ON THE FACTORING OF NUMBERS USING A QUANTUM COMPUTER

XIAO SUYANG

Abstract. We give an overview of an algorithm called Shor's algorithm, which factors numbers using a quantum computer. We explain the logic behind each step of the algorithm, and most importantly, it's order-finding subroutine that finds the integer r such that $g^r \equiv 1 \mod N$ with a time complexity of $\mathcal{O}(n^2)$, and highlight it's computational advantage over classical algorithms.

1. A brief introduction to computation

Factoring integers is a notoriously difficult problem, and an important one as well. The RSA cryptosystem, which is used to encrypt information online(such as encrypting online credit card transfers), relies on the fact that factoring large numbers(on the scale of 2^{1024}) is practically impossible[RSA78].

To better understand the difficulty of factoring in the perspective of computation, we begin by providing an approachable definition for computation and differentiate computation between classical computers and quantum computers[Sho97].

Definition 1.1. Classical computation is the use of **physical** devices to perform sequences of arithmetic or non-arithmetic calculation.

Definition 1.2. Quantum computation is the use of **quantum mechanical** devices to perform sequences of arithmetic or non-arithmetic calculation.

The only distinction of quantum computation is that the calculations are carried out using devices that exploit physical interactions described by quantum mechanics, as opposed to classical mechanics.

For a concrete example, consider the transistor, the physical system encoding the information of a bit. The current in a transistor $I(t)$ is described by the following equation:

$$I(t) = I_0 e^{-\frac{QV_g}{Kt}}(e^{\frac{QV_s}{Kt}} - e^{\frac{QV_g}{Kt}}) \tag{1.1}$$

Where V_g and V_s are the voltage at both ends of the transistor, K and q are constants. For a quantum counterpart of a bit, which we call a qubit, it encodes information using quantum mechanics. For example, the following equation:

$$\psi_n(x,t) = e^{-iE_n t}\sqrt{\frac{2}{L}}\sin(\frac{n\pi}{L}x) \tag{1.2}$$

describes a quantum mechanical system of a particle contained in a box, where x is the position amplitude and E_n is the energy level of that particle.

Technically, We can use quantum mechanics to describe the transistor. It is just that the scale of the currents of a transistor is so large that quantum effects are almost negligible, making classical mechanics more appropriate to describe the system.

Definition 1.3 (Polynomial time)**.** A problem can be solved in polynomial time when the number of steps needed to arrive at the answer to the problem is bounded above by a function that is polynomial w.r.t. the input size, given that each step takes a fixed amount of time.

Remark 1.4. We say that the time complexity of this problem is $\mathcal{O}(n^k)$, for some integer $k \geq 0$, and n the length of the binary representation of the input.

This is an important concept, because any problem that takes beyond polynomial time to solve, would take so long that it becomes impractical to implement in practice, even when the input size is not that large[Cor+09].

Next, we can use this concept to define classes of problems that are practical for us to solve on a computer, both classical and quantum:

Definition 1.5 (Class **P**)**.** A problem is efficiently solvable, or equivalently, belongs to the class **P**, when it takes a classical computer a polynomial time to compute the correct answer.

Definition 1.6 (Class **BQP**)**.** A problem belongs to the class **BQP** when it takes a quantum computer a polynomial time to compute the correct answer **with high probability**[1].

It is known that $\mathbf{P} \subseteq \mathbf{BQP}$, and widely conjectured that $\mathbf{P} \subset \mathbf{BQP}$[AB09]. In simpler terms, we believe that there are problems that can be solved in polynomial time by a quantum computer with high probability, but not in polynomial time by a classical computer.

The reason we conjecture $\mathbf{P} \subset \mathbf{BQP}$ is because we have already found faster quantum algorithms, such as Grover's algorithm for unstructured search with time complexity of $\mathcal{O}(\sqrt{n})$, that is faster than the classical limit of $\mathcal{O}(n)$[Gro96]. Although $\mathcal{O}(n)$ is already in polynomial time, it is nonetheless a definite advantage for a quantum computer, which incentives us to consider using a quantum computer on the factoring problem.

