CSE 599d - Quantum Computing
Shor's Algorithm

Dave Bacon
Department of Computer Science & Engineering, University of Washington

I. FACTORING

> *The problem of distinguishing prime numbers from composites, and of resolving composite numbers into their prime factors, is one of the most important and useful in all of arithmetic... The dignity of science seems to demand that every aid to the solution of such an elegant and celebrated problem be zealously cultivated* — Carl Gauss

The efficient factoring of numbers is a problem that has attracted the attention of humankind for probably as long as we have contemplated numbers. Indeed, today the most widely used system used to communicate securely over the internet has its security based on the difficulty of factoring numbers efficiently. A great deal of effort has been spent trying to find classical algorithms to factor numbers. Indeed, probably more than we will ever know has been spent on this problem: the National Security Agency is supposedly the largest employer of mathematicians in the world and it would be reasonable to assume that they have spent a considerable amount of attention attempting to break the cryptosystems whose hardness is related to the hardness of factoring. Thus it was quite remarkable when, in 1994, Peter Shor showed that quantum computers could efficiently factor numbers. A warning that these notes are not as easy as our previous notes. The factoring algorithm has a lot of technical details which we will go through, and these details are easy for those who have had a good founding in discrete math and algorithms in number theory, but they aren't so easy for the rest of us mortals.

II. REDUCING FACTORING TO ORDER FINDING

The first step in Shor's factoring algorithm is to reduce the problem of factoring an integer N to the problem of order finding. Let's try to understand this reduction. We will assume, without loss of generality, that N is not even.

Suppose that we find a solution to the equation $x^2 = 1 \bmod N$ which is not one of the trivial solutions $x = 1$ or $x = N - 1$. This equation can be written as $(x + 1)(x - 1) = 0 \bmod N$. This implies that N must have a common factor with $(x + 1)$ or with $(x - 1)$. This common factor cannot be N because $x + 1$ and $x - 1$ are not equal to N (those were the trivial solutions.) Thus we find that a factor of N is either $gcd(x + 1, N)$ or $gcd(x - 1, N)$, where gcd is the greatest common divisor.

1. An Aside on Euclid's Algorithm

But how do we compute the gcd? An efficient method for computing the greatest common divisor goes all the way back to Euclid! It is based on the following observation: if $a > b$ are integers and r is the remainder when a is divided by b, then assuming $r \neq 0$, $gcd(a, b) = gcd(b, r)$. Why is this so? Well $a = qb + r$, which we can rewrite as $r = a - qb$. Thus whatever divides a and b must also divide r (and also divide b). Similarly we have that any divisor of b and r must also be a divisor of a (and also divide b.) Thus $gcd(a, b) = gcd(b, r)$. What about the case where $r = 0$? Well in this case you have just found that $a = qb$, and thus the gcd of these two numbers is b.

Euclid's algorithm works simply by repeatedly using $gcd(a, b) = gcd(b, r)$. Given $a > b$, we compute the remainder of a divided by b. If this is zero we are done (the gcd is b.) If not, we take b and r and compute the remainder r' of b divided by r. Using $gcd(a, b) = gcd(b, r) = gcd(r, r')$, if r' is zero, then we know that $gcd(a, b) = r$. If not, we can continue in this manner until eventually the remainder is zero. This is Euclid's algorithm. This process is always guaranteed to terminate, since each remainder will be smaller than the number we divide by.

What is the running time of this algorithm? A good way to get a bound is to notice that the remainder is always smaller than the number being divided by a factor of two. Suppose you are dividing a by b. Then if b is less than or equal to $a/2$, the remainder must be less than $a/2$. If b is greater than $a/2$, then the it will only divide b once and the remainder will be less than $a/2$. Thus the number of divisions we will need to perform in Euclid's algorithm is bounded by $\log_2 a$. Thus we see that Euclid's algorithm is efficient.

