Shor's Algorithm

Duc Anh Nguyen
Tufts University
Chelsea, MA
duc.nguyen651872@tufts.edu

Aleksandre Avaliani
Tufts University
Medford, MA
aleksandre.avaliani@tufts.edu

***Abstract*—Quantum computing has the potential to revolutionize various fields, including cryptography, with Shor's algorithm being one of the most promising quantum algorithms for breaking classical encryption schemes. However, Shor's algorithm still has its limitations with the reliance on the construction of the unitary operation and error correction codes.**

This research paper delves into the specifics of Shor's algorithm and its limitations. We explore the challenges of implementing the algorithm, including the need for large-scale quantum computers, difficulty in implementing the Modular Exponentiation operator, Quantum Fourier Transform and the vulnerability of quantum computers to errors.

We implement Shor's algorithm in QISKIT using a simplified modular multiplication function. We estimate the success rate of the algorithm, as well as the time complexity to factor large numbers. We compare our results to reading material available online and deduce that, while our naive implementation is worse than what is available online, no current implementation of the algorithm matches the time complexity described by Peter Shor in his original paper.

Furthermore, we discuss the scalability of Shor's algorithm and the potential future work that can be done to make Shor's algorithm a reality, including improving the hardware and software capabilities of quantum computers.

***Index Terms*—Shor's algorithm, quantum computing, error correction**

I. Introduction

With the increasing complexity of mathematical and computational problems, researchers have been on the lookout for more efficient algorithms to solve them. Enter Shor's algorithm - a breakthrough in quantum computing that has the potential to revolutionize the field of cryptography by providing a polynomial-time solution to integer factorization. The computational complexity of integer factorization forms the backbone of most cryptographic systems. Naive approaches to the problem yield algorithms with time complexity scaling exponentially with the number of bits that are used to encode the number. Modern algorithms with time complexity strictly better than exponential exist, but no classical algorithm is capable of factoring integers in polynomial time. By leveraging the principles of quantum mechanics, Shor's algorithm can solve the integer factorization problem faster than any known classical algorithm.

While the algorithm shows great promises, however, there are several challenges that need to be overcome before it can be practically implemented. In particular:

- Depending on the implementation, the algorithm requires qubits up to multiple times the number of bits in the integer it is attempting to factor. Quantum computers this large do not currently exist.
- Imperfections cause each quantum operation (each quantum gate) to have *error*. The only known method of completely mitigating this error is through encoding logical qubits onto multiple physical qubits, further increasing the number of qubits required.
- To achieve polynomial time complexity, the algorithm requires a difficult-to-implement modular exponentiation function.
- Quantum Fourier Transform (QFT), which is required by the algorithm, is computationally expensive and difficult to implement on a quantum circuit.

The first two are issues of quantum computer engineering and material science. As such, they are out of the scope of this paper. Analysis of the last two points are included in Section IV.

The challenges posed in the practical implementation of Shor's algorithm raise serious questions about its scalability and the probability of its success. This paper aims to provide a comprehensive analysis of the key components that need to be addressed to make Shor's algorithm a reality, despite the numerous fallacies associated with its implementation. Specifically, we will examine potential solutions to the modular exponential function construction problem, explore the details of implementing the Quantum Fourier Transform (QFT), and evaluate the effectiveness of these methods. By providing a thorough breakdown of the key challenges and potential solutions to Shor's algorithm, this paper aims to contribute to the ongoing discussion surrounding the future of quantum computing and its impact on the field of cryptography.

II. Quantum Computing

We will provide a brief exposition of the fundamental concepts of quantum computing required for the implementation of Shor's Algorithm, including quantum superposition and quantum entanglement. Additionally, we shall discuss the unitary operations and some basic quantum gates [13].

A. Quantum States and Superposition

A quantum bit, or so called qubit, is the fundamental unit of quantum information. Unlike classical bits, which can only exist in one of two states (0 or 1), a qubit can exist in a superposition of the states 0 and 1, with a probability amplitude for each state. Mathematically, this is expressed as:

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

Where α and β are complex probability amplitudes, $|0\rangle$ and $|1\rangle$ are the orthonormal basis states that correspond to the two possible states of a classical bit. The total probabilities can be presented as:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2}$$

One of the well known ways to describe a qubit state is through a Bloch Sphere:

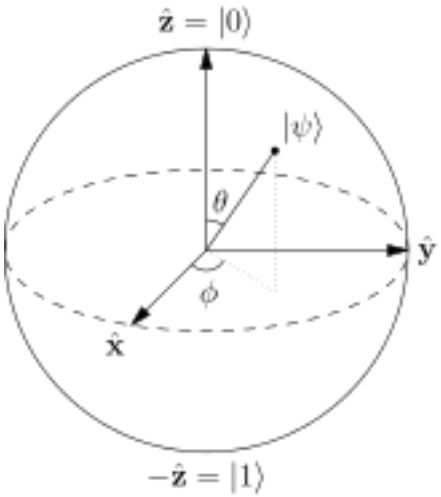

Fig. 1. All states of a qubit $|\psi\rangle$ being represented as the polar coordinate of $|\theta\rangle$ and $|\phi\rangle$. The z pole corresponds to the state $|0\rangle$ and $|1\rangle$.

