Quantum Computing and Shor's Algorithm

Tristan Moore

June 7, 2016

Abstract

This paper covers the basic aspects of the mathematical formalism of quantum mechanics in general and quantum computing in particular, underscoring the differences between quantum computing and classical computing. This paper culminates in a discussion of Shor's algorithm, a quantum computational algorithm for factoring composite numbers that runs in polynomial time, making it faster than any known classical algorithm for factorization. This paper serves as a survey of *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer* by Peter Shor[3].

Contents

1 Introduction

With the devolopment of computability thoery, many important problems in computer science and mathematics exist for which there is no known polynomial-time algorithm. The physicist Richard Feynman seems to have been the first to observe that quantum mechanics might allow for faster computation than would be allowed on a classical Turing machine, and thus raised the possibility that a quantum mechanical computer might allow for more robust solutions to problems that have long plagued classical computer science. Among the problems for which no known polynomial-time deterministic algorithm exists is the integer factorization problem, which is thought to be difficult enough to compute that it forms the basis of most of modern cyrptological systems. Shor's algorithm, which this paper provides an exploration of, provides an efficient polynomial time algorithm that operates on a quantum computer. Although it has not been formally proven, this has led to speculation that the class of problems effectively computable by a quantum Turing machine might be broader than the class of problems computable on classical Turing machines.

2 Basics and Definitions

Here we will state results from the mathematical formulation of quantum mechanics and various aspects of quantum computing, as well as various results from other branches of mathematics as necessary. Many will be given without proof, and these terms will be referred to throughout the paper. The reader may skip ahead to the main discussion and refer back to this section as necessary.

Definition 1 (Hilbert Space)**.** *A Hilbert space H is a complex vector space equipped with an inner product $\langle x, y\rangle$ that assigns a complex number c for every pair $x, y \in H$, which satisfies the following properties:*

1. *The inner product of a pair of elements is the complex conjugate of the reverse:*
$$\langle y, x\rangle = \overline{\langle x, y\rangle}$$
2. *The product should be linear with respect to the first argument:*
$$\langle ax_1 + bx_2, y\rangle = a\langle x_1, y\rangle + b\langle x_2, y\rangle$$
3. *The inner product is positive definite:*
$$\langle x, x\rangle \geq 0$$
With equality holding precisely when $x = 0$.
4. *The product is antilinear with respect to the second argument:*
$$\langle x, ay_1 + by_2\rangle = \overline{a}\langle x, y_1\rangle + \overline{b}\langle x, y_2\rangle$$

For our purposes, Hilbert Spaces will be used to define the *state space* of our system. The state space of a quantum system has dimensionality equal to the number of possible outcomes of a measurement of a given property of the system. The various outcomes are each represented as vectors that they form a normalized, orthogonal basis for the state space. The inner product in this space is typically defined in terms of the basis vectors b_i of the system, such that:

$$\langle b_i b_j \rangle = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$$

Each of these b_i denote a possible outcome of measurement, with the set of all b_i's corresponding to the set of all possible measurements of a system. Thus, the inner product as defined above can be stated informally as 'selecting' the component of a state corresponding to a given outcome.

Definition 2 (Quantum State)**.** *A quantum state is a vector in the Hilbert space associated with a system that assigns to each possible outcome of an experiment a probability.*

Typically quantum states are represented in *ket notation*, where a general state $|\psi\rangle = \sum a_i |b_i\rangle$, where $|b_i\rangle$ represents a basis vector for the state space. The a_i are complex numbers associated with each basis vector such that the possibility of measuring a given outcome $|b_i\rangle$ is given by $|a_i|^2$. These should be normalized for all legitimate quantum states, and in this paper we assume that any given quantum state is normalized. An important note: total phase does not impact any measurement of a quantum system, i.e. $|\phi\rangle = e^{i\alpha} |\psi\rangle$ implies that $|\phi\rangle$ is an equivalent state to $|\psi\rangle$. This can be seen by attempting to measure any observable: $|e^{i\alpha} a_i^2| = |a_i|^2$, so the two states are observably indistinguishable.

