INTEGER FACTORIZATION USING SHOR'S ALGORITHM AND ITS IMPLEMENTATION ON A QUANTUM COMPUTER

A Project Work Submitted to

DEPARTMENT OF PHYSICS
Khwopa College
(affiliated to Tribhuvan University)
Dekocha-6, Bhaktapur

In the Partial Fulfillment of the Requirements for the Award of
Degree of
B.Sc in Physics

By
Aman Ganeju
T.U. Exam Roll No: 4080023
T.U. Registration No: 5-2-408-273-2016

February, 2021

DECLARATION

I hereby declare that the work presented in this dissertation is a genuine work done originally by me and has not been submitted anywhere for the award of any degree. All the sources of information have been specifically acknowledged by reference to the author(s) or institution(s).

.....

Aman Ganeju

Date:

An Undertaking of Bhaktapur Municipality

KHWOPA COLLEGE

(Affiliated to Tribhuvan University)

Dekocha-6, Bhaktapur, Nepal

Date:.....

RECOMMENDATION

This is to certify that Mr.**Aman Ganeju** has completed this project work work entitled "**INTEGER FACTORIZATION USING SHOR'S ALGORITHM AND ITS IMPLEMENTATION ON A QUANTUM COMPUTER**" as a partial fulfillment of requirements of B.Sc. in Physics under our supervision and guidance. To our knowledge, this research has not been submitted for any other degree, anywhere else.

We, therefore, recommend this project work for acceptance and approval.

.....
Om Suwal, PhD.
Supervisor
Professor
Kathmandu University
Dhulikhel, Nepal

.....
Dibakar Sigdel, PhD.
Co-supervisor
Data Scientist
University of California,
Los Angeles, USA.

.....
Shree Krishna Bhattarai, PhD.
Co-supervisor
Department of Physics
University of North Carolina
Charlotte, USA.

An Undertaking of Bhaktapur Municipality
KHWOPA COLLEGE
(Affiliated to Tribhuvan University)
Dekocha-6, Bhaktapur, Nepal

Date:.....

LETTER OF APPROVAL

On the recommendation of supervisor **Om Suwal, PhD.** and **Dibakar Sigdel, PhD.**, this project work submitted by Mr.**Aman Ganeju** entitled "**INTEGER FACTORIZATION USING SHOR'S ALGORITHM AND ITS IMPLEMENTATION ON A QUANTUM COMPUTER**" has been approved for examination and submitted to the Tribhuvan University in partial fulfillment of the requirements of B.Sc. in Physics.

.....
Mira Prajapati
Incharge
Department of Physics and Environment Science
Khwopa College

An Undertaking of Bhaktapur Municipality
KHWOPA COLLEGE
(Affiliated to Tribhuvan University)
Dekocha-6, Bhaktapur, Nepal

Date:. . . . . .

CERTIFICATE OF ACCEPTANCE

This dissertation entitled "**INTEGER FACTORIZATION USING SHOR'S ALGORITHM AND ITS IMPLEMENTATION ON A QUANTUM COMPUTER**"submitted by Mr **Aman Ganeju** has been examined and accepted as a partial fulfillment of the requirements of B.Sc. in Physics.

Evaluation Committee

.....
Supervisor
Om Suwal, PhD.
Professor
Kathmandu University,
Dhulikhel, Nepal

.....
External Supervisor

Professor

.....
Incharge
Mira Prajapati
Department of Physics and Environment Science
Khwopa College

ACKNOWLEDGEMENT

This project would not have been possible without the help and cooperation of many. I would like to thank the people who helped me directly and indirectly in the completion of this project work.

First I would like to express my gratitude to my supervisors **Om Suwal, PhD. Dibakar Sigdel, PhD.**, and Shree Krishna Bhattarai, PhD.for providing their kind support, excellent guidance, insightful questions and timely support throughout the time of research and writing of project.

I would like to express my gratitude to the Mira Prajapati, Incharge, Department of Physics and Environmental Science, Khwopa College for providing her kind support and suggestions.

I would also like to thank my classmates: Rodip Datheputhe, Supriya Dhakal, Yousha Thapa Magar, Anis Gautam, and my brother Sagar Ganeju for their steadfast and strong support and engagement with this project.

Aman Ganeju

Date:

Abstract

Classically factoring a large number is known for having large complexity and a popular cryptosystem: RSA cryptosystem is based on the difficulty of the problem. However, Shor's algorithm introduces an efficient way for integer factorization using quantum computation performing the task in polynomial time with bounded error. It reduces the problem of factorization to period finding problem and uses the advantages of Quantum Modular Exponentiation and Quantum Fourier Transform to increase its efficiency.

This paper introduces the basics of quantum computing, number theory, complexity theory and theoretical formulation of Shor's algorithm that transform this NP(non-polynomial) class problem to BQP(Bounded-error quantum polynomial) class problem. In this paper, we implement an example of Shor's algorithm on quantum computer at IBM quantum to factor an integer 15 to its factors.

Keywords : Factorization, quantum, complexity, RSA,

TABLE OF CONTENTS

List of Figures

List of Tables

Acronyms

BQP Bounded Quantum Polynomial. 28, 55

FFT Fast Fourier Transform. x, 14, 17, 81, 83

MEF Modular exponentiation function. 20, 22, 36–38, 61

QC Quantum computer. 4, 5, 8–12, 17

QFT Quantum Fourier Transform. ix, 17

QM Quantum Mechanics. 3

Chapter1: Introduction

1.1 Introduction

In "*Disquisitiones Arithmeticae*", Gauss stated, "Any composite number can be resolved into prime factors in only one way" (Shapiro, 2008, p.44) [27]. That is, a composite number can be factored into a set of unique prime integers. Factorization of composite integers into their primes is considered as an interesting mathematical problem and has been an active area of research.[7] Although no classical algorithm has been developed to complete the task efficiently. A secure cryptographic technique called RSA(Rivest-Shamir-Adleman) cryptosystem was developed using the advantage of the inefficiency in solving this problem. RSA cryptosystem is used in securing our banking transactions, emails, and internet access communications.

We categorize a problem's efficiency by its complexity or the amount of resources consumed to solve the problem. The problem of integer factorization is considered NP(non-polynomial) problem classically. That is, it takes non-polynomial resources(time or space) to solve the problem. The best classical algorithm used for integer factorization called General Number Field Sieve has sub-exponential complexity.

In this study, we discuss an algorithm for integer factorization called Shor's algorithm. It was developed by Peter Shor in 1994.[28] Shor's algorithm is a quantum algorithm, that is, it runs on a quantum computer which is inherently different from classical computers. The working mechanism of QCs is based on quantum mechanical properties like entanglement and superposition. This gives QCs a certain kind of advantages over classical ones. Shor's algorithm has polynomial time complexity with bounded error. That is, it performs better than the best classical algorithm for the task.

Shor's algorithm takes an odd composite integer and factorizes it into its factors. It reduces the factorization problem to period finding problem. Use of quantum

modular exponentiation and quantum Fourier transform cause exponential speed up in period finding.[31] And finally, with the use of number theory principles, the obtained period can be used to determine a factor with some bounded probability.

The first experimental implementation of Shor's algorithm on a quantum computer was performed by a group at IBM in 2001 using nuclear magnetic resonance to factor an integer 15. [32] The largest integer ever factored using Shor's algorithm is 21, performed in 2012 by scientists at university of Birstol.[21]

Chapter2: Background

2.1 Quantum Mechanics

Quantum Mechanics(QM) was formulated to overcome some important limitations in classical mechanics; the inability of Newtonian mechanics to explain radiation properly and to make an appropriate prediction of phenomena in a microscopic scale.[33] QM is the best known theoretical foundation for describing our known universe. In QM, a Hilbert space[1] called state space is associated with any physical system and is denoted by its state vector $|\Psi\rangle$ [2] [24](Neilsen and Cheung,2005). The state vector $\Psi(x, y, z, t)$, a function of space and time, gives all the information about the system. Although, Ψ alone does not have much meaning by itself. The only quantity having physical meaning is the square of its magnitude $P = |\ \Psi\ |^2 = \langle\Psi|\Psi\rangle$, which gives the probability density of the state. [22](Murugeshan, 2015). Here every dynamical variable in classical mechanics or observable like position, momentum, energy, is an operator. Each operator is unitary matrices which on acting upon state vector $|\Psi\rangle$ gives the possible result as their eigenvalues. Since the observable are operators in QM, the act of observation affects the state of the system. Operation by an arbitrary operator A is denoted by $\langle\Psi|A|\Psi\rangle$[3].

2.2 Quantum Computing

Feynman, in 1981, talked about a possibility of a computer that will do exact simulation same as by nature.[12] It has been theoretically and experimentally proven that nature follows the rules of Quantum mechanics at microscopic levels. He added an example of polarization of photon(a two-state system) to make his point that "quantum mechanics can't seem to be imitable by a local classical computer".[12](Feynman1999, 485) Classical computer uses classical physics which are

[1] Hilbert space is an infinite-dimensional inner product vector space. We use a finite-dimensional Hilbert space for our study

[2] $|.\rangle$ is a Dirac's ket

[3] $\langle\Psi|A|\Psi\rangle$ means the inner product of $|\Psi\rangle$ and $A\,|\Psi\rangle$

deterministic in nature while quantum mechanics is probabilistic and is reversible. Another significant issue for classical mechanics is that it uses Boolean algebra, which states distributive property. But quantum mechanics does not necessarily follow the property[3]. "A Quantum computer is a device that leverages certain properties described by quantum mechanics to perform computation." [17] (Hidary, 2019, 3) Quantum computers(QC) uses properties like entanglement, superposition, and reversibility.

Different algorithms have been designed in the field whose equivalence in classical computing performs rather slow and lacks efficiency in performance. Shor's prime number factoring algorithm, Grover's search algorithm are some notable examples. However, no efficient and functional quantum computer has been built yet. Physicists and engineers have been using different architectures to realize a working efficient QC. Superconducting qubits, trapped ion qubits, Nuclear magnetic resonance, silicon-based spin qubit are some physical systems that currently realize quantum computation, on a small scale.[17] Regardless of that, studying and discussing quantum algorithms does not necessarily require a workable quantum computer and not even detailed knowledge about the physics of quantum computers.[5]

The first experimental implementation of Shor's algorithm on a quantum computer was performed by a group at IBM in 2001 using nuclear magnetic resonance to factor an integer 15. [32] The largest integer ever factored using Shor's algorithm is 21, performed in 2012 by scientists at the University of Bristol.[21] Classical computers use bits as the basic unit of information. A bit takes one of the two states: 0 or 1. Qubit is similar to the classical bit, however, the main significant difference is that qubits may be in the superposition of both the states $|0\rangle$ and $|1\rangle$. A system described by state vector $|\Psi\rangle$ is said to be in superposition of state $|0\rangle$ and $|1\rangle$ if

$|\Psi\rangle$ is the linear combination of $|0\rangle$ and $|1\rangle$, written as

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

where α and β are complex coefficients called amplitudes of each states following normalization rule: $|\alpha|^2 + |\beta|^2 = 1$. "A qubit can exist in a continuum of states between $|0\rangle$ and $|1\rangle$ - until it is observed."(Neilsen and Chuang, 2002, pg. 3)[24] As stated earlier: the act of observation affects the state of the system, measurement of the state vector $|\Psi\rangle$ collapse superposition of the state to one of the two basis states $|0\rangle$ or $|1\rangle$. The square of their amplitude gives the probability of obtaining each state upon measurement. However, direct measurement does not give any information about α or β. They are obtained by measuring the quantum state of the state vector and other similar state vectors resembling it. The accuracy of measuring α and β is based on the number of measurements made or numbers of similar quantum states measured.

A single qubit represented as $|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$(also written as $\begin{pmatrix}\alpha\\ \beta\end{pmatrix}$) is a two dimensional unit vector in a complex vector space $\mathbb{C}^2$ spanned by basis vectors $|0\rangle = \begin{pmatrix}1\\ 0\end{pmatrix}$ and $|1\rangle = \begin{pmatrix}0\\ 1\end{pmatrix}$ and a two qubit system would be represented by a four dimensional unit vector spanned by basis states $|00\rangle$, $|01\rangle$,$|10\rangle$ and $|11\rangle$. Similarly, a n qubit system would be spanned by 2^n different basis states from 0 to $2^n - 1$ binary equivalent, written as

$$|\Psi\rangle = \sum_{m=0}^{2^n-1} a_m\,|m\rangle \tag{2.2}$$

, where $|m\rangle = |x_1x_2...x_{m-1}x_m\rangle$ has the binary representation such that $m = 2^{m-1}x_1 + 2^{m-2}x_2 + ... + 2^1x_{m-1} + 2^0x_m$, $x_i \in \{0, 1\}$. So a n qubit system represents and manipulates 2^n classical equivalent states. This gives a slight idea about the supremacy QC has over classical computers. A single qubit lie in a $\mathbb{C}^2$ space and similarly, a n qubit system lies in a space with the dimension equal to the tensor

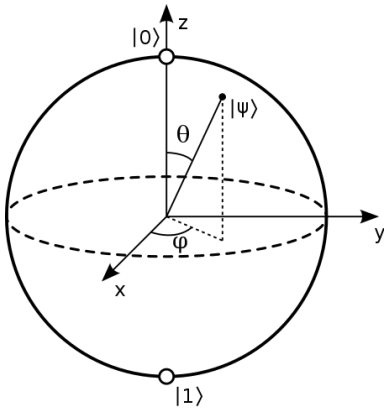


Figure 2.1: Qubit representation in a Bloch sphere

product of each qubits,

$$\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes ... \otimes \mathbb{C}^2 = (\mathbb{C}^2)^{\otimes n} = \mathbb{C}^{2^n} \tag{2.3}$$

.[5] Hence, each state $|m\rangle$ can be written as the tensor product: $|m\rangle = |x_1\rangle \otimes |x_2\rangle \otimes ... |x_n\rangle$.

Another useful and intuitive way of representing a state of the qubit is by using state vector in the Bloch sphere. The state vector is given by equation:

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

where θ, ϕ and γ are real numbers. Different values of θ and ϕ sweeps each point of the sphere. Each point on the Bloch sphere can be a possible state of a qubit. Most of the operations on qubits can be visualized or interpreted as the rotation of the state vector in the Bloch sphere.

Any two-level quantum system can be realized as a physical qubit, for instance, the ground state and excited state of an atom or two spin states of a 1/2 spin electron or horizontal and vertical polarization of the photon.[10] But qubits are delicate things; they lose their coherence rapidly and their state is affected by small interaction. This is a reason why a general-purpose quantum computer is hard and challenging to build. Associated with physical qubits are two components encoding information: magnitude and relative phase, together resulting to give the amplitude

of each state in the system.[18] The probability of measuring each state during the measurement is determined by the magnitude of each state. The difference in the phase of wave-function associated with the superimposed physical qubits gives the phase of the qubits. The angle ϕ in equation(2.4) gives the phase of the qubit. We can see that the phase of the qubit does not affect the probability of measuring each state, since the phase factor($e^{i\phi}$) has a norm of unity. Despite the information of phase is senseless at measurement, the phase interaction in multiple qubit systems proves to have compelling effects on state vectors, which could be used to our advantage.

Another important property qubits possess is entanglement. It is a special type of correlation among states in a special type of superposition. In a system with two entangled particles, the measurement of one automatically triggers a correlated state of another particle, indefinite of separation in space. [17](Hidary, 2019, p. 6) This special property allows us to retrieve information of a qubit from an entangled pair by measuring the other qubit.

2.3 Quantum operators or Gates

In classical computers, logic gates like NOT, AND, XOR are used to manipulate bits. Likewise, quantum gates do manipulation of states of qubit/s. However, unlike classical operators, quantum operators are limited to reversible logical operations. So, irreversible classical gates like AND, OR, NAND are not directly accessible. To maintain reversibility, each quantum gate is represented by unitary matrix U,that is, $UU^\dagger = I$ where $U^\dagger$ is adjoint matrix [4] of U. Also, the postulate of QM suggests that all quantum operations are linear. Hence an operation on a qubit would imply operation on all the states in superposition. Operating an operator A on the state equation(2.1) would result $A|\Psi\rangle = \alpha A|0\rangle + \beta A|1\rangle$. Similarly operator acting on a n-qubit register $|x\rangle$ for each $x \in \{0, 1, ..., 2^{n-1}\}$ is a mapping $|x\rangle \rightarrow A|x\rangle$.[5] The

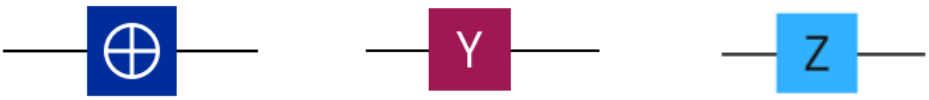
[4]Adjoint of a matrix A is the conjugate transpose matrix of A

operation of a gate on a qubit is the dot product of the operator matrix with the qubit.

2.3 Single qubit gates

Most of the operations on a qubit can be visualized as a rotation or combination of rotation of state vector in the Bloch sphere. For instance, some most important gates X, Y, Z gates perform a rotation of state vector by angle π or 180° about respectivex, y, and z axis in Bloch sphere. The matrix representation of these three gates/operation is called Pauli Matrices. They are given as

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$



(a) X-gate (b) Y-gate (c) Z-gate

Figure 2.2: X, Y and Z gate

X gate on a qubit$|\Psi\rangle$ given by (2.1) is

$$X|\Psi\rangle = X(\alpha|0\rangle + \beta|1\rangle) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix}$$

X gate transform $|0\rangle$ to $|1\rangle$ and transform $|1\rangle$ to $|0\rangle$. It is analogous to the NOT gate in classical computing.

Hadamard gate(H gate) is another important single-qubit gate in QC. This gate causes two consecutive rotations on qubit: 90° about Y-axis followed by 180° about the X-axis. The Hadamard matrix is

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

H gate when acted on a basis state(say $|0\rangle$) produces a superposition of all basis state ($|0\rangle$ and $|1\rangle$): $H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $H|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$

Figure 2.3: Hadamard Gate

The phase shift gate can be used to manipulate the relative phase of a qubit. Changing the relative phase implies rotating state vector by angle ϕ about z-axis with respect to another. For doing that one can either rotate both $|0\rangle$ and $|1\rangle$ by angles $-\phi/2$ and $\phi/2$ respectively or only rotate $|1\rangle$ by angle ϕ. In doing so, the amplitude of the output state vector is the same as that of the original. The phase shift gate is

$$R_\phi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} = e^{i\phi/2} \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix}$$

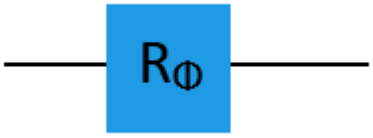


Figure 2.4: Phase-Shift Gate

When $\phi = \pi$, $R_\phi = Z$ gate

2.3 Multi qubit Gates

The efficiency of QC is realized only by the use and association of the multiple numbers of qubits. CNOT, SWAP, CCNOT, C-R_ϕ are some important multi-qubit gates in QC. CNOT is a Controlled-NOT gate. A controlled gate is a multi-qubit gate with a set of qubits classified as control qubits and a qubit as a target qubit, such that upon meeting a certain condition in control qubit, an operation is acted upon the target qubit. In other words, control qubits control the state of the target qubit.

A CNOT gate takes two inputs: control bit and target bit. It leaves the target

bit unchanged if the control bit is $|0\rangle$ and performs X gate or NOT gate if the control bit is $|1\rangle$. Since it is a two-qubit gate, its matrix is of order 4×4.

