Shor's Algorithm for Factoring Numbers

Riccardo La Marca 1795030

July 1, 2023

Contents

1 Introduction

Shor's algorithm is a quantum computing algorithm for finding the prime factors of an integer. While in a classical computer it is believed that *factoring* is a very hard problem, and no classical algorithm exists that are able to solve the problem in polynomial time, Shor's have found an efficient algorithm for an ideal quantum computer that is able to factor any given number in $O((\log N)^2(\log\log N)(\log\log\log N))$ where N is the number to be factorized. Considering $n = \log N$ we have the total complexity of $O(n^2 \log n \log\log n)$ where log is referred to as the logarithm in basis 2.

The important of such an algorithm can be directly found in cryptography. Until now, the most important and secure *public-key* encryption scheme is the RSA. The safety of RSA is entirely based on the assumption that factoring large integers in polynomial time is computationally intractable (at least on classical computers). Given the Shor's algorithm seems to be possible in poly-logarithmic time which pose a big deal for cryptography. However, we have to recall something that must not bee forgiven: actually RSA keys are about 2048-bits long. In the following we are going to show an implementation of the Shor's algorithm that requires $2n + 3$ qubits, where n is the number of bits for the number of the factorized. Being $n = 2048$, we requires about 4099 qubits. Moreover, recall that a multi-qubits system can be represented as the tensor product between sub-systems's statevectors, i.e., meaning:

$$|\Psi\rangle = |\psi_1 ... \psi_n\rangle = \bigotimes_{i=1}^{n} |\psi_i\rangle = \sum_{i=1}^{2^n} \Psi_i |\Psi_i\rangle$$

which requires approximately 2^n probabilities amplitudes, given n the number of qubits. Thus, for a 4099 qubit system we requires $2^{4099} \approx 8.3551 \times 10^{1233}$ amplitudes. At this point no quantum computer with that feature can exists due to the high temperatures and due to the energy efficiency costs.

Shor's algorithm is based on few important technologies of quantum computing: **Quantum Fourier Transform**, **Phase Estimation** and **Order Finding**, **Modular Multiplication and Exponentiation** that we are going to discuss in a while. However, first I'm going to show you the entire algorithm for factoring that will use the Shor's algorithm as the quantum part.

2 The Factoring Algorithm

Here is the overall algorithm using for factorize a number $N \geq 15$.

Algorithm 1 Factoring Algorithm with Shor Quantum Part

Require: $N \geq 15$
  $F = []$
  if N is even **then**
    $F \leftarrow F + [2]+$ Restart the algorithm with $N/2$
  end if
  if $\exists a \geq 1$ and $b \geq 2$ such that $N = a^b$ **then**
    $F \leftarrow F + [\underbrace{a...a}_{\text{b times}}]$ and Return F
  end if
  while Not all $1 < x < N$ have been controlled **do**
    $x \leftarrow$ select at random in the range $[2, N-1]$
    if x has been already used **then**
      Select another x
    end if
    $g \leftarrow \gcd(x, N)$
    if $g > 1$, it is not trivial **then**
      $F \leftarrow F + [g]+$ Restart with N/g and Break the loop
    end if
    $f =$ Compute Shor on x and N
    if f is empty **then**
      Select another x
    else
      $F \leftarrow F + f$ and Break the loop
    end if
  end while
  if No new factors using Shor **then**
    $F \leftarrow F + [N]$
  end if
  Return F

2.1 The Quantum Part - Shor's Algorithm

The idea of the Shor's algorithm is directly linked with the so-called *period finding*, or often called *order finding*, problem. Given two positive integers x, N such that $x < N$ the *order* of x modulo N is the least positive integer r such that $x^r \equiv 1 \mod N$. Once such a r has been found then we compute two different GCD:

$$\gcd(a^{r/2} + 1, N) \quad \gcd(a^{r/2} - 1, N)$$

If at least one between these two greater common divisor are non-trivial, meaning they are greater than 1 and different from N, then we can return it as factors of N. If neither the first nor the second are non-trivial, we restart the Shor's algorithm with a new random $1 < x < N$. Here is the complete algorithm.

Algorithm 2 Shor's Algorithm

Require: $N \geq 15$
Require: $1 < x < N$
  $r \leftarrow$ Order of x modulo N
  $g_1 \leftarrow \gcd(a^{r/2} + 1, N)$
  $g_2 \leftarrow \gcd(a^{r/2} - 1, N)$
  $f \leftarrow []$
  if g_1 is non-trivial **then**
    $f \leftarrow f + [g_1]$
  end if
  if g_2 is non-trivial **then**
    $f \leftarrow f + [g_2]$
  end if
  Return f

Computing the order of a number modulo another positive integer requires the computation of the so-called *modular exponentation*, which requires *order finding* and the *quantum fourier transformation*. Hence, let's start by describing the QFT.

3 Quantum Fourier Transformation

The *Fourier Transformation* is one of the most known and important function transformation. It is used in many scientific fields like maths, computer science with signal and audio processing, and also in Quantum Mechanics, since many physical quantities are linked by a Fourier Transformation relation, like *momentum* and *position*. The Fourier Transform comes into two principal form: a *continuous* form and a *discrete* form.

