Shor's Algorithm for Polynomial-time integer factorization

Rahul Mitra
PHYS 405 Senior Exercise
Department of Physics
Trinity College
Hartford, Connecticut, USA
email: rahul.mitra@trincoll.edu

***Abstract*— Modern computer security systems are built upon one-way or trap-door functions. Such one-way functions have given birth to a class of protocols known as asymmetric cryptographic systems. One-way functions are "easy" to compute in one direction while being "hard" or computationally very expensive to compute in the other. Integer factorization is an example of such a problem i.e., it is very efficient to multiply two numbers to get a product but factoring that product is computationally difficult. Numerous cryptographic systems rely on integer factorization belonging to the Nondeterministic Polynomial time (NP) class of computational problems. Shor's algorithm, published in 1994, presents a Quantum Algorithm to factorize an integer in polynomial time. In this paper, Shor's algorithm and its implications will be discussed. First, the paper will provide motivation for studying Shor's algorithm from a security perspective. After which, the paper will introduce some necessary background in the analysis of algorithms and the notion of computational difficulty. The Quantum Fourier Transform (QFT) and the Quantum Phase Estimation (QPE) will be discussed at length. In section VI, the paper will explain how Shor's algorithms leverages the QPE to factor in polynomial time. The paper will conclude by presenting the results from some recent quantum implementations of the algorithm. This paper attempts to describe the physics and the importance of Shor's algorithm for integer factorization.**

***Keywords* — integer factorization, Shor's algorithm, computationally difficulty, quantum fourier transform, quantum phase estimation**

I. Introduction

Numerous security schemes rely on computationally difficult problems to achieve their security. The most famous examples include, but are not limited to, RSA encryption [1], Blum-Blum-Schub (BBS) randomization [2], Diffie-Hellman Key Exchange [3] and El-Gamal encryption [4]. The security of these schemes rely on trap door functions, i.e., functions that have are easy to compute from $D \rightarrow R$ but difficult to compute from $R \rightarrow D$, without some parameter t.

As an example and motivation for studying Shor's algorithm, consider briefly, RSA encryption. Bob has a public key, (n, e) and a private key, (n, d). Here, d is the trapdoor parameter. If Alice wants to send a message, M to Bob, she encrypts the messages as C where,

$$C = M^e (mod\ n) \tag{1}$$

Bob receives C and recovers M as,

$$M = C^d (mod\ n) \tag{2}$$

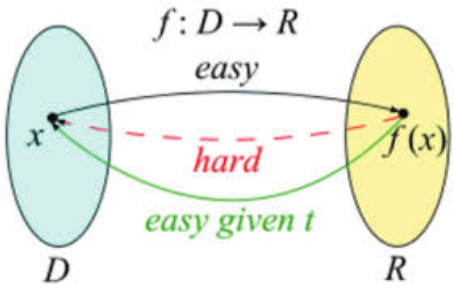


Fig. 1: Nature of trap door functions.

Two additional requirements needed for the correctness of RSA include:

$$gcd(\phi(n), e) = 1, \tag{3}$$

where $\phi(n)$ is Euler's totient function [5] defined as,

$$\phi(n) = (p-1)(q-1) \tag{4}$$

for some large number, n that is the product of two primes, p and q. Also,

$$de\ mod\ \phi(n) = 1 \tag{5}$$

Eq (1), (2) and (4) are very easy to compute because integer multiplication is very efficient on a classical computer (integer exponentiation maybe represented as a sequence of repeated multiplications). Euler's algorithm [6] provides a very efficient method for computing Eq (3). RSA encryption is a deep, rich and well-understood topic that is not the primary focus of this paper. As such, I will not be discussing the origins of Eq (1) - (5). The reader is directed to [1] for an in-depth discussion of RSA. However, it is very important to realize that without parameter, d i.e., the trapdoor parameter, Bob cannot easily decrypt Alice's message. In fact, to decrypt message, M without d would require Bob to find $\phi(n)$ i.e., Bob would have to find the prime factors of n. This is known is as the *factoring problem* and it is of huge significance to security schemes. Shor's Algorithm provides an efficient means of going from $R \rightarrow D$ without the trapdoor parameter, d.

Trap door functions have birthed asymmetric cryptosystems that have distinct keys for encryption and decryption.

These asymmetric systems rely on computationally difficult problems i.e., computing $R \rightarrow D$ is infeasible on a classical computer without the trapdoor parameter, d. Some examples include the *factoring problem* used in RSA [1], the *discrete log problem* used in the Diffie-Hellman Key Exchange [3] and the *elliptic curve discrete log problem* seen in the Elliptic Curve El Gamal protocol [7].

The next section includes a brief introduction to the notion of computation and algorithm analysis following which some prerequisites for Shor's Algorithm will be discussed.

II. Computation and Analysis

A. Model of Computation

1) The Turing Machine: The fundamental computational model on which algorithms are said to run on is the *Turing Machine* [8]. This is an idealized computer with a simple set of instructions and has unbounded memory. A *Turing Machine* has four components [9]:

(a) a program which is the input to the machine
(b) a finite state control which defines the actions of the machine on the input
(c) an infite tape which is the machine's memory
(d) a read/write tape-head which points to the position on the tape which is currently readable or writable

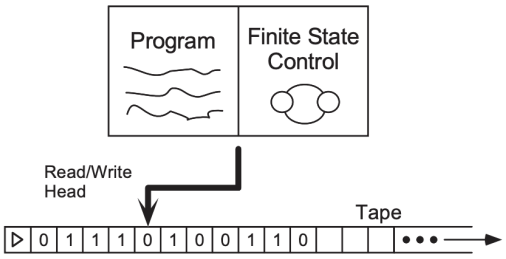


Fig. 2: Components of a Turing Machine [9].

