Simulating Quantum Circuits using the Qiskit library

Project-II (CS47006) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Hardik Pravin Soni

(20CS30023)

Under the supervision of

Professor Indranil Sen Gupta

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Spring Semester, 2023-24

April 7, 2024

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: April 7, 2024
Place: Kharagpur

(Hardik Pravin Soni)
(20CS30023)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Simulating Quantum Circuits using the Qiskit library**" submitted by **Hardik Pravin Soni** (Roll No. 20CS30023) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2023-24.

Indranil SenGupta

Date: April 7, 2024

Place: Kharagpur

Professor Indranil Sen Gupta
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Abstract

Name of the student: **Hardik Pravin Soni** Roll No: **20CS30023**
Degree for which submitted: **Bachelor of Technology**
Department: **Department of Computer Science and Engineering**
Thesis title: **Simulating Quantum Circuits using the Qiskit library**
Thesis supervisor: **Professor Indranil Sen Gupta**
Month and year of thesis submission: **April 7, 2024**

Simulating quantum circuits using the Qiskit library

Simulating quantum circuits is an essential tool for developing and testing quantum algorithms and applications. Qiskit is a popular open-source library for quantum computing that provides a variety of tools for simulating quantum circuits, including:

- A quantum circuit simulator that can simulate circuits up to millions of qubits.
- A noise model simulator that can simulate the effects of noise on quantum circuits.
- A statevector simulator that can simulate the state of a quantum system after applying a circuit.
- A density matrix simulator that can simulate the evolution of a quantum system after applying a circuit.

Qiskit's simulation tools are easy to use and provide a variety of features, such as the ability to:

- Simulate circuits with parameterized gates.
- Simulate circuits with classical control.
- Measure the expectation values of observables.
- Visualize the state of a quantum system.

Qiskit's simulation tools are used by researchers and practitioners around the world to develop and test quantum algorithms and applications. For example, Qiskit has been used to simulate quantum circuits for quantum machine learning, quantum chemistry, and quantum cryptography.

In this abstract, we provide a brief overview of how to simulate quantum circuits using the Qiskit library. We also discuss some of the key features of Qiskit's simulation tools and provide examples of how they can be used to simulate different types of quantum circuits.

Keywords: Qiskit, quantum simulation, quantum circuits, quantum algorithms, quantum machine learning, quantum chemistry, quantum cryptography

Acknowledgements

I express my sincere gratitude to my supervisor, Professor Indranil Sengupta, for granting me the opportunity to work under his mentorship. Throughout my B.Tech project, he provided insightful guidance and constructive feedback. He was consistently supportive and understanding.

I am grateful for the chance to have worked on this project and to have learned from the people I have met along the way. This project has been challenging but rewarding, and I am proud of what I have achieved. I would also like to extend my warmest gratitude to my family and friends at IIT Kharagpur for their unwavering support and motivation throughout my journey.

Contents

List of Figures

List of Tables

Abbreviations

QC	**Q**uantum **C**ircuit
QB	**Q**uantum **B**its

Symbols

ψ	Eigen Vector
$\|\|\rangle$	Quantum Bit
H	Hadamard Gate
X	Pauli X Gate
Y	Pauli Y Gate
Z	Pauli Z Gate
$SWAP$	Swap Quantum Gate
$CNOT$	Controlled-NOT Gate
$CP(\lambda)$	Controlled Phase Gate
$\mathbb{C}$	Complex Number
$\oplus$	Tensor Product
θ	Phase
$CROT_k$	Two-qubit controlled rotation in Block-diagonal Form

Chapter 1

Introduction

1.1 Introduction

Recent years have witnessed a surge in both financial investment and research interest in the field of quantum computing (4). Leading corporations, including Google and IBM, are actively engaged in the development of increasingly potent and versatile quantum computers.
The potential applications of quantum computation encompass a vast array, ranging from the simulation of intricate quantum interactions within chemical systems [7, p. 305] (11) to the implementation of algorithms that harness the unique capabilities of quantum mechanics, offering a significant computational advantage over classical methods.
A significant challenge impeding the realization of fully functional quantum computers lies in achieving scalability. This entails the ability to consistently increase the number of qubits (quantum bits) within a single system or chip. Furthermore, mitigating the impact of noise and decoherence, which introduce errors within the quantum circuit, is crucial for reliable computation.

The paramount challenge in harnessing the potential of quantum computing lies in mitigating the disruptive influence of noise. Quantum computers rely on quantum bits, inherently susceptible to errors compared to their classical counterparts. These delicate qubits are readily affected by external perturbations or interactions with neighboring qubits. This phenomenon, known as noise or decoherence, significantly elevates the likelihood of acquiring erroneous computational outcomes.

1.2 Research question

Before quantum computers are able to reach the levels of qubits needed for currently known error-correction methods and therefore achieving a high probability of receiving correct results, studies need to be conducted on how performance improves when levels of noise decrease. For our thesis we, therefore, ask this question:

> How do varying noise levels and sources affect the success rate and runtime of Shor's algorithm?

1.3 Purpose

The purpose of this thesis is to present an experiment simulating Shor's algorithm with and without the effects of noise, the analysis of the data gathered from said simulation, a discussion of the results of the analysis and any conclusions that can be drawn thereof.

1.4 Method

The data will be generated by performing quantum computing simulations of Shors's algorithm with the Qiskit framework. The Qiskit framework contains the functionality of creating custom noise models that can be applied to quantum computer simulations, as well as containing emulations of real quantum computers. Noise models are used to emulate the noise and decoherence that appear in real-world quantum computers. Through measurement and statistical analysis, we will look for a correlation between noise levels and the performance of Shor's Algorithm. With the analysis, we will then look for how different noise levels affect performance in regard to the probability of correct results and correct phase estimation. We will also measure the runtimes of the simulations.

1.5 Delimitations

For our study, we have chosen only to focus on a version of Shor's algorithm (a specific circuit) designed for factoring the number 15. This circuit only needs 12 qubits which greatly increases the number of times we can run the simulations. The performance requirements and quantum circuit complexity increase the more qubits are used in a simulation. We have chosen to simulate a quantum computer mostly due to our need of being able to control the noise but there are also a limited number of qubits available through free cloudware platforms (as of the time of writing the free service level of IBMQ cloudware platform offers at most 7 qubits) which would limit the depth of the circuit we need. This study is also limited to the effects of applying three error causes, as our custom noise models, to the circuit. These three represent hardware, interaction and environmental causes of noise. They were chosen specifically for being from different sources as well as being the causes of some of the more influential noise. Three is also a good limit within the scope of this study both in terms of work and knowledge required as well as the time constraints. Because of the scope of this paper and the complexity of quantum computing, some of the concepts mentioned in this paper are not given a deeper explanation. An example of this is the Quantum Fourier Transform (QFT) and its inverse (QFT†) which is a critical part of Shor's algorithm.

1.6 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 postprocessing 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

1.7 Outline

In Chapter 2 we introduce the basic concepts of quantum computing, give a brief explanation of Shor's algorithm, introduce the quantum gates used in our implementation of the algorithm, and explain how noise affects quantum computers and the causes behind it. 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 deals with explaining the methodology used for executing this study. It will do so by first describing the tools used, split into two parts. The first explains the programming language Qiskit, which is used to simulate quantum circuits. It also explains the implementation of the quantum circuit part of Shor's algorithm. And the second, the creation and use of Qiskit's built-in functions for custom noise. models. Secondly, we will describe the methods used to analyse the data gathered from the experiment.

Chapter 5 presents the result of the study in the form of tables and graphs with accompanying explanations. And finally, in Chapter 6 we discuss the outcome of the study, our conclusions, the limitations we experienced, and our recommendations for further research.

Chapter 2

Background

In this chapter, we will cover essential ideas in quantum computing and integer factorization in cryptography. Only the fundamental principles required to comprehend the content being delivered.

2.1 Basics of Quantum computing

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.

2.1.1 Qubits and their properties

The difference between Quantum computing and classical computing is the use of quantum bits, or qubits, which are the building blocks of quantum information processing. Classical bits either take on a **0** or a **1** state, whereas qubits can exist in a superposition of both states simultaneously. This is represented using braket notation as $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$,

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{,and } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Because of this property, a qubit has the ability to process more information at one time than a single classical bit and therefore has the possibility of solving certain problems more efficiently than a classical computer.
Another important property of qubits is that they can be entangled. When two qubits are entangled, the state of one qubit cannot be described independently of the state of the other. In a system of n qubits, there are 2^n possible basis states, and the quantum state can be represented as a linear combination of these basis states, with each basis state coefficient corresponding to an amplitude . This two-state system can be visualised by a sphere, where the north and south poles represent the basis states, and a point on the surface of the sphere represents a possible quantum state of the qubit. This sphere is known as the **Bloch sphere**, and it provides a geometric representation of a qubit's state space.

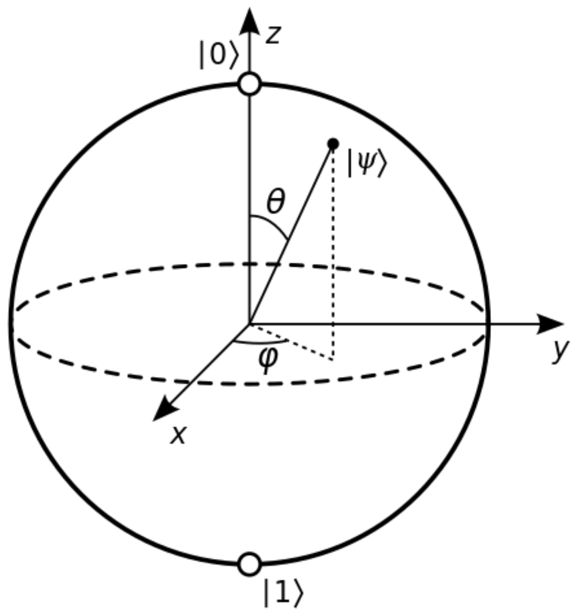


FIGURE 2.1: Bloch Sphere:- Representation of the pure state space of a qubit

2.1.2 Quantum Gates

Since qubits can exist in a superposition of both 0 and 1 simultaneously quantum gates are used to manipulate the state of qubits and perform quantum operations. A specific structure and order of these quantum gates on certain qubits is what is called a quantum circuit. In quantum circuits, these gates are comparable to logic gates used in classical computing. The base gates used for our implementation of Shor's algorithm are as follows:

Pauli-X Gate The Pauli-X gate is a single-qubit gate that inverts the state of a qubit. In the context of Shor's algorithm, X gates are used in controlled multiplication gates for starter guesses (a) for the algorithm. The matrix representation of the X gate is:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Hadamard Gate The Hadamard gate (H) is a single-qubit gate that creates a superposition state from the initial qubit state (27). In Shor's algorithm, the Hadamard gate is used to initialise counting qubits (qubits responsible for modular multiplication) in a superposition state. The matrix representation of the Hadamard gate is:

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

Swap gate The Swap gate is a two-qubit gate that exchanges the states of two qubits. In Shor's algorithm, Swap gates are used to implement modular multiplication within controlled multiplication gates. The matrix representation of the Swap gate is:

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Controlled Phase Gate The controlled phase (CP) gate is a two-qubit gate that introduces a relative phase shift between basis states. In the context of Shor's algorithm, it is used for the implementation of the inverse Quantum Fourier Transform (QFT†). The matrix representation of the CP gate with a phase λ is:

$$\mathrm{CP}(\lambda) = \begin{pmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,e^{i\lambda} \end{pmatrix}$$

2.2 Quantum Fourier Transforms

2.3 Shor's algorithm

The core idea of Shor's algorithm is reducing the problem of factoring a large integer $N = p * q$ to finding the period of the related function $(a^x) \mod (N)$, where a is an arbitrary chosen co-prime of N, $1 < a < N$. By finding the repeat period p so that $(a^p) \mod (N) = 1$, $p \neq 0$, number theory gives us the possibility that the prime factors of N are the greatest common divisors between N and $a^{\frac{p}{2}} + 1$, or between N and $a^{\frac{p}{2}} - 1$. Shor's algorithm is split into two parts, one quantum computing part where the periodicity information is extracted and a classical part where the period is used for calculating the prime factors. These parts will be described in more detail in the following two sections.

2.3.1 Quantum computing part

In classical computing, this relation between the factoring problem and finding the repeat period of a co-prime of N does not speed up the process of finding the prime numbers. However, a quantum computer can make use of the Quantum Fourier Transform (QFT) to efficiently find the repeat period p. The QFT is a linear transformation on quantum bits that allows the efficient extraction of the periodicity information from the quantum state representing the function $a^x \mod (N)$. In the context of Shor's algorithm, the QFT operates on a quantum state that encodes the values of the function $a^x mod(N)$ over a range of x values. By applying the QFT to this state, the algorithm transforms the quantum state into a new state, where the amplitudes of the basis states are proportional to the Fourier coefficients of the original function [8, p. 217]. This concept is called quantum phase estimation (QPE) [8, p. 221]. The transformed state highlights the periodic structure of the function, making it easier to extract the repeat period p. When the QFT has been applied, the next step is to measure the transformed quantum state, which collapses the state into a single basis state with a probability proportional to the square of its amplitude. This measurement yields an estimate of the period p, which can then be used with the continued fractions algorithm to find an estimate of the phase.

2.3.2 Classical computing part

Once the period p is obtained, the quantum computing part of the algorithm has finished and a classical computer can take over. With the repeat period p, the prime factors of N can be determined by calculating the greatest common divisors between N and $a^{\frac{p}{2}} + 1$, or between N and $a^{\frac{p}{2}} - 1$. If the period p found is odd or the greatest common divisors are both trivial factors of N, the algorithm will have to be run again with a new guess of a (in an optimal system where noise did not affect the results). This process continues until the prime factors are found.

2.4 Quantum Noise and its Sources

Quantum systems do not operate free from outside influence and there are many sources that interact with and in turn, influence a quantum system. This unwanted influence can cause errors in qubit gate operations and for the system to decohere into a purely classical state, acting as a normal bit with a value of 0 or 1 [8]. Within the scope of this thesis, the most important sources stem from: the need to control qubits so that they can perform specific operations, reading the state of qubits and the external system within which the quantum system exists. These factors, interactions and influence is what is called Noise. Three highly influential sources of Noise are gate infidelities, measurement errors and thermal relaxation errors (19), which we will, in turn, describe more in detail.

2.4.1 Gate Infidelities

Another term for Gate Infidelities is depolarizing channel, and it describes the probability of a qubit depolarizing into a probabilistic mixture of pure states, a mixed state. This can also be seen as a contraction of the Bloch sphere. The effect can be viewed as contractions in regard to any combination of the axis used to describe the Bloch sphere if the depolarization is not uniform. For example: a contraction of the $\hat{y} - \hat{z}$ plane of the Bloch sphere would represent a bit-flip (or Pauli-X) error (27).

2.4.2 Measurement Error

Measurement Errors describe the probability of getting the wrong value when reading a qubit. The act of reading a qubit causes the superposition to collapse into a pure state and the interference of reading can itself influence which pure state the qubit collapses into. The error probability of a measurement error describes the chance of measuring a qubit as having the value t when the true value should have been s, where t and s are arbitrary pure state values (27)(2).

2.4.3 Thermal Relaxation Error

Thermal Relaxation Errors are caused by interactions between the qubits and the environment and take the form of two different errors: (i) relaxation and (ii) dephasing of the qubits over time. Relaxation involves an exchange of energy between qubits and the environment causing the qubits to deexcite into the ground state (normally $|0\rangle$) or excite into the exited state (normally $|1\rangle$). Dephasing is the decay of coherence over time where a system decoheres into a classical state, as described previously. Relaxation and dephasing times are commonly denoted as T1 and T2 respectively (27)(2).