3.1 The Continuous and Discrete Fourier Transform

The Continuous Fourier Transform is an operator that takes as input a function $f : \mathbb{R} \to \mathbb{R}$ defined over the real domain and returns a new function $\hat{f}$ in the real domain as well using the following transformation:

$$\hat{f}(\xi) = \mathcal{F}[f(x)] = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \xi x} \mathrm{d}\xi \tag{1}$$

such that, whenever the independent real variable x represents **time**, then the variable ξ represents **frequency**. The Fourier Transform is an invertible function, and the inverse is represented as:

$$f(x) = \mathcal{F}^{-1}[\hat{f}(\xi)] = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{-2\pi i \xi x} \mathrm{d}x \tag{2}$$

Obviously, the Fourier Transformation has some important properties that will not be discussed in in this place. The other types of Fourier Transformation is the *Discrete Fourier Transformation* which instead acts on discrete samples of sequence of complex numbers returning another sequence of complex numbers. That is, let $x_1, ..., x_N \subseteq \mathbb{C}^N$ be the input sequence and $X_1, ..., X_N \subseteq \mathbb{C}^N$ the transformed sequence, we have that x_i is mapped to X_i following:

$$X_j = \sum_{k=0}^{N-1} x_k e^{-\frac{2\pi i}{N} jk} = \sum_{k=0}^{N-1} x_k \left[\cos\left(\frac{2\pi}{N} jk\right) - i \sin\left(\frac{2\pi}{N} jk\right) \right] \tag{3}$$

In the more general terms, the DFT (Dicrete Fourier Transformation) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the *Discrete-time Fourier Transform*. The DTFT can be seen as the link between the continuous and

the discrete form, since the DTFT is a continuous function acting on discrete data. The DTFT from uniformly spaced samples it produces a function of frequency that is a periodic summation of the continuous Fourier Transform of the original continuous function. Just like the continuous form, also the discrete form has an inverse, which can be computed as:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \tag{4}$$

One important property of the DFT is that it is a unitary transformation, meaning that it is possible to create a unitary matrix that represents the operator. Recall that a matrix is *unitary* whenever it is Hermitian, meaning that the inverse is equal to the conjugate transpose. In order to obtain the unitary matrix, let start with the following matrix representation

$$F = \begin{bmatrix} \omega_N^{0\cdot 0} & \omega_N^{0\cdot 1} & \dots & \omega_N^{0\cdot (N-1)} \\ \omega_N^{1\cdot 0} & \omega_N^{1\cdot 1} & \dots & \omega_N^{1\cdot (N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^{(N-1)\cdot 0} & \omega_N^{(N-1)\cdot 1} & \dots & \omega_N^{(N-1)\cdot (N-1)} \end{bmatrix}, \quad F^{-1} = 1/NF^* \tag{5}$$

where $\omega_N = e^{-i2\pi/N}$ is a primitive Nth root of unity. With unitary normalization constant $1/\sqrt{N}$, the DFT becomes a unitary transformation with unitary matrix U defined as:

$$U = \frac{1}{\sqrt{N}} F, \quad U^{-1} = U^\dagger, \quad |\det U| = 1 \tag{6}$$

with

$$U_{m,n} = \frac{1}{\sqrt{N}} \omega_N^{(m-1)(n-1)} = \frac{1}{\sqrt{N}} e^{-\frac{2\pi i}{N}(m-1)(n-1)} \tag{7}$$

3.2 Definition of the QFT

The **Quantum Fourier Transformation** is similar in its action to the DFT, but it is applied to a generate state vector $|x\rangle = x_0|0\rangle + ... + x_{N-1}|N-1\rangle$. The QFT changes the probabilities amplitudes of the input state vector according to the following expression:

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{\frac{2\pi i}{N} jk} \tag{8}$$

At this point let $|x\rangle$ and $|y\rangle$ defined on the orthonormal basis $|0\rangle, ..., |N-1\rangle$ such that $|y\rangle$ is the QFT of $|x\rangle$, we have the following mapping:

$$|x\rangle = \sum_{j=0}^{N-1} x_j |j\rangle \rightarrow |y\rangle = \sum_{k=0}^{N-1} y_k |k\rangle = \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} x_j e^{\frac{2\pi i}{N} jk} |k\rangle \tag{9}$$

It is hard to see from the formula but, just like the DFT, the QFT is a unitary operator. The fact that is unitary implies important consequences, the most important is the fact that we can build a quantum circuit to compute the QFT that will act like a quantum logical gate (because it is unitary) and can be used in many quantum computing algorithms. To show that it is indeed a unitary operator, we can build the corresponding quantum circuit.

3.3 The QFT is a unitary operator

In order to prove that the QFT is a unitary operator we have to build the corresponding quantum circuit. To this end, in the following, we are going to consider $N = 2^n$, where n is some positive integer. Given this N, the two state vectors $|x\rangle$ and $|y\rangle$ are now defined in a new computational basis that is $|0\rangle, ..., |2^n - 1\rangle$. Moreover, it is useful to represent the generic state vector $|j\rangle$ using the binary representation $j = j_1 ... j_n = j_1 2^{n-1} + j_2 2^{n-2} + ... + j_n 2^0$. Finally, it is convenient to adopt the notation $0.j_l j_{l+1} ... j_m$ to represent the binary fraction $\frac{j_l}{2} + \frac{j_{l+1}}{4} + ... + \frac{j_m}{2^{m-l+1}}$. Then we obtain the following product representation:

$$\begin{aligned} |j\rangle &= \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi i}{2^n} kj} |k\rangle = \\ &= \frac{(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1} j_n}|1\rangle) ... (|0\rangle + e^{2\pi i 0.j_1 j_2 ... j_{n-1} j_n}|1\rangle)}{2^{n/2}} \end{aligned} \tag{10}$$

This product representation is useful to build an efficient circuit for the Quantum Fourier Transform. The figure below represents the circuit but not shown are swap gates at the end of the circuit which reverse the order of the qubits, or normalization factors of $1/\sqrt{2}$ in the output.

