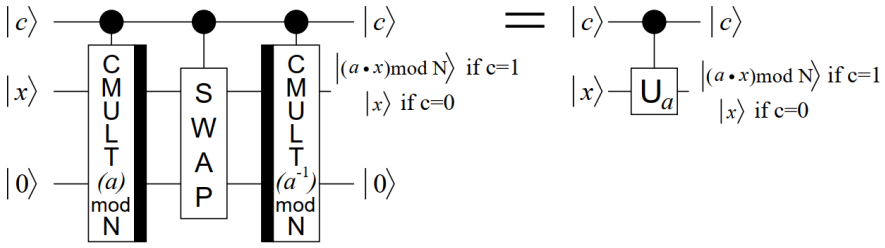


Figure 3.11: Controlled U_a gate [28]

1. Apply the $CMULT(A)$ mod N gate to $|c\rangle|x\rangle|0\rangle$
2. Apply a controlled-$SWAP$ between the two registers (i.e. apply $SWAP$ if $|c\rangle = |1\rangle$) for n qubits since the most significant qubit of (ax) mod N will always be 0 because of the one extra qubit storing the overflow in the $\Phi ADD(a)$ gate.
3. Apply the $CMULT(a^{-1})$ mod N gate where a^{-1} is the inverse of a and can be computed classically in polynomial time using the Extended Euclidean algorithm and will exist since a and N are co-prime. Hence $CMULT(a^{-1})$ mod N transforms $|c\rangle|x\rangle|b\rangle \Longrightarrow |c\rangle|x\rangle|(b - a^{-1}x) \bmod N\rangle$.

The resulting gate from these series of operations will be denoted $C-U_a$ for controlled-U_a and its action is summarily as follows:

1. if $|c\rangle = |0\rangle$, it does nothing

2. if $|c\rangle = |1\rangle$, the two registers are transformed as follows:

$$|x\rangle|0\rangle \rightarrow |x\rangle|(ax) \bmod N\rangle \rightarrow |(ax) \bmod N\rangle|x\rangle \rightarrow |(ax) \bmod N\rangle\,|(x - a^{-1}ax) \bmod N\rangle \\ = |(ax) \bmod N\rangle|0\rangle$$

Since at the end of the computation, the bottom register returns to $|0\rangle$, this extra register can be considered as part of the $C - U_a$ gate implying that

$$C - U_a : |x\rangle \longrightarrow |(ax) \bmod N\rangle$$

This is exactly the needed gate in the quantum order-finding circuit as shown in figure 3.4. To get the modular exponentiation operations $(C - U_a)^n$, $C - U_{a^n}$ can be computed directly since:

$$(a^n x) \bmod N = \underbrace{(a \ldots (a(ax) \bmod N) \bmod N \ldots) \bmod N}_{n \text{ times}}.$$

where $a^n \bmod N$ is computed classically.

3.7. Implementation variants of the Shor's algorithm

In this section, we will discuss possible optimization variants of the Shor's factoring algorithm. The optimizations apply to the QFT and Quantum modular exponentiation subroutines. Here, we will focus on minimizing the number of qubits needed for a successful implementation of the Shor's algorithm.

There are two versions of the Quantum modular exponentiation operation. One uses a classical adder while the second uses a QFT adder. The QFT adder has been discussed in this thesis in section 3.6 while more details on the classical plain adder which uses the NOT, CNOT and Toffoli gates can be found in [23].

In section 3.1, we discussed two variants of the QFT namely the standard QFT and the semiclassical QFT. Different combinations of the Quantum modular exponentiation and QFT can be selected to implement the Shor's factoring algorithm. These are as follows:

1. Variant 1 (Classical approach) [23]: Quantum modular exponentiation with classical adder + Standard QFT: In this variant the algorithm needs $7n + 3$ qubits where n is the number of bits of the number N to be factored.

2. Variant 2: Quantum modular exponentiation with classical adder + semiclassical QFT. This however does not reduce the number of qubits required hence we omit this.

3. Variant 3: Quantum modular exponentiation with QFT adder + Standard QFT. This reduces the number of required qubits to $4n + 2$. This variant is simulated in section 4.3.2 and we will refer to it as the shor_standard_QFT variant.

4. Variant 4: Quantum modular exponentiation with QFT adder + semiclassical QFT. This further optimizes the number of required qubits to $2n + 3$. This variant is simulated in section 4.3.3 and we will refer to it as the shor_semiclassical_QFT variant.

Everything we need to know about the shor_standard_QFT variant has been discussed in section 3.1.1 and 3.4 and the idea of the Semiclassical QFT has been discussed in section 3.1.2. Hence what remains is to discuss how Semiclassical QFT can be incorporated in the Shor's factoring algorithm.

3.7.1. Shor's algorithm using Modular Exponentiation with Quantum Adder and Semiclassical QFT

Since in Shor's algorithm (section 3.3) immediately after the $QFT^{\dagger}$ is performed the register is measured, it makes it possible to interleave the measurements of each individual qubits with the steps of QFT. Then classical bit values of the measurements could be used in controlling subsequent quantum gates.

The idea of the Quantum Modular Exponentiation with QFT Adder semiclassical QFT using one controlling qubit instead of $2n$ qubits done in the standard QFT was proposed by Zalka [26]. Stephane Beauregard then worked on this idea and developed the circuit which implemented it as discussed in [28].

The quantum controlled rotations $C - U_{a^{2^j}}$ could be replaced with 'semi-classically' controlled rotations of the subsequent qubits [40, 26, 27]. Hence, the control bit is measured and, if the outcome is 1, the rotation is performed quantumly. This is possible because the controlled-U gates all commute and $QFT^{\dagger}$ can be applied semi-classically and so we can compute the inverse QFT semi-classically.

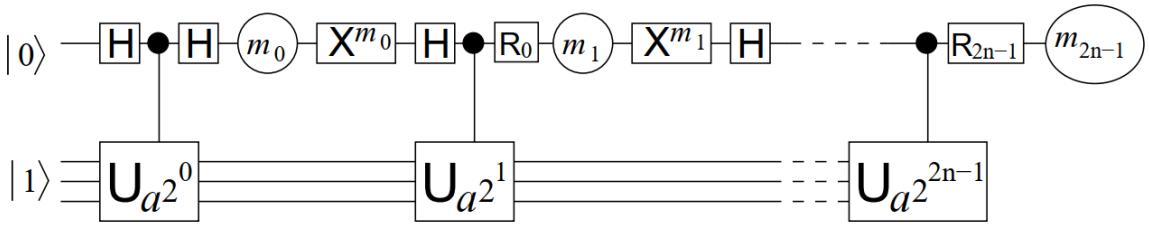


Figure 3.12: Factoring using the one control qubit technique [28]

By applying the operations sequentially as shown in figure 3.12, the answer can be gotten bit by bit in each iteration. Each measured bit determines the unitary transformation that will be applied after every controlled-U step before the next measurement. Hence this simulates the inverse Quantum Fourier Transform $QFT^{\dagger}$ being followed by a measurement as shown in figure 3.4. With this, only $2n + 3$ qubits are used to factor an n-bit number N and its complexity analysis will be shown in section 3.8.

3.8. Complexity Analysis of the Shor's Algorithm

Given an n-bit number N, the complexity analysis is done by keeping track of the number of qubits, order of the number gates and order of the depth of the circuit (and sub-circuits) in each subroutine that makes up the order finding circuit. We recall from section 3.4 and 3.6 that the order finding circuit uses only single qubit gates up to doubly controlled conditional phase shift gates and up to doubly $CNOT$ gates. As shown in [44], since these gates can be implemented using a constant number of single qubit gates and $CNOTs$, these gates will be considered as elementary quantum gates in this analysis.

The $\Phi ADD(a)$ gate in section 3.6 requires $n+1$ qubits (an extra qubit added to prevent addition overflow) and $O(n)$ single qubit gates in constant depth. When a control qubit is added to the circuit, the depth becomes $O(n)$ since the conditional phase shifts are sequentially performed.

The doubly controlled $\Phi ADD(a)$ mod N circuit in figure 3.8 requires $n+4$ qubits (that is $|c_1\rangle, |c_2\rangle, |0\rangle$ and $n+1$ in $|\Phi(b)\rangle$). It also requires $O(nk_{max})$ gates but has a depth of $O(n)$ independent of k_{max} because the $QFTs$ can be parallelized [28]. The $CMULT(a)$ mod N circuit is a total of n doubly controlled $\Phi ADD(a)$ mod N so requiring $2n+3$ qubits, $O(n^2k_{max})$ gates and a depth of $O(n^2)$.

The controlled-$SWAP$ on n qubits needs $O(n)$ gates and depth respectively hence the $C-U_a$ circuit which requires two of the $CMULT(a)$ mod N circuit and one controlled-$SWAP$ needs $2n+3$ qubits, $O(n^2k_{max})$ gates and a depth of $O(n^2)$.

For the entire order-finding circuit, $2n$ of the $C-U_a$ circuits are needed hence requiring $2n+3$ qubits, $O(n^3k_{max})$ gates and depth of $O(n^3)$. Using the exact QFT in the adder gate, $k_{max} = n$ while using the approximate QFT, $k_{max} = O\left(log\left(\frac{n}{\epsilon}\right)\right)$ and number of gates is $O(n^3log(n))$ for any ϵ polynomial in $\frac{1}{n}$.

Out of the $2n+3$ qubits used in this circuit, one is used as an ancilla for the modular addition, another is used to prevent overflow from the addition operation and n are used as an ancillary register to get modular multiplication from successive additions.

4. Simulation of Shor's Algorithm

We present in this chapter the results of the simulation of Shor's algorithm using a simulator. Firstly in section 4.1, we introduce the IBM Quantum Experience simulation environment. In section 4.2, we give a detailed breakdown of the calculations for a simplified case. Section 4.3 presents a detailed work-through of simulation cases using two variants of the algorithm and a constant optimized version. Section 4.4 presents and discusses the actual results of the different simulations.

4.1. Architecture of the Quantum Computer and Simulator Used

We used the IBM QASM_simulator available publicly on IBM Quantum Experience [24]. The key module on the IBM Quantum Experience platform is the Qiskit library and users can write quantum programs in the Python programming language. A Python user can also install the Qiskit library in their local Python environment and use locally. The IBM Quantum computing platform has both a graphical environment (called Quantum composer) and a source code editor in the form of Python Jupyter notebook (called Quantum lab). Due to the complicated nature of the circuit to implement the Shor's factoring algorithm and the need to automate several steps including the classical pre-processing and post-processing, we wrote program codes in the Quantum lab as it gives a lot of flexibility. Figure 4.1 shows the interface of the IBM Quantum lab.

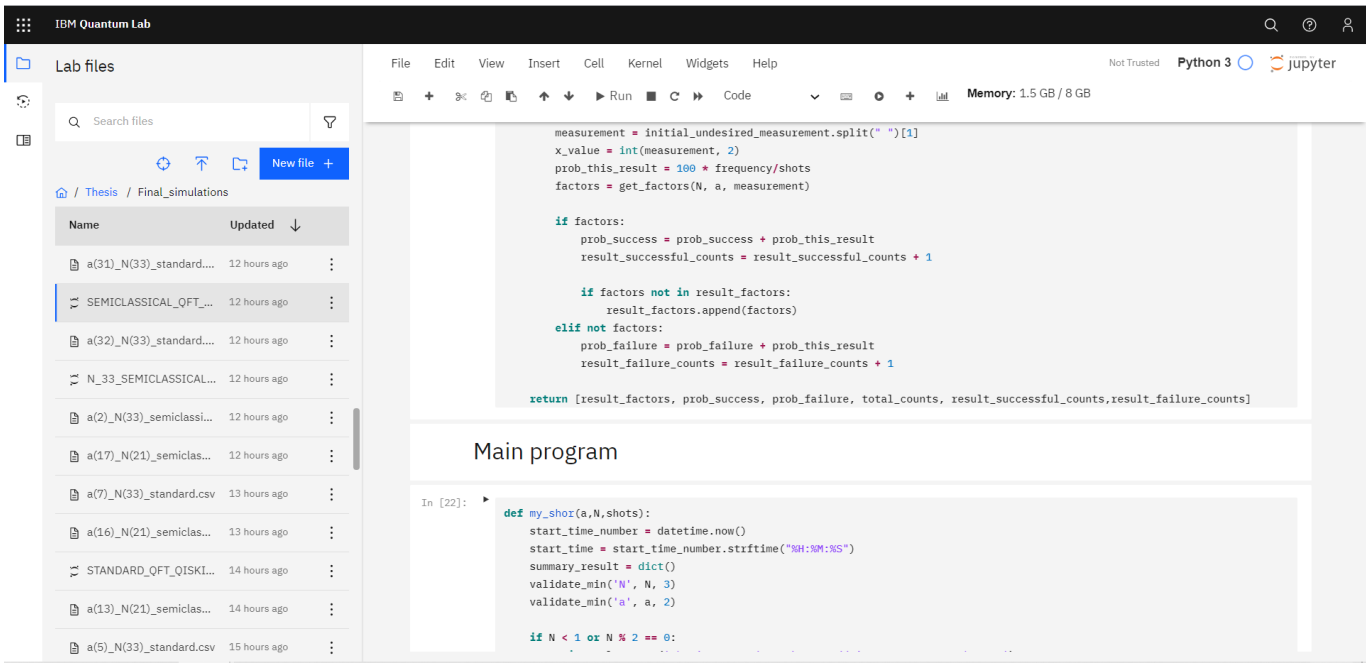


Figure 4.1: IBM Quantum Lab

IBM Qiskit Aer is a simulator for quantum circuits that includes realistic noise models and the QASM Simulator is the main Qiskit Aer backend. This backend simulates the

execution of a quantum circuit on a real quantum device and outputs measurement counts as well as the final quantum state vector of the device at the end of the simulation. Figure 4.2 summarizes the QASM simulator system specifications as at the time of running the simulations. We note that most of the parameters listed in the figure didn't change throughout the simulation period which took a few days.[1]

Version Information

Qiskit Software	Version
Qiskit	0.25.3
Terra	0.17.1
Aer	0.8.2
Ignis	0.6.0
Aqua	0.9.1
IBM Q Provider	0.12.3
System information	
Python	3.8.8 | packaged by conda-forge | (default, Feb 20 2021, 16:22:27) [GCC 9.3.0]
OS	Linux
CPUs	8
Memory (Gb)	31.400047302246094

Thu May 20 14:30:38 2021 UTC

Figure 4.2: Qiskit version information

4.2. Example using Shor's algorithm to factor $N = 15$ manually

In this section, we begin by factoring $N = 15$ manually using the steps in the Shor's algorithm outlined in 3.3. The steps are generally divided into 3 parts namely the classical pre-processing, quantum order finding algorithm and classical post-processing. The classical pre-processing steps are outlined below:

Step 1 (Check if N is even): $15 = 1 \pmod 2$ hence N is not even

Step 2: Check if $N = a^b$ for integers $a \geq 1$ and $b \geq 2$. Manually checking for all numbers up to $\sqrt{N}$ shows that N is not of the form a^b

[1] At first, the system specification and version information wasn't recorded during the trial and testing phases. However, the data used in the analysis were from experiments that ran under these exact specifications.