2.5 Related Work

This next section introduces some previously conducted relevant work related to quantum computing and Shor's algorithm. Peter W. Shor expresses, in his paper "Polynomial-Time Algorithms for Prime Factorisation and Discrete Logarithms on a Quantum Computer", that digital computers have generally been believed to be an efficient universal computing device but that this may not be true when quantum mechanics is taken into consideration (33). The main thesis of the paper shows that quantum computers can use characteristics of quantum mechanics like superposition and entanglement to speed up computing and operate on multiple bits of information at once. Shor proposes two new algorithms on how this can be conducted:- One for efficient prime factorisation and one for solving discrete logarithms, and notes that the current best algorithm for finding prime numbers, the number field sieve, has a time complexity of $\exp(c(\log n)^{\frac{1}{3}}\ (\log\log n)\frac{2}{3})$ for some constant c in order to factor an integer n. The quantum algorithm Shor discovers has a time complexity of $O((\log n)^2(\log\log n)(\log\log\log n))$ on a quantum computer, along with a polynomial (in log n) amount of post-processing time on a classical computer - which speeds up the factorisation exponentially. The paper expresses the findings about these algorithms as groundbreaking, as the prime factorisation algorithm (which is now called Shor's algorithm (2)) for example could break RSA encryption if found viable, but it also explains that there are many hurdles in the way of building efficient quantum computers. Since quantum bits cannot be cloned - as stated in the No-cloning theorem - it rules out using the same error correction methods as in classical computers, and the paper suggests that issues with precision and decoherence may be incredibly difficult to solve. Shor notes that most algorithms in computer science have turned out to be polynomial or NP-complete and that quantum computers may not be widely useful unless they can solve NP-complete problems.

The paper "Entanglement simulation of Shor's algorithm" is mostly concerned with the correlation between entanglement and the degree of mixing of the initial qubits and the control (ancillary) qubits (25). The inclusion of two different models of noise, and the analysis of their effects on the efficiency of an implementation of Shor's algorithm when applied with varying degrees of probability, is what is most relevant for this study. The results support our hypothesis in regards to lower efficiency (more runs of the algorithm required) the more noise that is introduced. This study only analysed how varying amounts of noise probability affected the number of runs needed, and does so with two specific noise models.

The paper "Near term implementation of Shor's Algorithm using Qiskit" by Casimer DeCusatis, et al, investigates a near-term implementation of Shor's algorithm using IBM's quantum computing framework Qiskit and discusses the limitations of current practical uses in quantum computing (2). It discusses the current uses of the IBMQ System One quantum computer which had its largest publicly available system being 16 qubits at the time of writing. The system can be accessed through a cloud interface and uses a graphical user interface (GUI) called Circuit Composer to build quantum circuits. The paper provides a Circuit Composer diagram with a 4-qubit input register for factoring the number 15 using Shor's algorithm and shows the results of computing it on a simulated quantum computer and a real one. It notes that the Qiskit framework can be easily adopted for research or education purposes, and comes with the conclusion that while current quantum computing may be good for educational purposes, limitations of noise and the lack of qubits, hinder performance too greatly to achieve any practical results. ”Investigation of Noise Effects for Different Quantum Computing Architectures in IBM-Q at NISQ Level” investigates the effects of noise on NISQ-level quantum computers [20]. At the current date, all quantum computers are Noisy Intermediate-Scale Quantum (NISQ) level, which implies that they contain 50-100 qubits and are not yet fault-tolerant. The paper tested the effects of decoherence on quantum computers with different qubit connections and found that the accuracy of the results produced by the IBM quantum computer Athens was the highest and where the standard deviation was the least. The qubits of this computer had 4 connections in a linear fashion which showed that when qubit connections increased the deviations and the deterioration in the results increased.

Chapter 3

Building Blocks for 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.

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

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

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

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 $|x_i x_j\rangle$ 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

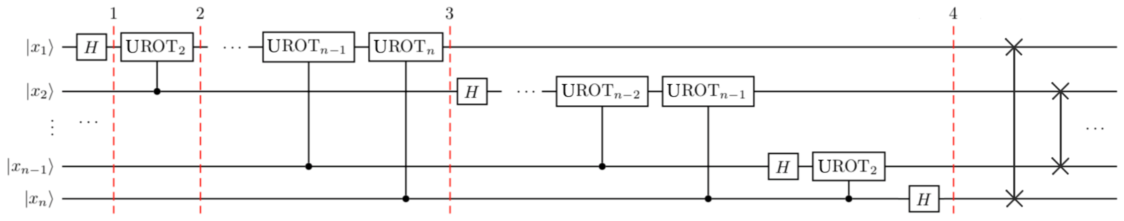


FIGURE 3.1: The Quantum Fourier Transform Circuit for $N = 2^n$ (35)

Starting with an n-qubit input state $|x_1x_2 \ldots x_n\rangle$, steps to implement the circuit are as follows:

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

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

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

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

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 (27).

In IBM Qiskit (12), 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 (27) 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 (17) 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 (17). 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 (34). 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 $U|\psi\rangle = e^{\boldsymbol{2\pi i\theta}}|\psi\rangle$. Here $|\psi\rangle$ is an eigenvector of U and $e^{\boldsymbol{2\pi i\theta}}$ is its corresponding eigenvalue. The general quantum circuit for QPE is shown in figure 3.2.

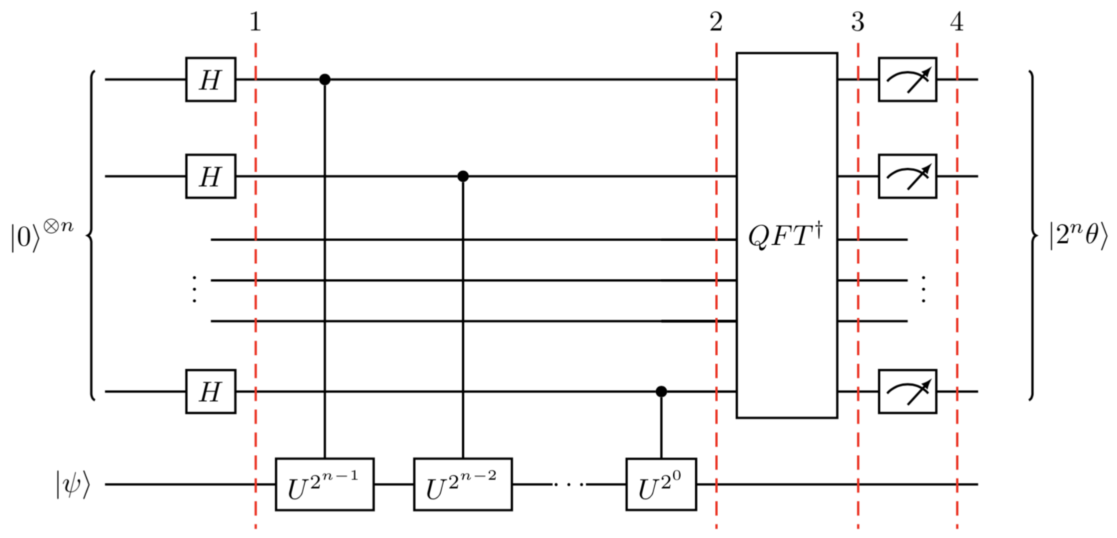


FIGURE 3.2: The Quantum Phase Estimation Circuit for $N = 2^n$ (35)

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 $U|\psi\rangle = e^{\boldsymbol{2\pi i\theta}}|\psi\rangle$ 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 $|0\rangle \otimes |\psi\rangle + |1\rangle \otimes e^{2\pi i\theta}|\psi\rangle = \big(|0\rangle + e^{2\pi i\theta}|1\rangle\big) \otimes |\psi\rangle$ to get:

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

$$= \frac{1}{2^{\frac{n}{2}}}\sum_{k=0}^{2^n-1} e^{\boldsymbol{2\pi i\theta k}}|k\rangle \otimes |\psi\rangle \tag{3.3}$$

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

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

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 (27).

3.3 Shor's Algorithm

Quantum factorization generally consists of classical pre-processing, quantum algorithm for order-finding and classical post-processing (27; 6). 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 (27) 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.
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 $a^{r/2} \neq -1(\mathrm{mod} N)$ (27; 6). 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 (6). 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

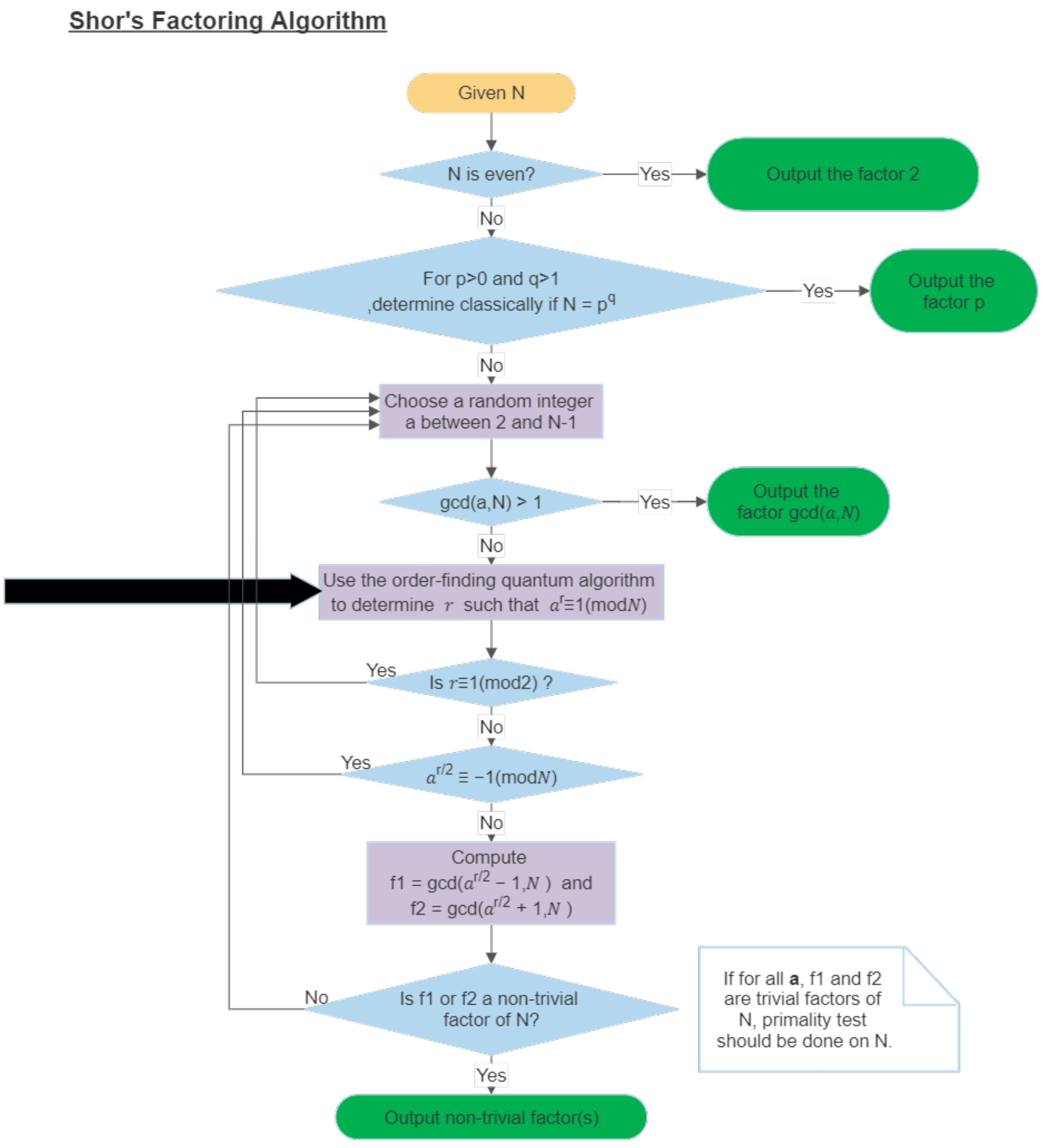

Shor's Factoring Algorithm
Given N
N is even?
Yes
Output the factor 2
No
For p>0 and q>1 ,determine classically if N = p^q
Yes
Output the factor p
No
Choose a random integer a between 2 and N-1
gcd(a,N) > 1
Yes
Output the factor gcd(a,N)
No
Use the order-finding quantum algorithm to determine r such that a^r≡1(modN)
Yes
Is r≡1(mod2) ?
No
Yes
a^{r/2} ≡ −1(modN)
No
Compute f1 = gcd(a^{r/2} − 1,N) and f2 = gcd(a^{r/2} + 1,N)
No
Is f1 or f2 a non-trivial factor of N?
Yes
Output non-trivial factor(s)
If for all a, f1 and f2 are trivial factors of N, primality test should be done on N.

FIGURE 3.3: Overview of Shor's factoring algorithm

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:

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

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

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:

$$\begin{aligned} |u_s\rangle &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi isk}{r}} |a^k \bmod N\rangle \\ U|u_s\rangle &= e^{\frac{2\pi is}{r}} |u_s\rangle \end{aligned} \tag{3.5}$$

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

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:

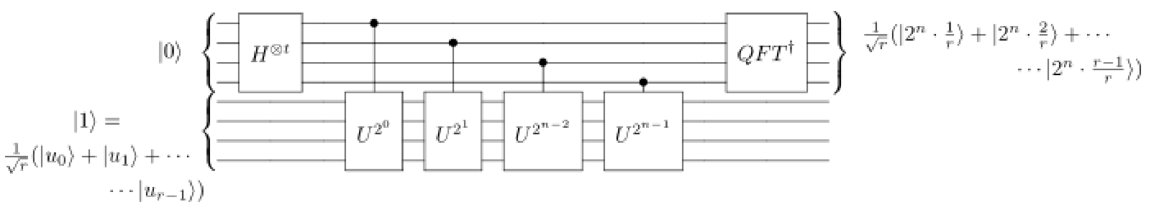

$|0\rangle$
$H^{\otimes t}$
$QFT^\dagger$
$|1\rangle = \frac{1}{\sqrt{r}}(|u_0\rangle + |u_1\rangle + \cdots \cdots |u_{r-1}\rangle)$
U^{2^0}
U^{2^1}
$U^{2^{n-2}}$
$U^{2^{n-1}}$
$\frac{1}{\sqrt{r}}(|2^n \cdot \frac{1}{r}\rangle + |2^n \cdot \frac{2}{r}\rangle + \cdots \cdots |2^n \cdot \frac{r-1}{r}\rangle)$

FIGURE 3.4: Shor's period finding algorithm for U^{2^j} (35)

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

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 exponent's 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 (26) 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 (27).

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 $O\left(n^3\right)$ 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 (27):

Theorem 2. Suppose N is an n bit composite number, and x is a non-trivial solution to the equation $x^2 = 1(\text{mod}N)$ in the range $1 \leq x \leq N$, that is, neither $x = 1(\text{mod}\ N)$ nor $x = N-1 = -1(\text{ mod } N)$. 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 $O\left(n^3\right)$ 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 (27):

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

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 (26).

3.6.1 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 (23) 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.

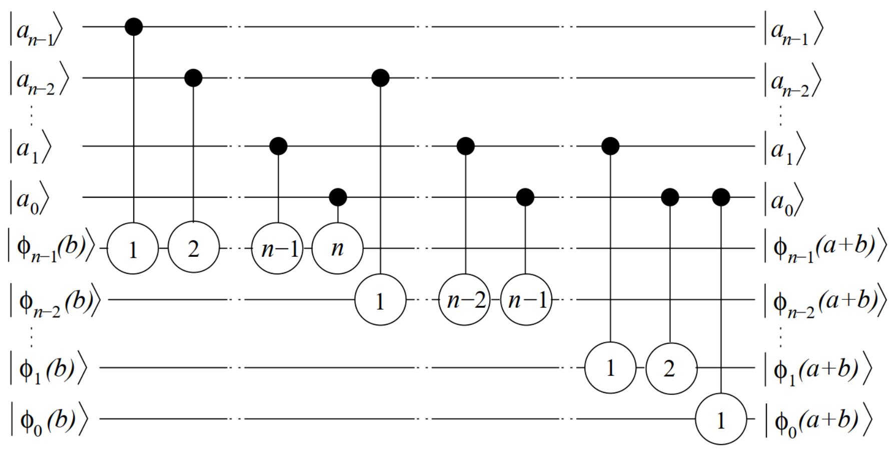


FIGURE 3.5: Quantum addition as given in (26)

3.6.2 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.