On a classical computer, factoring integers is a very slow process. The best algorithm we have, known as the general number field sieve, takes an exponential time to solve[2]. Hence, we have yet to find an algorithm that factors numbers in polynomial time. The rest of the report will cover one such algorithm that does factor number in polynomial time, on a quantum computer.

2. The factoring algorithm

Consider an arbitrary number N, for simplicity's sake, let N be an odd semiprime, where $N = p \cdot q$, and both p and q are odd primes. Consider the set $S = \{1, 2, \cdots, N-1\}$, and pick a number g from this set as a guess.

Straight away, we can check $\gcd(g, N)$ efficiently using the Euclid's algorithm. The details of Euclid's algorithm will be omitted, but we give without proof that Euclid's algorithm has a time complexity of $\mathcal{O}(n^3)$, where n is the length of the binary representation of N[NC10].

If $\gcd(g, N) \neq 1$, we are done as $\gcd(g, N)$ is a factor of N. In the case where $\gcd(g, N) = 1$, we know that there exists an integer r such that $g^r \equiv 1 \mathrm{mod} N$, or alternatively, $g^r = mN + 1$ for some integer $m \geq 1$. The existence of r comes from the fact that, the set S is a finite set of all values of all possible values modulo N. By the pigeonhole principle, you can find $i \neq j$ where

[1] In computer science, this means in k runs of the algorithm, the probability of producing a correct answer in any one of those runs is $1 - \frac{1}{2^{ck}}$ for some $c \geq 0$

[2] The exact time complexity is $e^{kf(n)}$, where $f(n) = \log n^{\frac{1}{3}} \log\log n^{\frac{2}{3}}$, $k = (\frac{8}{3}^{\frac{2}{3}} + o(1))$

$g^i \equiv g^j \mathrm{mod} N$, and WLOG let $i > j$, then $g^r = g^{i-j} \equiv 1 \mathrm{mod} N$.

Then, we have $(g^{\frac{r}{2}} + 1)(g^{\frac{r}{2}} - 1) = mN$. When r is even, meaning both terms are integers, and when $g^{\frac{r}{2}} + 1 \not\equiv 0 \mathrm{mod} N$, meaning no term is a complete factor of m, we must have one of $(g^{\frac{r}{2}} + 1)$ or $(g^{\frac{r}{2}} - 1)$ sharing a factor with N. Again, we can use the Euclid's algorithm to compute both $\gcd(g^{\frac{r}{2}} + 1, N)$ and $\gcd(g^{\frac{r}{2}} - 1, N)$ to find either p or q.

Theorem 2.1 (Probability of generating a factor using g)**.** *Suppose $N = p \cdot q$ is an odd semiprime. Let g be uniformly chosen at random from $S = \{1, 2, \cdots, N-1\}$. Let r be the order of x modulo N. Then,*

$$p(r \text{ is even and } x^{\frac{r}{2}} \not\equiv -1 mod N) \geq 1 - \frac{1}{2^2}$$

This theorem is meant to convince the reader that, picking any arbitrary guess g from the set S will lead to a very good guess, **with high probability** of success. To summarize, we have the following:

Shor's Algorithm[Sho97]:

- Randomly pick $g \in \{1, 2, \cdots N-1\}$, check $\gcd(g, N)$ using Euclid's algorithm
- If $\gcd(g, N) \neq 1$, return $\gcd(g, N)$
- Else if $\gcd(g, N) = 1$, find r such that $g^r \equiv 1 \mathrm{mod} N$
  - If r is even and $x^{\frac{r}{2}} \not\equiv -1 \mathrm{mod} N$, compute $\gcd(g^{\frac{r}{2}} + 1, N)$ and $\gcd(g^{\frac{r}{2}} - 1, N)$ using Euclid's algorithm. Test to see which one a factor of N. return that factor.
  - Else, pick a different g, and repeat.