Step 3: Randomly choose x where $1 < x < 14$ for e.g. $x = 7$. Next compute $d = gcd(7, 15)$. By Euclidean algorithm,

$$15 = 7(2) + 1$$
$$7 = 1(7) + 0$$

Hence $d = 1$ implying 7 and 15 are co-prime.

Next, we move to the second part which uses the quantum order finding algorithm to obtain the order r of the chosen x mod N.

Step 4: Use the QPE to find the order r of 7 (mod 15). The number of bits n representing $N = 15$ is:

$$n = \lceil log_2(N + 1) \rceil = \lceil log_2(14) \rceil = 4$$

We initialize the circuit by creating 2 quantum registers ("counting_register(c)" and "acting_register(a)") and each containing 4 qubits. We also create 1 classical register containing 4 classical bits to store the result of the measurements. Then we perform the following QPE steps:

Step 4i: All qubits are initialized to ground state $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ as shown in figure 4.3 so we have $|0\rangle^{\otimes 4}|0\rangle^{\otimes 4}$

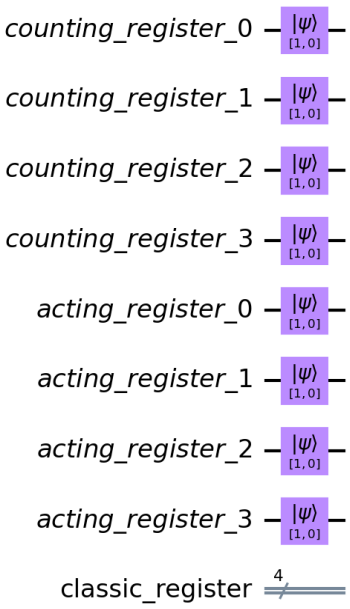


Figure 4.3: Initialization of circuit

Step 4ii: We apply Hadamard gate to the counting_register and Pauli-X gate to the last qubit in the acting_register as shown in figure 4.4 to get

$$\left[H^{\otimes 4}|0\rangle^{\otimes 4}\right]|0\rangle^{\otimes 4} = \frac{1}{4}[|0\rangle + |1\rangle + |2\rangle + \cdots + |15\rangle]|1\rangle$$

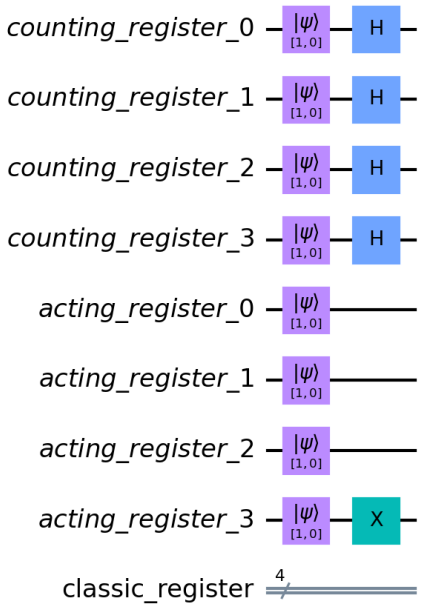


Figure 4.4: Application of Hadamard and X-gate

Step 4iii: We apply $U|y\rangle = |ay \pmod{15}\rangle$ and powers of $U|y\rangle$ given by $U^{2^j}|y\rangle = |a^{2^j}y \pmod{15}\rangle$ for $j = 0, \cdots, n-1$ to the qubits in the acting_register controlled by the qubits in the counting_register to get

$$= \frac{1}{4}\Big[|0\rangle|7^0(\text{mod } 15)\rangle + |1\rangle|7^1(\text{mod } 15)\rangle + |2\rangle|7^2(\text{mod } 15)\rangle + \cdots + |15\rangle|7^{15}(\text{mod } 15)\rangle\Big]$$

Expanding and simplifying further gives

$$= \frac{1}{4}\Big[|0\rangle|1\rangle + |1\rangle|7\rangle + |2\rangle|4\rangle + |3\rangle|13\rangle + |4\rangle|1\rangle + |5\rangle|7\rangle + |6\rangle|4\rangle + |7\rangle|13\rangle$$

$$+|8\rangle|1\rangle + |9\rangle|7\rangle + |10\rangle|4\rangle + |11\rangle|13\rangle + |12\rangle|1\rangle + |13\rangle|7\rangle + |14\rangle|4\rangle + |15\rangle|13\rangle\Big]$$

Step 4iv: Taking measurements on the acting_register (say we measured $|13\rangle$), we have:

$$\frac{1}{2}\left[|3\rangle + |7\rangle + |11\rangle + |15\rangle\right] \otimes |13\rangle$$

Step 4v: We apply $QFT^\dagger$ on the counting_register to get

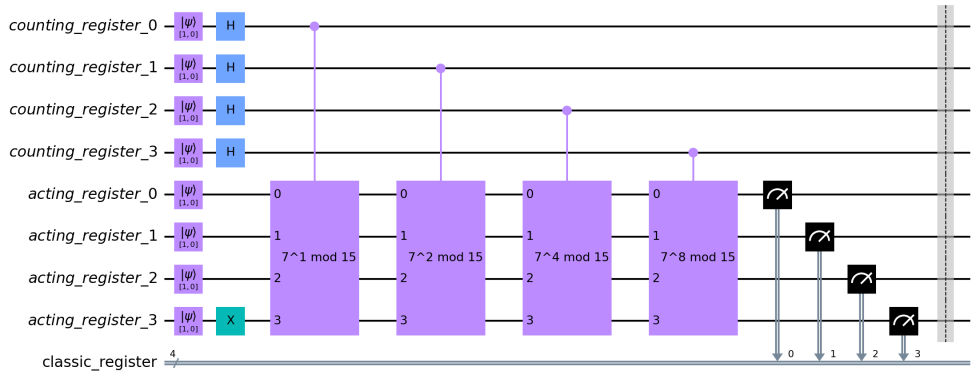


Figure 4.5: Application of the U-gate and measuring the acting_register

$$\begin{aligned} QFT^{\dagger}|3\rangle &= \frac{1}{\sqrt{16}} \sum_{y=0}^{15} e^{-2\pi i \cdot 3y/16} |y\rangle \\ &= \frac{1}{4} \sum_{y=0}^{15} e^{-\pi i \cdot 3y/8} |y\rangle \end{aligned} \tag{4.1}$$

$$QFT^{\dagger}|7\rangle = \frac{1}{4} \sum_{y=0}^{15} e^{-\pi i \cdot 7y/8} |y\rangle \tag{4.2}$$

$$QFT^{\dagger}|11\rangle = \frac{1}{4} \sum_{y=0}^{15} e^{-\pi i \cdot 11y/8} |y\rangle \tag{4.3}$$

$$QFT^{\dagger}|15\rangle = \frac{1}{4} \sum_{y=0}^{15} e^{-\pi i \cdot 15y/8} |y\rangle \tag{4.4}$$

$$\begin{aligned} QFT^{\dagger}|c\rangle &= \frac{1}{8} \sum_{y=0}^{15} \left[e^{-i \cdot \frac{3\pi y}{8}} + e^{-i \cdot \frac{7\pi y}{8}} + e^{-i \cdot \frac{11\pi y}{8}} + e^{-i \cdot \frac{15\pi y}{8}} \right] |y\rangle \\ &= \frac{1}{8} \left[4|0\rangle + 4i|4\rangle - 4|8\rangle - 4i|12\rangle \right] \end{aligned}$$

Where some terms cancelled illustrating how we leverage the effect of quantum interference.

Step 4vi: Measure the counting_register (figure 4.7) to get $|0000\rangle \equiv |0\rangle$ or $|0100\rangle \equiv |4\rangle$ or $|1000\rangle \equiv |8\rangle$ or $|1100\rangle \equiv |12\rangle$ with roughly equal probability of 1/4 as shown in figure 4.8.

At this point, we will continue with classical post-processing steps to obtain the order r and finally get the factors.

Step 5: Analysing the different possible results, we apply the continued fractions algorithms where necessary depending on the measured outcomes. Hence we have the following scenarios:

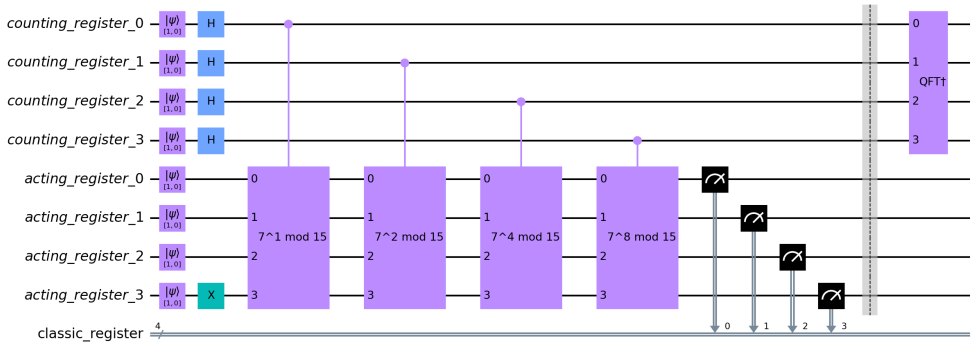


Figure 4.6: Application of the inverse QFT on the counting_register

1. Measured $|0\rangle$: This is trivial and we have to restart. So with probability $\approx \frac{1}{4}$, we were unsuccessful.

2. Measured $|4\rangle$: Phase $\phi = \frac{s}{r} = \frac{4}{2^4} = \frac{1}{4}$. Hence by the continued fractions algorithm in this easy case, $r = 4$. Since r is even, we proceed to the next classical post-processing step where we compute the following:

$$x \equiv a^{r/2} (\mathrm{mod} N) = 7^{4/2} (\mathrm{mod} N) = 4$$

   Now, $x + 1 = 5 \implies gcd(5, 15) = 5$ and $x - 1 = 3 \implies gcd(3, 15) = 3$. Hence, with probability $\approx \frac{1}{4}$, we have gotten the factor of $N = 3 \times 5$ in this case

3. Measured $|8\rangle$: Phase $\phi = \frac{s}{r} = \frac{8}{2^4} = \frac{1}{2}$. Hence $r = 2$ or $r = 4$. The latter case is similar as above for the case of measuring $|4\rangle$ and for the former, even though $7^2 \not\equiv 1 (\mathrm{mod} 15)$, we get one of the factors as follows:

$$x \equiv 7^{2/2} (\mathrm{mod} N) = 7$$

   Now, $x + 1 = 8 \implies gcd(8, 15) = 1$ and $x - 1 = 6 \implies gcd(6, 15) = 3$. with this first factor, the second factor is trivial to obtain. Hence with another probability $\approx \frac{1}{4}$, we have gotten the factor of $N = 3 \times 5$ in this case.

4. Measured $|12\rangle$: Phase $\phi = \frac{s}{r} = \frac{12}{2^4} = \frac{3}{4}$. Hence $r = 4$ and we get similar result as the case of measuring $|4\rangle$. So finally with another probability of $\approx \frac{1}{4}$, we have gotten the factor of $N = 3 \times 5$ in this case.

Overall, we have seen that with probability $\approx 3/4$, we get the factors of $N = 15 = 3 \times 5$ at the first run. Running this simulation more times for different starting values increases the chance of being successful in obtaining the desired factors.

This was a very easy case to illustrate the Shor's algorithm. As can be seen in figure 4.11, the period is 4 and as shown above, using this period gives us the factors of $N = 3 \times 5$.

