Exploring Shor's Algorithm

How quantum algorithms can be used to efficiently factorize integers

Ahmed Akhtar
Prof. Shivaji Sondhi

January 27, 2016

Abstract

This paper discusses Shor's algorithm, an efficient factoring algorithm for quantum computers. We start with an overview of the mathematics necessary to understand the reduction of factoring to order-finding and then present the proof of this reduction. We then cover the Quantum Fourier Transform, Phase Estimation, and Order-finding algorithms for a quantum device and how they allow for efficient factoring. This work is written at a level comfortable for undergraduate students of physics.

Introduction

Peter Shor's landmark 1994 paper "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer", for which he earned the prestigious 1998 Nevanlinna prize [1], was written at a time when quantum computing was in its infancy. It gave two efficient[1] quantum algorithms, one for factoring and one for discrete logarithms, for which there still exist no efficient solution on classical computers. The fastest known classical algorithm for factoring takes time exponential in the input size: the number of bits needed to represent N.[2] Written for undergraduate students of physics, this paper will give an explanation of the former: Shor's Algorithm for factoring. It is a fundamentally probabilistic algorithm that, given an integer N, can give a non-trivial factor of N in polynomial time with high probability.

The factoring problem can be stated as follows. Given a composite positive integer N, find a non-trivial factor (neither 1 or N) of N. No efficient solution for arbitrary N has been found that can be run on a classical (or digital) computer. Large numbers which are the product of a few, large primes are hardest to factorize classically. The difficulty of this problem has inspired encryption methods, such as the RSA (Rivest, Shamir, and Adleman) public key cryptosystem, that can only be decrypted by finding the divisors of enormous semi-prime integers.

Apart from its applications in decryption and prime decomposition, Shor's algorithm is fascinating because it challenges some fundamental principles of computer science. Your everyday, digital computer is believed to be a universal computing device–that is, any other physical computing device can be simulated efficiently (in the resources used by the device) by a digital computer[3]. However, the fact that a computer based in quantum mechanics can solve certain problems faster than a digital computer raises the question of whether quantum computers are in fact the more powerful, more universal model of computation. On the other hand, if the belief is true, then there should exist a way to simulate a quantum computer efficiently, which would then provide us with an efficient algorithm for factoring on a classical computer [6].

Shor's algorithm is the first concrete example that quantum mechanics has a vast computational power. Since then, quantum computation has come a long way. New quantum algorithms for factoring have been developed. As of today, the largest semi-prime integer factored on a quantum computer is $56,153$ [7]. Will our smart-phones, tablets and laptops ever utilize quantum computation? However uncertain the future of technology is, it is undeniably promising.

Shor's Algorithm: An Overview

Shor's algorithm takes an L-bit integer N as input and outputs a single non-trivial factor of N in $O((\lg N)^3) = O(L^3)$ operations or gates. It is an efficient solution to the factoring problem, because the number of operations it requires to produce a factor is polynomial in L. One procedure is summarized below, and the details and proofs of why this procedure works will follow after [4, pp 233-234].

1. If N is even, then return 2. Checking if a binary integer is even amounts to checking if the least significant digit is a 0, so this operation takes $O(1)$ operations. If N isn't even, proceed to the next step.

2. Check if N is the power of some integer greater than or equal to one. In other words, check if $N = a^b$ for some $a \geq 1, b \geq 2$. This step is $O(L^3)$. If this step succeeds in finding a, b, then

[1]For what "efficient" means, see the appendix.

[2]The *General Number Field Sieve* is the current fastest classical algorithm for factoring large integers. It is exponential in cube root of the number of bits needed. [2]

[3]The Quantitative Church-Turing Thesis (Vergis et. al [1986]) states this of a Turing machine which can be efficiently simulated by a digital computer.

return a. Otherwise proceed to the next step.[4]

3. Randomly choose a integer x between 1 and $N-1$. If $\gcd(x, N) > 1$, return the common factor, otherwise move onto the next step. As we will show, computing the $\gcd(x, N)$ is $O(L^3)$.

4. If you've made it to this step, then you have found a number x co-prime to N. Thus we can find the order r modulo N of x. With probability at least $\frac{1}{2}$, r is even and $x^{\frac{r}{2}} \not\equiv -1 \mod N$. Thus, $x^{r/2}$ is a non-trivial solution to $y^2 \equiv 1 \mod N$. This guarantees a non-trivial factor from either $\gcd(x^{r/2}-1, N)$ or $\gcd(x^{r/2}+1, N)$. We check both results to see if they're factors of N. If r was odd or $x^{\frac{r}{2}} \equiv -1 \mod N$ then repeat step 3 [4, p 629].

If you did not follow the steps of the procedure above, don't be disconcerted. The goal of this paper is to give a complete explication of the physics and math behind this procedure. We will discuss what a quantum computer is, the circuit model of quantum computation, the various parts of Shor's algorithm that allow you to complete the steps above, and the number theory behind why this procedure works. The remarkable thing about Shor's algorithm is the final step. In fact, classically we can do all of the steps above efficiently except step 4. The defining advantage of Shor's Algorithm is the ability to find the order modulo N of an integer. Only on a quantum computer can this be done efficiently.

Before moving on, suppose you wanted to know the complete prime factorization of some positive composite integer N. Taking for granted that Shor's algorithm finds a non-trivial factor in $O(L^3)$, we can determine the complete prime factorization of a composite integer as follows.

First, let's define a function $f(n)$ that takes as input a positive integer n, and returns two non-trivial factors of n (a, b). If n is prime, it prints n and returns $(1, n)$. Let the recursive function $g(N)$ be defined as the following:

```
g(N) {
    $f(N) \to (a, b)$
    IF ($a$ or $b$ is 1)
        STOP
    ELSE
        $g(a)$
        $g(b)$
}
```

To summarize, we recursively factorize the factors of N until we end up with a complete prime factorization. We know f is computable efficiently (up until we've found a prime factor, which can be assumed probabilistically when Shor's algorithm fails to find a factor in a reasonable amount of time) and we know the number of primes in the decomposition is less than or equal to $\lg N$, so a trivial upper limit on the number of times the f will be called is $(\lg N)^2$ resulting in an algorithm that gives the complete prime factorization with high probability in $O(L^5)$.

1 Number Theory, Group Theory, and the Reduction of Factoring to Order Finding

The goal of this section is to convince you that the problem of factoring reduces to the problem of order-finding. That is to say, if we can find the order of an element modulo N efficiently, we can factor N efficiently. In the next section, we will see how Shor's Algorithm for quantum computers can compute order. Combining this with the classical algorithms in step 1, 2, and 3 above, we end

[4]This step is for the case that $N = p^m$ where p is an odd prime.

up with a complete and efficient factoring algorithm. To understand the reduction, some math is needed.

1.1 Elementary Number Theory and the Greatest Common Divisor

The most important theorem in elementary number theory (at least for the purposes of this paper) is the division algorithm. It states that for any pair of positive integers n, m, there exists a unique pair of integers q, r s.t.

$$n = mq + r \qquad 0 \leq r \leq m - 1 \tag{1}$$

In the equation above, we have divided n by m: n is the dividend, m is the divisor, q is the quotient, and r is the remainder. We say that if $r = 0$, then m divides n. This is written symbolically as $m|n$.

$$m|n \iff \exists k \in \mathbb{Z}|\, mk = n \tag{2}$$

Fact 1: If $c|a$ and $c|b$, then c divides any linear combination of a and b.

Proof: Let $xa+yb = d$ be some linear combination of a and b. Since $c|a$ and $c|b$, $k_1c = a$ and $k_2c = b$. Plugging this into the linear combination, we get $d = xk_1c+yk_2c = (xk_1+yk_2)c = k_3c$ where k_3 is some integer. From the preceding definition, we see that this means that $c|d$. $\square$

We can define the greatest common divisor of two positive integers, written symbolically $\gcd(a, b)$, as the greatest positive integer that divides both a and b.

Computing the greatest common divisor is a key part of Shor's algorithm. When we choose a random x between 1 and $N - 1$ (step 3 of the algorithm), computing the $\gcd(x, N)$ will either give us a common non-trivial factor of x and N, in which case our job is done, or it will equal 1, telling us that x and N are co-prime. As we will see, if x and N are co-prime, then x has an order modulo N. The first important fact that we should prove is that one can find the $\gcd(a, b)$ in time polynomial in the number of bits needed to represent a and b. To do this, we will have to prove some facts about the gcd first.

A priceless fact about the greatest common divisor is that the smallest positive linear combination of two integers is the same as their gcd.

Theorem 1: If a and b are integers and $s = ax+by$ is their smallest positive linear combination[5], then $\gcd(a, b) = s$

Proof: We will prove this by showing that $s \leq \gcd(a, b)$ and $\gcd(a, b) \leq s$.

First, we will show $\gcd(a, b)|s$, thus $\gcd(a, b) \leq s$. $\gcd(a, b)$ is the greatest *common* factor of a and b, so $\gcd(a, b)$ divides both a and b. By Fact 1, $\gcd(a, b)$ will divide any linear combination of a and b, so $\gcd(a, b)|s$.

Second, I claim that s is a common factor of a and b, so $s \leq \gcd(a, b)$. I will prove $s|a$. Then by symmetry, $s|b$. $\square$

Claim: $s|a$

We will do a proof by contradiction. Suppose $a = sk + r$ where $1 \leq r \leq s - 1$. Then $r = a - sk = a - (ax + by)k = (1 - kx)a - (bk)y$. Thus, we see that our assumption that s does not divide a implies the existence of a positive linear combination of a and b less than s, which is impossible. $\blacksquare$ [4, p 627].

[5]Existence of s: Consider the set of positive linear combinations of a and b. The set is non-empty. From the well-ordering principle, a smallest element of that set exists.

It follows from this theorem and Fact 1 that any common factor of a and b will divide $\gcd(a, b)$, since $\gcd(a, b)$ is a linear combination of a and b.

We have one more statement left to prove before we can show the efficiency of computing the gcd.

