Shor's Algorithm and the Quantum Fourier Transform

Fang Xi Lin

McGill University
fangxi.lin@mail.mcgill.ca

Abstract

Large numbers have traditionally been believed to be difficult to factor efficiently on a classical computer. Shor's quantum algorithm gives a way to factor integers in polynomial time using a quantum computer. In addition, the algorithm also allows the computation of discrete logarithms in polynomial time. The algorithm relies in a crucial way on the quantum Fourier transform. We will briefly introduce quantum mechanics and quantum computation, then describe both the quantum Fourier transform and Shor's algorithm in detail.

Introduction

The problem of how to factor a large integer efficiently has been studied extensively in number theory. It is generally believed that factorization of a number n is hard to do in a efficient way. That is, it cannot be done in a number of steps which is polynomial in the length of the integer we're trying to factor[1]. The RSA cryptosystem, among others, relies on the presumed difficulty of this task. Classically, the fastest known algorithm is the General Number Field Sieve (GNFS) algorithm, which works in super-polynomial, but sub-exponential time.

In 1994, Peter Shor discovered an algorithm that can factor numbers in polynomial time using a quantum computer[10], a drastic improvement over the GNFS. Shor's algorithm consists of a classical and a quantum part. The classical part involves modular exponentiation via repeated squaring, which can be performed quickly. The quantum part involves a "quantum Fourier transform". We will prove that in certain cases, the quantum Fourier transform can be constructed in polynomial time, wherein lies the efficiency of the algorithm.

Although Shor's factoring algorithm is much more publicized, Shor's ideas will allow us to compute discrete logarithms, which is also believed to be a hard task for a classical computer in the same sense that factoring numbers is.

There have been several successful experimental demonstrations of the factoring algorithm. In 2001, A group from IBM was able to factor 15 using a quantum computer of 7 qubits implemented using nuclear magnetic resonance[13]. Since then, 15 and 21 were factored using photonic qubits. However, there were concerns that some of these experimentations were not true compilations of Shor's algorithm[12].

1 Computation and Complexity Classes

The ability to compute is limited by two *resources*: space (memory) and time. The difficulty of computing allows problems to be categorized into different *complexity classes*. This is the subject of

[1]Note that a number of size d has input length $\log d$.

study for a computation complexity theorist. We will, however, try to give some intuitive insight into the theory of complexity classes.

Consider an algorithm which takes in an input of length n (for example, the number of digits in a number). We call this a *polynomial time algorithm* if it doesn't take more than $C\,n^k$ steps, for some fixed $C,\ k > 0$, to compute the answer. We denote this by $\mathcal{O}(n^k)$. These are considered efficient algorithms (although n^{1000} is not really efficient in practice). The class of problems solved by these algorithms is called $\mathtt{P}$.

Another important complexity class is called $\mathtt{NP}$. This is the class of problems whose solutions can be verified in polynomial time. For example, once we find the factorization of some number $N = pq$, we can efficiently verify that $pq = N$. Indeed, we have $\mathtt{P} \subseteq \mathtt{NP}$ (the reverse inclusion is a open problem, one of the seven Millennium problems).

Both complexity classes presented above are bounded by time. There are also a number of complexity classes bounded by space. $\mathtt{PSPACE}$ is such a class, which contains problems that can be solved with a polynomial amount of bits in input size.

There are two more complexity classes which are important to this paper. The first is the $\mathtt{BPP}$, which are problems that can be solved with a bounded probability of error in polynomial time. The second one is the $\mathtt{BQP}$, which is essentially the same thing on a quantum machine. Factoring numbers using Shor's algorithm is $\mathtt{BQP}$.

The known relationship between these complexity classes is:

$$\mathtt{P} \subseteq \mathtt{BPP}, \mathtt{NP}, \mathtt{BQP} \subseteq \mathtt{PSPACE} \tag{1}$$

In addition, we also have $\mathtt{BPP} \subseteq \mathtt{BQP}$. The relationship between $\mathtt{BPP}$, $\mathtt{BQP}$ and $\mathtt{NP}$ is unknown.

2 Quantum Mechanics and Quantum Computation

Shor's algorithm is a *quantum* algorithm. That is, it exploits the fact that we can have *superpositions* of *quantum states*. We will follow Nielsen and Chuang[8] to build up some concepts in quantum mechanics, quantum computation, and quantum circuits in the following section.

2.1 Quantum Mechanics

There are multiple formulations of quantum mechanics. The one which will be useful to us is the matrix mechanics (due to Heisenberg) formulation. Without digging into subtle details, objects in quantum mechanics have essentially a one-to-one correspondence to objects in linear algebra. The quantum mechanical vector spaces are called *Hilbert* spaces, which are normed complex vector spaces. We will assume that the Hilbert spaces are finite dimensional in this paper.

The standard quantum mechanical notation[2] for a (column) vector in a vector space is a $|\psi\rangle$ (a *ket*). It is used to represent a quantum state. The notation for a dual (row) vector is $\langle\phi|$ (a *bra*). The inner product between two vectors is then denoted $\langle\phi|\psi\rangle$ (a *bra-ket*), which yields in complex number in general. In general, we require that the quantum states be normalized, i.e. $|\langle\psi|\psi\rangle|^2 = 1$. Physically, $|\langle\phi|\psi\rangle|^2$ is the probability of observing $|\psi\rangle$ in the state $|\phi\rangle$. We will call the zero vector 0 instead of $|0\rangle$ since the latter will be used for something else.

The matrices are called *operators* in quantum mechanics. We will require that all our operators to be unitary, i.e., given an operator A, we need $A^\dagger := (A^T)^* = A^{-1}$. We will use $\langle\phi|\,A\,|\psi\rangle$ to denote the inner product between $\langle\phi|$ and $A\,|\psi\rangle$.[3]

[2]Due to Paul Dirac

[3]Equivalently, between $\langle\phi|\,A^\dagger$ and $|\psi\rangle$

There is a useful way of representing linear operators called the *outer product* representation. The outer product is a ket-bra, which is a linear operator defined by $(|\psi\rangle\langle\phi|)|\phi'\rangle := \langle\phi'|\phi\rangle|\psi\rangle$.