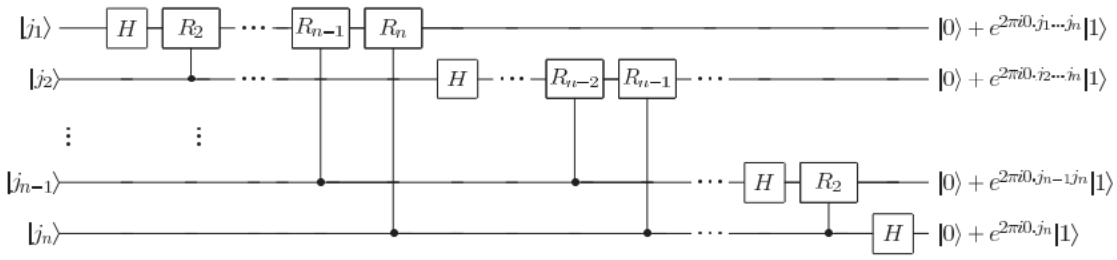

Figure 1: Quantum Fourier Transform Circuit

In the figure above two principal gates are used: H, denoting the **Hadamard** gate and R_k denoting the rotational gate, such that:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$$

Finally, the corresponding unitary matrix for the QFT is:

$$U_{QFT} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \omega_N^{-yx} |y\rangle\langle x| \tag{11}$$

How many gates? We have n inputs, where as usual $n = \log_2 N$. To each input, in the first stage, is applied the Hadamard gate for a total of n H-gate. At this point, for each input $1 \leq k \leq n$ a number of rotational gates are applied, precisely, $n - k$ gates. This means that $|j_1\rangle$ will have $n - 1$ rotational gates, $|j_2\rangle$ will have $n - 2$ and so on, up to a total of $(n-1) + (n-1) + ... + 1$. Finally, the total number of gates for the QFT circuit is $n + (n-1) + ... = n(n+1)/2$, meaning $O(n^2)$ gates.

4 Phase Estimation

The second ingredient for the Shor's algorithm is a more general procedure called *Phase Estimation*. We will see that in general the circuit for the Shor's algorithm kindly is the same of phase estimation, in particular the same of a specific instance of the PE problem called *Order Finding*. The key ingredient for PE is the Quantum Fourier Transform. Suppose a unitary operator U has an eigenvector $|u\rangle$ with eigenvalue $e^{2\pi i\varphi}$, where φ is unknown. The goal of phase estimation algorithm is to estimate φ. Given this, it is

quite easy to see why QFT is one key component in this problem. To perform the estimation we assume having a *black box* capable of preparing a non-trivial eigenvector $|u\rangle$ as long as performing the controlled-U^{2^j} operation, for suitable non-negative integers j. The use of a black blox indicates that the phase estimation procedure is not a complete quantum algorithm in its own right. This means that we should look at the phase estimation as a kind of subroutine or a module, we will see better this aspect when discussing about *Order Finding*.

The Phase Estimation subroutine uses two quantum register and two stages: the first one with t qubits in the state $|0\rangle$ (the choice of t depends also on the accuracy that we would like to have t when estimating), while the second one is initialized with the state vector $|u\rangle$. The first stage, that we can see in Figure 2, applies a bunch of Hadamard gates to each input of the first register and then t controlled-U^{2^i} gates are applied to the second register.

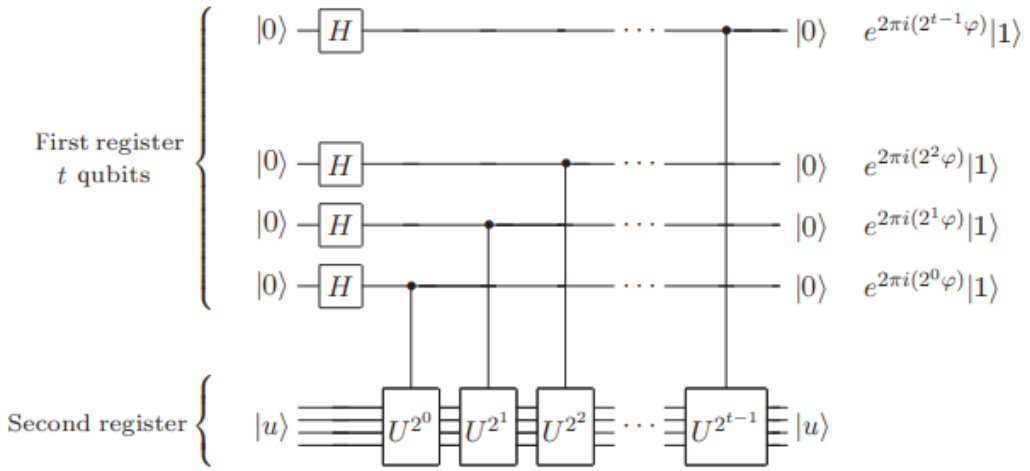

Figure 2: Phase Estimation circuit for the First stage

According to the circuit the final state of the first register is given by:

$$\frac{1}{2^{\frac{t}{2}}}\left(|0\rangle + e^{2\pi i 2^{t-1}\varphi}\right)\left(|0\rangle + e^{2\pi i 2^{t-2}\varphi}\right)...\left(|0\rangle + e^{2\pi i 2^{0}\varphi}\right) = \frac{1}{2^{\frac{t}{2}}}\sum_{k=0}^{2^t-1} e^{2\pi i \varphi k}|k\rangle \quad (12)$$

While the state of the second register remains the same $|u\rangle$. At this point, if we consider $N = 2^t$ and we represent φ with the binary fraction notation that we have adopted previously, i.e., $\varphi = 0.\varphi_1\varphi_2...\varphi_t$ we obtain the following state:

$$\frac{1}{2^{\frac{t}{2}}}\Big(|0\rangle + e^{2\pi i 0.\varphi_t}\Big)\Big(|0\rangle + e^{2\pi i 0.\varphi_{t-1}\varphi_t}\Big)...\Big(|0\rangle + e^{2\pi i 0.\varphi_1\varphi_2...\varphi_t}\Big) \tag{13}$$

That is just the product representation that we have used in the Quantum Fourier Transform and indeed it is the QFT of the state $|\varphi\rangle = |\varphi_1...\varphi_t\rangle$. Finally, the second stage of Phase Estimation is, obviously, applying the inverse of the QFT to apply the following transformation:

$$\frac{1}{2^{\frac{t}{2}}}\sum_{j=0}^{2^t-1} e^{2\pi i\varphi j}|j\rangle|u\rangle \rightarrow |\tilde{\varphi}\rangle|u\rangle \tag{14}$$

where $|\tilde{\varphi}\rangle$ is the best estimator for the phase φ when measured. Finally, the overall circuit for Phase Estimation is the one in the figure below.