Definition 3 (Operator)**.** *An operator is a matrix corresponding to an observable property of a quantum state. Measuring the observable corresponds to selecting an eigenstate of the unitary matrix associated with the observable in accordance to the probabilities contained in the wave function. Quantum operators have the following properties:*

1. *All operators are linear and unitary, i.e.* $\mathbf{A^{-1}} = \overline{\mathbf{A^T}}$
2. *Consequently, all eigenvectors are orthogonal and form a complete orthonormal basis.*

For our purposes, operators will be used to create the quantum circuitry used to transform the quantum system under consideration. Operators in this capacity are sometimes called *Quantum Gates.* These will be discussed in more detail later.
Applying an operator to a system corresponds to transforming one quantum state to another in some manner, while measuring a system involves applying the associated operator and then selecting out a single outcome from the set of basis vectors with probability corresponding to the $|a_i|^2$. Note that measuring "collapses" the wave function: repeating a measurement of the same system will always return the same value. For example, consider $|\psi\rangle = \frac{1}{sqrt2} |a\rangle + \frac{1}{\sqrt{2}} |b\rangle$. Then the probability of measuring either $|a\rangle$ or $|b\rangle$ is equal to $\left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}$.

Suppose that we measure the system and report a measurement of $|a\rangle$. Then after the measurement, $|\psi\rangle = |a\rangle$, and any repeated measurement will always return $|a\rangle$.

Definition 4 (Qubit)**.** *A quantum bit, or qubit, is a 2-state quantum system corresponding to a bit of data on a classical computer. In quantum computing, a qubit is represented as a linear superposition of 1's and 0's, expressed in ket-notation, i.e. $|0\rangle$ and $|1\rangle$. For systems of multiple qubits, it is often customary to collapse the state into multi-qubit states, as in $|0101\rangle$, which corresponds to a 0 on the first qubit, a 1 on the second, and so on. Therefore, our computational basis has 2^n basis states, corresponding to every possible measurement of each qubit.*

Definition 5 (Quantum Entanglement)**.** *An quantum system is said to be entangled when pairs or groups of particles are generated such that they cannot be entirely described independently, rather they must be described together as an interacting object to fully describe the system.*

As an example of an entangled system, consider the wave function $|\psi\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$. In this case, there are two possible measurements of the system: either a 0 is measured on the first qubit and a 1 on the second, or a 1 is measured on the first qubit and a 0 on the second. Suppose we only measure the first qubit of this system, and report a measurement of 1. We can then conclude that the system is now in the state $|10\rangle$, because after measuring the first qubit, the state collapses into the second eigenvector. We can then say that the first and second qubits are *entangled*, because a measurement of the first qubit will change the . However, consider the state $|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$. These qubits are *not* entangled. Suppose we measure the first qubit. In either eigensate, the second qubit is in the 0 state, and thus is independent of any measurement on the first qubit, so the second state remains unchanged by our measurement. The same is true for a measurement on the second qubit: Because the second qubit will always be measured to be in the 0 state, the first qubit will remain unaltered by said measurement, and in this case will continue to exist in an equal superposition of $|0\rangle$ and $|1\rangle$.
One important result of quantum entaglement is the following theorem:

Theorem 1 (No Cloning Theorem)**.** *Given two arbitrary quantum states $|\psi\rangle$ and $|\phi\rangle$ which share a common Hilbert space* $\mathbf{H}$*, then it is impossible to copy $|\psi\rangle$ onto $|\phi\rangle$ without destroying $|\psi\rangle$.*

A proof of this is beyond the scope of the paper, but can be read about in [4]

Definition 6 ($\mathbb{BQP}$)**.** *A decision problem is said to be in class $\mathbb{BQP}$ if there exists a quantum algorithm that solves the problem with a uniform polynomial-size quantum gate array such that the probability of returning an incorrect answer is* *at most* $1/3$. *Note: the probability bound is arbitrary, for any desired accuracy can be achieved by running the algorithm a polynomial number of times and taking the majority vote. This is the definition we will take for efficiently solvable problems on a quantum computer.*

Here we define the Euler Totient Function and a theorem of Hardy that will be used for establishing the polynomial-time bound on Shor's algorithm.

Definition 7 (Euler Totient Function)**.** *The Euler Totient Function* $\phi(n)$ *is a function on* $\mathbb{N}$ *defined to be the number of positive integers* $\leq n$ *that are relatively prime to* n*. For example,* $1, 2, 4, 7, 8, 11, 13, 14$ *are all relatively prime to* 15*, so* $\phi(15) = 8$.