The *Turing Machine* is the most powerful mathematical model of computation there is. Any operation/algorithm that runs on a modern computer can be simulated by a *Turing Machine*. In fact, the *Church-Turing thesis* states that the class of functions computable by a Turing machine corresponds exactly to the class of functions which are computable by an algorithm on a classical computer [9]. It should be noted here that quantum computers also obey the *Church-Turing thesis* i.e., quantum computers and *Turing machines* can compute the same class of functions. However, significantly for this paper, there are functions which can be computed much more efficiently on a quantum computer than is "believed" to be possible on a classical computer. Integer factorization is one such function. Shor's algorithm provides an efficient method to factorize an integer on a quantum computer.

2) Circuits: The notion of a circuit will become extremely significant in sections IV and V of this paper. A circuit is a collection of logic gates [9]. Simply defined, a logic gate is a function $f : \{0,1\}^k \rightarrow \{0,1\}^l$, for some fixed k input bits and l output bits. Standard logic gates include **AND, OR, NOT, NAND** and **NOR**. Computation with circuits, known as the *circuit model of computation* [9] is equivalent to the *Turing Machine*. To qualify this equivalence, the notion of a uniform circuit family needs to be discussed. A uniform circuit family is a set of circuits, $\{C_n\}$ (indexed by n) that has n input bits and a finite number of extra work bits and output bits. More particularly, an algorithm/program running on a *Turing Machine* generates a uniform circuit family, if the machine, on input n generates a description of C_n. These uniform circuit families, generated by *Turing Machines* can be made to compute the same class of functions as *Turing Machines* themselves. As such, *Turing Machines* and circuits are computationally equivalent [9].

B. Analysis of Algorithms

1) Asymptotic Notation: This notation allows for analyzing the essential behavior of a function, irrespective of minute changes in computational model [10]. More precisely, this notation provides a way of analyzing computation (bits accessed for operations on integers, gates needed for circuit design, comparisons performed for sorting algorithms etc.) without being explicit about the exact implementation. For example, suppose some algorithm to add two n bit integers needed $f(n) = 24n + 13$ gates to perform the computation. Asymptotically, $24n$ would dominate $f(n)$. As such, it is concluded that this algorithm asymptotically grows linearly as the input size i.e., the number of bits, n grows. In this instance and in the rest of the paper, the running time of an algorithm can be expressed as some function, $f(n)$ of the input size, n. $f(n)$ is descriptive of the number of primitive operations an algorithm needs to perform in order to complete some computation. As mentioned before, these primitive operations may include number of bits accessed for operations on integers, number of logic gates needed when building a circuit, number of vertices touched in some graph computation, number of comparisons needed for a sorting algorithm etc.

Mathematically, one can set an upper bound on the behaviour or "running time" of an algorithm using the *O* or "big oh" notation. In particular, $f(n)$ is said to be $O(g(n))$, if there are constants c and n_0 such that for all $n \geq n_0$, $f(n) \leq cg(n)$. Big O analysis is highly significant when studying Shor's algorithm because it allows us to quantify exactly how much more efficiently a quantum computer can factorize a number when compared to a classical computer. One may also formulate a lower bound on the running time of an algorithm using the Ω or "big omega" notation. $f(n)$ is said to be $\Omega(g(n))$, if there are constants c and n_0 such that for all $n \geq n_0$, $f(n) \geq cg(n)$. Finally, one can conclude that the running time of an algorithm is "tightly bounded" by some function $g(n)$ if and only if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$. When this is the case, $f(n)$ is said to be $\Theta(g(n))$. Fig. 3 graphically shows these definitions.

2) Complexity Classes: These classes help us to group algorithms with distinct asymptotic bounds into different

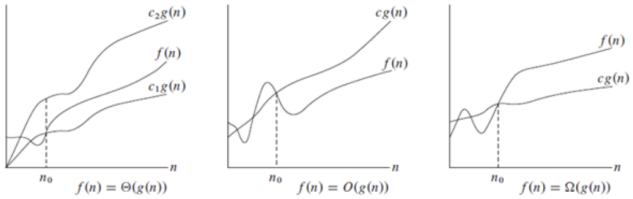


Fig. 3: Asymptotic Bounds [10].

sets or "classes". While there are a plethora of complexity classes, the two most important and relevant for this paper is the class, **P** (polynomial time) and the class **NP** (non-deterministic polynomial time). The *Turing Machine* model of computation is used to formally define **P** and **NP**.