2. Back to Shor

So now we've seen that if we can find $x^2 = 1 \bmod N$, which is not $x = 1$ or $x = N - 1$, then we can factor N. Suppose we pick a random number y between 1 and $N - 1$. We can check whether $gcd(y, N)$ is equal to one or not. If it is not equal to one, then we have found a factor of N. If it is equal to one, then y is coprime with N. The order of y is the smallest integer r such that $y^r = 1 \bmod N$ (recall that the group of numbers coprime with N forms a group and importantly for us has a multiplicative inverse.) Now it might happen that r is even. If this is the case then we have found a solution, $x = y^{r/2}$ to $x^2 = 1 \bmod N$. Thus if we can show that the probability of a random coprime number y having an even order is high, then we see that we have reduced the problem of factoring to the the problem of finding the order of a number.

So is it true that a randomly chosen coprime y will have an even order modulo N? (The details of this calculation were taken from Nielsen and Chuang "Quantum Computation and Quantum Information" but all mistakes are mine!)

Let's begin to answer this consider this question for N equal to an odd prime. $N - 1$ is then an even number. Let 2^m be largest power of two which divides $N - 1$. The group of numbers coprime to N forms a cyclic group: that is every such number can be written as $g^t \bmod N$ for some generator g. Call r the order of g^t modulo N. If t is odd, then since $g^{tr} \bmod N = 1$ implies that tr divides $N - 1$ and t is odd, r is divisible by 2^m. If t is even, then since $g^{\frac{t(N-1)}{2}} = (g^{N-1})^t = 1 \bmod N$, this implies that r divides $\frac{N-1}{2}$ and hence that 2^m does not divide r. Thus we have shown that if N is an odd prime, and 2^m is the largest power of two which divides $N - 1$, then exactly half of the numbers coprime to N will have an order which is divisible by 2^m and half will not.

Okay this is a great start, but we aren't interested in the N is prime case. We are interested in general N. How do we fix this? First consider the case where N is a power of an odd prime number $N = p^\alpha$. Then the same argument we used in the previous argument can be repeated, but this time with $N - 1$ replaced by $N^{\alpha-1}(N - 1)$. Thus we have shown that if N is a power of an odd prime, and 2^m is the largest power of two which divides $N^{\alpha-1}(N - 1)$, then exactly half of the numbers coprime to N will have an order which is divisible by 2^m and half will not.

Now we can go for the big one and talk about the general case of a number which is a product of powers of an odd prime $N = p_1^{\alpha_1} \cdots p_s^{\alpha_s}$. Now the group of numbers coprime to N, $\mathbb{Z}_N^*$, is equal to the direct product of the prime factor cyclic groups of this number: $\mathbb{Z}_N^* = \mathbb{Z}_{p_1^{\alpha_1}} \times \cdots \times \mathbb{Z}_{p_s^{\alpha_s}}$. Choosing x randomly with uniform probability over numbers coprime to N is equivalent to choosing x_i randomly with uniform probability over numbers coprime to $p_i^{\alpha_i}$ such that $x = x_1 \ldots x_s$. Let r_i be the order of x_i modulo $p_i^{\alpha_i}$. Then we have already seen that with probability one half, the numbers coprime to $p_i^{\alpha_i}$ have an order which is even, and in fact is a multiple of 2, 2^{d_i}. Consider the case where r is odd. Then each r_j must be odd. This case occurs with probability 2^{-s} from our previous results. Further we need to worry about the case where r is even but $x^{r/2} = -1 \bmod N$. Then $x^{r/2} = -1 \bmod p_i^{\alpha_i}$, so r_j does not divide $r/2$. But this implies that all of the d_i must be equal. Why? Because r_j all divide r and in fact r is a common multiple of the r_j, so if one of the d_i has an extra power of 2, then removing this the other should still divide $r/2$. But this is what we have shown is not true. So this implies that all of the d_i must be equal. This will happen with probability at most 2^{s-1}. Thus we have shown that the probability that a random number x coprime to N, then the probability that the order of this number is even and does not satisfy $x^{r/2} = -1 \bmod N$ is a least