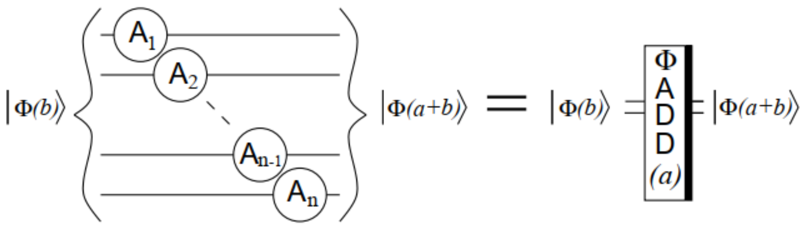


FIGURE 3.6: Adder gate (26)

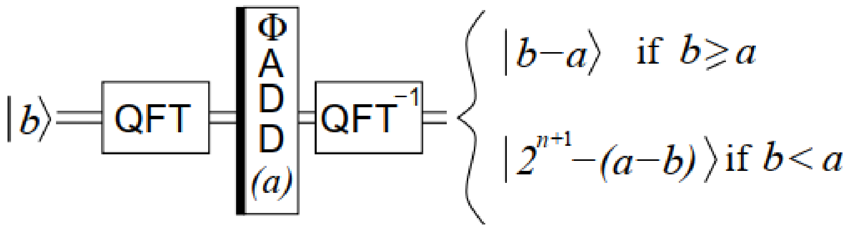


FIGURE 3.7: Adder gate inverse (26)

3.6.3 Modular adder gate

The $\Phi ADD(a)$ gate, is used in building a modular adder gate (figure 3.8) and we denote it as $\Phi ADD(a) \bmod N$. 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:

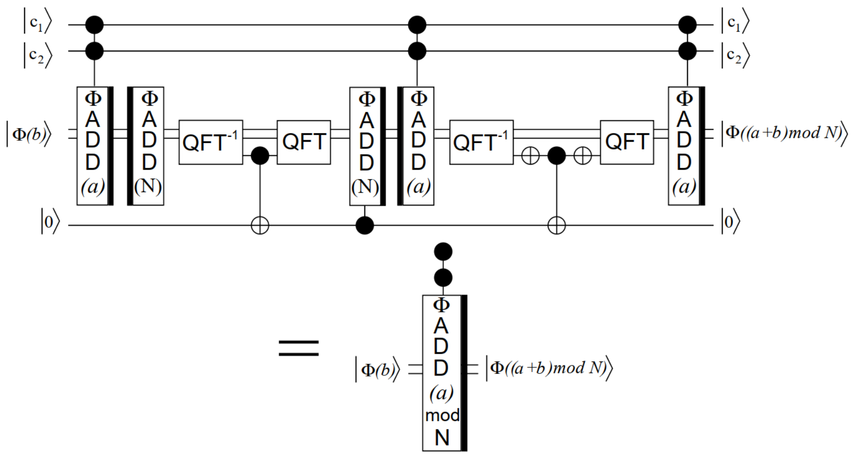


FIGURE 3.8: Modular adder gate (26)

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 $(a+b)$ mod $(N-a)$ so as to compare $(a+b)$ mod 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 $\Phi ADD(a)$ mod 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 $b < N$.

3.6.4 Controlled-SWAP

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

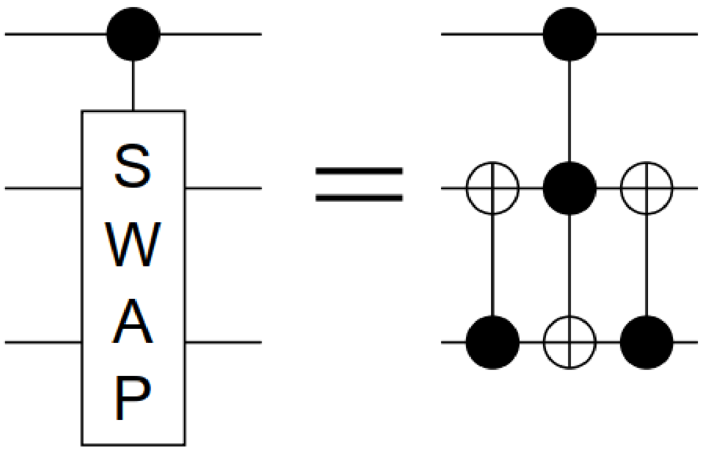


FIGURE 3.9: Controlled SWAP gate (26)

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.

3.6.4.1 Controlled multiplier gate

The doubly controlled $\Phi ADD(a)$ mod N gate is used in building a controlled multiplier gate denoted $CMULT(A)$ mod N as shown in figure 3.10. Its inputs are

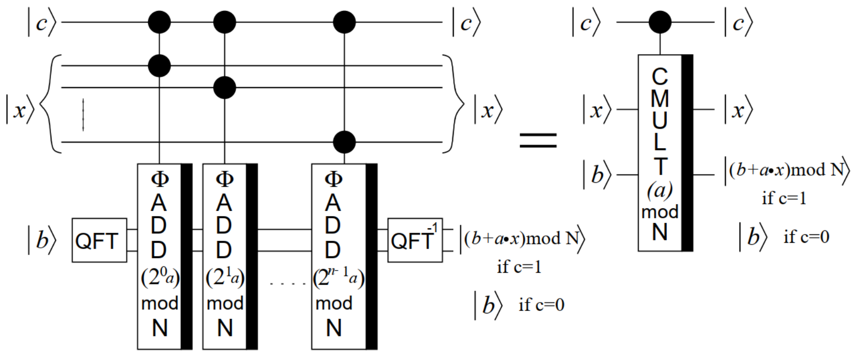


FIGURE 3.10: Controlled multiplier gate (26)

$|c\rangle|x\rangle|b\rangle$ 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 $|c\rangle|x\rangle|b\rangle$

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

$$(ax) \bmod N = \left(\dots\left(\left(2^0 a x_0\right) \bmod N + 2^1 a x_1\right) \bmod N + \dots + 2^{n-1} a x_{n-1}\right) \bmod N \tag{3.7}$$

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 $0 \le i < n$. 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:

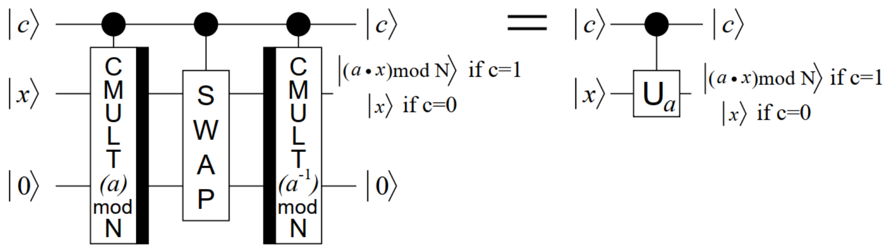


FIGURE 3.11: Controlled U_a gate (26)

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 $|c\rangle|x\rangle|b\rangle \implies |c\rangle|x\rangle|(b - a^{-1}x) \bmod N\rangle$.

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:

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

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:

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

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 3.6.1 while more details on the classical plain adder which uses the NOT, CNOT and Toffoli gates can be found in (18).

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) (18): 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 $4n + 2$.

4. Variant 4: Quantum modular exponentiation with QFT adder + semiclassical QFT. This further optimizes the number of required qubits to $2n + 3$.

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 (21). Stephane Beauregard then worked on this idea and developed the circuit which implemented it as discussed in (26).

The quantum controlled rotations $C - U_{a^{2^j}}$ could be replaced with 'semi-classically' controlled rotations of the subsequent qubits (21; 24). 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.

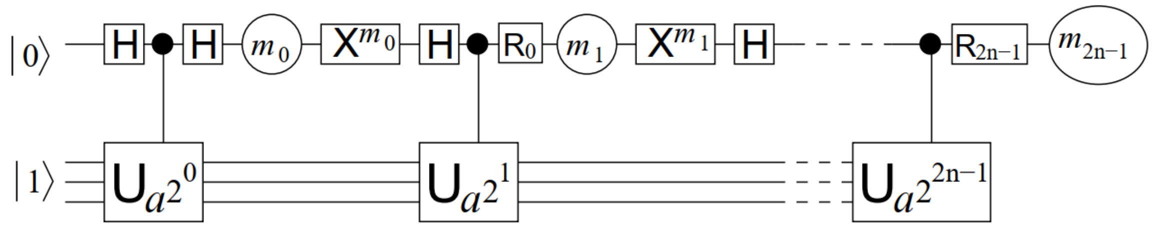


FIGURE 3.12: Factoring using the one control qubit technique (26)

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 (22), 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 (26). The $CMULT(a)$ mod N circuit is a total of n doubly controlled $\Phi ADD(a)$ mod N so requiring $2n+3$ qubits, $O(n^2 k_{max})$ 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, $O(n^2 k_{max})$ 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, $O(n^3 k_{max})$ 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 $O(n^3 log(n))$ 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.

Chapter 4

Methodology

4.1 Qiskit

Qiskit is an open-source software development kit (SDK) founded by IBM Research for working with quantum computers (13). It allows users to develop quantum algorithms, simulate quantum circuits, and execute them on real quantum hardware or simulators. Qiskit offers a suite of tools and libraries for building custom noise models, emulating noise and simulating them on quantum computer simulators (13). The SDK is written for use in Python. In our implementation of Shor's algorithm, the circuit was constructed with the help of `https://learning.quantum.ibm.com/` and the noise models were built and run with the Qiskit Aer quantum computer simulator (13). The constant optimized circuit was explicitly built for factoring the number 15 and used 12 qubits, 8 counting qubits used for modular multiplication in the phase estimation part of Shor's algorithm, and 4 ancillary qubits used to store intermediate results. See Figure 4.1 for a visual interpretation of the implementation of the circuit. In the next section we explain the implementation of all previously described circuits in further detail.

4.2 Implementation of Different Shor's Circuits

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 5, we will present the results from the other simulations and analyse these results. As discussed in Sub-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 + semi-classical QFT
3. **Variant 3**: Quantum modular exponentiation with QFT adder + Standard QFT
4. **Variant 4**: Quantum modular exponentiation with QFT adder + semi-classical QFT

With respect to these variants, the simulation was done along the following line:

- Simulating Variant 1 with a Constant-Optimized Quantum Circuits for Modular multiplication which we referred to as **shor_normal_constant** in Section 4.2.1 along with noise models analysis.
- 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 4.2.4

- Simulating variant 4 by using the modular exponentiation circuit which uses Semiclassical QFT and which we referred to as **shor_semiclassical_QFT** in Section 4.2.5

4.2.1 Shor's Algorithm: 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 $U|y\rangle = |ay \mod 15\rangle$, we used a simplified version of the modular exponentiation circuit based on the results from Igor et al. (2012) (32). This simplified the modular multiplication circuit for different choices of $a \mod N$ as shown in Figure 4.1. Furthermore, we notice that by a manual computation of these constant-optimized circuits for $f(x) = Cx \mod 15$, 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

$$U^{2^j} = |a^{2^j} y \mod N\rangle \tag{4.1}$$

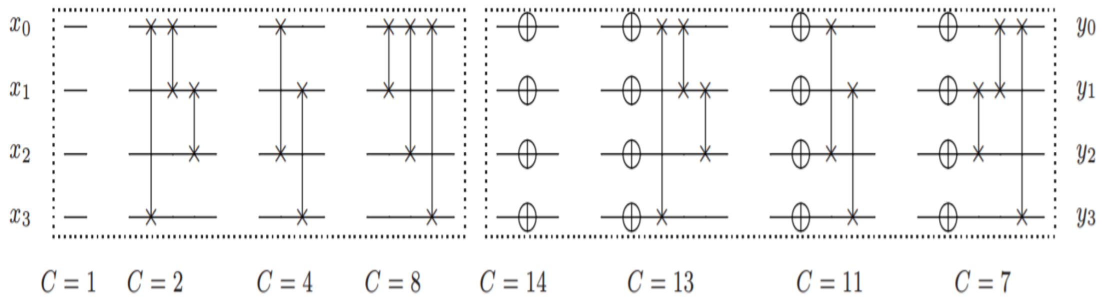


FIGURE 4.1: Circuits for f(x) = Cx mod 15, gcd(C, 15) = 1, (left); C = 2k, (right) C = 15 - 2k

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 4.1 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 4.2. Measurement outcomes are shown in Figure 4.4.

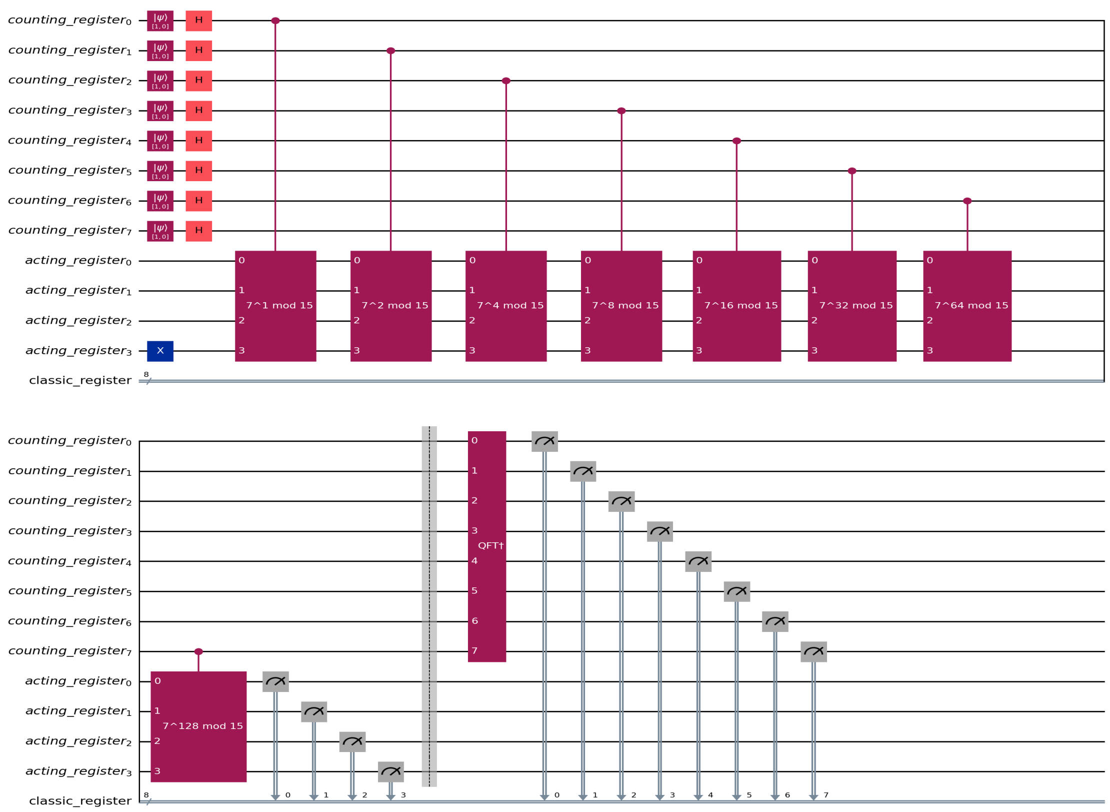


FIGURE 4.2: Constant-Optimized Shor's Circuit

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

- Using continued fractions algorithm, we get $\frac{0}{1}$, $\frac{1}{4}$, $\frac{1}{2}$ and $\frac{3}{4}$. Hence $r = 2$ or 4. This clearly agrees with the graph of $7^x (\mod 15)$ shown in figure 4.3 showing that the period is indeed r = 4
- Since r is even and $a^{\frac{r}{2}} + 1 = 7^2 + 1 = 5(\mod 15)$ is a non-trivial factor of 15, we compute $gcd(5, 15) = 3$ and hence we get the factors of $15 = 35$.

This entire circuit used 12 qubits and 8 classical bits.