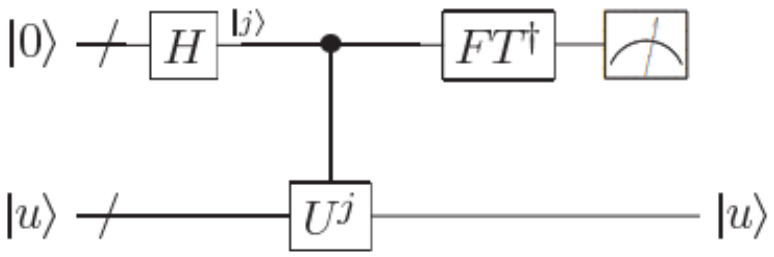

Figure 3: Phase Estimation circuit

5 Order Finding and Shor

Order Finding is the last building block before arriving to Shor's algorithm. Indeed, the quantum part of the factorization given by Shor is an order finding problem: given two positive integers x, N where $x < N$ we define the *order* (period) of x modulo N as the least positive integer r such that $x^r \equiv 1 \mod N$. The order finding problem is the problem of determining such a least positive r given x and N. This is the most important part of the entire algorithm because this problem is proved to be hard on classical computers, with a time complexity of $O\Big((\log N)^k\Big)$ for any k. However, there is an efficient algorithm for quantum computing that can handle this problem based on phase estimation. In other words, we could consider the order finding algorithm as a particular case of phase estimation acting on the unitary operator U defined as

$$U|y\rangle \equiv |xy \mod N\rangle \tag{15}$$

where $y \in \{0,1\}^L$, given L the number of bits. Notice that the unitary operator acts trivially for any $N \leq y \leq 2^L - 1$, because we use the convention that $xy(\mod N)$ is just y. On the other hand, the operator does not act trivially when given to $0 \leq y \leq N-1$. Once we have the unitary operator U, recall that for phase estimation we also require to know the eigenstates of the operator. With simple calculations we can prove that a general state written in the following form:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi isk}{r}\right] |x^k \mod N\rangle \tag{16}$$

are the eigenstates of U for integer $0 \leq s \leq r-1$, where $\exp\left[\frac{-2\pi isk}{r}\right]$ are the corresponding eigenvalues. Then applying the Phase Estimation procedure we can extract the best estimator for the phase $\varphi = s/r$, and then with a little bit more calculation finally obtain the order r.

There are two important requirements for us to be able to apply the phase estimation procedure: (1) we must be able to efficiently prepare an eigenstate $|u_s\rangle$ with a non-trivial eigenvalue; (2) we must be able to efficiently implement a controlled-U^{2^j} operator for any integer j. The first one is easily satisfied by observing the following fact:

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle = |1\rangle \tag{17}$$

The second requirements is also satisfied by using a procedure known as *Modular Exponentiation*. Finally, by using Phase Estimation we end up with the best estimator $|\tilde{\varphi}\rangle = |\tilde{s/r}\rangle$. However, we would like to find r and we do not know neither s nor, of course, r. *How is it done*? When measured $|\tilde{\varphi}\rangle$ gives us $\varphi \approx s/r$, and a-priori we know that φ is a rational number. The problem can be reduced to find the nearest fraction to φ from which we might obtain the order r. In order to do this, the idea is to apply the so-called *Continued Fraction Expansion*.

At the end Shor's algorithm is the one shown in Algorithm 2 where the quantum part is applied when is requested to find the order of x modulo N that is done by using order finding. Finally, the circuit for order finding is the following:

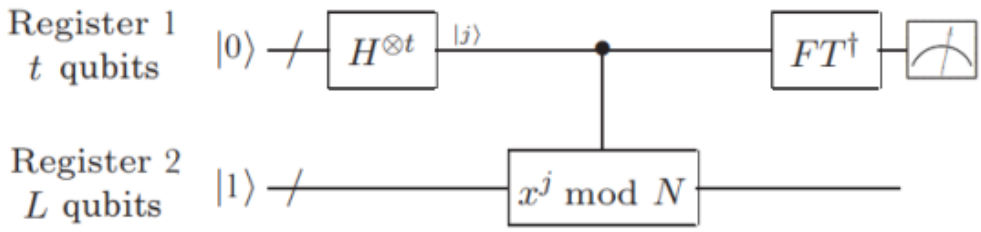

Figure 4: Order Finding Circuit

As we easily see, the circuit is the one from phase estimation but with modular exponentiation. We start from the state:

$$|0...0\rangle|0...01\rangle = |0\rangle^{\otimes_t} r^{t-1}\sum_{s=0}^{r-1}\sum_{k=0}^{r-1}\exp\left[-\frac{2\pi isk}{r}\right]|x^k \mod N\rangle \tag{18}$$

(1) Then after applying Hadamard we obtain the state:

$$2^{-t/2}\sum_j |j\rangle r^{-1}\sum_{s=0}^{r-1}\sum_{k=0}^{r-1}\exp\left[-\frac{2\pi isk}{r}\right]|x^k \mod N\rangle \tag{19}$$

(2) After the modular exponentiation we end up with the state:

$$2^{-t/2} \sum_j |j\rangle r^{-1} \sum_{s=0}^{r-1} \sum_{k=0}^{r-1} \exp\left[-\frac{2\pi isk}{r} - \frac{2\pi isj}{r} \right] |x^k \mod N\rangle \tag{20}$$

(3) After measuring the second register, the state is:

$$r^{-1} \sum_{s=0}^{r-1} \left[2^{-t/2} \sum_j \exp\left(-\frac{2\pi isj}{r} \right) |j\rangle \right] \exp\left(-\frac{2\pi isk}{r} \right) |x^k \mod N\rangle \tag{21}$$

(4) After applying the Inverse Quantum Fourier Transform the state is:

$$r^{-1} \sum_{s=0}^{r-1} |\tilde{s/r}\rangle \exp\left(-\frac{2\pi isk}{r} \right) |x^k \mod N\rangle \tag{22}$$

(5) Applying CFE we can retrieve the candidate r.