3. The quantum order-finding subroutine

One part that we did not mention in the previous section is on how to find the period r. Indeed, we have reduced the problem of factoring into finding the order r such that $g^r \equiv 1 \mod N$. This is a process that is extremely slow on a classical computer, where we have to repeatedly compute g^j modulo N until we see a repetition. A rough estimate for the time complexity is $\mathcal{O}(2^n)$, where n is the length of the binary representation of N[NC10].

On a quantum computer, however, we can find r much more efficiently. We call this part of the algorithm the quantum order-finding subroutine.

3.1. Mathematical foundation of quantum computation.

In order to be able to explain the sub-routine, we first establish the smallest building blocks of quantum circuits, which are qubits, and quantum gates.

Postulate 1 (State space)**.** *Associated with any isolated physical system is a complex vector space with an inner product(a Hilbert space). We call this the state space $\mathcal{H}$, where all possible physical states the system can be in are represented by vectors of unit norm in this state space.*

Remark 3.1. Conversely, every linear combination of existing state vectors with a unit norm represents another possible physical state. This is known as the superposition property.

This postulate, proposed as a result of much trial and error from experiments, sets up the space that a qubit "lives" in[NC10].

To give a physical example of a qubit, we return to equation (1.2) and consider $\psi_n(x) = \sqrt{\frac{2}{L}} \sin(\frac{n\pi}{L} x)$, the stationary state of the particle that is fixed in time. This function corresponds to the "state" of the particle at the energy level E_n. We consider the particle's lowest two energy states only, with $|\psi_1(x)\rangle$ and $|\psi_2(x)\rangle$ are the state vectors:

$$|0\rangle = |\psi_1(x)\rangle = \sqrt{\frac{2}{L}}\sin(\frac{1\cdot\pi}{L}x)$$
$$|1\rangle = |\psi_2(x)\rangle = \sqrt{\frac{2}{L}}\sin(\frac{2\cdot\pi}{L}x) \tag{3.1}$$

i.e., when the particle is at the energy level E_1, we name it $|0\rangle$, and when the particle is at the energy level E_2, we name it $|1\rangle$, to be consistent with classical bits. We call these the basis states of a qubit. By Remark 3.1, any linear combination of $|0\rangle$ and $|1\rangle$, such as $\alpha|0\rangle + \beta|1\rangle$ with $\alpha^2 + \beta^2 = 1$ represents another valid state of the particle, being both at energy level E_1 and E_2.

Remark 3.2. When we perform a **measurement**, the superposition collapses to either one of the basis states.

Remark 3.3. When we have multi-qubit systems, such as two qubits, we consider the combined space $\mathcal{H}_1 \otimes \mathcal{H}_2$. This is called the tensor product, and is used to represent the state spcae of multi-qubit systems.

Consider a smaller system, such as $|1\rangle \otimes |0\rangle \otimes |0\rangle \otimes |1\rangle$. It is equivalant to $|[1001]_2\rangle = |9\rangle$. This is useful because we don't need to manipulate a single particle to it's 9th energy level to encode the number 9 Instead, we can use 4 individual qubit systems and only use the first two energy levels. This is much more efficient in practice.

Postulate 2 (Time evolution)**.** *The evolution over time of a closed quantum system is described by a unitary transformation. That is, the state $|\psi(x,t_1)\rangle$ of the system at time t_1 is related to the state $|\psi(x,t_2)\rangle$ at time t_2 by*

$$|\psi(x,t_2)\rangle = U|\psi(x,t_1)\rangle$$

Where U is the matrix representation of a unitary transformation.

The benefit of the quantum computer comes directly from these postulates: Postulate 1 allows for parallel computation, where if we can somehow encode all powers of g modulo N into individual state vectors, we can manipulate the system physically, to form a new vector that is a linear combination of the individual ones. Performing operation on this new vector is equivalent to performing operation simultaneously on the composite states.