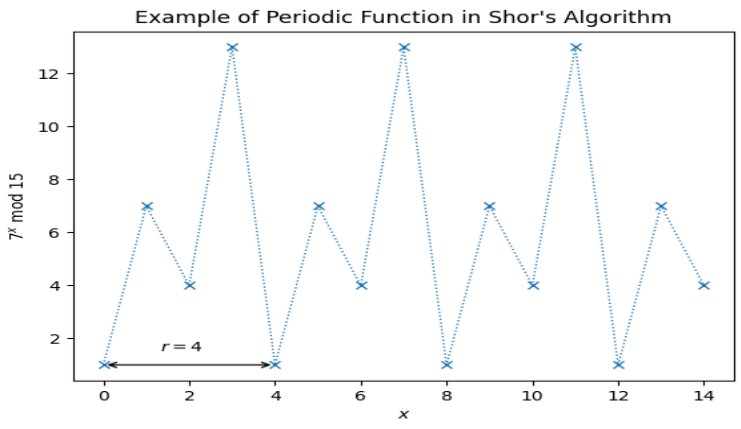
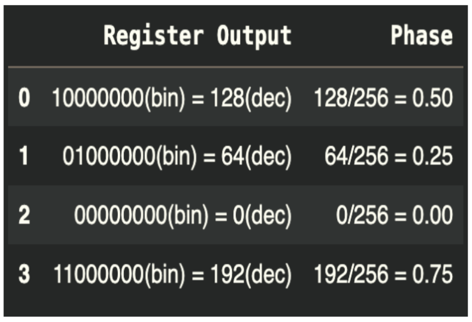


FIGURE 4.3: Graph of $(7^x) \mod 15$



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

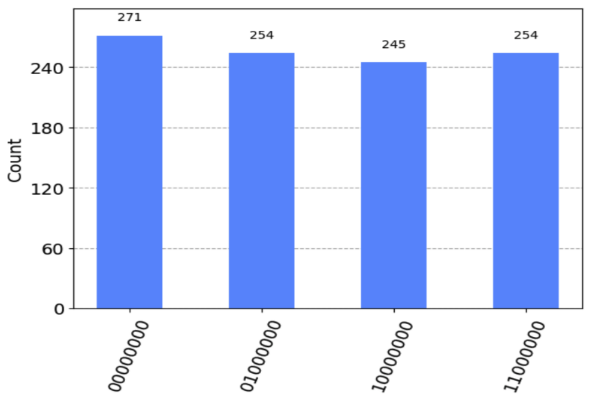


FIGURE 4.4: Measurement results and Measured Phases

4.2.2 Description of Noise Models Employed

A custom noise model was generated for our quantum circuit to emulate noise from a real quantum computer by simulating median error readouts from the IBM Guadalupe quantum computer. The IBM Guadalupe quantum computer was selected due to its 16-qubit capacity being closest in range to our 12-qubit circuit, which provides us with an accurate evaluation of noise levels for quantum computers of this size. The noise model incorporated readout errors, single and two-qubit depolarizing errors, and thermal relaxation errors using median values from the Guadalupe quantum computer (12). The corresponding medians were for the Readout-, CNOT- and Pauli-X gate error rates, and the T1 and T2 times. The medians can be seen in table 4.1. The custom model constructed was then used to run simulations with scaled error probabilities and scaled T1 and T2 times from 100% of their actual median readings to 50% and 25%, analysing how the phases measured after runs differed.

Error	**Error Value**
Pauli-X error rate	$2.385e^{-4}$
CNOT error rate	$8.369e^{-3}$
Readout error rate	$1.745e^{-2}$
T1 relaxation time (us)	97.64
T2 dephasing time (us)	111.17

Table 4.1: Median error values from the IBM Guadalupe quantum computer

To evaluate the accuracy of our custom noise model, we compared its results to noise-free circuit simulations and simulations using Qiskit Aer's emulated noise model. The library fake_provider in Qiskit Aer enabled us to implement an emulated noise model called FakeGuadalupeV2 from the quantum computer our error probabilities were based on (12). By running simulations with our custom noise model and the FakeGuadalupeV2 backend, we could assess the similarity between the two.

4.2.2.1 Custom noise model

The custom noise model constructed focused on three quantum error channels for modelling errors:

Gate Infidelities Depolarizing errors were applied on our single-qubit gates (Pauli-X gates and Hadamard gates) with the error value of the median readout for Pauli-X errors on IBM Guadalupe. They were also applied on our two-qubit gates (SWAP gate and Controlled Phase gate) with the error value of the median readout for CNOT errors on IBM Guadalupe.

Measurement Error Readout errors were applied to all qubits with the error value of the median readout for Readout errors on IBM Guadalupe.

Thermal Relaxation Error Thermal relaxation errors were applied to single-qubit gates (Pauli-x gate and Hadamard gates) with the median readout T1 and T2 times on IBM Guadalupe.

4.2.2.2 Noise profiles

A few models with variations of errors applied were constructed. These were Shor's algorithm circuit with

1. No noise, which was used for baseline evaluation of Sum of Squares Error(SSE) compared to the noisy circuits and for analysing differences in success rate.
2. Aer's built-in noise model FakeGuadalupeV2 emulating noise from the IBM Guadalupe quantum computer.

3. Our custom noise model emulating noise levels from IBM Guadalupe but with three error channels applied on the specific gates used in our implementation of Shor's algorithm.
4. Our custom noise model emulating noise levels from IBM Guadalupe but with only one type of error applied.
5. our custom noise model emulating noise levels from IBM Guadalupe but with combinations of the errors chosen applied.

Our main interest were that of simulating our quantum circuit with the most complete custom noise model and to compare it with a simulation without any noise. To compare the validity of the complete noise model we also used an emulation of a real actual quantum computer, the one we ourselves based the values for our custom model on, for a simulation. Lastly we simulated the different quantum error channel, with *gate infidelities* divided into one-qubit gate errors and two-qubit error, each separately and then all combinations of different channels. This is summarised in table 4.2.

4.2.3 Evaluation of performance

4.2.3.1 Success Rate

In Shor's algorithm the most straight-forward way to evaluate performance is to see if the algorithm is successful in finding a factor. Consequently, when evaluating performance with different noise models applied our main measurement of success were the amount of times the phases measured from the quantum part of the computation succeeded in finding the correct repeat period, so that a factor calculated from $gcd(N, a^{\frac{p}{2}} + 1), or gcd(N, a^{\frac{p}{2}} - 1)$ could be found. As such, for success rate we counted the amount of times the algorithm found a factor (successful attempt), and the amount it did not (failed attempt). Every noise simulation were run 100,000

Table 4.2: The different noise models tested on our Shor's algorithm quantum circuit. Our custom noise model is divided into three different categories: the complete model (full), single error source model (single) and different combinations of error sources (combination)

Noise Model	**Description**
No Noise	Shor's algorithm circuit simulated without any noise.
FakeGuadalupeV2	A snapshot of real operational values from IBM's Guadalupe quantum computer that can be used as, part of Qiskits *fake_backend* library, when simulating quantum circuits.
Custom Full	This is our main model, implementing the gate infidelities (for 1-qubit and 2-qubit gates), measurement error and thermal relaxation error. The error probabilities were taken from the most recent values given by IBM for the Guadalupe quantum computer.
Custom Single	Runs with the 1-qubit, 2-qubit, measurement and thermal relaxation errors noise models applied in turn for each run. Every complete run only had one of these errors applied.
Custom Combination	Noise models of every combination of the four noise channels: gate infidelities for 1- and 2-qubit gates, measurement errors and thermal relaxation errors.

times each with all possible guesses of a (a = 2, 4, 7, 8, 11, 13) for the number 15 and from these runs we could extract a percentage of successful versus failed attempts.

4.2.3.2 SSE

Sum of Squares Error (SSE) is another metric used to evaluate the performance of Shor's algorithm, which focuses on the impact of noise by comparing it to noiseless simulations. SSE represents the sum of the squared differences between each data point and its group's average and displays the dispersion of data points within a cluster. When all data points in a cluster are identical, the SSE is 0. To calculate SSE, the noiseless circuit for each guess of a served as the baseline, and measurements were taken against circuits with noise for each guess as well as across all error probabilities. The formula for SSE is:

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{4.2}$$

Where n represents the number of observations (100,000), y_i represents the mean of the noiseless observed frequencies, and $\hat{y}_i$ represents the mean of the noisy observed frequencies for each noisy circuit run.

4.2.3.3 Runtime

The runtimes for all circuit simulations were also observed. These are highly dependent on the type of system the simulations are performed on but were only used as a metric for analysing the performance of quantum circuit simulations, all run on the same system.
The runtime was recorded from the start of the 100,000 simulations for one noise model with one guess of a to when the simulations were finished. The transpiling of the circuit was not included in the recorded time.
The system used for the simulations was a MacBook Air (13-inch, 2020), 1.8 GHz Dual-Core Intel Core i5, 8 GB 1600 MHz DDR3, Intel HD Graphics 6000 1536 MB.

4.2.3.4 T-test

A t-test was also conducted between the custom noise model at all three scaled error values compared to the FakeGuadalupeV2 noise model to assess the 21 similarity between the two. The calculation was performed as a two-tailed, twosampled equal variance t-test

4.2.4 Shor's Algorithm: 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 3.1. It uses the standard implementation of the QFT as discussed in Section 3.1.1. 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.

We applied the multiplication gates as showed in the work by Stephane Beauregard 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 3.10 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 4.6 where the circuits for $\phi ADD(a)$, $\phi ADD(a) modN$, controlled-SWAP and

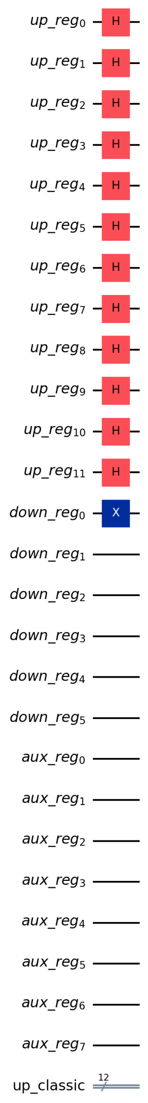


FIGURE 4.5: Initialization of the Circuit

CMULT(A) mod N are all embedded within the modular exponentiation U^{2^j} shown in the figure. Running this

circuit 1024 times on IBM QASM_simulator returned 84 different possible measurements 52 with varying frequency. Figure 4.7 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

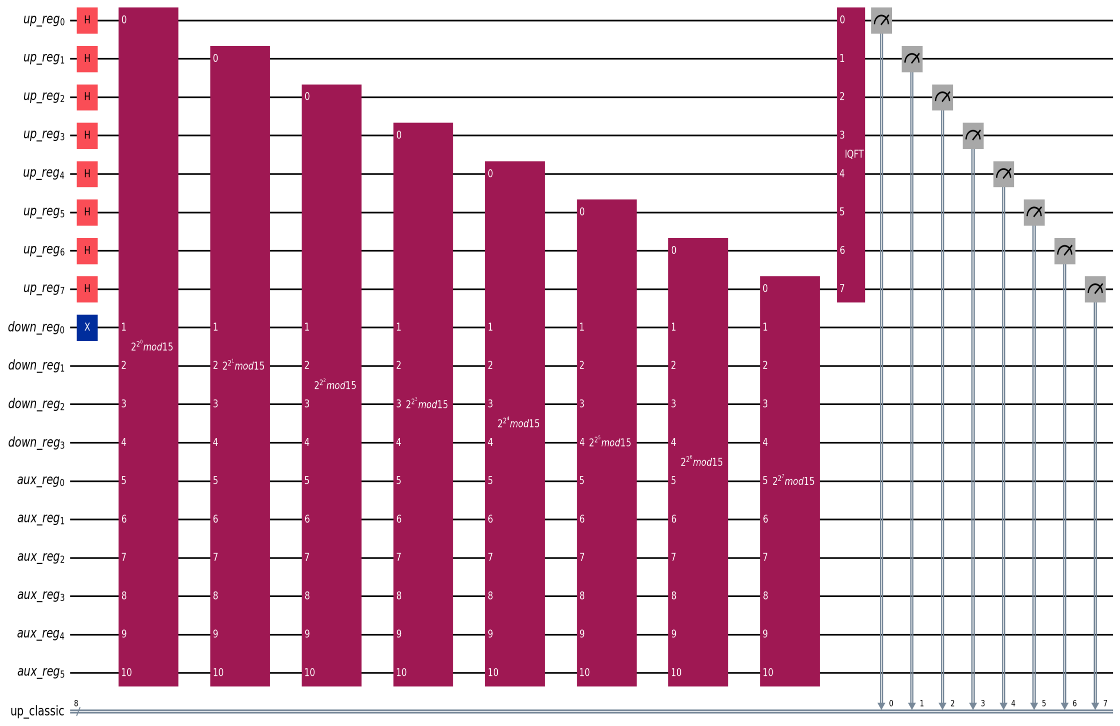


FIGURE 4.6: Final circuit of the shor_standard_QFT variant for factoring N = 15

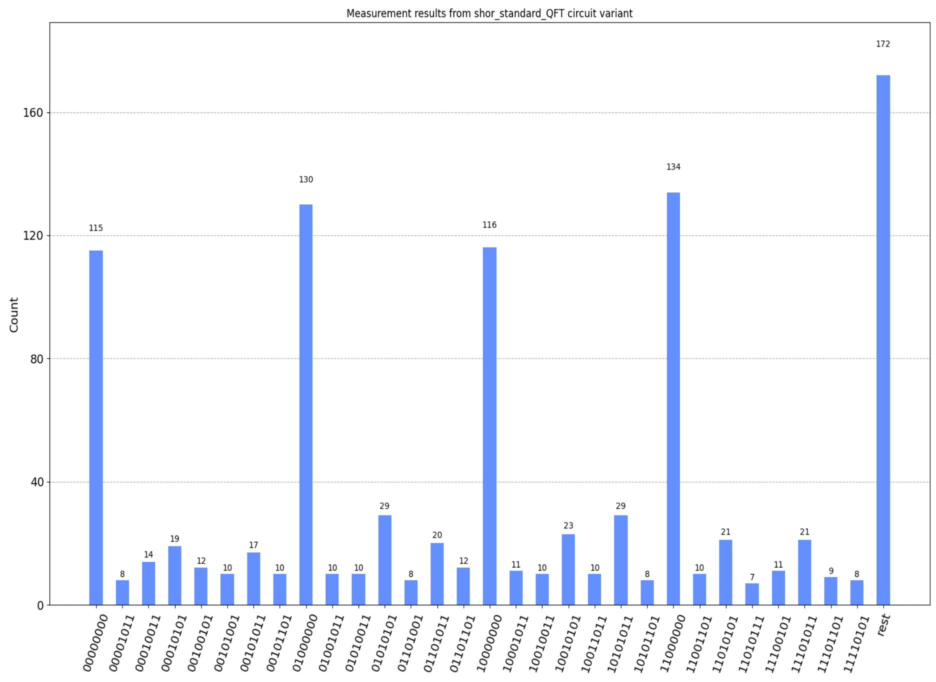


FIGURE 4.7: Measurement results of the shor_standard_QFT variant for factoring N = 15

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 × 5. We also notice from the results that the probability of success $P(s) \geq 0.126+0.113+0.130 = 0.369$. 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.

4.2.5 Shor's Algorithm: 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 semi-classically 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 auxiliary 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 4.6 while the final part is shown in figure 4.9.
- 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 128 different possible measurements with varying frequency.