Lemma 1: Suppose we divide a by b (a and b are positive integers) and the remainder r is non-zero. Then $\gcd(a, b) = \gcd(b, r)$.

Proof: We will show each side of the equality divides the other.

From the division algorithm, we get $r = a - bq$. Therefore, r is a linear combination of a and b. Thus $\gcd(a, b)|r$ (by F1). By definition, $\gcd(a, b)|b$. Therefore, $\gcd(a, b)|\gcd(b, r)$ (by Theorem 1 and Fact 1).

We know that $\gcd(b, r)|b$. From the division algorithm, $a = bq + r$, thus a is a linear combination of b and r, therefore $\gcd(b, r)|a$ (by F1). Thus, $\gcd(b, r)|\gcd(a, b)$. $\square$ [4, 628]

We now have everything we need to understand Euclid's efficient algorithm for finding the gcd of two integers. Suppose our two positive integers are a and b, and W.L.O.G. $a \geq b$. Suppose we compute q and r from the division algorithm by dividing a by b. If $r = 0$, then $\gcd(a, b) = b$ and we are done[6]. Else, $r = r_1 \neq 0$, so $\gcd(a, b) = \gcd(b, r_1)$ (by Lemma 1). We now repeat the process on b and r_1, this time dividing b by r_1. If the remainder r_2 is 0, then $\gcd(b, r_1) = r_1 = \gcd(a, b)$. Otherwise we have $\gcd(a, b) = \gcd(b, r_1) = \gcd(r_1, r_2)$. Repeating the process, we see that if the first zero remainder is r_{m+1}, then r_m is $\gcd(a, b)$ [4, p 628].

As an example, let's compute the $\gcd(1290, 635)$.

$$\begin{aligned} 1290 &= 2 \times 635 + 20 & r_1 &= 20 \\ 635 &= 31 \times 20 + 15 & r_2 &= 15 \\ 20 &= 1 \times 15 + 5 & r_3 &= 5 \\ 15 &= 3 \times \underline{5} + 0 & r_4 &= 0 \end{aligned}$$

With just four calculations of the division algorithm, we determined that $\gcd(1290, 635) = 5$.

The swiftness of the algorithm is due to the fact that the remainders approach zero quickly. We can put a bound on how fast the remainders approach zero by noticing that $r_{i+2} \leq r_i/2$. To understand this, first consider the case where $r_{i+1} \leq r_i/2$. In this case, the bound holds since $r_{i+2} \leq r_{i+1}$ [7]. Now consider the case where $r_{i+1} > r_i/2$. Dividing r_i by r_{i+1}, we get $r_i = 1 \times r_{i+1} + r_{i+2}$. Thus, $r_{i+2} = r_i - r_{i+1} < r_i/2$.

Suppose that a is an L bit binary integer. Then b and all the remainders can be represented in L bits. Since the size of the remainders gets at least halved every other step, the number of divisions we need to do is $O(2 \lg a)$ which is also $O(L)$. Each division takes $O(L^2)$ arithmetical operations, so Euclid's efficient algorithm for finding the gcd is $O(L^3)$ [4, p 629].

1.2 Modular Arithmetic, Co-primality, and Order

Two positive integers are co-prime if they don't share any common factors greater than 1. In other words, a and b are co-prime iff $\gcd(a, b) = 1$.

Two integers a and b are equivalent modulo N if they have the same remainder r when divided by N. Likewise, a and b are equivalent modulo N if their difference is a multiple of N. In fact, we can partition the set of integers $\mathbb{Z}$ into N distinct classes where the members of each class are equivalent to each other modulo N[8].

[6] $b|b$ and $b|a$, and the common factor can't be bigger than b

[7] r_{i+2} is remainder from dividing r_i by r_{i+1}, so $0 \leq r_{i+2} \leq r_{i+1}$ from the division algorithm

[8] The three sets of integers that follow are referred to as the equivalence classes modulo 3.

$$a \equiv b \mod N \iff \exists k \in \mathbb{Z} \,|\, a - b = kN \tag{3}$$

$$\begin{gathered}
... - 2N \equiv -N \equiv 0 \equiv N \equiv 2N... \mod 4 \\
.... - 2N + 1 \equiv -N + 1 \equiv 1 \equiv N + 1 \equiv 2N + 1... \mod N \\
... \\
.... - 2N + (N-1) \equiv -N + (N-1) \equiv N - 1 \equiv N + (N-1) \equiv 2N + (N-1)... \mod N
\end{gathered}$$

$$\begin{gathered}
... - 6 \equiv -3 \equiv 0 \equiv 3 \equiv 6... \mod 3 \\
.... - 5 \equiv -2 \equiv 1 \equiv 4 \equiv 7... \mod 3 \\
.... - 4 \equiv -1 \equiv 2 \equiv 5 \equiv 8... \mod 3
\end{gathered}$$

Equivalence plays the role of equality in modular arithmetic. In fact, it supersedes equality. Two integers are equivalent if equal, but not always equal if equivalent. When thinking in terms of remainders, we can see easily that equivalence is symmetric, transitive, and reflexive. It is symmetric, because if a has the same remainder as b, then b has the same remainder as a. It is reflexive because a has the same remainder as itself. It is transitive because if a has the same remainder as b, and b has the same remainder as c, then a has the same remainder as c.

We can also substitute any integer in a modular equation with some other member of its equivalence class. Suppose that $ab \equiv c \mod N$ and that $a \equiv d \mod N$. Then $db = (a - kN)b = ab - kbN$. Thus, $db - ab = (kb)N$, and so $ab \equiv db \equiv c \mod N$. Likewise, if $a + b \equiv c \mod N$ and $a \equiv d \mod N$, then $a + b = kN + d + b$. Rearranging, $(a + b) - (d + b) = kN$, thus $a + b \equiv d + b \equiv c \mod N$.

The order r of an integer x modulo N is the smallest positive integer such that $x^r \equiv 1 \mod N$. We write this symbolically as $ord(x, N) = r$ or $|x| = r$. When does x have an order modulo N? It is clear that some x do not have an order for a given N, e.g. $x = 2, N = 4$.

$$|x| = r \iff r \text{ is the smallest positive integer s.t. } x^r \equiv 1 \mod N \tag{4}$$

It turns out that the answer to the previous question is closely tied to another question: When does an integer have a multiplicative inverse modulo N? Given some integer a, we want to know if $\exists a^{-1}$ s.t. $aa^{-1} \equiv 1 \mod N$. For example, if $a = 2$ and $N = 4$, then there is no inverse. This is because if we multiply 2 by an odd number, we will get 2, and if we multiply by an even number we will get 0.

Theorem 2: An integer x has an inverse modulo N iff it is co-prime with N.

Proof: If x is co-prime with N, we know $\gcd(x, N) = 1$ or equivalently $ax + bN = 1$ for some integers a, b. Rearranging, $ax - 1 = bN$, and so $ax \equiv 1 \mod N$. The inverse of x is a (and vice versa).

Now suppose that x has an inverse modulo N. Then $x^{-1}x \equiv 1 \mod N$, and therefore $x^{-1}x - 1 = cN$. Rearranging, we see that $x^{-1}x - cN = 1$. The left hand side is a linear combination of x, N. It must be the smallest positive linear combination of x, N since 1 is smallest positive integer. From Theorem 1, $\gcd(x, N) = 1$. $\square$ [4, p 627]

It follows that inverses are unique modulo N. Suppose b and b' are two inverses of a. Then $ab \equiv 1 \equiv ab' \mod N$. It follows from transitivity that $ab \equiv ab' \mod N$. Multiplying both sides by

a^{-1} (doesn't matter which inverse), we see $b \equiv b' \mod N$. Similarly, if $a^{-1} \equiv c \mod N$, then by substitution c acts also as an inverse for a.

We now have everything we need to answer the first question: When does an integer have an order modulo N?

Theorem 3: x has an order r modulo N iff x is co-prime with N.

Proof: If x has order r, then $x^r \equiv 1 \equiv x(x^{r-1}) \mod N$, so x has an inverse modulo N. By Theorem 2, x must be co-prime to N.

Suppose x is co-prime with N. Consider the powers of x modulo N: $x, x^2, x^3 ... \mod N$. Since there are only N distinct integers modulo N, there have to exist two equivalent powers of x. Suppose $x^n \equiv x^m \mod N$, with $n > m$. Multiplying both sides by x^{-1} m-times, we see $x^{n-m} \equiv 1 \mod N$. Thus, the set $\{v \,|\, v \in \mathbb{Z}^+, x^v \equiv 1 \mod N\}$ is non-empty and has a smallest element r by the well-ordering principle. It follows that there are r distinct powers of x modulo N: $x, x^2, x^3 ... x^{r-1}, 1$. $\square$

Suppose that $|x| = r$ modulo N. Then the following lemma holds:

Lemma 2: $x^n \equiv 1 \mod N \iff r|n$

Proof: If $r|n$, then n can be written $n = kr$. It follows $a^n \equiv a^{kr} \equiv a^{r^k} \equiv 1^k \equiv 1 \mod N$

If $x^n \equiv 1 \mod N$, it follows from the division algorithm that $x^{qr+r'} \equiv 1 \mod N$ where $0 \leq r' \leq r - 1$. $x^{qr+r'} \equiv x^{qr}x^{r'} \equiv x^{r'} \equiv 1 \mod N$. The remainder r' cannot be positive because then r would not be the order. Therefore, $r' = 0$. $\square$

The set of positive integers co-prime with N and less than N form a group under multiplication modulo N. This group is designated $\mathbb{Z}_N^*$. The size of this group is written as a function of N, $\varphi(N)$, and is referred to as Euler's totient function. We show that $\mathbb{Z}_N^*$ meets all the requirements of a group:

1. *Associativity:* Multiplication is associative by definition: $\forall a, b, c \in \mathbb{Z}_N^*$, $a(bc) \equiv (ab)c \mod N$ $\blacksquare$

2. *Closed under multiplication:* Let $a, b \in \mathbb{Z}_N^*$. We need to show their product is in the set. Their product, $ab \mod N$, is between 0 and $N-1$ so we know that it is less than N. From the division algorithm, $ab = (cN + 1)(dN + 1) = (cdN^2 + cN + dN) + 1 = kN + 1$. Rearranging, $ab - kN = 1$. From T1, $\gcd(ab, N) = 1$. $\blacksquare$

3. *Contains identity:* The set contains 1, since 1 is co-prime with N. 1 is the multiplicative identity. $\blacksquare$

4. *Closed under inverses:* Let $a \in \mathbb{Z}_N^*$. Each element in the set has an inverse, from T2 ($\exists a^{-1}$). From the same theorem, it is clear that a^{-1} must be co-prime with N since it also has an inverse. We can choose the inverse so that it is between 1 and $N - 1$. Thus $a^{-1} \in \mathbb{Z}_N^*$. $\square$

From Lagrange's theorem, it follows that the order r of an element x (written $ord(x)$ or $|x|$) divides $\varphi(N)$.