$$Pr \geq 1 - \frac{1}{2^{s-1}} \tag{1}$$

So are we done in showing that order finding can be used to factor? Almost. We have to worry about the case where m is a prime to a power: $N = p^\alpha$, $\alpha \geq 2$. Actually there is an efficient *classical* algorithm for this case. In this case, α will be less than $\log_2 N$. Thus we can basically try all possible α's with only linear overhead. How do we do this? Work with a fixed α. First compute $\log_2 N$ with enough precision (what this efficiency needs to be will be clear from the rest of the algorithm.) Then divide this number by b and compute the two integers nearest to 2^x, where x is $\log_2 N$ divided by b. This gives us the two integers nearest $2^{\log_2 N/\alpha} = N^{1/\alpha}$. We can then check whether either of the integers when taken to the αth power is N. All of the manipulations we have described can be done in $O(\log_2^2 N)$ operations. Cycling over α we obtain a $O(\log_2^3 N)$ algorithm for the case of $N = p^\alpha$.

So we've done it. We've showed that if you can solve the order finding problem, then you can factor. Now this is good: we've reduce the complicated number theoretic task of factoring to a task which is very different from the original task. A task that a quantum computer can sink its teeth into.

III. ORDER FINDING

The order finding problem is as follows. We are given positive integers x and N with $x < N$ and x coprime to N (they share no common factors.) Then the order finding problem is to find the smallest positive integer r such that $x^r \bmod N = 1$. r is called the order of x in N.

A quantum algorithm for order finding works in two steps. One is to make a phase estimation problem and the second is to use a continued fraction algorithm. We will begin by understanding the phase estimation algorithm.

Consider the following unitary transform on n qubits, $2^{n-1} < N \leq 2^n$

$$U|y\rangle = \begin{cases} |xy \bmod N\rangle \text{ if } 0 \leq y \leq N-1 \\ |y\rangle \text{ otherwise} \end{cases} \tag{2}$$

Now I have said that this transform is unitary. Why is this so? Well since x is coprime to N, x has a multiplicative inverse modulo N. This means that the transform described above is a permutation matrix (a matrix with only zeros and ones and with the condition that there is only one one per column and per row.) All permutation matrices are unitary.

What are the eigenstates of U? They are given by

$$|v_t\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i k t}{r}\right] |x^k \bmod N\rangle \tag{3}$$

To check that they are indeed eigenstates and find their eigenvalues, we calculate

$$\begin{aligned} U|v_t\rangle &= U\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i k t}{r}\right] |x^k \bmod N\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i k t}{r}\right] |x^{k+1} \bmod N\rangle \\ &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i (k-1) t}{r}\right] |x^k \bmod N\rangle = \exp\left[\frac{-2\pi i t}{r}\right] |v_t\rangle \end{aligned} \tag{4}$$

Now we see that the eigenvalues of $|v_t\rangle$ contain information about the order r. Thus we would like to perform phase estimation for these eigenstates using the phase estimation algorithm. But there is an immediate problem. How do we prepare $|v_t\rangle$? Well we can't do this directly. But what we can do is prepare a superposition of such states:

$$\frac{1}{\sqrt{r}} \sum_{t=0}^{r-1} |v_t\rangle = \frac{1}{\sqrt{r}} \sum_{t=0}^{r-1} \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i k t}{r}\right] |x^k \bmod N\rangle \tag{5}$$

which, using $\sum_{t=0}^{r-1} \exp\left[\frac{-2\pi i k t}{r}\right] = r\delta_{k,0}$ becomes,

$$\frac{1}{\sqrt{r}} \sum_{t=0}^{r-1} |v_t\rangle = |1\rangle \tag{6}$$

Now what happens if we feed in a superposition of different eigenstates into the phase estimation algorithm? Well with one run of the algorithm we obtain an estimate of the eigenvalue of the different eigenstates with a probability given by the amplitude of the different eigenstates squared. Thus if we run phase estimation on $|1\rangle$, we will obtain, on one run an estimate for the eigenvalue $\frac{t}{r}$, for uniformly random $0 \leq t \leq r-1$. This is good because this will help us find r!