The following equation describes the general state of the qubit on the Bloch sphere:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{3}$$

Reason of Importance: Quantum superposition allows a vector of qubits to exist in superposition of states. An operation on a superposition of inputs results in a superposition of outputs. The power of quantum computers comes from the fact that a set of inputs can be encoded into a single superposition. This superposition can be manipulated by operators, and the output will be the *superposition of all the inputs manipulated by the operators*. While isolating a desired output from the superposition yields no benefit (as the rest of the outputs are discarded), the output superposition can be further manipulated to provide general information about all the outputs. How Shor's algorithm leverages this property is explained in Section N.

B. Quantum Entanglement

Quantum Entanglement is a phenomenon in which two or more quantum particles become correlated, such that the state of one particle depends on the state of the other particle, even when they are separated by large distances. This correlation is not possible in classical physics and is a unique property of quantum mechanics.

The mathematical expression for quantum entanglement:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{4}$$

The state $|00\rangle$ represents the two qubits being in the state $|0\rangle$, and the state $|11\rangle$ represents the two qubits being in the state $|1\rangle$. The state $|\psi\rangle$ represents an entangled state in which the two qubits are correlated, and the state of one qubit depends on the state of the other qubit.

Reason of Importance: Quantum entanglement is crucial in Shor's Algorithm, as it allows for the simultaneous manipulation of multiple qubits. In particular, Shor's Algorithm uses a technique called "phase estimation" (this will be discussed later on), which relies on entanglement between multiple qubits to extract the period of a function.

C. Unitary Operators

A Unitary operator is a linear operator that preserves the inner product and the norm of the vectors it acts upon. In other words, it preserves the probabilities of the possible measurement outcomes.

Unitary operators are reversible, meaning that their inverse is also a unitary operator. This reversibility is important in quantum computing, as it ensures that the evolution of a quantum state can be undone.

The mathematical expression for a unitary operator acting on a quantum state can be:

$$|\psi'\rangle = U|\psi\rangle \tag{5}$$

where U is the unitary operator and $|\psi\rangle$ is the initial quantum state. The Unitary condition for an operator U is given by:

$$U^\dagger U = UU^\dagger = I \tag{6}$$

where $U^\dagger$ (pronounced U-dagger) denotes the conjugate transpose of U and I is the identity operator. This condition ensures that the operator is reversible.

Reason of Importance: The significance of unitary operators in Shor's algorithm stems from their capability to transform the qubits' state in a controlled and predictable manner. In Shor's algorithm, the implementation of the two crucial components, namely, Modular Exponentiation and Quantum Fourier Transform, depends on unitary operators. They are utilized to transform the qubits' states into a superposition of all potential measurement outcomes. Section III will provide further elaboration on the extension of this application.

D. Quantum Gates

Quantum gates are fundamental building blocks of quantum circuits and are used to manipulate qubits during computation. They perform specific unitary transformations on the state of a qubit or multiple qubits, thereby altering the probability distribution of the quantum state. These transformations enable

quantum algorithms to carry out complex calculations and perform operations that are not feasible on classical computers.

Hadamard Gate: The Hadamard gate is a single-qubit gate that puts a qubit into an equal superposition of the $|0\rangle$ and $|1\rangle$ states. It performs a rotation of π around the Z axis, followed by a rotation of $\frac{\pi}{2}$ around the Y axis in the Bloch Sphere. Its mathematical expression is:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{7}$$

Pauli-X Gate: The Pauli-X gate is a single-qubit gate that performs a classical NOT operation on the state of a qubit. It performs a rotation of π around the X axis in the Bloch Sphere. Also called the NOT gate. Its mathematical expression is:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{8}$$

Pauli-Y Gate: The Pauli-Y gate is a quantum gate that acts on a single qubit, flipping the sign of the complex amplitude of the state $|1\rangle$ and exchanging it with the state $|0\rangle$. It performs a rotation of π around the Y axis in the Bloch Sphere. Its mathematical expression is:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{9}$$

Pauli-Z Gate: The Pauli-Z gate is a single-qubit gate that acts as a phase shift gate. It performs a rotation of π around the Z axis in the Bloch Sphere. Its mathematical expression is:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{10}$$

CNOT Gate: The simplest gate to create entanglement between two qubits, meaning that it creates a two qubit state which cannot be expressed as a direct product of two single qubit states. Its mathematical expression is:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{11}$$

Reason of Importance: In Shor's algorithm, quantum gates are used to implement two core components: Modular Exponentiation and Quantum Fourier Transform. These operations require precise manipulation of the qubit states, which is achieved through a sequence of unitary operations. Moreover, quantum gates are necessary for implementing error correction codes that can preserve the integrity of quantum information.