Fact 2: $ord(x)|\varphi(N)$

Proof: This is because the powers of x are a subgroup, and the size of the group is r, and according to Lagrange's theorem the size (also termed 'order') of the subgroup must divide the size of the group[9].

The totient function $\varphi(N)$ can be calculated using the prime decomposition of N. From the Fundamental Theorem of Arithmetic,

[9]A proof of Lagrange's theorem can be found in any introductory group theory text. See Pinter in the references for a lucid introductory text on Group Theory.

$$N = p_1^{\alpha_1} p_2^{\alpha_2} ... p_l^{\alpha_l} \quad \text{where the } p_i \text{ are prime}$$

First consider the case where $N = p$, where p is prime. Since every positive integer less the p is co-prime with p, $\varphi(p) = p - 1$. Now consider the case where $N = p^m$. The only positive integers less than N that are not co-prime with N are multiples of p: $p, 2p, 3p ... p^m - 3p, p^m - 2p$, and $(p_m - p) = (p^{m-1} - 1)p$. Thus, the totient function is $\varphi(p^m) = p^m - 1 - (p^{m-1} - 1) = p^{m-1}(p - 1)$.[4, p 631]

To consider the general case for N, we will first prove that the totient function factorizes if N is the product of co-prime integers a and b.

Lemma 3: $\varphi(ab) = \varphi(a)\varphi(b)$ if a and b are co-prime.

Proof: Every pair of numbers $(x_a,\, x_b)$ where x_a is between 1 and a and x_b is between 1 and b inclusive has a one-to-one correspondence mapping to an integer x between 1 and ab, by the Chinese Remainder Theorem[10] [4, p 631].

The x_a co-prime with a form a group, $\mathbb{Z}_a^*$. The x_b co-prime with b form a group, $\mathbb{Z}_b^*$. Their direct product $\mathbb{Z}_a^* \times \mathbb{Z}_b^*$ (the pairs $(x'_a,\, x'_b)$ where x'_a and x'_b are group elements from $\mathbb{Z}_a^*$ and $\mathbb{Z}_b^*$ respectively) is a group. Similarly, the x co-prime with ab are a group, $\mathbb{Z}_{ab}^*$. We want to show these groups have equal size by finding a one-to-one correspondence mapping between them.

We can use the function $f(x) = (x_a, x_b)$ used in the Chinese Remainder Theorem in the appendix but relabel it $f*$ to signify that it only acts on the sets in the groups. First we show that it maps $x \in \mathbb{Z}_{ab}^*$ to $\mathbb{Z}_a^* \times \mathbb{Z}_b^*$. x can be written as $dx = 1 - c(ab)$ by T2.

$$\begin{aligned}
x &= x_a b b^{-1} + x_b a a^{-1} - k(ab) \\
1 &= c(ab) + x_a d b b^{-1} + x_b d a a^{-1} - k d(ab) \\
1 &= (cb + x_b d a^{-1} + k d b)a + (d b b^{-1}) x_a \\
1 &= (ca + x_a d b^{-1} + k d a)b + (d a a^{-1}) x_b
\end{aligned}$$

We see that x_a is co-prime with a and x_b is co-prime with b if x is co-prime.[11] Thus, the function $f*$ maps from $\mathbb{Z}_{ab}^*$ to $\mathbb{Z}_a^* \times \mathbb{Z}_b^*$ injectively. We know it's injective because the function f is injective over the whole domain, so it must be over some subset as well.

To show it's surjective, we just need to show that if x_a and x_b are co-prime with a and b respectively, then x is co-prime with ab.

If x_a is co-prime with a, then $cx_a + da = 1$ for some c, d. We know that $x = x_a + ka$ for some k. Rearranging, we see that x is co-prime with a. By symmetry, x is co-prime with b. Since it is co-prime with a and b, it must be co-prime with ab[12]. So $f*$ must be surjective on $\mathbb{Z}_a^* \times \mathbb{Z}_b^*$.

Since $f*$ is one-to-one correspondence, the size of $\mathbb{Z}_{ab}^*$ must be the same size as $\mathbb{Z}_a^* \times \mathbb{Z}_b^* = \varphi(a)\varphi(b)$. Thus, $\varphi(ab) = \varphi(a)\varphi(b)$. $\square$

If N has a prime factorization $p_1^{\alpha_1} p_2^{\alpha_2} ... p_l^{\alpha_l}$, then Theorem 4 follows trivially from Lemma 3.

Theorem 4:

$$\varphi(N) = \prod_{i=1}^{l} p_i^{\alpha_i - 1}(p_i - 1)$$

We will need to utilize one final theorem before moving on to the reduction.

[10] See Appendix

[11] Alternatively, this tells us $\varphi(ab) \leq \varphi(a)\varphi(b)$. When we determine that $f*$ surjective in the second half of this proof, we show that $\varphi(ab) \geq \varphi(a)\varphi(b)$. Thus, $\varphi(ab) = \varphi(a)\varphi(b)$.

[12] Since x is co-prime with a, it shares no factors with a. Likewise, it shares no factors with b. It follows that it cannot share any factors with ab. Think of a and b in terms of their prime decompositions if this is confusing.

Theorem 5: $\mathbb{Z}^*_{p^\alpha}$, where p is an odd prime, is cyclic [4, 632].

If a group is cyclic, every element of it can be written as a positive power of some element in the group called the generator. In other words, $\exists g \in \mathbb{Z}^*_{p^\alpha}$ s.t. $\forall x \in \mathbb{Z}^*_{p^\alpha}, x \equiv g^k \mod p^\alpha$ for $1 \leq k \leq \varphi(p^\alpha)$. Every element in the set can be identified uniquely by k.

1.3 Factoring Reduces to Order Finding

We wish to find an algorithm that produces a non-trivial factor of N. We will show that given the ability to find the order r modulo N of a positive integer x efficiently, one can find a non-trivial factor for N efficiently.

Here is an overview of how the algorithm works: The algorithm begins by choosing an x randomly from 1 to $N-1$. There needs to be an efficient way to find a factor if x is co-prime with N, which happens with a high-probability if N is the product of a few large primes. Supposing x is co-prime with N, one can show that it is likely that $\gcd(x^{r/2}-1, N)$ or $\gcd(x^{r/2}+1, N)$ will return a non-trivial factor for N. Due to the efficiency of gcd and the high probability of success, choosing x just a few times guarantees a non-trivial factor for N in polynomial time!

We will need to prove some simple theorems to understand why this procedure works.

Lemma 4: Let $y \in \mathbb{Z}^*_N$, and suppose y is a non-trivial solution to $x^2 \equiv 1 \mod N$ (that is, $y \neq \pm 1 \mod N$), then $\gcd(y+1, N)$ or $\gcd(y-1, N)$ will give a non-trivial factor of N.

Proof: $y^2 \equiv 1 \mod N$ implies that $y^2 - 1 = kN = (y+1)(y-1)$. Since $N|(y+1)(y-1)$, it must share a common factor with $y+1$ or $y-1$ or both. The only troublesome case is if $y = N-1$, because then $\gcd(y+1, N) = N$ and $\gcd(y-1, N) = 1$ if N is odd[13]. By assumption, $y \neq N-1$, so the theorem works.[4, p 633]. $\square$

Lemma 5: Let p be an odd prime and let y be a randomly chosen element from $\mathbb{Z}^*_{p^\alpha}$. Then with probability exactly 1/2, the largest power of 2, 2^d, which divides $\varphi(p^\alpha)$ also divides r.

Proof: From Theorem 4, $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$ and so $d > 0$ since the size of the group is even. From Theorem 5, y can be written as g^k for some k, $0 \leq k \leq \varphi(p^\alpha) - 1$. k uniquely identifies each element in the group.

Consider the case where k is odd. $(g^k)^r \equiv 1 \equiv g^{kr} \mod p^\alpha$ by definition. Since $\varphi(p_\alpha)$ is the order of g, $\varphi(p^\alpha)|kr$ (by L2). k is odd, therefore it contains no factors of 2, and thus $2^d|r$.

Now consider the case where k is even. Noticing that $g^{\varphi(p^\alpha)k/2} \equiv 1 \equiv (g^k)^{\varphi(p^\alpha)/2} \mod p^\alpha$, we conclude that $r|\varphi(p^\alpha)/2$, so 2^d cannot divide r.

Since k has equal probability of being even or odd (picking y is like picking k and there are an even number of allowed k values), we see that there is a probability of 1/2 that 2^d divides r and 1/2 that it doesn't [4, 633]. $\square$.

Theorem 6: Let $N = p_1^{\alpha_1} p_2^{\alpha_2} ... p_m^{\alpha_m}$ be an odd composite positive integer. Suppose x is chosen randomly from $\mathbb{Z}^*_N$ and has order r. Then the probability that r is odd or $x^{r/2} \equiv -1 \mod N$ is less than or equal to $\frac{1}{2^{m-1}}$.