Suppose a *Turing Machine* has states q_Y for "yes" and q_N for "no" in its finite state control. Supposed this machine is trying to compute a property about some input, say x. Some examples of such computation include the primality of x, the factors of x, the parity of x etc. Notice that there are only two final states (if the machine ever halts) - q_Y and q_N which imposes that the nature of computation provide a binary answer. This does not take away from the sophistication of the problems a *Turing Machine* can solve. In fact, any computational problem may be reduced to a decision problem i.e., to a problem with a binary answer. A problem is said to be in **TIME**$(f(n))$ if this machine halts i.e., stops computation at either q_N or q_Y on some input x in time $O(f(n))$ where n is the length of x. All problems solvable in **TIME**(n^k) for some finite k are said to be *polynomial time problems*. The collections of problems in **TIME**(n^k) makes up the class, **P**. All *polynomial time problems* are efficiently solved by a classical computer.

To define the class **NP**, the highly significant *factoring problem* is used. The factoring problem asks the question, for some integer n, what are its factors? More precisely, there are two requirements for a problem to be in **NP**. First, if m is a non-trivial factor of n, then halting the *Turing Machine* at state q_Y can be done in *polynomial time*. Second, if m is not a non-trivial factor of n, then halting the *Turing Machine* in state q_N can be done in *polynomial time*. Intuitively, the **NP** class essentially constitutes problems whereby, if given some solution, it is very efficient to check whether that solution is correct but very inefficient to actually compute that solution itself. The best-known classical factoring algorithm, the *general number field sieve* [11] runs in *exponential time* i.e, this algorithm would run in **TIME**$(h(n))$ on a *Turing Machine* where $h(n)$ is an exponential function. This is a highly significant fact because Shor's algorithm actually uses quantum superposition to improve this computation to a *polynomial time* one.

Proving that a given problem cannot be solved in *polynomial time* is a complex issue and one that is beyond the scope of this paper. **P** class problems is obviously a subset of the **NP** class of problems. Whether or not all problems in **NP** are in **P** is a question that you can win a $1 million prize for answering [12] but unfortunately, that issue is also beyond the scope of this paper. However, most theorists are fairly confident that **P** $\neq$ **NP** i.e., there are problems in **P** that are not in **NP** although a formal proof is yet to be presented.

III. Quantum Computation

1) Qubits: Much like classical bits, quantum bits ("qubit") exist in two possible states, when measured [9]. These are the states, $|0\rangle$ and $|1\rangle$. While a classical bit must be either in state 0 or state 1, a qubit can be in a state other than $|0\rangle$ and $|1\rangle$ up until the point of measurement. It is possible to create a *superposition* state which is a simple *linear combination of states* such as:

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{6}$$

The states $|0\rangle$ and $|1\rangle$ are known as the *computational basis states*. The numbers, α and β exist in the complex plane. Therefore, the state of a qubit is said to exist in a two-dimensional complex vector space. Also, the state vectors $|0\rangle$ and $|1\rangle$ are given by,

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{7}$$

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{8}$$

When measuring a qubit, there is a *collapse* to either state $|0\rangle$ with probability $|\alpha|^2$ or state $|1\rangle$ with probability $|\beta|^2$. Since there are only two possible states for a single qubit system, $|\alpha|^2 + |\beta|^2$ must equal 1. When a superposition state adheres to this property, it is said to be *norm preserving* or *normalized*.

The state, $|\Psi\rangle$ may also be written in terms of the relative phase between the states, $|0\rangle$ and $|1\rangle$. In this notation, the relative phase, ϕ, α and β may be confined to real numbers.

$$|\Psi\rangle = \alpha|0\rangle + e^{i\phi}\beta|1\rangle, \ \alpha, \beta, \phi \in \mathbb{R} \tag{9}$$

α and β may be further defined in terms of a single variable, θ.

$$\alpha = \cos\left(\frac{\theta}{2}\right) \tag{10}$$

$$\beta = \sin\left(\frac{\theta}{2}\right) \tag{11}$$

With these substitutions, the state $|\Psi\rangle$ is still normalized and θ, $\phi \in \mathbb{R}$. This allows us to represent a state on the surface of a sphere, known as the *Bloch Sphere* where $\theta \in [0, \pi]$, $\phi \in [0, 2\pi]$ and $r = 1$ because the the state must be normalized i.e, the magnitude of the qubit is 1.

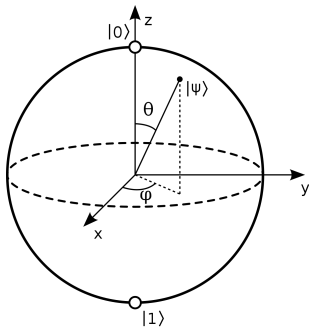
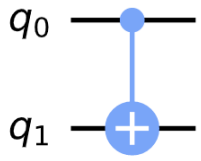


Fig. 4: Bloch Sphere representation of a qubit state.



(a) Quantum circuit representing the CNOT gate.

Input (t,c)	Output (t,c)
00	00
01	11
10	10
11	01

(b) Truth table for the CNOT gate. The target bit is represented by t and the control bit is shown by c.

Fig. 5: CNOT quantum circuit and truth table

2) Single Qubit Quantum Gates: Quantum gates allows for changing the state of a qubit. Gates acting on a single qubit can be represented by a 2×2 matrix [9]. However, these gates, after acting on a qubit state, must result in a *norm-preserving* state. The norm preservation of the resultant states is guaranteed by the fact that all quantum gates must be *unitary* i.e, if some gate is described by the matrix, U then $UU^\dagger = I$. As an example, consider the Pauli-X matrix.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{12}$$

Clearly, $XX^\dagger = I$. Applying X on state $|0\rangle$,

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \tag{13}$$

From Eq (13), it is seen that the Pauli-X gate acts as the classical **NOT** gate flipping the amplitudes of the $|0\rangle$ state vector and the $|1\rangle$ state vector.