III. Shor's Algorithm

A. Integer Factorization and Historical Context

Integer factorization, the process of decomposing a composite number into its prime factors, has been a mathematical challenge for centuries. It has been considered as one of the most important and difficult problems in number theory. The difficulty of this problem is due to the fact that there is no known efficient classical algorithm to factorize large numbers into their prime factors. This notion is the foundation of the RSA encryption algorithm, which is widely used in today's network security.

RSA is a public-key cryptography system invented by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. The security of RSA is based on the difficulty of factoring large composite numbers. The algorithm works by generating a public and private key pair, where the public key is used to encrypt messages and the private key is used to decrypt them [14]. The security of the system relies on the fact that it is computationally infeasible to factorize large numbers into their prime factors.

A plethora of classical algorithms have tackled this problem. The simplest to understand (but most laborious) algorithm is trial divison, explained below. State-of-the art algorithms are capable of factoring large integers in subexponential time, but none are capable of approaching polynomial time. Some of the best known classical algorithms are included below:

Trial Division: This algorithm is the simplest method of factorizing two integers. It works by checking if the number is divisible by any smaller number except '1'. The algorithm starts with the smallest prime number '2' and divides the integer by it. If the remainder is zero, the divisor is a factor of the integer. If not, the algorithm moves to the next prime number and repeats the process until either a factor is found or all the primes up to the square root of the integer have been checked. If no factor is found, then the integer is assumed to be a prime number. Trial division is exceedingly easy to implement, but suffers from exponential time complexity.

The time complexity of this algorithm is:

$$O(\sqrt{N}) \tag{12}$$

where N is the integer to be factored. This is equal to

$$O(\sqrt{2^n}) = O(2^{n/2}) \tag{13}$$

where n is the number of bits used to represent the integer N.

General Number Field Sieve: This is currently the fastest known algorithm for factoring large integers. The algorithm works by first selecting a set of numbers that are relatively prime to the number being factored [16]. These numbers are then used to find a set of pairs of numbers that satisfy a set of linear equations. Next, the algorithm searches for sets of pairs of numbers that satisfy a set of quadratic equations. These sets of pairs are then combined to form polynomials, which are then factored over a finite field. Finally, the factors of the polynomial are combined to produce the factors of the original number.

The time complexity of this algorithm is:

$$O\left(\exp\left(\sqrt[3]{\frac{64}{9}}(\ln n)^{\frac{1}{3}}(\ln \ln n)^{\frac{2}{3}}\right)\right) \tag{14}$$

B. Introduction

Shor's algorithm is a quantum computing algorithm designed for integer factorization, first introduced by the mathematician Peter Shor in 1994. This algorithm is considered to be among the most promising approaches in the field of quantum computing. Shor's algorithm capitalizes on the fundamental principles of quantum mechanics such as superposition and entanglement to efficiently discover the prime factors of an integer N. It is noteworthy that the most efficient classical algorithm for this task, the General Number Field Sieve, which exhibits sub-exponential time complexity, required a period of two years to factorize a 768-bit integer using a distributed computing approach [18].

In contrast, the time complexity of Shor's algorithm is:

$$O((\log n)^3) \tag{15}$$

The figure presented below illustrates the difference in time complexity between the General Number Field Sieve (GNFS) and Shor's algorithm. The x-axis represents the number of bits, while the y-axis represents the time required for factorization.

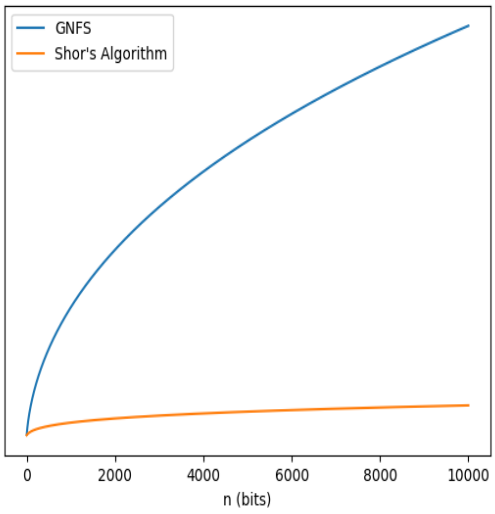

Fig. 2. Time complexity comparison of Shor's Algorithm and GNFS

It is evident that Shor's algorithm outperforms GNFS in terms of time complexity, which is why it is considered revolutionary in the field.