3.2. **Quantum circuit.** Next, we give an overview of the quantum circuit that carries out the order-finding subroutine, and we will first explain each component of the circult:

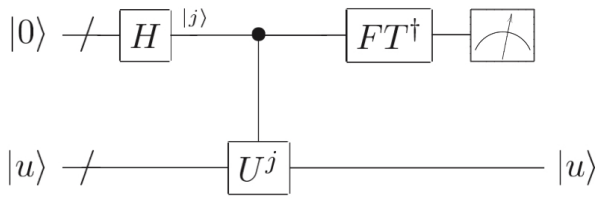

FIGURE 1. Overview of quantum circuit for period finding

Register 1 is denoted by $|0\rangle$, and contains m qubits, with a size constraint of $2N^2 \geq 2^m \geq N^2$. Register 1 is used to encode all the exponents we can raise g to, where 2^m is the number of possible combinations. Hence, the size constraint for 2^m is to guarantee that we raise g to enough powers

to make the periodicity repeat many times[NC10].

Register 2 is denoted by $|u\rangle$ and contains n qubits, where the size of n is around $\log_2(g)$ to provide a binary representation of the number g.

The H represents the Hadamard transform, and puts all qubits in register 1 in a superposition; the U represents Controlled-U^{2^i} transformation, and they are "black boxes" that uses the superposition in register 1 to put g into a superposition of all $g^j \mod N$; lastly, FT represents the Quantum Fourier Transform that extracts period of $g^j \mod N$.

3.3. **Encoding Information using qubits.** The first step of the algorithm is to encode the information of modular exponentiation into qubits. The following figure gives more details to this section of the circuit:

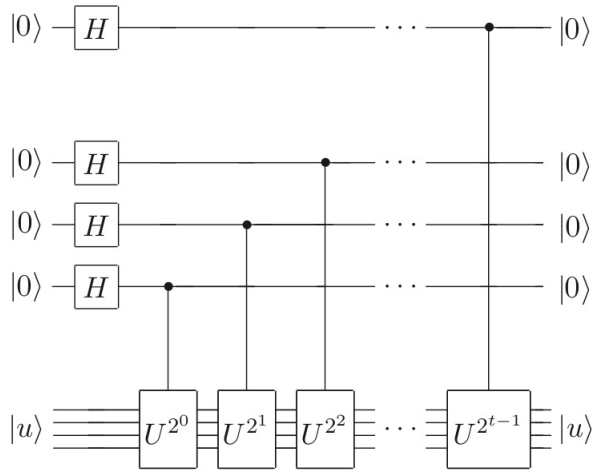

FIGURE 2. Encoding information into the qubits

Consider a single qubit system, with the basis $B_Q\{|0\rangle, |1\rangle\}$. In vector notation, we can write $|0\rangle$ as $(1, 0)$. The following transformation, called the Hadamard transformation, acts on $|0\rangle$ in the following way:

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

It is easy to check that H is unitary. Essentially, we put the state $|0\rangle$ into a superposition of $|0\rangle$ and $|1\rangle$, with equal probability. By applying the Hadamard transform on every qubit on the first register, we get the following:

$$|\psi_1\rangle = H^m|\psi_0\rangle = \bigotimes_{j=1}^{m} H|0\rangle \otimes \bigotimes_{i=1}^{n}|0\rangle = \bigotimes_{j=1}^{m}\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \otimes \bigotimes_{i=1}^{n}|0\rangle \tag{3.2}$$

Now consider the expression for all the superpositions in register 1 only. We expand this portion further, by distributing the tensor product on individual terms:

$$\begin{aligned}\bigotimes_{j=1}^{m}\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) = \frac{1}{\sqrt{2^m}}\big(&|0\rangle\otimes\cdots\otimes|0\rangle\otimes|0\rangle\\ &+|0\rangle\otimes\cdots\otimes|0\rangle\otimes|1\rangle\\ &+|0\rangle\otimes\cdots\otimes|1\rangle\otimes|0\rangle\\ &+|0\rangle\otimes\cdots\otimes|1\rangle\otimes|1\rangle\\ &\cdots\\ &+|1\rangle\otimes\cdots\otimes|1\rangle\otimes|1\rangle\big)\end{aligned} \tag{3.3}$$