Finally, we have to talk a bit about efficiency. The Shor's algorithm is composed by many factors and many sub-algorithms which can bring more complexity to the overall time-complexity. First of all we have the *Euclidean Algorithm* to compute the greatest common divisor that on integers $x > y$ requires at most $O(\log N)$ steps. The Continued Fraction Expansion is also related wit the Euclidean algorithm and again it requires $O(\log N)$ steps. The computation of the Quantum Fourier Transform of m qubits, luckily, requires $O(m)$ steps. Finally, the controlled-U_a gates can be implemented efficiently using modular exponentiation and requires $O(n^2 \log n \log\log n)$ time and $O(n \log n \log\log n)$ space. Finally, the overall time of a single iteration is completely dominated by the time complexity given by the controlled-U_a transformations.

Now, let's get more into the details of Continued Fraction Expansion first and then Modular Exponentiation, where I will discuss of a more efficient Shor circuit using $2n + 3$ qubits, where as usual $n = \log_2 N$.

5.1 Continued Fraction Expansion

The basic idea behind CFE is to describe a real number in terms of integers alone, using a continuous fraction expressed as:

$$[a_0, a_1, ..., a_M] \equiv a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\ldots + \frac{1}{a_M}}}} \tag{23}$$

where $a_0, ..., a_M$ are positive integer (for application to quantum computing it is convenient to set $a_0 = 0$). We define the *mth convergent* ($0 \leq m \leq M$) to this continued fraction to be $[a_0, ..., a_m]$. The *Continued Fraction Algorithm* is a method for determining the continued fraction expansion of an arbitrary real number. Consider now the following theorem.

Theorem: Suppose s/r is a rational number such that

$$\left|\frac{s}{r} - \varphi\right| \leq \frac{1}{2r^2} \tag{24}$$

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

Since φ is an approximation of s/r accurate to $2L + 1$ bits, it follows that $|s/r - \varphi| \leq 2^{-2L-1} \leq 1/2r^2$ since $r \leq N \leq 2^L$. Summarizing, we obtain that the continued fraction algorithm applied to φ will return two positive integers with no common factor s' and r' such that $s'/r' = s/r$ and thus we can use r' as our best candidate and check it by computing the modular exponentiation and control if it is congruent 1 modulo N. In the following the pseudo-code for the algorithm of continued fraction espansion.

Algorithm 3 Continued Fraction Expansion

Require: $\varphi \approx s/r$ (in binary)
Require: N The input integer to the factoring algorithm
  $\varphi \leftarrow \text{decimal}(\varphi)$
  if φ is negative there are no continued fraction **then**
    EXIT
  end if
  $r \leftarrow 2^{\log_2 \varphi}$ (2 to the power of number of bits of φ)
  $x \leftarrow \varphi/r$ (Initialize the value for CFE)
  $c \leftarrow 0$ (Initialize a counter)
  $b \leftarrow [\text{floor}(x)]$
  $t \leftarrow [x - b[0]]$
  while $c < N$ **do**
    if $c > 0$ **then**
      $b \leftarrow b \cup [\text{floor}(1/\text{t}[\text{c} - 1])]$
      $t \leftarrow t \cup [(1/t[c-1]) - b[c-1]]$
    end if
    $a \leftarrow 0$ (Initialize the continued fraction with 0)
    while $i \in [\text{size}(b) - 1, ..., 0]$ **do**
      $a \leftarrow 1/(b[i+1] + f)$
    end while
    $a \leftarrow a + b[0]$
    $d \leftarrow \text{get\_denominator}(a)$
    $c \leftarrow c + 1$
    if d is odd or $d \geq 1000$ **then**
      CONTINUE
    end if
    if $t[c-1] = 0$ **then**
      RETURN (we have already found the number)
    end if
    YIELD d (generate the candidate)
  end while

At the end, each candidate produced by the continued fraction expansion procedure will be processed by Algorithm 2 and all the final factors will be returned.

5.2 Modular Exponentiation

In general, we can see the application of the controlled-U^{2^j} gates in phase estimation be equivalent to:

$$\begin{aligned} |z\rangle|y\rangle \rightarrow |z\rangle U^{z_t 2^{t-1}} ... U^{z_1 2^0}|y\rangle = \\ = |z\rangle|x^{z_t 2^{t-1}} \times ... \times x^{z_1 2^0} y(\mod N)\rangle = \\ = |z\rangle|x^z y(\mod N)\rangle \end{aligned} \tag{25}$$

where $x^z(\mod N)$ is the modular exponentation and z is the content of the first register. To easily compute the modular exponentiation we use the so-called *reversible computation*. The basic idea is to reversibly compute the modular exponentiation and save the result in a third register, and then reversibly multiply the content of the second register by $x^z(\mod N)$ to erase the third register. At the end, in the most basic form, modular exponentiation has two stages: in the first stage we compute $x^{2^j}(\mod N)$ for all j up to $t-1$, and in the second stage we are going to compute the product of all the components, meaning:

$$x^z(\mod N) = \prod_{j=0}^{t-1} x^{z_{t-j} 2^{t-j+1}}(\mod N) \tag{26}$$

Selecting $t = 2L + 1 + \lceil \log(2 + \frac{1}{2\varepsilon}) \rceil = O(L)$, a total of $t - 1 = O(L)$ squaring operations is performed at a cost of $O(L^2)$ each, up to a total cost of $O(L^3)$ for the first stage and second stage. More efficient algorithms have been developed for modular exponentiation, and in the following we are going to see one of them, maybe the most efficient one according to literature.