C. Implementation

Shor's algorithm is a quantum algorithm that can be used to efficiently factorize large composite integers into their prime factors. This algorithm exploits the quantum mechanical properties of superposition and entanglement to achieve exponential speedup compared to classical algorithms. The algorithm involves the following steps:

1) Choose the number N to be factorized, which is assumed to be a semi-prime number with two prime factors p and q.
$$N = pq \tag{16}$$
2) Choose a random number a such that a and N are co-prime (meaning that their greatest common divisor is 1). Notice that due to the semi-prime nature of N, it is exceedingly easy to come up with an integer a that is co-prime to N (that is, every integer that is not p or q will be co-prime to N). Nevertheless, an efficient algorithm for checking whether two integers are co-prime exists and is called Euclid's Algorithm, named after Greek mathematician, logician, and geometer Euclid.
3) Find the period r of the modular exponentiation function described below.
$$a^r \mod N \tag{17}$$
This is done solving for
$$a^r = 1 \mod N \tag{18}$$
This step is where the quantum advantage comes into play. The best known classical algorithm to solve this function is the Schönhage-Strassen algorithm with the time complexity of $O(l^2 \log\log l)$. With quantum computing, we can encode all desired values for r into a single superposition, and then take this superposition through a sequence of operations that leaves only the superposition of values that satisfy the problem statement. The time complexity of this approach is difficult to estimate without quantum hardware, but the depth of the quantum circuit performing this operation (which is directly correlated with time complexity) is discussed in Section IV.
Measuring this superposition will result in a single r satisfying the problem statement. Notice that $r = 0$ is always a solution to the equation. It is not the only one either: just like any periodic function, equation (18) has infinitely many solutions, with only one solution being the fundamental period. It is not possible to isolate the fundamental period, so any measurement taken is used for the next step, and in the case that it does not work, this step is repeated until a different measurement is taken.
4) Once the period r is obtained, a factor of N can be computed with high probability by using these two equations.
$$gcd(x^{r/2} + 1, n) \tag{19}$$
$$gcd(x^{r/2} - 1, n) \tag{20}$$
If the period r obtained gives us a nontrivial factor of N, we may stop the algorithm. However, there may be cases where the obtained period is not the correct one (such as when the measured period is odd or zero), in which case the previous step needs to be repeated until the equations above result in a nontrivial factor of N. Further discussion is in Section V.

Notice: the reason this approach gives factors of N is due to a property of being (semi-)prime, described by Euler's Theorem. The explanation is outside the scope of this paper. For further reading, consult Leonhard Euler's original work, [17]

After reviewing the steps of Shor's algorithm, it is apparent that only step 2 requires the capabilities of a quantum computer, as the remaining steps can be easily performed on classical computers. These steps are known as the classical pre- and post-processing.

The modular exponentiation function is one that can easily be implemented on a classical computer (using a method known as *repeated squaring*), but is exceedingly difficult to implement on a quantum circuit [1]. The reason it needs to be implemented on a quantum circuit is because the function needs to act on a quantum superposition, something that classical circuits are not capable of. The next section will discuss the details.

IV. Period Finding

A. Modular Exponentiation

Implementing (18) in a quantum circuit is identical to implementing a unitary operation that acts on a known input, multiplying it by a power of a and giving the result in $\mod N$.

$$U(x)|y\rangle = |ya^x \mod N\rangle \tag{21}$$

where y is some quantum state and $U(x)$ is a unitary operator. We can set $y = 1$ to get

$$U(x)|1\rangle = |a^x \mod N\rangle. \tag{22}$$

As a quantum superposition can only be measured once, we require a second register to hold the output. The qubits that make up this register are referred to ancilia qubits and are the reason why all implementations of Shor's algorithm require more qubits than just those used to denote N. This translates the equation we want our circuit to implement to

$$U(x)|1,0\rangle = |1,a^x \mod N\rangle. \tag{23}$$

The issue with the approach described above stems from the fact that this modular exponentiation circuit is exceedingly hard to implement. Instead, we choose an easier approach by using modular multiplication instead of exponentiation.

$$U|1,0\rangle = |1,a \mod N\rangle \tag{24}$$

This approach makes it easy to implement the operation, but also comes with a drawback: to get the same effect as the previous $U(x)$, U needs to be applied x times, which increases the depth of the circuit, which, in turn, is directly correlated with time complexity. Essentially, using this approach sacrifices time complexity for the sake of ease of implementation. Example implementations of $U(x)$ are included in [15] [6], but both are theoretical works, meaning that the do not demonstrate the functionality, only the mathematical proof of it.

B. Quantum Fourier Transform

In classical computing, the Discrete Fourier Transform (DFT) takes a finite sequence of equally spaced samples of a function and transforms it into a sequence of complex numbers, representing the original function in the frequency domain. The time complexity of DFT is $O(n^2)$ which required the brute force calculation of n x n matrix multiply with the vector to be transformed. The Fast Fourier Transform (FFT) do this more efficiently by breaking the sequence of data points into smaller subsets, performing DFT on these subsets, and then combining the results to obtain the final frequency components. This process takes advantage of the symmetry of the DFT, allowing for fewer computations to be performed.