Proof: From the Chinese Remainder Theorem, picking a random x is the same as picking a random $x_j \in \mathbb{Z}^*_{p_j^{\alpha_j}}$ for all j. Define r_j as the order modulo $p_j^{\alpha_j}$ of x_j, 2^{d_j} as the largest power of 2 dividing r_j, and 2^d as the largest power of 2 that divides r.

By definition, $x^r \equiv 1 \mod N$. Thus $N|(x^r - 1)$, and so $p_j^{\alpha_j}|(x^r-1)$. Thus, $x^r \equiv 1 \mod p_j^{\alpha_j}$. From the CRT, this means that $x_j^r \equiv 1 \mod p_j^{\alpha_j}$, and so $r_j | r \, \forall j$ (by L2).

We first consider the case where r is odd. Since r is odd, each of the r_j must be odd, and so $d_j = 0$ for each of the j.

[13] If it was even, we already know a non-trivial factor: 2

Next, if r is even and $x^{r/2} \equiv -1 \mod N$, we see that $x_j^{r/2} \equiv -1 \mod p_j^{\alpha^j}$ $\forall j$. Thus, it cannot be that $r_j | (r/2)$ (otherwise the RHS would be 1). But since $r_j | r$, it follows $d_j = d$ $\forall j$.

From Lemma 5, $p(d_j = d') = 1/2$ where $2^{d'}$ is the largest power of 2 which divides $\varphi(p_j^{\alpha_j})$. The total probability must be 1, so $p(d_j \neq d') = 1/2$. It follows that the probability of d_j taking on any value is at most $\frac{1}{2}$.

Putting it all together, $p(x^{r/2} \equiv -1 \mod N) \leq p(d_1 = d_2 = ...d_m = d) \leq \frac{1}{2^m}$. The probability $d_j = 0$ is at most $1/2$, so $p(\text{r is odd}) \leq \frac{1}{2^m}$. Adding the probabilities, $p(x^{r/2} \equiv -1 \mod N \text{ or r is odd}) \leq \frac{1}{2^{m-1}}$ [4, p 634]. □

We see that Theorem 6 implies

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

Thus, in picking an element randomly from $\mathbb{Z}_{\mathbb{N}}^*$, we have a probability of at least 1/2 of getting a factor from that element (by substituting $x^{r/2}$ with y in Lemma 4).

This presents huge advantages when factoring large, semi-prime integers. Any classical algorithm would be exponential in the L, essentially checking every number between 1 and $\sqrt{N}$, but the chance of Shor's algorithm failing to find a factor of a semi-prime integer after being computed just 5 times is less than 5%, and each run is $O(L^3)$.

All of this hinges on the ability to calculate order efficiently, of course. On a classical computer, this is currently impossible to do efficiently. Using quantum mechanics, Shor was able to show that it can be done–but first you must have a quantum computer.

2 The Qubit and Quantum Circuits

2.1 Quantum equivalents of bits and logic gates

In classical computation, the smallest piece of information is a bit. It can be either 0 or 1. Everything on your computer is represented by sequences of zeros and ones which are interpreted by your machine to be numbers, text, addresses, etc.

In quantum computation, the smallest piece of information is the *qubit*. The defining feature of a quantum computer is that it manipulates information in the form of qubits. A quantum computer can also have classical components: there are certain problems that your average digital computer is very good at solving and a quantum computer can utilize a classical computer for such subproblems.

A qubit can be in a superposition of 0 and 1. One simple model for the qubit could be a spin half particle in the S_z basis. In this model, we can say that the qubit is in a 0 state when it is spin up and a 1 state when it is spin down. Thus, in the S_z basis

$$|0\rangle \to \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle \to \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Classically, a logic gate is a mapping from one set of bits to another. For example, the logic gate AND takes as input two bits and outputs 1 iff both inputs are 1. Another example is the $CNOT$ or controlled-NOT gate, which has one target bit, and one control bit. If the control bit is on, then the target bit is flipped, otherwise both bits pass through unchanged. The first output has the value of the control bit, and the second output has the value of the sum of the two input bits modulo two. Unlike the previous gate, $CNOT$ is an invertible, or *reversible*, logic gate. Given the output, you know the input. In other words, there isn't any information loss.

C	T	C	(C $\oplus$ T)
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Table 1: The truth table for CNOT. If U was the equivalent quantum-CNOT operator, it would act on control qubit $|c\rangle$ and target qubit $|t\rangle$ as $U|c\rangle|t\rangle = |c\rangle|c \oplus t\rangle$.

Landauer's principle says that for every bit of information loss, there is at least some fixed, finite energy dissipation proportional to the temperature[14][3]. Thus, logic gates that are reversible must be energy efficient because they have no information loss. We have a motivation to build our circuits out of logic gates that are reversible, because then over each operation, there will be no energy loss, and so overall the circuit will be energy efficient. A proof of Landauer's principle will not be provided as it is secondary to understanding Shor's algorithm.

The natural choice for operators, or logic gates, on a quantum computer are unitary operators. Unitary operators are diagonalizable because they are normal (by Spectral Decomposition), and therefore they are easier to implement. Also, unitary operators are by definition invertible. Any quantum circuit built out of unitary operators will be energy efficient.

Def: An operator U is unitary $\iff U^\dagger U = \mathbb{I}$

Besides being energy efficient, the Solovay-Kitaev theorem says that any unitary operator can be approximated to arbitrary accuracy through a finite number of fixed, universal two-level unitary operators [4, Ch 4]. What this means physically is that given some complicated quantum logic gate that takes n bits as input and outputs n bits, one can construct this quantum logic gate out of some small set of logic gates that only act on one or two qubits at a time. As long as one knows how to construct those universal two-level unitary operators, one can construct any unitary operator. The take away from this is that, in theory and in practice, unitary operators are the natural choice for logic gates on a qubit system. Again, the proof for this is nontrivial and only plays a secondary role in understanding Shor's Algorithm, so we will not cover it.[15] For the remainder of this paper, if we have a unitary operator, we will assume it to be implementable.

Sets of qubits combine how they would in regular quantum mechanics: via a tensor product. The computational basis of a set of t qubits is given by the tensor product of t qubit basis states.

$$|1\rangle_1 \otimes |0\rangle_2 \otimes |1\rangle_3 \otimes |1\rangle_4 \otimes ... |0\rangle_t \equiv |1011...0\rangle$$
$$|0\rangle_1 \otimes |0\rangle_2 \otimes |1\rangle_3 \otimes |0\rangle_4 \otimes ... |0\rangle_t \equiv |0010...0\rangle$$
$$|1\rangle_1 \otimes |1\rangle_2 \otimes |1\rangle_3 \otimes |1\rangle_4 \otimes ... |1\rangle_t \equiv |1111...1\rangle$$

A set of qubits (or a register) measured in the first state means the first qubit is 1, the second qubit is 0, the third qubit is 1, the fourth qubit is 1, and so on. There are 2^t basis states. We can think of each basis state as representing some binary integer. Thus, it is often useful to write the basis states in the form on the right. For example, the last basis state represents the integer $2^t - 1$.

One of the advantages of the qubit is its potential to store far more information than a classical bit. A classical bit can have only two distinct values, 0 or 1. Since the qubit is an arbitrary unit vector in a 2-d Hilbert space, there are an infinite number of distinct (but non-orthogonal) qubit states. Suppose we have a state $|\psi\rangle$. Then $|\psi\rangle$ can be written as

[14] A minimum energy dissipation of $kT \ln 2$ per bit of information lost.

[15] A proof of it can be found in chapter 4 of Nielsen and Chuang.

$$\begin{aligned} &r_\alpha e^{i\theta_\alpha} |0\rangle + r_\beta e^{i\theta_\beta} |1\rangle \\ &\equiv r_\alpha |0\rangle + \sqrt{1-r_\alpha^2} e^{i(\theta_\beta - \theta_\alpha)} |1\rangle \end{aligned}$$

In the second line, we normalize and ignore an overall phase. Replacing r_α with $\cos\frac{\theta}{2}$ and $\theta_\beta - \theta_\alpha$ with ϕ, we can see that it takes two independent real parameters to define a qubit. In fact, each qubit maps to a real, unit-length 3-vector $\vec{r}$ parameterized by polar angle θ and azimuthal angle ϕ. The set of these 3-vectors is called the Bloch sphere [4, p 174].

$$\cos\tfrac{\theta}{2} |0\rangle + \sin\tfrac{\theta}{2} e^{i\phi} |1\rangle \rightarrow (\sin\theta\cos\phi,\, \sin\theta\sin\phi,\, \cos\theta)$$

We see that when $|\psi\rangle$ is $|0\rangle$ or $|+z\rangle$, $\vec{r} \rightarrow (0,0,1)$, and when $|\psi\rangle$ is $|+x\rangle$, $\vec{r} \rightarrow (1,0,0)$ and when $|\psi\rangle$ is $|+y\rangle$, $\vec{r} \rightarrow (0,1,0)$.

Whether we really have access to the information stored in a qubit (in the form of its two real parameters) is a different question. Any sort of measurement destroys that information. However, quantum algorithms often take advantage of the fact that nature seems to keep track of this information for us.

2.2 The Quantum Fourier Transform

Shor's algorithm is successful because it can find the multiplicative order modulo N of an integer efficiently. One of the key quantum circuits behind his algorithm is the quantum Fourier transform.