Now let A be an operator and let $|j\rangle$ be any orthonormal basis. Then any quantum state $|\psi\rangle$ can be written as $|\psi\rangle = \sum_j \alpha_j |j\rangle$ and $\langle j|\psi\rangle = \alpha_j$. Therefore we have

$$\left(\sum_j |j\rangle\langle j|\right)|\psi\rangle = \sum_j \langle j|\psi\rangle|j\rangle = \sum_j \alpha_j|j\rangle = |\psi\rangle \tag{2}$$

for any arbitrary $|\psi\rangle$. Hence, we have $I = \sum_j |j\rangle\langle j|$. This allows us to represent any operator A as a linear combination of outer products since

$$A = IAI = \sum_{j,k} |j\rangle\langle j| A |k\rangle\langle k| = \sum_{j,k} \langle j| A |k\rangle |j\rangle\langle k| \tag{3}$$

so that $\langle j| A |k\rangle$ is the j, k^{th} entry of the matrix representation of A.

We will make use of the tensor product, $\otimes$, extensively. In quantum mechanics, tensor products are used to describe multi-particle systems (in our case, multi-qubit systems).

Definition 1. Let $\mathcal{H}_A$ be an N dimensional Hilbert space with orthonormal basis $|j\rangle$ and let $\mathcal{H}_B$ be an M dimensional Hilbert space with orthonormal basis $|k\rangle$. Then the *tensor product* $\mathcal{H}_A \otimes \mathcal{H}_B$ is the NM dimensional system spanned by the pair $|j\rangle \otimes |k\rangle$ where

$$\left(\langle j'| \otimes \langle k'|\right)\left(|j\rangle \otimes |k\rangle\right) = \langle j'|j\rangle\langle k'|k\rangle = \delta_{jj'}\delta_{kk'}. \tag{4}$$

To every vector $|\psi_A\rangle = \sum_j \alpha_j |j\rangle \in \mathcal{H}_A$ and $|\psi_B\rangle = \sum_k \beta_k |k\rangle \in \mathcal{H}_B$ we can associate a vector $|\psi_A\rangle \otimes |\psi_B\rangle = \sum_{jk} \alpha_j\beta_k |j\rangle \otimes |k\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$. We call the resulting state a *product state*. We sometimes suppress the $\otimes$ symbol and write $|\psi_A\rangle \otimes |\psi_B\rangle$ as $|\psi_A\rangle|\psi_B\rangle$ or even $|\psi_A\psi_B\rangle$. Not all states in $H_A \otimes H_B$ can be written as $|\psi_A\rangle \otimes |\psi_B\rangle$ for some $|\psi_A\rangle \in \mathcal{H}_A$ and $|\psi_B\rangle \in \mathcal{H}_B$ (otherwise we would have an $N + M$ dimensional system!). The states which are not product states are said to be *entangled*.

Similarly, for every operator A acting on $|\psi_A\rangle$ and operator B acting on $|\psi_B\rangle$, we can define an operator $A \otimes B$ acting on $|\psi_A\rangle \otimes |\psi_B\rangle$ as $(A \otimes B)(|\psi_A\rangle \otimes |\psi_B\rangle) = A|\psi_A\rangle \otimes B|\psi_B\rangle$.

2.2 Quantum Computation

Just as classical computers use bits to compute, quantum computers use *quantum bits* or *qubits*. A qubit is a vector $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$. Operators which act on qubits are two-by-two unitary matrices. We call these operators *quantum gates*, since they are analogous to the logic gates in classical computer science. In general, a circuit of quantum gates, or a *quantum circuit*, looks like

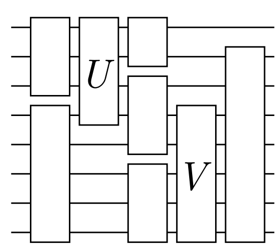

Figure 1: *A quantum circuit. Time goes from left to right, wires represent qubits, and rectangles represent unitary operators/quantum gates.*

The most important gate to us will be the *Hadamard gate*.

Definition 2. Let $\{|0\rangle, |1\rangle\}$ be a basis. Then the *Hadamard gate* is

$$H \to \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{5}$$

In particular, it sends the basis vectors to a uniform superposition of the basis vectors. That is, we get $\frac{1}{2}$ chance of measuring either $|0\rangle$ or $|1\rangle$. In a quantum circuit, it is represented by

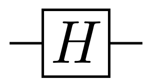

Figure 2: *A Hadamard gate*

As we will see in section 4, the quantum Fourier transform can be represented with Hadamard gates. We will also need the notion of a *controlled gate*.

Definition 3. If U is a single qubit unitary operation, a *controlled-U* is a two qubit operation on a *control* and a *target qubit* such that if the control qubit is set, then the gate will act on the target qubit. If not, the target qubit is left alone. That is, we have

$$U_c : \begin{cases} |0\rangle|0\rangle \mapsto |0\rangle|0\rangle \\ |0\rangle|1\rangle \mapsto |0\rangle|1\rangle \\ |1\rangle|0\rangle \mapsto |1\rangle U|0\rangle \\ |1\rangle|1\rangle \mapsto |1\rangle U|1\rangle \end{cases} \tag{6}$$

where the left qubit is the control qubit and the right qubit is the target qubit. In a circuit, it looks like

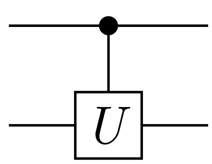

Figure 3: *A controlled-U gate. On the top is the control qubit, on the bottom is the target qubit.*

3 Classical Part: Reduction to Order Finding

Shor's algorithm does not allow us to factor a number directly. Instead, it allows us to find the order of an element a modulo n in polynomial time. We will show that the problem of finding a non-trivial factor to n can be reduced (efficiently) to finding the order of a non-trivial element in $\mathbb{Z}/n\mathbb{Z}$[6].

Lemma 4. *Given a composite number n, and x non-trivial square root of 1 modulo n (i.e. $x^2 = 1 \pmod{n}$ but x is neither 1 nor $-1 \mod n$, then either $\gcd(x-1, n)$ or $\gcd(x+1, n)$ is a non-trivial factor of n.*

Proof. Since $x^2 \equiv 1 \pmod{n}$, we have $x^2 - 1 \equiv 0 \pmod{n}$. Factoring, we get $(x-1)(x+1) \equiv 0 \pmod{n}$. This implies that n is a factor of $(x+1)(x-1)$. Since $(x \pm 1) \not\equiv 0 \pmod{n}$, n has a non-trivial factor with $x+1$ or $x-1$. To find this common factor efficiently, we apply Euclid's algorithm to get $\gcd(x-1, n)$ or $\gcd(x+1, n)$. $\square$

Example 5. Let $n = 55 = 5 \cdot 11$. We find that 34 is a square root of $1 \mod n$ since $34^2 = 1156 = 1 + 21 \cdot 55$. Computing, we get $\gcd(33, 55) = 11$ and $\gcd(35, 55) = 5$.