Consider the formula for the discrete Fourier transform (DFT) of a vector x of size N:

$$X_k = \sum_{j=0}^{n-1} x_n e^{-i2\pi kj/n} \tag{25}$$

The time complexity of FFT is:

$$O(n \log n) \tag{26}$$

The FFT algorithm exploits the fact that the DFT can be computed recursively by splitting the input vector into even and odd parts, computing the DFT of each part separately, and then combining the results. This reduces the number of computations required from n^2 to $n \log n$.

Quantum Fourier Transform, on the other hand, takes advantages of quantum mechanics of superposition to perform DFT in parallel. The QFT can be thought of as a sequence of quantum gates that implement the DFT formula in a highly efficient manner [2].

The time complexity of QFT is:

$$O(\log N)^2 \tag{27}$$

In mathematical terms, the QFT operates on a quantum state $|\psi\rangle$ of n qubits, where $|\psi\rangle$ can be written as a linear combination of 2^n basis states:

$$|\psi\rangle = \sum_{k=0}^{2^n-1} c_k|k\rangle \tag{28}$$

The QFT acts on this state to produce a new state $|\phi\rangle$:

$$|\phi\rangle = \sum_{j=0}^{2^n-1} (\sum_{k=0}^{2^n-1} e^{\frac{2\pi ijk}{2^n}}|j\rangle) \tag{29}$$

where i is the imaginary unit and j is the index of the output qubit.

In simple terms, the QFT maps a quantum state $|\psi\rangle$ to a new quantum state $|\phi\rangle$, where the amplitudes of $|\phi\rangle$ are related to the frequency components of $|\psi\rangle$ [10]. It does this by applying a series of controlled-phase rotations to the state $|\psi\rangle$ and then applying Hadamard gates to each qubit [2]. The next subsection will see this being applied in Shor's algorithm.

C. Implementation

Using the extent of the knowledge that we discussed from the previous sections, these are the steps to solve the modular exponentiation problem with quantum computing:

1) Select a number q such that $n^2 < q < 2n^2$. The selection of q enables the transformation of QFT to be carried out in polynomial time [1].
2) Create two quantum registers denote as s and t where s is the input register and t is the output register. The state of our quantum registers can be expressed as:

$$|s, t\rangle \tag{30}$$

3) Load an equally weighted superposition of all the qubits into register s using the Hadamard gate, and prepare the state of register t to be $|0\rangle$. This step is crucial for the algorithm to function correctly and achieve the desired outcome [1].
The state of the quantum registers can be expressed as:

$$\frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x, 0\rangle \tag{31}$$

With x being the states of the input register s and 0 being the prepared state of the output register t.
4) Apply the unitary operators with the function $f(x) = a^x \mod N$ for each bit in register s and store the results in the output register t.
The state of the quantum registers can be expressed as:

$$\frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x, a^x \mod N\rangle \tag{32}$$

5) Denote the value of the output register t as k where:

$$a^x = k \mod N \tag{33}$$

The next step is to measure the output register t. This measurement collapses the second register into the state corresponding to the observed value k. This collapse has the side effect of collapsing the first register as well, which is caused by the phenomenon of quantum entanglement. More specifically, the state of the first register collapses into an equal superposition of all possible values of x between 0 and $q-1$ such that $a^x = k \mod N$. This is because the function $f(x) = a^x \mod N$ is periodic with period r, and any value of x that satisfies the equation $a^x = k \mod N$ must also satisfy $a^{x+r} = k \mod N$.
6) Apply the Quantum Fourier Transform on the input register s and transform it into the following expression with $|c\rangle$ being the coefficients of QFT:

$$|x\rangle = \frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |c\rangle * e^{2\pi \frac{ixc}{q}} \tag{34}$$

QFT is used to extract the period r from the superposition of values in the register s. It converts the superposition of values in the first register into a frequency distribution of $\frac{q}{r}$ that can be measured to determine the period r.
7) Replace x with r in the function $f(x) = a^x \mod N$ and compare the result with the value k extracted from register n. If the value is different, errors might have occurred during computation, so the algorithm must be re-executed. There are several reasons behind this error, which will be discussed in Section V.
8) If we reached this step, use the period r to perform step 4 of the implementation of Shor's algorithm discussed in Section III.