We will present the following theorem of Hardy and Wright for a bound on the totient function:

Theorem 2 $(\lim \frac{\phi(n) \log \log(n)}{n} = e^{-\gamma})$**.**

A proof of this can be found in [2]. Another theorem we will require is the Chinese Remainder theorem, although it will only be briefly cited during the discussion our factorization algorithm.

Theorem 3 (Chinese Remainder Theorem)**.** *Suppose that* $n_1 \dots n_k$ *are pairwise coprime integers. Then for a given sequence of integers* $a_1 \dots a_k$*, there exists an integer* x *that is a solution to the following:*

$$x = a_1 \mod (n_1)$$

$$\dots$$

$$x = a_k \mod (n_k)$$

A proof can be found in [2].

3 Quantum Computing

In this section we will give an overview of quantum computing, and in particular the components necessary to understand Shor's algorithm. To begin, consider a system of n two-state components. Normally, the object would be completely describable with n bits. For a quantum computer, however, this system requires $2^n - 1$ complex numbers, specifically this state is represented as a single point in a 2^n dimensional vector space. For each of the 2^n permutations of the system, there exists a basis vector in the state space for our system, such that the set of all 2^n of these vectors form a complete basis for the system. As an example, the state vector $|000 \dots 0\rangle$ would correspond to the state such that every bit is given the value 0, while the state vector $|10101 \dots\rangle$ corresponds to the state with the first qubit in the 1 state, the second in the 0 state, and so on. We then consider the state $|\psi\rangle$ of the system to be the sum of each state vector $|S_i\rangle$ with an associated complex number a_i.

$$\psi = \sum a_i \, |S_i\rangle$$

If the machine were to be measured at any point, then the state collapses into a single eigenstate of the observable, with the probability of any given $|S_i\rangle$ being selected is equal to $|a_i|^2$. After measurement, the quantum state of the system will be precisely the S_i that was returned by the measurement, thus destroying information of previous state. Therefore, when carrying out our quantum computing algorithm, we must be careful to never measure the state until after all computations have been made.
To perform operations on the quantum state in question, unitary operators will be used to implement logic gates. We will define one such gate here that will recur in our discussion of Shor's algorithm. The Hadamard Gate acts on a single qubit and maps $|0\rangle$ to $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$. In matrix form:

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

For our system, we must add two restrictions to make our gates of a uniform complexity class. The first is that the design of the gate array must be decidable in polynomial time, as per a classical algorithm. The second is that each number used in each matrix must be actually computable. Specifically, the first $\log(n)$ digits for a given entry must be computable in polynomial time. Both of these requirements are implemented to ensure that non-computable information is not hidden in the design of an efficient circuit that can therefore not be compared to a Turing Machine.

4 Reversible computation

Because quantum logic gates are reversible at all points and all transformations are unitary, it follows that a calculation is feasible on a quantum computer if and only if it is reversible. Any deterministic calculation can be made reversibly[], and from previous studies into reversible computation[], we can compute any polynomial time algorithm reversibly so long as the input x is stored in the system. Note that, by the no cloning theorem, we cannot do this in a quantum system. Instead, these steps are done by a classical subroutine prior to using a quantum circuit for the actual computation. A deterministic algorithm can be modified to be reversible under a quantum computer using a process described in [2]. Thus, to create an algorithm A that takes $x \to A(x)$, A must be $1:1$ and both A and A^{-1} must be computable in polynomial time. First, we take x to $(x, A(x))$ using the previously described method. If we instead are given a A^{-1} that can be computed in polynomial time and are asked to operate on $A(x)$, we can map to $(A(x), A^{-1}(A(x))) = (A(x), x)$, making this a reversible process.

Although this shows that an algorithm can be made reversible for only a constant increase in time, this also shows that the process takes as much space as time, and if the classical algorithm uses little space but a larger amount of time, then this method will produce an algorithm that will greatly increase the amount of space required. However, for some algorithms we can construct a different method to create a reversible algorithm without too steep of a price in space or time: A discussion follows for the example of modular exponentiation, which will play a part in quantum factorization and provide an example of one such method to limit the use of space.