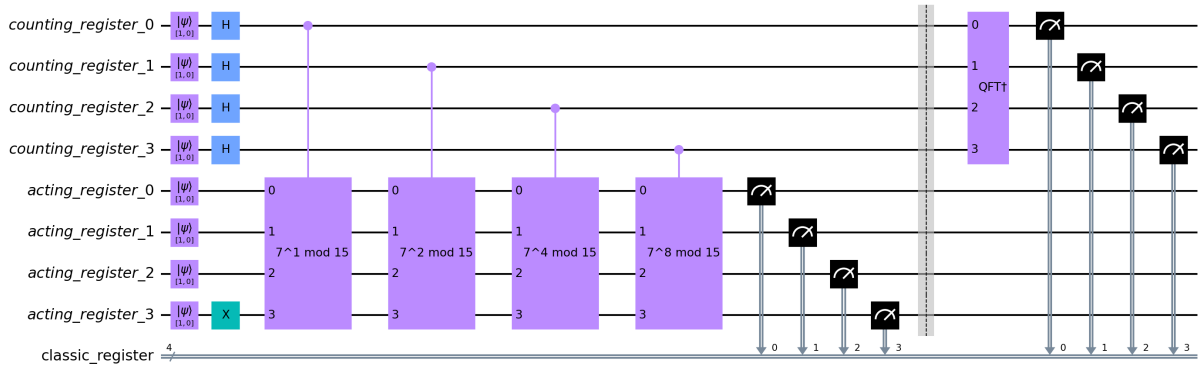


Figure 4.7: Taking measurements on the counting_register

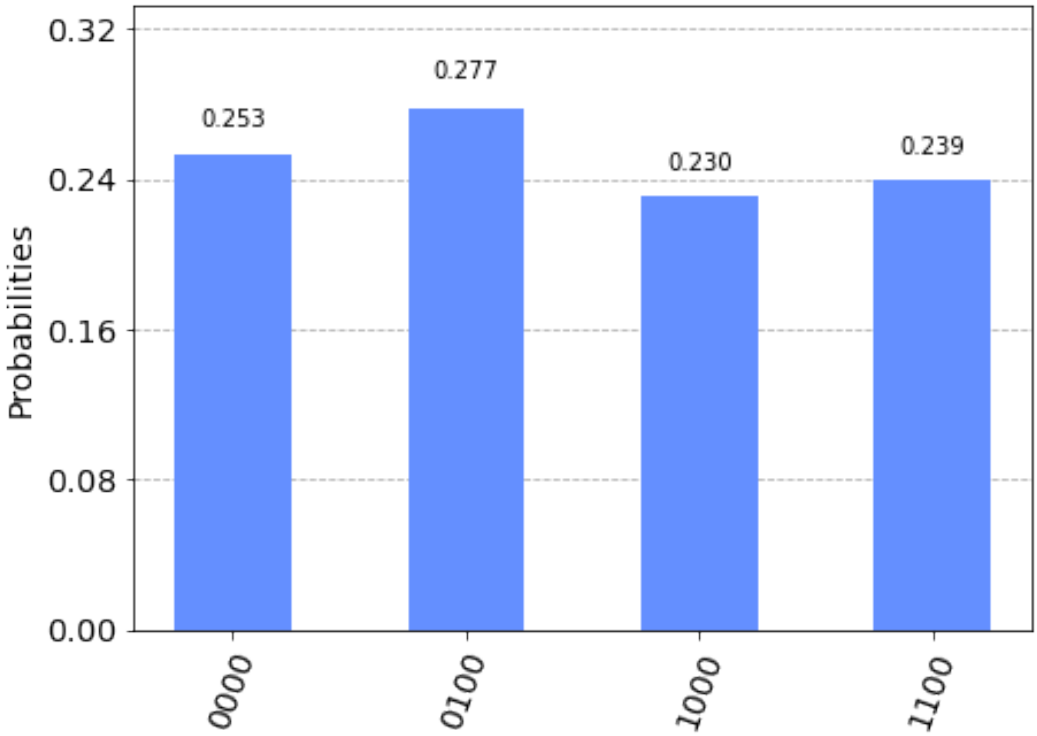


Figure 4.8: Measurement outcomes

4.3. Computer simulations

In the following computer simulations, we used Shor's algorithm with various modifications to factorise the numbers $N = 15$, 21 and 33. To illustrate the details of the steps taken in this computer simulations, we will consider the case of $N = 15 = 1111_2$. Hence the number of bits (n) representing N is 4. Later in section 4.4, we will present the results from the other simulations and analyse these results. As discussed in section 3.7, some variants of implementing the Shor's algorithm include the following combination.

1. Variant 1: Quantum modular exponentiation with classical adder + Standard QFT
2. Variant 2: Quantum modular exponentiation with classical adder + semiclassical QFT
3. Variant 3: Quantum modular exponentiation with QFT adder + Standard QFT
4. Variant 4: Quantum modular exponentiation with QFT adder + semiclassical QFT

Quantum modular exponentiation with QFT adder was explained in section 3.6 and the explanation of Quantum modular exponentiation with classical adder proposed by Vedral, Barenco and Ekert in 1996 and implemented in a reversible way can be found in [23]. The standard QFT was discussed in section 3.1.1 and the semiclassical QFT was discussed in 3.1.2. With respect to these variants, the simulation was done along the following line:

- Using Constant-Optimized Quantum Circuits for Modular multiplication which we referred to as shor_normal_constant in section 4.3.1
- Simulating variant 3 by using the modular exponentiation circuit with QFT adder which uses the standard QFT and which we referred to as shor_standard_QFT in section 4.3.2
- Simulating variant 4 by using the modular exponentiation circuit which uses Semiclassical QFT and which we referred to as shor_semiclassical_QFT in section 4.3.3

4.3.1. Computer Simulation: Constant-Optimized Circuits (shor_normal_constant)

To factor $N = 15$, we begin by randomly chose $a = 7$ and used 8 counting bits. To create the circuit for $U|y\rangle = |ay \bmod 15\rangle$, we used a simplified version of the modular exponentiation circuit based on the results from Igor et al. (2012) [45]. This simplified the modular multiplication circuit for different choices of $a \bmod N$ as shown in 4.9.

Furthermore, we notice that by a manual computation of these constant-optimized circuits for $f(x) = Cx \bmod 15$, the circuits for 2 and 13, 7 and 8, 4 and 11 all give similar circuit output with the difference being only the X gate applied to 7, 11, 13. Hence the following steps which we apply for the case of choosing $a = 7$ can be easily reproduced for other choices of a with very little modifications. To create the function U^x, we repeated it x times because

$$U^{2^j}|y\rangle = \left|a^{2^j} y \bmod N\right\rangle. \tag{4.5}$$

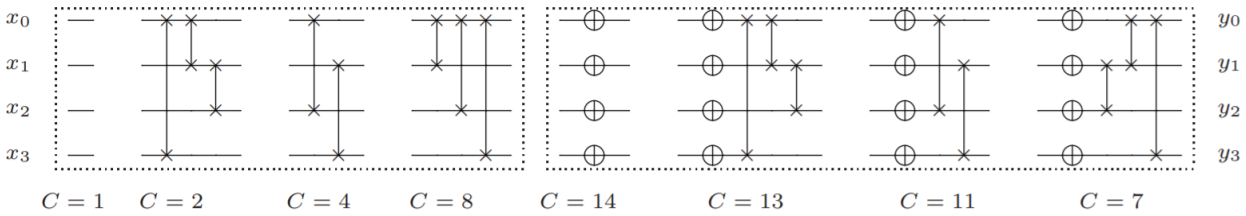


Figure 4.9: Circuits for $f(x) = Cx \bmod 15$, $gcd(C, 15) = 1$, (left); $C = 2k$, (right) $C = 15 - 2k$.

The detailed steps are below:

1. Initialise a quantum circuit with two registers. The first having 8 qubits and the second having 4 qubits.
2. We put the qubits in the first register to a maximal superposition state by applying the Hadamard gate.
3. Next we apply an X gate to the last qubit in the second register $|0\rangle$
4. In applying the controlled-U operation, we use the optimized version given in figure 4.9 which amounts to performing 3 swap operations and repeating x times to implement U^x.
5. $QFT^\dagger$ is applied to the 8 qubits in the first register and we measure these qubits to get the final circuit shown in figure 4.10. Measurement outcomes are shown in figure 4.12.

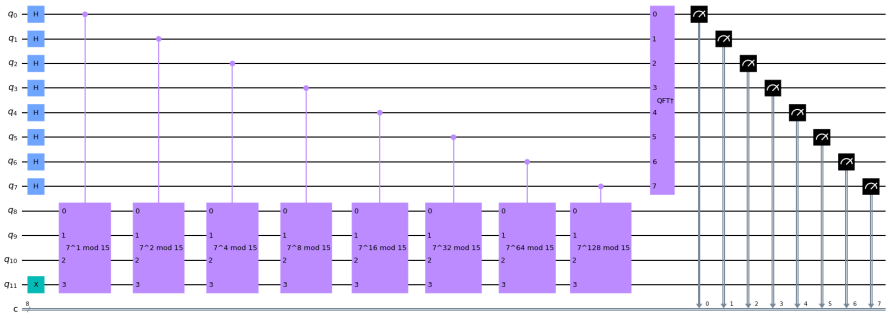


Figure 4.10: Constant-optimized Shor's circuit

Results from the measurements are post-processed classically by doing the following

- Using continued fractions algorithm, we get 0/1, 1/4, 1/2 and 3/4. Hence r = 2 or 4. This clearly agrees with the graph of $7^x \pmod{15}$ shown in figure 4.11 showing that the period is indeed $r = 4$
- Since r is even and $a^{r/2} + 1 = 7^2 + 1 = 5 \pmod{15}$ is a non-trivial factor of 15, we compute $gcd(5, 15) = 3$ and hence we get the factors of $15 = 3 \times 5$.

This entire circuit used 12 qubits and 8 classical bits.

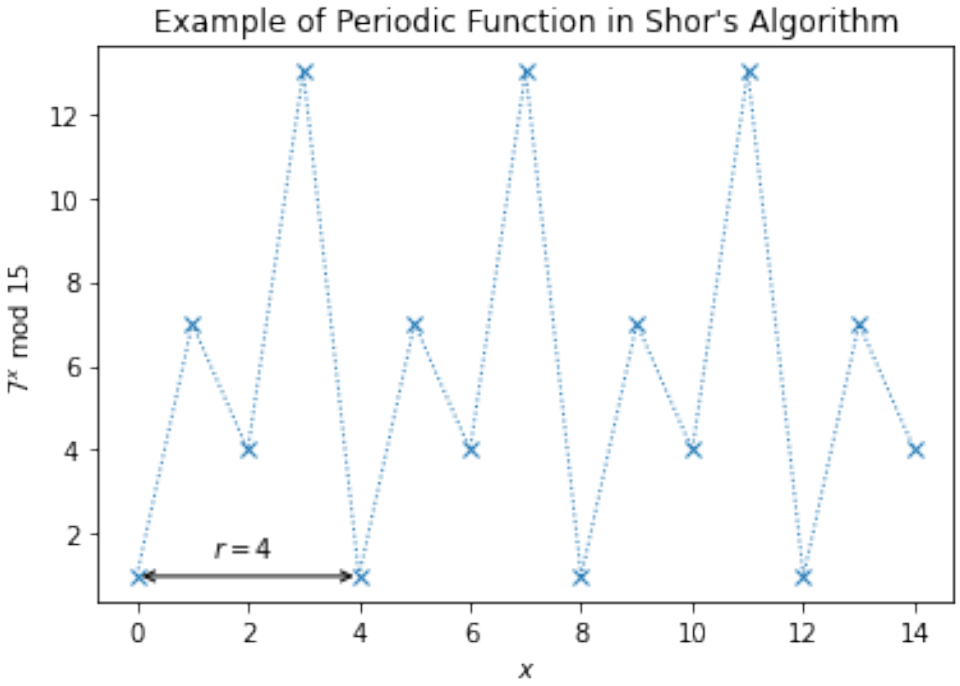
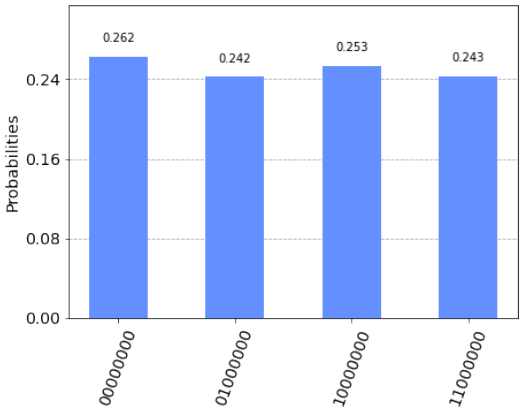


Figure 4.11: Graph of $7^x \pmod{15}$



	Register Output	Phase
0	01000000(bin) = 64(dec)	64/256 = 0.25
1	00000000(bin) = 0(dec)	0/256 = 0.00
2	10000000(bin) = 128(dec)	128/256 = 0.50
3	11000000(bin) = 192(dec)	192/256 = 0.75

Figure 4.12: Measurement results and Measured Phases

4.3.2. Simulation: Quantum modular exponentiation with QFT adder + standard QFT (shor_standard_QFT)

This variant implements the circuit for modular exponentiation using Quantum Fourier transform Adder as described in section 3.6. It uses the standard implementation of the QFT as discussed in section 3.1. This circuit requires $4n+2$ qubits to factor an n-bit integer N and this implementation variant will be referred to as the **shor_standard_QFT** variant. For consistency, We also chose $a = 7$ and created the following registers:

- An auxiliary quantum register "aux_reg" with $n + 2$ qubits. This was used for the addition and multiplication operations
- A quantum register "up_reg" with $2n$ qubits where the standard QFT was performed
- A quantum register "down_reg" with n qubits where the multiplications were made
- A classical register "up_classic" with $2n$ classical bits where the measured values of the qubits are stored.

Next, we initialized the "down_reg" to $|1\rangle$ by applying the X gate and created maximal superposition in "up_reg". Thus the total number of qubits used was $4n+2 = 4(4)+2 = 18$ qubits.