5.2.1 The $2n + 3$ qubits Shor's circuit

I decided to talk about this implementation in this section because all the improvements comes from how efficiently modular exponentiation is implemented. This comes directly from the literature, and in particular from the paper Circuit For Shor's Algorithm using 2n + 3 qubits (Stephane Beauregard). First we need to consider the circuit for the addition. In this case let us consider that this circuit takes as input two registers of n qubits, i.e., $|a_0 ... a_{n-1}\rangle$ and $|\phi_0(b) ... \phi_{n-1}(b)\rangle$ where $|\phi(b)\rangle$ is the QFT of $|b\rangle$. From Addition on Quantum Computer (Draper) we can build a circuit taking as input

this two register, compute the classical phase shift, the apply the Inverse Quantum Fourier Transform on the second register and takes back the final state $|(a+b) \mod 2^n\rangle$. This is the circuit corresponding only to the first part of the previous described circuit, i.e., essentially only the application of the controlled-phase shifts gates:

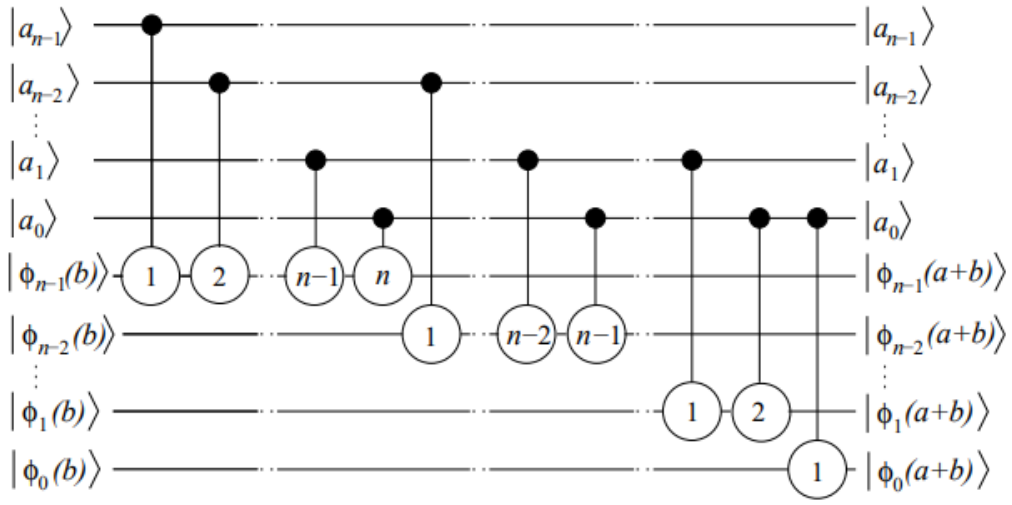

Figure 5: The Quantum Addition described by Draper

Where the controlled-i gates are the classical controlled-phase shift gates represented by the unitary matrix:

$$cP_k = \begin{bmatrix} I & 0 \\ 0 & P_k \end{bmatrix} \quad \text{where} \quad P_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{\pi i}{2^{k-1}}} \end{bmatrix} \tag{27}$$

Let us call this circuit the ΦADD circuit, and we obtain the circuit in Figure 6.

We may notice two important things about this circuit: it has only one input compared with the previous one, and the general A_i gates are classically computed combinations of phase shifts. It is possible to prove that these two circuits are actually the same. First of all, let's see what happens to each input $|\phi_i(b)\rangle$ in the original circuit when the controlled-P gate is applied. As usual, from theory, we know that since it is a controlled-P gate is acts non-trivially on the input only if the corresponding control bit is in state $|1\rangle$. In

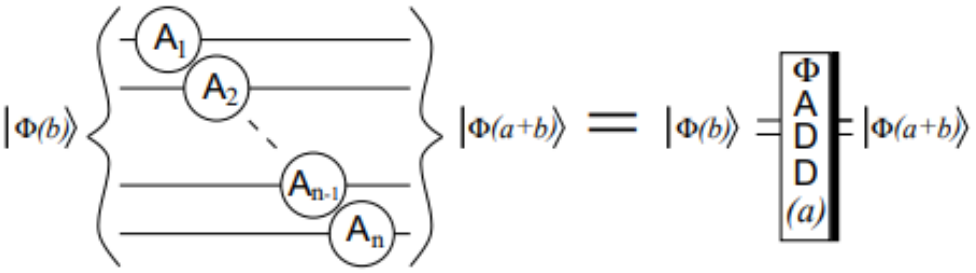

Figure 6: The Single input ΦADD circuit

Figure 5 we have that multiple controlled-P gates acts serially on the same input $|\phi_i(b)\rangle$, meaning that the final state for that input is computed as:

$$|\phi_i(b+a)\rangle = cP_i \times ... \times cP_0 |\phi_i(b)\rangle \tag{28}$$

For example, considering $i = n-1$, we obtain that:

$$|\phi_{n-1}(a+b)\rangle = \left(\prod_{k=n-1}^{0} cP_k \right) |\phi_{n-1}(b)\rangle \tag{29}$$

Recalling the fact that cP_i are controlled-P gates, we can rewrite the previous expression as:

$$|\phi_{n-1}(a+b)\rangle = \left(\prod_{k\in[n-1,0] \text{ s.t. } |a_k\rangle=|1\rangle} cP_k \right) |\phi_{n-1}(b)\rangle = \begin{bmatrix} I & 0 \\ 0 & e^{i\pi\Phi} \end{bmatrix} \tag{30}$$

where

$$\Phi = \sum_{k\in[n-1,0] \text{ s.t. } |a_k\rangle=|1\rangle} \frac{1}{2^{k-1}} \tag{31}$$

Finally, if we can pre-compute the angles from the input value $|a\rangle = |a_0 ... a_{n-1}\rangle$, and we can, we obtain a new operator Γ_k such that:

$$\Gamma_k |\phi_k(b)\rangle = |\phi_k(a+b)\rangle \tag{32}$$

This new operator Γ_k are just the corresponding A_k in Figure 6. Notice that, in order to prevent overflow during the addition, we need one more bit,

hence $n+1$ qubits for the quantum register instead of n, so that $\phi(b)$ is effectively the QFT of an $(n+1)$-qubit register containing a n-bit number. This means that the most significant qubit before applying the QFT preceding the addition is always $|0\rangle$. The thick bar on the right in Figure 6 is used to distinguish the ΦADD with its unitary inverse.