The quantum Fourier transform on an orthonormal basis of N states labelled $|0\rangle, |1\rangle, |2\rangle, \ldots |N-1\rangle$ acts linearly on the basis vectors of that state as follows:

$$|j\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{2\pi i jk}{N}} |k\rangle \tag{5}$$

We can show that the operator is unitary because it preserves norms. Because the operator is unitary, we know that it can be implemented using our universal 2-level unitary operators. Thus, the quantum Fourier transform is a realizable quantum circuit.[16]

Proof: The quantum Fourier transform is unitary

It suffies to show $\langle m| U^\dagger U |n\rangle = \delta_{mn}$

$$\begin{aligned} \langle m| U^\dagger U |n\rangle &= \frac{1}{\sqrt{N}} \sum_{k'=0}^{N-1} e^{-\frac{2\pi i mk'}{N}} \langle k'| \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{2\pi i nk}{N}} |k\rangle \\ &= \frac{1}{N} \sum_{k'=0}^{N-1} \sum_{k=0}^{N-1} e^{\frac{2\pi i (n-m)k'}{N}} \langle k'|k\rangle \\ &= \frac{1}{N} \sum_{k'=0}^{N-1} e^{\frac{2\pi i (n-m)k'}{N}} \\ &= \frac{1}{N} N \delta_{mn} \\ &= \delta_{mn} \quad \square \end{aligned}$$

Consider an n-qubit state. The number of basis states of the n-qubit system is $2^n = N$. The integers that each basis state represents in binary go from 0 to $N-1$. Using the numbering of basis

[16]See part B of the Appendix if the fourth line of the proof seems sketchy

states defined in the quantum Fourier transform, $|0\rangle, |1\rangle, |2\rangle ... |j\rangle ... |N-1\rangle$, the state $|j\rangle$ refers to the qubit basis state $|j_1 j_2 j_3 ... j_n\rangle$ that has the binary representation of the integer j. For example, the state $|5\rangle$ in the quantum Fourier basis is the qubit basis state $|000...101\rangle$.

How does a n-qubit basis state transform after the quantum Fourier transform is applied to it? It turns out that the resulting state has a useful form in terms of the individual bits that make up the state. We will use this form of the quantum Fourier transform when deriving the Phase Estimation algorithm, which can be used for order finding.

$$\begin{aligned}
|j\rangle &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{2\pi i j k}{N}} |k\rangle \\
&= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi i j k}{2^n}} |k\rangle \\
&= \frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} \sum_{k_3=0}^{1} ... \sum_{k_n=0}^{1} e^{\frac{2\pi i j \left(\sum_{l=1}^{n} 2^{n-l} k_l\right)}{2^n}} |k_1 k_2 k_3 .... k_n\rangle
\end{aligned}$$

In the second equation we replaced N with 2^n. In the third equation, we are replacing k with its corresponding binary representation $k_1 k_2 ... k_n$, with k_i representing the i^{th} bit given by the i^{th} qubit in the tensor product space.

$$|j\rangle = \frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} ... \sum_{k_n=0}^{1} \left(e^{2\pi i j 2^{-1} k_1} |k_1\rangle\right) \otimes \left(e^{2\pi i j 2^{-2} k_2} |k_2\rangle\right) \otimes ... \left(e^{2\pi i j 2^{-n} k_n} |k_n\rangle\right)$$

We can now separate the individual sums over each bit of k, and represent the state as a product state. The first term of this new product state will represent what the first bit of the input, $|j_1\rangle$, goes to when the quantum Fourier transform is applied to $|j\rangle$.

$$= \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i j 2^{-1}} |1\rangle\right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i j 2^{-2}} |1\rangle\right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i j 2^{-3}} |1\rangle\right) \otimes ... \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i j 2^{-n}} |1\rangle\right)$$

Thus our quantum Fourier transform circuit will map the first qubit to the first term in the product above, the second qubit to the second term in the product, and so on. However, we can simplify our representation even further.

The integer j in binary is represented by its bits $j_1 j_2 j_3 ... j_n$ each of which can be 0 or 1. Dividing j by some power of two, 2^k, we see that $2^{-k} j = j_1 j_2 ... j_{n-k} . j_{n-k+1} ... j_n$ (if this is confusing, think about the decimal equivalent. When we divide by 10^k, we are moving the decimal place back k places). Writing j as the sum of its integer and fractional part, we see the coefficient $e^{2\pi i (j 2^{-k})} = e^{2\pi i (j_1 j_2 ... j_{n-k})} e^{2\pi i (0.j_{n-k+1} ... j_n)}$ where the fractional part is converted back to decimal the normal way by dividing each bit by the appropriate power of 2. Since the first term in the product will be a $e^{2\pi i} = 1$ raised to some integer power, we can replace $e^{2\pi i j 2^{-k}}$ with $e^{2\pi i (0.j_{n-k+1} ... j_n)}$. The quantum Fourier transform can now be written as

$$\frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i (0.j_n)} |1\rangle\right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i (0.j_{n-1} j_n)} |1\rangle\right) \otimes ... \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i (0.j_1 j_2 ... j_n)} |1\rangle\right) \tag{6}$$

We now know what each bit will go to in the quantum Fourier transform in terms of the other bits. To implement the gate, we first need to define a unitary operator

$$R_k \equiv \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i 2^{-k}} \end{pmatrix}$$

We will also need another important quantum operator called the Hadamard gate.

$$H \equiv \frac{1}{\sqrt{2}}(\sigma_X + \sigma_Z) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

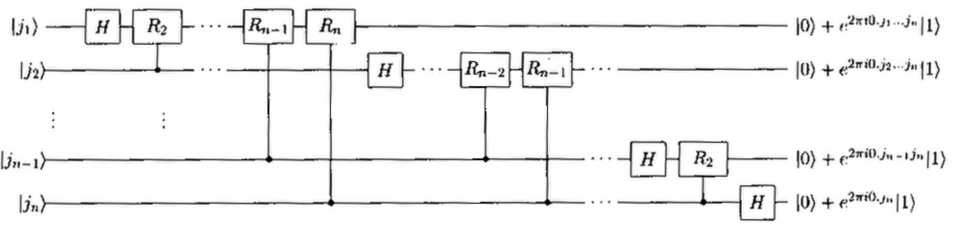

$|j_1\rangle$
$|j_2\rangle$
$|j_{n-1}\rangle$
$|j_n\rangle$
H
R_2
R_{n-1}
R_n
R_{n-2}
$|0\rangle + e^{2\pi i 0.j_1 \dots j_n}|1\rangle$
$|0\rangle + e^{2\pi i 0.j_2 \dots j_n}|1\rangle$
$|0\rangle + e^{2\pi i 0.j_{n-1} j_n}|1\rangle$
$|0\rangle + e^{2\pi i 0.j_n}|1\rangle$

Figure 1: The Quantum Fourier Transform Subroutine [4, p 219]

Lets follow the action of the circuit on the first bit. First the Hadamard gate is applied on the first qubit. If $|j_1\rangle = |0\rangle$, then $H|\mathrm{j}_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, and if $|j_1\rangle = |1\rangle$ then $H|\mathrm{j}_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Thus, we can see that the action of the Hadamard gate on the qubit is actually

$$H|j_1\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(j_1/2)}|1\rangle\right) = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(0.j_1)}|1\rangle\right)$$

Now we follow the operation of the R operators. Each adds a phase onto the $|1\rangle$ eigenstate controlled by a different bit of input. We can write the effect of the R_2 operator controlled by the 2nd qubit on the 1st qubit as the matrix below, because if $j_2 = 0$, then the operator goes to the identity. Otherwise, if $j_2 = 1$, it does its defined operation. Thus, we can see that the application of the phase operators R on the first qubit results in the state $\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(0.j_1 j_2 \dots j_n)}|1\rangle\right)$.

$$\begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i j_2 2^{-2}} \end{pmatrix}$$

Repeating the process for each bit and comparing the output with (6), we see that the circuit above gives the Fourier transform but with the bits in the reverse order. To put them in the correct order, we need $\frac{n}{2}$ CROSSOVER[17] gates, with each CROSSOVER gate using a small, finite number of universal gates to implement. The first qubit requires n gates, the second qubit requires $n-1$, the third qubit requires $n-2$, and so on. Adding up the gates in the above circuit, we get n^2 gates. Therefore, the cost of the quantum Fourier transform in the implementation above is $O(n^2)$, where n is the number of qubits [4, pp 216-220]. We will use our efficient implementation of the QFT to implement phase-estimation, which is a key step towards order finding.

2.3 Phase Estimation

Next, we move onto the Phase Estimation algorithm. Phase estimation determines the eigenvalue of a unitary operator given the operator and the eigenvector associated with that eigenvalue. It utilizes the inverse quantum Fourier transform, which is found by just taking the Fourier transform circuit in reverse and flipping each gate with its adjoint.[18] Mathematically, the inverse QFT is the adjoint of the QFT operator.

[17]The CROSSOVER gate allows you to swap qubits.

[18]This is the general procedure for finding the inverse circuit to any quantum operation. Note that the inverse QFT will be just as efficient (in the number of gates) as QFT.

Phase Estimation: Given unitary operator U, eigenvector $|u_s\rangle$ s.t. $U|u_s\rangle = e^{2\pi i\varphi_s}$, estimate φ_s[19].

We have two registers, the first contains t qubits and the second register contains the number of bits necessary to store $|u_s\rangle$. At the end of the operation, the first register will store φ_s. We also need to have a method for generating U raised to distinct powers of two from 2^0 to 2^{t-1}. For now, assume that we can efficiently generate these t gates. The number of bits in the first register, t, will be the proportional to the accuracy of our estimate of the phase. The phase $\varphi_s \in [0,1)$ because all complex numbers of modulus one can be represented by their angle in the complex plane.