But can we really perform the phase estimation algorithm? To do this we need to be able to prepare $|1\rangle$, which is pretty darn simple. But what about the controlled U^{2^k} operations? This also can be done and is performed using a procedure called modular exponentiation. There are many fascinating ways to do this. Here I will discuss a straightforward, if wasteful method. The transform we wish to enact is

$$|z\rangle|y\rangle \to |z\rangle U^z |y\rangle = |z\rangle|yx^z\rangle \tag{7}$$

We can factor this as

$$|z\rangle|y\rangle \to |z\rangle U^{z_1 2^{t-1}} U^{z_2 2^{t-2}} \cdots U^{z_n 2^0} |y\rangle \tag{8}$$

Thus we see that we need to figure out how to implement a controlled U^{2^k} gate. We note, without further comment, that there exists classical circuits which square modulo N a number x (these may be irreversible, we will use the tricks we learned before to make these reversible.) Thus we can compute x^2 modulo N by squaring x, then calculate x^4 modulo N by squaring x^2, etc. This allows us to calculate x^{2^j} modulo N using only j such squarings. Then we can copy this result (by multiplication, not addition modulo N onto the register we wish to multiply) and get rid of

the garbage we previously produced. Using our grade school multiplication techniques, we see that this will require $j(\log N)^2$ multiplications. Doing this for each of the n conditional operators will produce the desired transforms in $O(\log^3 N)$ multiplications.

Thus we see that all the tools we need for an efficient quantum algorithm for using the phase estimation algorithm in this way are in place. What will be the outcome of this algorithm? Well we will obtain a superposition of the form

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |s/r\rangle \tag{9}$$

When we measure this register, we will therefore obtain a random s/r for $s = 0$ to $r - 1$. How can we use this to find r?

Well we know that s and r are integers which are bounded integers. Thus we know that s/r is really a rational number. At the end of our phase estimation algorithm we obtain an *estimate* for s/r. If this estimate is good enough, then we may be able to find s/r. We do this using the continued fraction algorithm.

3. Continued Fraction Algorithm

A continued fraction is the representation of a real number by a sequence (possible infinite) of integers:

$$[a_0, a_1, \ldots, a_n] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\cdots + \cfrac{1}{a_n}}}} \tag{10}$$

where the a_i are positive integers. If allow the first a_1 to be zero, then any positive rational number will have a continued fraction expansion which converges. Given a positive rational number we can find this expansion by using the continued fractions algorithm. The continued fractions algorithm, actually is just a version of Euclid's algorithm! How does this work? Well suppose our rational number is $\frac{p}{q}$. Then we can apply the first step of Euclid's algorithm and obtain $\frac{p}{q} = (p \text{ div } q) + (p \bmod q)/q$, where div stands for integer division without remainder. p div q and $p \bmod q$ are exactly what we compute in the first step of Euclid's algorithm. Now we pull a fast one and invert the second term

$$\frac{p}{q} = (p \text{ div } q) + \cfrac{1}{\cfrac{q}{p \bmod q}} \tag{11}$$

Continuing on in this manner, we can now take the $\frac{q}{p \bmod q}$ term and do the same thing on it. Thus we see that we are running Euclid's algorithm, but keeping around the remainders and the divisors. By the same reasoning that Euclid's algorithm converged, we also know that this algorithm will converge and that it will converge quickly.

Now suppose that we have a rational number x which is an approximation to some reduced fraction $\frac{p}{q}$. How close does this rational number need to be in order to guarantee that the when we perform the continued fractions expansion on x, one of the sequences of continued fractions will contain the continued fraction expansion for $\frac{p}{q}$? Well suppose that $x = \frac{p}{q} + \frac{\delta}{2q^2}$ with $|\delta| < 1$. Now when we perform the continued fraction expansion for x we will obtain the continued fraction for $\frac{p}{q}$ as part of this expansion. Why does this work? Well start to run the continued fractions algorithm on $x = \frac{2pq+a}{b2q^2}$, where we have expressed δ as a rational fraction, $\delta = \frac{a}{b}$. Then this the divisor is $2pqb + a$ div $2q^2b$. Since $a < 2q^2b$, this will be just be $2pq$ div $2q^2 = p$ div q. Thus we see that at this point the the algorithm yields the same continued fraction expansion as for p/q. The modulo left over from this division will be $2pqb + a \bmod 2q^2b$. The next division will thus be $2q^2b$ div $(2pqb + a \bmod 2q^2b)$. This will be equal to q div $(p \bmod q)$ if a times this answer is less than $2q^2b$. When will this fail? Recall that the values in the continued fraction are decreasing exponentially fast. Our answers will be decreasing exponentially and will terminate when after $\log_2 q$ times for the exact p/q algorithm. Starting with $2q^2b$ and cutting this in half every iteration we end up with $2qb$ at this $\log_2 q$ iteration. It is at this point, our next step will not necessarily satisfy the inequality a less than this value $2qb$. Thus we see that our algorithm will successful for $\log_2 q$ iterations.