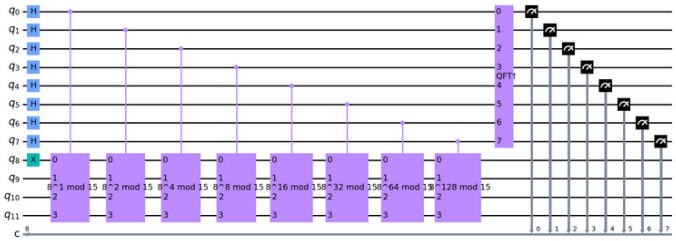

Fig. 3. QISKIT Circuit for Shor's Algorithm. Does not contain classical pre- and post-processing

V. Experimental Architecture and Results

A. Implemented Circuit

Using the simplified modular multiplication function (instead of the modular exponentiation function), we implemented Shor's Algorithm in QISKIT. The diagram of the circuit for $N = 15, a = 8$ is shown in Figure 3. As the number we are trying to factor is 15, our circuit uses 4 ancilia qubits for a total of $4 + 8 = 12$ qubits.

B. Measurments

As mentioned in Section III and further discussed in Section VI, not all measurements from the quantum circuit are usable. As such, the circuit has to be run until a good period r has been found. Included in the figure FIGURE NUMBER are the results of running the phase estimation circuit 1024 times.

Notice that even for N as small as 15, the circuit finds 4 unique periods for $a = 7$. One of these periods is 0, while another one is odd, meaning that 50% of measured periods are not useful for factoring N. We can model the number of runs until the first success as a Bernoulli random variable with parameter p, which is equal to the ratio of successful results to all measured results. For our case, $p = 0.5$ and thus the number of expected runs until a successful result is achieved is $\frac{1}{p} = 2$. This is confirmed by experimentation, which oversaw that the attempts required to factor $N = 15 = 3 * 5$ ranged from 1 to 3. We can generalize this equation to all numbers:

$$a^r = 1 \mod N \tag{35}$$

Multiplying both sides by a, we get

$$a^{r+1} = a \mod N \tag{36}$$

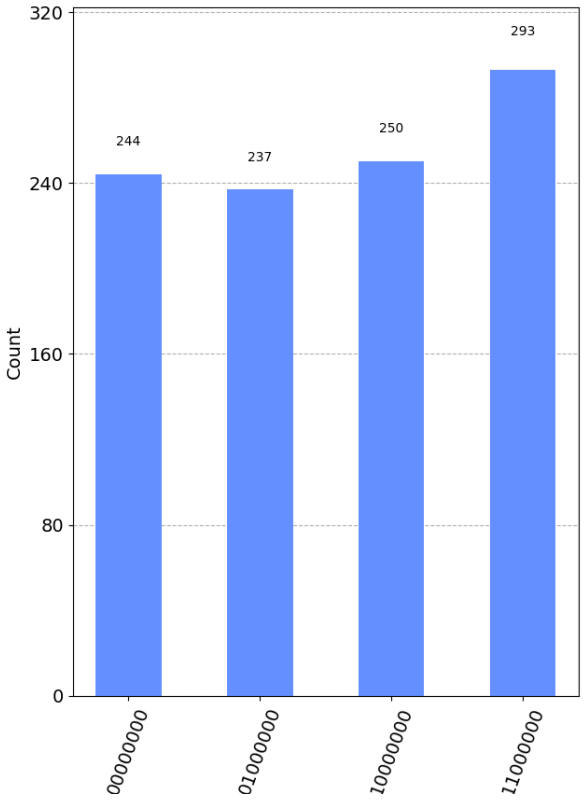

Fig. 4. Output of the implemented circuit after 1024 runs. The X axis contains measurements, while the Y axis contains the count of each measurement

Continuing to do so, we get

$$a^{r+k} = a^k \mod N \tag{37}$$

We see that the sequence of powers of a modulus N repeats with period r. This means that, since the sequence repeats every r terms, the number of periods is equal to period r.

From the measured periods, we can assume that one will be 0, and can estimate that the half of the rest is odd, giving us a ratio of useful and not useful results of $\frac{r-1}{2r}$. Plugging this back into the expectation formula for a Bernoulli random variable, we deduce that the expected number of runs required to get a successful result is

$$\frac{2r}{r-1} \tag{38}$$

where r is a period of $a^x = 1 \mod N$. This number approaches 2 as r approaches infinity, so we state that the expected number of runs until a good measurement is made is 2 in the worst-case scenario.

C. Time Complexity Analysis

The depth of the modular multiplication function is 1. This is because the operation performs up to $\log_2 n$ SWAP operations, where n is the number of qubits used to denote N, but all SWAP operations can be performed in parallel. As the modular multiplication function must be applied 2^b times to the b-th qubit, the maximum total depth of the modular exponentiation circuit is

$$\sum_{b=0}^{n} 2^b = 2^{n+1} - 1 \tag{39}$$

Notice that, due to using the simplified modular multiplication function, the depth of the circuit scales exponentially with the number of qubits. We can assume that if the circuit was implemented on an actual quantum computer, the time it takes to run the circuit would be directly proportional to the depth. As the equations above are upper bounds of depth, we can assume that these values directly translate to time complexity. Thus, the overall time complexity of the circuit, which is the sum of the time complexities of the modular exponentiation function and inverse QFT is