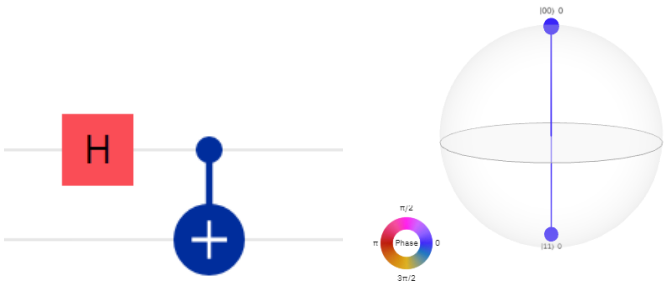
$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 2.5: CNOT gate

Mathematically, the transformation is

$CNOT\,|00\rangle \rightarrow |00\rangle\,, CNOT\,|01\rangle \rightarrow |01\rangle\,, CNOT\,|10\rangle \rightarrow |11\rangle\,, CNOT\,|11\rangle \rightarrow |10\rangle$

CNOT gate is used to create entanglement in QC. To realize a entangle system let us construct a quantum circuit as shown in fig(2.6). Before measurement, the qubit q_0 would be in state $\frac{1}{\sqrt{(2)}}(|0\rangle + |1\rangle$,i.e superposition state. The CNOT gate on the qubits result qubit q_1 to be in state $|0\rangle$ if the qubit q_0 is in state$|0\rangle$ or to be in state $|1\rangle$ if the qubit q_0 is in state $|1\rangle$. Here, just by knowing the state of the qubit q_0, the state of the qubit q_0 could be known.[1] This represents the entanglement of the two qubits. The final state of the system before measurement is $\frac{1}{\sqrt{(2)}}(|00\rangle + |11\rangle$. This state is one of the Bell state or EPR pairs. [24]*(Nielsen and Chuang, 2010,p. 25)*



(a) Circuit Diagram (b) Q-sphere for Bell state

Figure 2.6: Circuit for Bell state and visualization

Controlled-U gate can be considered as the generalized version of CNOT gate.[5] Consider U be a t qubit operator and $|\Psi\rangle$ be a quantum register with t qubits as target register and $|q\rangle$ be the control qubit. Controlled-U gate on a system $|q\rangle |\Psi\rangle$ allows the qubit $|q\rangle$ to control the operation U on register $|\Psi\rangle$. That is, the output is $U|\Psi\rangle$ if $|q\rangle = 1$ and the output is $|\Psi\rangle$ if $|q\rangle = |q\rangle$. The circuit diagram for the Controlled-U gate is as shown in figure(2.7).

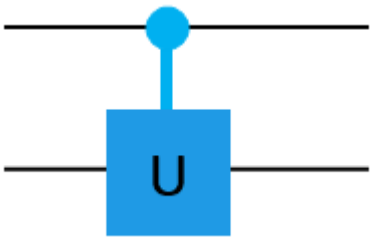


Figure 2.7: Controlled-U gate

After studying the condition of reversibility for quantum gates, one may speculate that irreversible classical logic gates could not be realized in QC. Fortunately, it was already realized that all the classical gates can be converted into reversible operators, so there's a unitary operator for all the irreversible gates. [5] (Cheung, p10) So it could be generalized that quantum gates are the generalization of the classical gates. An example of this is the realization of the XOR gate by the CNOT gate. CNOT operation on system $|a, b\rangle$ is

$$CNOT\,|a, b\rangle = |a, b \oplus a\rangle$$

, where $b \oplus a$ is the XOR operation on a and b. The generalized form of conversion of the irreversible circuit into an irreversible one is as follows:

$$|x, y\rangle \begin{cases} |x\rangle \\ \\ |y \oplus f(x)\rangle \end{cases} \tag{2.5}$$

2.3 Quantum Algorithm

Quantum Computational approach; use of different properties used in QC like superposition, entanglement, quantum parallelism[5], has been used to develop different quantum algorithms that would perform a certain task with superiority than classical approaches. The Quantum Computational approach opens a large prospect to develop other algorithms to solve problems or do tasks, which are difficult to solve classically or to decrease the complicity of the classical algorithm. Different algorithmic paradigms are built under the quantum computing approaches and quantum algorithms are built using the combination of such paradigms. Some of the established quantum algorithmic paradigms are Quantum Fourier transform (QFT), the Grover Operator (GO), the Harrow-Hassidim-Lloyd (HHL) method for linear systems, variational quantum eigenvalue solver (VQE), and direct Hamiltonian simulation (SIM).[6](Coles, 2018). In this paper, we will be discussing the implementation of the Quantum Fourier transform (QFT) in building different algorithms.

2.4 Quantum Fourier Transform

Quantum Fourier Transform (QFT) is rather a popular quantum algorithmic paradigm that acts as the foundation for several important algorithms including the infamous Shor's algorithm which factors integers in polynomial time, compared to the best classical method performing in sub-exponential time.[19] In QC, information about the state of the qubit is encoded in magnitude and phase of the system. However,

[5]The property of operators in QC to operate onto all the superimposed states simultaneously.[23]

information about magnitude can only be accessed easily and information encoded in phase is directly inaccessible. So information encoded in phase is considered as a part of hidden information stored in the qubit. QFT allows us to retrieve such hidden information stored in-phase and magnitude[18]. QFT retrieves the information by performing Fourier transform of quantum mechanical amplitudes.[24] *(Nielsen and Chuang, 2010, p216)* So QFT is considered as the quantum equivalent of Discrete Fourier transform.

2.4 Discrete Fourier Transform

Discrete Fourier Transform is the discrete form of Fourier transform, specifically designed for computation. DFT takes a set of time-domain discrete points and transforms them into frequency domain points. DFT has a very wide application in classical computers specially signal-processing, data compression, data cleaning, and filtration.

DFT takes a set of N complex numbers: $x_0, x_1, x_2, ..., x_{N-1}$ as input, and transformed them to a another set of N complex numbers: $y_0, y_1, y_2, ..., y_{N-1}$ given by relation:[24] *(Nielsen and Chuang, 2010, p 217)*

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi ijk/N} x_j = \sum_{j=0}^{N-1} w^{jk} x_j, \tag{2.6}$$

where $w = e^{2\pi i/N}$

For all the y_k, the transformation becomes $Y = W \cdot X$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ . \\ . \\ . \\ y_{N-1} \end{pmatrix} = \frac{1}{\sqrt{N}} \begin{pmatrix} w^0 & w^0 & w^0 & . & . & . & w^0 \\ w^0 & w^1 & w^2 & . & . & . & w^{N-1} \\ w^0 & w^2 & w^4 & . & . & . & w^{2(N-1)} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ w^0 & w^{N-1} & w^{2(N-1)} & . & . & . & w^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ . \\ . \\ . \\ x_{N-1} \end{pmatrix} \tag{2.7}$$

It has an algorithmic complexity of $O(n^2$. By some clever manipulation using the recursive nature of the problem, an improved version namely Fast Fourier Transform(FFT) was developed. FFT is widely used for most practical applications and is considered the best technique for discrete Fourier transform. It has complexity of $O(n \log n)$. "The recursive nature of the solution of FFT sets a stage for studying QFT"[20] *(Loceff, 2015,p 591)*

2.4 Quantum Fourier Transform

Consider a n qubit system given by $|\Psi\rangle = \sum_{k=0}^{N-1} c_k |k\rangle$(2.2) where $N = 2^n$. Quantum Fourier Transform on the linear combination $|\Psi\rangle$ is a transformation to a another linear combination $\sum_{k=0}^{N-1} d_k |j\rangle$ where $j = 0, 1, 2, ..., N-1$ defined by

$$|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} w^{jk} |k\rangle , \tag{2.8}$$

where $w = e^{2\pi i/N}$.

The transformation is

$$\sum_{k=0}^{N-1} d_k |j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} c_k e^{2\pi ijk/N} |k\rangle \tag{2.9}$$

[20]

We can see this is a mapping from the vector of N probability amplitude of original system $|\Psi\rangle$ to another vector of new N amplitudes of state for the system. The

transformation matrix for mapping is

$$W = \frac{1}{\sqrt{N}} \begin{pmatrix} w^0 & w^0 & w^0 & . & . & . & w^0 \\ w^0 & w^1 & w^2 & . & . & . & w^{N-1} \\ w^0 & w^2 & w^4 & . & . & . & w^{2(N-1)} \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ w^0 & w^{N-1} & w^{2(N-1)} & . & . & . & w^{(N-1)(N-1)} \end{pmatrix} \tag{2.10}$$

The matrix W is unitary since $W \cdot W^\dagger = W^\dagger \cdot W = I$. Hence, QFT circuit is a reversible i.e. is a valid quantum circuit.

2.4 Construction of QFT circuit

We would need to disintegrate the transformation matrix (2.10) of QFT to the number of individual gates applied to single qubits or multiple qubits for the construction of the circuit. QFT on n qubit system with quantum state $|j\rangle$ undergoes transformation given by

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} w^{jk} |k\rangle , \tag{2.11}$$

where $w = e^{2\pi i/N}$.

where $|j\rangle = |j_1 j_2 ... j_{n-1} j_n\rangle$ has the binary representation such that $j = 2^{n-1} j_1 + 2^{n-2} j_2 + ... + 2^1 j_{n-1} + 2^0 j_n = \sum_{l=1}^{n} 2^{n-l} j_r$, $j_i \in \{0, 1\}$. Also fraction binary representation $0.j_1 j_{l-1} ... j_m$ is given by $\frac{j_1}{2} + \frac{j_{l+1}}{4} + ... \frac{j_m}{2^{m-l+1}}$

So,

$$|j\rangle = \frac{1}{\sqrt{N}} \sum_{k_1 k_2 ... k_n \in \{0,1\}}^{N-1} w^{j \sum_{r=1}^{n} k_r 2^{n-r}} |k_1 k_2 k_3 ... k_n\rangle \tag{2.12}$$

$$= \frac{1}{\sqrt{N}} \sum_{k_1 k_2 ... k_n \in \{0,1\}} \bigotimes_{r=1}^{n} w^{j k_r 2^{n-r}} |k_r\rangle \tag{2.13}$$

$$= \frac{1}{\sqrt{N}} \bigotimes_{r=1}^{n} \Big(\sum_{k_r \in \{0,1\}} w^{j 2^{n-r} k_r} |k_r\rangle \Big) \tag{2.14}$$

$$= \frac{1}{\sqrt{N}}\{\bigotimes_{r=1}^{n}\big(|0\rangle + w^{j2^{n-r}}|1\rangle\big)\} \tag{2.15}$$

$$= \frac{1}{\sqrt{N}}\bigotimes_{r=1}^{n}\big(|0\rangle + e^{\frac{2\pi i 2^{n-r}}{2^n}}|1\rangle\big) \tag{2.16}$$

$$= \frac{1}{\sqrt{N}}\bigotimes_{r=1}^{n}\big(|0\rangle + e^{2\pi i j 2^{-r}}|1\rangle\big) \tag{2.17}$$

$$= \frac{1}{\sqrt{N}}\bigotimes_{r=1}^{n}\big(|0\rangle + e^{2\pi i \sum_{l=1}^{n} 2^{n-l} j_r 2^{-r}}|1\rangle\big) = \frac{1}{\sqrt{N}}\bigotimes_{r=1}^{n}\big(|0\rangle + e^{2\pi i \sum_{l=1}^{n} 2^{n-l-r} j_r}|1\rangle\big) \tag{2.18}$$

And let $S = \sum_{l=1}^{n} 2^{n-l-r} j_l$ put $r = n$, $S = \sum_{l=1}^{n} 2^{-l} j_r = \frac{j_1}{2} + \frac{j_2}{4} + ... + \frac{j_n}{2^n} = 0.j_1 j_2 ... j_n$,
put $r = n-1$, $S = \sum_{l=1}^{n} 2^{-(l-1)} j_r = j_1 + \frac{j_2}{2} + \frac{j_3}{4} + ... + \frac{j_n}{2^{n-1}} = j_1 + 0.j_2 j_3 ... j_n$
Similarly,at $r = 1$, $S = \sum_{l=1}^{n} 2^{(n-1)-l} j_r = j_1 + j_2 + j_3 + ... + j_{n-1} + 0.j_n$
But $e^{2\pi i j_1} = e^{2\pi i (j_1 + j_2)} = e^{2\pi i (j_1 + j_2 + ... j_{n-1})} = 1$
So, we could write

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}}\big(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\big)\big(|0\rangle + e^{2\pi i 0.j_{n-1} j_n}|1\rangle\big)\big(|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n}|1\rangle\big) \tag{2.19}$$

Now let us use these equations to generate gates, we are familiar with

1. The first part of equation $|t_n\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\big)$ can be realized as a result of applying Hadamard gate on qubit $|j_n\rangle$, that is, if $j_n = 0, |t_n\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big) = H|0\rangle$
   if $j_n = 1, |t_n\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle = H|1\rangle$
2. The second part $|t_{n-1}\rangle = \frac{1}{\sqrt{N}}\big(|0\rangle + e^{2\pi i 0.j_{n-1} j_n}|1\rangle\big)$ can be realized as the result of applying Hadamard gate on $|j_{n-1}\rangle$ followed by controlled phase rotation by $|j_{n-1}\rangle$ on $|j_n\rangle$, $|t_{n-1}\rangle = H_{n-1}\big(|0\rangle + e^{2\pi i j \frac{n}{4}|1\rangle} = H_{n-1} R_{\frac{n}{2}}|j_{n-1}\rangle$
3. Similarly, the other terms can be realized by applying the Hadamard gate followed by a series of controlled phase rotation. So, $|t_1\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_n}|1\rangle$

The final obtained set of qubits must be reversed for it to match with the equation [2.18]. It is obtained by $\frac{n}{2}$ number of a qubit swap operation. The Circuit diagram

for generalized QFT is as shown in the figure.

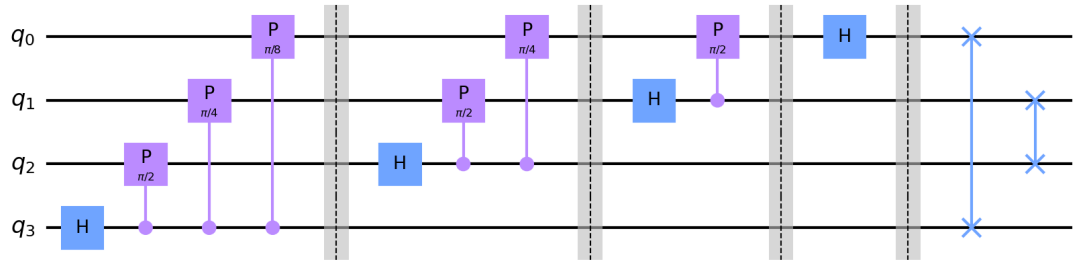


Figure 2.8: Circuit Diagram of Quantum Fourier transform(QFT) on a four qubit system

Complexity of QFT

From the figure, the first line has n gates, the second one has $n-1$, and so on. There are $\sum n = \frac{n(n-1)}{2}$ number of gates for QFT transformation and $\frac{n}{2}$ swap gates for further arrangement. Hence the algorithm complexity of the circuit is polynomial. It has $\mathcal{O}(n^2)$ complexity. Comparing with classical FFT whose complexity is $\mathcal{O}(n2^n)$, QFT seems highly promising. However, there are some major complications that make QFT ineffective for classical FFT operation. Such complications are errors in measurement of all amplitudes, collapse of superposition upon measurement, and difficulty in creating the initial state.

2.4 Phase Estimation

Phase estimation is not a complete algorithm to do a particular task but a sub-routine that performs important operation in QC for determining the information encoded in phase of qubit [18]. Phase estimation has QFT as its key component during the procedure [24]. Global phases are unobservable by direct measurement. For example, state $|\Psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\frac{\pi}{2}}|1\rangle)$ and $|\Psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{-i\frac{\pi}{2}}|1\rangle)$ gives same result on measurement.

This subroutine takes two registers: first registers of t qubits initialized at state $|0\rangle$ and second $|1\rangle$ which is an eigenvector to a unitary operator U^{φ} and eigenvalue is $e^{2\pi\varphi}$ (where φ is unknown). QFT based phase estimation define phase φ to be

exact integer multiple of $\frac{1}{2^n}$ and is encoded as binary fraction $\varphi = 0.x_1x_2...x_n$ [24]. Phase estimation extract phase information from register $|u\rangle$ and encodes it in the Fourier basis of 1st register. And upon performing inverse Fourier transform such information could be extracted.

The first step for phase estimation is Hadamard operation on 1st register $|\Psi_1\rangle = \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |u\rangle$

Circuit proceed by applying a controlled U operation on a second register with U raised to successive power of 2 as shown in the figure().

$|\Psi_2\rangle = \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle U |u\rangle$

$= \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{2\pi\varphi} |j\rangle |u\rangle$

We use the information of phase kickback to proceed and perform inverse quantum Fourier transform to get:

$$|\Psi_1\rangle = |\tilde{\varphi_u}\rangle |u\rangle$$

And finally measuring the first register would give the state $|\tilde{\varphi_u}\rangle$

The number of qubits in the first register(t) is important in phase estimation as it defines the accuracy and precision of the phase of qubits in register $|u\rangle$. For example, for 3 qubit 1st register, the circuit could accurately estimate angles $\frac{2\pi}{8}n$ where n = 0, 1, 2, . . 7, as phase of state $|u\rangle$. Similarly, for 4 qubit register, only if $|u\rangle$ is at a phase among one of the angles: $\frac{2\pi}{16}n$ where n= 0, 1, 2, . . . 15, the circuit could accurately find phase.

$|\tilde{\varphi_u}\rangle$ is measured to be one of $R = 2^t$ states. If the measured state is $i \in [0, 2^t-1]$, it suggests the state $|u\rangle$ has angle $\frac{2\pi i}{n}$.

If the number of a register is not enough i.e. resolution of ψ is not enough, the output ends up in superposition around the closest possible value. One could estimate a low-resolution phase with some probability.

2.5 Number theory Background

Let us introduce some background of the number theory necessary for the study.

2.5 Modulo Operation

The reminder r upon dividing a number b by a is output of modular operation: b *mod* a, where *mod* denotes modular operation. Mathematically, it is represented as :

$$b \bmod a = r \text{ or } b \equiv r \pmod{a}$$

Modulo operation is equivalent to division as : $b = q * a + r$ where q: an integer ($q \in \mathbb{Z}$), is the quotient of division operation of b, dividend by a divisor. Some important properties in modular operation are:

- (b mod a) mod a = b mod a
- b^x mod a = 0 for all x $\in \mathbb{Z}$
- ((-b mod a) + (b mod a)) mod a = 0
- (b + c) mod a = ((b mod a) + (c mod a)) mod a
- b*c mod a = ((b mod a) (c mod a)) mod a

2.5 Greatest Common Divisor(GCD)

The greatest common divisor of two integers P and Q denoted by GCD(P, Q) is the largest integer r such that r exactly divides P and Q exactly, i.e. $r \mid P$ and r | Q [6].

Definition 2.5.1. Two integers a and b are defined as relatively prime to each other, if $GCD(a, b) = 1$, expressed as $(a \phi b)$.

We use the Euclidean algorithm to find GCD.

Lemma 1. *For* $P, Q \in \mathbb{N} : P > Q$ *and* $p = q * Q + r$, $GCD(P, Q) = GCD(Q, r)$ *and* $GCD(a, 0) = a$

[6] a | b denotes a divisibility relation such that b is exactly divisible by a or a is a factor of b

Euclidean Algorithm

To compute the GCD of integers P and Q : $(P > Q)$, the euclidean algorithm is as follows:

1. Initialize $r_0 = P$ and $r_1 = Q$
2. Compute r_2 such that $r_0 = q_0 * r_1 + r_2$
3. if $r_2 = 0$, $GCD(P, Q) = r_1$
4. Since $GCD(r_0, r_1) = GCD(r_1, r_2)$, compute $GCD(r_1, r_2)$. So, put $r_0 = r_1$ and $r_1 = r_2$ and go to step 2

Time complexity of Euclidean Algorithm

We have, $p = q * Q + r$ with $P > Q > r \in \mathbb{Z}$ If $Q \leq \frac{P}{2}$, then $r < \frac{P}{2}$ since $r < Q$ and if $Q > \frac{P}{2}$, then $r = P - Q \leq \frac{P}{2}$. We can see, the reminder r_0 is halved in every two iterations. So, an even r_2 becomes 0 with at most $2logP$ iterations. Hence the number of steps is $\mathcal{O}(2logP) = \mathcal{O}(logP)$. Now, each of the steps is division operation and has $\mathcal{O}(log^2 P)$ complexity. Hence the total algorithmic complexity of the Euclidean algorithm is $\mathcal{O}(log^3 X)$ where X is the largest between P and Q.