Lemma 6. *Let n be odd, then at least half the elements in $(\mathbb{Z}/n\mathbb{Z})^\times$ have even order.*

Proof. Suppose $\mathrm{ord}(x) = r$ is odd. Then $(-x)^r = (-1)^r x^r = (-1)^r = -1 \pmod{n}$. Hence $-x$ must have order $2r$, which is even. Therefore, at least half the elements in $(\mathbb{Z}/n\mathbb{Z})^\times$ have even order. □

Equipped with these tools, we will proceed to prove the main result that allows us to reduce factorization of n to finding the order of an element in $\mathbb{Z}/n\mathbb{Z}$.

Theorem 7. *Let n be an odd integer and let $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$ be the prime factorization of n. Then the probability that a uniform randomly chosen $x \in \mathbb{Z}/n\mathbb{Z}$ has even order r and $x^{r/2} \not\equiv -1 \pmod{n}$ is at least $1 - \frac{1}{2}^{k-1}$.*

Proof. By the Chinese Remainder Theorem, choosing $x \in (\mathbb{Z}/n\mathbb{Z})^\times$ (uniform) randomly is equivalent to choosing $x_i \in (\mathbb{Z}/p_i^{e_i}\mathbb{Z})^\times$ for each p_i randomly. Let r be the order of x and let r_i be the order of x_i. In particular, $x^{r/2}$ is never 1 modulo n. We want to show that the probability of either r being odd or $x^{r/2} \equiv -1 \pmod{n}$ is at most $\frac{1}{2}^{k-1}$.

Note that $r = \mathrm{lcm}(r_1, r_2, \dots, r_k)$ (where lcm denotes the least common multiple). To see this, $x^r \equiv 1 \pmod{n} \Rightarrow x^r \equiv 1 \pmod{p_i^{e_i}}$, hence r is a multiple of each r_i. It is the least such number and hence the least common multiple of the r_i's.

Suppose that r is odd. This happens only if all of the r_i's are odd. r_i is odd with probability at most one-half by lemma 6. Hence, r is odd with probability at most $\frac{1}{2}^{k}$.

Now suppose that r is even. We still have to worry about the possibility that $x^{r/2} \equiv \pm 1 \pmod{n}$. By the Chinese Remainder Theorem, this happens only if $x^{r/2} \equiv \pm 1 \pmod{p_i^{e_i}}$ for every p_i. We need to avoid these cases since $\equiv +1$ means r wasn't the order, and $\equiv -1$ doesn't yield a useful factorization. The probability of choosing an x such that one of these two cases happens is $2 \cdot 2^{-k} = 2^{-k+1}$.

Combining the probabilities, we get a success probability of at least $(1 - 2^{-k})(1 - 2^{-k+1}) \geq 1 - 3 \cdot 2^{-k}$. □

By lemma 4 and theorem 7, given a composite number n and the order r of some $x \in \mathbb{Z}/n\mathbb{Z}$, we can compute $\gcd(x^{r_x/2} \pm 1, n)$ efficiently using Euclid's algorithm. This gives a non trivial factor of n unless r is odd or $x^{r_x/2} \equiv -1 \pmod{n}$. In particular, if n is a semi-prime, i.e. it is a product of two primes p, q, then theorem 7 implies that n will be factored with probability $\frac{1}{2}$.

4 Fourier Transforms

Since finding a factor of n given the order of some element in $\mathbb{Z}/n\mathbb{Z}$ can be done efficiently even on a classical computer, it still remains to be shown that we can find the order of the element efficiently. It is unknown how to quickly find the order of a given element on a classical computer, but Shor's order finding algorithm will allow us to do so by employing a quantum computer. The order finding algorithm relies crucially on a unitary operator F_n, the "quantum Fourier transform" (QFT) operator, which acts like a discrete Fourier transform. We assume the knowledge of the usual Fourier transform for the following section.

Before discussing the quantum Fourier transform, we will talk a bit about the discrete Fourier transform (DFT) as well as the Fast Fourier Transform (FFT) algorithm. The QFT will be constructed to be essentially the equivalent of the FFT with a quantum circuit[11].

4.1 Discrete Fourier Transform

Definition 8. [4] Let $f = (f_0, f_1, \ldots, f_{N-1})$ be a vector in $\mathbb{C}^N$. The *discrete Fourier transform* is a map

$$\begin{aligned} \mathcal{F} :& \mathbb{C}^N \to \mathbb{C}^N \\ & f \mapsto \tilde{f} \end{aligned}$$

defined by

$$\tilde{f}_j := \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \zeta^{-jk} f_k \tag{7}$$

where $\zeta = \exp(\frac{2\pi i}{N})$ is the N^{th} root of unity.

We will use $\tilde{f}$ to denote the DFT of f. As in the case of the usual Fourier transform, there's an inverse Fourier transform given by the expected formula.

Lemma 9. *The inverse discrete Fourier transform, $\mathcal{F}^{-1}$, is given by*

$$f_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{n-1} \zeta^{jk} \tilde{f}_k \tag{8}$$

We can check how the Fourier transform acts on the standard basis. Let $\{e^1, e^2, \ldots, e^N\}$ be the standard basis of $\mathbb{C}^N$, where e^l denotes the vector has has 1 at the l^{th} component and 0 elsewhere (i.e. $e^l_j = \delta_{jl}$. Then the DFT of e^l is given by

$$\tilde{e}^l_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \zeta^{-jk} \delta_{jl} = \frac{1}{\sqrt{n}} \zeta^{-jl} \tag{9}$$

The matrix representation of $\mathcal{F}$ in the standard basis is

$$\mathcal{F} \to \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \zeta^{-1} & \zeta^{-2} & \cdots & \zeta^{-(N-1)} \\ 1 & \zeta^{-2} & \zeta^{-4} & \cdots & \zeta^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \zeta^{-(N-1)} & \zeta^{-2(N-1)} & \cdots & \zeta^{-(N-1)^2} \end{pmatrix} \tag{10}$$

Example 10. Consider the $N = 2$ DFT. We have $\zeta^{-1} = \exp(-\frac{2\pi i}{2}) = -1$ and so

$$\mathcal{F} \to \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{11}$$

which is a rotation by 45°, and also looks like a Hadamard gate! We will see more of this when we discuss the QFT.