Figure 4.10 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:

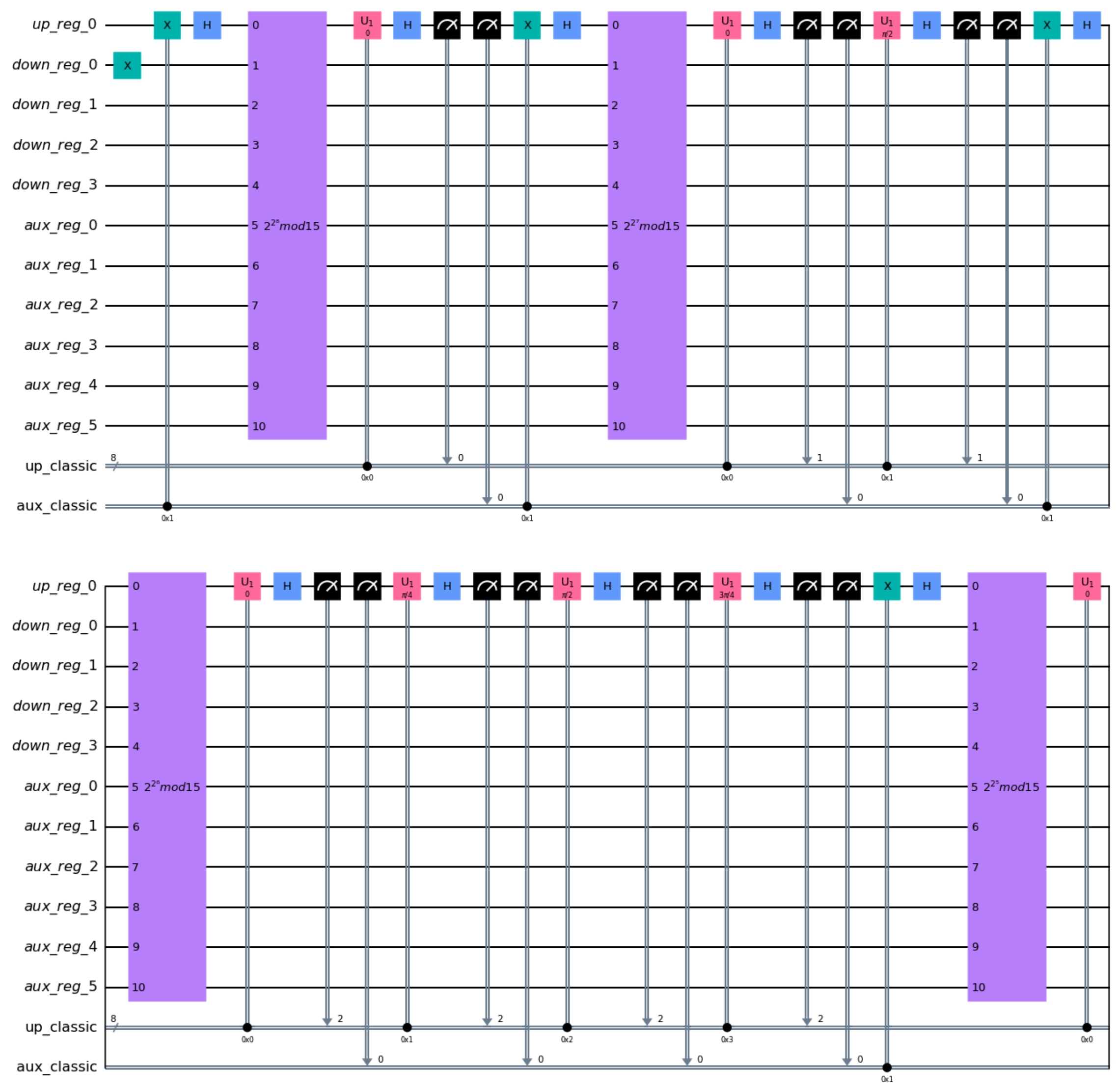


FIGURE 4.8: Beginning part of the Shor's factorization circuit for N = 15 with **Semiclassical QFT**

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 $\phi = \frac{s}{r}$.

3. We applied the continued fraction algorithm to the phase ϕ with a possible maximum denominator of 103 since we know the size of N and to save time and computing resources.

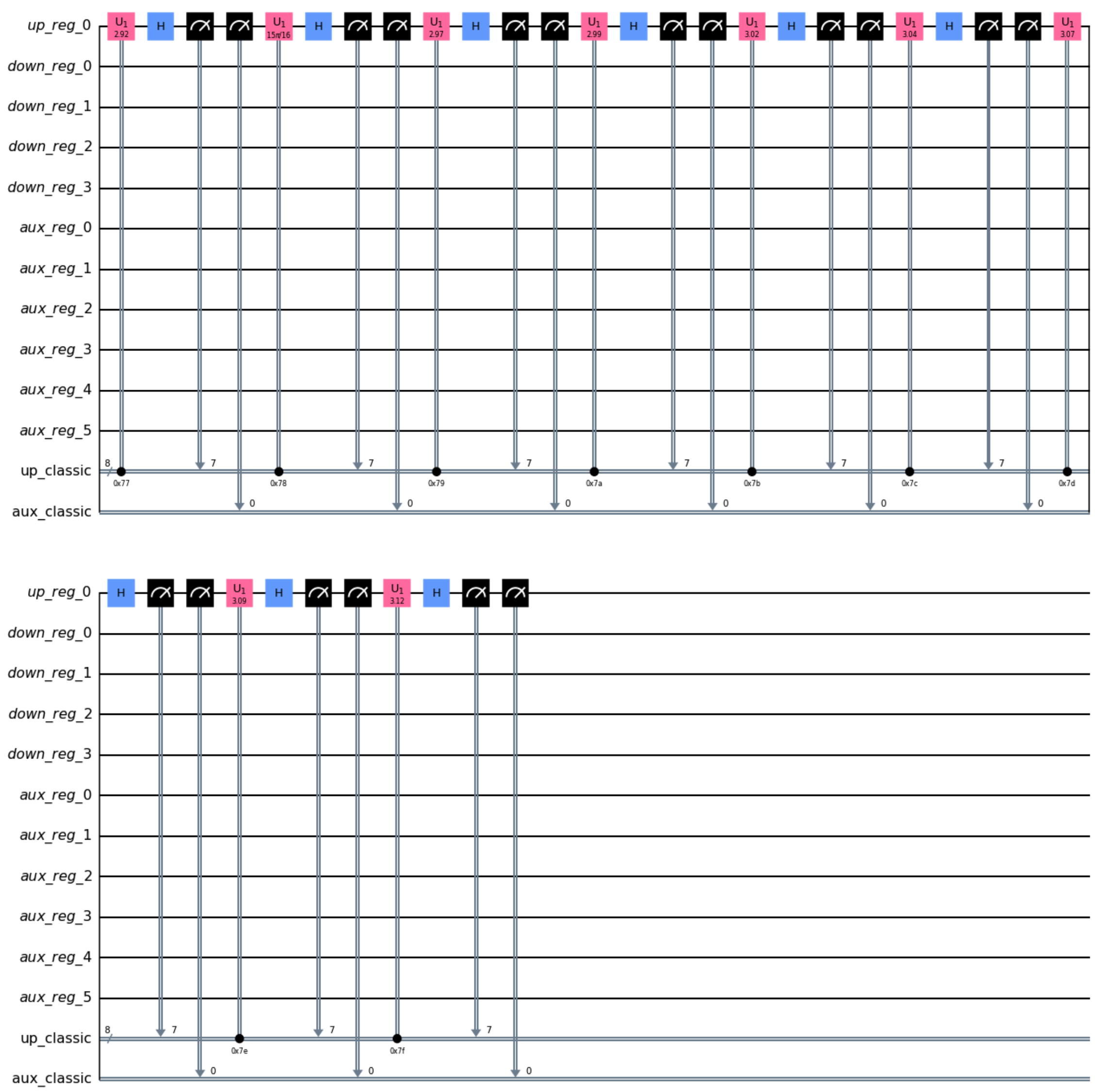


FIGURE 4.9: Final part of the Shor's factorization circuit for N = 15 with **Semi-classical QFT**

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

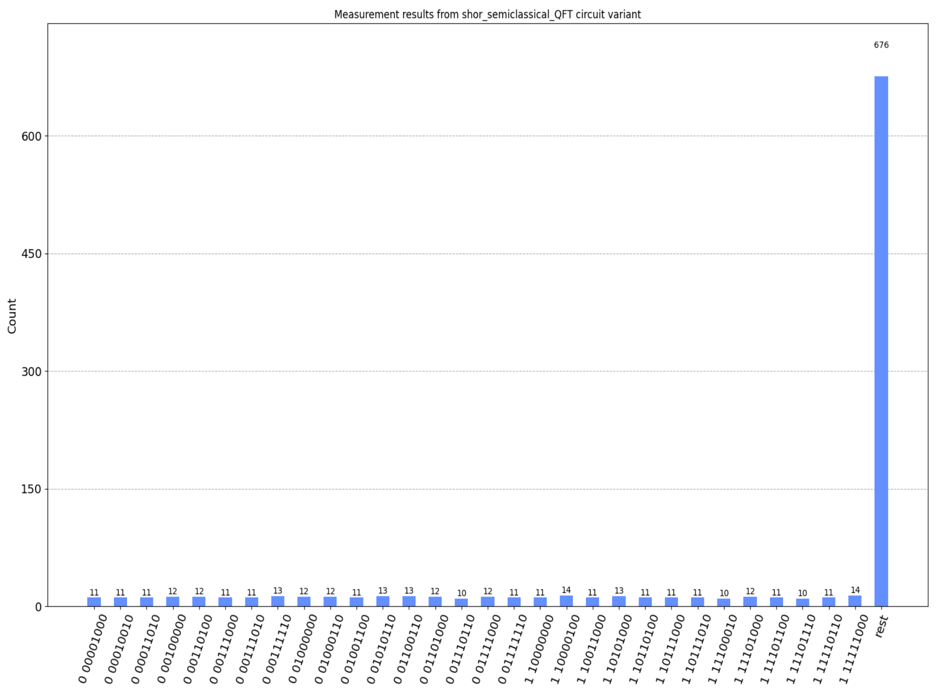


FIGURE 4.10: Measurement results of the **shor_semiclassical_QFT** variant for factoring N = 15

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 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.

Chapter 5

Results and Observations

5.1 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 and the results for counts and phases for 15 are presented before in Figure 4.4. The constant optimized circuit we used is different for every N and already precomputed. So let's us first discuss the Noise Model analysis made on the constant optimized normal circuit as this is the most optimized circuit(based on current resources) we have for number N, amongst all the variants we have developed, refer 5.1.1. Next, we proceed with analysing the execution time of the rest two implementation variants (4.2.5) in section 5.1.2. In section 5.1.3, we analyse the success rates between the two main variants shor_standard_QFT and shor_semiclassical_QFT and in section 5.1.4, 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.

5.1.1 Noise Model Results on shor_normal_constant

We conducted simulations for each distinct noise profile, testing all possible guesses of a (a = 2, 4, 7, 8, 11, 13) for the number 15, 100,000 each for all noise models. The average success rate and the sum of squares errors (SSE) were calculated for these runs. The outcomes are presented in Table 5.1 for the primary noise profiles and for individual error applications, Figure 5.1 for the primary noise profiles, and Figure 5.4, which illustrates the correlation between SSE and success rates across all noise profiles.

Success Rate Figure 5.1 displays average success rates for all guesses of a for the noiseless circuit, the emulated Guadalupe noise model, and the custom noise model with scaled error probabilities. The figure demonstrates a connection where lower error probability simulations with the custom noise model lead to higher success rates - which range from 65.804% with error probabilities scaled 25%, to 63.858% with 100% scaled errors. However, a noteworthy remark is that the success rate of the emulated Guadalupe noise model is closer to the success rate of the custom noise model with error probabilities scaled to 25% than it is to any of the other custom noise model simulations. This is more easily seen in Table 5.2, which displays the differential success rate between the custom noise models and the emulated Guadalupe noise model compared to the optimal run. Simulations of Shor's algorithm for each of the error sources alone, each with 25%, 50% and 100% of the baseline error rates

TABLE 5.1: Success rates of finding a factor without noise, with our custom noise model scaled to 25%, 50%, and 100% of its original error values, with the FakeGuadalupeV2 noise model, and with single errors applied only. Success rates and SSE display averages after runs with all guesses of a.

Noise model applied	**SSE**	**Success Rate %**
No noise	-	66.623%
Custom noise model, 25%	63.270	65.804%
Custom noise model, 50%	235.571	64.978%
Custom noise model, 100%	819.823	63.858%
Guadalupe noise model	1434.859	65.654%
1qb errors only, 25%	0.447	66.682%
1qb errors only, 50%	0.495	66.678%
1qb errors only, 100%	0.270	66.654%
2qb errors only, 25%	23.716	66.103%
2qb errors only, 50%	92.447	65.634%
2qb errors only, 100%	333.129	64.932%
RO errors only, 25%	10.363	66.224%
RO errors only, 50%	39.879	65.816%
RO errors only, 100%	152.275	65.031%
TR errors only, 25%	0.381	66.659%
TR errors only, 50%	0.388	66.642%
TR errors only, 100%	0.952	66.693%

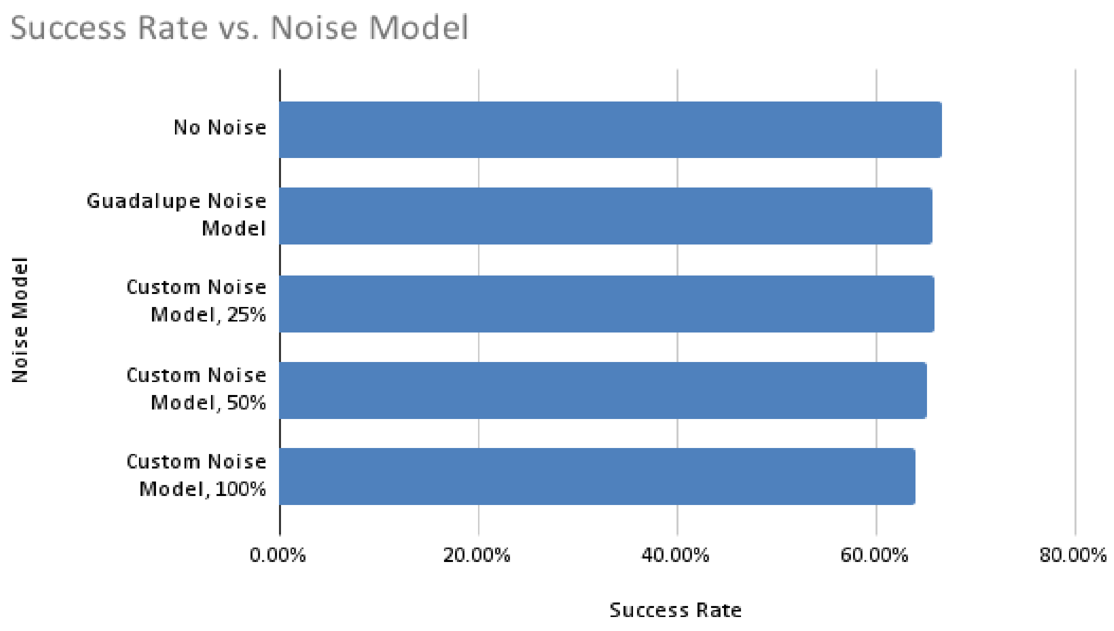


FIGURE 5.1: Success rates of finding a factor with noise, without noise, and with our custom noise model scaled to 25%, 50%, and 100% of its original value. Success rates and SSE display averages after runs with all guesses of a

were also run. All but the thermal relaxation error run show a higher success rate the lower the error rate. With the error applied to two-qubit gates and readout errors being the most pronounced. The same can be seen in the results for the single-qubit error runs but here the differences are much smaller. For the runs, applying only the thermal relaxation errors, the 25% and 50% follow the same trend but here the 100% error rate had the highest success rate. The results are displayed in Figure 5.2.