2.5 Modular exponentiation function and its order

A Modular exponentiation functionMEF is a function in form $f(x) = z^x \text{modn}$ for $x \in \mathbb{Z}$. It is a periodic function and its period is equal to the order of the function.

Definition 2.5.2. Order of modular function $z^x \ mod \ n$ is the smallest integer $x > 1$ such that

$$z^x \equiv 1 \text{ (mod n) or } z^x \text{ mod n} = 1$$

$f(x)$ is an injective periodic function.[20] That means if a is the period, then for every $x \in 0, 1, 2, ..., a - 1$ maps to distinctly unique element with periodicity at

interval of a. i.e. For all $y \in \{0, 1, , 2, ..., r-1\}$, $f(y) = f(y+a) = f(y+2a) = ... = f(y+(m-1)a) \Rightarrow f(y) = f(y+ja)\rfloor_{j=0}^{m-1}$ where $m = \frac{N}{a}$ is frequency of a.

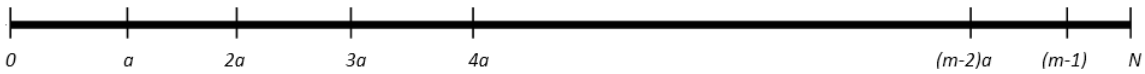


Figure 2.9: Line scale showing periodicity of $f(x)$

Hence, we can partition the whole input domain: $x \in [0, N-1] = \mathbb{Z}$ into m or m+1 partitions called cosets, where m is the number of the periodicity of the system. We can write,

$\mathbb{Z} = Q \cup (Q+a) \cup (Q+2a) \cup ... \cup (Q+(m-1)ma) \cup < Q+ma >$

$= [0, a-1] \cup [a, 2a-a] \cup ... \cup [(m-1)a, ma-1]$

$= \bigcup_{j=0}^{m-1} \{x+ja\}_{x=0}^{N-ma-1}$

Example: (1). Say z= 3, N= 16, $f(x) = 3^x \pmod{16}$

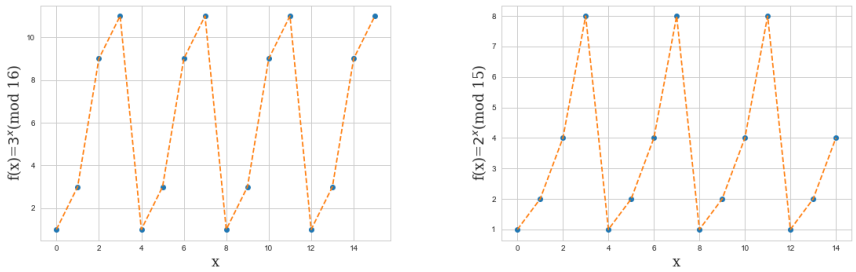
f(0)	f(1)	f(2)	f(3)	f(4)	f(5)	f(6)	f(7)	f(8)	f(9)	f(10)	f(11)	f(12)	f(13)	f(14)	f(15)
1	3	9	11	1	3	9	11	1	3	9	11	1	3	9	11

Table 2.1: MEF for function $f(x) = 3^x \bmod 16$

(2). z= 2, N= 15, $f(x) = 2^x \pmod{15}$

f(0)	f(1)	f(2)	f(3)	f(4)	f(5)	f(6)	f(7)	f(8)	f(9)	f(10)	f(11)	f(12)	f(13)	f(14)
1	2	4	8	1	2	4	8	1	2	4	8	1	2	4

Table 2.2: MEF for function $f(x) = 2^x \bmod 15$



(a) Output for $f(x) =^x \pmod{16}$ (b) Output for $f(x) = 2^x \pmod{15}$

Figure 2.10: Different cases of periodicity of MEF

Example(I) shows the case for m= N which is an ideal case and is impractical for generalization and Example(II) shows the case for $ma \neq N$ which is considered the general case.

For easiness in the general case, we define a number $\tilde{m}$ such that

$$\tilde{m} = \begin{cases} m+1 & \text{for first few elements in } [0, a-1] \\ m & \text{for remaining elements in } [0, a-1] \end{cases}$$

Lemma 2. *Suppose $z < N$. z is co-prime with N, then there exist a function $a^{\psi(n)} = 1(modN)$. [24](Neilsen et al. 2002,p631)*

Lemma 2 states that for two co-primes z<N, the MEF always has an order 'a'.

2.5 Continued Fraction

A continued fraction is a successive sequence of a fraction of sums and quotients of integer in the form:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cfrac{1}{a_5 + \cdots}}}}}$$

A sequence in form $[a_0 : a_1, a_2, a_3, a_4, a_5, \dots]$ is used to represent the whole object. And the continued fraction sequence is

$$\{\frac{n_0}{d_0}, \frac{n_1}{d_1}, \frac{n_2}{d_2}, \frac{n_3}{d_3}, \cdots,\}$$

, where

$$\frac{n_0}{d_0} = a_0,\ \frac{n_1}{d_1} = a_0 + \frac{1}{a_1},\ \frac{n_2}{d_2} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2}},\ \frac{n_3}{d_3} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3}}}, \cdots$$

Example: π

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cdots}}}}$$

With sequence: $\{3, \frac{22}{7}, \frac{333}{106}, \frac{355}{115}, \frac{103993}{33102}, \cdots\}$

A continued fraction to a finite number of terms is called a finite continued fraction. All rational numbers have a finite continued fraction and irrational numbers have an infinite continued fraction.

Convergence of continued fraction

The k^{th} convergent of a continued fraction $[a_0 : a_1, a_2, a_3, a_4, a_5, \cdots a_n]$ is $C_k = \frac{n_k}{d_k}$ is the k^{th} term in the continued fraction sequence and is the k^{th} approximation of a_n.[2]

Some important properties of a continued fraction are

1. Every odd index convergents are strictly decreasing and every even index convergents are strictly increasing.

   Since the convergents eventually converge to a point, every odd index congruent is greater than every even index convergents.

For a continued fraction of x, $\frac{n_k}{d_k} = \begin{cases} > x & \text{if k is odd} \\ < x & \text{if k is even} \end{cases}$

It can be visualized using an example and the figure below.

$$\frac{127}{55} = 2 + \cfrac{1}{3 + \cfrac{1}{4 + \cfrac{1}{4}}}$$

and sequence is $\{2, \frac{7}{3}, \frac{30}{13}, \frac{127}{55}\}$

2. The k^{th} convergent $\frac{n_k}{d_k}$ of finite simple continued fraction $x = [a_0 : a_1, a_2, a_3, a_4, \cdots]$ follows: $\mid \frac{n_k}{d_k} - \frac{n_{k-1}}{d_{k-1}} \mid = \frac{1}{d_k d_{k-1}}$

3. For all the convergents $\frac{n_k}{d_k}$ in a finite continued fraction sequence of $x = d_K$, the denominator is strictly increasing.i.e. $d_k > d_{k-1}$ for all $0 < k < K$

Approximation

Definition 2.5.3. An convergent $\frac{n_k}{d_k}$ of continued fraction of x is said to be an good approximation of x, if $\frac{n_i}{d_i} \neq \frac{n_k}{d_k}$ and $d_i < d_k$ for all $0 < i < k$.[26]

$$\left| x - \frac{n_k}{d_k} \right| < \left| x - \frac{n_i}{d_i} \right|$$

for all $0 < i < k$

It means $\frac{n_k}{d_k}$ is closest to x than all other convergent with denominator less than d_k.

Lemma 3. *"Let x be an real number. If the rational number $\frac{n}{d}$, where $d \geq 1$ and $GCD(n, d) = 1$, satisfies*

$$\left| x - \frac{n}{d} \right| < \frac{1}{2d^2} \tag{2.20}$$

,then $\frac{n}{d}$ is one of the convergents $\frac{n_k}{d_k}$ in the continued fraction sequence of x." [2](Burton, 2011)

Continued fraction Algorithm

Algorithm 1: Algorithm to output a approximate convergents of a continued fraction bounded by an error

Data: A number: num, error:E

Result: Convergent: $\frac{n_k}{d_k}$

Initialization: Input number=num, Error= E ;

Compute $a_0 = \lfloor num \rfloor$;

if $(\frac{num}{a_0} \neq 1)$ **then**

  Compute $a = \lfloor \frac{1}{num - a_0} \rfloor$;

  Compute $n_0 = a_0; d_0 = 1$;

  Compute $n_k = n_1 = a * a_0 + 1; d_k = d_1 = a$;

  while $(\left| \frac{n_k}{d_k} - num \right| \geq E)$ **do**

    Compute $a = \lfloor \frac{1}{num - a} \rfloor$;;

    Compute $n_k = a * n_1 + n_0$, $d_k = a * d_1 + d_0$;

    Set $n_0 = n_1$, $d_0 = d_1$;

    Set $n_1 = n_k$, $d_1 = d_k$;

  Output: $\frac{n_k}{d_k}$;

else

  Output: num;

2.6 Complexity

Computational complexity is the study of problem analysis to categorize them based on resources like time, space, and energy required to solve the problem.[24] One can speculate that a search algorithm is harder and require more resources than an addition algorithm. Different problems may require a different amount of resources. Even for a single problem, different algorithms, requiring different amounts of time or space, can be used to determine the same output. The algorithm is the set of steps performing mathematical operations to solve problems. So, they are independent of

the hardware used or software implemented. This makes computational complexity a general way to study problems and define efficiency.

The complexity of a problem or algorithm is defined based on the size of the input: bit-length or number of input data, n. It studies how a problem grows as the size of input increases. Conventionally, we use Big $\mathcal{O}$ notation and Big Ω notation to define complexity. Big $\mathcal{O}$ is used to express the upper-bound of the worst-case during the execution of an algorithm. And Big Ω represents the lower-bound of the worst case. We will be using Big $\mathcal{O}$ notation more often than Big Ω in the study. An algorithm has the complexity of $\mathcal{O}(g(n))$ where $g(n)$ is a function of the size of input or number of steps n, which means the algorithm will at most use $g(n)$ amount of space(memory) or take $g(n)$ number of steps to complete. An important property of Big $\mathcal{O}$ notation is that an algorithm is as good as its worst sub-operation or sub-routine. That is, if $f(n)$ is the resources used in an algorithm defined as $f(n) = a_0 n^k + a_1 n^{k-1} + \cdot + a_k$, then f is $\mathcal{O}(n^k)$.

Complexity is categorized into subcategories according to the resources used. Time complexity and Space complexity are widely used to measure the efficiency of an algorithm. Time complexity is the study of an algorithm based on the number of steps or time taken to complete. Space complexity deals with the memory or space needed to store data during the execution of the algorithm.

In classical computing, there are different complexity classes based on space and time complexity. Figure(2.11) shows various complexity classes with increasing complexity with an increase in the area of the region of the class. Some common and significant complexity classes are:

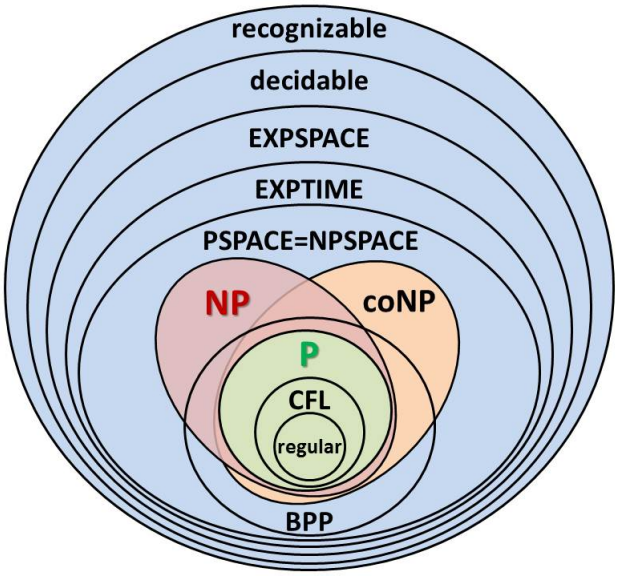


Figure 2.11: Some common complexity classes with their relations [8]

1. P class

   P – polynomial class problems are the problems that have polynomial time complexity. This class includes problems with constant $\mathcal{O}(1)$, linear $\mathcal{O}(n)$, logarithmic $\mathcal{O}(log(n))$, log normal $\mathcal{O}(nlog(n))$ and quadratic $\mathcal{O}(n^2)$ and other polynomial complexity

2. NP class

   NP – Non-polynomial class problem has non-polynomial time complexity. The problem in this class has solutions verifiable in polynomial time. However, finding the solution takes non-polynomial time like exponential time.

3. PSPACE

   PSPACE – Polynomial space problem consumes memory in polynomial scale as the input size grows. Figure (2.11) shows the hierarchical position of complexity.

4. BPP

   BPP – Bounded-error probabilistic polynomial class problem is a different type of problem that has a bounded probability of being solved in polynomial time. Polynomial class problem problems are included in it. Usually, problems

with a success probability of more than 2/3 are included in this class

5. BQP

   BQP - Bounded-error Quantum polynomial problems are similar to BPP problems, except that these problems are solved by quantum computer or quantum algorithm. It covers some problems of NP class and P class problems. The success of an algorithm of this class is bounded by some high probability. We will prove Shor's algorithm is a BQP algorithm later in the study.

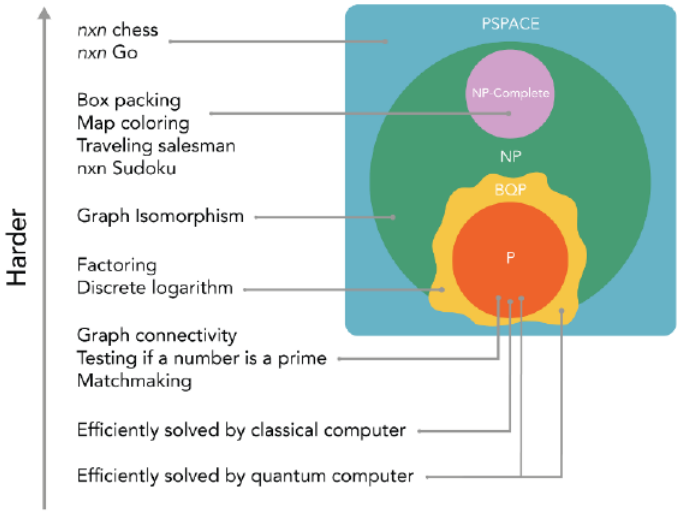


Figure 2.12: Figure showing some common complexity classes with their examples [17](Hidary, 2019, p41)

Chapter3: Thesis Problem and Objective

3.1 Thesis Problem

Consider an number $M \in \mathbb{Z}$ and its prime factorisation $M = P_1^{e_1} P_2^{e_2} ... P_k^{e_k}$ where each $P_i\rfloor_{i=1}^{k}$ and $e_i\rfloor_{i=1}^{k}$ are prime number and their integer exponent respectively, then find each primes of M: P_i's. It is called integer factorisation problem.

The uniqueness of prime factorization suggests that the prime factors P_i's are unique to integer M.1 Integer factorization problem is a difficult problem in number theory. Based on its hardness, it has some important applications, which we will discuss later in the study.

3.1 Problem Re-scaling

Here, we downscale the problem to a simpler problem, yet with equal complexity and output.

Claim 1. An integer $M = P_1^{e_1} P_2^{e_2} ... P_k^{e_k}$ as its prime decomposition can be expressed as an integer $I = P.Q$

Proof. Suppose an integer $I = P.Q$ where P is a prime number and $Q \in \mathbb{Z}$
we have, $M = P_1^{e_1} P_2^{e_2} ... P_k^{e_k}$
we can rewrite it as $M = P_1 P_1^{e_1 - 1} P_2^{e_2} ... P_k^{e_k}$
let $P = P_1$, $Q = P_1^{e_1 - 1} P_2^{e_2} ... P_k^{e_k}$
$M = P.Q$
Hence $M \equiv I$ □

From the above conjecture, we can represent M as $I = P.Q$. And once we find an efficient algorithm to find a prime factor of I, say P, we can use the same algorithm to decompose Q if it is a composite number. Hence we reduce the problem of finding prime factors of an integer to a problem to develop an efficient algorithm to find a single prime. Let us assume such an algorithm exists and name it Algorithm-S.
The algorithm for integer factorisation is:

Algorithm 2: Algorithm for integer factorisation

Data: An integer to be factored: M

Result: Prime factors: P_1, P_2 , . . . P_k

initialization: $I = M$, i = 1;

while *($I \neq 1$)* **do**

    Apply Algorithm-S on I to find P_i;

    $Q = I/P_i$;

    while *(Q mod P_i =0)* **do**

        temp = I/P_i;

    $I = temp$;

    i=i+1;

3.2 Objective

The aim of this thesis is to explore Shor's algorithm for factoring a composite integer and implement it on a quantum computer. Following objectives are set to meet the aim.

1. Study workings of Shor's algorithm
2. Calculate the probability of success of the algorithm
3. Compute complexity of the algorithm
4. Construct a circuit for implementation of the algorithm
5. Run algorithm on a quantum simulator and a quantum computer
6. Study application of Shor's algorithm in breaking RSA cryptography

Chapter4: Method

4.0 Classical solution

There are several classical methods of integer factorization. Some of the notable methods are

Brute force

To factor an integer I, one could run an algorithm from 2 to I to check if any of them divides I exactly. The ones that divide exactly or numbers that are not co-prime with I ($\neg x \phi I$) are factors. This algorithm would take $O(N)steps = O(2^w)$ steps where $w = \log_2 I$ is the length of I.

With a little analysis, we could prove that any factor of n would be less than the square root of $I(P < \sqrt{I})$. Hence, the loop could be reduced to $\sqrt{I}$ times. It has $O(\sqrt{I}) = O(2^{\frac{w}{2}})$ complexity which is exponential.

GNFS - General Number Field Sieve

It is the best known classical algorithm for prime number factorization developed by John Pollard in 1996. It factorizes large integer into its factor in a super polynomial or sub-exponential time. The asymptomatic complexity of GNFS is $O(e^{(\log I)^{\frac{1}{3}}(\log(\log I))^{\frac{2}{3}}}) = O(e^{w^{\frac{1}{3}}(\log w)^{\frac{2}{3}}})$ where I is number to be factored and $w = \log I$[16]

4.1 Shor's Algorithm

Shor's Algorithm is a quantum algorithm for composite integer factorization with polynomial time complexity.[28] This algorithm could factor a number I that has a product of two primes : P and Q such that $I = P.Q$ with $\mathcal{O}(log^3(I))$ complexity for large I. Shor developed this algorithm in 1994. The paper is considered a "landmark paper " in quantum computing.[17] (Hidary, 2019) This algorithm sets a significant practical milestone that sparked the interest of a lot of physicists,

computer scientists, and mathematicians in the field.

Integer factorization is a problem that is considered an NP (Non-polynomial) class problem in classical computing for its exponentially increasing complexity. Shor's Algorithm exploits parallelism, interference, and other properties of a quantum computer to get a quantum speed up to its classical counterparts.

The following graph shows the difference in the complexity of Different algorithms with Shor's algorithm.

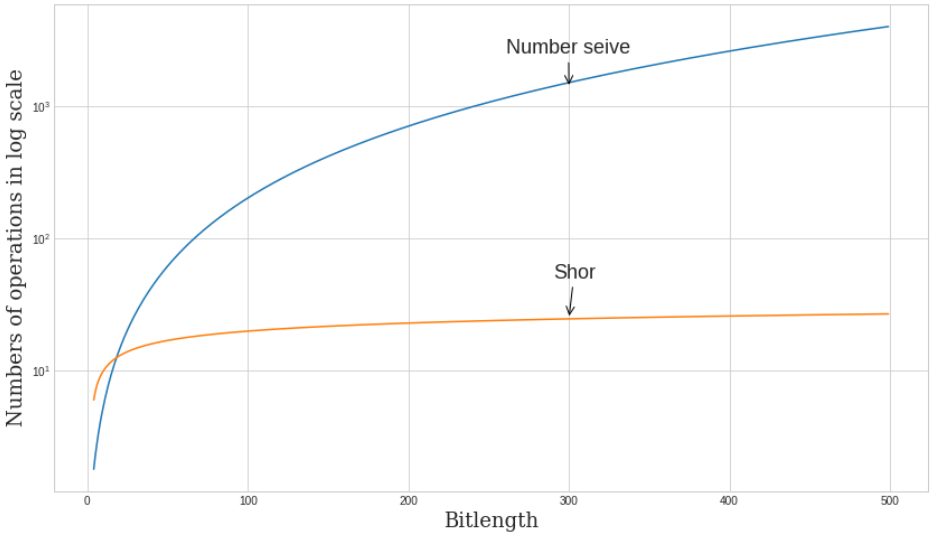


Figure 4.1: Figure showing complexity of Shor's algorithm and number field seive

4.2 Workings of Shor's Algorithm

Let $I \in \mathbb{Z}$ be a composite number, the goal is to find a prime number p such that p exactly divides I or $p|I$.

we divide I into three categories for the study:

I I is even, $I = 2z$, $z \in \mathbb{Z}^+$

II I is power of a prime(p), $m = p^k$, $k \in \mathbb{Z}^+$

III I is odd and is not power of prime

Case I

For an even I, it is easy to find the prime factor, that is, 2. The next prime factor can be determined using the found factor.