[4] We use the physicists and mathematicians' convention to define the DFT and everything that follows. Computer scientists usually have the sign on the exponent of ζ reversed.

Since performing the DFT on a vector f is like a matrix multiplication, it takes N (complex) multiplications and $N-1$ additions for each component and there are N components so we have N^2 multiplications and $N(N-1)$ additions. Since additions can be efficiently computed, the speed is limited by the N^2 multiplications. Hence, we shall only consider the number of multiplications. As we shall see shortly, the FFT will allow us to perform DFT in fewer operations by exploiting some symmetries of the DFT.

Many of the properties of the DFT are analogous to the properties of the FT. For example, we claim the DFT convolution theorem holds[14].

Definition 11. A *circular* or *cyclic convolution* of f and g, denoted by $f*g$, is a map $\mathbb{C}^N \times \mathbb{C}^N \to \mathbb{C}^N$ given component-wise by

$$(f*g)_j = \sum_{k=0}^{N-1} f_k\, g_{j-k} \tag{12}$$

where $g_{-a} := g_{N-a}$.

Theorem 12. (Circular Convolution Theorem) *Let $h = f*g$, then $\tilde{h}_j$, the j^{th} component of $\tilde{h}$, is given by $\tilde{f}_j \cdot \tilde{g}_j$ where $\cdot$ denotes the usual multiplication.*

4.2 Fast Fourier Transform

For the following, we assume that $N = 2^m$.

Consider the roots of unity, ζ. Observe that

$$\begin{aligned} \zeta^{j+N/2} &= \exp\left(\frac{2\pi i \cdot (j+N/2)}{N}\right) \\ &= \exp\left(\frac{2\pi i j}{N}\right)\exp(\pi i) \\ &= -\zeta^j \end{aligned} \tag{13}$$

and similarly

$$\zeta^{j+N} = \zeta^j. \tag{14}$$

This suggests that we may be able to split f into smaller parts.

Since f has $N = 2^m$ components, we can divide f into an even and an odd part. More precisely, we define $f_{\text{even}} = (f_0, f_2, \ldots, f_{N-2})$ and $f_{\text{odd}} = (f_1, f_3, \ldots, f_{N-1})$. Now, we can rewrite equation (7) as

$$\begin{aligned} \tilde{f}_j &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N/2-1} \zeta^{-j\cdot 2k} f_{\text{even}_k} + \frac{1}{\sqrt{N}} \sum_{k=0}^{N/2-1} \zeta^{-j\cdot(2k+1)} f_{\text{odd}_k} \\ &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N/2-1} \zeta^{-j\cdot 2k} f_{\text{even}_k} + \frac{\zeta^{-j}}{\sqrt{N}} \sum_{k=0}^{N/2-1} \zeta^{-j\cdot 2k} f_{\text{odd}_k} \\ &= \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{N/2}} \sum_{k=0}^{N/2-1} \zeta^{-j\cdot 2k} f_{\text{even}_k} + \frac{\zeta^{-j}}{\sqrt{N/2}} \sum_{k=0}^{N/2-1} \zeta^{-j\cdot 2k} f_{\text{odd}_k} \right) \\ &= \frac{1}{\sqrt{2}} \left(\tilde{f}_{\text{even}_j} + \zeta^{-j} \tilde{f}_{\text{odd}_j} \right) \end{aligned} \tag{15}$$

where in the second step we pull out a factor of ζ^{-j}, called the *twiddle factor*, out of the f_{odd} term, and in the fourth step we apply the definition of the DFT to f_{even} and f_{odd}.

Since the DFT is periodic with period n (i.e. $\tilde{f}_{j+N} = \tilde{f}_j$), f_{even} and f_{odd} are periodic with period $N/2$. Hence, combining (13), (14), (15), we get

$$\sqrt{2}\tilde{f} = \tilde{f}_{\text{even}_j} + \zeta^{-j}\tilde{f}_{\text{odd}_j} \qquad \text{if } 0 \leq j \leq N/2-1 \tag{16}$$

$$\sqrt{2}\tilde{f} = \tilde{f}_{\text{even}_j} - \zeta^{-j}\tilde{f}_{\text{odd}_j} \qquad \text{if } N/2 \leq j \leq N-1 \tag{17}$$

This gives us a way to compute the DFT of a vector of size N in terms of smaller vectors of size $N/2 = 2^{m-1}$. To compute f_{even} and f_{odd}, we require $(N/2)^2 + (N/2)^2 = N^2/2$ multiplications. To compute $\zeta^{-j}\tilde{f}_{\text{odd}}$, we require another $N/2$ multiplications. For the $\sqrt{2}$, we can "collect" them and then multiply the total factor of $\sqrt{2^m} = \sqrt{N}$ in at the end of the recursion as N additional multiplications (instead of multiplying in every step). Hence, we require $N^2/2 + N/2$ multiplications in total, which is about a factor of two faster than the N^2 multiplications in the original DFT.

Since the smaller vectors still have length divisible by 2, we can apply this procedure recursively until we get a DFT of size 2, which are all "classical Hadamard transforms" as in example 10. In general, the FFT allows us to compute the DFT in $\mathcal{O}(N \log N) = \mathcal{O}(2^m \log 2^m)$ operations, which is still exponential in m.