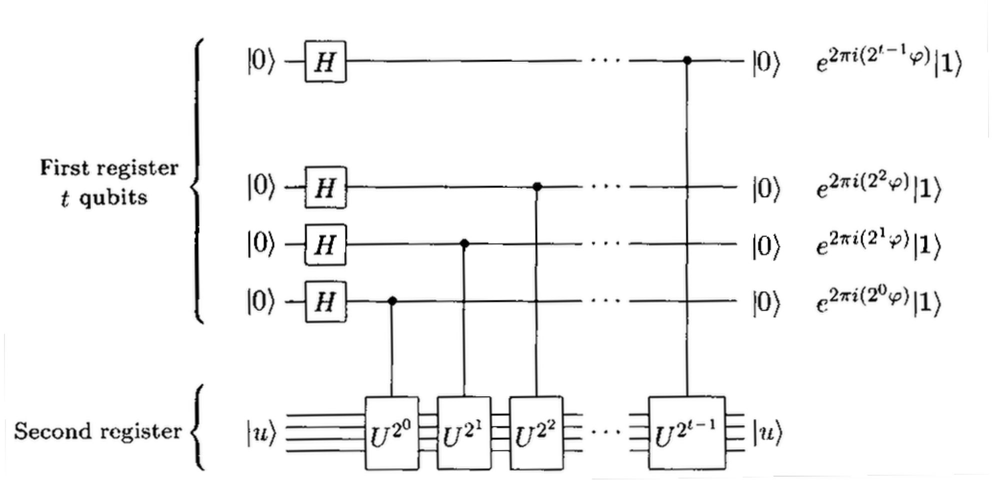

First register
t qubits
Second register
$|0\rangle$
H
$|u\rangle$
U^{2^0}
U^{2^1}
U^{2^2}
$U^{2^{t-1}}$
$|0\rangle$ $e^{2\pi i(2^{t-1}\varphi)}|1\rangle$
$|0\rangle$ $e^{2\pi i(2^2\varphi)}|1\rangle$
$|0\rangle$ $e^{2\pi i(2^1\varphi)}|1\rangle$
$|0\rangle$ $e^{2\pi i(2^0\varphi)}|1\rangle$

Figure 2: Part of the Phase Estimation Algorithm. To complete the algorithm, we act with $U_{QFT}^{\dagger}$ on the first register [4, p 222]

Following the last qubit, we see that it acted on by the Hamadard operator from the zero state. This brings it to the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Note that $|u_s\rangle \to e^{2\pi i\varphi_s}|u_s\rangle$ if its control bit is $|1\rangle$ and otherwise is unaffected. Since the last qubit has equal probability of being spin up and spin down, we can write the product state of the last qubit-second register system after the action of the first controlled-U gate as

$$\frac{1}{\sqrt{2}}\left(|0\rangle|u_s\rangle + |1\rangle(e^{2\pi i\varphi_s}|u_s\rangle)\right) = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(2^0\varphi_s)}|1\rangle\right)\otimes|u_s\rangle$$

The second controlled-U gate is controlled by the second to last qubit, and will add a phase of $(e^{2\pi i\varphi_s})^2 = e^{2\pi i(2^1\varphi_s)}$ if the second to last qubit is in the state $|1\rangle$. Thus the tensor product state of the second to last qubit, last qubit, and 2nd register is

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(2^1\varphi_s)}|1\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(2^0\varphi_s)}|1\rangle\right)\otimes|u_s\rangle$$

Notice that this state gives equal probability of no phase being added, the first phase being added, the second phase being added, and both phases being added (as expected). Continuing with the pattern, the final state of the first register is given by

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(2^{t-1}\varphi_s)}|1\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(2^{t-2}\varphi_s)}|1\rangle\right)\otimes\ldots\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(2^0\varphi_s)}|1\rangle\right)$$

[19]It should be noted that all eigenvalues of a unitary operator have modulus 1.

We can rewrite the state above as

$$\frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi i\varphi_s k}\left|k\right\rangle \tag{7}$$

The goal is to represent φ_s in t bits. Supposing $\varphi_s = 0.\varphi_{s1}\varphi_{s2}...\varphi_{st}$, we see that multiplying the phase by 2^{t-1} moves the decimal point over $t-1$ places, resulting in $\varphi_{s1}\varphi_{s2}...\varphi_{st-1}.\varphi_{st}$, and we can therefore rewrite the product state in the first register. In this form, it is easy to see that this is precisely the Fourier transform of $\left|\varphi_s\right\rangle = \left|\varphi_{s1}\varphi_{s2}...\varphi_{st-1}\varphi_{st}\right\rangle$. To get $\left|\varphi_s\right\rangle$, we simply do the inverse Fourier transform on the result of the first register.

$$\tfrac{1}{\sqrt{2}}\left(\left|0\right\rangle + e^{2\pi i(0.\varphi_{st})}\left|1\right\rangle\right)\otimes\tfrac{1}{\sqrt{2}}\left(\left|0\right\rangle + e^{2\pi i(0.\varphi_{st-1}\varphi_{st})}\left|1\right\rangle\right)\otimes ...\tfrac{1}{\sqrt{2}}\left(\left|0\right\rangle + e^{2\pi i(0.\varphi_{s1}\varphi_{s2}...\varphi_{st-1}\varphi_{st})}\left|1\right\rangle\right)$$

$$(U_{QFT}^{\dagger}\otimes\mathbb{I})\frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi i\varphi_s k}\left|k\right\rangle\left|u_s\right\rangle = \left|\tilde{\varphi}\right\rangle\left|u_s\right\rangle \tag{8}$$

Where $\tilde{\varphi}$ is the t-bit representation of the phase of the eigenvalue φ. Note that if φ_s could be written exactly in t-bits (as we assumed before), then doing a measurement in the computational basis after the inverse QFT would give φ_s exactly. If it cannot be written in this manner, however, we end up with a high probability of measuring φ_s within some error, given below [4, pp 220-222].

In order to get φ_s to n-bit accuracy with probability at least $1-\epsilon$, we choose to have[20]

$$t = n + \left[\lg\left(2+\frac{1}{2\epsilon}\right)\right] \tag{9}$$

Thus, we have exhibited an algorithm that will estimate the phase of the eigenvalue of a unitary operator. What's more, we can increase the accuracy and the probability of success by adding more qubits. The next step is to show that the Phase Estimation algorithm is polynomial in the number of gates used. The inverse quantum Fourier transform requires as many gates as the quantum Fourier transform which is quadratic in the number of bits, so we only need to show that we can find the t powers of U in polynomial time. Whether this is possible depends on U itself.

2.4 Efficient Order Finding through Quantum Algorithms

The success of Shor's Algorithm is its ability to do order-finding efficiently. The Order-finding algorithm, given positive integers x, N, returns the order r of x modulo N.

Presently, no classical algorithm can find the order of x efficiently. That is, no classical algorithm that can find r is in $O(L^k)$ for some $k\in\mathbb{Z}$, where L is the number of bits needed to store N i.e. $L\approx\lg N$.[21] However, Shor's algorithm can find r in $O(L^3)$.

The Order-finding algorithm utilizes the Phase Estimation algorithm and another operator U_x which does the following:

$$U_x\left|y\right\rangle = \left|xy \mod N\right\rangle \text{ with y} \in\{0,1\}^L \tag{10}$$

First, we note that U_x isn't a surjective mapping from $\{0,1\}^L$ to $\{0,1\}^L$, and so it can't be unitary (unitaries are bijective by definition). As defined above, U_x can only map to vectors in the range $\left|0\right\rangle,\left|1\right\rangle,...\left|N-1\right\rangle$. Next, we consider the first N values of y. Let $0\le m,n\le N-1$ and $m\neq n$. Then

[20]Chapter 5 of [4] gives a detailed explanation for this result.
[21]In general, $L=$Floor$(N)+1$, which is roughly $\lg N$.

$$\begin{aligned} U_x\,|m\rangle &= |xm \mod N\rangle = |c_1 \mod N\rangle \\ U_x\,|n\rangle &= |xn \mod N\rangle = |c_2 \mod N\rangle \end{aligned}$$

Could U_x map $|m\rangle$ and $|n\rangle$ to the same vector? Then $c_1 = c_2 \mod N$, and therefore $xm = xn \mod N$. We know x is co-prime to N by definition, so it has an multiplicative inverse modulo N. Multiplying both sides by this inverse gives $m = n \mod N$. However, since m and n are between 0 and $N-1$, $m = n$. This is a contradiction, so we know that when $0 \le y \le N-1$, U_x will map bijectively to another vector in that range. When y is outside that range, we require that U_x map to y so that the function is invertible $\forall y$.

With this new requirement, we can check if U_x is unitary. The proof will follow the same form as the proof for the QFT. If m, n are between 0 and $N-1$, then

$$\begin{aligned} \left(U_x^\dagger U_x\right)_{mn} &= \langle m|\, U_x^\dagger U_x\,|n\rangle \\ &= \langle xm \mod N|xn \mod N\rangle \\ &= \delta_{(xm \mod N),(xn \mod N)} = \delta_{mn} \end{aligned}$$

Otherwise, if m, n are both N or greater

$$\left(U_x^\dagger U_x\right)_{mn} = \langle m|\, U_x^\dagger U_x\,|n\rangle = \langle m|n\rangle = \delta_{mn}$$

And finally, if m is in one range, and n is in another, then obviously $\left(U_x^\dagger U_x\right)_{mn} = 0$ because each will get mapped to a different part of the image. $\square$

The Order-finding algorithm works by applying the Phase Estimation algorithm on U_x and one of its eigenvectors $|u_s\rangle$. So what are the eigenvectors of U_x? Consider the vector defined below. It is clear that there are r unique eigenvectors of this form, for $s = 0, 1, 2...r-1$. These are the eigenvectors for the subspace in which U_x acts non-trivially. We can confirm that it's an eigenvector by multiplying it by U_x.

$$|u_s\rangle \equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i s k}{r}\right] |x^k \mod N\rangle \tag{11}$$

$$\begin{aligned} U_x\,|u_s\rangle &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i s k}{r}\right] |x^{k+1} \mod N\rangle \\ &= \frac{1}{\sqrt{r}} \sum_{k=1}^{r-1} \exp\left[\frac{-2\pi i s (k-1)}{r}\right] |x^k \mod N\rangle \\ &= e^{2\pi i s/r}\left(\frac{1}{\sqrt{r}} \sum_{k=1}^{r-1} e^{-2\pi i s k/r}\,|x^k \mod N\rangle + \frac{1}{\sqrt{r}} e^0\,|x^0 \mod N\rangle\right) \\ &= e^{2\pi i s/r}\left(\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i s k/r}\,|x^k \mod N\rangle\right) = e^{2\pi i s/r}\,|u_s\rangle \end{aligned}$$

$$U_x\,|u_s\rangle = e^{2\pi i s/r}\,|u_s\rangle \tag{12}$$

The Phase Estimation algorithm, given U_x and $|u_s\rangle$, would find (s/r) from which we hope to determine r, the order of x modulo N. However, there are some obstacles to this: first, we need an efficient way to do phase estimation; second, we need to be able to prepare $|u_s\rangle$ without knowledge of r (since that is what we are trying to find); third, once we have some n-bit approximation of (s/r), how do we get r?

