Quantum Computing Notes
Prof. Hamzeh Roumani
EECS, York University, Toronto, Ont., Canada

THE SHOR ALGORITHM

1. The Problem

Given an integer N > 1, factor it. If N is prime, the problem becomes trivial (only 1 and N divide N) so let us add "N is not prime" to the promise (note that this is easy to check since primality is in P). In addition, let us focus on finding just one non-trivial factor (i.e. neither 1 nor N) because once we find such a factor, we divide N by it to get a second, and if either is not prime, we use the same approach to factor it, and this ultimately leads to all the prime factors of N. Hence, our problem becomes: *Given a non-prime integer N>1, find any non-trivial factor of it*.

This easy-to-state problem has the entire security of the Internet resting on its difficulty! The most commonly used encryption algorithm on the web is RSA, and it involves a publicly known integer that is equal to the product of two secret primes. Factoring it leads to discovering these two primes, and thus, to breaking the encryption. In the context of RSA, N involves hundreds of digits (thousands of bits).

2. Examples

N = 67,893 → 21

N = 172,453 → 5563

N =695,681,049,241→ 771,401

3. A Classical Algorithm

The non-trivial factors of N must be in [2, N-1], so we can check them, one-by-one to see if any divides N (the check can be done efficiently by computing the GCD of N and the candidate factor). This approach grows linearly with N, so it is exponential in n, the number of bits in the binary representation of N. You can speed up the process by narrowing the range to [2, √N], but regardless, the problem has a $O(2^n)$ complexity. A randomized algorithm can sample this range, but to no avail: the complexity remains exponential.

- The Euclidean algorithm for GCD is based on GCD(x, y) = GCD(y, x mod y), where x > y > 0. Use your favorite programming language to implement this algorithm and test it. What is the complexity in terms of n, the number of bits in the representations of x and y.
- Suppose that N is a product of two primes and that we sampled the [2, N-1] range. What is the probability of failing to find a factor after k sample points? Estimate the value of k that makes this probability less than 50%.

4. The Algorithm at a Glance

Follow these steps to find a factor of N:

1. **Generate a random integer a in [2,N–1].**
   This integer will be used as a base to be raised to an exponent.
2. **If a shares a factor with N (i.e. GCD(N,a) ≠ 1), we are done!**
   In this case the GCD is the sought factor of N.
3. **We now have: GCD(N,a) = 1.**
   This is almost always the case–astronomically unlikely to stumble on a factor of N.
4. **Consider the function *f(x) = a^x mod N,* where x is a positive integer.**
   The values of this function are all in [0,N–1].
5. **Find the value(s) of x at which *f* is 1, and let r be the smallest such value.**
   f has to be 1 at some value of x (by Euler theorem).
6. **If r is odd, the algorithm fails. Restart by going back to Step #1.**
   It can be shown that r will be even with probability ≥ ½.
7. **Since *$a^r = 1$,* then *$(a^{r/2} – 1)(a^{r/2} + 1) = 0$* (all modulo N).**
   We have two numbers multiplying to 0, i.e. to a multiple of N.
8. **The first number cannot be 0 (a multiple of N).**
   Because this would contradict r being the smallest exponent that leads to 1.
9. **If the second number is zero, the algorithm fails. Restart by going back to Step #1.**
   It can be shown that $a^{r/2} + 1 \neq 0$ (mod N) *with probability ≥ ½.*
10. **Since both numbers are not zero yet their product is, we are done!**
    GCD($a^{r/2} \pm 1$, N) are factors of N.

Interestingly, only Step#5 needs a quantum computer; the remaining steps can easily be done on a classical computer. As we shall see, this step is called "period finding".

- Let N = 68,911, which is a product of two primes 137 and 503. Present a quantitative argument to show that it is unlikely to guess a such that GCD(N,a) ≠1.
- For the following questions, assume that N=15 and a=2
- Tabulate the function *f(x)* above for all values of x in [1,N-1].
- How many values of x map to 1?
- Determine the value of r. Is it even?
- Compute $a^{r/2} + 1$ mod N. Is it 0 modulo N?
- Based on your findings and on the above algorithm, what are the factors of N?