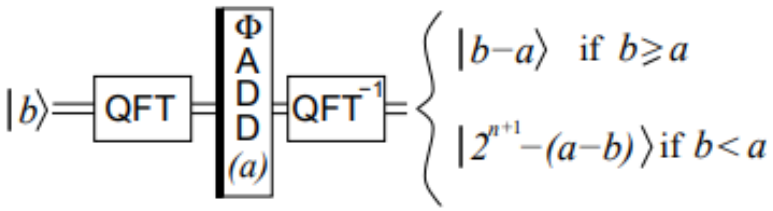

Figure 7: The application of the $\Phi ADD^{-1}(a)$ on $|\phi(b)\rangle$

Once we have defined the $\Phi ADD(a)$ gate we can build the corresponding *Modular Adder Gate*, let's call it $\Phi ADD(a)MOD(N)$, such that:

$$\Phi ADD(a)MOD(N)|\phi(b)\rangle = |\phi((a+b) \mod N)\rangle \tag{33}$$

It is also called *doubly controlled-Modular Adder* gate because it implements doubly controlled-ΦADD gates. This is almost the last building block for the modular exponentiation circuit and it is based on three principal circuit (complex gates): $\Phi ADD(a)$, $\Phi ADD(N)$, QFT and all the respective inverse. Figure 8 shows the circuit of the modular adder gate.

How does it works? It is a two stage circuit, in the first stage we compute the actual modular addition, while in the second stage we restore the value of the ancilla qubit $|0\rangle$ that has been modified during the overall computation. First of all, the circuits takes as input a total of $n+3$ qubits, where the first two qubits $|c_1c_2\rangle$ are the two controller qubits, the following n comes from $|\phi(b)\rangle$ and finally, the last one is due to the ancilla qubit $|0\rangle$ used to prevent overflow. Let's see the two stage more in detail.

First Stage. The first part of the first stage is to apply the doubly controlled-$\Phi ADD(a)$ gate on $|\phi(b)\rangle$ which returns $|\phi(a+b)\rangle$ if and only if $c_1 = c_2 = 1$, otherwise the output of the entire circuit will be trivially $|\phi(b)\rangle$. Then we apply the reverse $\Phi ADD^{-1}(N)$ gate on $|\phi(a+b)\rangle$ to obtain $|\phi(a+b-N)\rangle$. It may happens that $a+b < N$, then in this case we do not need to subtract

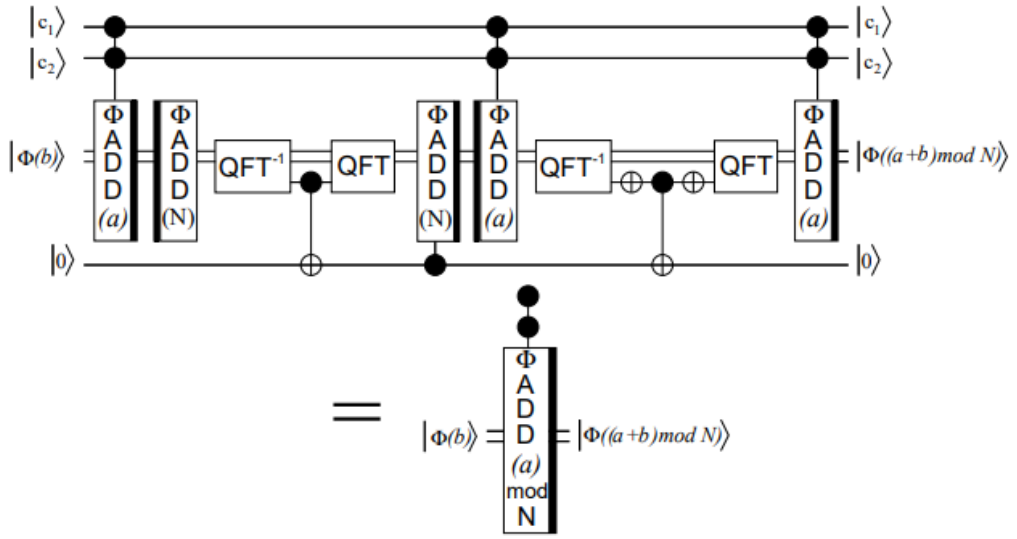

Figure 8: The $\Phi ADD(a)MOD(N)$ gate circuit

N, because $a+b(\mod N) = a+b$. We can control this by looking at the most significant bit of $a+b-N$. To access the MSB we first need to restore the original value from the QFT, hence applying the IQFT on $|\phi(a+b-N)\rangle$ we obtain $|a+b-N\rangle$. We can use the previously obtain MSB as a controller qubit for the cNOT gate applid to the ancillary qubit $|0\rangle$ that will become $|1\rangle$ if the MSB is 1 (as usual by definition of controlled gates). At this point we reapply the QFT, hence reobtaining $|\phi(a+b-N)\rangle$ and then applying the controlled-$\Phi ADD(N)$ we obtain $|\phi(a+b)\rangle$ if the ancillary qubit is $|1\rangle$, meaning that $a+b<N$, otherwise we remain in the state $|\phi(a+b-N)\rangle$, which is actually the output of the first stage that corresponds to $|\phi((a+b) \mod N)\rangle$.

Second Stage. In this stage we want to restore the ancillary qubit to the original value. First we apply $\Phi ADD^{-1}(a)$ on $|\phi((a+b) \mod N)\rangle$ obtaining $|\phi((a+b) \mod N-a)\rangle$. Then apply the IQFT to obtain $|(a+b) \mod N-a\rangle$ and check the MSB. At this point the MSB will be $|0\rangle$ if and only if $(a+b) \mod N \geq a$ that happens only if $a+b<N$. Then, we apply the X (Not)-gate on the MSB that will become $|1\rangle$ if $a+b<N$, then apply the cNOT on the ancillary qubit, that will become $|0\rangle$ only if it was previously $|1\rangle$ and $a+b<N$. Once we have restored the ancillary qubit, we restore also the MSB, then apply the QFT obtaining $|\phi((a+b) \mod N-a)\rangle$,

finally applying $\Phi ADD(a)$ we remove the factor a obtaining the final result $|\phi((a+b) \mod N)\rangle$.