The bottleneck of Shor's algorithm comes from the step for modular exponentiation. Modular exponentiation is, given n, x and r, find $x^r \mod (n)$. This is computed classically by repeatedly squaring $x \mod (n)$ to get $x^{2^i} \mod (n)$ for $i \leq \log_2(r)$ and then multiply a subset of these powers $\text{mod}(n)$ to find $x^r \mod (n)$. For numbers consisting of N bits, this operation is $O(N^2 \log N \log \log N)$ using the Schonhange-Strassen algorithm [3]. Making this reversible would take $O(N^2 \log N \log \log N)$ in space, but one can reuse the space from the repeated squarings. Thus, we can create an algorithm that takes the same amount of space as the classical algorithm, which in this case is $O(N \log N \log \log N)$. Although this method does not scale effectively for small numbers, we will choose to disregard this case, because these cases can be easily treated using a method similar to long division, and can be done in $O(N^3)$ time and $O(N)$ space [3].

We will now formulate a method for constucting a reversible gate array that can compute $(r, x^r \mod (n))$ in $O(N)$ space and $O(N^3)$ time. The basis of this method is a gate array that takes input b and computes $b + c \mod (n)$. Note that c and n are specified by the architecture of the gate, while b is the input of the gate. For a classical algorithm, addition $\text{mod}(n)$ is computable in $O(\log(n))$ time, so as follows from the previous discussion, we can create a quantum gate array that is $O(\log n)$ in both space and time. The algorithm for computing $(r, x^r \mod (n))$ is essentially identical to the classical algorithm. First, repeatedly square x l times to obtain $x^{2^i} \mod (n)$ for all $i < l$. Then, to obtain $x^r \mod (n)$ we multiply the powers $x^{2^i} \mod (n)$ such that 2^i appears in the binary expansion of r. In our algorithm, we only need consider the case where r is treated as the input for the system, while n and x are both constants that can be built into the structure of the array, which makes the actual computation of this product much simpler, because n and x can simply be built into the structure of the gate array and need not be considered as input values. The process for finding r is expressed by the following pseudocode, where r_i represents the ith digit in the binary expanion of r:

```
$power = 1$;
for $i = 0$ to $r - 1$ do
    if $r_i == 1$ then
        $power = power * x^{2^i}$;
    end
end
return power;
```

Notice that r is not changed by the code, and *power* returns the value $x^r \mod (n)$. Therefore, this code takes the input $(r, 1)$ to $(r, x^r \mod (n))$. Note that, when constructing a gate array, $x^{2^i} \mod (n)$ can be computed by a classical subroutine and then implemented into the construction of the array. Thus, implenting this requires a gate array that takes an input b to $bc \mod (n)$, where c and n are allowed to depend on the specific architecture of the array. Note that if $\gcd(c, n) \neq 1$, then we cannot create a reversible gate array, because then there are distinct b_1 and b_2 such that $b_1c \mod (n) = b_2c \mod (n)$, so there is no reversible algorithm such that we can distinguish the b_1 from the b_2. Thankfully, this will not be a concern for us, and we only need to deal with the case where there are no common factors of c and n for our factoring algorithm. The first stage of this gate is essentially multiplication by repeat addition. It is shown in pseudocode below:

```
$result = 0$;
for $i = 0$ to $r - 1$ do
    if $b_i == 1$ then
        $result = result + 2^i c \mod (n)$;
    end
end
return $result$
```

This subroutine took b as an input and produced $(b, bc \mod (n))$ as an output. Therefore, we need to erase b to obtain the desired result. We assumed that $\gcd(c, b) = 1$, so there exists a $c^{-1} \mod (n)$ such that $cc^{-1} = 1 \mod (n)$. Note that applying c^{-1} via multiplication to $bc \mod (n)$ will reversibly take $bc \mod (n)$ to $(bc \mod (n), bcc^{-1} \mod (n)) = (bc \mod (n), b)$. Because this is a reversible operation, it can be reversed to erase b, thus completing our algorithm. Here is the pseudocode for this stage:

```
for $i = 0$ to $r - 1$ do
    if $result_i == 1$ then
        $b = b - 2^i c^{-1} \mod (n)$;
    end
end
return $b$
```

5 Quantum Fourier Transform

In this section we present the Quantum Fourier transform, which acts on a quantum state much like a discrete Fourier transform would on a classical computer. The Quantum Fourier transform is a unitary transform that can be constructed in polynomial time on a quantum computer. Consider a number a with $0 \leq a < q$ for some q. Then the quantum Fourier transform takes $|a\rangle$ to

$$\frac{1}{\sqrt{q}} \sum_{n=0}^{q-1} |n\rangle \; e^{2\pi i n a/q}$$

This Fourier transform is representable by the unitary matrix as follows:

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \dots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \dots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

Where $\omega = e^{2\pi i/N}$ is the Nth primitive root of unity. This operator will be central to Shor's algorith and will be given the name A_q for the given q.