Example 13. (Fast multiplication of two polynomials[5]) Let $p(x) = \alpha_0 + \alpha_1 x + \ldots + \alpha_{N-1}x^{N-1}$ and $q(x) = \beta_0 + \beta_1 x + \ldots + \beta_{N-1}x^{N-1}$ be two polynomials with complex coefficients. Then

$$\begin{aligned} p(x)q(x) &= \left(\sum_{i=0}^{N-1} \alpha_i x^i\right)\left(\sum_{j=0}^{N-1} \beta_j x^j\right) \\ &= \left(\sum_{k=0}^{2N-2} \lambda_k x^k\right) \end{aligned} \tag{18}$$

where

$$\lambda_k = \sum_{l=0}^{N-1} \alpha_l \beta_{k-l} \tag{19}$$

Therefore, computing $p(x)q(x)$ and hence computing the λ_k's directly takes N^2 multiplications. However, equation (19) looks very much like a discrete convolution. Let us view $p(x), q(x)$ as vectors with their coefficients as components. That is, $p \to (\alpha_0, \alpha_1, \ldots, \alpha_{N-1})$, $q \to (\beta_0, \beta_1, \ldots, \beta_{N-1})$. Now, we can append 0 as necessary to p and q to make them $2N$ dimensional vectors (because we want p and q to have the same dimension as $p * q$). We can express the λ_k's as

$$\lambda_k = \sum_{l=0}^{2N-1} \alpha_{l \bmod 2N} \cdot \beta_{k-l \bmod 2N} \tag{20}$$

which gives us the circular convolution of p and q! Hence, we can compute λ_k's by first performing the DFT on the p and q vectors which takes $N \log N$ multiplications. Then, we multiply the resulting vector component-wise which takes $2N$ multiplications. Finally, we take the inverse DFT and obtain the coefficients λ_k.

4.3 Quantum Fourier Transform

Finally, we move onto the main topic of this section, the quantum Fourier transform. We will follow our steps for constructing the DFT and FFT.

Definition 14. Let $\{|0\rangle, |1\rangle, \ldots, |N-1\rangle\}$ be an orthonormal basis for a quantum system and let $|\phi\rangle = \sum_{j=0}^{N-1} |j\rangle$ be a quantum state. Then the *quantum Fourier transform* F_N is a map defined by

$$|\phi\rangle = \sum_{j=0}^{N-1} |j\rangle \mapsto \sum_{j=0}^{N-1} \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \zeta^{-jk} |k\rangle \tag{21}$$

In particular, the basis states transform as

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \zeta^{-jk} |k\rangle \tag{22}$$

And hence we get an representation for F_N

$$F_N = \frac{1}{\sqrt{N}} \sum_{j,k=0}^{N-1} \zeta^{-jk} |k\rangle \langle j| \tag{23}$$

Note that since $\bar{\zeta} = \zeta^{-1}$, we have

$$F_n^\dagger = \frac{1}{\sqrt{N}} \sum_{j,k=0}^{N-1} \zeta^{jk} |k\rangle \langle j| \tag{24}$$

We can easily check that the quantum Fourier transform is unitary.

$$\begin{aligned}
F_N F_N^\dagger &= \frac{1}{N} \sum_{j,k=0}^{N-1} \zeta^{-jk} |k\rangle \langle j| \sum_{r,s=0}^{N-1} \zeta^{rs} |r\rangle \langle s| \\
&= \frac{1}{N} \sum_{j,k,r,s=0}^{N-1} \zeta^{rs-jk} |k\rangle \langle j|r\rangle \langle s| \\
&= \frac{1}{N} \sum_{j,k,r,s=0}^{N-1} \zeta^{rs-jk} \delta_{jr} |k\rangle \langle s| \\
&= \frac{1}{N} \sum_{k,s=0}^{N-1} \left(\sum_{r=0}^{N-1} \zeta^{r(s-k)} \right) |k\rangle \langle s| \\
&= \sum_{k,s=0}^{N-1} \delta_{ks} |k\rangle \langle s| \\
&= \sum_{k=0}^{N-1} |k\rangle \langle k| = I
\end{aligned} \tag{25}$$

where we used the fact that $\sum_{r=0}^{N-1} \exp\left(\frac{2\pi i r(s-k)}{N}\right) = N\delta_{sk}$.

As in the case of DFT, constructing F_N naïvely is not very efficient. We will try to implement the QFT as a quantum circuit efficiently. Recall that we assumed $N = 2^m$. Consider the Fourier transformed state (22), if we express j in binary as $j_1 j_2 \dots j_m \in \{0,1\}^m$ and $j = j_1 2^{m-1} + j_2 2^{m-2} + \dots + j_m 2^0$, we will see that it is in fact a product state.

Theorem 15. *$|j\rangle$ is a product state which can be written as a product of m qubits*[3]

$$|j\rangle \to \frac{1}{\sqrt{N}} \left(|0\rangle + e^{-2\pi i (0.j_m)} |1\rangle\right) \otimes \left(|0\rangle + e^{-2\pi i (0.j_{m-1} j_m)} |1\rangle\right) \otimes \dots \otimes \left(|0\rangle + e^{-2\pi i (0.j_1 j_2 \dots j_m)} |1\rangle\right) \tag{26}$$

where $(0.j_1 j_2 \dots j_m) = j_1 2^{-1} + j_2 2^{-2} + \dots + j_m 2^{-m}$ denotes the binary fraction.

Proof. To see this, write out the binary expansion of $|k\rangle$.

$$\begin{aligned}
|j\rangle \to \frac{1}{\sqrt{N}} \sum_{k=0}^{n-1} \zeta^{-jk} |k\rangle &= \frac{1}{\sqrt{N}} \sum_{k_1, k_2, \dots, k_m \in \{0,1\}} \zeta^{-j \sum_{r=1}^{m} 2^{m-r} k_r} |k_1\rangle |k_2\rangle \dots |k_m\rangle \\
&= \frac{1}{\sqrt{N}} \sum_{k_1, k_2, \dots, k_m \in \{0,1\}} \bigotimes_{r=1}^{m} \zeta^{-j 2^{m-r} k_r} |k_r\rangle \\
&= \frac{1}{\sqrt{N}} \bigotimes_{r=1}^{m} \left(\sum_{k_r \in \{0,1\}} \zeta^{-j 2^{m-r} k_r} |k_r\rangle \right) \\
&= \frac{1}{\sqrt{N}} \bigotimes_{r=1}^{m} \left(|0\rangle + \zeta^{-j 2^{m-r}} |1\rangle\right) = \frac{1}{\sqrt{n}} \bigotimes_{r=1}^{m} \left(|0\rangle + e^{-2\pi i j 2^{m-r}/2^m} |1\rangle\right) \\
&= \frac{1}{\sqrt{N}} \bigotimes_{r=1}^{m} \left(|0\rangle + e^{-2\pi i j 2^{-r}} |1\rangle\right)
\end{aligned} \tag{27}$$

where in the second step we expanded the exponential as product and regrouped terms. Notice the similarities between this procedure and the FFT (we essentially did the whole FFT recursion in one fell swoop).