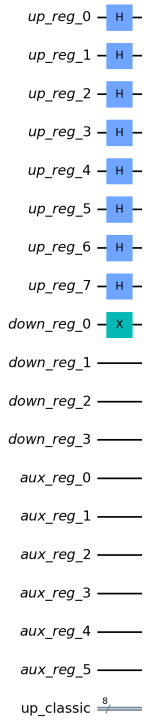


Figure 4.13: initialization of the circuit

We applied the multiplication gates as showed in the work by Stephane Beauregard [28] and discussed in section 3.6 in order to create the exponentiation. The actual modular exponentiation operation U^x was discussed in section 3.6 with circuit diagram in figure 3.11

showing its dependence on the $CMULT(A)$ mod N operation. To finish up, we applied $QFT^{\dagger}$ and took measurements. The final circuit is shown in figure 4.14 where the circuits for $\Phi ADD(a)$, $\Phi ADD(a)$ mod N, controlled-$SWAP$ and $CMULT(A)$ mod N are all embedded within the modular exponentiation U^{2^j} shown in the figure. Running this

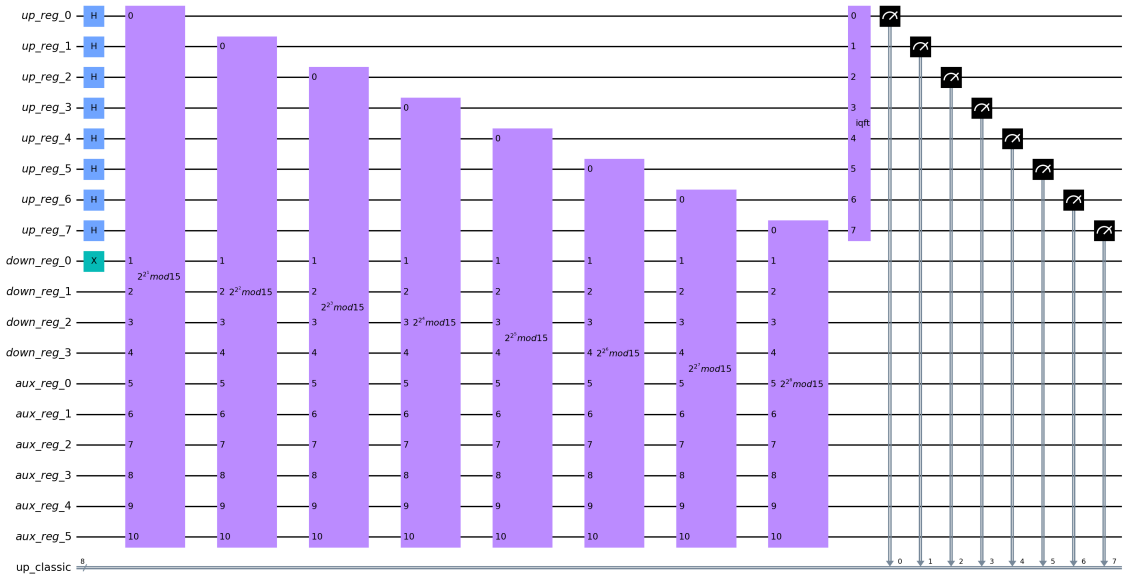


Figure 4.14: Final circuit of the shor_standard_QFT variant for factoring $N = 15$

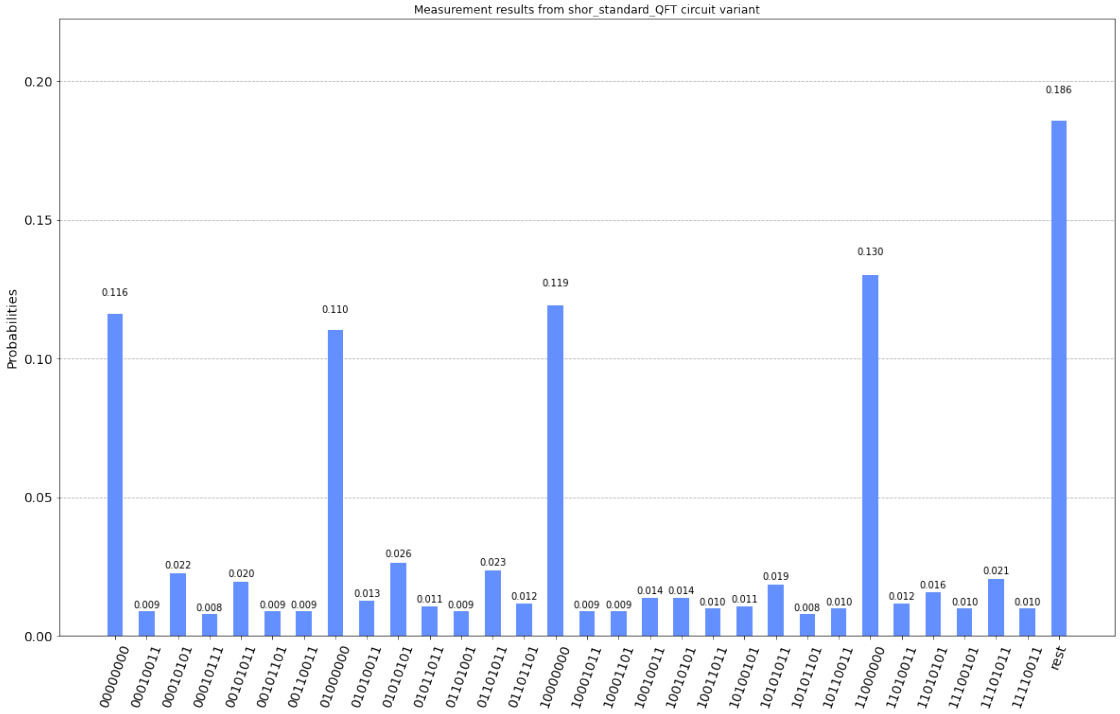


Figure 4.15: Measurement results of the shor_standard_QFT variant for factoring $N = 15$

circuit 1024 times on IBM QASM_simulator returned 84 different possible measurements

with varying frequency.

Figure 4.15 shows the top 30 outcomes with a probability of being measured. We note that this seemingly noisy data is due to the realistic noise model being implemented on the IBM QASM_simulator which simulates how a real quantum computer will perform. These "errors" in a quantum computer can come from many sources including dephasing and decoherence of the qubits, quantum state preparation, quantum gate operations or even measurements. However, four results (K) standout namely $|00000000\rangle$, $|01000000\rangle$, $|10000000\rangle$ and $|11000000\rangle$ each corresponding to 0, 64, 128 and 192 in integers. For a measurement outcome K, the phase $\phi = \frac{K}{2^n} = \frac{s}{r}$ where r is the period of $a = 7$ we are trying to get and can be gotten from the continued fraction algorithm. We notice that $\frac{K}{2^8}$ for the different values of K corresponds to 0, 0.25, 0.5 and 0.75. From the computations in section 4.2, we get the answer to the factorization of $15 = 3 \times 5$.

We also notice from the results that the probability of success $P(s) \geq 0.110 + 0.119 + 0.130 = 0.359$. By theorems 1, 2 and 3, we are quite certain that with a few more trials, the correct order will be deduced and the much lower probability of obtaining the right measurement outcome seen here is mainly due to noise.

4.3.3. Simulation: Quantum modular exponentiation with QFT adder + semiclassical QFT (shor_semiclassical_QFT)

This implementation is based on the Modular exponentiation with the QFT adder discussed in section 3.6 together with the QFT implemented semiclassically discussed in section 3.1.2. This variant needs $2n + 3$ qubits to factor an n-bit integer N and we will refer to this variant as shor_semiclassical_QFT. Similar to the shor_standard_QFT version, we chose similar parameters and performed similar operations. The major change was that we replaced the standard QFT implementation with semiclassical QFT. We describe the key implementation differences below:

- We initialize a quantum register with only 1 qubit where the semiclassical QFT was performed
- We initialized a classical register with 1 bit used to reset the state of the top single qubit to 0 if the previous measurement was 1
- We initialized the down register to 1
- For each round in $2n$ cycles, we applied the following

    1. An X gate to up_reg classically controlled by the classical auxilliary register
    2. An Hadamard gate to up_reg
    3. Modular exponentiation operation using $CMULT(A)$ mod N applied the necessary amount of times.
    4. At the i^{th} round of the $2n$ cycles, for another set of 2^i sub-rounds, we further apply the following operations
        (a) A phase rotation gate u_1 to up_reg classically controlled by the up classical register having a value equal to the index of this sub-round
        (b) An Hadamard gate to up_reg
        (c) Two measurements of the single qubit in the up register. The first measurement is stored in the i^{th} classical bit in the up classical register while the second measurement is stored in the single bit in the aux_classic register

- At the end of these cycles of operations, we take final measurement of up_reg. Since the circuit is too large to be shown in this report, the beginning part of the circuit is shown in figure 4.16 while the final part is shown in figure 4.17. The complete circuit is accessible at [46] by zooming accordingly [1].
- Next we compiled the circuit with an optimization level of 3 on the QASM_simulator and ran it with 1024 shots.

As expected, the results were very noisy due to the large number of QASM_instructions with each instruction introducing its own noise. From the 1024 simulations, there were

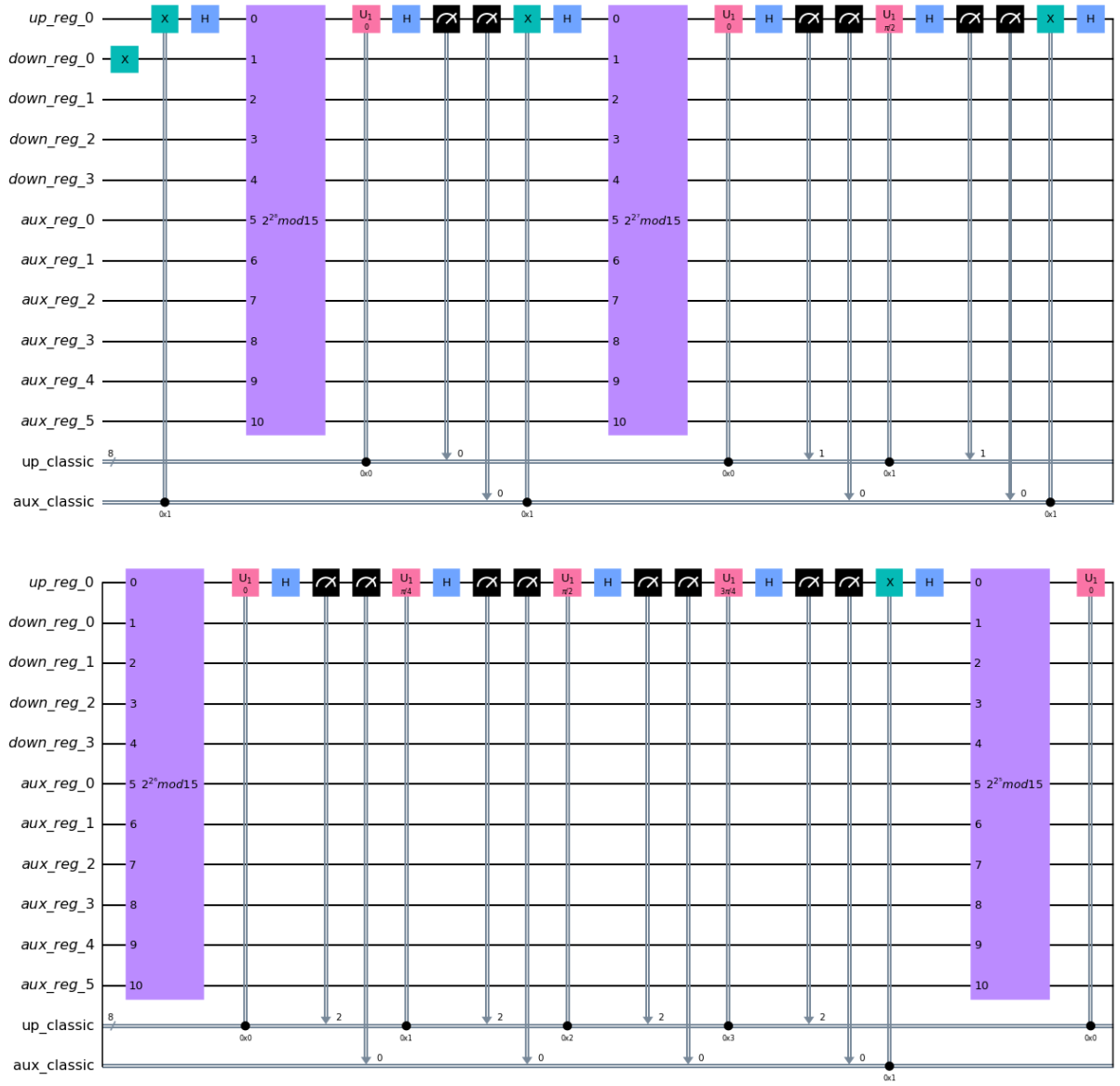


Figure 4.16: Beginning part of the Shor's factorization circuit for $N = 15$ with semiclassical QFT

128 different possible measurements with varying frequency.