Case II

Case II is also a classically polynomial problem. For $m = p^k$, $k \in \mathbb{Z}^+$, the exponent k satisfies $logI > k$ since $I > p$ and $p = \lfloor \sqrt[k]{I} \rfloor$.
Iterating j from 2 to $logI$ on $p_j = \lfloor \sqrt[j]{I} \rfloor$ would give $(logI - 1)$ values for p_j. The p_j's is checked if $(p_j)^j = I$. The smallest p_j to satisfy the condition is the prime factor.

Case III

Shor's algorithm only solves the factoring problem for the third case. The exponentially large classical complexity for the factoring problem, we discussed earlier arise due to case III. We assume I to be odd and is not the power of prime for the rest of the study.

4.3 Shor's algorithm: Number Theory Approach

We assume Case III: I is not even and is not the power of the prime

1 Choose a random number : $1 < x < I$

2 If $GCD(x, I) \neq 1$, then factor $= GCD(x, I)$ where GCD = greatest common divisor

3 If $GCD(x, I) = 1$, find period or order 'a' of a function $f(r) = x^r mod, r \in \mathbb{Z}(I)$

4 If period: a is odd or $x^{\frac{a}{2}} \equiv -1(modI)$, we restart the algorithm from step 1.

5 If a is even and $x^{\frac{a}{2}} \equiv 1(modI)$ then, factors of I are: $p = GCD(x^{\frac{a}{2}} + 1, I)$ and $q = GCD(x^{\frac{a}{2}} - 1, I)$.

Random selection of x

If the GCD of randomly selected x and I is not 1, it means x and I share a factor and the GCD gives the factor of N. If $GCD(x, I) = 1$, it means x and I are co-prime with each other.

Find order of modular exponential function

Since x and I are co-prime, by theorem (2), order or period of function $x^r mod I$ exist. The process of order finding is the most difficult task in Shor's algorithm. The complexity of order finding grows exponentially as the bit-length of I. Figure(4.2)and (4.3) shows the classical growth of complexity of the task. We will discuss the

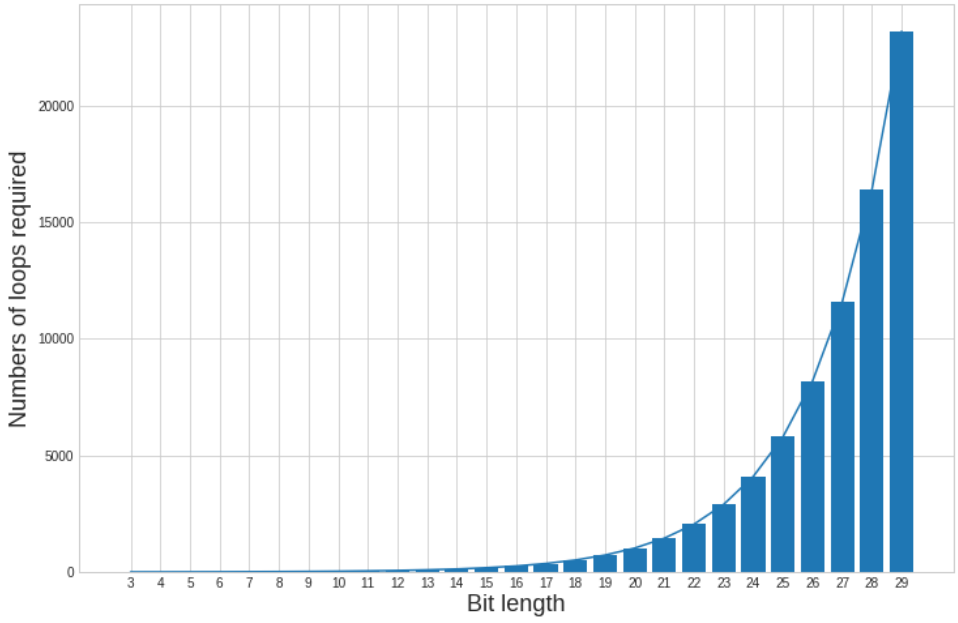


Figure 4.2: Graph of bit-length versus maximum number of loops needed to find period for the integer

process of the order finding in polynomial time using Quantum Computer later in the study.

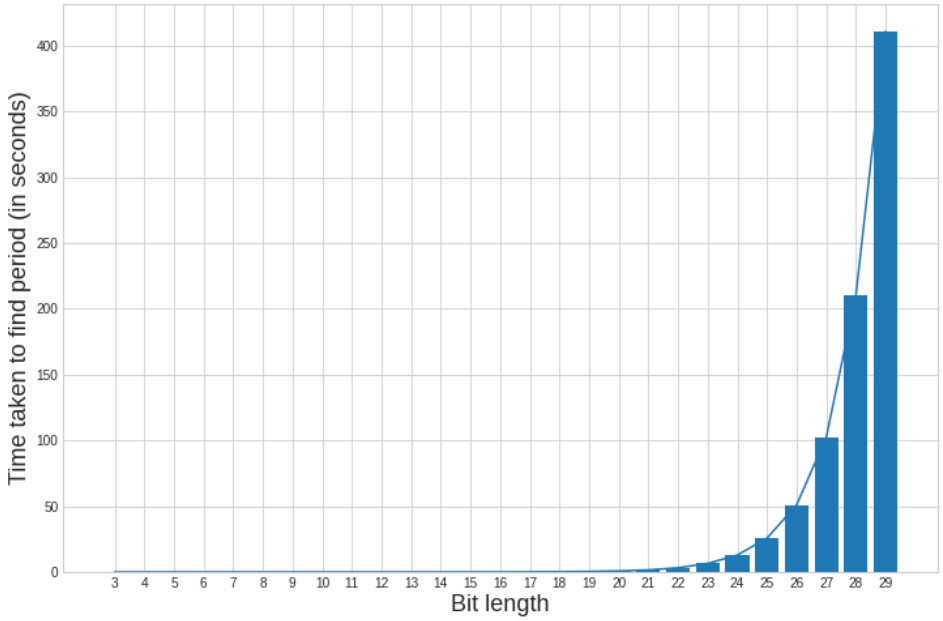


Figure 4.3: Graph of bit-length versus time required to find period for the integer

Determine factors using order 'a'

Theorem 1. *Suppose I is a composite number and $0 < x < I$; $x \in \mathbb{Z}$ such that function $f(r) = x^r mod I$ has even period, a and $x^a \not\equiv -1 mod I$, then I has non trivial factors given by $GCD(x^{\frac{a}{2}} - 1, I)$ and $GCD(x^{\frac{a}{2}} + 1, I)$ [24]*

Proof. Since a is period of function: $f(r) = x^r \text{mod I}$, $x^a \equiv 1 \text{ mod } I$

$$\Rightarrow (x^{\frac{a}{2}} - 1)(x^{\frac{a}{2}} + 1) \equiv 0 \text{ mod } I$$

Also, we know, $(x^{\frac{a}{2}} \not\equiv 1 \text{ mod } I)$ or $(x^{\frac{a}{2}} - 1) \not\equiv 0 \text{ mod } I)$ as it contradicts that a is order of function. Also, by definition, we have, $(x^{\frac{a}{2}} + 1) \not\equiv 0 \text{ mod } I$.

Hence we could find the non-trivial factors that are: $GCD(x^{\frac{a}{2}}+1, I)$ and $GCD(x^{\frac{a}{2}} - 1, I)$.

Determination of non-trivial factor of I requires $(x^{\frac{a}{2}} + 1) \not\equiv 0$ mod I because $(x^{\frac{a}{2}} + 1) \equiv 0$ mod I implies $I|(x^{\frac{a}{2}}+1)$, which gives trivial factor I upon applying Euclidean algorithm for GCD. QED □

After finding the period of the function, we do not always get non-trivial factors.

For any randomly selected x, the obtained period 'a' can be even or odd. One can see an odd period would give decimal exponent in term $(x^{\frac{a}{2}} - 1)$, giving trivial factors. Also, an even power satisfying the condition $(x^{\frac{a}{2}} - 1) \not\equiv 0$ mod I would contradict a is the order of x mod I. By lemma (1), only even order satisfying condition $x^{\frac{a}{2}} \not\equiv -1$(mod I), gives non trivial factor of I: $GCD(x^{\frac{a}{2}} + 1, I)$ and $GCD(x^{\frac{a}{2}} - 1, I)$. Hence, we factorized an integer I. Now let us talk about the likelihood of a randomly selected x to give an even period satisfying condition $x^{\frac{a}{2}} \not\equiv -1(modI)$.

Theorem 2. *Let n be an odd integer and let* $n = p_1^{e1} p_2^{e2} ... p_k^{ek}$ *be the prime factorization of n, Then the probability that a uniform randomly chosen x has even order a and* $x^{\frac{a}{2}} \equiv -1$*(mod I) is at least* $(1 - \frac{1}{2^{k-1}})$ *[24]*

For an odd integer composed of two prime, by theorem(2) the probability of getting odd period or satisfying condition $x^{\frac{a}{2}} \equiv -1$ (mod I) is less than $\frac{1}{2}$. So, it is independent of the integer. For T number of trials, the probability of getting an even order satisfying condition $x^{\frac{a}{2}} \not\equiv -1$(mod I) is greater than $(1 - \frac{1}{2^T})$. Hence, if period finding task is completed in polynomial time, we efficiently solved the integer factorization problem in constant time

4.4 Quantum part of Shor's Algorithm

All of the above steps except period finding can be done classically in polynomial time. Shor's algorithm uses a quantum computer to find period of MEF with polynomial time complexity.

The basic procedure during the implementation of Shor's algorithm is as follows: Let us denote the composite integer to be factored as $I \in \mathbb{Z}^+$

1 Initialization of two registers:

$|0\rangle^n \otimes |0\rangle^r \; I^2 \leq 2^n \leq 2I^2, \; r > \log(I) + 1$

2 Create superposition in 1st register to $N = 2^n$ states: $\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \, |0\rangle$

3 Apply function $f(x) = z^x \bmod I$ for random z co-prime to I ($z\phi I$) on second register:

$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \otimes |f(x)\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \otimes |z^x mod I\rangle$

4 Conceptually measure second register to collapse large superposition to smaller superposition:

$\frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} |x_i + ja\rangle$, where m is frequency of $f(x)$ and a is the period of $f(x)$

5 Apply QFT on 1st register:

$= \frac{1}{N} \sum_y w^{x_0 y} |y\rangle$

6 Measure input register to get frequency : m

7 Use Continued fraction algorithm and GCD to get period a from frequency m

8 Check if a is the period of $f(x) = z^x \bmod I$

9 find factors using period a: $GCD(z^{\frac{a}{2}} \pm 1, I)$

The diagram of the period finding circuit on a quantum computer is as shown in the figure below:

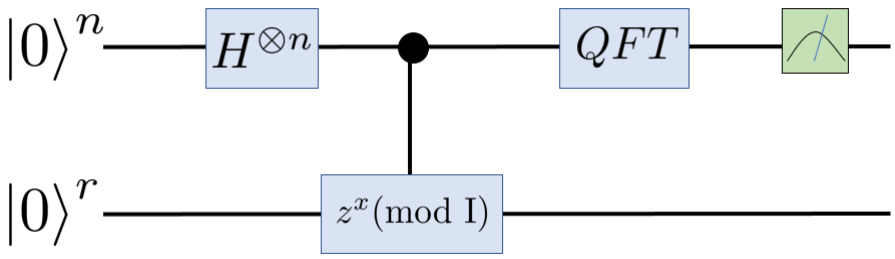


Figure 4.4: Circuit diagram for Shor's period finding subroutine

4.4 Step 1

Initialize two quantum registers: input register($|0\rangle^n$ and period register($|0\rangle^r$) where n and r are size of respective registers. Input register stores values of x that are computed to MEF: $f(x) = z^x \bmod I$ and period register stores values for $f(x)$

We determine value of n such that $I^2 \leq 2^n \leq 2I^2$ [20], an efficient way is setting $n = \lfloor 2 \log I \rfloor$. [4]. Now, since for all z, $x \in \mathbb{Z}_I^+, I \geq f(x)$, we set $r = \lfloor \log(I) \rfloor + 1$

We represent the state of the system by

$$|\Psi_1\rangle = |0\rangle^n |0\rangle^r \tag{4.1}$$

4.4 Step 2

Apply Hadamard gate to 1st register to create a uniform superposition of $N = 2^n$ states from 0 to $N - 1$. This is initializing the parallelism on the circuit. All the superimposed states serve as inputs for $f(x)$

$$|\Psi_2\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle^n |0\rangle^r = \frac{1}{\sqrt{N}} (|0\rangle + |1\rangle + ... + |N-1\rangle) |0\rangle^r \tag{4.2}$$

4.4 Step 3

A random number , $z \in \mathbb{Z}_I^+ : z \phi I$, is chosen classically. Then the function $f(x) = z^x$ mod I is applied to period register.

Here input register is computed to function in the second register. This is achieved by using multiple entangling gates between registers. This step embarks entanglement or co-relation between the two registers.

The state of system is represented mathematically as

$$|\Psi_3\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |f(x)\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle |z^x mod I\rangle$$

As discussed in MEF section 2.5.3, since $f(x)$ is injective periodic function, we can write $|\Psi_3\rangle$ as

$$|\Psi_3\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{a-1} (\sum_{j=0}^{\tilde{m}-1} |x + ja\rangle^n) |f(x)\rangle^r \tag{4.3}$$

4.4 Step 4

Now, we perform Conceptual measurement[1] on the period register which collapses the massive superposition into a smaller one.

We can write

$$|\Psi_3\rangle = \sqrt{\frac{\tilde{m}}{N}} \sum_{x=0}^{a-1} (\frac{1}{\sqrt{\tilde{m}}} \sum_{j=0}^{\tilde{m}-1} |x + ja\rangle) |f(x)\rangle^r$$

Suppose $|f(x_0)\rangle$ be the output of second register upon measurement. This measurement collapses $x \in 0, 1, 2, ..., a-1$ to a single value: x_0(say). So, the state become:

$$|\Psi_4\rangle = \frac{1}{\sqrt{\tilde{m}}} \sum_{j=0}^{\tilde{m}-1} |x_0 + ja\rangle |f(x_0)\rangle$$

We could ignore $|f(x_0)\rangle$ in the equation as it has no effect on the system. So,

$$|\Psi_4\rangle = \frac{1}{\sqrt{\tilde{m}}} \sum_{j=0}^{\tilde{m}-1} |x_0 + ja\rangle \tag{4.4}$$

4.4 Step 5

Next, we apply quantum Fourier transform on the input register. The state becomes

$$|\Psi_5\rangle = QFT |\Psi_4\rangle$$

$$= \frac{1}{\sqrt{\tilde{m}}} \sum_{j=0}^{\tilde{m}-1} \frac{1}{\sqrt{N}} (\sum_{y=0}^{N-1} w_N^{(x_0+ja)y} |y\rangle^n)$$

$$|\Psi_5\rangle = \frac{1}{\sqrt{\tilde{m}N}} \sum_{y=0}^{N-1} w_N^{x_0 y} \sum_{j=0}^{\tilde{m}-1} w_N^{jay} |y\rangle^n \tag{4.5}$$

Since second register has been measured(conceptually), $\tilde{m}$ is definite. Let us define two cases to analyze post QFT remains.

1 $\tilde{m}a = ma = N$

[1]Conceptual measurement is a type of measurement that is just done as a mathematical formulation and is usually optional or avoided in practice. It is done for clear understanding of concept or easiness in theoretical study. Also, such measurement does not have any effect on the following circuit

2 $ma \neq N$

Case I

Case II is observed less frequently than the case I. 2.2

When $ma = N$, we have.

$w_N^{jay} = e^{\frac{2\pi jay}{N}} = e^{\frac{2\pi ijy}{N}} = w_m^{jy}$

and, $\sum_{j=0}^{\tilde{m}-1} w_N^{jay} = \sum_{j=0}^{\tilde{m}-1} w_m^{jy} = \begin{cases} m & \text{if } y = 0(modI) \\ 0 & \text{if } y \neq 0(modI) \end{cases}$

Hence equation(4.5) becomes

$$|\Psi_5\rangle = \frac{1}{\sqrt{mN}} \sum_{y=0}^{N-1} w_N^{x_0 y} m \, |y\rangle = \sqrt{\frac{m}{N}} \sum_{y=0}^{N-1} w_N^{x_0 y} \, |y\rangle$$

Since $y = 0(\text{mod m})$, $y = cm$ for $c = 0, 1, 2, ...(a-1)$

$$|\Psi_5\rangle = \frac{1}{\sqrt{a}} \sum_{c=0}^{a-1} w_N^{x_0 cm} \, |cm\rangle^n \tag{4.6}$$

Case II

We are usually more likely to see case II. It can be considered as the general case and hence, the state is given by the equation(4.5).

4.4 Step 6: Measurement

Now, we measure the post-QFT state to obtain the desired output. Upon measurement, let $C = \{y_c\}_{c=0}^{a-1}$ be the set of output measured with high probability. The goal is to determine periodic frequency m so that we can compute period $a = \frac{N}{m}$

For Case I, the state $|\Psi_5\rangle$ 4.6 collapses to set $\{cm\}_{c=0}^{a-1}$ upon measurement. Hence, for case I, $C = \{y_c\}_{c=0}^{a-1} = \{cm\}_{c=0}^{a-1}$. From this, it seems easy to determine m. But only those $\{cm\}$'s for which c and a are co-prime to each other($c\phi a$) gives nontrivial output. Using Theorem(5) we know that there is more than 60% chance that c obtained at random from set $[0, a-1]$ and a are co-prime, i.e. $P(c\phi a) \geq \frac{1}{2}$

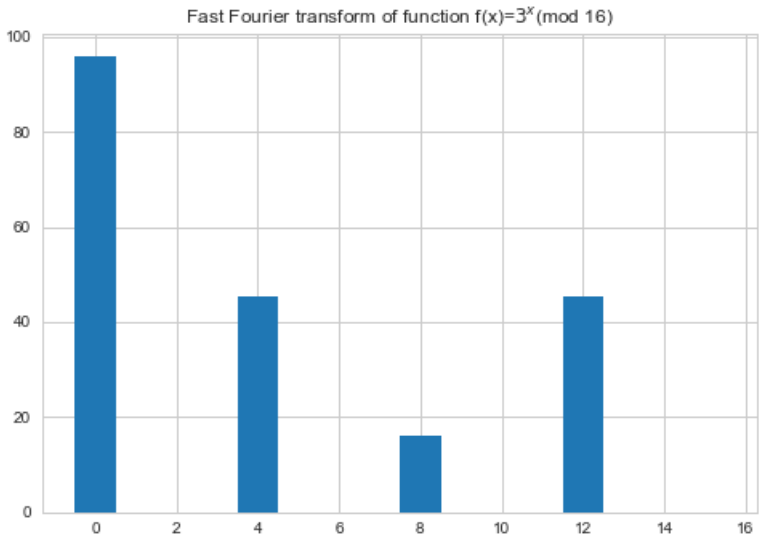


Figure 4.5: Histogram representing discrete Fourier transform on example: 2.1

Hence, it is assured that a nontrivial output is obtained more than half the time. Then we obtain the periodic frequency m by applying the Euclidean algorithm on the set of the non-trivial $\{cm\}$'s.