Expanding the j in the "twiddle factor" in binary, we get

$$\begin{aligned}
\exp\left(-2\pi i \sum_{l=1}^{m} 2^{m-l} j_l / 2^r\right) &= \exp\left(-2\pi i \sum_{l=1}^{m} 2^{m-r-l} j_l\right) \\
&= \exp\left(-2\pi i (0.j_{m-r+1} j_{m-r+2} \dots j_m)\right)
\end{aligned} \tag{28}$$

so that (27) gives

$$|j\rangle \to \frac{1}{\sqrt{n}} \bigotimes_{r=1}^{m} \left(|0\rangle + e^{-2\pi i (0.j_{m-r+1} j_{m-r+2} \dots j_m)} |1\rangle\right) \tag{29}$$

as was required. □

Note that the value of $e^{-2\pi i (0.j_{m-r+1} j_{m-r+2} \dots j_m)}$ is either 1 or -1, like a Hadamard transformed qubit. Moreover, note that the last qubit depends on all the input qubits but the dependence decreases as we go further. We can use this to construct a quantum circuit. We first need a new quantum gate.

Definition 16. A *rotation gate* is a unitary operator defined as

$$R_s := \begin{pmatrix} 1 & 0 \\ 0 & \exp\left(\frac{-2\pi i}{2^s}\right) \end{pmatrix} \tag{30}$$

Now consider the following circuit which almost gives us the desired transformation

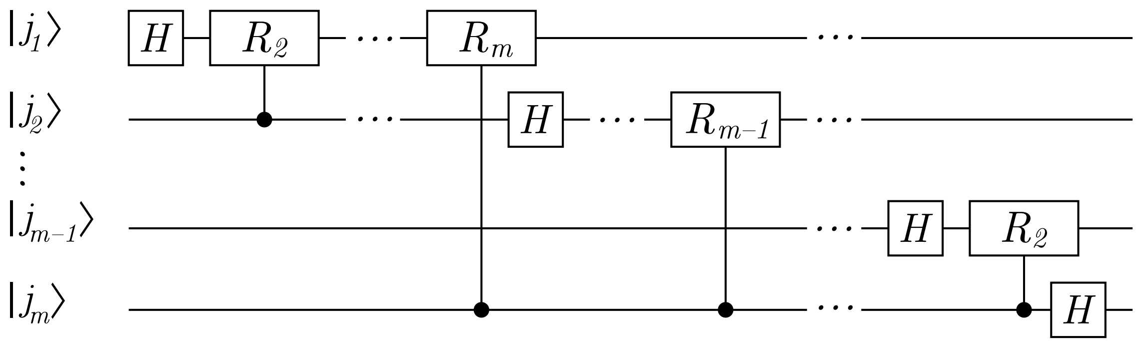

Figure 4: *A quantum circuit for efficient quantum Fourier transform*

Applying H to $|j_1\rangle$, the first qubit of $|j\rangle = |j_1\rangle |j_2\rangle \ldots |j_m\rangle$, we get

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{-2\pi i(0.j_1)} |1\rangle) \otimes |j_2\rangle \ldots |j_m\rangle$$

Applying the controlled R_2 to this, we get

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{-2\pi i(0.j_1 j_2)} |1\rangle) \otimes |j_2\rangle \ldots |j_m\rangle$$

Keep going through the circuit until we get

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{-2\pi i(0.j_1 j_2 \ldots j_m)} |1\rangle) \otimes |j_2\rangle \ldots |j_m\rangle$$

for the first qubit. For the second qubit, we do the same thing and get

$$\frac{1}{\sqrt{2^2}}(|0\rangle + e^{-2\pi i(0.j_1 j_2 \ldots j_m)} |1\rangle) \otimes (|0\rangle + e^{-2\pi i(0.j_2 \ldots j_m)} |1\rangle) \otimes |j_3\rangle \ldots |j_m\rangle$$

and so on until the m^{th} qubit, after which we have

$$\frac{1}{\sqrt{2^m}} \left(|0\rangle + e^{-2\pi i(0.j_1 j_2 \ldots j_m)} |1\rangle\right) \otimes \left(|0\rangle + e^{-2\pi i(0.j_2 \ldots j_m)} |1\rangle\right) \otimes \ldots \otimes \left(|0\rangle + e^{-2\pi i(0.j_m)} |1\rangle\right)$$

which is almost what we wanted, except in the reverse order! To remedy this, we add $\lfloor m/2 \rfloor$ swap gates at the end of the circuit.

We can count the number of gates in the circuit. From bottom up, we have $1 + 2 + \ldots + m = \sum_{j=1}^{m} = m(m+1)/2$ Hadamard gates and controlled rotations gates. In addition, we have the $\lfloor m/2 \rfloor$ swap gates we put in at the end. Hence, the circuit is polynomial in m. This is an exponential speed up over the classical FFT! However, since the QFT acts on quantum states, we can't just apply the QFT to data sets as with the DFT. Moreover, we could only construct this when we had $N = 2^m$. In general, we can construct QFT in polynomial time only if N is smooth. There are ways to get around this[7], but we won't cover them.

5 Quantum Part: Order Finding Algorithm

The results of section 3 reduced the problem of factoring n to finding the order r of an element a in $(\mathbb{Z}/n\mathbb{Z})^\times$. Shor provides such an algorithm by employing the QFT. Although there's an equivalent way of doing this using a quantum circuit via phase estimation[5], we will be following Shor's approach[11]. Readers interested in the alternative approach can consult our reference for the phase estimation algorithm[2].

For the following section, we will assume that n is an composite odd integer which is not a power of prime (the algorithm fails otherwise). If n is even, we can just factor out all the powers of 2 until we get an odd integer, then run the algorithm on the resulting integer. We can test whether n is a prime efficiently using classical primality tests such as the Miller-Rabin test[6],[9] and the AKS test[1]. We can also test if n is a power of prime efficiently by taking the k^{th} root of n until $n^{1/k} < 2$.