Theorem 4 (The Quantum Fourier Transform is in P.)**.** *We will consider the special case where q is a power of 2, and deduce that for other q we can approximate a solution by considering a nearby power of 2. Suppose that $q = 2^n$. Suppose that a is an integer represented in quantum binary as $|a_{n-1}a_{n-2}\dots a_0\rangle$. Allow H_j to indicate a Hadamard gate operating on the jth qubit, and allow $S_{j,k}$ to indicate the following matrix:*

$$S_{j,k} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta_{k-j}} \end{bmatrix}$$

Where $\theta_{k-j} = \pi/2^{k-j}$. Note that this is a unitary matrix, and therefore is a constuctable quantum gate. Consider the following sequence of matrices:

$$H_{n-1}, S_{n-2,n-1}, H_{n-2}, S_{n-3,n-2}, S_{n-3,n-2}, \dots H_1, S_{0,n-1}, S_{0,n-2}, \dots S_{0,2}, S_{0,1}, H_0$$

That is, we apply the H_k from H_{n-1} to H_0, and bewteen H_k and H_{k-1} we apply every $S_{j,k}$ such that $k > j$. We want to show that this tranforms the state to $\frac{1}{q^{1/2}} \sum_b \exp(2\pi iac/q)\,|b\rangle$, where $|b\rangle$ is the bit-reversal of $|c\rangle$, or the number obtained by reversing each bit of $|c\rangle$. Note, however, that it is quite simple to simply read the numbers backwards or manually reverse them, and will not impact runtime.
This circuit consists of $\frac{n(n-1)}{2}$ gates. Thus, this sequence runs in polynomial

time. To prove that this does, in fact, perform a Fourier transform on q, note that we have n H gates, each containing a factor of $\frac{1}{\sqrt{2}}$*, so the total scalar factor from all of these gates is* $\frac{1}{\sqrt{2}^n} = \frac{1}{q^{1/2}}$*. Therefore, the only concern is the total phase factor* $e^{2\pi i/n}$ *Note that the* $S_{j,k}$ *do not change the amplitude of any of the components, and only introduce a phase shift. There is thus only one way to change a bit* a_i*, and that is through the appropriate gate* H*. This adds a phase shift of* π *to the bits if they are both 1, and otherwise does not change the system. Further,* $S_{j,k}$ *adds a phase of* $\pi/2^{k-j}$ *to the phase of both* a_j *and* b_k *if and only if they are both 1. Thus, the phase on the path from* $|a\rangle$ *to* $|b\rangle$ *is:*

$$\sum_{0 \le j < n} \pi a_j b_j + \sum_{0 \le j < k < n} \frac{\pi a_j b_k}{2^{k-j}}$$

Where the first summation represents the phase shift from each Hadamard operator applied to the system and the second comes from the $S_{j,k}$ *gates. Note that the first summand is just the special case in the second where* $j = k$*. Therefore, we can rewrite the whole expression as:*

$$\sum_{0 \le j \le k < n} \frac{\pi a_j b_k}{2^{k-j}}$$

Recall that c was the bit-reversal of b. Therefore, we reexpress the summation in the following manner:

$$\sum_{0 \le j \le k < n} \frac{\pi a_j c_{n-1-k}}{2^{k-j}}$$

If we substitute i for $n-1-k$ *in the sum, we get:*

$$= \sum_{0 \le j+i < l} \frac{2\pi 2^j 2^i a_j c_i}{2^n}$$

Note that multiples of 2π *do not affect the phase of the overall system, and therefore we can ignore them for our purposes. Therefore, we need only sum over* j *and* i *less than* n*, obtaining:*

$$= \sum_{j,i=0}^{n-1} \frac{2\pi 2^j 2^i a_j c_i}{2^n} = \frac{2\pi}{2^n} \sum_{j=0}^{n-1} 2^j a_j \sum_{i=0}^{n-1} 2^i c_i$$