Analysis of Case II on measurement is more complex than Case I because the state$|\Psi_5\rangle$ (4.5) does not have an integer frequency: $\frac{N}{a}$ as period a does not divide N exactly. Consider the example 2.10(b), we can see the function does not complete its full cycle at N. Discrete Fourier transform on the data is shown in the figure(4.6). It shows multiple points with higher probabilities but does not seems to be multiple of any integer like for case I.

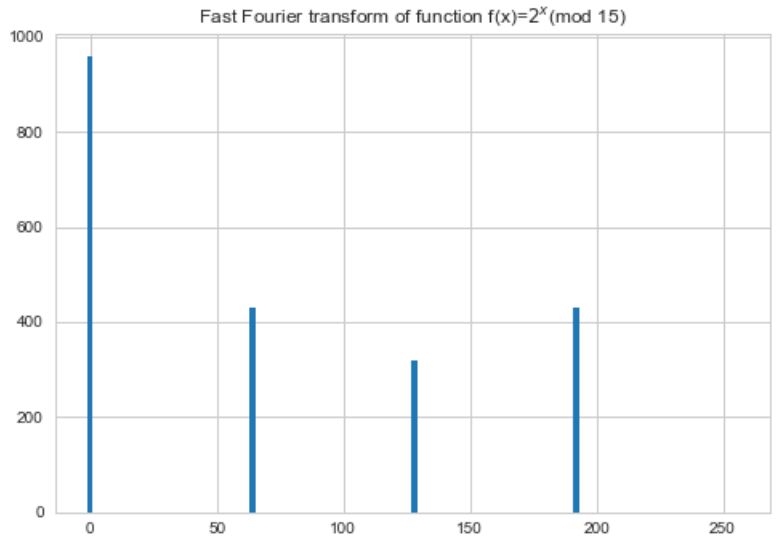


Figure 4.6: Histogram representing discrete Fourier transform on example: 2.2

Let us redefine the set $C = \{y_c\}_{c=0}^{a-1}$ for Case II as

$$ay_c \in [cN - \frac{a}{2}, cN + \frac{a}{2}] \tag{4.7}$$

$$-\frac{a}{2} \leq ay_c - cN < \frac{a}{2} \Rightarrow |\frac{y_c}{N} - \frac{c}{a}| \leq \frac{1}{2N} \tag{4.8}$$

$$-\frac{a}{2} \leq \hat{y_c} < \frac{a}{2} \text{ where } \hat{y_c} = ay_c - cN \tag{4.9}$$

The set $C = \{y_c\}_{c=0}^{a-1}$ is the set of outputs with maximum likelihood upon measurement. The condition above is applied so that the each y_c is expected to converge to $\{cm\}_{c=0}^{a-1}$. We will discuss the relevance of the condition 4.7 in the probability section that the set C is the set with maximum likelihood.

4.4 Step: 7

We have measured outputs from the quantum computers and now, we analyze the outputs in a classical computer. We use continued fraction algorithm(CFA) and Euclidean Algorithm(EA) on output set $C = \{y_c\}_{c=0}^{a-1}$ to get the period: a.

Lemma 4. *From the set* $\{y_c\}_{c=0}^{a-1}$, $\frac{c}{a}$ *is in the nearest neighbourhood of* $\frac{y_c}{N}$ *for any* $0 < c < a - 1$.

Proof. We know,

$$\left|\frac{c+1}{a} - \frac{c}{a}\right| = \left|\frac{\pm 1}{a}\right| \geq \frac{1}{a^2} > \frac{1}{I^2}$$

Also we have, $I^2 < N$ and from equation 4.7,

$$\left|\frac{y_c}{N} - \frac{c}{a}\right| \leq \frac{1}{2N} \leq \frac{1}{2I^2} \leq \frac{1}{I^2} \tag{4.10}$$

So, it is clearly seen that $\frac{c}{a}$ is in vicinity of $\frac{y_c}{N}$ around $\frac{1}{I^2}$, while $\frac{c+1}{a}$ lie outside the vicinity. This proves that "$\frac{c}{a}$ is uniquely close to $\frac{y_c}{N}$."(Loceff, 2015)[20] □

Let $R = \frac{y_c}{N}$ and since $a < N$, from equation 4.10,

$$\left|\frac{y_c}{N} - \frac{c}{a}\right| \leq \frac{1}{2I^2}$$

$$\left|R - \frac{c}{a}\right| \leq \frac{1}{2I^2} \leq \frac{1}{2a^2} \tag{4.11}$$

From lemma(3), $\frac{c}{a}$ lies on the convergent sequence of continued fraction for $\frac{y_c}{N}$, given c and a are relatively prime to each other. We will discuss there is a decent probability that c and a are relatively prime later in the study. Since, lemma(4) suggests $\frac{c}{a}$ is uniquely close to $\frac{y_c}{N}$, we apply continued fraction algorithm 1 on a set $R = \frac{y_c}{N}$ with error $\frac{a}{2I^2}$.

Claim 2. Continued fraction Algorithm on $\frac{y_c}{N}$ with error $\frac{a}{2I^2}$ outputs unique $\frac{c}{a}$.

Proof. Since $\frac{c}{a}$ is convergent of $\frac{y_c}{N}$ and within the neighbourhood of $\frac{1}{2I^2}$, the output of CFA would be an convergent $\frac{n_k}{d_k} \leq \frac{c}{a}$.

If CFA outputs a convergent: $\frac{n_k}{d_k} < \frac{c}{a}$, it means,

$$\left|R - \frac{n_k}{d_k}\right| \leq \frac{1}{2I^2}$$

But, d_k is strictly increasing according to the properties of convergent of continued fraction and error limit is $E < \frac{1}{2I^2}$.

We have, equation(4.11):

$$\left|R - \frac{c}{a}\right| \leq \frac{1}{2I^2}$$

$\frac{c}{a}$ exactly fits the condition. Hence, $\frac{c}{a}$ must be the output of the Algorithm. □

Since CFA has complexity $O(log^3 N)$, We computed an set $\frac{c}{a}$ from $\{y_c\}_{c=0}^{a-1}$ with $O(log^3 N)$ complexity. But only those $\frac{c}{a}$ produces nontrivial output for which c and a are co-prime to each other.

Now, we apply Euclidean algorithm on such set $\frac{c}{a}\rfloor_{(c\phi a)}$ to get a value $\frac{1}{a}$. Reciprocal of the value gives the period of the function $f(x)$.

4.4 Step 8 and 9

We would check if the calculated period: a is correct. If it is wrong, we would rerun the algorithm for different random z. There is a bounded probability that a correct period would be found.

Then, we would determine the factors classically by applying Euclidean Algorithm: $GCD(z^{\frac{a}{2}} + 1, N)$ and $GCD(z^{\frac{a}{2}} - 1, N)$

4.5 Probability of Success

We have discussed the steps to determine the factors of a number. But we have introduced the likelihood of happening of certain events. Also, the probabilistic nature of quantum computers causes the result to be expressed in terms of probability. So, as we have discussed, it is not certain that we always obtain the factors. In this section, we will discuss the probabilities involved in the steps during the implementation.

We could enlist the areas for involvement of probabilities as:

1 Selection of z co-prime with I

2 Measurement of y_c's satisfying condition(4.8)

3 Measurement of y_c's with c co-prime to a

4.5 Measurement of y_c's satisfying condition(4.8)

From step 6, it is evident that for case I: $ma = N$, we obtain each y_c with probability $\frac{1}{a}$. Hence, there is a high likelihood of measuring y_c for case I.

For case II, we have stated earlier that there is a high likelihood that we obtain y_c's among other y upon measuring the input register. We will prove the statement is true and discuss its relevance.

From step 7, we have equation (4.5):

$$|\Psi_5\rangle = \frac{1}{\sqrt{\tilde{m}N}} \sum_{y=0}^{N-1} w_N^{x_0 y} \sum_{j=0}^{\tilde{m}-1} w_N^{jay} |y\rangle^n \tag{4.12}$$

Upon measurement, the probability for measuring $|y\rangle$ is

$$P(y) = \left| \frac{1}{\sqrt{\tilde{m}N}} w_N^{x_0 y} \sum_{j=0}^{\tilde{m}-1} w_N^{jay} \right|^2$$

Here, $\sum_{j=0}^{\tilde{m}-1} w_N^{jay}$ is a geometric series,

$$\sum_{j=0}^{\tilde{m}-1} w_N^{jay} = \left| \frac{1-(w_N^{jay})^{\tilde{m}}}{1-w_N^{jay}} \right| = \left| \frac{1-e^{\frac{2\pi i a y \tilde{m}}{N}}}{1-e^{\frac{2\pi i a y}{N}}} \right|$$

The probability of measuring y_c satisfying equation (4.8) is

$$P(y_c) = \frac{1}{\tilde{m}N} \left| \frac{1-e^{\frac{2\pi i a y_c \tilde{m}}{N}}}{1-e^{\frac{2\pi i a y_c}{N}}} \right|^2$$

Also from 4.9, $\hat{y_c} = a y_c - cN$. Expressing the above equation in-term of $\hat{y_c}$:

$$P(y_c) = \frac{1}{\tilde{m}N} \left| \frac{1-e^{\frac{2\pi i \hat{y_c} \tilde{m}}{N}}}{1-e^{\frac{2\pi i \hat{y_c}}{N}}} \right|^2 = \frac{1}{\tilde{m}N} \left| \frac{1-e^{i\alpha\tilde{m}}}{1-e^{i\alpha}} \right|^2 \because \alpha = \frac{2\pi i \hat{y}_c}{N} \tag{4.13}$$

Since y_c's are the outputs that have a high significance in our problem, our objective is to determine the lower bound of $P(y_c)$ so that it is explicit for how many times the quantum circuit needs to be executed before getting the required result. For that, we determine the lower bound of the numerator and upper bound of the denominator in $P(y_c)$.

From Euler's identity, we have,

$$1 - e^{i\phi} = 1 - (cos\phi + isin\phi)$$

$$= 2sin\frac{\phi}{2}(cos\frac{\phi}{2} - isin\frac{\phi}{2})$$

$$\Rightarrow \left|1 - e^{i\phi}\right| = \left|2sin\frac{\phi}{2}\right|$$

Upper Bound for denominator

We know, $|sin\phi| \leq |\phi|$. This statement can be clearly visualized graphically by the figure 4.7. We can clearly see the line $y = |\phi|$ is clearly above curve $y = |sin\phi|$

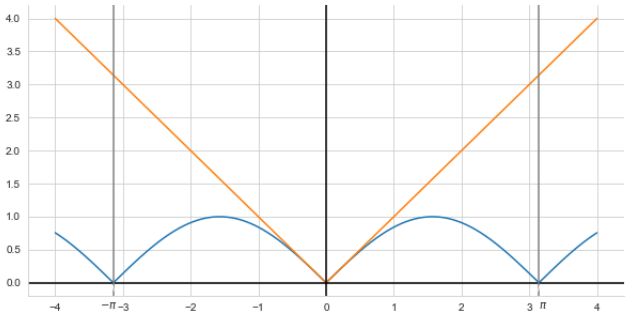


Figure 4.7: Graph of curves $|sin\phi|$ and $|\phi|$

Hence,

$$\left|1 - e^{i\alpha}\right| = 2\left|sin\frac{\alpha}{2}\right| \leq |\alpha|$$

The upper bound of the denominator is $|\alpha|$.

Lower bound of numerator

The numerator of the fraction has a term $\tilde{m}$ on it whose value is could be m or $m+1$ but distinct. We need to define the lower bound for each case and determine the lowest.

(a). $\tilde{m} = m$

We intend to find the lower bound of term $\left|1 - e^{i\alpha\tilde{m}}\right| = 2\left|sin\frac{\alpha}{2}\right|$

From the definition of $\hat{y_c}$(4.9), we have,

$$-\frac{a}{2} < \hat{y_c} < \frac{a}{2}$$

$$\Rightarrow -\frac{\pi am}{N} < \frac{2\pi am\hat{y_c}}{N} < \frac{\pi am}{N}$$

Since $\frac{am}{N} < 1$, $-\frac{\pi am}{N} < -\pi < \frac{2\pi am\hat{y_c}}{N} < \frac{\pi am}{N} < \pi$

$$\Rightarrow -\pi < \frac{2\pi am\hat{y_c}}{N} = \alpha m < \pi$$

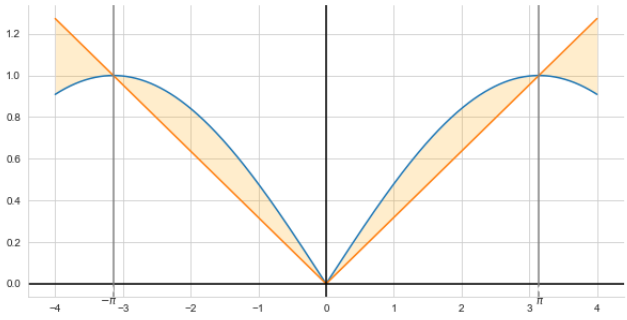


Figure 4.8: Graph of curves $\left|sin\frac{\phi}{2}\right|$ and $\left|\frac{\phi}{\pi}\right|$

The graph on the figure (4.8 shows the plot of curve $y = \left|sin\frac{\phi}{2}\right|$ and a line $y = \left|\frac{\phi}{\pi}\right|$. For a limit $-\pi < \phi < \pi$, the graph shows $\left|sin\frac{\phi}{2}\right| > \left|\frac{\phi}{\pi}\right|$. So, the lower bound of $\left|a - e^{i\phi}\right|$ in interval $[-\pi/2, \pi/2]$ is $\left|\frac{\phi}{\pi}\right|$

Hence, we can write,

$$\left|1 - e^{i\alpha m}\right| = 2\left|sin\frac{\alpha m}{2}\right| \geq 2\left|\frac{\alpha m}{\pi}\right|$$

It can also be proven analytically using calculus.

Then, the minimum likelihood of obtaining y_c using equation(4.13):

$$P(y_c)_{\tilde{m}=m} = \frac{1}{mN}\left|\frac{1 - e^{i\alpha\tilde{m}}}{1 - e^{i\alpha}}\right|^2 > \frac{1}{mN}\left|\frac{2\left|\frac{\alpha m}{\pi}\right|}{|\alpha|}\right|^2$$

$$P(y_c)_{\tilde{m}=m} \leq \frac{4m}{\pi^2 N} \tag{4.14}$$

(b). $\tilde{m} = m + 1$

For this sub-case, we follow the same steps with some minor changes. We had,

$$-\frac{a}{2} < \hat{y_c} < \frac{a}{2}$$

$$\Rightarrow -\frac{\pi a(m+1)}{N} < \frac{2\pi am\hat{y_c}}{N} < \frac{\pi a(m+1)}{N}$$

Since $\frac{a(m+1)}{N} < 1$ and $\frac{a}{N} < \frac{1}{2}$,

$$-\frac{\pi a(m+1)}{N} < \frac{-3\pi}{2} < \frac{2\pi a(m+1)\hat{y_c}}{N} < \frac{\pi a(m+1)}{N} < \frac{3\pi}{2}$$

$$\Rightarrow \frac{-3\pi}{2} < \frac{2\pi a(m+1)\hat{y_c}}{N} = \alpha(m+1) < \frac{3\pi}{2}$$

Figure 4.9: Graph of curves $\left|sin\frac{\phi}{2}\right|$ and $\left|k\frac{\phi}{\pi}\right|$, $k = \frac{2}{3}sin\frac{3\pi}{4}$

Here,in the graph, curve $\left|sin\frac{\phi}{2}\right|$ exactly bound a line $\left|\frac{k\phi}{\pi}\right|$ where $k = \frac{2}{3}sin\frac{3\pi}{4}$ in the interval $[-3\pi/2, 3\pi/2]$. In contrast to the previous sub-case, a constant k is multiplied to the term $\left|\frac{\phi}{\pi}\right|$ for balancing. We can clearly state $\left|sin\frac{\phi}{2}\right| > \left|k\frac{\phi}{\pi}\right|$ where $k = \frac{2}{3}sin\frac{3\pi}{4}$. So, the lower bound of $\left|1 - e^{i\alpha(m+1)}\right|$ in interval $[-3\pi/2, 3\pi/2]$ is $\left|k\frac{\alpha(m+1)}{\pi}\right|$

Hence, the minimum likelihood of obtaining y_c is

$$P(y_c)_{\tilde{m}=m+1} = \frac{1}{(m+1)N}\left|\frac{1-e^{i\alpha(m+1)}}{1-e^{i\alpha}}\right|^2 > \frac{1}{(m+1)N}\left|\frac{2\left|\frac{k^2\alpha(m+1)k^2}{\pi}\right|}{|\alpha|}\right|^2$$

$$P(y_c)_{\tilde{m}=(m+1)} \geq \frac{(4(m+1)k^2)}{\pi^2 N} \tag{4.15}$$

We can generalize the minimum likelihood of measuring y_c for cases $\tilde{m} = m$ or $m+1$ as

$$P(y_c)_{\tilde{m}} \geq \frac{4\tilde{m}k^2}{\pi^2 N} \tag{4.16}$$

But there are 'a' numb such $\{y_c\}$'s obtained after measurement: $y_1, y_2, \ldots, y_{a-1}$. So minimum probability of measuring any one of the $\{y_c\}'s$ is

$$P(y_c)_{\tilde{m}} \geq \frac{(4a\tilde{m}k^2)}{\pi^2 N} \tag{4.17}$$

Let us study the worst-case scenario in obtaining any one of $\{y_c\}'s$. We have assumed a condition: $2a < N$ earlier in the study. The worst-case for the minimum likelihood of measuring y_c is when $3a > N$. For minimum case, this implies $\frac{am}{N} > \frac{1}{2}$ and $\frac{a(m+1)}{N} > 1$

$$P(y_c)_{\tilde{m}} \geq \frac{4k^2}{\pi^2}\frac{a(m+1)}{N} > \frac{4k^2}{\pi^2} = 0.090$$

But typically, we have $a << I < N$, so $\frac{a\tilde{m}k^2}{N} \equiv 1$. Then the inequality becomes:

$$P(y_c)_{\tilde{m}} \geq \frac{4k^2}{\pi^2}\frac{a\tilde{m}}{N} > \frac{4}{\pi^2} = 0.405$$

Hence, the probability of measuring any one of $\{y_c\}'s$ is

$$P(y_c)^{min} = \begin{cases} 0.090 & \text{For worst case} \\ 0.405 & \text{typically} \end{cases}$$

Now, let us compute the maximum likelihood of obtaining y_c. The upper bound of the numerator could be easily determined using the relations defined above. We know,

$$\left|a = e^{i\alpha\tilde{m}}\right| = 2sin(\frac{\alpha}{2}) \leq \alpha\tilde{m}$$

However, we need to determine new limit for the lower bound of denominator. We

have,

$$-\frac{a}{2} < \hat{y_c} < \frac{a}{2}$$

$$\Rightarrow -\frac{\pi a}{N} < \frac{2\pi a\hat{y_c}}{N} < \frac{\pi a}{N}$$

Since $\frac{a}{N} < \frac{1}{2}$,

$$-\frac{\pi a}{N} < \frac{\pi}{2} < \frac{2\pi a\hat{y_c}}{N} < \frac{\pi a}{N} < \frac{\pi}{2}$$

$$\Rightarrow \frac{\pi}{2} < \alpha < \frac{\pi}{2}$$

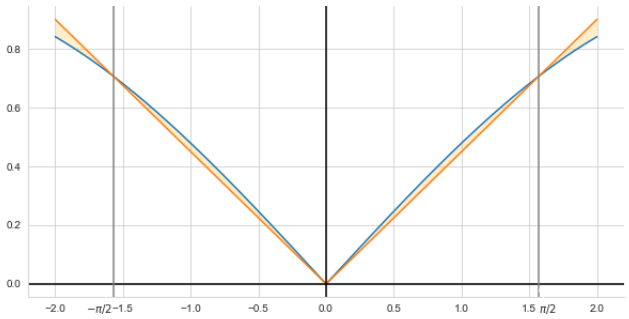


Figure 4.10: Graph of curves $\left|sin\frac{\phi}{2}\right|$ and $\left|k\frac{\phi}{\pi}\right|$, $k = 2sin\frac{\pi}{4}$

Here, in the graph, curve $\left|sin\frac{\phi}{2}\right|$ exactly bound a line $\left|\frac{l\phi}{\pi}\right|$ where $l = 2sin\frac{\pi}{4}$ in the interval $[-\pi/2, \pi/2]$. In contrast to the previous sub-case, a constant k is multiplied to the term $\left|\frac{\phi}{\pi}\right|$ for balancing. We can clearly state $\left|sin\frac{\phi}{2}\right| > \left|l\frac{\phi}{\pi}\right|$ where $l = 2sin\frac{3\pi}{4}$. So, the lower bound of $|1 - e^{i\alpha}|$ in interval $[-\pi/2, \pi/2]$ is $\left|l\frac{\alpha}{\pi}\right|$.