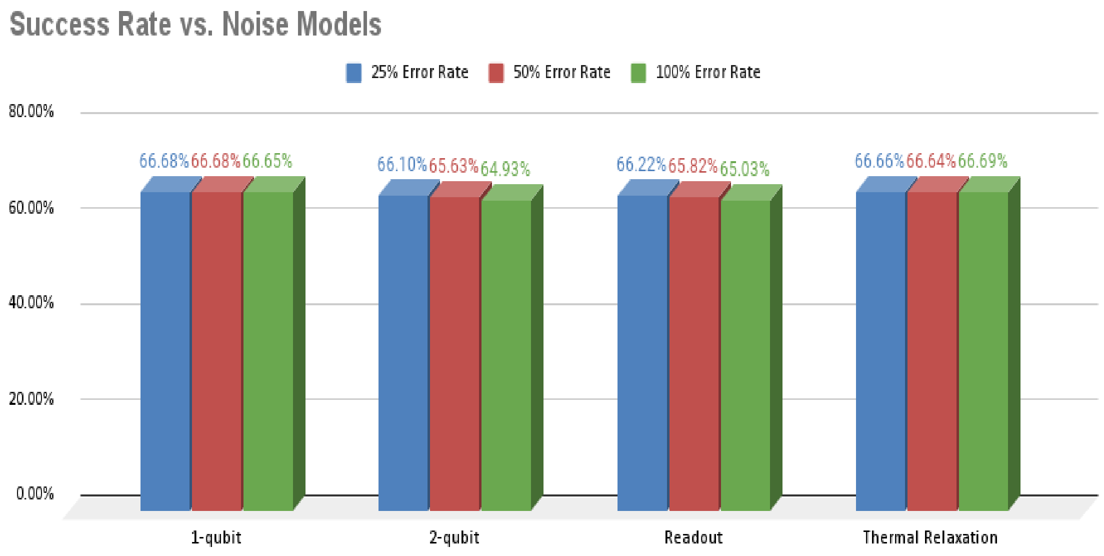


FIGURE 5.2: Success rates when running the circuit with noise models consisting of single noise sources and 25%, 50% & 100% error probabilities

TABLE 5.2: The success rate differentials of the complete noise model with varying error probabilities in comparison to the average success rate of a simulated optimal run (a run with no noise model applied). The average success rate of a optimal run was 66.62%.

Noise Model	**Success Rate Differential**
Guadalupe	-0.97%
Custom, 25% error probability	-0.82%
Custom, 50% error probability	-1.65%
Custom, 100% error probability	-2.76%

We also ran simulations with all different combinations of the different error sources with the same baseline error rates as the previous simulations. Just as with the single error source runs there is one outlier with the thermal relaxation error as part of the noise model used, but this time it is in combination with the one-qubit gate error source. All other results follow the pattern of: the lesser probability for noise induces error, the higher the success chance. One can also see (in bar-groupings 1, 3 and 7 from the right) that the greatest impact on the success chance was when both two-qubit gate errors and readout error are combined, in line with what could be seen in figure 5.2.

These results are also displayed in Table 5.3, where the differentials compared to the optimal runs are displayed for the custom single and combination runs.

Table 5.3: The average success rate differentials of single and combined error source noise models in comparison to the average success rate of a simulated optimal run. The average success rate of a optimal run was 66.62%.

Noise Model	**Success Rate Differential**
1-qubit, 2-qubit & Readout	-1.84%
2-qubit & Readout	-1.75%
2-qubit, Readout & Thermal relaxation	-1.71%
2-qubit	-1.06%
1-qubit, 2-qubit & Thermal relaxation	-1.02%
1-qubit & 2-qubit	-1.01%
2-qubit & Thermal relaxation	-1.00%
Readout	-0.93%
1-qubit & Readout	-0.87%
1-qubit, Readout & Thermal relaxation	-0.84%
Readout & Thermal relaxation	-0.82%
Thermal relaxation	0.04%
1-qubit & Thermal relaxation	0.05%
1-qubit	0.05%

SSE Figure 5.3 displays the correlation between the success rate and SSE for the average values obtained from multiple runs using all possible guesses of a and various noise profiles. The figure reveals a negative correlation, suggesting that a higher SSE value corresponds to a lower success rate. The success rate range between 63.5% to 65.6%, while the SSE values range from 0 (for a noiseless circuit) to 819,8. However, there is one exception. An outlier on the graph points to an SSE of 1434.9 and a success rate of 65.7%. This data point is associated with the average SSE results calculated for the emulated Guadalupe noise model. The cause for this deviation

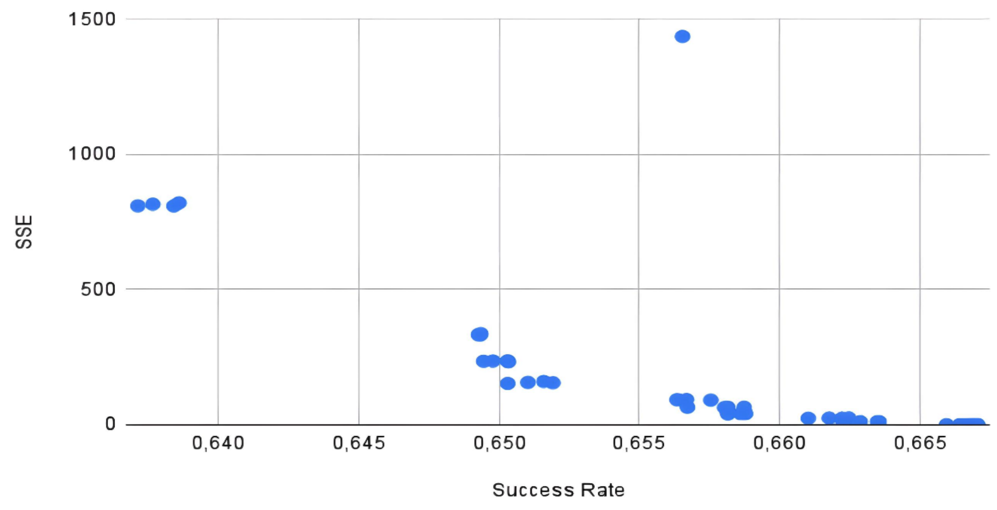


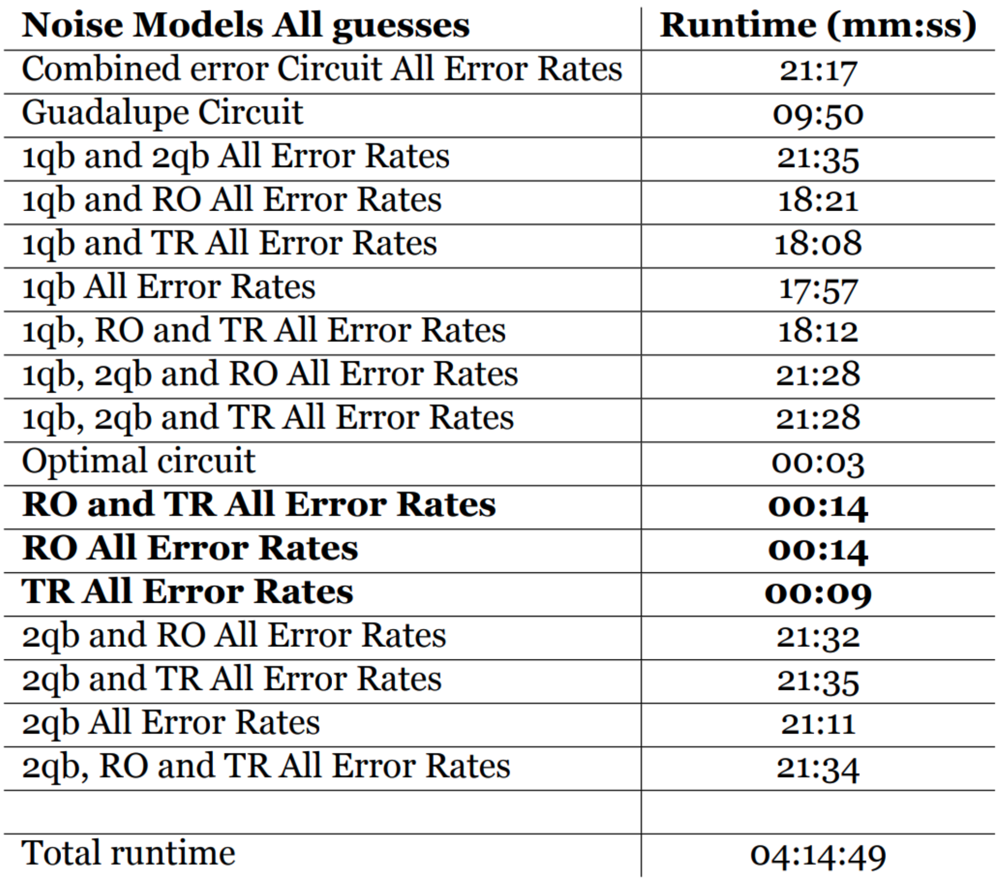
FIGURE 5.3: Scatter plot displaying correlation between success rates and sum of squares error for all noise profiles. Success rates and SSE display averages after runs with all guesses of a.

lies in the run with a = 11, which resulted in a success rate of 50.1% and an SSE of 5538.7 for the Guadalupe noise model. This significantly skews the average SSE for this model upwards

Runtime Table 5.4 display the runtimes when the noise sources were used in all various combinations for custom noise models. From the runtimes, for the simulations where these noise models were used, it becomes clear that the one-qubit and two qubit gate errors have a much greater effect on the performance when simulating Shor's algorithm. Even though the one and two-qubit gate errors have a greater effect on the simulation runtimes it can be seen in figure 5.4 that, when a standard deviation graph is created of all runtimes, a fairly symmetrical bell curve can be seen. The curve favours the right side since, in a combination of the four noise sources, the most influential two are part of a majority of the noise models used.

T-Test A t-test was conducted on the custom noise model and the FakeGuadalupeV2 model as a means to compare the two since the former is based on the latter.

Table 5.4: The runtimes for the various noise source combinations used to simulate Shor's algorithm quantum circuit. The much shorter runtimes, only affected by the readout and thermal relaxation errors, are highlighted in bold letters.



Noise Models All guesses	**Runtime (mm:ss)**
Combined error Circuit All Error Rates	21:17
Guadalupe Circuit	09:50
1qb and 2qb All Error Rates	21:35
1qb and RO All Error Rates	18:21
1qb and TR All Error Rates	18:08
1qb All Error Rates	17:57
1qb, RO and TR All Error Rates	18:12
1qb, 2qb and RO All Error Rates	21:28
1qb, 2qb and TR All Error Rates	21:28
Optimal circuit	00:03
RO and TR All Error Rates	**00:14**
RO All Error Rates	**00:14**
TR All Error Rates	**00:09**
2qb and RO All Error Rates	21:32
2qb and TR All Error Rates	21:35
2qb All Error Rates	21:11
2qb, RO and TR All Error Rates	21:34
Total runtime	04:14:49

As presented in Table 5.5: the calculated p-values do not indicate that the two have a different mean.

Table 5.5: P-values from t-test results when comparing success rates from circuit runs with the FakeGuadalupeV2 backend noise model and our custom noise model with different error rates. The calculation was performed as a two-tailed, two-sampled equal variance t-test.

Data Sample	**p-value**
Guadalupe and Custom noise model, 100%	0.8130124595
Guadalupe and Custom noise model, 50%	0.9293575653
Guadalupe and Custom noise model, 25%	0.9842521732

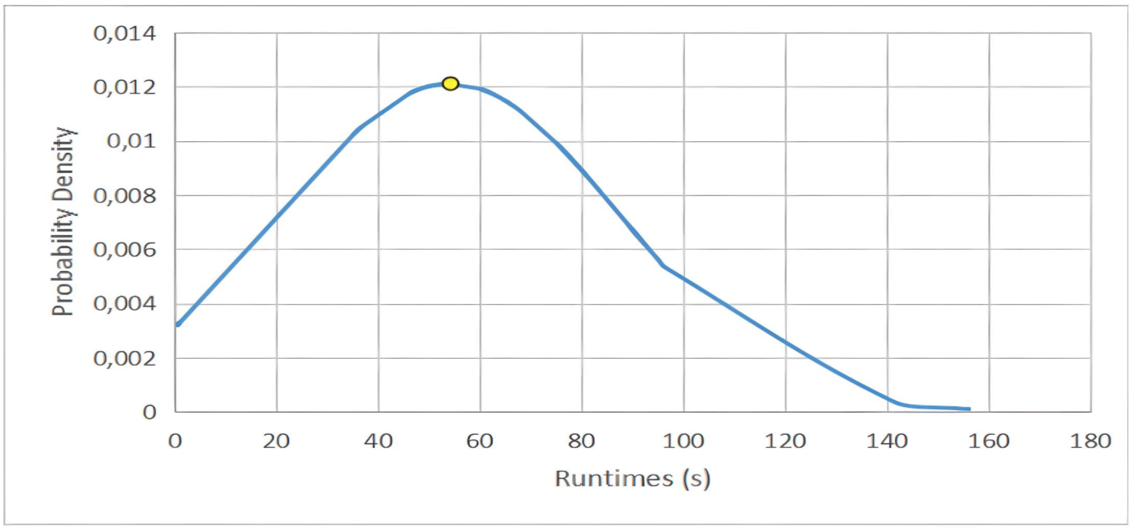


FIGURE 5.4: Standard deviation graph of all runtimes for each simulation run. The yellow dot display the runtime mean.

5.1.2 Execution time analysis

In figure 5.5, 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. We see that in the Figure 5.5, 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. 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

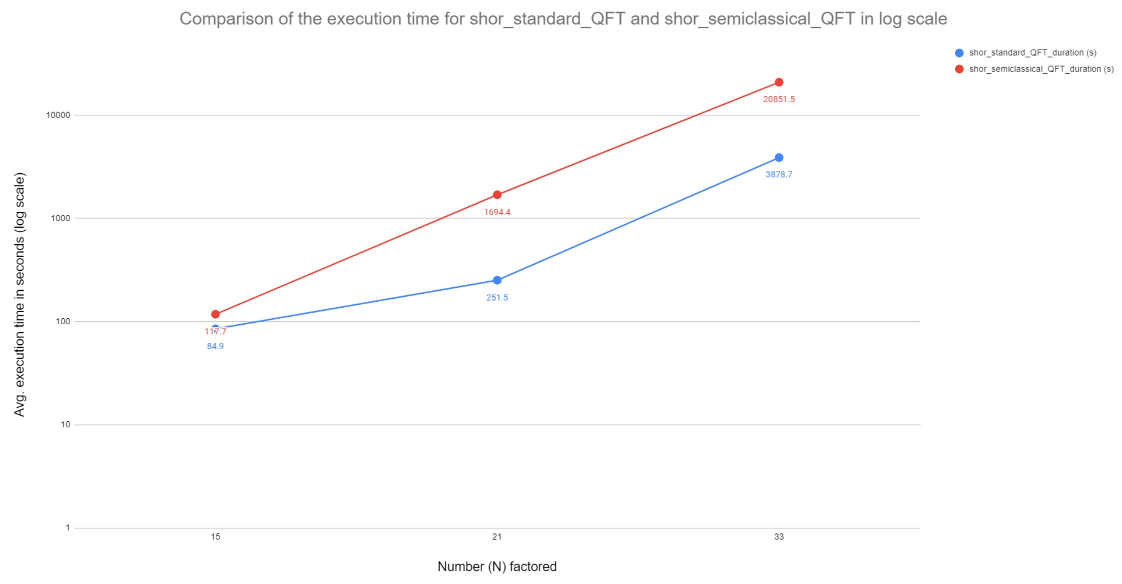


Figure 5.5: Comparison of the execution time for shor_standard_QFT and shor_semiclassical_QFT in log scale

in section 5.1.4.

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 5.6 and 5.7 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 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 respectively.