It turns out that the first challenge has a solution. Using methods of reversible computation, we can compute the t gates required for phase-estimation of U_x requiring only $O(L^3)$ additional operations. For the purposes of understanding Shor's algorithm, this is as much as we need to know[22] [4, p 228].

To answer the second challenge, we note that

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle = |1\rangle \tag{13}$$

In fact, this is a particular case of a more general observation that we will prove below.

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}e^{2\pi isk/r}|u_s\rangle = |x^k \mod N\rangle \tag{14}$$

$$\begin{aligned}\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}e^{2\pi isk/r}|u_s\rangle &= \frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}e^{2\pi isk/r}\frac{1}{\sqrt{r}}\sum_{k'=0}^{r-1}\exp\left[\frac{-2\pi isk'}{r}\right]|x^{k'} \mod N\rangle \\ &= \frac{1}{r}\sum_{k'=0}^{r-1}|x^{k'} \mod N\rangle\sum_{s=0}^{r-1}e^{2\pi is(k-k')/r} \\ &= \frac{1}{r}\sum_{k'=0}^{r-1}|x^{k'} \mod N\rangle\,(r\delta_{k,k'}) = |x^k \mod N\rangle \qquad \square\end{aligned}$$

Plugging in $k = 0$ for (14), we get (13). Thus, the state $|1\rangle$ is an equal superposition of the eigenstates of the operator U_x.

Now, consider $t = 2L + 1 + \lg\left(2 + \frac{1}{2\epsilon}\right)$ bits prepared in the $|0\rangle$ state for the first register of phase transformation algorithm and in the second register, which only needs L qubits, we put the state $|1\rangle$. After the application of the first unitary, $U_x^{2^0}$, the tensor product of the last qubit and the second register will be

$$\begin{aligned}&\frac{1}{\sqrt{2}}\left(|0\rangle|1\rangle + |1\rangle U_x|1\rangle\right) \\ &\frac{1}{\sqrt{2}}\left(|0\rangle|1\rangle + |1\rangle\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}e^{2\pi is/r}|u_s\rangle\right) \\ &\frac{1}{\sqrt{2}}\left(|0\rangle|1\rangle + |1\rangle|x^1 \mod N\rangle\right)\end{aligned}$$

Where in the second line we used (13) combined with (12) and in the third line we used (14). Now consider the tensor product of the last two qubits and the second register after the application of the 2nd unitary, $U_x^{2^1}$.

[22] Some insight as to how this works: If the contents of the first and second registers of phase estimation are $|z\rangle$ and $|y\rangle$ respectively, then the result of phase estimation is simply $|z\rangle|x^z y \mod N\rangle$. To see how we can compute x^z mod N efficiently, consider the following example: to can compute 2^{16} efficiently, we only need to do 4 multiplications. 2×2 to get 2^2, then $2^2\times 2^2$ to get 2^4, then $2^4\times 2^4$ to get 2^8, then $2^8\times 2^8$ to get 2^{16} [4, p 228].

$$\frac{1}{2}\big(|0\rangle|0\rangle|1\rangle + |0\rangle|1\rangle|x^1 \mod N\rangle + |1\rangle|0\rangle|x^2 \mod N\rangle + |1\rangle|1\rangle|x^3 \mod N\rangle\big)$$

Temporarily leaving out the factor of $\frac{1}{\sqrt{2^t}}$, the result of the Phase Estimation algorithm before the inverse QFT is

$$\sum_{j=0}^{2^t-1}|j\rangle|x^j \mod N\rangle = \sum_{j=0}^{2^t-1}|j\rangle\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}e^{2\pi isj/r}|u_s\rangle$$
$$= \sum_{j=0}^{2^t-1}|j\rangle\frac{1}{\sqrt{r}}\Big(e^{2\pi ij(0/r)}|u_0\rangle + e^{2\pi ij(1/r)}|u_1\rangle e^{2\pi ij(2/r)}|u_2\rangle ... e^{2\pi ij(\frac{r-1}{r})}|u_{r-1}\rangle\Big)$$
$$= \frac{1}{\sqrt{r}}\sum_{j=0}^{2^t-1}e^{2\pi ij(0/r)}|j\rangle|u_0\rangle + \frac{1}{\sqrt{r}}\sum_{j=0}^{2^t-1}e^{2\pi ij(1/r)}|j\rangle|u_1\rangle + ...\frac{1}{\sqrt{r}}\sum_{j=0}^{2^t-1}e^{2\pi ij(\frac{r-1}{r})}|j\rangle|u_{r-1}\rangle$$

The inverse QFT acts on each sum linearly. Comparing each sum with (8), we see that the inverse QFT will give an equal superposition of all the possible phases in the first register: $0, \frac{1}{r}, \frac{2}{r}, ...\frac{r-1}{r}$. The result of the Phase Estimation algorithm is therefore

$$\frac{1}{\sqrt{r}}\left(|\tilde{\frac{0}{r}}\rangle|u_0\rangle + |\tilde{\frac{1}{r}}\rangle|u_1\rangle + |\tilde{\frac{2}{r}}\rangle|u_2\rangle + ...|\frac{r\tilde{-}1}{r}\rangle|u_{r-1}\rangle\right) \tag{15}$$

Thus, a measurement of the first register will give an approximation to (s/r) accurate to $2L+1$ bits, with a probability of error less than ϵ. Increasing the number of bits in the first register can put a very low upper-bound on the error, so at the end of the Phase Estimation algorithm, we can expect to have an approximation to (s/r) for some s ranging from 0 to $r-1$.

Using the classical continued fractions algorithm, we can find a reduced fraction form of (s/r) in $O(L^3)$ arithmetical operations [4, p 229]. Let's refer to our reduced fraction approximation as (s'/r').

If s and r and co-prime, then the reduced fraction's denominator r' will match with r. We can check if it's correct by checking if $x^{r'} \equiv 1 \mod N$. If the test fails, the most likely result is that s and r share some common factors. That means r' is some factor of r.

If we repeat the process for the same x and end up with another pair (s''/r''), there's some probability that s'' and s' are co-prime [23]. If this is the case, the LCM of r' and r'' (given by their product divided by their gcd) will give the order r that we are looking for [4, pp 226-229].

Thus, at the end of all this, we are left with an accurate estimation of the order r of an integer x modulo N. The computation uses $O(L^3)$ gates or operations, and so it is polynomial in the input size. Given the order, we can solve the factoring problem efficiently using the classical reduction algorithm from section 1.

2.5 An example: N = 21

Let's factor $N = 21 = 3 \times 7$ using Shor's algorithm. N will fail the two steps of Shor's algorithm in the overview, so we choose a random element x from 0 to $N-1$. Suppose we choose $x = 10$ and compute the gcd. As expected, x is co-prime with N so we continue with the Order-finding algorithm. In binary $N = 10101$, so $L = 5$. Choosing $\epsilon = .25$, we get $t = 13$ qubits in the first register.

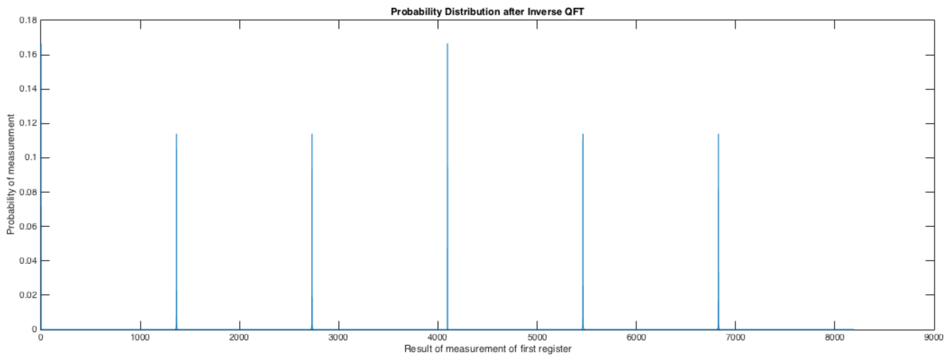
[23] One can bound this probability a number of ways. One lower-bound is $p(\gcd(s', s'') = 1) \geq 1/4$ [4, p 231].

The result of phase estimation before the inverse QFT is $\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^{13}-1} |j\rangle |x^j \mod N\rangle =$

$$\frac{1}{\sqrt{2^{13}}}\left(|0\rangle|1\rangle + |1\rangle|10\rangle + |2\rangle|16\rangle + |3\rangle|13\rangle + |4\rangle|4\rangle + |5\rangle|19\rangle + |6\rangle|1\rangle + |7\rangle|10\rangle ...\right) \tag{16}$$

To simplify the application of the inverse QFT, we assume that the second register is measured[24] to be 16. So the final result of phase estimation is

$$U^{\dagger}_{QFT}\sqrt{\frac{6}{2^{13}}}\left(|2\rangle + |8\rangle + |14\rangle + |20\rangle ... |8186\rangle\right) \tag{17}$$



We used MATLAB to complete the computation, generating the probability distribution for the result of measuring the first register [25]. The most probable results of measurement are 0, 1365, 2731, 4096, 5461, and 6827. Suppose we measured 1365. The algorithm would interpret this as the fraction $1365/2^{13}$ and the continued fractions algorithm would return 1/6 (which is equal to $1365/2^{13}$ to an 11-bit approximation). We see that $r' = 6$ gives the order of 10 modulo 21. As expected, r is even and $10^{r/2} \equiv 10^3 \equiv 13 \neq -1 \mod 21$. We are guaranteed a non-trivial factor in computing $\gcd(12, 21)$ and $\gcd(14, 21)$. The first computation gives 3 and the second gives 7, the prime factors of 21.