Hence, the maximum likelihood for obtaining y_c is

$$P(y_c)^{max} \leq \frac{1}{\tilde{m}N}\left|\frac{2\alpha\tilde{m}}{\pi l\alpha}\right|^2 = \frac{4(m+1)}{\pi^2 l^2 N}$$

,$\tilde{m} = m + 1$ for maximum likelihood.

Now, as a metric of the likelihood for not measuring y_c, we determine the ratio of minimum and the maximum likelihood of y_c as

$$\mathcal{P}_{ratio} = \frac{P(y_c)^{min}}{P(y_c)^{max}} \geq \frac{\frac{4mk^2}{\pi^2 N}}{\frac{4(m+1)l^2}{\pi^2 N}} = \frac{m}{m+1}\frac{k^2}{l^2} \geq \frac{2}{3}\frac{k^2}{l^2} \tag{4.18}$$

We know,for large I and N, $\frac{m}{m+1} \approx 1$ and $k, l \approx 1$. So, $\mathcal{P}_{ratio}$ can be expressed as:

$$\mathcal{P}_{ratio} > \begin{cases} 0.72 & \text{worst case} \\ 1-\epsilon & \text{otherwise} \end{cases}$$

ϵ is the error limit.

4.5 Measurement of y_c's with c co-prime to a

We measure y_c's with a certain high probability upon a set of measurement. We have described that we use continued fraction algorithm on the set $\frac{y_c}{N}$ to determine a set $\frac{c}{a}$. But out of this set $\frac{c}{a}$, only these elements that satisfy condition $c\phi a$, i.e c and a are co-prime, are non-trivial. In this subjection, we will discuss the probability of randomly selected c to satisfy the condition $c\phi a$.

We can say, the probability $P(c\phi a)$ is the probability that there is no common prime factor of a and c. $P(c\phi a) = P$(no common factor between a and c) Let $2, 3, 5, 7, ...p_k$ be the set of prime numbers less than a, then,

$P(c\phi a) = P(\neg(2 \mid c \wedge 2 \mid a) \wedge (3 \mid c \wedge 3 \mid a) ... \wedge (p_k \mid c \wedge p_k \mid a))$

$$= \Pi_{p_i=primes}^{p_k<a} P(\neg(p_i \mid c \wedge p_i \mid a))$$

And we know, $P(p \mid c) < \frac{1}{p}$ and $P(p \mid a) < \frac{1}{p}$

Then $P(\neg(p \mid c) \wedge (p \mid a)) \geq 1 - \frac{1}{p^2}$

Hence, $P(c\phi a) > \Pi_{p_i=primes}^{p_k<a}(1 - \frac{1}{p^2})$

This looks similar to the Riemann Zeta function in Euler's product form.

Riemann Zeta function is a special type of Dirichlet series analytic in domain in complex plane c given by $\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$,where Real part of s, $Re(s) > 1$. The function converges when $Re(s) > 1$.

A special case of Riemann zeta function as $s = 2$ is of interest to us.

$$\zeta(1) = \frac{\pi^2}{6} = \sum_{n=1}^{\infty} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + ...$$

It is a very popular identity of Pi and can be proved (Appendix) easily.

Lemma 5. *Riemann Zeta function,* $\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$, $Re(s) > 0$ *can be written in form*

$$\zeta(s) = \Pi_{n=1}^{\infty} \frac{1}{1 - \frac{1}{p_n^s}}$$

where p_n *is the sequence of all the primes.*

Using Theorem (5)

$$\Pi_{n=1}^{\infty}(1 - \frac{1}{p^2}) = \frac{1}{\zeta(2) = \frac{6}{\pi^2}}$$

$$P(c\phi a) > \frac{6}{\pi^2} \approx 0.607$$

Thus there is 60% likelihood that we obtain non-trivial function $\frac{c}{a}$.

4.5 Finalizing Probability

In this subsection, we shall integrate all the probability to see the probability of success of the algorithm.

We have, the set of high probability measurements $c = y_{c_{c=0}}^{a=1}$. Let us define a new set

$$B = y_p : y_p \in C \wedge (b\phi a)$$

$$B \leq C$$

Let $|C|$ and $|B|$ be the size of each set C and B respectively.

The ratio of size of the sets $\frac{|B|}{|C|}$ gives the probability of $P(b\phi a)$. So,

$$\frac{|B|}{|C|} = P(b\phi a) > 0.607$$

Here our goal is to determine probability of obtaining the set B i.e $P(B)$

$$P(B) = P(B \mid C).P(C)$$

$$= \frac{\sum_{b \in B} P(y_b)}{\sum_{c \in C} P(y_c)}.P(y_c)$$

Let $P(y_b^{min})$ and $P(y_c^{max})$ be minimum likelihood of obtaining y_b and maximum likelihood of obtaining y_c.

Then, $P(B) > \frac{|B|}{|C|}\frac{P(y_b^{min})}{P(y_c^{max})}.P(y_c)$

For worst case, $P(B) > 0.607 \times 0.72 \times 0.405(\frac{6}{\pi^2} \times 0.72 \times \frac{4k^2}{\pi^2}) \approx 0.177$

For typical case, $P(B) > 0.607 \times 1 \times 0.405 \approx 0.246$

Hence, the probability of failure in this algorithm is bounded. Shor's algorithm is called the bounded error algorithm for this reason.

A constant time algorithm is an algorithm that completes after a fixed number of trials of loops, T where T is independent of the size of the algorithm N

Theorem 3 (Constant Time Complexity theorem(CTC theorem)). *If a probabilistic, looping algorithm having size N, has a bounded non-zero probability of success for a single loop, independent of N, then it is a constant time algorithm. [20](loceff, 2015)*

$P(success) = p > 0$, *p is independent of N*

We can see, this period finding subroutine of Shor's algorithm satisfies the necessary condition of the CTC theorem(3). So, it is a constant time algorithm. The number of trials for a constant time algorithm with error tolerance ϵ and probability of success of single-trial p is given by:

$$T = \left\lfloor \frac{log(\epsilon)}{log(1-p)} \right\rfloor + 1$$

Thus, the period finding subroutine for a certain 'z' is likely to output period 'a' in

$$T = \left\lfloor \frac{log(\epsilon)}{log(1-P(B)} \right\rfloor + 1$$

4.6 Complexity of Algorithm

Now let us discuss the complexity of the algorithm. We can divide the algorithm into 5 parts: Hadamard gates, QFT, oracle operation for $f(x)$, CFA (continued fraction algorithm), and other classical subroutines.

We have computed most of the section of the algorithm on the n qubit system. So, the complexity obtained will be expressed in terms of N or n. We aim to express the complexity in terms of the integer I.

We have,

$$\frac{N}{2} \leq I^2 \leq N$$

$$log(N) - log(2) \leq 2log(I) \leq logN$$

$$\Rightarrow \mathcal{O}(log(N)) \leq \mathcal{O}(I) \leq \mathcal{O}$$

Hence, $\mathcal{O}(N) = \mathcal{O}(I)$ is the relation between the Big $\mathcal{O}$ operation on I and N. Further on, we will replace $\mathcal{O}(N)$ by $\mathcal{O}(I)$ for consistency and generalization.

The complexity of Hadamard gates on n qubits is $\mathcal{O}(log(N)) = \mathcal{O}(log(I))$ and QFT is $\mathcal{O}(log^2(I))$. We had discussed CFA has complexity of $\mathcal{O}(log^3(I))$. Other classical operations include obtaining GCD whose complexity is less than $\mathcal{O}(log^3(I))$. So they can be ignored during complexity computation. Determination of complexity of oracle operation needs thorough analysis of the function f(x).

The complexity for oracle operation of function $f(x)$ is same as the complexity for determining $f(x)$.

We have,

$$f(x) = z^x mod I$$

We need to compute the term z^x. let us express x n binary form as $x = \sum_k x_k 2^k$. Since, $x < n$, we can write $x = \sum_{k=0}^{n-1} x_k 2^k$. So,

$$z^x = z^{\sum_{k=0}^{n-1} x_k 2^k} = \Pi_{k=0}^{n-1} z^{x_k 2^k}$$

Subroutine	Complexity
Hadamard gates	$\mathcal{O}(log(I))$
QFT	$\mathcal{O}(log^2(I))$
Oracle function	$\mathcal{O}(log^3(I))$
CFA	$\mathcal{O}(log^3(I))$
Classical operations	$\mathcal{O}(log^3(I))$

Table 4.1: Complexity of processes in Shor's algorithm

An efficient way to compute the set $\{2^k\}_{k=0}^{n-1}$ is to perform repeated squaring z till the power is $(n-1)$ as $2^j = (2^{j-1})^2$. This process only requires n-2 steps. Also multiplication by x_k is a decision problem as $x_k = \{0, 1\}$ and has constant complexity. Since, each multiplication operation is $\mathcal{O}(log^2(N))$, by adding $\mathcal{O}(log(N))$ operation, we get the set $\{2^k\}_{k=0}^{n-1}$ by $\mathcal{O}(log^3(N))$ operations. This operation is done classically.

Now, we use set $\{2^k\}_{k=0}^{n-1}$ to determine z^x by $n = log(N)$ product operations each of $\mathcal{O}(log^2(N))$ on quantum computer followed by modulo operation. So, the total operations is $\mathcal{O}(log^3(N))$. Hence, oracle operation is $\mathcal{O}(log^3(I))$.

Thus, the subroutines of Shor's factoring algorithm with their complexity is tabulated as:

Since each subroutine is independent of others, the complexity of the algorithm is $\mathcal{O}(log^3(I))$. Hence we can conclude that Shor's algorithm is a quantum algorithm, capable of factoring integers in polynomial complexity with bounded error probabilities. Thus Shor's algorithm is a BQP(bounded-error quantum polynomial) algorithm.

Chapter5: Application

5.1 Threat to RSA cryptography

Securing information transmission and communication by scrambling(encryption) and unscrambling(decryption) the information or signal using a key(cipher) is cryptography. It aims in reducing the possibility of interception of a message by an unintended party and maintain the confidentiality of the message. Private key cryptography and public-key cryptography are some popular categories of cryptography. RSA cryptography is a public-key cryptosystem. RSA cryptosystem is one of the first public-key cryptographic systems. It was developed by Ronald Rivest, Adi Shamir, and Leonard Adleman in 1977.[4]It is based on the difficulty of the prime factorization problem. This cryptosystem is used in different web browsers, emails, banking transactions, and for secured communication. It is implemented in many cryptographic libraries like OpenSSL, wolfCrypt, and others.

It uses a large composite integer I with prime decomposition $I = p.q$. A public key composed of the integer I and a number a: $GCD(I, (p-1)(q-1)) = 1$ is shared as between Alice and Bob over a public channel. Alice uses the public key(a, I) to encrypt the message as follows:
Say 'm' be the message(binary) and s be the encrypted message.

$$s = m^a mod\, I$$

Bob generates a private key using public keys and factors of I as

$$p = a.bmod(p-1)(q-1)$$

Bob uses the private key: K=(b, N) to decrypt the message.

$$m = c^b mod\, I$$

As we can see, anybody can encrypt the message using the publicly available

public key. Whereas only Bob who has the private key can only decrypt the message, given the integer I is large enough.

Hence, the RSA cryptosystem solely relies on the assumption that large integer factorization is not computationally feasible over some large time. As we have discussed earlier, factoring a large number is very difficult. The largest RSA number to be factored till now is RSA-250, which is a 250 digit long number with 829 bits. It took roughly 2700 core-years, using Intel Xeon Gold 6130 CPUs.[11] Almost all currently used RSA cryptosystem uses 1024 bit or larger numbers like RSA-2048 and RSA-4096.

Shor's algorithm factors pose a threat to RSA cryptosystem since it factors integers in polynomial time with bounded probability. Researchers argue a 2018 bit key that takes 300 trillion years for a classical computer to break, can be broken by a 4099 qubit quantum computer in about 10 seconds.(Zhang et al., 2020) [34]. Hence if such a quantum computer exists, it would likely deem information security involving RSA cryptosystem useless. Information like emails and banking transactions would not seem to be as secure as it is.

Chapter6: Method of implementation

Following methods are used to implement Shor's algorithm to factor an integer.

6.1 Hardware

Since Shor's algorithm is a quantum algorithm, a quantum computer is necessary to implement an example of the factorization of an integer. Fortunately, IBM Quantum allows some of its prototype quantum computers to be used by the public for research and other applications via its interface platform IBM Quantum Experience(ibmq).[9] IBM Quantum experience is a cloud-based platform that allows users to send quantum algorithm or program as a job for it to be executed on the quantum computer at IBM. Quantum computers at IBM are housed with superconducting qubits composed of coupled Josephson junction, also known as transmon qubits. [14]. IBM classify their quantum computer into open and premium system.[13] They allow the use of open quantum systems via backends at IBM Quantum for enthusiasts and researchers. At the time of writing, there are six open quantum systems available as given in the table below.

Name	Qubits
ibmq_santiago	5
ibmq_athens	5
ibmq_16_melbourne	15
ibmq_armonk	1

Table 6.1: Open Quantum systems at IBM available to public

In this study, the program or circuit for Shor's algorithm is executed on a simulator and a quantum so that comparisons can be made. Following devices were used to execute the circuit:[13]

- *ibmq_qasm_simulator*
- *ibmq*_16_*melbourne*

6.1 *ibmq_qasm_simulator*

ibmq_qasm_simulator is a cloud quantum simulator based at IBM quantum. It is a high-performance simulator that executes a quantum circuit written on QASM(Quantum Assembly Language). As the name suggests this simulator is not based on the quantum behavior of material but on the mathematical formulation of quantum computation. Hence they produce error-free outputs at the expense of an increase in complexity. The basic gates it uses are "U1, U2, U3, U, P, R, RX, RY, RZ, ID, X, Y, Z, H, S, SDG, SX, T, TDG, SWAP, CX, CY, CZ, CSX, CP, CU1, CU2, CU3, RXX, RYY, RZZ, RZX, CCX, CSWAP, MCX, MCY, MCZ, MCSX, MCP, MCU1, MCU2, MCU3, MCRX, MCRY, MCRZ, MCR, MCSWAP, UNITARY, DIAGONAL, MULTIPLEXER, INITIALIZE, KRAUS, ROERROR, DELAY"[29]. The version of qasm simulator used for the study is 0.1.547.

ibmq_qasm_simulator is used for the study rather than a local simulator to maintain consistency of output measurements in term of randomness or correctness of the output

6.1 *ibmq_16_melbourne*

'*ibmq_16_melbourne*' quantum system was used for the implementation because it was the only public system with enough qubits to run our circuit. *ibmq_16_melbourne* is a 15 qubits quantum system with quantum volume 8. It uses CX, ID, RZ, SX, and X as a basic set of gates with an average CNOT error: 0.03348 and an average readout error: 0.07161.[29] In a physical system, a coupling map is used to select a set of required qubits with the least CNOT error between all qubits. The coupling map of *ibmq_16_melbourne* is as shown in figure(6.1).

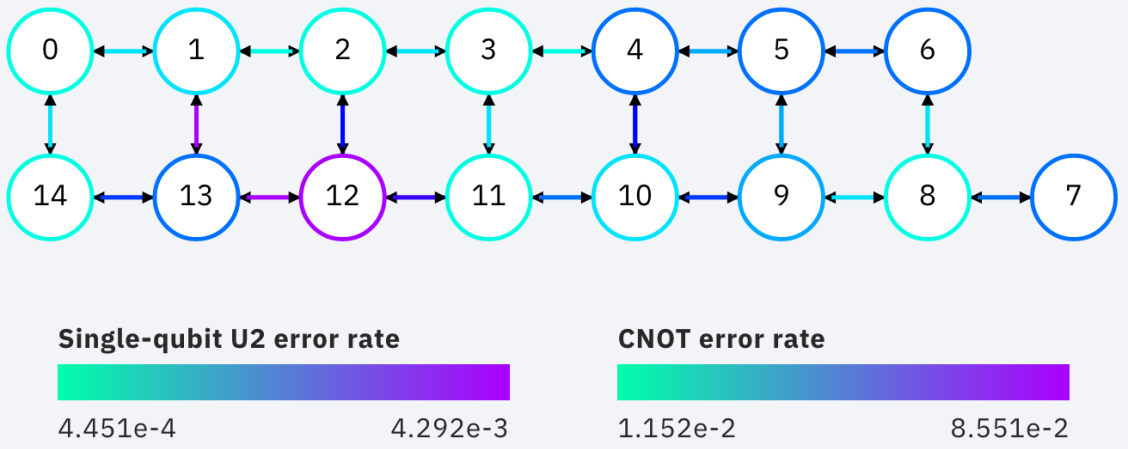


Figure 6.1: Topology diagram and coupling map of *ibmq_16_melbourne*

The figure shows the different single-qubit error for different qubits and different CNOT gate errors for each coupling. The selection of qubits to use for the execution of circuit should be done so as to reduce the error in the circuit. Implementation of the circuit in this study requires 6 qubits for execution.Hence, qubits 0,1,2,3,13,14 were chosen for execution.

6.2 Software

Python programming language, primarily one of its packages: QISKIT was used for our study and jupyter notebook was used as the text editor. QISKIT is an open-source software development kit that allows designing and run quantum circuits and working with quantum algorithms.[25] It has local simulators prepacked with it to run circuits. Also, it creates an interface between a local computer with IBM quantum to run on quantum systems or simulators provided by IBM. QISKIT has four main components: *Terra*, *Ignis*, *Aqua*, and *Aer* based on their purpose.[30] *Terra* deals with the foundation of quantum programs like quantum circuits, gate control, pulse control, and others. *Ignis* deals with errors and noises in the system and error correction. *Aqua* aims at the application of the quantum system for applications like optimization, chemistry. *Aer* includes high-performance simulators emulators.

QISKIT has a feature for visualization of circuits and outputs or results. Visual-

ization of circuits makes the understanding of workings of circuit at hardware level. Result visualization helps in better representation and faster understanding. It has built-in visualization tools to generate useful plots like histograms, Bloch sphere, qsphere of the results.

The version of QISKIT we worked on is

$\{'qiskit-terra' :' 0.16.2',$

$'qiskit-aer' :' 0.7.3',$

$'qiskit-ignis' :' 0.5.1',$

$'qiskit-ibmq-provider' :' 0.11.1',$

$'qiskit-aqua' :' 0.8.1',$

$'qiskit' :' 0.23.3'\}$

6.3 Circuit Design

Due to the limitation of qubits and depth of circuit, we implement a simplified compiled version of Shor's algorithm to factor a small number I=15 with base z=2 for MEF. The simplified complied version of the algorithm is inspired by a paper by [15]. This paper suggests a low depth circuit with less number of qubits for implementation of the algorithm for composite integers composed of Fermat primes[1]. However, this algorithm requires pre-knowing the order of the required Modular exponentiation function.

In the algorithm itself, the most difficult task: modular exponentiation is compiled into a set of few gates. For our case, the MEF gives a set $\{1, 2, 3, 4\}$. Their binary equivalence can be encoded in two qubits. hence we set the period register of two qubits. For the input register, we set the number of qubits given by the equation: $n = \lfloor log_2(15) \rfloor = 4$. To store the output, we add a 4 qubit classical register. First, Hadamard gates are added to the input register to create superposition. As for MEF, two C-NOT gates CNOT(IQ_1, PQ_1) and CNOT(IQ_2, PQ_2), where

[1]Fermat's prime are prime numbers in the form 2^{2n}, for some integer 'n'

IQ_1 , IQ_2 are qubits of input register in increasing significance and PQ_1 , PQ_2 are qubits of the period register in increasing significance. Finally, the Quantum Fourier transform is applied to the input register. At last. the input register(IQ) is measured and stored in classical register. The circuit diagram for it is as shown in the figure below.