Notice that each term in the sum contains the same information as $|[j_1\cdots j_m]_2\rangle$, where $[j_1\cdots j_m]_2$ is a binary number of length m, with each j_i equal to 0 or 1. Then, we can say (3.3) is equivalant to the following:

$$\begin{aligned}(3.3) &= \frac{1}{\sqrt{2^m}}(|[0\cdots00]_2\rangle+|[0\cdots01]_2\rangle+|[0\cdots10]_2\rangle+|[0\cdots11]_2\rangle+\cdots+|[1\cdots11]_2\rangle)\\ &= \frac{1}{\sqrt{2^m}}(|0\rangle+|1\rangle+|2\rangle+|3\rangle+\cdots+|2^m-1\rangle) = \frac{1}{\sqrt{2^m}}\sum_{j=0}^{2^m-1}|j\rangle\end{aligned}$$

Which is a superposition of all numbers from 0 to 2^m-1. Conceptually, consider the scenario where we measure all m qubits in register 1 simultaneously. Then, we will have a equal probability of getting any string of binary numbers of length m, since at every position, we have a $\frac{1}{2}$ probability of getting either $|0\rangle$ or $|1\rangle$.

The next step is to manipulate register 2 to encode the information of $g^j \bmod N$. The key step is to write $|j\rangle$ as $|[j_1\cdots j_m]_2\rangle$, it's binary representation. This is a form that we use very often.

In order to raise g to the j, we consider:

$$g^j = g^{[j_1\cdots j_m]_2} = g^{j_1 2^{m-1}+\cdots+j_m 2^0} = g^{j_1 2^{m-1}} g^{j_2 2^{m-2}}\cdots g^{j_m 2^0}$$

.

Notice by writing g in this form, each j_i corresponds to the j_ith qubit in register 1. Using a concrete example, consider if register 1 had 4 qubits, and after measurement, we get

$$|j_1\rangle\otimes|j_2\rangle\otimes|j_3\rangle\otimes|j_4\rangle = |1\rangle\otimes|0\rangle\otimes|0\rangle\otimes|1\rangle \cong |[1001]_2\rangle = |9\rangle \tag{3.4}$$

Correspondingly, we want register 2 to collapse to the state

$$|g^9 \bmod N\rangle = |g^{[1001]_2} \bmod N\rangle = |g^{1\cdot2^3+0\cdot2^2+0\cdot2^1+1\cdot2^0} \bmod N\rangle \tag{3.5}$$

We start with $|u\rangle = |[0\cdots01]_2\rangle = |1\rangle$. When passing through each controlled-U^{2^i} gate, we apply:

$$\begin{cases} U|u\rangle = |u\cdot g^{2^i} \bmod N\rangle & |j_i\rangle = 1\\ U|u\rangle = |u\rangle & |j_i\rangle = 0\end{cases} \tag{3.6}$$

Essentially, with respect to each $|j_i\rangle$ which acts as a control, we apply the map $|u\rangle \mapsto |u\cdot g^{2^j} \bmod N\rangle$ when $|j_i\rangle = |1\rangle$, and does not apply the map when $|j_i\rangle = |0\rangle$. Furthermore, since we are applying

the controlled-U^{2^i} operations when register 1 is still in superposition ($|j_i\rangle$ has a probability of being either 0 or 1), we say that the qubits in register 2 is "entangled" with qubits in register 1[Pha19].