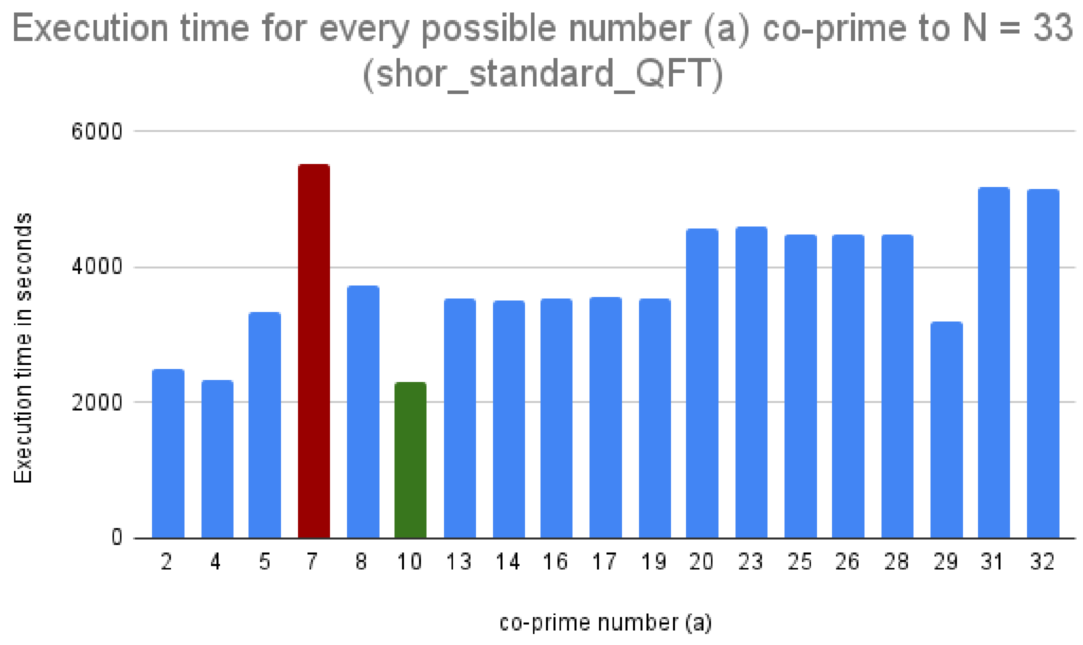


Figure 5.6: Execution time for every possible number (a) co-prime to $N = 33$ (shor_standard_QFT)

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.

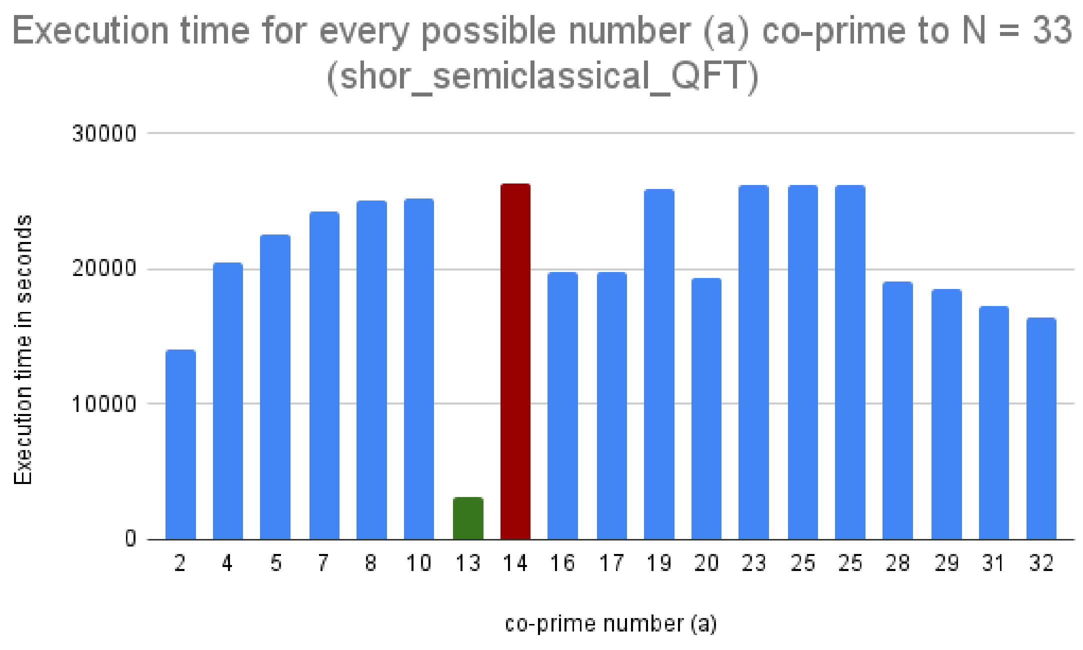
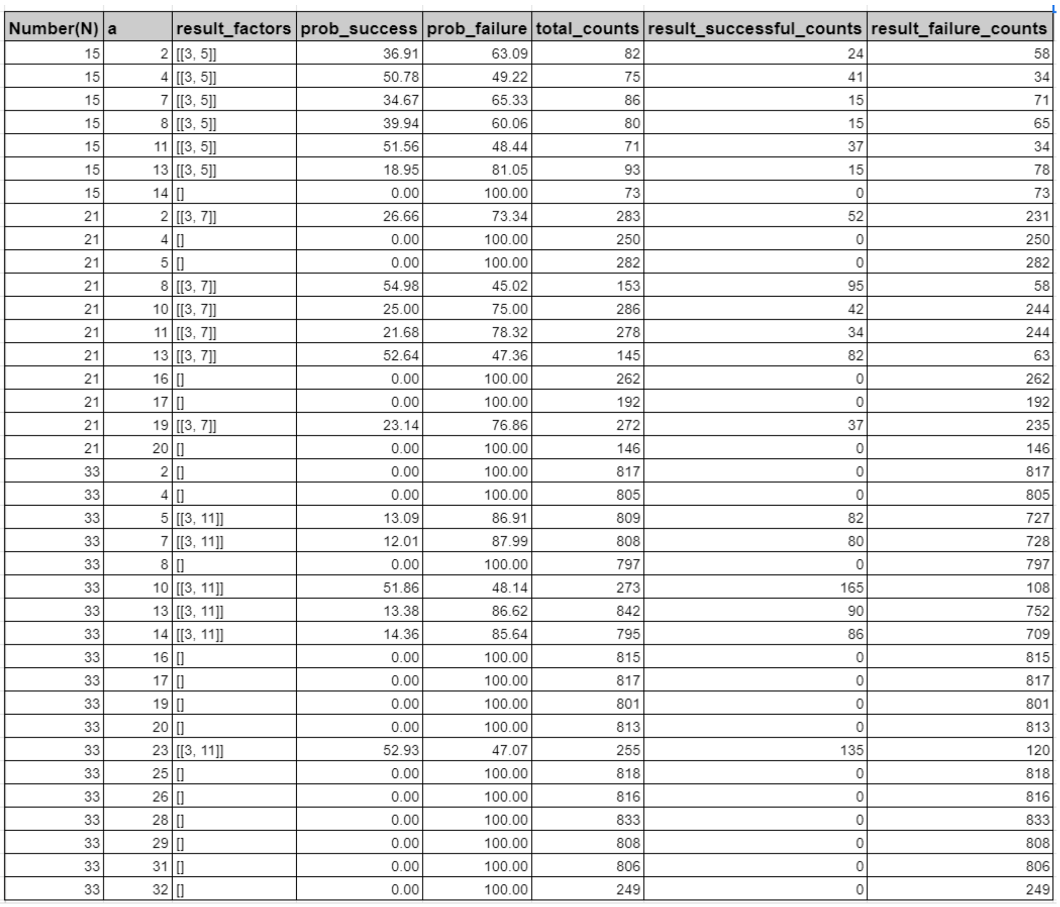


FIGURE 5.7: Execution time for every possible number (a) co-prime to $N = 33$ (shor_semiclassical_QFT)

5.1.3 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 (6), the algorithm returns the correct period r only with some probability.

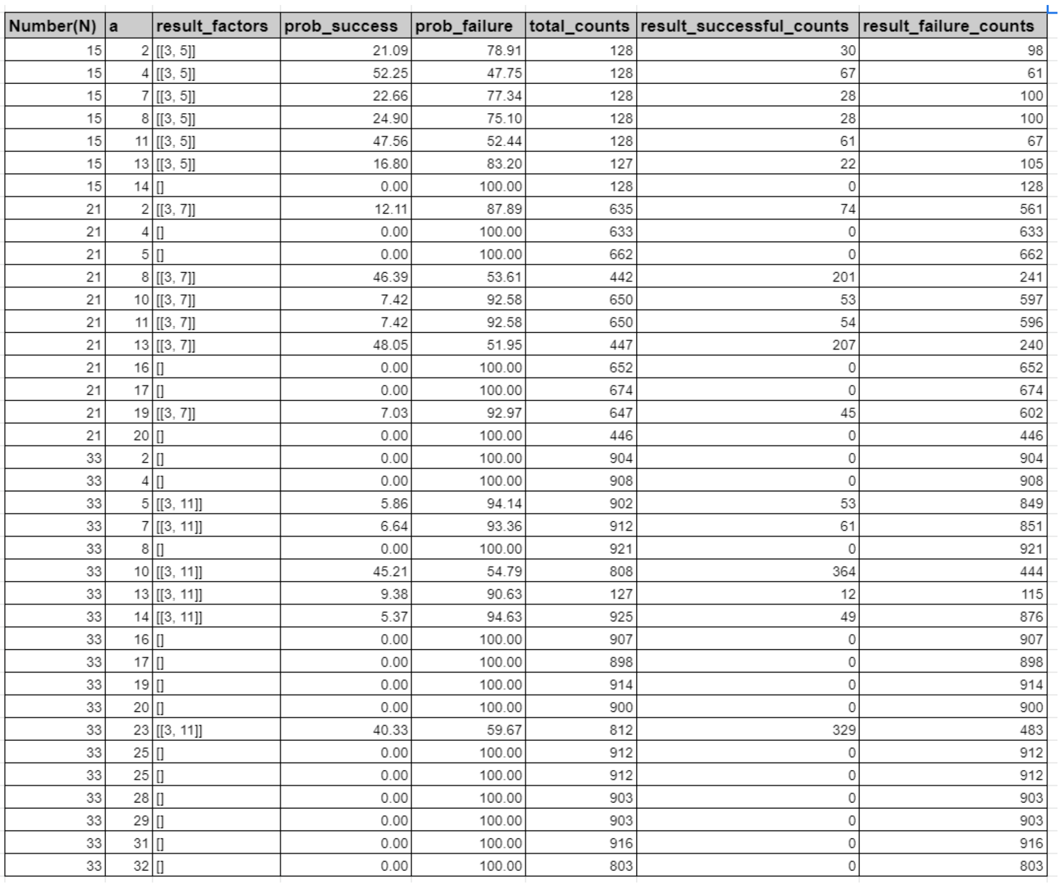
The standard approach that a candidate (measurement outcome) for an order yield by the classical post-processing phase is to test whether it is the correct order or not. If this fails, then the quantum order finding algorithm has to be executed again. In table 5.8 and 5.9, we show the success rates for 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



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

$N = 15$, $a = 8$ 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 (28) and on average ranges between 20% to 40%. In addition as shown in figure 5.10, 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



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 5.9: Success rate for numbers $N = 15$, 21 and 35 considered in the shor_semiclassical_QFT variant

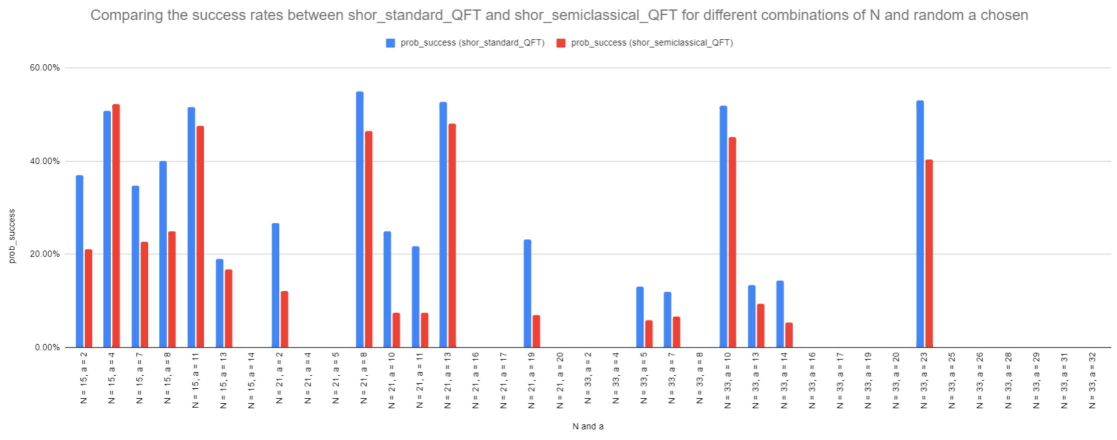


FIGURE 5.10: Comparing the success rates between shor_standard_QFT and shor_semiclassical_QFT for different combinations of N and random a chosen

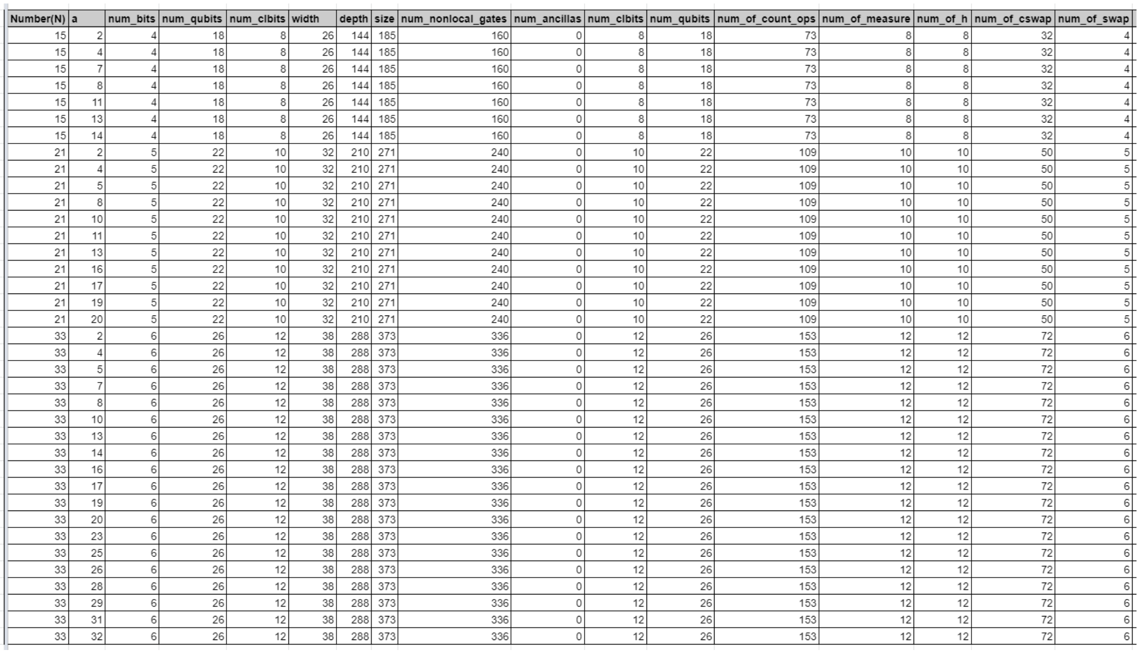
shor_semiclassical_QFT variant, we observe little significant impact on the success rate of the computation if either of the two variants is chosen. Also, the overall lower success rate in the shor_semiclassical_QFT variant may be explained as the result of many QASM instructions and measurements which with high probability introduces more gate and measurement errors even though it uses less qubits.