Given n, we choose $N = 2^m$ such that $n^2 \leq N < 2n^2$ (i.e. choose the unique power of 2 in that range). We will be working with two *registers* (two arrays of qubits). The first one will hold a number $x \pmod N$, the second one will hold a number mod n. Each of them holds m qubits. At first, the registers are

$$|0\rangle \otimes |0\rangle$$

We put the first register in the uniform superposition of numbers $x \pmod N$ by using the QFT, $|0\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \zeta^{-0x} |x\rangle$. We get

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

Now suppose $f(x) = a^x \pmod N$. Note that the period of f is the same as $\operatorname{ord}(a) = r$. Given some base a, Can we compute $f(x)$ efficiently? The answer is yes, we can just exponentiate by repeated squaring!

Example 17. Let $a = 2$, $N = 15$. Suppose we want to compute $f(10)$. Naïvely, this requires 10 multiplications. However, we can apply repeated squaring

$$2^{10} = (2^2)^5$$
$$(2^2)^5 = (2(2^2)^2)^2$$

and thus

$$2^2 = 2 \cdot 2$$
$$2^4 = 2^2 \cdot 2^2$$
$$2^5 = 2^4 \cdot 2$$
$$2^{10} = 2^5 \cdot 2^5$$

which requires 4 multiplications instead of 10. Notice that if we were calculating $f(20)$, we would only need 5 instead of 20 multiplications.

We need to apply f to the contents of the first register and store the result of $f(x)$ in the second register. To do so, we can construct f as a quantum function[11]. It turns out that this is the bottleneck of the algorithm since implementing f on a quantum computer requires a lot of quantum gates. Still, Shor's algorithm is much faster than factoring on a classical computer. We have then

[5] I like this way much better, however, I couldn't include it in the main text due to time constraints.

$$\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle\otimes|f(x)\rangle \tag{32}$$

Apply the QFT to the first register, we get

$$\begin{aligned}\frac{F_N\otimes I}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle\otimes|f(x)\rangle &= \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}(F_N|x\rangle)\otimes|f(x)\rangle \\ &= \frac{1}{N}\sum_{x,y=0}^{N-1}\zeta^{-xy}|y\rangle\otimes|f(x)\rangle\end{aligned} \tag{33}$$

We perform a measurement and compute the probability that we get a particular state $|y\rangle|f(k)\rangle$, where $0\le k<r$. Summing over all the possibilities,

$$\left|\frac{1}{N}\langle y|\langle f(k)|\sum_{x,y=0}^{N-1}\zeta^{-xy}|y\rangle|f(x)\rangle\right|^2 = \left|\frac{1}{N}\sum_{x:f(x)\equiv f(k)}\zeta^{-xy}\right|^2 \tag{34}$$

The sum is over all $x, 0\le x<N$ such that $f(x)\equiv f(k) \pmod n$ i.e. $a^x\equiv a^k \pmod n$. Since $\operatorname{ord} a = r$, this is equivalent to summing over all x such that $x = k \mod r$. Writing $x = br+k$, the probability is then

$$\left|\frac{1}{N}\sum_{b=0}^{\lfloor (N-k-1)/r\rfloor}\zeta^{-(br+k)y}\right|^2 = \left|\frac{1}{N}\sum_{b=0}^{\lfloor (N-k-1)/r\rfloor}\zeta^{-bry}\right|^2 \tag{35}$$

since $|\zeta^{-ky}|^2=1$. Moreover, since $\zeta^{-bry+N}=\zeta^{-bry}$, we can reduce ry modulo n. Replace ry by $\{ry\}$, where $N/2\le\{ry\}\le N/2$. We can approximate the sum inside by an integral. So

$$\frac{1}{N}\sum_{b=0}^{\lfloor (N-k-1)/r\rfloor}\zeta^{-b\{ry\}} \simeq \frac{1}{N}\int_{b=0}^{\lfloor (N-k-1)/r\rfloor} e^{-2\pi i b\{ry\}/N}\,db + \mathcal{O}\left(\frac{1}{N}\right) \tag{36}$$

Let $u = rb/N, du = dbr/N$, we get

$$\frac{1}{r}\int_{u=0}^{\frac{r}{N}\left\lfloor\frac{(N-k-1)}{r}\right\rfloor} e^{-2\pi i u\{ry\}/r}\,du + \mathcal{O}\left(\frac{1}{N}\right) \tag{37}$$

Now since $k<r$, approximating the upper bound of the integral by 1 will only give us an error of $\mathcal{O}(\frac{1}{N})$ and so we get

$$\frac{1}{r}\int_{u=0}^{1} e^{-2\pi i u\{ry\}/r}\,du + \mathcal{O}\left(\frac{1}{N}\right) \tag{38}$$

Now suppose that $-\frac{1}{2}\le\frac{\{ry\}}{r}\le\frac{1}{2}$. We can integrate the function and see that the integral is minimized when $\frac{\{ry\}}{r}=\pm\frac{1}{2}$. The integral will evaluate to $\frac{2}{\pi r}$ if this were the case. However, this happens if there exists a constant d such that

$$-\frac{r}{2}\le ry-dN\le\frac{r}{2}\Leftrightarrow\left|\frac{y}{N}-\frac{d}{r}\right|\le\frac{1}{2N} \tag{39}$$

This looks familiar! It is in fact the error bound for the best approximation of $\frac{y}{N}$. That is, we want to find a best approximation of $\frac{d}{r}$ such such $r < n$. There is at most one such fraction since $N > n^2$. We can compute $\frac{d}{r}$ by computing the continued fraction of $\frac{y}{N}$ and truncate where necessary. If $\frac{d}{r}$ is in its lowest terms and $\gcd(d, r) = 1$, then we get r and can use it for the rest of the algorithm, which is done classically. If not, the algorithm fails.

There are $\phi(r)$ numbers relatively prime to r. Moreover, there are r values for a^k since $\mathrm{ord}(a) = r$. Hence, there are $r\phi(r)$ states which allows us to obtain r, and each state occurs with probability of at least $\frac{1}{3r^2}$. Therefore, we will get r with probability at least $\frac{\phi(r)}{3r}$. Since $\frac{\phi(r)}{r} > \frac{C}{\log\log r}$ for some constant C[4], we can repeat the algorithm $\mathcal{O}(\log\log r)$ times and almost guarantee that we find r.

6 Discrete Logarithms

Just as the RSA cryptosystem is based off the presumed difficulty of factoring a number classically, the Diffie-Hellman key exchange protocol is based off the presumed difficulty of computing the discrete logarithms efficiently. We will consider how to apply the ideas we developed in the previous sections to compute discrete logarithms. We will only treat the special case when $p - 1$ is smooth (i.e. its prime factors are all less than $\log^C p$ for fixed C) and refer the reader to Shor's paper for the general case[11]. The general case is a bit more technical but contains the same ideas.