Figure 4.18 shows the top 30 outcomes with a probability of being measure and it's difficult to really select with a high level of certainty what the likely measurement will be. To this end, we processed the result programatically as follows:

1. We converted the measurement outcomes from binary to base 10 then sorted them by their frequency

2. We divided each outcome by 2^8 to get the phase $\phi = \frac{s}{r}$

[1]The complete image of the final circuit could not be uploaded alongside the thesis because of the limit on the size of attachments. Hence it can be accessed at https://raw.githubusercontent.com/martynscn/Thesis/main/shor_semiclassical_final_circuit_long.png

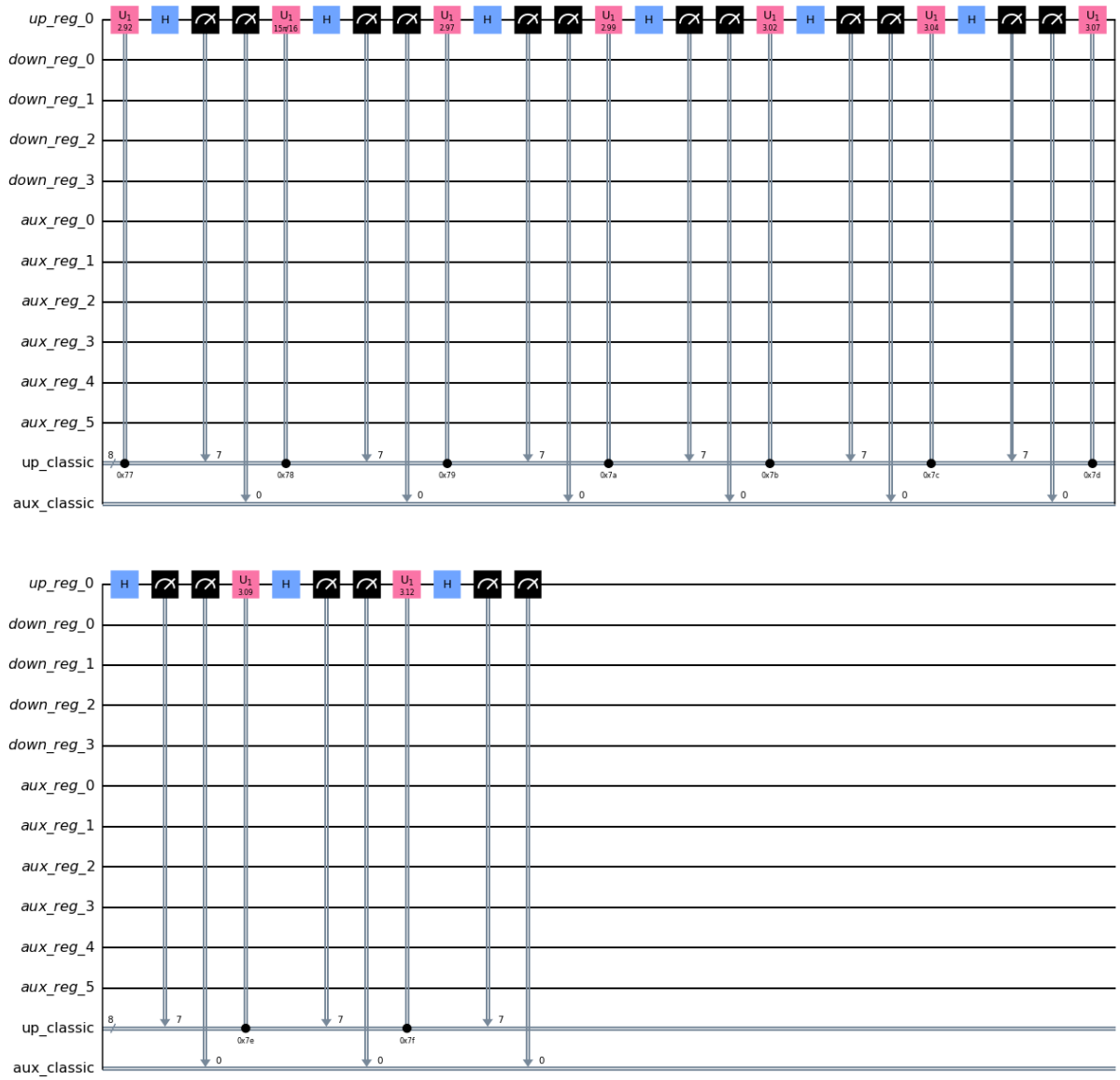


Figure 4.17: Final part of the Shor's factorization circuit for $N = 15$ with semiclassical QFT

3. We applied the continued fraction algorithm to the phase ϕ with a possible maximum denominator of 10^3 since we know the size of N and to save time and computing resources.

4. If the continued fraction terminates successfully and yield $\phi = \frac{s}{r}$, we compute $p = gcd(a^{\frac{n}{2}} + 1, N)$ and $q = gcd(a^{\frac{n}{2}} - 1, N)$.

5. We confirm that p and/or q are non-trivial factors of N and if true, then we record the factorization along with the probability of that measurement outcome to be used in determining the overall success rate.

After performing these steps, we found that in at least 22.36% of the time out of 1024 shots, our algorithm yielded the right answer. We say at least here because we discarded those measurement outcomes which in applying the continued fraction algorithm was giving

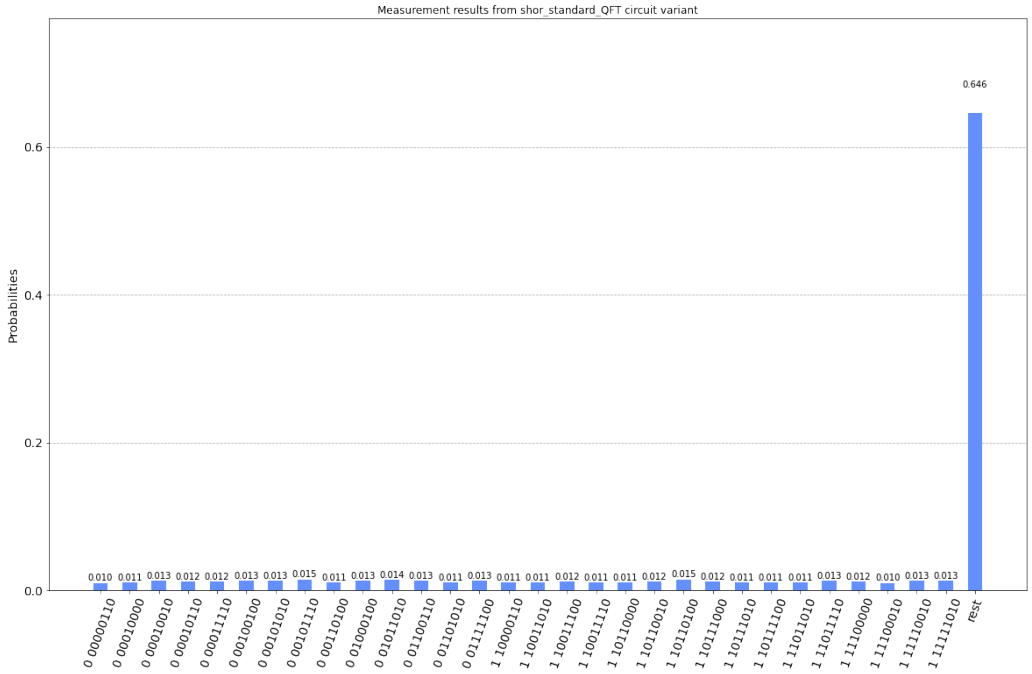


Figure 4.18: Measurement results of the shor_semiclassical_QFT variant for factoring $N = 15$

denominators greater than 1000. If this threshold was moved higher, the probability of obtaining the period r is certain to at least remain the same or better still increase.

4.4. Simulation Results

As we previously described in section 4.2, we simulated three variants of the Shor's algorithm namely shor_normal_constant, shor_standard_QFT and shor_semiclassical_QFT. However, the shor_normal_constant variant is not interesting to analyse and compare because it is not scalable. The constant optimized circuit we used is different for every N and already precomputed. Hence, we begin with analysing the execution time of the two implementation variants (4.3) in section 4.4.1. In section 4.4.2, we analyse the success rates between the two main variants shor_standard_QFT and shor_semiclassical_QFT and in section 4.4.3, we give a summary statistics of the complexity of their circuits.

Each simulation we performed was done with 1024 shots for better result outcomes. We did the simulations for the numbers 15, 21 and 33 while attempts were also made to also factor numbers 35, 39, 51, 55 and 57. However, the constraint on the amount of memory available to public users in IBM Quantum Experience platform was 8gb RAM and the capacity of this resource was easily reached by running up to 7 parallel operations each lasting more than an hour. We got partial results (for some a coprime to N) for the numbers 35, 39, 51, 55 and 57 and complete results for the numbers 15, 21 and 33 hence for a holistic analysis, we present the results for $N = 15$, 21 and 33. The full raw data (including partial data from 35, 39, 51, 55 and 57) is available at [56]. while the relevant complete raw and processed data used in this analysis (15, 21 and 33) is available at [57]

4.4.1. Execution time analysis

In figure 4.19, we show the relation between the number N factored and the execution time (in logarithm scale) for 1024 shots. This execution time takes into account the classical pre-processing, the quantum processing and the classical post-processing where we employ the continued fractions algorithm. We remind here that the three numbers considered 15, 21 and 33 are each 4, 5 and 6 bits numbers so their result could fairly approximate the result for other 4, 5 and 6 bits semi-prime numbers.

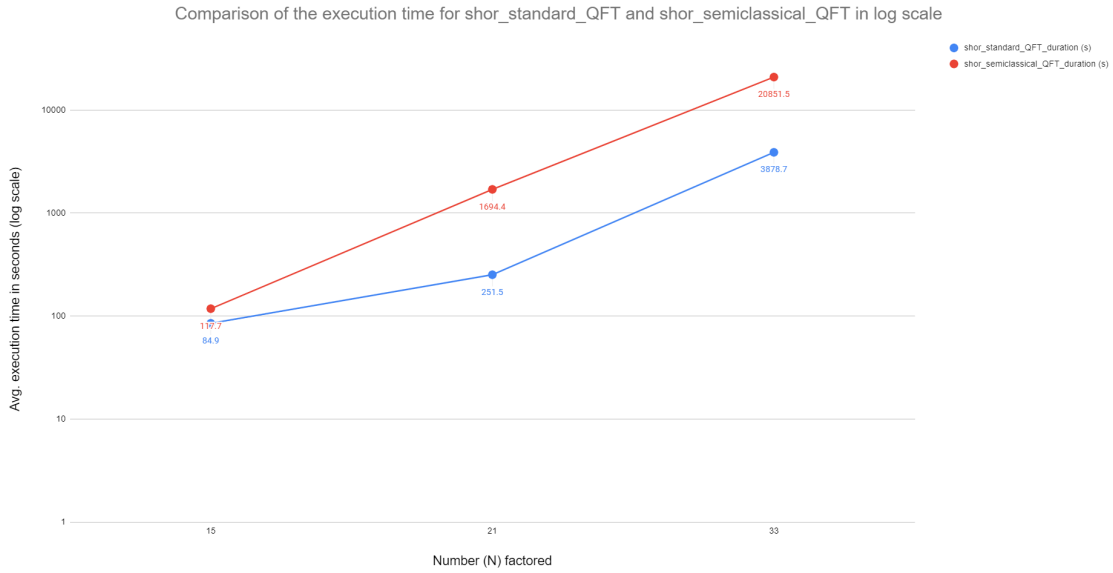


Figure 4.19: Comparison of the execution time for shor_standard_QFT and shor_semiclassical_QFT in log scale

We see that in the figure 4.19, both variants have exponential computational complexity. This agrees with the theory as the quantum circuits simulated on classical computers would have at least exponential computational complexity [47, 48]. It was also proven that quantum algorithms can only be efficiently simulated using quantum systems in [7]. We observe that in the case of shor_semiclassical_QFT, the results of the measurement appear to lie exactly on the trend line. Measured execution times for the shor_standard_QFT variant slightly differed from the trend line. It was rather surprising that the execution time for the shor_semiclassical_QFT variant was overall higher than the shor_standard_QFT variant since it uses less qubits. However this could be explained given the huge number of QASM instructions in the shor_semiclassical_QFT variant when compared to shor_standard_QFT. We present this level of information in section 4.4.3.

Normally, a is chosen at random at the start of the Shor's algorithm and it is important to investigate what impact the chosen number has on the execution time. Figures 4.20 and 4.21 shows the execution times (for 1024 shots) for every number a coprime to $N = 33$ respectively for the shor_standard_QFT and shor_semiclassical_QFT variants. These times were taken into account in the analysis regardless of the successful or unsuccessful outcome of the iteration. We noticed that there was no impact of the success or failure on

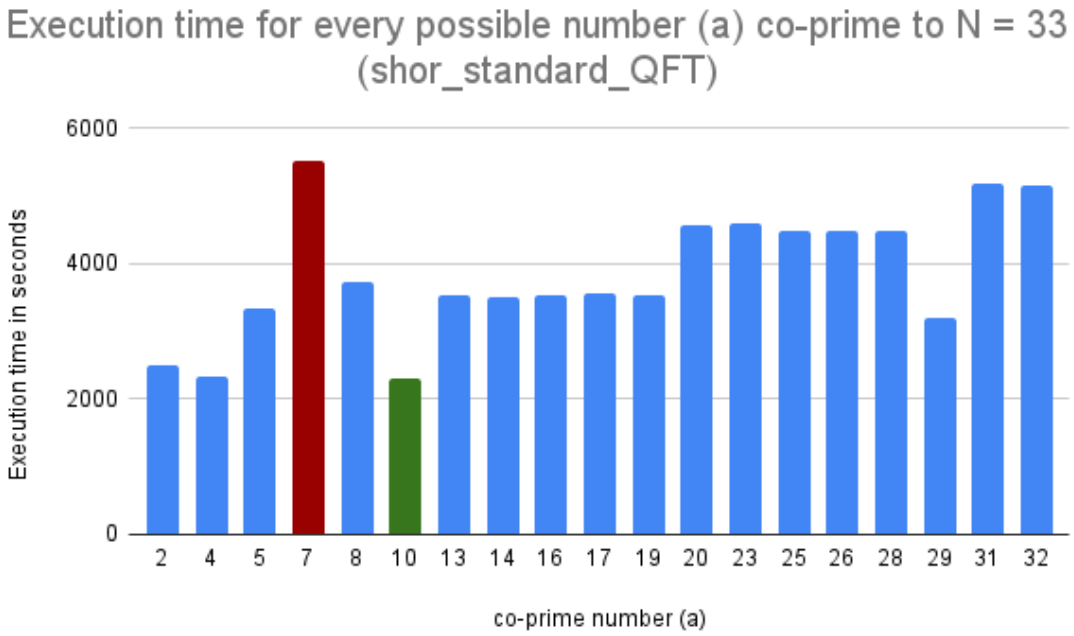


Figure 4.20: Execution time for every possible number (a) co-prime to $N = 33$ (shor_standard_QFT)