Recall that $q = 2^n$*,* $a = \sum_{j=0}^{n-1} 2^j a_j$*, and likewise for c. Therefore, the phase collapses to* $\frac{2\pi ac}{q}$*, which is precisely the phase ampitude for* $|a\rangle \rightarrow |c\rangle$*. Therefore, this actually is the quantum fourier transform, and as shown above this is a polynomial-time reversible quantum algorithm, thus concluding our proof.*

Because many of these $S_{j,k}$ are only adding a small phase factor when $k-j$ is large, implementing some of these gates would be challenging from a physical standpoint. Thankfully, these tiny phase factors can be neglected to approximate a Fourier transform of sufficient accuracy for factoring. Incidentally, this also will substantially speed up the algorithm, as most of the $S_{j,k}$ gates are removed. More can be found in the discussion in [1].

6 Shor's algorithm

Currently, the best classical algorithm for factoring large numbers is the number field sieve, which is $O(\exp(\log(n)^{1/3} log \log(n)^{2/3}))$ [3]. Here we introduce Shor's algorith, which will factor an integer in $O((\log(n))^2 \log\log(n) \log\log\log(n))$. What will be presented here will not actually be an algorithm for integer factorization, but instead will be an algorithm for finding the order of an element x in the multiplicative group, in other words finding the least r such that $x^r = 1 \mod (n)$. It is known that factorization can be reduced to finding the order of the element, and thus this algorithm is sufficient for a factorizing algorithm with only perhaps some polynomial-time classical proccessing, as described in [3]. Roughly speaking, factoring an odd integer n, given an algorithm for computing the order of a random integer $x \mod (n)$, can be done by finding its order r, and computing $\gcd(x^{r/2} - 1, n)$ using the Euclidean algorithm, which is in polynomial time. Since $(x^{r/2} - 1)(x^{r/2} + 1) = x^r - 1 \equiv 0 \mod (n)$, the $\gcd(x^{r/2} - 1, n)$ will only be a trivial divisor if r is odd or $x^{r/2} \equiv -1 \mod (n)$. We will now present a brief sketch of a proof that, when applied to random $x \mod (n)$, this procedure will produce a factor of n with a probability of at least $1 - \frac{1}{2^{N-1}}$, where n has N distinct prime factors. Consider $n = \prod_1^N p_i^{k_i}$, where p_i is prime, and let r_i be the order of $x \mod (p_i^{a_i})$. Then r is therefore the least common multiple of the r_i's. Now consider the largest power of 2 dividing each r_i. This algorithm fails if and only if these all agree. Note that if they are all 1, then r is clearly odd and $r/2$ is not an integer. If they are all equal and greater than 1, then $x^{r/2} \equiv -1 \mod (n)$ because $x^{r/2} \equiv -1 \mod (p_i^{a_i})$ for each i. Then, by the Chinese Remainder Theorem, choosing an $x \mod (n)$ at random is the same as choosing an $x_i \mod (p_i^{a_i})$ for all i, where $p_i^{a_i}$ is, as before, the ith prime factor of n raised to its respective power. The multiplicative group is cyclic for any odd prime power p^α, so for any odd prime power $p_i^{a_i}$ the probability is at most $\frac{1}{2}$ having any particular power of two as the largest divisor of its order r_i. Thus, each of these powers of 2 has at most a probability of $\frac{1}{2}$ of agreeing with the previous ones, so all N of them agree with a probability of at most $\frac{1}{2^{N-1}}$. Because each of these represents a failed selection, this implies that the chance that we select an appropriate x is at least $1 - \frac{1}{2^{N-1}}$. This works as long as n is odd and not a power of a prime, but mercifully there already exist algorithms that work classically to factor prime powers [3].

We now present a general proof for finding the order of $x \mod (n)$ on a quantum computer. Our computer will consist of two registers each containing integers represented in binary, as well as some workspace that will be reset to $|0\rangle$ at the end of each subroutine.
Suppose we are given some x and n. To compute the least r such that $x^r \equiv 1 \mod (n)$, first find the q such that $n^2 \leq q \leq 2n^2$ and that q is some integer power of 2. Note that from now on q, x, and n are constants and will not be modified for the duration of the algorithm.
Put the first register in the uniform superposition of states representing integers