4. To the Finish Line

Okay, so now are we done? Well we need to show that we can obtain, using the phase estimation algorithm a detailed enough approximation of s/r. If you recall from our discussion of the phase estimation algorithm, if we use

n bits for the QFT part of the phase estimation algorithm, then the probability of getting a phase which differs from the correct phase by $\frac{c}{2^n}$ is bounded by

$$Pr(c) \leq \frac{1}{4c^2} \tag{12}$$

We can use this to bound the total probability that our algorithm will fail to obtain a m bit approximation of the true phase.

$$Pr(fail) \leq 2 \sum_{c=2^{n-m}}^{2^{n-1}} Pr(c) \leq \frac{1}{2} \sum_{c=2^{n-m}}^{2^{n-1}} \frac{1}{c^2} \leq \frac{1}{2} \int_{2^{n-m}-1}^{2^{n-1}} \frac{1}{c^2} dc \leq \frac{1}{2^{n-m+1}} \tag{13}$$

Thus if we want to obtain an m bit approximation to the true phase, and want the algorithm for fail with probability at most ϵ, we should choose n by requiring that

$$\epsilon = \frac{1}{2^{n-m+1}}. \tag{14}$$

This in turn implies that if we use

$$n = m + \log_2 \frac{1}{2\epsilon} \tag{15}$$

then we will obtain an algorithm which fails with probability at most ϵ and produces an m bit approximation to the true phase.

Okay, so what does this mean for our quantum phase estimation for the order finding problem? Suppose that we obtain a $m = 2 \log N + 1$ approximation to s/r. Then $|s/r - x| \leq \frac{1}{2^m} = \frac{1}{2N^2}$. This in turn implies that we have satisfied the continued fraction convergence requirement described above (because r is bounded by N.)

Thus we have all of the pieces put together. We use order finding to produce an estimate of s/r. We can make the precision of this estimate high enough that our continued fraction will contain a convergent of s/r. Now there is still one way that the algorithm might fail. This is that s and r may share a common factor. Then the continued fraction algorithm will return the reduced r and not r itself. How do we deal with this? One way to deal with it is to simply note that the fraction of number of prime numbers less than r is at least $\frac{r}{2 \log r}$. This implies that if we run the algorithm $O(\log r) = O(\log N)$ times, we will obtain an s which is coprime with r and thus the continued fraction algorithm will succeed. This is a rather wasteful way to do this, but it works. A better way is to run the algorithm twice, obtain two r's and s's and then to take the least common multiple of the two r's. Provided that the two s's share no common factors, this r will be the correct r. One can show that this probability for random s's is greater than 1/4.

IV. BREAKING RSA CRYPTOGRAPHY WITH QUANTUM COMPUTERS

Why should we care about efficient algorithms for factoring? A very good reason is that such efficient algorithms can be used to break many public key cryptosystems, and in particular the public key cryptosystem known as RSA.

RSA has an interesting history. In 1974, Diffie and Hellman, came up with a public key systems for key exchange. Then in 1978, Rivest, Shamir, and Adleman came up with the RSA public key encryption algorithm. This was the history most of us knew, up until a few years ago, when it was revealed that British mathematicians working for the British intelligence agency GCHQ actually had invented these protocols a few years earlier. Indeed Clifford Cooks in 1973 invented what is essentially the RSA cryptosystem and his friend Malcolm Williamson invented what is essentially the Diffie-Hellman cryptosystem sometime in 1973. What is amazing about these early derivations was that they came up with pretty much exactly the same schemes! Now that, is strange.