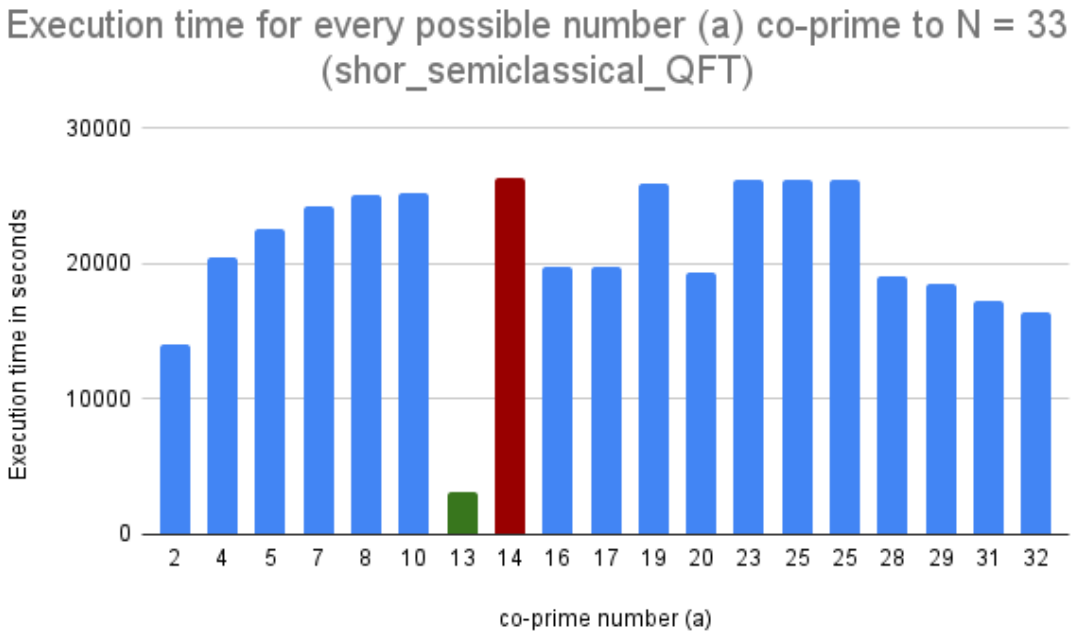


Figure 4.21: Execution time for every possible number (a) co-prime to $N = 33$ (shor_semiclassical_QFT)

the execution time of the algorithm as for example, in the case of shor_standard_QFT, the execution time for $a = 7$ and $a = 10$ were both the highest and lowest respectively and both a values were successful with varying degree of probability. Similar statement holds in the shor_semiclassical_QFT variant for $a = 14$ and $a = 13$. It is quite remarkable

to note the very low execution time for $a = 13$ in the shor_semiclassical_QFT variant. More experiments (each consisting of 1024 shots) for this particular case ($a = 13$) showed the execution times all ranged between 52 minutes 25 seconds and 1 hour 8 minutes 35 seconds.

In summary, the shor_standard_QFT seems to perform a little better than the shor_semiclassical_QFT variant. However, this could be because of the longer number of QASM instructions and classical postprocessing using continued fraction algorithm. This reason could be valid because the different possible measurement outcomes from shor_standard_QFT was fewer (84) than that of shor_semiclassical_QFT (128) variant hence more classical postprocessing was needed in the case of shor_semiclassical_QFT. Nevertheless, it is important for the performance of the algorithm to choose at random the number a co-prime to N before every iteration of the algorithm. This tries to ensure almost equal probability to choose better or worse starting value.

4.4.2. Success rate analysis

In this section, we analyse the success rate of both simulated variants which we define as the number of executions (out of 1024 shots) that our algorithm yields the correct period. As discussed in [42], the algorithm returns the correct period r only with some probability.

The standard approach that a candidate (measurement outcome) for an order yield by the classical post-processing phase is to test whether it is the correct order or not. If this fails, then the quantum order finding algorithm has to be executed again. In table 4.22 and 4.23, we show the success rates for the numbers $N = 15$, 21 and 35 considered for the shor_standard_QFT and shor_semiclassical_QFT variant consecutively. For each number N and each a chosen, it shows the average success rates.

It is interesting to see the combinations of N and a that gives us over 50% success rates in the shor_standard_QFT variant. These are $a = 4$ and 11 for $N = 15$, $a = 8$ and 13 for $N = 21$, $a = 10$ and 23 for $N = 33$. However, in the shor_standard_QFT variant, we see that only the combination of $N = 15$ and $a = 4$ gives a success rate of over 50%. The theoretical success rate has been shown to depend on the order r, which in turn depends on the number a co-prime to N [30] and on average ranges between 20% to 40%.

In addition as shown in figure 4.24, it is very interesting to see the high correlation between the success rate in both variants. However, this could be very much expected because the underlying theory regarding the period r for each number N and a combination doesn't change with a change of algorithm. Hence we see similar patterns because we are performing the same computation (of order finding) but implemented in two different ways. Though the overall success rate of shor_standard_QFT variant is higher than the shor_semiclassical_QFT variant, we observe little significant impact on the success rate of the computation if either of the two variants is chosen. Also, the overall lower success rate in the shor_semiclassical_QFT variant may be explained as the result of many QASM instructions and measurements which with high probability introduces more gate and measurement errors even though it uses less qubits.

Number(N)	a	result_factors	prob_success	prob_failure	total_counts	result_successful_counts	result_failure_counts
15	2	[[3, 5]]	36.91	63.09	82	24	58
15	4	[[3, 5]]	50.78	49.22	75	41	34
15	7	[[3, 5]]	34.67	65.33	86	15	71
15	8	[[3, 5]]	39.94	60.06	80	15	65
15	11	[[3, 5]]	51.56	48.44	71	37	34
15	13	[[3, 5]]	18.95	81.05	93	15	78
15	14	[]	0.00	100.00	73	0	73
21	2	[[3, 7]]	26.66	73.34	283	52	231
21	4	[]	0.00	100.00	250	0	250
21	5	[]	0.00	100.00	282	0	282
21	8	[[3, 7]]	54.98	45.02	153	95	58
21	10	[[3, 7]]	25.00	75.00	286	42	244
21	11	[[3, 7]]	21.68	78.32	278	34	244
21	13	[[3, 7]]	52.64	47.36	145	82	63
21	16	[]	0.00	100.00	262	0	262
21	17	[]	0.00	100.00	192	0	192
21	19	[[3, 7]]	23.14	76.86	272	37	235
21	20	[]	0.00	100.00	146	0	146
33	2	[]	0.00	100.00	817	0	817
33	4	[]	0.00	100.00	805	0	805
33	5	[[3, 11]]	13.09	86.91	809	82	727
33	7	[[3, 11]]	12.01	87.99	808	80	728
33	8	[]	0.00	100.00	797	0	797
33	10	[[3, 11]]	51.86	48.14	273	165	108
33	13	[[3, 11]]	13.38	86.62	842	90	752
33	14	[[3, 11]]	14.36	85.64	795	86	709
33	16	[]	0.00	100.00	815	0	815
33	17	[]	0.00	100.00	817	0	817
33	19	[]	0.00	100.00	801	0	801
33	20	[]	0.00	100.00	813	0	813
33	23	[[3, 11]]	52.93	47.07	255	135	120
33	25	[]	0.00	100.00	818	0	818
33	26	[]	0.00	100.00	816	0	816
33	28	[]	0.00	100.00	833	0	833
33	29	[]	0.00	100.00	808	0	808
33	31	[]	0.00	100.00	806	0	806
33	32	[]	0.00	100.00	249	0	249

Figure 4.22: Success rate for numbers $N = 15$, 21 and 35 considered in the shor_standard_QFT variant

Number(N)	a	result_factors	prob_success	prob_failure	total_counts	result_successful_counts	result_failure_counts
15	2	[[3, 5]]	21.09	78.91	128	30	98
15	4	[[3, 5]]	52.25	47.75	128	67	61
15	7	[[3, 5]]	22.66	77.34	128	28	100
15	8	[[3, 5]]	24.90	75.10	128	28	100
15	11	[[3, 5]]	47.56	52.44	128	61	67
15	13	[[3, 5]]	16.80	83.20	127	22	105
15	14	[]	0.00	100.00	128	0	128
21	2	[[3, 7]]	12.11	87.89	635	74	561
21	4	[]	0.00	100.00	633	0	633
21	5	[]	0.00	100.00	662	0	662
21	8	[[3, 7]]	46.39	53.61	442	201	241
21	10	[[3, 7]]	7.42	92.58	650	53	597
21	11	[[3, 7]]	7.42	92.58	650	54	596
21	13	[[3, 7]]	48.05	51.95	447	207	240
21	16	[]	0.00	100.00	652	0	652
21	17	[]	0.00	100.00	674	0	674
21	19	[[3, 7]]	7.03	92.97	647	45	602
21	20	[]	0.00	100.00	446	0	446
33	2	[]	0.00	100.00	904	0	904
33	4	[]	0.00	100.00	908	0	908
33	5	[[3, 11]]	5.86	94.14	902	53	849
33	7	[[3, 11]]	6.64	93.36	912	61	851
33	8	[]	0.00	100.00	921	0	921
33	10	[[3, 11]]	45.21	54.79	808	364	444
33	13	[[3, 11]]	9.38	90.63	127	12	115
33	14	[[3, 11]]	5.37	94.63	925	49	876
33	16	[]	0.00	100.00	907	0	907
33	17	[]	0.00	100.00	898	0	898
33	19	[]	0.00	100.00	914	0	914
33	20	[]	0.00	100.00	900	0	900
33	23	[[3, 11]]	40.33	59.67	812	329	483
33	25	[]	0.00	100.00	912	0	912
33	25	[]	0.00	100.00	912	0	912
33	28	[]	0.00	100.00	903	0	903
33	29	[]	0.00	100.00	903	0	903
33	31	[]	0.00	100.00	916	0	916
33	32	[]	0.00	100.00	803	0	803

Figure 4.23: Success rate for numbers $N = 15$, 21 and 35 considered in the shor_semi-classical_QFT variant

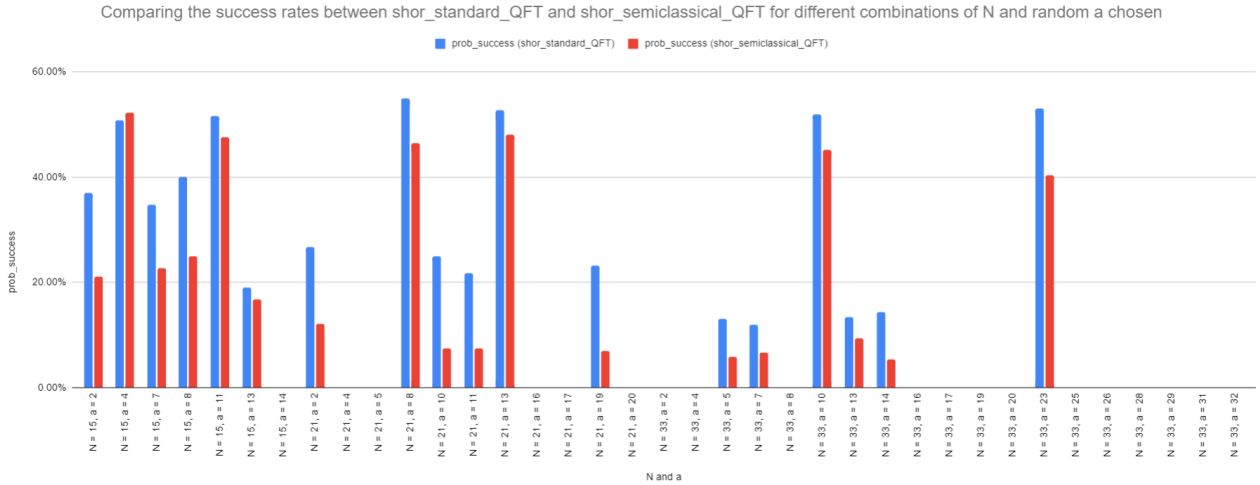


Figure 4.24: Comparing the success rates between shor_standard_QFT and shor_semi-classical_QFT for different combinations of N and random a chosen

4.4.3. Circuit metrics summary

In this section, we briefly present a summary of some metrics of the circuits in the two variants. We comment that after the quantum circuit is built, it is decomposed which is the expansion of a gate in a circuit using its decomposition rules. Next, the decomposed circuit is transpiled which is the process of rewriting the input decomposed quantum circuit to match the topology of the specific quantum device. Transpilation could also be used as a way of optimizing the quantum circuit for execution on noisy quantum systems. The process entails the following [49]:

1. Virtual circuit optimization
2. Qubit gate decomposition for all 3+ multi qubit gates
3. Placement on physical qubits
4. Routing on restricted topology
5. Translation to basis gates
6. Physical circuit optimization

Number(N)	a	num_bits	num_qubits	num_clbits	width	depth	size	num_nonlocal_gates	num_ancillas	num_clbits	num_qubits	num_of_count_ops	num_of_measure	num_of_h	num_of_cswap	num_of_swap
15	2	4	18	8	26	144	185	160	0	8	18	73	8	8	32	4
15	4	4	18	8	26	144	185	160	0	8	18	73	8	8	32	4
15	7	4	18	8	26	144	185	160	0	8	18	73	8	8	32	4
15	8	4	18	8	26	144	185	160	0	8	18	73	8	8	32	4
15	11	4	18	8	26	144	185	160	0	8	18	73	8	8	32	4
15	13	4	18	8	26	144	185	160	0	8	18	73	8	8	32	4
15	14	4	18	8	26	144	185	160	0	8	18	73	8	8	32	4
21	2	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	4	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	5	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	8	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	10	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	11	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	13	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	16	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	17	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	19	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
21	20	5	22	10	32	210	271	240	0	10	22	109	10	10	50	5
33	2	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	4	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	5	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	7	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	8	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	10	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	13	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	14	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	16	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	17	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	19	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	20	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	23	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	25	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	26	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	28	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	29	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	31	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6
33	32	6	26	12	38	288	373	336	0	12	26	153	12	12	72	6