A very significant quantum gate is the Hadamard gate given by the matrix,

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{14}$$

Applying H on $|0\rangle$ gives,

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} |0\rangle \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} &= \frac{1}{\sqrt{2}} [|0\rangle + |1\rangle] = |q\rangle \end{aligned} \tag{15}$$

The state $|q\rangle$ is commonly referred to as the $|+\rangle$ state. Similarly,

$$H|1\rangle = \frac{1}{\sqrt{2}} [|0\rangle - |1\rangle] = |-\rangle \tag{16}$$

The states $|+\rangle$ and $|-\rangle$ are very significant because they lie on the equatorial plane of the Bloch sphere i.e., the state vectors, $|0\rangle$ and $|1\rangle$ have been rotated away from the poles of the Bloch sphere. The Hadamard gate applies a change of basis from the Z-basis (also known as the computational basis defined earlier) to the X-basis, also called the Fourier basis.

3) Multiple Qubits and Multi Qubit Gates: While single qubits have two possible states, two qubits would have four possible states. If a is a four qubit state vector, there would be four complex amplitudes i.e.,

$$|a\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle \tag{17}$$

This can also be represented as 4-D vector given by,

$$|a\rangle = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix} \tag{18}$$

As always, the 4-D state vector $|a\rangle$ must be *normalized*. Two separate qubits, $|b\rangle = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$ and $|a\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$ can be represented as a collective single state using a *tensor product* [9].

$$\begin{aligned} |ba\rangle = |b\rangle \otimes |a\rangle &= \begin{bmatrix} b_0 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \\ b_1 \times \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \end{bmatrix} \\ &= \begin{bmatrix} b_0 a_0 \\ b_0 a_1 \\ b_1 a_0 \\ b_1 a_1 \end{bmatrix} \end{aligned} \tag{19}$$

In general, an n qubit system will require 2^n complex amplitudes to keep track of the whole system. Because the number of qubits required grows exponentially, it very difficult to simulate a quantum computer (or quantum circuits) on any classical system [9].

The **CNOT** gate takes two qubits as input. This gate is a conditional gate that performs an X-gate on the target qubit, if the state of the control qubit is $|1\rangle$. A sample output of the **CNOT** gate is given by fig. 5.

The **CNOT** shown in the circuit in fig. 5 takes the following matrix form,

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{20}$$

So if,

$$|a\rangle = \begin{bmatrix} a_{00} \\ a_{01} \\ a_{10} \\ a_{11} \end{bmatrix} \tag{21}$$

then,

$$CNOT|a\rangle = \begin{bmatrix} a_{00} \\ a_{11} \\ a_{10} \\ a_{01} \end{bmatrix} \tag{22}$$

Notice that the CNOT gate swapped the amplitudes of $|01\rangle$ and $|11\rangle$ in the state vector, $|a\rangle$.

In classical computation, the **NAND** gate is known as the universal gate because any function can be computed from a composition of **NAND** gates alone. In Quantum Computing, the analogous rule states the following: "Any multiple qubit logic gate may be composed from **CNOT** gates and single qubit gates" [9].

IV. Quantum Fourier Transform

1) Definition: The discrete Fourier transform takes some vector $x = (x_0, x_1, ..., x_{N-1})$ and maps it to some other vector $y = (y_0, y_1, ..., y_{N-1})$ using the following,

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk} \tag{23}$$

where,

$$\omega_N^{jk} = e^{\frac{2\pi i jk}{N}} \tag{24}$$

Similarly, the Quantum Fourier transform (QFT), acts on a state $\sum_{i=0}^{N-1} x_i|x\rangle$ and maps to another quantum state $\sum_{i=0}^{N_1} y_i|y\rangle$ using,

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk} \tag{25}$$

where, ω_N^{jk} is defined as above. The QFT only affects the amplitudes of the state. The QFT can also be represented as the following map,

$$|x\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \omega_N^{xy} |y\rangle \tag{26}$$

In unitary matrix form,

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \omega_N^{xy} |y\rangle\langle x| \tag{27}$$

where, $\langle x|$ is the row vector representation of the column vector, $|x\rangle$. Any row vector, "bra" has a corresponding column vector, "ket" and these vectors can be converted between one another using the conjugate transpose [13].

Essentially, the QFT transforms between two basis states i.e, it takes state vectors in the Computational Basis and transforms it to a state vector in the Fourier Basis. States in the Fourier Basis are often represented as $|\widetilde{x}\rangle$ i.e,

$$QFT|x\rangle \rightarrow |\widetilde{x}\rangle \tag{28}$$

As stated above, the Hadamrd gate is the single qubit QFT because it takes the Computational Basis states, $|0\rangle$, $|1\rangle$ and transforms them to the Fourier Basis states, $|+\rangle$ and $|-\rangle$. Because the Hadamard gate acts on the single qubit state, $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, the number of amplitudes, $N = 2$. Similarly, all multiqubit states in the Computational Basis can be transformed to states in the Fourier basis. In particular, the QFT is defined for any arbitrary number of amplitudes, N.