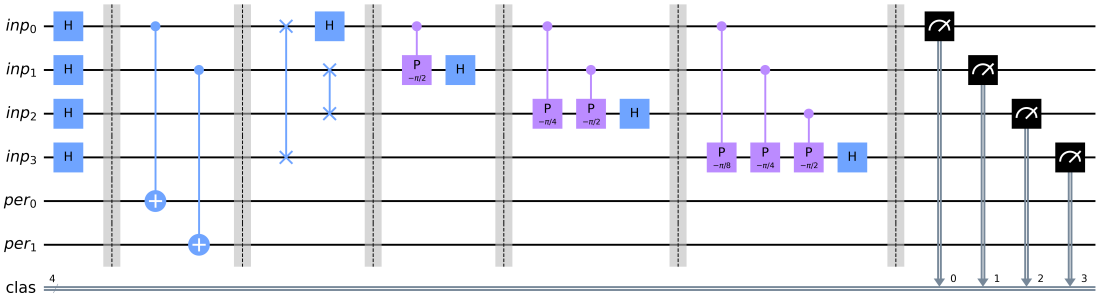


Figure 6.2: Circuit Diagram for implementation of compiled shor's algorithm

6.4 Circuit Execution

The circuit designed was then executed on a simulator and quantum computer. In both the systems, the circuit takes the following 6 sets of steps:

1 Creating

2 Transpiling

3 Validating

4 Queuing

5 Running

6 Completion

After the circuit is constructed, it needs to be transpiled so as to execute on the system. As mentioned above, the simulator and quantum system both have a set of basic gates. These systems could only execute circuits with these basic gates. The simulator has large set of basic gates and all the gates in the circuit is a subset of basic gates. So transpiling is easy for a simulator. However, the quantum system

only has five basic gates. So qiskit breaks down each operation into its basic gates and prepares a new circuit equivalent to the original circuit. For our study, the circuit was transpiled for the six qubits with the least errors. The circuit diagram after transpilation is as shown in figure(6.3). Now the transpiled circuit is validated by the IBM system and set for queuing. After queuing, the circuit is executed on the system. QISKIT allows to set the number of shots for a circuit to execute. For our study,a job for 8000 shots of the circuit was sent for execution. And finally, the result was obtained after the completion of execution. The result obtained need to be analyzed classically. The number theory principles were used to analyze the result to get the factor of the integer

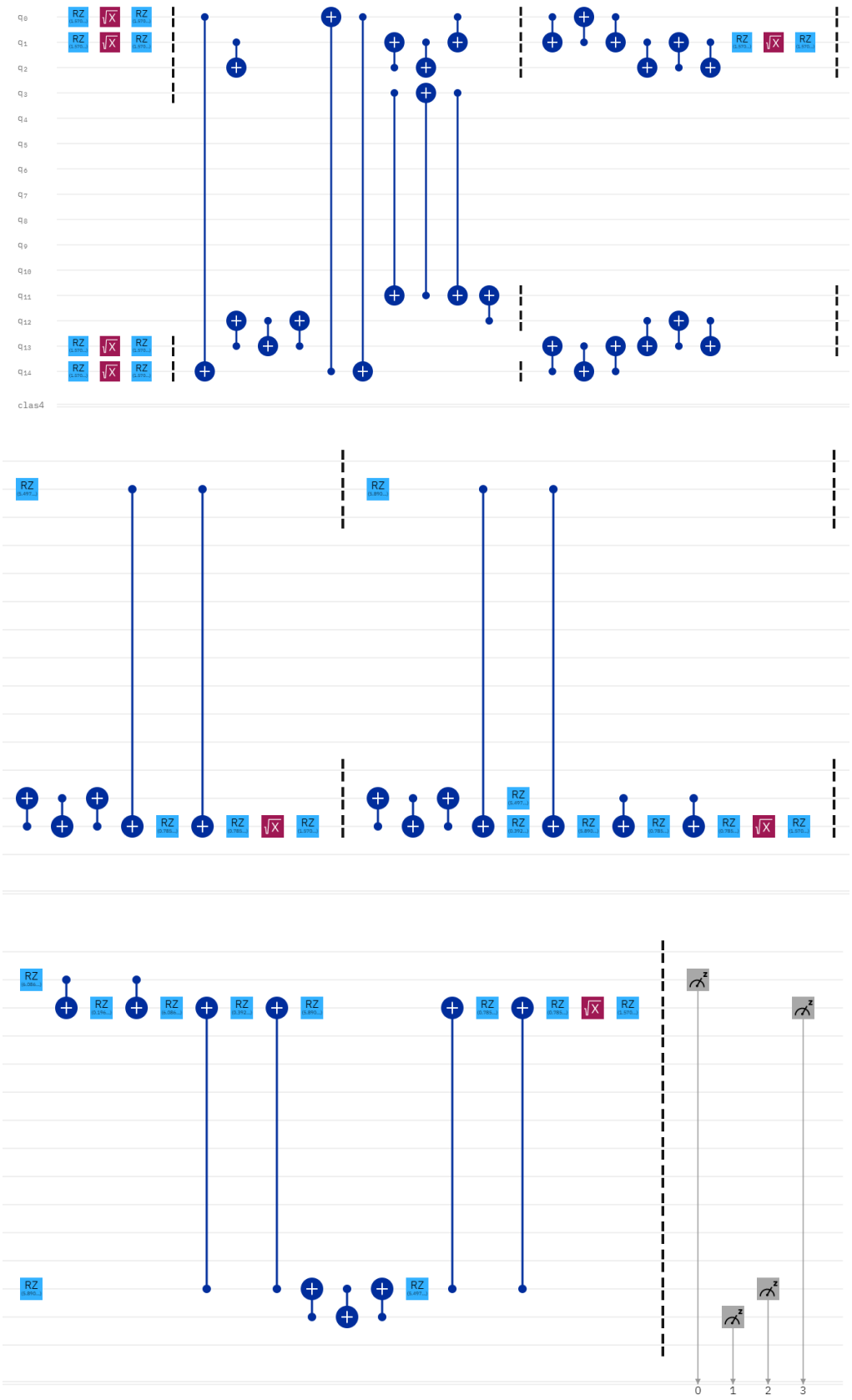


Figure 6.3: Circuit Diagram of upon transpilation of original circuit(6.2)

Chapter7: Results

Upon execution on simulator and quantum computer, it was observed that simulator took less time for all the process: transpiling, validating and running, than for quantum computers. On the simulator, it took 903 milliseconds(ms) for validating, 241 ms in queuing, and 41 ms to run the circuit for 8000 shots. It took a total of 3.8 seconds to completely execute the circuit. Whereas for the quantum system, it took a total of 24 minutes and 52.1 seconds to execute the circuit. Queuing took most of the time while sending the circuit. It took 957 ms for validating, 23 minutes, and 58.9 seconds for queuing. It took 20.7 seconds to run the circuit for 8000 shots on $ibmq\_16\_melbourne$.

The result counts upon running the circuit on the simulator and quantum system is as shown in table below.

Basis States	Simulator counts	Quantum counts
0000	2001	875
0001	0	511
0010	0	542
0011	0	336
0100	1990	713
0101	0	394
0110	0	456
0111	0	282
1000	1998	846
1001	0	460
1010	0	479
1011	0	321
1100	2011	678
1101	0	397
1110	0	428
1111	0	282

We used histogram plots to analyze the output. The histogram plots of the output is as shown in the figure(7.1, 7.2)

We could see the similarities between the result on the local simulator and quantum computer. We observed states:$\{0000\}, \{0100\}, \{1000\}$ and $\{1100\}$ have

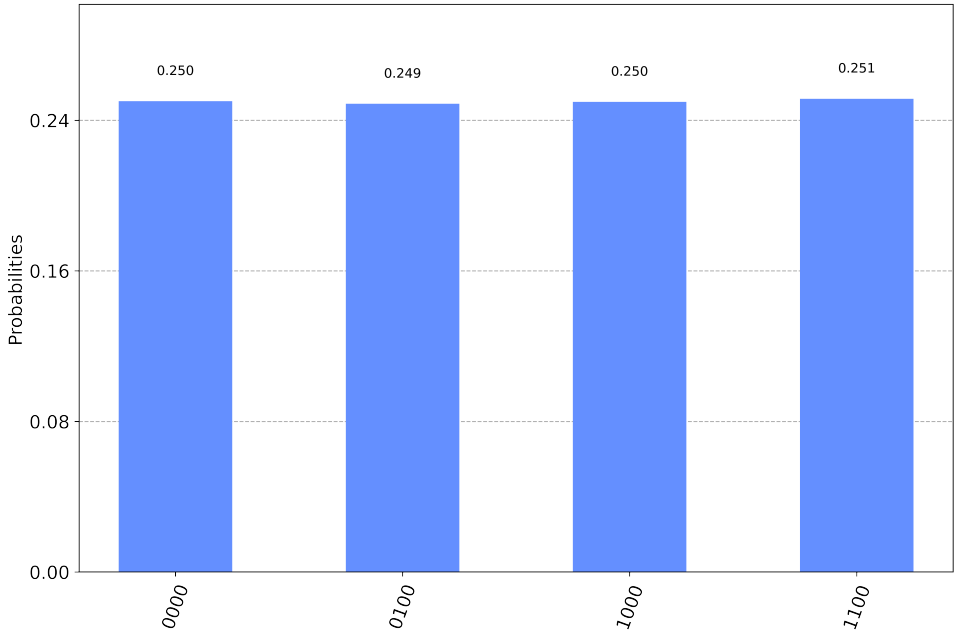


Figure 7.1: Results upon running circuit on quantum simulator

the maximum likelihood of observation on both the results. These binaries correspond to decimal values 0, 4, 8 and 12. We analyzed the output values classically to obtain the period of MEF and compute the factor of integer 15.

For final classical computation, we took GCD of the output values, that is, 4. Hence, the period of MEF with $z = 2$ and $I = 15$ is $a = 4$. The factors was computed from period as $GCD(2^2 + 1, 15) = 5$ and $GCD(2^2 - 1, 15) = 3$. Thus, we obtained values 3 and 5. We checked if these values are factors by dividing I by these values. We get the positive result after checking as well. Hence we obtained an correct result for the implementation.

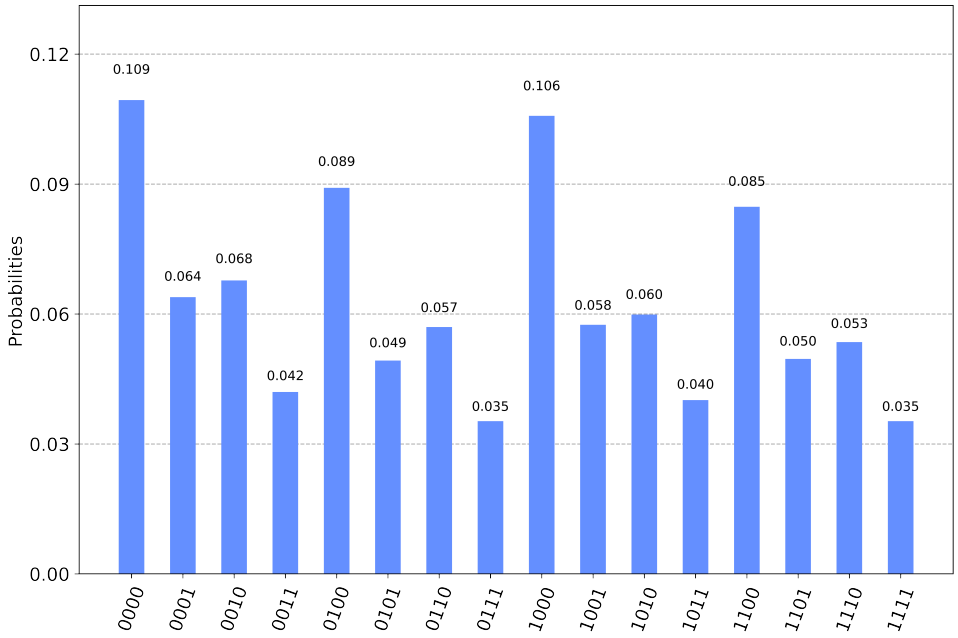


Figure 7.2: Results upon running circuit on quantum computer

Chapter8: Discussions

8.1 Discussion

Our results suggest the successful implementation of Shor's algorithm on a quantum computer and on the simulator. The correct results were obtained from the implementation experiment. We observed that the run-time of the quantum computer was longer than that in the simulator. Also, there was longer queuing for the quantum system than for the simulator. The reason for longer queuing is that $ibmq\_16\_melbourne$ is the largest open quantum system and its obvious that the system will have a longer queue. Also, qiskit is packed with local qasm simulator which make is less likely for users to use the $ibmq\_qasm\_simulator$. The longer run-time for a quantum computers might hint at a lower efficiency than that for the simulator. But it is so due to the inefficiency of the hardware of the system. One can think of quantum computers today as first-generation computers for quantum computing.

Comparing the results from quantum computers and the simulator, we could observe some differences in results between them. There were some states with some small probabilities on the results from the quantum system that are not present in the result of simulators. This is due to the errors in the quantum system due to hardware issues like decoherence, dephasing, transmon leakage, gate errors, and measurement errors. Large numbers of gates in the circuit cause qubits to lose their coherence termed as decoherence error.[24] Gate errors like CNOT gate errors are present as such gates are present in the circuit. Also, some errors are present due to unwanted interactions during measurement. Such errors are some of the hurdles quantum computing is facing at the moment.

We can decrease some errors in the result by decreasing the number of gates with higher error rates. In our circuit, we can omit the swap gates in Fourier transform and swap classical register, at last, to expect a result with lesser errors. Quantum

error correction is a field of study dedicated to decreasing errors in quantum computing. Several other techniques have been developed in the field and is an active area of research.

We've discussed Shor's algorithm might pose threat to information security due to its exponential speedup. But the threat is not imminent at present because of the errors and limitations in quantum computers available today. The largest number yet factored by a quantum computer using Shor's algorithm is 21[21]. The limitation of numbers of logical qubits and errors hinders the application of Shor's algorithm to factor large numbers. Inability to prepare a stable qubit and contain qubit properly affects the coherence of qubits. Another important factor affecting the performance of quantum computers is an imperfection in entanglements resulting in gate errors like CNOT errors. These impacts profoundly on the scalability of some quantum algorithms like Shor's algorithm.

Considering the threat to the current cryptosystem by Shor's algorithm, a new cryptosystem based on quantum computing called quantum key distribution(QKD) has been developed recently which proves to be foul-proof and more secure. This algorithm uses fragility of the state of a quantum system, that is measurement destroys the state of system, to its advantage to share encryption key and ensure secure transfer of a key.

Chapter9: Conclusion

Compared to all other algorithms for integer factorization, it is clear that Shor's algorithm on a quantum computer performs more efficiently than other classical counterparts. Shor's algorithm express the idea that quantum computers are capable of performing tasks with lower complexity than classical ones and its implementation further verifies it. Despite the success in implementing the algorithm, quantum computers available today are far away at making themselves approachable and advantageous for the general public. The current status of quantum computers does not allow scalable and applicable use of Shor's algorithm. Many types of research and efforts have been made in the field to make use of the algorithm.

Despite the reality that current quantum computers are not able to make use of Shor's algorithm, the introduction of Shor's algorithm itself has a profound effect on science. It popularized the concept of quantum computation to the research community and public and helped in further exploration of more capabilities and possibilities in quantum information/computation. This algorithm is one of the reasons that quantum computation and information is considered as one of the most promising fields of time. Quantum computing is expected to be the future of computing. It is highly likely that quantum computing and information will have remarkable applications in our lives directly or indirectly. Research, simulation, optimization, and security are some of the promising areas of the application of quantum computers.

References

[1] Sven Anderzén Rodenkirchen and Marcus Granström. *Performance evaluation of quantum Fourier transform on a modern quantum device*. 2018.

[2] D.M. Burton. *Elementary Number Theory*. Mcgraw-Hill, 2011. ISBN: 9780077418120. URL: https://books.google.com.np/books?id=3KiUCgAAQBAJ.

[3] Philip G Calabrese. "Toward a more natural expression of quantum logic with Boolean fractions". In: *Journal of philosophical logic* 34.4 (2005), pp. 363–401.

[4] Goong Chen, Louis Kauffman, and Samuel J. Lomonaco. *Mathematics of Quantum Computation and Quantum Technology*. Chapman & amp; Hall/CRC, 2007. ISBN: 1584888997.

[5] D. Cheung. "Using generalized quantum Fourier transforms in quantum phase estimation algorithms". Citeseer, 2003.

[6] Patrick J Coles et al. "Quantum algorithm implementations for beginners". In: *arXiv preprint arXiv:1804.03719* (2018).

[7] Mary Joan Collison. "The Unique Factorization Theorem: From Euclid to Gauss". In: *Mathematics Magazine* 53.2 (1980), pp. 96–100. ISSN: 0025570X, 19300980. URL: http://www.jstor.org/stable/2689956.

[8] BGSMath Communication. *Complexity-classes-diagram*. URL: https://bgsmath.cat/event/an-introduction-to-parameterized-complexity/complexity-classes-diagram/.

[9] Simon J Devitt. "Performing quantum computing experiments in the cloud". In: *Physical Review A* 94.3 (2016), p. 032329.

[10] David P DiVincenzo. "The physical implementation of quantum computation". In: *Fortschritte der Physik: Progress of Physics* 48.9-11 (2000), pp. 771–783.

[11] *factorization of rsa 250*. URL: https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html.

[12] Richard P Feynman. "Simulating physics with computers". In: *Int. J. Theor. Phys* 21.6/7 (1999).

[13] Chris Fisher. *Docs and Resources*. URL: https://quantum-computing.ibm.com/docs/manage/backends/.

[14] Chris Fisher. *Quantum Computing at IBM*. Apr. 2009. URL: https://www.ibm.com/quantum-computing/quantum-computing-at-ibm/.

[15] Michael R Geller and Zhongyuan Zhou. "Factoring 51 and 85 with 8 qubits". In: *Scientific reports* 3.1 (2013), pp. 1–5.

[16] Shah Muhammad Hamdi et al. "A Compare between Shors quantum factoring algorithm and General Number Field Sieve". In: *2014 International Conference on Electrical Engineering and Information & Communication Technology* (2014). DOI: 10.1109/iceeict.2014.6919115.

[17] Jack Hidary. *Quantum Computing: An Applied Approach*. Cham, Switzerland: Springer, 2019. ISBN: 978-3-030-23921-3. DOI: 10.1007/978-3-030-23922-0.

[18] E.R. Johnston, N. Harrigan, and M. Gimeno-Segovia. *Programming Quantum Computers: Essential Algorithms and Code Samples*. O'Reilly Media, 2019. ISBN: 9781492039655. URL: https://books.google.com.np/books?id=QqegDwAAQBAJ.

[19] Fang Xi Lin. "Shor's algorithm and the quantum Fourier transform". In: *McGill University* (2014).

[20] Michael Loceff. *A Course in Quantum Computing, Vol1*. 2015.

[21] Enrique Martin-Lopez et al. "Experimental realization of Shor's quantum factoring algorithm using qubit recycling". In: *Nature photonics* 6.11 (2012), pp. 773–776.

[22] K. Murugeshan R.and Sivaprasath. *Modern Physics*. S. Chand Limited, 2008. ISBN: 9788121928014. URL: https://books.google.com.np/books?id=kOdJVZ6HlgoC.

[23] Hartmut Neven et al. "QBoost: Large Scale Classifier Training with Adiabatic Quantum Optimization." In: *ACML*. 2012, pp. 333–348.

[24] Michael A Nielsen and Isaac Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2002. ISBN: 978-1-107-00217-3. URL: www.cambridge.org%5C9781107002173.

[25] *Open-Source Quantum Development*. URL: https://qiskit.org/.