How does the RSA public key cryptosystem work? Consider two parties, Alice and Bob, who wish to communicate securely such than an eavesdropper cannot figure out the message that Bob is communicating to Alice. To do this in the RSA protocol, Alice generates two large primes at random, p and q. She then computes $N = pq$ and $\varphi = (p-1)(q-1)$. φ is equal to the number of numbers coprime to N and is called the Euler φ function. Alice then choosing a random number e coprime with φ. Alice then computes d such that $de = 1 \bmod \varphi$. The public key, which Alice can reveal to the world, is N and e. The private key is d. Now to exchange a message, Alice reveals her public key to Bob. Bob then takes his message n and computes $cn^e \bmod N$ using his message and Alice's public key. He then sends this value c to Alice. Alice can now figure out the message n by taking c to the dth power modulo N. Then $c^d \bmod N = n^{ed} \bmod N$.

We will now show that this expression yields the value of the message, n. We know that there is an integer k such that $de = k\varphi + 1$. Let $x = k\varphi$. Then $n^{k\varphi} \bmod p = (n^{p-1})^{k(q-1)} \bmod p$ and by Fermat's little theorem this is equal to one: $n^{k\varphi} \bmod p = 1$. Similarly $n^{k\varphi} \bmod q = 1$ modq. By the Chinese remainder theorem, this implies that $n^{k\varphi} \bmod pq = 1$. Thus we see that$n^{ed} \bmod N = n^{k\varphi+1} \bmod N = n \bmod N$.

Now why is it hard for an eavesdropper to crack this code? Well the eavesdropper knows N and e and c, but not d. Thus the eavesdropper cannot act like Alice in decoding the message. Further it is thought that it is very difficult, from a computational complexity standpoint, to identify d from N and e. The exact computational difficulty of this problem is not known, but it is widely suspect to be hard for classical computers. But it is also known that if you can efficiently factor, then the problem is easy. Why? If you can efficiently factor, you can determine p and q in $N = pq$. This then allows you to act just like Alice: you can computer φ and then use e to calculate $de = 1 \bmod \varphi$. Then the eavesdropper can decrypt in the same way that Alice decrypts the message sent from Bob to Alice. Thus efficiently factoring breaks this public key cryptosystem!

So what does this all mean? Well it means that quantum computers seem to change the security of the RSA cryptosystem. In fact it is very hard to find public key cryptosystems these days which are not broken by quantum computers. This has led some to conjecture that there are no public key cryptosystems which are secure against attacks from quantum computation. The answer to this question is one of the motivations many in the quantum computing community have for examining these unbroken public key cryptosystems and trying to understand the hardness of these cryptosystems for quantum computers. Now of course, it could be that there is an efficient classical algorithm for factoring. This would be an amazing result, but it is totally within the realm of possibility. Should this keep us from studying quantum computation? No one can answer this question right now, because we don't know the exact relationship between the complexity of quantum computers and the complexity of classical computers. But there the deeper we dig into quantum computing, the more we find that quantum computers do have advantages over classical computers. For example, we know that there are settings in communication complexity where we have provable exponential separations between the classical world and the quantum world. This doesn't prove the separation between quantum and classical, but it does make us more and more suspicious. In a similar manner, algorithms like Simon's algorithm, make us more suspicious. At some point we may well know the answer to whether quantum computers are more powerful than classical computers for doing things like factoring, but until an efficient classical algorithm for factoring is found, we should certainly study quantum computing.

Finally, we should note that the public key cryptosystems broken by quantum computers are used fairly ubiquitously in classical computer science today. It may seem a sort of low level matter whether this is true. Some people, especially those whose job it is to be paranoid about security, have a different view on this matter. Here is a quote from Simon Singh which I think perhaps best summarizes this point of view:

> *Only time will tell if and when the problems of building a quantum computer can be overcome.As information becomes the worlds most valuable commodity, the economic, political and military fate of nations will depend on the strength of ciphers. Consequently, the development of a fully operational quantum computer would imperil our personal privacy, destroy electronic commerce and demolish the concept of national security. A quantum computer would jeopardize the stability of the world. Whichever country gets there first will have the ability to monitor the communications of its citizens, read the minds of its commercial rivals and eavesdrop on the plans of its enemies. Although it is still in its infancy, quantum computing presents a potential threat to the individual, to international business and to global security.* - Simon Singh