Define QFT_N that acts on a multiqubit state, $|x\rangle = |x_1.....x_n\rangle$, where x_1 is the most significant quibit. Then, from the definition above, $N = 2^n$.

Without proof, QFT_N is presented as,

$$QFT_N|x\rangle = \frac{1}{\sqrt{N}}(|0\rangle + e^{\frac{2\pi i}{2^1}x}|1\rangle) \otimes \frac{1}{\sqrt{N}}(|0\rangle + e^{\frac{2\pi i}{2^2}x}|1\rangle) \otimes \\ ... \otimes \frac{1}{\sqrt{N}}(|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}|1\rangle) \otimes \frac{1}{\sqrt{N}}(|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle) \tag{29}$$

For a strict derivation of Eq. (29), the reader is directed to [9].

2) Circuit that implements the QFT: The QFT ciruit makes use of two gates. The first is the single qubit Hadamard gate, H, where,

$$H|x_k\rangle = \frac{1}{\sqrt{2}}(|0\rangle + exp(\frac{2\pi i}{2}x_k)|1\rangle) \tag{30}$$

The second gate is a two qubit controlled rotation, $CROT_k$, whose matrix definition is given by,

$$CROT_k|x_k\rangle = \begin{bmatrix} 1 & 0 \\ 0 & UROT_k \end{bmatrix} \tag{31}$$

$UROT_k$ is defined as,

$$UROT_k = \begin{bmatrix} 1 & 0 \\ 0 & exp(\frac{2\pi i}{2^k}) \end{bmatrix} \tag{32}$$

Assume $CROT_k$ acts on a two qubit state, $|x_l x_j\rangle$. Assuming the first qubit is the control and the second qubit is the target, $CROT_k$'s action is defined by,

$$CROT_k|0x_j\rangle = |0\rangle x_j \tag{33}$$

and,

$$CROT_k|1x_j\rangle = exp(\frac{2\pi i}{2^k})x_j|1x_j\rangle \tag{34}$$

The circuit that implements a QFT is shown in fig. 6. Notice that the circuit has been partitioned into positions, $1, 2, 3$ and 4.

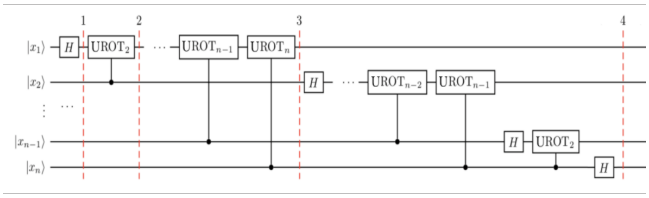


Fig. 6: Circuit that implements a QFT

Assume as input, an n-qubit input state, $|x\rangle = |x_1 x_2 ... x_n\rangle$.

1) After the first Hadamard gate at position 1 acts on qubit 1:

$$H|x_1 x_2 ... x_n\rangle = \frac{1}{\sqrt{2}}(|0\rangle + exp(\frac{2\pi i}{2}x_1)|1\rangle) \otimes |x_2 x_3 ... x_n\rangle \tag{35}$$

2) After the $UROT_2$ on qubit 1 controlled by qubit 2 in position 2, the state is given by

$$\frac{1}{\sqrt{2}}[|0\rangle + exp(\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2^1}x_1)|1\rangle] \otimes |x_2 x_3 ... x_n\rangle \tag{36}$$

3) After the last $UROT_n$ gate on qubit 1 controlled by qubit n in position 3, the state is given by

$$\frac{1}{\sqrt{2}}[|0\rangle + exp(\frac{2\pi i}{2^n}x_n + \frac{2\pi i}{2^{n-1}}x_{n-1} + ... \frac{2\pi i}{2^1}x_1)|1\rangle] \otimes |x_2 x_3 ... x_n\rangle \tag{37}$$

Observing that,

$$x = 2^{n-1}x_1 + 2^{n-2}x_2 + ... + 2^1 x_{n-1} + 2^0 x_n \tag{38}$$

Eq. (37) can be re-written as,

$$\frac{1}{\sqrt{2}}[|0\rangle + exp(\frac{2\pi i}{2^n}x)|1\rangle] \otimes |x_2 x_3 ... x_n\rangle \tag{39}$$

4) After the application of the same sequence of gates for qubits $2...n$, the final state is given by,

$$\frac{1}{\sqrt{2}}(|0\rangle + exp(\frac{2\pi i}{2^n}x)|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + exp(\frac{2\pi i}{2^{n-1}}x)|1\rangle) \otimes ... \otimes \frac{1}{\sqrt{2}}(|0\rangle + exp(\frac{2\pi i}{2^1}x)|1\rangle) \tag{40}$$

Eq. (40) is exactly the $QFT|x\rangle$ defined in Eq. (29) with the only difference being that the order of qubits is reversed. Swap operations are then used to reverse the ordering of qubits.