$a \mod (q)$, which leaves our machine in the state:

$$\frac{1}{q^{1/2}} \sum_{a=0}^{q-1} |a\rangle\ |0\rangle$$

Where the $|0\rangle$ represents the empty second register. This leaves every bit in the first register in the superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.
Now, we can compute $x^a \mod (n)$ in the second register, which is still reversible because a is stored in the first register. This leaves our machine in the state:

$$\frac{1}{q^{1/2}} \sum_{a=0}^{q-1} |a\rangle\ |x^a \mod (n)\rangle$$

If we now apply the Fourier Transform A_q to register 1, as described previously, we map $|a\rangle$ to

$$\frac{1}{q^{1/2}} \sum_{c=0}^{q-1} e^{2\pi iac/q}\ |c\rangle\ .$$

Therefore, our machine is now in the state:

$$\frac{1}{q} \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} e^{2\pi iac/q}\ |c\rangle\ |x^a \mod (n)\rangle\ .$$

Now, we observe our machine, measuring both the $|c\rangle$ and $|x^a \mod (n)\rangle$ states. For given k, the probability of the machine ending in $|c, x^k \mod (n)\rangle$ is given by

$$\frac{1}{q} \left| \sum_{a:x^a \equiv x^k} e^{2\pi iac/q} \right|^2$$

Because the order of x has been computed to be r, the sum is over all a satisfying $a \equiv k \mod (r)$. Writing $a = br + k$:

$$\left| \frac{1}{q} \sum_{b=0}^{(q-k-1)/r} e^{2\pi i(br+k)c/q} \right|^2$$

Notice that $e^{2\pi ikc/q}$ is constant and has modulus 1. Consider $rc = \{rc\}_q$, where $\{rc\}_q$ indicates the residue that is congruent to $rc \mod (q)$ and is in the range $-q/2 < \{rc\}_q \leq q/2$. We seek to show that if we can make $\{rc\}_q$ small enough then we can bound the error in the measurement by $1/3$ as is specified by class $\mathbb{BQP}$ solutions. We can expand the sum into the following integral to obtain an upper bound on the probability:

$$\frac{1}{q} \int_0^{r/q(q-k-1)/r} e^{2\pi ib\{rc\}_q/q} db + O\left(\frac{|q-k-1|}{rq}(e^{2\pi i\{rc\}_q/q} - 1)\right)$$

Consider the error term on the right. Suppose that we allow $|\{rc\}_q| \leq r/2$. Then the entire error term is bounded from above by $O(1/q)$. If we consider the integral, then for $|\{rc\}_q| \leq r/2$ we will show that the integral is large, meaning that the pprobability of obtaining a state of the form $\left|c, x^k \mod (n)\right\rangle$ is large. Note that this condition is only dependent on c. Suppose $u = rb/q$, $du = r/qdb$:

$$= \frac{1}{r} \int_0^{(q-k-1)/q} e^{2\pi i \{rc\}_q u/r} du$$

Because $k < r$, approximating the upper limit by 1 results in only a $O(1/q)$ error. Therefore:

$$\frac{1}{r} \int_0^1 e^{2\pi i \{rc\}_q u/r} du$$

Suppose we allow $\{rc\}/r$ to vary between $1/2$ and $-1/2$. Then the integral can be minimized with $\{rc\} = \pm r/2$. In this case the absolute value of the integral is $2/(\pi r)$. Squaring this to find a lower bound on the probability, we find that the probability is bound by $\frac{4}{\pi^2 r^2} < \dfrac{1}{3r^2}$, for sufficiently large n. The probability of seeing $\left|c, x^k \mod (n)\right\rangle$ is bounded by $1/3r^2$ if $\dfrac{-r}{2} < rc - dq \leq \dfrac{r}{2}$. Dividing by rq gives us $\left|\dfrac{c}{q} - \dfrac{d}{r}\right| \leq \dfrac{1}{2q}$. c and q are known, and there is at most one d that satisfies this inequality. We can obtain d/r in lowest terms via continued fraction analysis on c/q [3]. If d found this way is relatively prime to r, we can thus compute r to complete the problem. There are $\phi(r)$ possibile values of d such that d is relatively prime to r. There are also r values of x^k, so there are $r\phi(r)$ ways of obtaining a satisfactory $\left|c, x^k \mod (n)\right\rangle$. Each of these states occurs with probability $1/3r^2$. Therefore, we get a sufficient state with probability $\phi(r)/3r$. Using a theorem of Hardy, $\phi(r)/r > \delta/\log\log(r)$ for some δ. Thus, with $O(\log\log(r))$ repititions, we can achieve a satisfactory probability of finding a sufficient state.