Appendix

A. Algorithm Analysis

How do we judge the efficiency of an algorithm for solving a particular problem? First, we need a definition of *cost*. Cost is the consumption of some valuable resource. The valuable resource could be time, number of gates, number of operations, memory, energy, etc. An efficient algorithm, then, minimizes cost. For the quantum algorithms used in this paper, the resource is the number of quantum gates.

Since an algorithm is a sure-fire series of steps to get from some input to an answer, the cost of an algorithm is a function of the input only. To simplify matters, we consider the cost of an algorithm in terms of some dimension of the input. For example, input size. Note that for a particular input size there could be a lot of variation in the cost. A certain sorting algorithm could take advantage of when a list is mostly sorted and be less costly than it would be on an arbitrary input of the same size. Therefore, we are usually concerned with worst case performance.

[24]This can be assumed from the Principle of Implicit Measurement which says any unterminated wires can be taken as measured [4, p 187]. The second register has nothing done to it after (15), so it is unterminated.

[25]Code in appendix

Suppose $f(n)$ represents the cost of an algorithm on an input of size n. We say that

$$f(n) \in O(g(n)) \iff \exists c, n_o \mid \forall n > n_o, f(n) \leq c\, g(n) \tag{18}$$

The equation above gives a large n asymptotic limit on the cost, $f(n)$, of computing the algorithm on input size n. It gives an upper-limit. All polynomial functions $p(x)$ are in $O(2^n)$. This is because an exponential function on n will always eventually outgrow a polynomial function on n.

An algorithm is usually referred to as **efficient** if $\exists k \mid f(n) \in O(n^k)$ where k is some positive integer. Any polynomial or sub-polynomial function ($\log n, \sqrt{n}$, etc.) on n is efficient under that criteria. Problems for which an efficient algorithm does not yet exist are usually referred to as **hard** or **intractable** [4, Ch 3].

B. Proving the Sum of Equally Spaced Phases in the Complex Plane is 0

Statement: $\sum_{k=0}^{N-1} e^{\frac{2\pi i(n-m)k}{N}} = N\delta_{mn}$

Proof: The case where $n = m$ is trivial, so we consider the case where $n \neq m$. The proof uses the geometric sum formula. Let $s = 1 + e^{2\pi i(n-m)/N} + \left(e^{2\pi i(n-m)/N}\right)^2 + ... \left(e^{2\pi i(n-m)/N}\right)^{N-1}$. Note that s is equal to the left hand side of the statement above. By the formula for the finite geometric sum, $s = \frac{1-\left(e^{2\pi i(n-m)/N}\right)^N}{1-e^{2\pi i(n-m)/N}} = \frac{1-e^{2\pi i(n-m)}}{1-e^{2\pi i(n-m)/N}}$. Since $n - m \in \mathbb{Z}$, the numerator is zero. Also, since $n \neq m$, $0 < |n-m| < N$ and so the denominator is non-zero (recall that $|m\rangle$ and $|n\rangle$ are basis states labeled from 0 to $N-1$). Therefore, we have the result that the LHS is N when $n = m$ and 0 when $n \neq m$. $\square$

C. Chinese Remainder Theorem

The Chinese Remainder Theorem says that the set of equations

$$\begin{aligned} x \equiv& a_1 \mod m_1 \\ x \equiv& a_2 \mod m_2 \\ &... \\ x \equiv& a_n \mod m_n \end{aligned}$$

where the m_i are positive integers that are co-prime with each other has a solution for x.

Proof: Define $M = m_1 m_2 ... m_n$ and $M_i = (M/m_i)$. Since M_i is co-prime with m_i it has an inverse modulo m_i, M_i^{-1}. Then the solution is

$$x = \sum_{i=1}^{n} a_i M_i M_i^{-1}.$$

To see how his works, plug it into the equations above. For some j,

$$\begin{aligned} a_j \equiv& x \mod m_j \\ \equiv& \sum_{i=1}^{n} a_i M_i M_i^{-1} \mod m_j \\ \equiv& a_j M_j M_j^{-1} \mod m_j \\ \equiv& a_j \mod m_j \end{aligned}$$

The terms involving M_u, where $u \neq j$, will be equivalent to zero modulo m_j since they are multiples of m_j.

Additionally, we can say that any two solutions x and x' will be equivalent modulo M. It is clear from transitivity that

$$x - x' \equiv 0 \mod m_i \quad \forall i$$

Thus, $m_i|(x - x')$ for all i. Since each of the m_i are co-prime, they won't share any of the prime factors that make up $x - x'$, so $M|(x - x')$. Thus, $x - x' \equiv 0 \mod M$ [4, p 627]. Additionally, suppose that x is a solution and $y \equiv x \mod M$. Then $x \equiv y \mod m_i \, \forall i$, and by substitution y is also a solution to the system of equations. From this, it follows that the set of solutions is some equivalence class of M.

One final thing to note is that there is a bijective mapping between the $\{\{a_i\}\}$ and the solutions x modulo M. That is to say, the set $\{a_i\}$ determines a unique solution x modulo M, and the solution x determines a unique $\{a_i\}$.

Consider set A, the set of the set of allowed a_i in the equations. We constrain a_j s.t. $0 \leq a_j \leq m_j - 1$.[26] Each member of A is a set

$$\{a_1, a_2, a_3 ... a_n\}$$

For example, for $n = 3$ and $m_1 = 3, m_2 = 10, m_3 = 7$, the following are allowed members of A:

$$\{1, 9, 5\}$$
$$\{0, 0, 0\}$$
$$\{2, 5, 6\}$$

Set A has a total of M elements. This is because there are m_1 possible values for a_1, m_2 possible values for a_2, and so on.

Set B $= \{0, 1, 2, 3 ... M - 2, M - 1\}$.[27] There are M elements of B, so A and B have the same size.

Consider the mapping f that takes an element of A and maps it to the corresponding solution in B. We know that solutions for the same $\{a_i\}$ differ by multiples of M, so we know such a mapping will be a function (an element of A will map to one element of B). Using the Chinese Remainder Theorem, we see that an adequate f is one that maps $\{a_i\}$ to its corresponding solution, subtracting k multiples of M so the result is between 0 and $M - 1$.

$$f(\{a_1, a_2, ... a_n\}) = \left(\sum_{i=1}^{n} a_i M_i M_i^{-1} \right) - k_{\{a_1, a_2, ... a_n\}} M$$

We will show that f is injective, and thus f is surjective since it is a function on sets of equal size. Each distinct element of A maps to a different element of B, so every element of B has to get mapped to. Therefore, f is a one-to-one correspondence mapping. For every $\{a_1, a_2, ... a_n\}$ in the system of equations, there is a corresponding x and vice versa.

We need to show that if $f(\{a_1, a_2, ... a_n\}) = f(\{b_1, b_2, ... b_n\})$, then $a_i = b_i \forall i$.

[26] Any range of m_j consecutive integers would work.

[27] Any set of M consecutive integers would work.

Proof:

$$
\begin{aligned}
f(\{a_1, a_2, ...a_n\}) &= f(\{b_1, b_2, ...b_n\}) \\
\left(\sum_{i=1}^{n} a_i M_i M_i^{-1}\right) - k_{\{a_1,a_2,...a_n\}} M &= \left(\sum_{i=1}^{n} b_i M_i M_i^{-1}\right) - k_{\{b_1,b_2,...b_n\}} M \\
\sum_{i=1}^{n}(a_i - b_i) M_i M_i^{-1} &= k' M \\
\sum_{i=1}^{n}(a_i - b_i) M_i M_i^{-1} &\equiv k' M \mod m_j \\
a_j - b_j &\equiv 0 \mod m_j
\end{aligned}
$$

Since $0 \leq a_j, b_j \leq m_j - 1$, $a_j = b_j$. $\square$

2.6 D. Calculating the Inverse QFT on a vector

We used the following code to generate the figure in Section (2.5).

```
%Compute QFT matrix U%
N = 2^13;
U = ones(N, N);
w = exp(2 * pi * 1i / N);

for j = 1:N
        w_j = w^(j - 1);
        for k = 1:N
                if (k == 1)
                        U(j, k) = 1;
                else
                        U(j, k) = U(j, k - 1) * w_j;
                end
        end
end

U = (1 / sqrt(N)) .* U;

%To compute inverse QFT, we take the adjoint of U%
inverseU = U';

%Generate input vectors
s = linspace(2, 8186, 1365);
v = zeros(N, 1);
for n = s
        v(n + 1) = 1;
end

v = v / norm(v);

%apply transformation
```

```
result = inverseU * v;

%Calculate amplitudes
amplitudes = result .* conj(result);

figure
plot(0 : (N - 1), amplitudes)
title('Probability Distribution after Inverse QFT')
xlabel('Result of measurement of first register')
ylabel('Probability of measurement')
```

References

[1] "Fields Medalists / Nevanlinna Price Winner." *Fields Medalists / Nevanlinna Price Winner.* Mathunion.org, 1998. Web. 25 Dec. 2015.

[2] "General Number Field Sieve." *Wikipedia.* Wikimedia Foundation, 14 Dec. 2015. Web. 25 Dec. 2015.

[3] Landauer, R. "Irreversibility and Heat Generation in the Computing Process." *IBM Journal of Research and Development IBM J. Res. & Dev.* 5.3 (1961): 183-91. Web. 1 Jan. 2016.

[4] Nielsen, Michael A., and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge: Cambridge UP, 2000. Print.

[5] Pinter, Charles C. *A Book of Abstract Algebra.* New York: McGraw-Hill, 1982. Print.

[6] Shor, Peter. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." *ArXiv.org.* ArXiv, 25 Jan. 1996. Web. 25 Dec. 2015.

[7] Zyga, Lisa. "New Largest Number Factored on a Quantum Device Is 56,153." *New Largest Number Factored on a Quantum Device Is 56,153.* Phys.org, 28 Nov. 2014. Web. 25 Dec. 2015.