To analyze the performance of this circuit, it is observed that a Hadamard gate is applied to the first qubit followed by $(n-1)$ conditional rotations for a total of n gates. Similarly, a Hadamard gate is applied to the second qubit followed by $(n-2)$ conditional rotations and so on. There is an upper bound of $n/2$ swap gates that can be used. So the total number of gates is given by the function $f(n)$,

$$\begin{aligned} f(n) &= [n + (n-1) + (n-2) + ... + 1] + \frac{n}{2} \\ &= \frac{n(n+2)}{2} + \frac{n}{2} \end{aligned} \tag{41}$$

By the definition from section II, $f(n)$ is tightly bound by the n^2 term in Eq. (41) i.e., $f(n)$ is $\Theta(n^2)$. Comparing this to the performance of best classical algorithm to compute the Fourier Transform, the Fast Fourier Transform (FFT) which has a running time bounded by $\Theta(n2^n)$, it is seen that the QFT requires exponentially fewer gates than the FFT [9].

V. Quantum Phase Estimation

Quantum Phase Estimation (QPE) is a central part of Shor's algorithm. Given some unitary operator, U, the Quantum Phase Estimation Algorithm estimates θ given,

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle \tag{42}$$

Here, ψ and $e^{2\pi i\theta}$ are eigenvectors and eigenvalues respectfully of the unitary operator, U. QPE makes use of the inverse QFT. The inverese QFT, $QFT^\dagger$ is simply the QFT with all the gates inverted, noting that for arbitrary gates A, B and C,

$$(ABC)^\dagger = C^\dagger B^\dagger A^\dagger \tag{43}$$

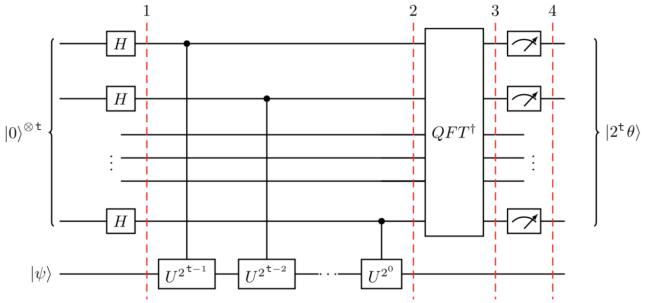


Fig. 7: Circuit that implements a QPE

The circuit that implements a QPE is shown in fig. 7.

1) $|\psi\rangle$ is in one set of qubit registers. The counting register has an additional set of t qubits which will store the value $2^t\theta$.

$$\psi_0 = |0\rangle^{\otimes t}|\psi\rangle \tag{44}$$

2) The circuit applies t Hadamard gates on the counting register, $H^{\otimes t}$.

$$\psi_1 = \frac{1}{2^{\frac{t}{2}}}(|0\rangle + |1\rangle)^{\otimes t}|\psi\rangle \tag{45}$$

3) The controlled unitary operation is applied on the target register storing, $|\psi\rangle$ only if the corresponding control bit is $|1\rangle$.

$$U^{2^j}|\psi\rangle = U^{2^{j-1}}U|\psi\rangle = U^{2^{j-1}}e^{2\pi i\theta}|\psi\rangle = e^{2\pi i 2^j\theta}|\psi\rangle \tag{46}$$

The circuit applies all t controlled operations with $0 \leq j \leq t-1$

$$\psi_2 = \frac{1}{2^{\frac{t}{2}}}(|0\rangle + e^{2\pi i\theta 2^{t-1}}|1\rangle) \otimes ... \otimes \frac{1}{2^{\frac{t}{2}}}(|0\rangle + e^{2\pi i\theta 2^1}|1\rangle) \otimes \frac{1}{2^{\frac{t}{2}}}(|0\rangle + e^{2\pi i\theta 2^0}|1\rangle) \otimes |\psi\rangle \tag{47}$$

Using the relation [9],

$$|0\rangle\otimes|\psi\rangle + |1\rangle\otimes e^{2\pi i\theta}|\psi\rangle = (|0\rangle + e^{2\pi i\theta}|1\rangle)\otimes|\psi\rangle \tag{48}$$

Eq. (47) can be rewritten as,

$$\psi_2 = \frac{1}{2^{\frac{t}{2}}}\sum_{k=0}^{2^{t-1}} e^{2\pi ik}|k\rangle \otimes |\psi\rangle \tag{49}$$

where k is the integer representation of the n bit binary numbers.

4) Comparing Eq. (49) with Eq. (40) (noting that $(\sqrt{2})^t = 2^{\frac{t}{2}}$) from Section IV, it is seen that ψ_2 is the exact result of applying a QFT by replacing x (in Eq. (40)) with $2^t\theta$ (in Eq. (49)). To recover the state $2^t\theta$, the $QFT^\dagger$ should be applied.

$$|\psi_3\rangle = \psi_2 \overset{QFT^\dagger}{\longrightarrow} \frac{1}{2^t}\sum_{x=0}^{(2^t-1)}\sum_{k=0}^{(2^t-1)} e^{\frac{-2\pi ik}{2^t}(x-2^t\theta)}|x\rangle \otimes |\psi\rangle \tag{50}$$

5) Finally, a measurement is made on the first register. ψ_3 "peaks" near $x = 2^t\theta$ when $2^t\theta \in \mathbb{Z}$ [9].

$$|\psi_4\rangle = |2^t\theta\rangle \otimes |\psi\rangle \tag{51}$$

In the case when $2^t\theta \notin \mathbb{Z}$, the above expression still peaks near $x = 2^t\theta$ with probability slightly better than $\frac{4}{\pi^2} \approx 40\%$.