Figure 4.25: Summary metrics for the shor_standard_QFT decomposed circuit

After the transpilation process, some metrics of the circuit could be higher or lower depending on the initial design of the circuit implementing the algorithm. In the following figures, we summarize the metrics for the decomposed and transpiled versions of the circuits implementing both variants of the algorithm. Some abbreviations have been made and they are explained below [50]:

1. Number of binary digits (num_bits): Number of bits of the number N
2. Number of qubits (num_qubits)

Number(N)	a	num_bits	num_qubits	num_clbits	width	depth	size	num_nonlocal_gates	num_ancillas	num_clbits	num_qubits	num_of_count_ops	num_of_x	num_of_measure	num_of_h
15	2	4	18	8	26	6391	10223	8096	0	8	18	10	129	8	1218
15	4	4	18	8	26	6393	10227	8096	0	8	18	10	129	8	1218
15	7	4	18	8	26	6456	10293	8156	0	8	18	10	129	8	1218
15	8	4	18	8	26	6391	10223	8096	0	8	18	10	129	8	1218
15	11	4	18	8	26	6455	10291	8156	0	8	18	10	129	8	1218
15	13	4	18	8	26	6456	10293	8156	0	8	18	10	129	8	1218
15	14	4	18	8	26	6455	10291	8156	0	8	18	10	129	8	1218
21	2	5	22	10	32	11862	20545	16560	0	10	22	11	201	10	2232
21	4	5	22	10	32	11862	20545	16560	0	10	22	11	201	10	2232
21	5	5	22	10	32	11925	20611	16620	0	10	22	11	201	10	2232
21	8	5	22	10	32	11647	20331	16344	0	10	22	11	201	10	2232
21	10	5	22	10	32	11938	20625	16632	0	10	22	11	201	10	2232
21	11	5	22	10	32	11861	20543	16560	0	10	22	11	201	10	2232
21	13	5	22	10	32	11707	20391	16404	0	10	22	11	201	10	2232
21	16	5	22	10	32	11862	20545	16560	0	10	22	11	201	10	2232
21	17	5	22	10	32	11925	20611	16620	0	10	22	11	201	10	2232
21	19	5	22	10	32	11937	20623	16632	0	10	22	11	201	10	2232
21	20	5	22	10	32	11721	20407	16416	0	10	22	11	201	10	2232
33	2	6	26	12	38	20171	36437	29790	0	12	26	11	289	12	3782
33	4	6	26	12	38	20267	36539	29880	0	12	26	11	289	12	3782
33	5	6	26	12	38	20256	36523	29874	0	12	26	11	289	12	3782
33	7	6	26	12	38	20341	36615	29952	0	12	26	11	289	12	3782
33	8	6	26	12	38	20298	36571	29910	0	12	26	11	289	12	3782
33	10	6	26	12	38	19273	35415	28980	0	12	26	10	289	12	3866
33	13	6	26	12	38	20282	36551	29898	0	12	26	11	289	12	3782
33	14	6	26	12	38	20362	36549	30024	0	12	26	10	289	12	3866
33	16	6	26	12	38	20290	36477	29952	0	12	26	10	289	12	3866
33	17	6	26	12	38	20170	36435	29790	0	12	26	11	289	12	3782
33	19	6	26	12	38	20340	36613	29952	0	12	26	11	289	12	3782
33	20	6	26	12	38	20281	36465	29946	0	12	26	10	289	12	3866
33	23	6	26	12	38	19303	35535	28956	0	12	26	11	289	12	3782
33	25	6	26	12	38	20290	36477	29952	0	12	26	10	289	12	3866
33	26	6	26	12	38	20362	36549	30024	0	12	26	10	289	12	3866
33	28	6	26	12	38	20306	36491	29970	0	12	26	10	289	12	3866
33	29	6	26	12	38	20322	36511	29982	0	12	26	10	289	12	3866
33	31	6	26	12	38	20266	36537	29880	0	12	26	11	289	12	3782
33	32	6	26	12	38	19291	35439	28992	0	12	26	10	289	12	3866

Figure 4.26: Summary metrics for the shor_standard_QFT transpiled circuit

Number(N)	a	num_bits	num_qubits	num_clbits	width	depth	size	num_nonlocal_gates	num_clbits	num_qubits	num_of_count_ops	num_of_measure	num_of_cx
15	2	4	11	9	20	18768	35469	14304	9	11	6	510	14272
15	4	4	11	9	20	18768	35469	14304	9	11	6	510	14272
15	7	4	11	9	20	18768	35469	14304	9	11	6	510	14272
15	8	4	11	9	20	18768	35469	14304	9	11	6	510	14272
15	11	4	11	9	20	18768	35469	14304	9	11	6	510	14272
15	13	4	11	9	20	18768	35469	14304	9	11	6	510	14272
15	14	4	11	9	20	18768	35469	14304	9	11	6	510	14272
21	2	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	4	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	5	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	8	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	10	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	11	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	13	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	16	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	17	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	19	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
21	20	5	13	11	24	38448	74503	29150	11	13	6	2046	29100
33	2	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	4	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	5	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	7	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	8	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	10	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	13	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	14	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	16	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	17	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	19	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	20	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	23	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	25	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	25	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	28	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	29	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	31	6	15	13	28	76596	144541	52920	13	15	6	8190	52848
33	32	6	15	13	28	76596	144541	52920	13	15	6	8190	52848

Figure 4.27: Summary metrics for the shor_semiclassical_QFT decomposed circuit

3. Number of classical bits (num_clbits)

4. Number of qubits plus classical bits in the circuit (width)

Number(N)	a	num_bits	num_qubits	num_clbits	width	depth	size	num_nonlocal_gates	num_clbits	num_qubits	num_of_count_ops	num_of_x	num_of_measure	num_of_h	num_of_cx	num_of_t
15	2	4	11	9	20	6854	10699	7808	9	11	9	9	510	1483	1728	64
15	4	4	11	9	20	6853	10697	7808	9	11	9	9	510	1481	1728	64
15	7	4	11	9	20	6916	10759	7868	9	11	9	9	510	1483	1788	64
15	8	4	11	9	20	6851	10699	7808	9	11	9	9	510	1483	1728	64
15	11	4	11	9	20	6916	10761	7868	9	11	9	9	510	1485	1788	64
15	13	4	11	9	20	6914	10759	7868	9	11	9	9	510	1483	1788	64
15	14	4	11	9	20	6917	10761	7868	9	11	9	9	510	1485	1788	64
21	2	5	13	11	24	15155	23377	15840	11	13	9	11	2046	3365	3240	100
21	4	5	13	11	24	15152	23377	15840	11	13	9	11	2046	3365	3240	100
21	5	5	13	11	24	15216	23437	15900	11	13	9	11	2046	3365	3300	100
21	8	5	13	11	24	14915	23141	15624	11	13	9	11	2046	3345	3024	100
21	10	5	13	11	24	15225	23449	15912	11	13	9	11	2046	3365	3312	100
21	11	5	13	11	24	15147	23377	15840	11	13	9	11	2046	3365	3240	100
21	13	5	13	11	24	14976	23205	15684	11	13	9	11	2046	3349	3084	100
21	16	5	13	11	24	15152	23377	15840	11	13	9	11	2046	3365	3240	100
21	17	5	13	11	24	15210	23437	15900	11	13	9	11	2046	3365	3300	100
21	19	5	13	11	24	15226	23449	15912	11	13	9	11	2046	3365	3312	100
21	20	5	13	11	24	14988	23217	15696	11	13	9	11	2046	3349	3096	100
33	2	6	15	13	28	35737	50987	28926	13	15	9	13	8190	8045	5742	144
33	4	6	15	13	28	35826	51077	29016	13	15	9	13	8190	8045	5832	144
33	5	6	15	13	28	35813	51071	29010	13	15	9	13	8190	8045	5826	144
33	7	6	15	13	28	35894	51147	29088	13	15	9	13	8190	8043	5904	144
33	8	6	15	13	28	35863	51107	29046	13	15	9	13	8190	8045	5862	144
33	10	6	15	13	28	34885	50085	27972	13	15	10	13	8190	7953	4788	72
33	13	6	15	13	28	35842	51095	29034	13	15	9	13	8190	8045	5850	144
33	14	6	15	13	28	35901	51151	29088	13	15	9	13	8190	8047	5904	144
33	16	6	15	13	28	35826	51077	29016	13	15	9	13	8190	8045	5832	144
33	17	6	15	13	28	35729	50987	28926	13	15	9	13	8190	8045	5742	144
33	19	6	15	13	28	35900	51147	29088	13	15	9	13	8190	8043	5904	144
33	20	6	15	13	28	35818	51071	29010	13	15	9	13	8190	8045	5826	144
33	23	6	15	13	28	34934	50129	28020	13	15	10	13	8190	7949	4836	72
33	25	6	15	13	28	35826	51077	29016	13	15	9	13	8190	8045	5832	144
33	25	6	15	13	28	35826	51077	29016	13	15	9	13	8190	8045	5832	144
33	28	6	15	13	28	35839	51095	29034	13	15	9	13	8190	8045	5850	144
33	29	6	15	13	28	35840	51107	28974	13	15	10	13	8190	7973	5790	72
33	31	6	15	13	28	35826	51077	29016	13	15	9	13	8190	8045	5832	144
33	32	6	15	13	28	34917	50097	28056	13	15	9	13	8190	8025	4872	144

Figure 4.28: Summary metrics for the shor_semiclassical_QFT transpiled circuit

5. Depth of the circuit [i.e. the length of the critical path in the circuit] (depth)
6. Total number of gate operations in the circuit (size)
7. Number of non-local gates in the circuit [for e.g. CNOTs] (num_nonlocal_gates)
8. Total number of operations including its multiplicities [this does not include measurement operations] (num_of_count_ops)
9. Number of measurement operations (num_of_measure)
10. Number of C-NOT operations (num_of_cx)
11. Number of Hadamard operations (num_of_h)
12. Number of T-gate operations [or T-counts] num_of_t

Figures 4.25, 4.26, 4.27 and 4.28 shows the summary metrics of the decomposed and transpiled circuits from the implementation of the the shor_standard_QFT and shor_semiclassical_QFT variant respectively. We notice how the transpiled circuit implementing the shor_semiclassical_QFT variant is higher than the shor_standard_QFT in many metrics except in the number of qubits and circuit width (which depends on the number of qubits). Hence this could further explain the reason why the measurement outcomes in the shor_semiclassical_QFT variant was a lot more noisier than the shor_standard_QFT variant.

5. Conclusion and potential future work

In this chapter, we summarize the thesis. In section 5.1, we discuss the achievement of the goals of the thesis. Section 5.2 presents the key conclusions from the simulations of the Shor's algorithm and in section 5.3, we present areas of possible further work

5.1. Goals Achievement Discussion

The goals of the thesis has been described in section 1.4. In this section we discuss how the goals have been achieved in the thesis.

5.1.1. Review of Shor's Algorithm

In chapter 3, we have described the Shor's algorithm details. We explained each part of the algorithm including important concepts such as Quantum Fourier Transforms, Quantum Phase Estimations and Modular Exponentiation. We also introduced basic subroutines needed to perform the Shor's order finding algorithm. In addition, we discussed the standard implementation of the Shor's algorithm in section 3.3 and a variant using semi-classical Quantum Fourier Transforms in section 3.1.2. We provided circuits leading up to the Quantum Modular Exponentiation, Quantum Phase Estimation and the Quantum Fourier Transforms. In section 3.1.2, presented an optimization variant of the Quantum Fourier Transforms and we compared the implementation of these variants in terms of the number of qubits needed as well as number of gates and circuit depth. In section 3.8, we discussed the complexity analysis of the two implementation variants and in section 2.2, we showed how an efficient integer factoring algorithm such as Shor's factoring algorithm can be used in breaking the RSA cryptosystem.

5.1.2. Simulation of Shor's Algorithm

We have implemented the quantum circuits for Shor's algorithm which was described in chapter 4 using the QASM simulator in IBM Quantum Experience Platform. We have tested the implementations and observe that it gives correct results according to theoretical assumptions and realistic implementation of the current quantum systems available. We have prepared and executed simulation cases in order to compare the implementation variants.

5.1.3. Analysis of simulation results

In section 4.3, we presented the outcomes from our simulations. We analyzed a performance metric given by the execution time. We have also compared the success rate of order finding achieved by the different algorithm variants and summarized their circuit complexity.

5.2. Simulation Results Summary

We have simulated two implementation variants of the Shor's Factoring algorithm using the QASM simulator offered on IBM Quantum Experience Platform. We compared the results in terms of computational complexity, the algorithm's success rate of finding the order and circuit metrics.
Both implementation variants had exponential computational complexity. This is correct according to the theoretical assumptions regarding the simulation of quantum computation on classical computers [47]. However, one of the variants (shor_standard_QFT) performs a little better than the other (shor_semiclassical_QFT). This could be because of the much higher QASM instructions needed in the shor_semiclassical_QFT variant.

The success rates of both implementations correlated well for different combination of values of N and a. However, one of the variants (shor_standard_QFT) had a slightly higher success rates overall.

5.3. Further Work

In this thesis, we have simulated and provided the results of the simulation of two variants of the Shor's Algorithm. There are other possibilities that could be done. While we used solely the Quantum Adder, the Classical Adder can be used. Quantum Modular Exponentiation with Classical Adder can be combined with both the standard QFT and the semiclassical QFT.

In Shor's algorithm, the number a with $gcd(a, N) = 1$ is chosen at random at the beginning of the algorithm. It could be useful to see the analysis from a number theoretical point of view of how these values (or even class of values) of a can effect the result in an overall success rate of finding the order of r.

In this thesis, our implementation variant using semiclassical QFT optimized the required number of qubits from $2n$ to just 1 where n denotes the number of bits of the semi-prime N to be factored. There are also variants of Shor's algorithm that optimises the number of operations or include the possibility of parallel computations. It will be very useful to see how these possibility of parallelizing the computation could be combined with the semiclassical implementation of the QFT to build an order finding circuit that both reduces the number of qubits required and the number of operations.