5.1.4 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 (14):

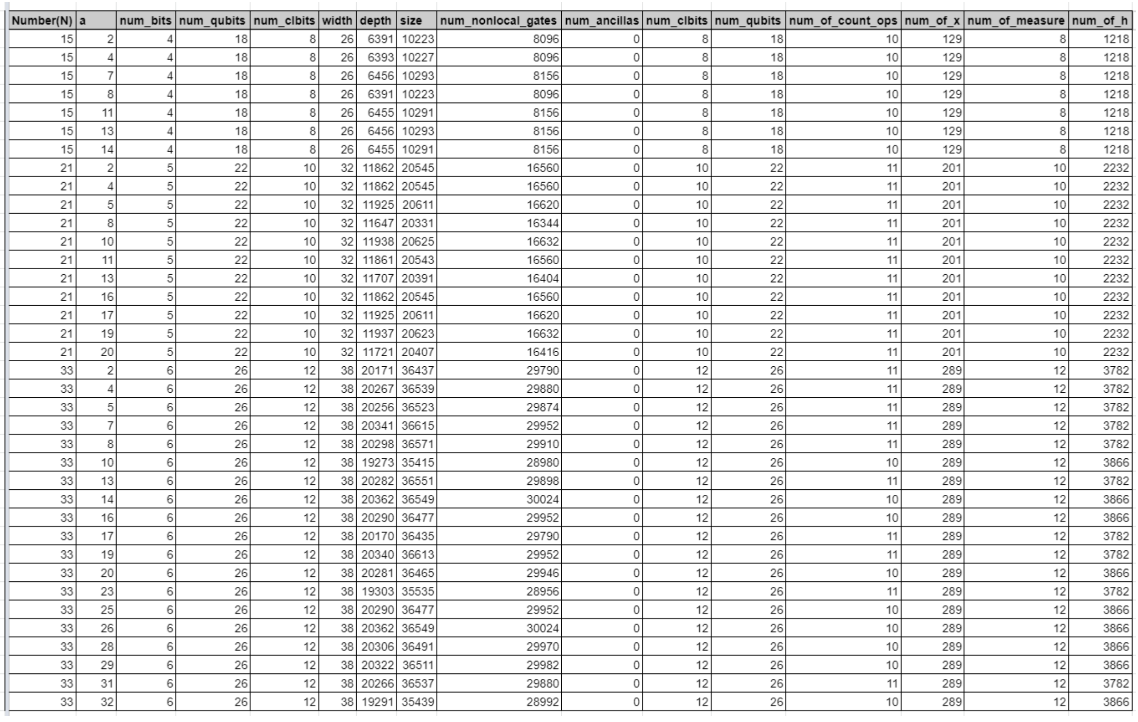
1. Virtual circuit optimization
2. Qubit gate decomposition for all 3+ multi qubit gates
3. Placement on physical qubits
4. Routing on restricted topology
5. Translation to basis gates
6. Physical circuit optimization

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 (15):



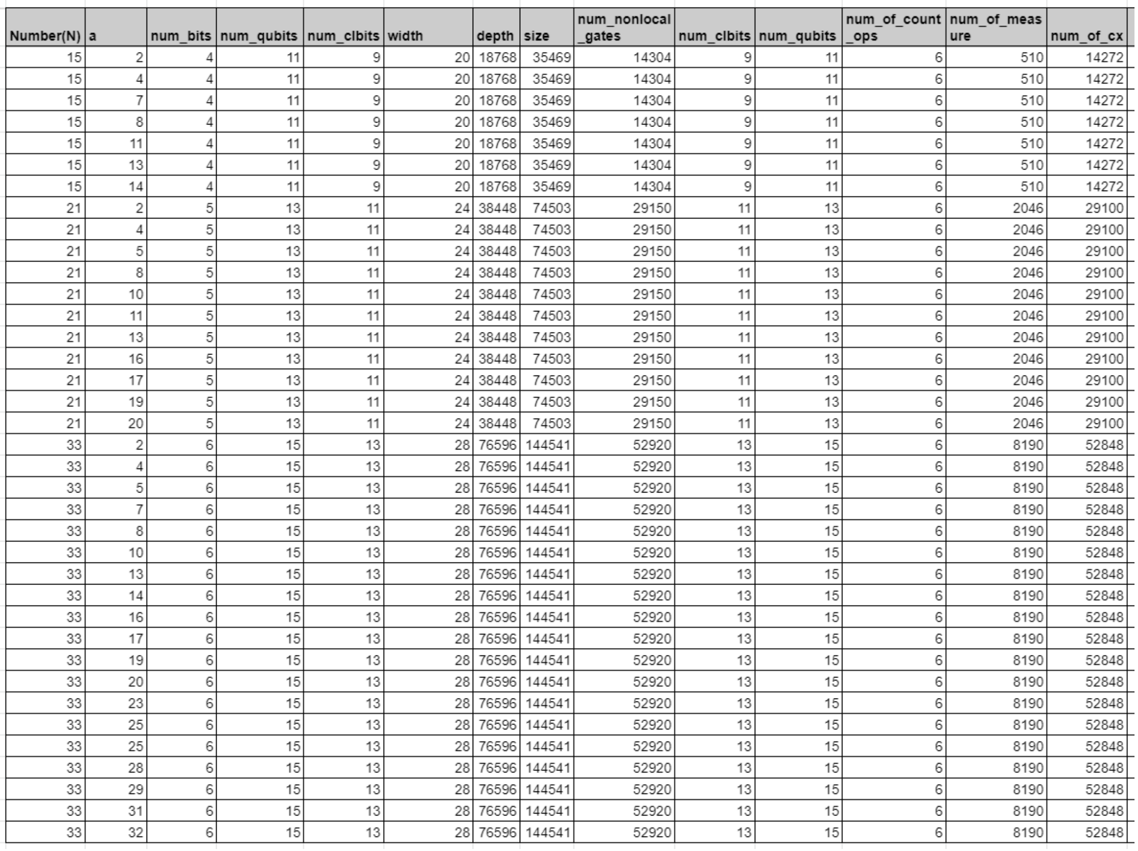
Number(N)	a	num_bits	num_qubits	num_clbits	width	depth	size	num_nonlocal_gates	num_ancillas	num_clbits	num_qubits	num_of_count_ops	num_of_measure	num_of_h	num_of_cswap	num_of_swap
15	2	4	18	8	26	144	185	160	0	8	18	73	8	8	32	4
15	4	4	18	8	26	144	185	160	0	8	18	73	8	8	32	4
15	7	4	18	8	26	144	185	160	0	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 5.11: Summary metrics for the shor_standard_QFT decomposed circuit



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

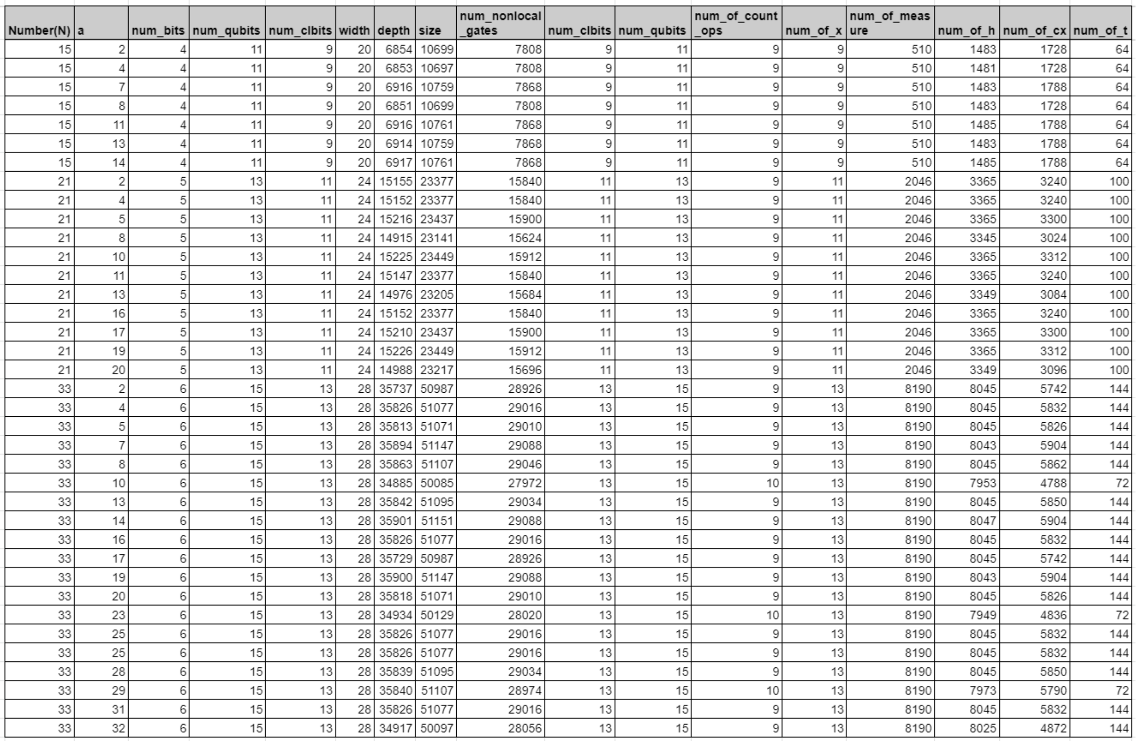
FIGURE 5.12: Summary metrics for the shor_standard_QFT transpiled circuit



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_meas ure	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 5.13: Summary metrics for the shor_semiclassical_QFT decomposed circuit

1. Number of binary digits (num_bits): Number of bits of the number N
2. Number of qubits (num_qubits)
3. Number of classical bits (num_clbits)
4. Number of qubits plus classical bits in the circuit (width)
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)



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 5.14: Summary metrics for the shor_semiclassical_QFT transpiled circuit

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 5.11, 5.12, 5.13 and 5.14 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.

Chapter 6

Conclusions

We have simulated three implementation variants of the Shor's Factoring algorithm using the QASM simulator offered on IBM Quantum Experience Platform. First being the shor_normal_constant which utilizes the fact of knowing prior information about the number and based on a and N does a constant optimized modular exponentiation. We have done the **Noise Model** Analysis on shor_normal_constant as for a given N and a, it would naturally be the fastest variant to compute the factors. Comparing the custom noise model at different error probabilities and the noise model emulated from IBM Guadalupe, the t-test results in Table 5.1 show that there is no significant difference between the success rates obtained from the circuit runs. The p-values are all above 0.05, indicating that the null hypothesis cannot be rejected. Looking at Figure 5.1 and Figure 5.4 the results display a tendency of higher success rates running Shor's algorithm when errors decrease. This indicates that the results are in line with previous studies performed on noise effects regarding Shor's algorithm. However, even though the custom noise model did display higher success rates when error probabilities decreased, a 50% decrease in the original error rates only improved the success rate by about 1%, and a 75% decrease in original error rates only improved the success rate by about 2%. This would indicate a positive linear effect on performance when noise decreases, but improvements are relatively small as the error probabilities continue declining. Focusing on SSE in Table 5.1 the custom noise model does show significant improvements as error probabilities decline. This effect may indicate that lower error probabilities do have a greater effect as the size of the integer being factored increases, since larger numbers being factored would require larger circuits and more qubits, causing greater discrepancies in phases acquired from the QFT. The runtimes, shown in table **??**, give very interesting insights at least when it comes to simulating quantum circuits. It can, in our measured runtimes, clearly be seen that simulating one and two-qubit error channels carry with it a much higher performance overhead than readout and thermal relaxation errors. Whereas this would have a one-to-one or similar overhead were this specific type of circuit to be run on a real quantum computer and not a simulation, is outside the scope of this study. Setting aside these results, several limitations were placed in our research, which 32 indicate that conclusions from the study should be considered with a degree of caution.

First of all, our implementation of Shor's algorithm only handled the case of factoring the number 15 with 8 counting qubits and 4 ancillary qubits, which is a small problem size. Larger problem sizes with more qubits and complex circuits might exhibit different behaviour in terms of noise sensitivity and success rate improvements. It is therefore important to note that our findings might not apply to larger problem sizes and configurations of Shor's algorithm.

Secondly, the custom noise model employed in our research may not be representative of the possible noise types and sources present in real-world quantum systems, even though it did provide some insight into the effects of noise on the algorithm's performance. We did try to emulate noise levels from IBM Guadalupe by using median error probability readouts, but these errors and error probabilities were specifically applied to the gates in our implementation without further consideration of what errors usually affect which gates, or the difference in error rates on certain qubits. An interesting observation running variations of errors for the custom noise model was the difference between depolarizing errors applied on single-qubit gates and depolarizing errors applied on two-qubit gates, displayed in Figure 5.2. The application of single-qubit errors had a negligible impact on success rates, whereas two-qubit errors displayed nearly the same differences in error rates between error probability scaling as readout errors. This is attributed to the fact that two-qubit errors were applied to our swap gates and controlled phase gates, which are more critical in computing Shor's algorithm correctly. In combination with what the runtimes showed: even greater importance could be attributed to the two-qubit error.

Even though both the readout and one-qubit errors have a big impact on running a Shor's algorithm circuit in terms of success rate and performance respectively. While this finding may be obvious, it does highlight the importance of understanding where noise should be minimised to maximise impact when using quantum computers. Similar findings were also noted in papers like "Investigation of Noise Effects for Different Quantum Computing Architectures in IBM-Q at NISQ Level" (20).

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. 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.

Bibliography

[1] Arute, Frank et al. *Quantum supremacy using a programmable superconducting processor.* In: *Nature* 574.7779 (2019), pp. 505–510.

[2] DeCusatis, Casimer and Mcgettrick, Emily. Near term implementation of Shor's Algorithm using Qiskit. In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC).* IEEE, 2021, pp. 1564–1568.

[3] Georgopoulos, Konstantinos, Emary, Clive, and Zuliani, Paolo. Modeling and simulating the noisy behavior of near-term quantum computers. In: *Physical Review A* 104.6 (2021), p. 062432.

[4] Hellström, Ian. *Quantum Computer Roadmaps.* Sept. 2022. Retrieved from `https://databaseline.tech/quantum.html` (visited on 03/17/2023).

[5] Hidary, Jack D. *Quantum Computing: An Applied Approach.* 1st ed. Springer, 2019. ISBN: 978-3-030-23922-0.

[6] 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)

[7] Johnston, E.R., Harrigan, N., and Gimeno-Segovia, M. *Programming Quantum Computers: Essential Algorithms and Code Samples.* O'Reilly Media, Incorporated, 2019. ISBN: 9781492039686. Retrieved from `https://books.google.se/books?id=LZY1vgEACAAJ`.

[8] Johnstun, Scott and Van Huele, Jean-François. Understanding and compensating for noise on IBM quantum computers. In: *American Journal of Physics* 89.10 (2021), pp. 935–942.

[9] Nielsen, Michael A. and Chuang, Isaac L. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* 10th ed. USA: Cambridge University Press, 2011. ISBN: 1107002176.

[10] Parker, S and Plenio, MB. Entanglement simulations of Shor's algorithm. In: *Journal of Modern Optics* 49.8 (2002), pp. 1325–1353.

[11] Preskill, John. Quantum computing in the NISQ era and beyond. In: *Quantum* 2 (2018), p. 79.

[12] Quantum, IBM. IBM Quantum. Retrieved from `https://quantum-computing.ibm.com/services/resources?tab=systems&skip=10&system=ibmq_guadalupe` (Accessed on April 27, 2023).

[13] Team, Qiskit Development. Qiskit Documentation. Retrieved from `https://qiskit.org/documentation/` (Accessed on April 27, 2023).

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

[15] IBM Quantum Circuits documentation: https://qiskit.org/documentation/stubs/qiskit.circui

[16] Wikimedia Commons, Smite-Meister. Bloch sphere. URL: `https://commons.wikimedia.org/wiki/File:Bloch_sphere.svg` (Accessed on April 27, 2023).

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

[18] 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)

[19] Georgopoulos, Konstantinos, Emary, Clive, and Zuliani, Paolo. Modeling and simulating the noisy behavior of near-term quantum computers. In: *Physical Review A* 104.6 (2021), p. 062432.

[20] Yetis, Hasan and Karakoes, Mehmet. Investigation of Noise Effects for Different Quantum Computing Architectures in IBM-Q at NISQ Level. In: *2021 25th International Conference on Information Technology (IT)*. Feb. 2021, pp. 1–4. DOI: 10.1109/IT51528.2021.9390130.

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

[22] 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)

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

[24] 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)

[25] Parker, S and Plenio, MB. Entanglement simulations of Shor's algorithm. In: *Journal of Modern Optics* 49.8 (2002), pp. 1325–1353.

[26] 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)

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

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

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

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

[31] 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)

[32] 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]`

[33] Shor, Peter W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. In: *SIAM review* 41.2 (1999), pp. 303–332.

[34] 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`

[35] 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.