VI. Shor's Algorithm

In section I, the importance and implications of factoring a number was introduced. Section II discussed how prime factorization on a classical computer has no efficient algorithm (as of yet, this may change in the future). In particular, the *general number field sieve*, the most efficient algorithm to factor a number on a classical computer runs in $O(exp(1.9(logN)^{\frac{1}{3}}(loglogN)^{\frac{2}{3}}))$, where N is the number of bits required to encode the input. In contrast, Shor's algorithm runs in $O((logN)^2(loglogN)(logloglogN))$. Ignoring the exact nature of the running times given by both algorithms, it is observed that Shor's algorithm is almost exponentially faster than the *general number field sieve*. A high-level overview of Shor's algorithm is shown in Algorithm 1.

Algorithm 1 Shor's Algorithm

1: **Problem: Find the prime factors of** N
2: $N = p \times q$, for primes p and q
3: **pick** a such that $gcd(a, N) = 1$
4: **find** smallest r such that $a^r \equiv 1 \mod (N)$
5: **if** r is even **then**
6: **define** $x \equiv a^{\frac{r}{2}}\ mod(N)$
7: **if** $x + 1 \not\equiv 0\ mod(N)$ **then**
8: $\{p, q\} = \{gcd(x + 1, N), gcd(x - 1, N)\}$
9: **done**
10: **end if**
11: **else**
12: **pick** another a such that $gcd(a, N) = 1$
13: **find** r such that $a^r \equiv 1 \mod (N)$
14: **go to line** 5
15: **end if**

In step 4, the smallest r such that $a^r \equiv 1 \mod (N)$ is known as the "order" of the function $a^r \mod (N)$. Also, in step 3, if a is picked such that $gcd(a, N) \neq 1$ i.e., $gcd(a, N) = a$, then a is one of the factors of N. N is then divided by a to find the other factor and the problem is trivially solved. While non-zero, the probability of picking such an a in step 3 is minuscule. Algorithm 1 is fundamentally how Shor's algorithm converts a factoring problem to a period finding problem. The problem now is redefined: "Given some N, find r". All the other steps can be done in polynomial time on a classical computer. The reader is directed to [6] for a detailed review on the modular arithmetic involved.

Shor's algorithm uses Quantum Phase Estimation (QPE) on the unitary operator, U. For some a, N and state $|y\rangle$, U's action is defined by,

$$U|y\rangle = |ay\ modN\rangle \tag{52}$$

For example, for $a = 3$, $N = 35$ and starting with state, $|1\rangle$, after applying U for r iterations, state $|1\rangle$ is re-obtained.

$$\begin{aligned} U|1\rangle &= |3\rangle \\ U^2|1\rangle &= |9\rangle \end{aligned} \tag{53}$$

This repeated application would continue until,

$$\begin{aligned} U^{(r-1)}|1\rangle &= |12\rangle \\ U^r|1\rangle &= |1\rangle \end{aligned} \tag{54}$$

For the given a and N, r happens to be 12. A superposition of states in this cycle would be given by,

$$|u_0\rangle = \frac{1}{\sqrt{r}}\sum_{k=0}^{(r-1)}|a^k\ modN\rangle \tag{55}$$

Notice that the eigenvalues of the states in this cycle don't have r in them. However, a different superposition ($|u_1\rangle$) could be constructed such that the phase of the k^{th} state is proportional to k. That is, in this cycle ($|u_1\rangle$), the phase is different for each of the computational basis states.

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{(r-1)} e^{-\frac{2\pi i k}{r}} |a^k \ mod N\rangle \tag{56}$$

So,

$$U|u_1\rangle = \frac{2\pi i}{r}|u_1\rangle \tag{57}$$

Eq. (57) does not represent the only eigenstate with this behaviour. To generalize this further, the phase in Eq. (56) can be multiplied by some integer, s to give

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{(r-1)} e^{-\frac{2\pi i s k}{r}} |a^k \ mod N\rangle \tag{58}$$

and,

$$U|u_s\rangle = \frac{2\pi i s}{r}|u_s\rangle \tag{59}$$

Notice that s produces a unique eigenstate where, $0 \leq s \leq (r-1)$. Summing over all possible values of s i.e., over all the eigenstates, all the states cancel out in the computational basis except $|1\rangle$ i.e,

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{(r-1)} |u_s\rangle = |1\rangle \tag{60}$$

Since, $|1\rangle$ is a superposition of all eigenstates, performing a QPE on U using the state $|1\rangle$, the measurement will peak [9] at the following value,

$$\phi = \frac{s}{r} \tag{61}$$

The range of s is 0 to $(r-1)$. The efficient continued fractions algorithm [6] can be used on ϕ to find r.

The most general form of U is given by,

$$U^{2^j}|y\rangle = |a^{2^j} y \ mod N\rangle \tag{62}$$

Computing a^{2^j} can be done efficiently on a classical computer using the repeated squaring algorithm [14]. The circuit that implements Shor's algorithm is shown in fig. 8.