$$O((\log n)^2 + 2^{n+1} - 1) = O(2^n) \tag{40}$$

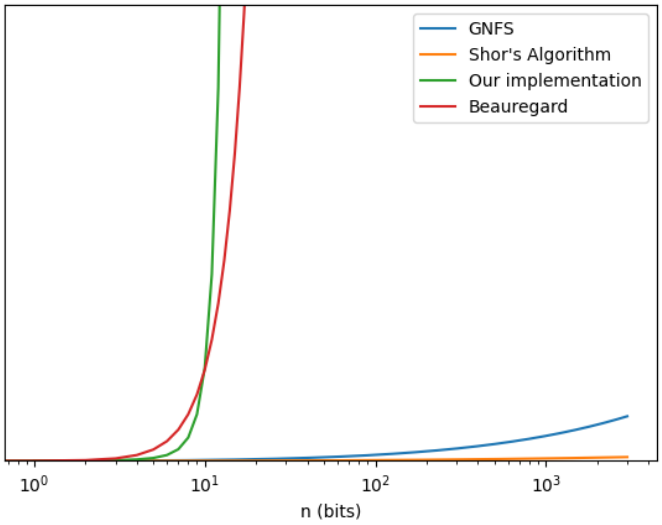

Fig. 5. Time Complexities of GNFS, Theoretical Shor's Algorithm, Beauregard [6], and our implementation

Figure 5 shows the comparison of our algorithm along with the one from [6], as well as GNFS and the theoretical Shor's algorithm. Our inability to implement a proper modular exponentiation function is the reason behind the poor performance. It must be noted that Beauregard [6] is the same magnitude as other implementations found online (see Subsection D), so no current implementation matches the theoretical minimum time complexity that Peter Shor described in his paper [1].

D. Scalability

Shor's algorithm is a prominent algorithm in quantum computing that holds the potential to revolutionize cryptography by factoring large numbers exponentially faster than any known classical algorithm. Despite its theoretical limitations, Shor's

algorithm remains a subject of ongoing research in the field of quantum computing.

This section aims to investigate the scalability of Shor's algorithm, particularly its subroutine, the Quantum Fourier Transform. We analyze and compare four papers by Beauregard [6], Vedral [4], Zadkam [7], and Gosset [5] to assess the scalability of their proposed circuits. We consider two essential metrics, the number of qubits required (space) and the depth of each circuit (time). In this context, the depth of a circuit refers to the minimum number of 2-qubit gates that must be applied sequentially to complete the circuit. A more comprehensive description of their circuits can be found in our bibliography.

TABLE I
OTHER IMPLEMENTATIONS OF SHOR'S ALGORITHM. n IS THE NUMBER OF QUBITS USED TO DENOTE N.

Circuit	Qubits	Depth
Beauregard [6]	$2n$	$32n^3$
Vedral [4]	$5n$	$240n^3$
Zalka [7]	$50n$	$2^{17}n^2$
Gosset [5]	$O(n^2)$	$O(n \log n)$

It is evident that a trade-off exists between the number of input qubits and the depth of circuits in quantum computing. While reducing the depth of a circuit can be achieved by utilizing additional qubits, this approach may increase the likelihood of errors occurring due to the inherent fragility of qubits. Conversely, increasing the depth of circuits can lead to more precise results, but it often requires a greater number of physical qubits. However, it is important to note that computational errors in these systems are typically neglected. Therefore, it is necessary to investigate the implementation of error correction codes using physical qubits to improve the accuracy of quantum algorithms. In this paper, we have only discussed the number of physical qubits required for error correction codes and have not delved into the details of these codes. Some of the well-known error correction codes include:

Shor code: Shor's code is a quantum error-correcting code that encodes a single logical qubit into nine physical qubits [12]. The purpose of the code is to correct a single bit flip error and a single phase flip error. The number of physical qubits needed for Shor's code is proportional to n^3, where n represents the number of logical qubits.

Steane code: The Steane code requires seven physical qubits for every logical qubit to implement error correction [8]. In terms of scalability, the Steane code is capable of detecting and correcting any single-qubit error and any combination of two-qubit errors. However, due to its inability to correct three or more errors, its scalability for larger quantum computations is limited.

Surface code: The Surface code works by placing physical qubits on the edges of a two-dimensional lattice or surface, forming a checkerboard pattern [11]. The physical qubits required will largely depend on the construction of the lattice, but it takes a minimum of thirteen physical qubits to implement a single logical qubit. A reasonably fault-tolerant logical qubit that can be used effectively in a surface code takes of order 10^3 to 10^4 physical qubits [13].

Based on the research we have conducted, it is evident that the scalability of Shor's algorithm remains a challenge, with physical qubits potentially growing exponentially in size to match the logical qubits of the system.