7 Quantum complexity classes

Currently, the relation between quantum complexity classes and their classical analogues is unknown. The practicallity of a quantum computer stems largely from its abillity to efficiently solve problems that are not thought to be solvable efficiently with a classical computer. If quantum computation cannot extend the class of functions that can be solved efficiently, then their use will most likely be largely relegated to mostly specialized and limited usage.
To begin our discussion of quantum complexity classes, we will begin with a brief overview of classical complexity classes. We make the following definitions in this section:

1. A problem is in $\mathbb{P}$ if it can be solved by a deterministic polynomial-time Turing Machine. These are the problems that are considered to be 'easy

enough' to compute efficiently on a Turing Machine. An important result of complexity theory is that primality testing is in $\mathbb{P}$.

2. A problem is in $\mathbb{NP}$ if it is solvable by a nondeterministic polynomial-time Turing machine. A nondeterministic polynomial-time Turing machine is allowed to have any number of paths to reach an answer, but at least one of these paths must result in a Turing Machine that accepts the input and terminates in polynomial time if the answer is yes, and every path must reject if the answer is no. We define a problem x to be in class $\mathbb{NP}$-COMPLETE $\subset \mathbb{NP}$ if, given any problem $p \in \mathbb{NP}$, p can be reduced to x with polynomial time and space modifications. In other words, an algorithm that can solve x in polynomial time can be used to show that all of $\mathbb{NP}$ can be solved in polynomial time.

3. Problem x has a *complement* $\overline{x}$ created by reversing the yes and no answers. For instance, consider the problem of primality testing. Then the complement is determining if a number is composite.

4. $\mathbf{co}-\mathbb{NP}$ is the class of problems x such that $\overline{x}$ is in $\mathbb{NP}$. It is generally considered to be highly unlikely that co-$\mathbb{NP} = \mathbb{NP}$. Therefore, if a problem is in $\mathbb{NP}$ and co-$\mathbb{NP}$-COMPLETE, then it is most likely not $\mathbb{NP}$-COMPLETE, because this would imply that $\mathbb{NP} =$ co-$\mathbb{NP}$.

The integer factorization problem is interesting in that there is no known polynomial-time solution, and yet it is not believed to exist in either $\mathbb{NP}$-COMPLETE or in co-$\mathbb{NP}$-COMPLETE. Because it is known to be $\mathbb{NP}$ and co-$\mathbb{NP}$, if it were to exist in either class this would imply $\mathbb{NP} =$ co-$\mathbb{NP}$, which is evidence to suggest that is not in either class. However, no efficient polynomial-time algorithm has been found. This leads some to suggest that the factorization problem instead lives in $\mathbb{NP}$-intermediate, a class of problems not thought to be in $\mathbb{NP}$-I (the I stands for intermediate), a hypothetical class of problems that are not in $\mathbb{P}$ or $\mathbb{NP}$-COMPLETE, which is non-empty if and only if $\mathbb{P} \neq \mathbb{NP}$. It would seem that the usefulness of quantum mechanical processing would extend naturally to at least a subset of $\mathbb{NP}$-I, if it were to exist. Note that integer factorization is not the only problem in $\mathbb{BQP}$ for which there is no known polynomial algorithm: the discrete logarithm is also shown in [3] to be in $\mathbb{BQP}$, which has also been proposed to exist in $\mathbb{NP}$-I. As the study of quantum computing is relatively young, even in comparison to complexity theory in general, it remains to be seen how far $\mathbb{BQP}$ can be extended. Ultimately, for $\mathbb{BQP}$ to see practicality in real-world computing problems, $\mathbb{BQP}$ would likely have to be extended to $\mathbb{NP}$-COMPLETE in order to be efficient for most computationally expensive and interesting problems.

8 References

1. D. Coppersmith, An approximate Fourier transform useful in quantum factoring, IBM Research Report RC 19642 (1994).

2. G. H. Hardy and E. M. Wright, An Introduction to the Theory of Numbers, Fifth ed.(1979), Oxford University Press, New York.

3. P. Shor, Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,SIAM J.Sci.Statist.Comput. 26 (1997) 1484

4. Wootters, William; Zurek, Wojciech. "A Single Quantum Cannot be Cloned". Nature 299 (1982): 802803