Bibliography

[1] Feynman, Richard; Leighton, Robert; Sands, Matthew: In *The Feynman Lectures on Physics, Vol. 3*, California Institute of Technology, p. 1.1 (1964) ISBN 978-0201500646

[2] Benioff, Paul: The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. In *Journal of Statistical Physics. 22 (5)*, 563–591 1980

[3] David Deutsch: Quantum computation. In *A comprehensive and inspiring guide to quantum computing*, Physics World, 1992

[4] Cho, Adrian: Breakthrough of the Year: The First Quantum Machine. In *Science Magazine*, 330 (6011): 1604 (2010-12-17)

[5] Tom Thompson: When silicon hits its limits. In *Byte*, (1996)

[6] Manin, Yu. I.: *Computable and Noncomputable*, Sov.Radio. pp. 13–15 1980

[7] Feynman, Richard: Simulating Physics with Computers. In *International Journal of Theoretical Physics. 21 (6/7)*, 467–488, June 1982 doi: 10.1007/BF02650179. URL http://dx.doi.org/10.1007/BF02650179.

[8] S. Wiesner: Conjugate coding. In *SIGACT News15 (1)* 78 - 88, (1983)

[9] C.H. Bennett, G. Brassard: Quantum cryptography: public key distribution and coin tossing. In *Proceedings of the International Conference on Computers, Systems & Signal Processing Bangalore, India*, pp. 175–179. December 10–12,1984

[10] Ekert, Artur K: Quantum cryptography based on Bell's theorem. In *Physical Review Letters. 67 (6): 661–663*, doi:10.1103/PhysRevLett.67.661 (5 August 1991)

[11] Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134. IEEE Computer Society, Washington (1994)

[12] Craig Gidney and Martin Ekerå: How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. December 6, 2019

[13] R. B. Griffiths and C.-S. Niu: Semiclassical Fourier transform for quantum computation. In *Physical Review Letters 76*, 3228-3231 (1996)

[14] C. Zalka: *Shor's algorithm with fewer (pure) qubits*, arXiv preprint quant-ph/0601097 (2006)

[15] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland: Surface codes: Towards practical large-scale quantum computation. In *Physical Review A 86*, 032324 (2012), arXiv:1208.0928

[16] M. Ekerå and J. Håstad: Quantum Algorithms for Computing Short Discrete Logarithms and Factoring RSA Integers. In *Proceedings of the 8th International Workshop on Post-Quantum Cryptography (PQCrypto 2017), Lecture Notes in Computer Science (LNCS), Vol. 10346*, (Springer, 2017) pp. 347-363

[17] M. Ekerå: Modifying Shor's algorithm to compute short discrete logarithms. In *Cryptology ePrint Archive, Report*, 2016/1128 (2016)

[18] M. Ekerå: Quantum algorithms for computing general discrete logarithms and orders with tradeoffs. In *Cryptology ePrint Archive, Report*, 2018/797 (2018)

[19] C. Gidney and A. G. Fowler: Flexible layout of surface code computations using AutoCCZ states. In *Journal*, arXiv preprint arXiv:1905.08916 (2019)

[20] C. Gidney: Approximate encoded permutations and piecewise quantum adders. In *arXiv preprint arXiv:1905.08488*,2019

[21] Franklin, Diana; Chong, Frederic T.: Challenges in Reliable Quantum Computing. In *Nano, Quantum and Molecular Computing*, pp. 247–266. (2004) doi:10.1007/1-4020-8068-9_8. ISBN 1-4020-8067-0

[22] Pakkin, Scott; Coles, Patrick: The Problem with Quantum Computers. In *Scientific American*, 10 June 2019

[23] V. Vedral, A. Barenco, and A. Ekert: Quantum networks for elementary arithmetic operations. In *Phys. Rev. A, 54*, pp. 147-153. Also on arXiv:quant-ph/9511018 (1996)

[24] IBM Quantum: https://quantum-computing.ibm.com/ (2021)

[25] D. Beckman, A.N. Chari, S. Devabhaktuni, and J. Preskill: Efficient networks for quantum factoring. In *Phys. Rev. A, 54*, pp. 1034- 1063. Also on arXiv:quant-ph/9602016 (1996)

[26] C. Zalka: Fast versions of Shor's quantum factoring algorithm. Also on arXiv:quant-ph/9806084 (1998)

[27] S. Parker and M.B. Plenio: Efficient factorization with a single pure qubit and logN mixed qubits. In *Phys. Rev. Lett., 85*, pp. 3049-3052. Also on arXiv:quant-ph/0001066 (2000)

[28] Stephane Beauregard: Circuit for Shor's algorithm using 2n+3 qubits. In *Quantum Information and Computation, Vol. 3, No. 2*, pp. 175-185. Also on arXiv:quant-ph/0205095 (2003)

[29] Michael. A. Nielsen and Isaac. L. Chuang.: *Quantum Computation and Quantum Information*. Cambridge University Press (Cambridge); 2010

[30] Mermin, David: Quantum Computer Science. An Introduction. Cambridge University Press, 2007

[31] Asher Peres, Daniel R. Terno: Quantum Information and Relativity Theory, 2004, https://arxiv.org/abs/quant-ph/0212023

[32] Nigel Smart: Cryptography: An Introduction (3rd Edition)

[33] Buhler, J. P.; Lenstra, H. W. Jr.; Pomerance, Carl (1993): Factoring integers with the number field sieve. In *Lenstra A.K., Lenstra H.W. (eds) The development of the number field sieve. Lecture Notes in Mathematics, vol 1554. Springer, Berlin, Heidelberg*, https://doi.org/10.1007/BFb0091539; Retrieved 11 April 2021

[34] Jonathan Sorenson: An Introduction to Prime Number Sieves. In *Computer Sciences Technical Report #909*, Department of Computer Sciences University of Wisconsin-Madison (January 2, 1990)

[35] Pomerance, Carl; Crandall, Richard: Prime Numbers: A Computational Perspective (Second ed.), New York: Springer. ISBN 978-0-387-25282-7. MR 2156291 (2005)

[36] A. Joux: A new index calculus algorithm with complexity L(1/4 + o(1)) in very small characteristic. In *Selected Areas in Cryptography — SAC*, Springer 2013, LNCS 8282 (2014)

[37] D. Coppersmith: An approximate Fourier transform useful in quantum factoring. In *IBM Research Report No. RC19642*. Also on arXiv:quant-ph/0201067 (1996)

[38] C. Moore and M. Nilsson: Parallel quantum computation and quantum codes. In *SIAM J. Comp., 31*, pp. 799-815. Also on arXiv:quant-ph/9808027 (2002)

[39] Robert B. Griffiths and Chi S Niu: Semiclassical Fourier Transform for Quantum Computation. In *Physical Review Letters, 76(17):3228–3231*, 1996. doi: 10.1103/PhysRevLett.76.3228. URL http://arxiv.org/abs/quant-ph/9511007

[40] M. Mosca and A. Ekert: The hidden subgroup problem and eigenvalue estimation on a quantum computer. In *Lecture Notes in Computer Science, 1509*, pp. 174-188. Also on arXiv:quant-ph/9903071 (1999)

[41] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca: *Quantum algorithms revisited*. Proc. R. Soc. London A, 454, pp. 339-354. Also on quant-ph/9708016; (1998)

[42] Shor, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. In *SIAM J. Comp., 26*, pp. 1484-1509. Also on quant-ph/9508027 (1997)

[43] T. Draper: Addition on a quantum computer, arXiv:quant-ph/0008033 (2000)

[44] A. Barenco, C. Bennett, R. Cleve, D.P. DiVincenzo, N. Margolus, P.Shor, T. Sleator, J.A. Smolin, and H. Weifurter: Elementary gates for quantum computation. In *Phys. Rev. A, 52*, pp. 3457-3467. Also on arXiv:quant-ph/9503016 (1995)

[45] Igor L. Markov; Mehdi Saeedi: Constant-Optimized Quantum Circuits for Modular Multiplication and Exponentiation. In *Quantum Information and Computation, Vol. 12, No. 5&6, pp. 0361-0394, 2012, arXiv:1202.6614v3 [cs.ET]*

[46] Complete circuit for the factorization of $N = 15$ Shor's algorithm with semiclassical QFT: https://raw.githubusercontent.com/martynscn/Thesis/main/shor_semiclassical_final_circuit_long.png

[47] Julia Wallace: Quantum Computer Simulators - A Review Version 2.1, (1999)

[48] H. De Raedt and K. Michielsen: Computational Methods for Simulating Quantum Computers. In *M. Rieth and W. Schommers, editors, Handbook of Theoretical and Computational Nanotechnology, volume 3: Quantum and molecular computing, quantum simulations*, chapter 1, page 248. American Scientific Publisher, 2006. URL http://arxiv.org/abs/quant-ph/0406210.

[49] IBM Transpiler: https://qiskit.org/documentation/apidoc/transpiler.html

[50] IBM Quantum Circuits documentation: https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.html

[51] Mermin, David: Breaking RSA Encryption with a Quantum Computer: Shor's Factoring Algorithm. In *Physics 481-681 Lecture Notes (PDF)*, Cornell University March 28, 2006

[52] L. K. Grover: A fast quantum mechanical algorithm for database search. In *Annual ACM Symposium on Theory of Computing*. USA, 1996, p. 212-219.

[53] T. Apostol.: *Introduction to Analytic Number Theory*. Undergraduate Texts in Mathematics. Springer-Verlag, 1976.

[54] D. Deutsch and R. Jozsa.: Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London A.*, 439:553-558. doi:10.1098/rspa.1992.0167, October 1992.

[55] G. H. Hardy and E. M. Wright.: *An Introduction to the Theory of Numbers.*Oxford University Press, 2008 (6th edition, revised by D. R. Heath-Brown and J. H. Silvermann; 1st edition published in 1938)

[56] Measurement Outcomes for Shor's algorithm with standard and semiclassical QFT: https://github.com/martynscn/Thesis/tree/main/Measurement_outcomes

[57] Complete raw and processed data for Shor's algorithm with standard and semiclassical QFT applied to numbers 15, 21 and 33: https://rebrand.ly/Complete_raw_and_processed_data_shors_simulation

[58] Kefa Rabah : Review of Methods for Integer Factorization Applied to Cryptography. In *Journal of Applied Sciences, 6*, pp. 458-481. DOI: 10.3923/jas.2006.458.481; URL: https://scialert.net/abstract/?doi=jas.2006.458.481 (2006)

[59] Diego F. Aranha : Integer factoring and related cryptosystems. Institute of Computing UNICAMP; URL: https://www.ic.unicamp.br/ rdahab/cursos/mo422-mc938/2016-2s/Welcome_files/fatoracao+Rabin.pdf (2006)

[60] Qiskit Aer: https://qiskit.org/documentation/apidoc/aer.html

[61] Johannes A. Buchmann, Denis Butin, Florian Göpfert, Albrecht Petzoldt: Post-Quantum Cryptography: State of the Art. Technische Universität Darmstadt, Fachbereich Informatik, Hochschulstraße 10, 64289 Darmstadt, Germany; URL: https://core.ac.uk/download/pdf/144828958.pdf; accessed on 19/03/2021

[62] M. R. K. Ariffin, M. A. Asbullah, N. A. Abu and Z. Mahad: A New Efficient Asymmetric Cryptosystem Based on the Integer Factorization Problem. Malaysian Journal of Mathematical Sciences 7(S): 19-37 (2013); URL: https://core.ac.uk/download/pdf/153818781.pdf; (2013); accessed on 19/03/2021

[63] Richard Feynman: Simulating physics with computers. In *International Journal of Theoretical Physics*, pp. 21(6-7), (1982)

[64] Bartłomiej Patrzyk: Review, analysis and simulation of quantum algorithms in cryptography, 2014

[65] Abraham Asfaw and Luciano Bello and Yael Ben-Haim and Sergey Bravyi and Nicholas Bronn and Lauren Capelluto and Almudena Carrera Vazquez and Jack Ceroni and Richard Chen and Albert Frisch and Jay Gambetta and Shelly Garion and Leron Gil and Salvador De La Puente Gonzalez and Francis Harkins and Takashi Imamichi and David McKay and Antonio Mezzacapo and Zlatko Minev and Ramis Movassagh and Giacomo Nannicni and Paul Nation and Anna Phan and Marco Pistoia and Arthur Rattew and Joachim Schaefer and Javad Shabani and John Smolin and Kristan Temme and Madeleine Tod and Stephen Wood and James Wootton: Learn Quantum Computation Using Qiskit, 2020, http://community.qiskit.org/textbook, Accessed on 22/11/2020.

6. List of used abbreviations and symbols

$|.\rangle$ Ket notation. Also represents a column vector

$\langle .|$ Bra notation. A row vector equal to the complex conjugate transpose of $|.\rangle$

$\dagger$ Complex conjugate transpose

* Complex conjugate of a vector

$|a\rangle\ \langle b|$ Ket-Bra notation representing the outer product of a and b.

$\langle b|\ |a\rangle$ Bra-Ket notation representing the inner product of a and b.

$|a\rangle\ |b\rangle$ Ket-Ket notation representing the tensor product of a and b.

$\otimes$ Tensor product

$\mathbb{R}$ Set of Real Numbers

$\mathbb{C}$ Set of Complex Numbers

List of attachments

1. **shor_normal_constant.ipynb:** This contains code for the implementation of the constant optimized version of Shor's algorithm.

2. **shor_standard_qft.ipynb:** This contains code for the implementation of the shor_standard_QFT variant of the Shor's algorithm.

3. **shor_semiclassical_QFT:** This contains code for the implementation of the shor_semiclassical_QFT variant of the Shor's algorithm.

4. **Zip files of csvs:** This contains data saved for the simulation experiments carried out for the two implementation variants of the Shor's algorithm

List of Figures