VI. FUTURE WORK

A. Modular Exponentiation

Our inability to implement the modular exponentiation circuit without repeatedly using the modular multiplication circuit caused our implemented algorithm to have time complexity that is worse than the trial division method. Future work should focus on attempting to implement this operator. [15] and [6] have theoretical implementations for this circuit, but they do not provide the experimental results of simulating their circuit, or implementing it on a quantum computer. The circuit in [15] is especially impressive as it claims to have a polynomial time complexity. The complexity of this circuit, however, makes it challenging to implement without having a guarantee that the circuit performs what it says it does.

B. Errors of Operators

Based on our research, it is clear that the scalability of Shor's Algorithm remains a challenging issue, particularly when it comes to the two core components: Modular Exponentiation and Quantum Fourier Transform. These components require a substantial number of physical qubits to implement, and errors can arise during their execution. In our experiment, we chose to neglect the usage of error correction codes and the errors that may occur in each logical qubit. However, it is crucial to consider the impact of such errors on the accuracy of the results obtained from the algorithm.

In a paper presented by C. Ray Hill [9], it was shown that quantum computation is efficient (polynomial in logN) when there are no errors in the quantum computer performing the calculation. However, the introduction of errors during the QFT period finding part of the quantum computation can degrade the algorithm's ability to find the period. The same paper also highlights that computation errors of the operators can break the polynomial scaling of Shor's algorithm. Therefore, hardware implementation techniques needs to be researched to ensure that the error rate of operators and gates does not influence the correctness of quantum computations. Error analysis should be the main focus of our future work.

VII. CONCLUSION

This work oversaw the design of Shor's integer factorization algorithm using quantum circuits. The success rate, as well as the time complexity of the implemented algorithm was measured and contrasted with existing reading material.

REFERENCES

[1] P. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," IEEE Computer Society Press, pp. 124-134, 1997.

[2] D. Camps, R. Van Beeumen, and C. Yang, "Quantum Fourier Transform Revisited," Numerical Linear Algebra with Applications, vol. 13, no. 8, pp. 687-698, 2006.
[3] A. G. Fowler and L. C. L. Hollenberg, "Scalability of Shor's algorithm with a limited set of rotation gates," IEEE Transactions on Computers, vol. 53, no. 7, pp. 887-893, 2004.
[4] V. Vedral, A. Barenco, and A. Ekert, "Quantum Networks for Elementary Arithmetic Operations," Physical Review A, vol. 52, no. 1, pp. 147-153, 1995.
[5] P. Gosset, "Quantum Carry-Save Arithmetic," quant-ph/9808061, 1998.
[6] S. Beauregard, "Circuit for Shor's algorithm using 2n+3 qubits," Quantum Information and Computation, vol. 3, no. 6, pp. 175-185, 2002.
[7] C. Zalka, "Fast versions of Shor's quantum factoring algorithm," arXiv preprint arXiv:quant-ph/9806084, 1998.
[8] A. Steane, "Multiple Particle Interference and Quantum Error Correction," Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 452, no. 1954, pp. 2551-2577, 1996.
[9] C. R. Hill and G. F. Viamontes, "Operator Imprecision and Scaling of Shor's Algorithm," Lockheed Martin Advanced Technology Laboratories, 2004.
[10] F. X. Lin, "Shor's Algorithm and the Quantum Fourier Transform. McGill University," 2004.
[11] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," Physical Review A, vol. 86, no. 3, pp. 032324-1-032324-25, 2012.
[12] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane, "Quantum error correction via codes over GF(4)," Institute for Defense Analyses, Princeton, New Jersey 08540, 1996.
[13] D. Mermin, *Quantum Computer Science: An Introduction*, Cambridge University Press, 2007.
[14] Rivest, R., Shamir, A., Adleman, L. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." Communications of the ACM, 21(2), 120-126, 1978.
[15] A. Pavlidis and D. Gizopoulos, "Fast quantum modular exponentiation architecture for shor's factorization algorithm," arXiv.org, 04-Nov-2013. [Online]. Available: https://arxiv.org/abs/1207.0511. [Accessed: 05-May-2023].
[16] A. K. Lenstra, H. W. Lenstra Jr., M. S. Manasse, and J. M. Pollard. "A general number field sieve algorithm for factoring polynomials".Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (FOCS), Nashville, TN, USA, 1981.
[17] L. Euler, "Commentarii Academiae Scientiarum Imperialis Petropolitanae," Google Books. [Online]. Available: https://books.google.com/books?id=-ssVAAAAYAAJamp;pg=RA1-PA141v=onepageamp;qamp;f=false. [Accessed: 05-May-2023].
[18] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. te Riele, A. Timofeev, and P. Zimmermann, "Factorization of a 768-bit RSA modulus," SpringerLink, 01-Jan-1970. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-14623-7-18. [Accessed: 05-May-2023].