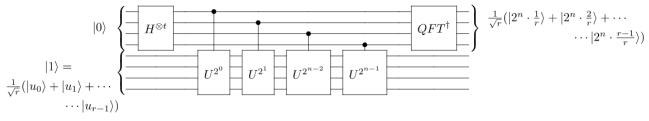


Fig. 8: Circuit for Shor's Algorithm

The top register contains the state, $|0\rangle$ onto which the order, r is to be encoded. The second register contains, $|1\rangle$, which is a superposition of states. After the U gates perform modular exponentiation, a $QFT^\dagger$ is performed on the top register. A measurement of the top register yields ϕ from Eq. (61) with high probability and, r can be obtained. Notice that Shor's algorithm, represented by fig. 8 fundamentally boils down to the QPE algorithm, represented by fig. 7.

VII. Discussion & Conclusion

Recall from Section VI that there is no exponential term in the running time of Shor's algorithm. The speed-up in Shor's algorithm stems from the cancelling out of states in the computational basis that leads to Eq. (60). Also, from Eq. (51), it is seen that measuring $2^n\theta$ on the top register where, θ is the phase is very likely in fig. 8. Once that measurement is made, the efficient classical continued fractions algorithm may be used to find r.

Significantly, Shor's algorithm solves the problem of integer factorization in polynomial time. There is no known classical algorithm that factorizes an integer in polynomial time. In fact, it is strongly believed that integer factorization does not belong to the class, **P** in classical computation i.e., the problem itself cannot be solved in polynomial time. The security protocols discussed in Section I of this paper heavily rely on integer factorization not belonging to **P** for their security.

Though originally published in 1994 [15], factorization using Shor's Algorithm is a relatively young field. The largest successful factoring was $N = 21$ published in 2012 [16]. This beat the previous record of $N = 15$ published in 2001 [17]. In recent years, physicists are realizing the limitations of current quantum hardware. In 2019, Amico et al. claims that Shor's, implemented on current hardware, fails to factor 35 [18]. Due to such hardware constraints, progress made on the pure implementation of Shor's algorithm has generally stopped in favor of combinatorial approaches using quantum adiabatic computers and parameter optimization [19]. Still, the existence of a polynomial time factoring algorithm has far-reaching implications. Shor's algorithm might be one quantum hardware advancement away from being realized in every quantum computer. If that is ever the case, security protocols that rely on **NP** problems will either become completely futile or will have to obtain their security from elsewhere.

VIII. Acknowledgements

First, I would like to thank the IBM Qiskit Textbook [20]. This is an amazing resource for someone with a Physics/Mathematics background hoping to get started with Quantum Computing. It's free and available online too!

Second and more importantly, I want to thank Professor Branning, Professor Walden, Professor Palandage and Professor Silverman. It's been an amazing four years and I learnt a lot from all of you. Thank you for everything.

References

[1] P. Mahajan and A. Sachdeva, "A study of encryption algorithms aes, des and rsa for security," *Global Journal of Computer Science and Technology*, 2013.

[2] A. Sidorenko and B. Schoenmakers, "Concrete security of the blum-blum-shub pseudorandom generator," in *IMA International Conference on Cryptography and Coding*. Springer, 2005, pp. 355–375.

[3] N. Li, "Research on diffie-hellman key exchange protocol," in *2010 2nd International Conference on Computer Engineering and Technology*, vol. 4. IEEE, 2010, pp. V4–634.

[4] A. V. Meier, "The elgamal cryptosystem," in *Joint Advanced Students Seminar*, 2005.

[5] D. Lehmer *et al.*, "On euler's totient function," *Bulletin of the American Mathematical Society*, vol. 38, no. 10, pp. 745–751, 1932.
[6] K. H. Rosen and K. Krithivasan, *Discrete mathematics and its applications: with combinatorics and graph theory*. Tata McGraw-Hill Education, 2012.
[7] Y. Tsiounis and M. Yung, "On the security of elgamal based encryption," in *International Workshop on Public Key Cryptography*. Springer, 1998, pp. 117–134.
[8] M. Sipser, "Introduction to the theory of computation," *ACM Sigact News*, vol. 27, no. 1, pp. 27–29, 1996.
[9] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.
[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.
[11] M. E. Briggs, "An introduction to the general number field sieve," Ph.D. dissertation, Virginia Tech, 1998.
[12] J. A. Carlson, A. Jaffe, and A. Wiles, *The millennium prize problems*. Citeseer, 2006.
[13] D. J. Griffiths and D. F. Schroeter, *Introduction to quantum mechanics*. Cambridge University Press, 2018.
[14] M. Stamp, *Information security: principles and practice*. John Wiley & Sons, 2011.
[15] P. 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.
[16] E. Martin-Lopez, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'brien, "Experimental realization of shor's quantum factoring algorithm using qubit recycling," *Nature photonics*, vol. 6, no. 11, pp. 773–776, 2012.
[17] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, "Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature*, vol. 414, no. 6866, pp. 883–887, 2001.
[18] M. Amico, Z. H. Saleem, and M. Kumph, "Experimental study of shor's factoring algorithm using the ibm q experience," *Physical Review A*, vol. 100, no. 1, p. 012305, 2019.
[19] B. Wang, F. Hu, H. Yao, and C. Wang, "Prime factorization algorithm based on parameter optimization of ising model," *Scientific reports*, vol. 10, no. 1, pp. 1–10, 2020.
[20] IBM Quantum Team, "Introduction to quantum computing and quantum hardware," 2020. [Online]. Available: http://qiskit.org/learn/intro-qc-qh