Let $x \equiv g^r \pmod{p}$. We want to compute r given x, g, p. Note that $f(a, b) = g^0 = 1$ only if $a \equiv -rb \pmod{p-1}$. We start out with three registers all initialized to $|0\rangle$.

$$|0\rangle \otimes |0\rangle \otimes |0\rangle$$

We can apply F_{p-1} to the first two registers (i.e. apply $F_{p-1} \otimes F_{p-1} \otimes I$) to obtain

$$\frac{1}{p-1} \sum_{a,b=0}^{p-2} |a\rangle |b\rangle |0\rangle \tag{40}$$

Now suppose $f(a, b) = x^a g^{-b} \pmod{p}$. We can compute $f(a, b)$ efficiently by repeated squaring as before. Put the result in register 3 and we get

$$\frac{1}{p-1} \sum_{a,b=0}^{p-2} |a\rangle |b\rangle |x^a g^{-b}\rangle \tag{41}$$

Apply the QFT to the first two registers again, we get

$$|\psi\rangle = \frac{1}{(p-1)^2} \sum_{a,b,c,d=0}^{p-2} \zeta^{-ac} \zeta^{-bd} |c\rangle |d\rangle |x^a g^{-b}\rangle \tag{42}$$

where $\zeta = \exp\left(\frac{2\pi i}{p-1}\right)$

We perform the measurement and compute the probabilities that we get the a particular state

$|c\rangle\,|d\rangle\,|g^k\rangle$.

$$\left|\langle c|\,\langle d|\,\langle g^k|\psi\rangle\right|^2 = \left|\frac{1}{(p-1)^2}\sum_{a,b:a-rk=b}\zeta^{-(ac+bd)}\right|^2 \tag{43}$$

$$= \left|\frac{1}{(p-1)^2}\sum_{b=0}^{p-2}\zeta^{-((b+rk)c+bd)}\right|^2 \tag{44}$$

$$= \left|\frac{1}{(p-1)^2}\sum_{b=0}^{p-2}\zeta^{-(brc+bd)}\right|^2 \tag{45}$$

if $d + rc \not\equiv 0 \pmod{p-1}$, the sum is over all the $(p-1)^{\text{st}}$ roots of unity and hence 0. If $d + rc \equiv 0 \pmod{p-1}$, we get

$$\left|\frac{1}{(p-1)^2}\sum_{b=0}^{p-2}\zeta^{-(brc+bd)}\right|^2 = \left|\frac{1}{(p-1)^2}\sum_{b=0}^{p-2}1\right|^2 \tag{46}$$

$$= \frac{1}{(p-1)^2} \tag{47}$$

Hence, we only need to measure pairs (c, d) such that $rc + d \equiv 0 \pmod{p-1}$. Then, we can then recover r by computing $r \equiv -c^{-1}d \pmod{p-1}$. The algorithm will fail unless a and $p-1$ are relatively prime. The probability of success is, as with factoring numbers, $\frac{\phi(p-1)}{p-1} > \frac{C}{\log\log p-1}$.

7 Conclusion

We showed that Shor's algorithm allows us to factor numbers much more quickly than classical algorithms. It runs in $\mathcal{O}(\log^3 n)$ where n is the number we are trying to factor. With Shor's algorithm, factoring becomes a BQP problem since we have a bounded probability of failure on each run of the algorithm. By applying the algorithm multiple times, we can be more and more sure to factor n. The main bottleneck of the algorithm is implementing modular exponentiation using a quantum circuit. The algorithm employs the QFT, which is basically a quantum version of the FFT, in a crucial way. In addition to factoring numbers, Shor's ideas also allowed us to compute discrete logarithms in polynomial time. Shor's algorithm is a real quantum algorithm which allows us to test the abilities of quantum computers. So far, we've only been able to factor numbers up to 21 using the algorithm since it requires coherent control of many qubits.

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. "PRIMES is in P". In: *Ann. of Math* 2 (2002), pp. 781–793.

[2] Donny Cheung. "Using Generalized Quantum Fourier Transforms in Quantum Phase Estimation Algorithms". MA thesis. Waterloo, Ontario, Canada: University of Waterloo, 2003.

[3] R. Cleve et al. "Quantum algorithms revisited". In: *Proceedings of The Royal Society* (1998).

[4] G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers (Sixth ed.)* Oxford University Press, 2008.

[5] A. Ignjatović. "Polynomial Multiplication and The Fast Fourier Transform (FFT)". 2013. URL: http://www.cse.unsw.edu.au/~cs3121/Lectures/Topic3.pdf.

[6] Gary L. Miller. "Riemann's Hypothesis and Tests for Primality". In: *Journal of Computer and System Sciences* 13.3 (1976), pp. 300–317.

[7] Michele Mosca and Christof Zalka. "Exact quantum Fourier transforms and discrete logarithm algorithms". In: *Symposium on Foundations of Computer Science* (1994). arXiv:quant-ph/0301093 [quant-ph].

[8] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[9] Michael O Rabin. "Probabilistic algorithm for testing primality". In: *Journal of Number Theory* 12.1 (1980), pp. 128 –138. ISSN: 0022-314X. DOI: 10.1016/0022-314X(80)90084-0. URL: http://www.sciencedirect.com/science/article/pii/0022314X80900840.

[10] Peter W. Shor. "Algorithm for Quantum Computation: Discrete Logarithms and Factoring". In: *Symposium on Foundations of Computer Science* (1994).

[11] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM J.Sci.Statist.Comput.* (1997).

[12] John A Smolin, Graeme Smith, and Alex Vargo. "Pretending to factor large numbers on a quantum computer". In: *Pre-Print* (2013). arXiv:1301.7007 [quant-ph].

[13] Lieven M.K Vandersypen et al. "Experimental Realization of Shor's Quantum Factoring Algorithm Using Nuclear Magnetic Resonance". In: *Letters to Nature* (2001).

[14] Ruye Wang. *Convolution theorem for Discrete Periodic Signal*. Apr. 2013. URL: http://fourier.eng.hmc.edu/e180/e101.1/e101/Fourier_Transform_D/node9.html.