[26] W. Raji. *An Introductory Course in Elementary Number Theory*. The Saylor Foundation, 2013. URL: https://books.google.com.np/books?id=SYRPAgAAQBAJ.

[27] Harold N Shapiro. *Introduction to the Theory of Numbers*. Courier Corporation, 2008.

[28] Peter W Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[29] *Systems & simulators*. URL: https://quantum-computing.ibm.com/systems?systems=all.

[30] *The Qiskit Elements¶*. URL: https://qiskit.org/documentation/the_elements.html.

[31] Rodney Van Meter and Kohei M Itoh. "Fast quantum modular exponentiation". In: *Physical Review A* 71.5 (2005), p. 052320.

[32] Lieven MK Vandersypen et al. "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance". In: *Nature* 414.6866 (2001), pp. 883–887.

[33] N. Zettili. *Quantum Mechanics: Concepts and Applications*. Wiley, 2009, pp. 1–3. ISBN: 9780470026786. URL: https://books.google.com.np/books?id=6jXlpJCSz98C.

[34] Lei Zhang, Andriy Miranskyy, and Walid Rjaibi. "Quantum Advantage and the Y2K Bug: A Comparison". In: *IEEE Software* 38.2 (2021), pp. 80–87. DOI: 10.1109/ms.2020.2985321.

Chapter10: Appendix

.1 Source code

Code for Shor's algorithm for integers composed of Fermat's prime: *shor for fermat.py*

```
1
2  '''Installation of some necessary packages'''
3  # !pip install qiskit   # install qiskit package
4  # !pip install pylatexenc
5  # # the output will be cleared after installation
6  # from IPython.display import clear_output
7  # clear_output()
8
9
10
11 '''Shor's algorithm for composite number composed of fermat
       primes'''
12 # Import necessary packages
13 get_ipython().run_line_magic('matplotlib', 'inline')
14 from matplotlib import pyplot as plt
15 import numpy as np
16 import math
17
18 pi = math.pi
19
20
21
22 '''Import qiskit modules and functions'''
23 from qiskit import QuantumRegister, ClassicalRegister,
       QuantumCircuit, execute,Aer, IBMQ
24 from qiskit.visualization import plot_histogram
25 from qiskit.compiler import transpile, assemble
26 from qiskit.quantum_info import state_fidelity
27
28
29
30 ''' Function to construct circuit for Quantum Fourier
       Transform
31 and inverse Quantum Fourier transform'''
32 def create_qft(qc, q_reg, n_qubit, swap=True, inverse=False,
       approx=0):
33     pi= math.pi
34     if inverse:
35         if swap:
36             for q in range(int(n_qubit/2)):
37                 qc.swap(q_reg[q],q_reg[n_qubit-q-1])
38         for i in range(n_qubit):
39             for j in range(i):
40                 qc.cp(-pi/math.pow(2,(i-j)), q_reg[j], q_reg
       [i])
```

```
41              qc.h(q_reg[i])
42              qc.barrier()
43
44
45      else:
46          for i in range(n_qubit-1,-1,-1):
47              qc.h(q_reg[i])
48              for j in range(i-1,-1,-1):
49                  if(pi)/(pow(2,(i-j)))>0:
50                      qc.cp(pi/math.pow(2,(i-j)), q_reg[i],
    q_reg[j])
51
52
53              qc.barrier()
54
55          if swap:
56              for q in range(int(n_qubit/2)):
57                  qc.swap(q_reg[q],q_reg[n_qubit-q-1])
58
59
60  '''Preparation of Quantum circuit'''
61  in_qreg = QuantumRegister(4,'inp')  # create input register
62  per_qreg = QuantumRegister(2,'per')  # create period
        register
63  creg = ClassicalRegister(4,'clas')  # create classical
        register
64
65  qc = QuantumCircuit(in_qreg, per_qreg, creg) #  initialize a
        quantum circuit
66
67  # Create superposition using hadamard gate
68  qc.h(in_qreg)
69  qc.barrier()  # add barrier to make the circuit
        representation clear
70
71  # Apply compiled modular exponentiation function between two
        register
72  qc.cx(in_qreg[0],per_qreg[0])
73  qc.cx(in_qreg[1],per_qreg[1])
74  qc.barrier()  # add barrier to make the circuit
        representation clear
75
76  # Apply inverse quantum fourier transformon input register
77  create_qft(qc, in_qreg, 4, swap=True, inverse=True)  # swap=
        False to omit swap operation
78
79  # apply measurement on input register and collapse to
        classical register
80  qc.measure(in_qreg, creg)
81
```

```
82 qc.draw(output = 'mpl',filename = 'shorfor15_2.png', scale =
       3)
83
84
85
86 '''Run the circuit in simulator: '''
87 # Load saved IBMQ account
88 # IBMQ.save_account
       ("**********************************************************************
89
90 IBMQ.load_account()
91 provider = IBMQ.get_provider(hub='ibm-q')
92 sim_backend = provider.get_backend("ibmq_qasm_simulator")
93 '''import least busy quantum backend'''
94 from qiskit.providers.ibmq import least_busy
95 n=6
96 qc_backend = least_busy(provider.backends(filters=lambda x:
       x.configuration().n_qubits >= n
97                                            and not x.
       configuration().simulator
98                                            and x.status().
       operational==True))
99 print("least busy quantum backend: ", qc_backend)
100
101
102 """Run the circuit in Processers: Use backend='sim_backend'
       for simulator
103 s and backend= 'qc_backend' for quantum coputers  """
104
105 # running circuit on simulator at IBM
106 job = execute(qc,backend=sim_backend,shots =8000 )
107 result = job.result()
108 counts=result.get_counts(qc)
109 counts
110
111
112 fig,ax = plt.subplots(figsize=(12,8))  # figsize=(16,8)
113 ax.set_aspect('auto')
114 plot_histogram(counts,ax=ax)
115 plt.savefig('histogram_result_fermat_sim.png',dpi=800)
116
117
118 # running circuit on quantum computer
119 job_qc = execute(qc,backend=qc_backend,shots =8000 )
120 result_qc = job_qc.result()
121 counts_qc=result_qc.get_counts(qc)
122 counts_qc
123
124
125 fig,ax = plt.subplots(figsize=(12,8))  # figsize=(16,8)
```

```
ax.set_aspect('auto')
plot_histogram(counts_qc,ax=ax)
plt.savefig('histogram_result_fermat_qc.png',dpi=800)
```

```
import qiskit
qiskit.__qiskit_version__
```

```
{'qiskit-terra': '0.16.2',
 'qiskit-aer': '0.7.3',
 'qiskit-ignis': '0.5.1',
 'qiskit-ibmq-provider': '0.11.1',
 'qiskit-aqua': '0.8.1',
 'qiskit': '0.23.3'}
```

Code for creating graph 2.10: *forMEFgraph.py*

```
""" Code to plot Modular exponentiation function outputs of
   two types """

#import necessary packages
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-notebook')
# plt.style.available
import numpy as np
from scipy.fft import fft

def gcd(a,b):
'''   return the greatest common divisor of a,b'''
    while (b):
      m = a % b
      a = b
      b = m
    return a

def periodfind(x,I,N):
    '''find period a of function f(x)=x^a mod N '''
    arr=[]
    for i in range(N):
        f= (x**i)%(I)
        arr.append(f)
    return arr

'''For case I'''
N=16
I=16
z=3
y=np.array(periodfind(z,I,N))
x=np.arange(N)
```

```
plt.style.use('seaborn-whitegrid')
plt.title('Period of function f(x)=${}^x$(mod {I})'.format(z
    ,I=I))
plt.xlabel('x')
plt.ylabel('f(x)=${}^x$(mod {I})'.format(z,I=I))
plt.plot(x,y,'o',linewidth=1)  #  ,'r-o'
plt.plot(x,y,'--')
plt.show()


'''For caseII'''
N=15
I=15
z=2
y=np.array(periodfind(z,I,N))
x=np.arange(N)
plt.style.use('seaborn-whitegrid')
plt.title('Period of function f(x)=${}^x$(mod {I})'.format(z
    ,I=I))
plt.xlabel('x')
plt.ylabel('f(x)=${}^x$(mod {I})'.format(z,I=I))
plt.plot(x,y,'o',linewidth=1)  #  ,'r-o'
plt.plot(x,y,'--')
plt.show()
```

Code to plot the graph showing the complexity of Shor's algorithm and number field sieve: *Complexity$_s$hor$_v$s$_s$ieve.py*

```
""" Code to plot the graph  showing complexity of ShorâĂŹs
    algorithm and number field seive"""

# import necessary packages
import math
import matplotlib.pyplot as plt

x=[i for i in range(4,500)]
# y1=[ 1.9*math.pow(i,3) for i in x[1:]]
y1 = [math.pow(2,(1.9*math.log2(math.pow(n,1/3))*math.log2(
    math.pow(math.log2(n),2/3)))) for n in x]
y2=[math.log2(math.pow(n,3)) for n in x]

plt.style.use('seaborn-whitegrid')
plt.figure(figsize=(14,8))
plt.xlabel('number of bits')
plt.ylabel('Numbers of operations in log scale')
plt.plot(x,y1)
plt.plot(x,y2)
plt.yscale("log")
plt.annotate(text='Number seive',xy=(300,1400), xytext
    =(280,2500),
             arrowprops=dict(arrowstyle='->',connectionstyle
```

```
    ="arc3"))
21 plt.annotate(text='Shor algorithm',xy=(300,25), xytext
    =(280,50),
22                 arrowprops=dict(arrowstyle='->',connectionstyle
    ="arc3"))
23 plt.savefig('Complexity_shor_vs_sieve.png')
```

.2 Numerical implementation

We discuss two approaches of numerical implementation: an approach using number theory background and an approach involving direct use of Shor's algorithm.

.2 Number theory approach

Consider an integer to be factored: $I = 15$

First we consider a random number $z = 2$

$GCD(15, 2) = 1$, 15 and 2 are co-prime

$f(x) = 2^x \bmod 15$ for $x \in \mathbf{N}$ gives

f(0)	f(1)	f(2)	f(3)	f(4)	f(5)	f(6)	f(7)	f(8)	f(9)	f(10)	f(11)	f(12)	f(13)	f(14)
1	2	4	8	1	2	4	8	1	2	4	8	1	2	4

Table 1: Example of implementation:$f(x) = 2^x \bmod 15$

The period of function $f(x)$ is 4.

Now, according to 1, the factors are $GCD(2^{\frac{4}{2}}+1, 15) = 5$ and $GCD(2^{\frac{4}{2}}-1, 15) = 3$

.2 Shor's algorithm approach

In this approach, we would use apply Shor's algorithm numerically on an integer. This method involves working with probabilities and histogram. We discuss the application of Shor's algorithm, on two different set of examples:

Example I: $m = \tilde{m}$

Let a number to be factored be $I = 15$

Using the inequality $I^2 \leq 2^n = N \leq 2I^2$, we get, $N = 256$ Also, we get $r = \lfloor 2logI \rfloor + 1 \simeq 4$

Hence, the initialized register is $\psi_1 = |0\rangle^8 |0\rangle^4$

Upon application of Hadamard gates on 1st register, we get

$$\psi_2 = \frac{1}{\sqrt{256}} \sum_{x=0}^{255} |x\rangle^8 |0\rangle^4 = \frac{1}{\sqrt{256}} (|0\rangle |0\rangle + |1\rangle |0\rangle, \ldots, |255\rangle |0\rangle)$$

The modular exponential function $f(x) = z^x \bmod I$ where z is a randomly selected integer less than I.

Let the randomly selected integer $z = 2$

So, $f(x) = 2^x \bmod 15$

First few terms of $f(x)$ are:

1	2	4	8	1	2	4	8
1	2	4	8	1	2	4	8
1	2	4	8	1	2	4	8
1	2	4	8	1	2	4	8
1	2	4	8	1	2	4	8
1	2	4	8	1	2	4	8
1	2	4	8	1	2	4	8
1	2	4	8	1	2	4	8

We can clearly see that it has the order, $a = 4$ and frequency $m = \frac{N}{a} = 64$

Now applying $f(x)$ on second register,

$$\psi_3 = \frac{1}{\sqrt{256}} \sum_{x=0}^{255} |x\rangle^8 |f(x)\rangle^4 = \frac{1}{\sqrt{255}} \sum_{j=0}^{63} \sum_{x=0}^{4} |x + 4j\rangle |f(x)\rangle$$

Upon conceptual measurement, let the second register collapse to $f(5)$. Then, the first register collapses correspondingly to following state:

$$\psi_4 = \frac{1}{\sqrt{64}} \sum_{j=0}^{63} |3 + 6j\rangle |2^3 mod 15\rangle$$

$$\psi_4 = \frac{1}{\sqrt{64}} (|3\rangle + |7\rangle + |11\rangle + \cdots + |255\rangle)$$

Upon application of QFT on first register, we get:

$$\psi_5 = \sqrt{\frac{m}{N}} \sum_{c=0}^{3} e^{\frac{2i\pi 64c}{m}} |64c\rangle$$

$$= \frac{1}{2}(|0\rangle + |64\rangle + |128\rangle + |192\rangle)$$

Upon measurement of first register, we obtain a set$\{y_c\} = \{0, 64, 128, 192\}$ with higher probabilities than others.

The histogram represents classical fast Fourier transform on $f(x)$ which represents the similar output as calculated numerically.

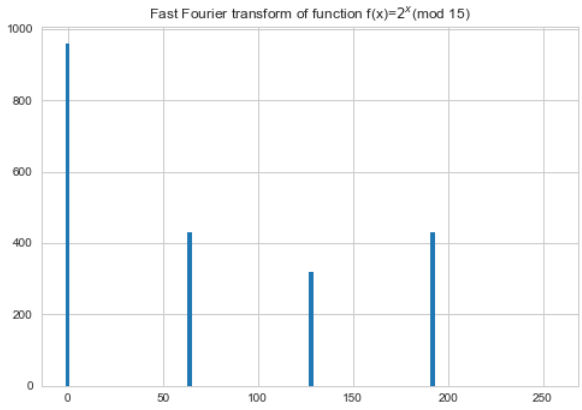


Figure 1: Output after applying FFT on $f(x)$

Now,we take the GCD of the possible states: $GCD(64, 128, 192) = 64$ The period/order is

$$a = \frac{N}{m} = \frac{256}{64} = 4$$

. S, the factors are $GCD(2^{\frac{a}{2}} \pm 1, 15) = GCD(5, 15)$ and $GCD(3, 15) = 5$ and 3.

Example II: $m \neq \tilde{m}$

For this example, we take a different number to be a factor, $I = 21$. Similarly as in example I, we get $N = 512, n = 9$ and $r = 4$.

Hence the initialized register be $\psi_1 = |0\rangle^9 |0\rangle^4$

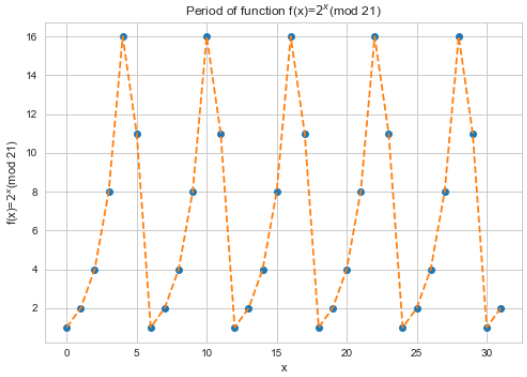
Upon applying Hadamard on 1at register, we get

$$\psi_2 = \frac{1}{\sqrt{512}} \sum_{x=0}^{511} |x\rangle^9 |0\rangle^4 = \frac{1}{\sqrt{512}} (|0\rangle |0\rangle + |1\rangle |0\rangle, \cdots, |511\rangle |0\rangle)$$

Let $z = 2$ be randomly selected integer for the modular exponential function $f(x) = z^x \bmod I$. So, $f(x) = 2^x \bmod 21$

First few terms of $f(x)$ are

1	2	4	8	16	11	1	2
4	8	16	11	1	2	4	8
16	11	1	2	4	8	16	11
1	2	4	8	16	11	1	2



From the figure(**??** and terms, we can clearly see it has the order, $a = 6$. The task is to determine the order computationally.

Now applying $f(x)$ on second register,

$$\psi_3 = \frac{1}{\sqrt{512}} \sum_{x=0}^{511} |x\rangle^9 |f(x)\rangle^4 = \frac{1}{\sqrt{64}} \sum_{j=0}^{\tilde{m}} \sum_{x=0}^{6} |x + 6j\rangle |f(x)\rangle$$

where $\tilde{m}$ is the frequency of $f(x)$.

Since at $x = 255$ the function $f(x)$ does not complete the full cycle,

$$\tilde{m} = \begin{cases} m = 86 & \text{for f(x)=\{1,2\}} \\ m = 85 & \text{for f(x)=\{4,8,15,11\}} \end{cases}$$

Upon conceptual measurement, let the second register collapse to $f(5)$. Then, the

first register collapses correspondingly to following state:

$$\psi_4 = \frac{1}{\sqrt{85}} \sum_{j=0}^{85} |5 + 6j\rangle \, |2^5 \, mod \, 21\rangle$$

$$\psi_4 = \frac{1}{\sqrt{85}}(|5\rangle + |11\rangle + |17\rangle + \cdots + |510\rangle)$$

Upon application of QFT on first register, we get:

$$\psi_5 = \frac{1}{\sqrt{512 * 85}} \sum_{j=0}^{512} \sum_{j=0}^{85} e^{\frac{2i\pi(5+6j)y}{256}} \, |y\rangle$$

According to the theory, upon measuring the 1st register, we get a set of outputs $y_c\rfloor_{c=0}^{5}$ with higher probabilities than others. The set $y_c\rfloor_{c=0}^{5}$ satisfies:

$$ay_c \in [cN - a/2, cN + a/2]$$

$$6y_c \in [cN - 6/2, cN + 6/2]$$

By computation, the $y_c's$ are $\{0, 85, 171, 256, 341, 427\}$.

The histogram represents classical fast Fourier transform on $f(x)$ which represents the similar output as calculated numerically.

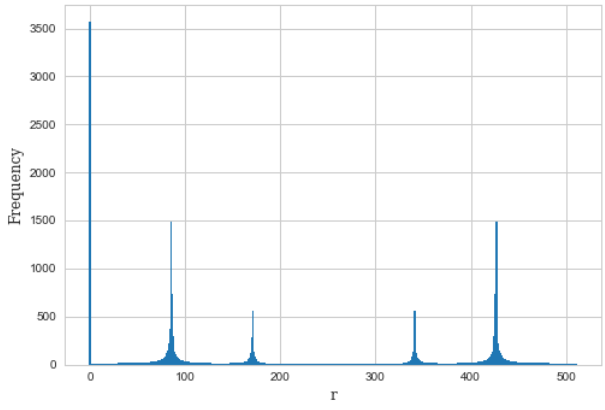


Figure 2: Output after applying FFT on $f(x)$

Now we use $y_c's$ to compute period,a. We apply Continued fraction algorithm on a set $\frac{y_c}{N}\rfloor_{c=0}^{5} = \frac{y_c}{512}\rfloor_{c=0}^{5}$. And from 5.3.7: Claim 2,Continued fraction Algorithm

on $\frac{y_c}{N}$ with error $\frac{a}{2I^2}$ outputs unique $\frac{c}{a}$. For example,

The convergents of $\frac{85}{512}$ are $\{\frac{1}{6}, \frac{42}{253}\}$

Then $\left|\frac{85}{512} - \frac{1}{6}\right| < \frac{1}{2*21^2}$ but $\left|\frac{85}{512} - \frac{42}{253}\right| > \frac{1}{2*21^2}$.

So, $\frac{85}{512}$ converges to $\frac{1}{6}$. Similarly, all elements in set $\{y_c\}$converges to a set $\{\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}, 1. \ \}$ We know, $\{\frac{y_c}{N}\}_{c=0}^{5}$ converges to $\{\frac{c}{a}\}_{c=0}^{5}$. Hence comparing the above set, the period is $a = 6$.

Similarly as above, the factors are obtained from $GCD(2^{\frac{6}{2}} \pm 1, 21)$= $GCD(7, 21)$ and $GCD(9, 21)$. The factors are 7 and 3.