Consider register 2 only, represented by the second tensor product in (3.2). After all the controlled-U^{2^i} operation, we have

$$U\bigotimes_{i=1}^{n}|0\rangle = U^{2^{m-1}}\cdots U^{2^0}\bigotimes_{i=1}^{n}|0\rangle = |g^j \bmod N\rangle \quad \text{for each } |j\rangle \tag{3.7}$$

To summarize the encoding process, we have the following:

$$\begin{aligned}
|\psi_0\rangle &= \bigotimes_{j=1}^{m}|0\rangle \otimes \bigotimes_{i=1}^{n}|0\rangle \\
|\psi_1\rangle = H^m|\psi_0\rangle &= \bigotimes_{j=1}^{m}\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\otimes\bigotimes_{i=1}^{n}|0\rangle \cong \frac{1}{\sqrt{2^m}}\sum_{j=1}^{2^m}|j\rangle\otimes\bigotimes_{i=1}^{n}|0\rangle \\
|\psi_2\rangle = U^n|\psi_1\rangle &= \frac{1}{\sqrt{2^m}}\sum_{j=1}^{2^m}|j\rangle\otimes|g^{j_1 2^{j-1}+\cdots+j_m 2^0}\bmod N\rangle = \frac{1}{\sqrt{2^m}}\sum_{j=1}^{2^m}|j\rangle\otimes|g^j \bmod N\rangle
\end{aligned} \tag{3.8}$$

After this process, we have encoded all powers j, that we raise g to, along with the value $g^j \bmod N$. This is convenient, as whenever we observe periodicity of $g^j \bmod N$, we can identify the different j that lead to this periodicity, which we can then use to find the period r.

As an exmaple[Pha19], lets consider trying to factor $N = 21$, then pick $m = 9$ where $2^9 = 512 \geq 21^2$. Let $g = 2$. Then, we have

$$\begin{aligned}
|\psi_1\rangle &= \frac{1}{\sqrt{512}}\sum_{j=1}^{512}|j\rangle\otimes\bigotimes_{i=1}^{n}|0\rangle \\
|\psi_2\rangle &= \frac{1}{\sqrt{512}}\sum_{j=1}^{512}|j\rangle\otimes|2^j \bmod N\rangle \\
&= \frac{1}{\sqrt{512}}(\,|0\rangle+|6\rangle+\cdots+|510\rangle)\otimes|1\rangle \\
&\quad +|1\rangle+|7\rangle+\cdots+|511\rangle)\otimes|2\rangle \\
&\quad +\cdots \\
&\quad +|5\rangle+|11\rangle+\cdots+|509\rangle)\otimes|11\rangle)
\end{aligned} \tag{3.9}$$

In this explicit form, we can easily see the periodicity of the powers j is 6. Computationally, we will need to apply a transformation on $|\psi_2\rangle$ to extract this answer.

3.4. **Quantum Fourier Transform.** To find the periodicity of the power j, we use the Quantum Fourier Transform. The classical Discrete Fourier Transform is described as the following:

Definition 3.4 (Discrete Fourier Transform)**.** The Discrete Fourier Transform acts on a vector $(x_0,\cdots,x_{N-1})$ and maps it to a vector $(y_0,\cdots,y_{N-1})$ according to the formula

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

Similarly, the quantum version is the following:

Definition 3.5 (Quantum Fourier Transform)**.** the quantum Fourier transform acts on a quantum state $\sum_{j=0}^{N-1} x_j |j\rangle$ and maps it to the quantum state $\sum_{j=0}^{N-1} y_j |j\rangle$ by the following map

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i \frac{jk}{N}} |k\rangle$$

It is easy to check that this map gives the same coefficient of y_k as the classical Discrete Fourier Transform[Pha19]. The fourier transform is used here because it picks up the periodicity of (3.9). Consider in the example in (3.9), and treat the value on the second register as the y value of a function and and $|j\rangle$ the x value. Then, the Quantum Fourier Transform will produce a corresponding frequency distribution. The frequency will be directly proportional to $\frac{1}{r}$, where r is the period[NC10]. Below, we give a more convenient definition for the Quantum Fourier Transform:

Definition 3.6 (Tensor product representation of Quantum Fourier Transform)**.** The following transformation on $|j\rangle$ is equivalent to the Quantum Fourier Transform:

$$\begin{aligned} |j\rangle &= |[j_1 j_2 \cdots j_m]_2\rangle \cong |j_1\rangle \otimes |j_2\rangle \otimes \cdots \otimes |j_m\rangle \\ &\mapsto \frac{1}{2^{\frac{m}{2}}} (|0\rangle + e^{2\pi i [0.j_m]_2} |1\rangle) \otimes (|0\rangle + e^{2\pi i [0.j_{m-1} j_m]_2} |1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i [0.j_1 \cdots j_m]_2} |1\rangle) \end{aligned}$$

This formula can be derived directly from the definition of quantum fourier transform, and it's derivation will be omitted[NC10]. Essentially, we convert the transformation on $|j\rangle$ acting on a single system, into transformation acting on m qubit systems individually, which is what we have.

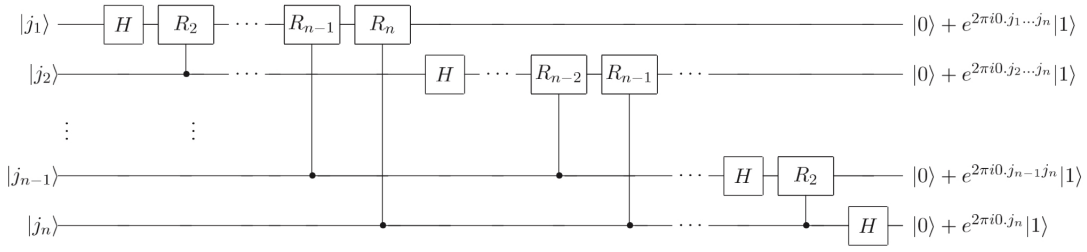

FIGURE 3. The Quantum Fourier Transform

Where the gates have the following matrix representation:

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

$$R_k = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{pmatrix}$$

If we count the number of gates, which equals to the number of steps needed to do the Quantum Fourier Transform, we have $1 + 2 + \cdots + m = \frac{(1+m)m}{2}$ steps, giving this transform a time complexity of $\mathcal{O}(n^2)(n = \log_2(N)$, from the constraint we have $2n+1 \geq m \geq 2n)$. To conduct this step on a classical computer, by doing direct computation using Definition 3.5, we need a total of $N^2 = 2^{2n}$ steps, and even with the help of definition 3.6, we still need $n \cdot 2^n$ steps, which has a time complexity of $\mathcal{O}(n2^n)$[NC10].

Overall considering all the steps, the slowest part of the algorithm actually comes from (3.7), where the modular exponentiation has a time complexity of $\mathcal{O}(n^3)$[NC10], which makes that the

algorithm's total time complexity. Nonetheless, this is considered an exponential speedup over the general number field sieve.

References

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: https://doi.org/10.1145/359340.359342.

[Gro96] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*. Philadelphia, Pennsylvania, United States: ACM Press, 1996.

[Sho97] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. URL: http://dx.doi.org/10.1137/S0097539795293172.

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. ISBN: 9780521424264. URL: https://books.google.com.hk/books?id=8Wjqvsoo48MC.

[Cor+09] T.H. Cormen et al. *Introduction to Algorithms, third edition*. Computer science. MIT Press, 2009. ISBN: 9780262033848. URL: https://books.google.com.hk/books?id=i-bUBQAAQBAJ.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[Pha19] Anna Phan. *qiskit-community-tutorials/algorithms/shor$_a$lgorithm.ipynbatmasterqiskit−community/qiskit−community−tutorials−−−github.com*. https://github.com/qiskit-community/qiskit-community-tutorials/blob/master/algorithms/shor_algorithm.ipynb. [Accessed 27-04-2024]. 2019.