Then we have to create the *Controlled-Multiplier* gate by using the doubly controlled-Modular Adder, which is trivially applying a series of doubly controlled-$\Phi ADD(k)MOD(N)$ with $k = 2^j a$, finally obtaining $|(b+ax) \mod N\rangle$. This circuit, that we will call $CMULT(a)MOD(N)$ takes the input $|c\rangle|x\rangle|b\rangle$ and returns $|c\rangle|x\rangle|(b+ax) \mod N\rangle$ if $|c\rangle = |1\rangle$ otherwise the state will remain the same. It is very simple to be implemented given the doubly controlled-modular adder, since we can rely on this simple identity:

$$(ax) \mod N = (...((2^0 a x_0) \mod N + 2^1 a x_1) \mod N + ... + 2^{n-1} a x_{n-1}) \mod N \tag{34}$$

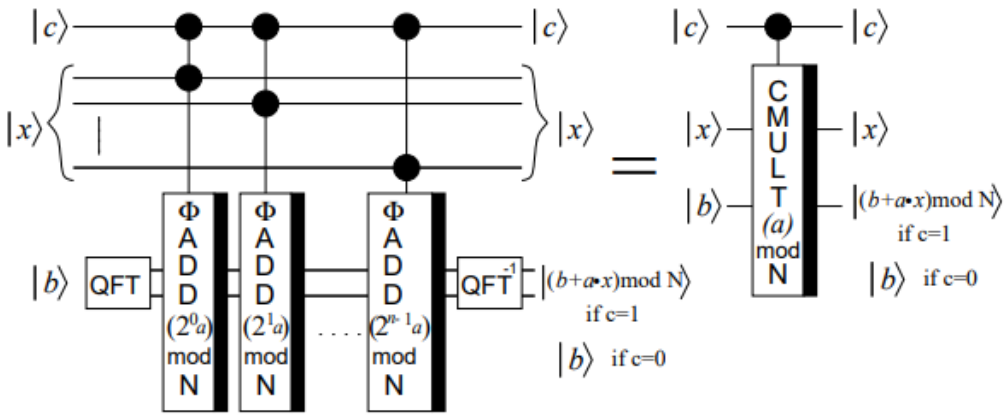

Figure 9: The $CMULT(a)MOD(N)$ gate circuit

Using the $CMULT(a)MOD(N)$ we obtain in the third register $|(b+ax) \mod N\rangle$. However, we would need instead is a controlled gate that takes $|x\rangle$ to $|(ax) \mod N\rangle$. This can be done using a trick that is called **reversible computation**: we first apply the $CMULT(a)MOD(N)$ on $|c\rangle|x\rangle|0\rangle$, then a controlled-SWAP that swaps the two register whenever $|c\rangle = |1\rangle$, and then we finish with the inverse $CMULT(a^{-1})MOD(N)$. Notice that we only need to control n qubits, not $n+1$ because the MSB introduced to prevent overflow will always be 0. Applying all these operations, we obtain the following sequence of states:

$$
\begin{aligned}
|x\rangle|0\rangle &\rightarrow |x\rangle|(ax) \mod N\rangle \\
&\rightarrow |(ax) \mod N\rangle|(x - a^{-1}ax) \mod N\rangle \\
&\rightarrow |(ax) \mod N\rangle|0\rangle
\end{aligned}
\tag{35}
$$

This new circuit is the controlled-U_a gate that we were looking for.

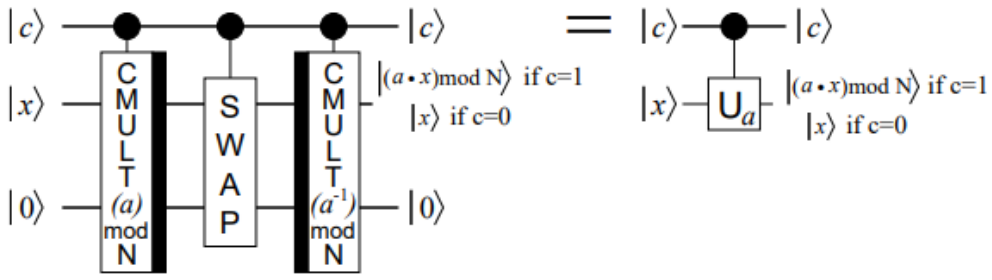

Figure 10: The controlled-U_a gate

Since the bottom register returns to $|0\rangle$ we can consider this as being part of the $c - U_a$ gate, that now effectively takes $|x\rangle$ to $|(ax) \mod N\rangle$. Now, we can use this gates as part of the order finding circuit. Notice that we do not need to apply n times the controlled-U_a gate, instead we can rely on:

$$
(a^n x) \mod N = (a...(a(ax) \mod N) \mod N...) \mod N \tag{36}
$$

Hence, applying one cU_{a^n} is equal to apply n times the cU_a gate.

Complexity Analysis. The $\Phi ADD(a)$ circuit, where a is a classical value, required $n + 1$ qubits because we need an extra qubit to prevent overflows. The doubly controlled $\Phi ADD(a)MOD(N)$ requires $n+4$ qubits. The $CMULT(a)MOD(N)$ circuit is only n doubly controlled $\Phi ADD(a)MOD(N)$ circuits. It thus takes $2n + 3$ qubits. Two of these circuits along with the controlled-SWAP are required for the $C - U_a$ circuit. For the whole order finding cricuit, that is, the whole quantum part of Shor's algorithm, we need $2n$ of these $C - U_a$ circuits. The final quantum resources are thus $2n + 3$ qubits and a depth of $O(n^3)$.