5. The Quantum Advantage

Finding the integer r in Step#5 is as difficult a problem as finding a factor of N because r can be any integer in [2,N-1], which is the same range in which we look for factors. We can turn the search for r into a search for a global property of a function (something quantum computers do very well) as follows: since r yields 1 when used as an exponent, we can write:

$$\forall\, x > 0:\; f(x+r) = a^{x+r} mod\ N = a^{x}\ mod\ N \times a^{r} mod\ N = a^{x}\ mod\ N = f(x)$$

Hence, the existence of r makes *f* a periodic function with period r. Quantum computing has proven extremely powerful in determining global properties of functions, periodicity in this case. Indeed, Simon's algorithm used constructive and destructive interference to determine a property of a bijective function *f*, namely: the value of s that makes *f(x⊕s) = f(x)*. This equality bears strong resemblance to the above *f(x+r)=f(x)* except that the operation is bitwise xor in the former and integer addition in the latter. Indeed, Simon's algorithm was a source of inspiration for Shor's. In conclusion, note that the quantum aspect of integer factorization is period finding.

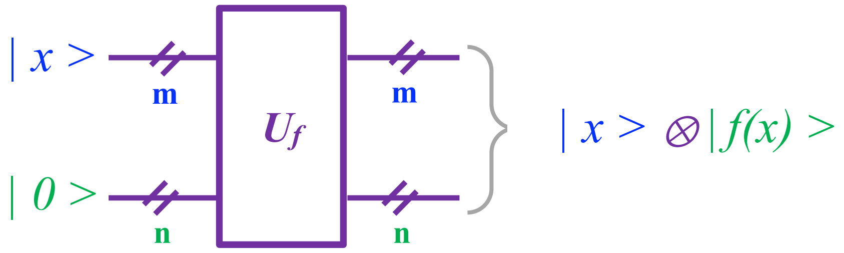
To get a feel for this function, let us tabulate it for a few values of x. Specifically, look at N=21. The base, a, must be coprime with N so, aside from the trivial a=1 case, it can take 11 different values. The table below tabulates the function at values of x in [2,N-1] for each possible choice of a, one row per choice. Examine the rows and observe the patterns:

a^x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8	16	11	1	2	4
4	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16
5	5	4	20	16	17	1	5	4	20	16	17	1	5	4	20	16	17	1	5	4
8	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1
10	10	16	13	4	19	1	10	16	13	4	19	1	10	16	13	4	19	1	10	16
11	11	16	8	4	2	1	11	16	8	4	2	1	11	16	8	4	2	1	11	16
13	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1
16	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4
17	17	16	20	4	5	1	17	16	20	4	5	1	17	16	20	4	5	1	17	16
19	19	4	13	16	10	1	19	4	13	16	10	1	19	4	13	16	10	1	19	4
20	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1

Note that the function is periodic in all rows. The period is 6 in six of the rows, 3 in two, and 2 in three. Also note that in a row with period p, we have $a^p \equiv 1$. Finally, note that when p is even and $a^{p/2}$ is not -1, then $a^{p/2} \pm 1$ do indeed contain factors of 21.

6. Period Finding: The Idea

We need to find the period of a function *f(x)=a^x mod N* given an oracle U_f that implements it in hardware. The domain of *f* is all the positive integers, and its codomain is [0,N-1]. To detect the period, the circuit must sample *f* at as many values of x as possible in the domain. To that end, let us make x m-qubit wide (which allows it to take classical values in [0,M-1], where $M=2^m$. As to the control input, it must have the same width as *f* (because the two get xor'ed), so it needs to be n-qubit wide, where n is the number of bits in N (n = 1 + floor of lgN). We will see later that an accurate estimate of the period obtains when $M \approx N^2$, which means m≈2n. We thus have m+n inputs and m+n outputs, and we will set all n control qubits to zero:



If we feed the oracle an equal superposition of all possible x values in [0,M-1], the output would be a superposition of 2^m states each of which has a value for x entangled with a corresponding value for *f(x)*:

$$\sum_{x=0}^{2^m-1} |x > \otimes |f(x) >$$

If we now measure the lower output |*f(x)*>, it will collapse to one of the values of *f*, say *F*, and the entangled upper output |x> will collapse to an equal superposition of all values of x that are consistent with *F*, i.e. values for which *f(x)=F*. Since *f* is periodic with period r, the upper superposition will involve values of x that are r apart, i.e. the upper output collapses to:

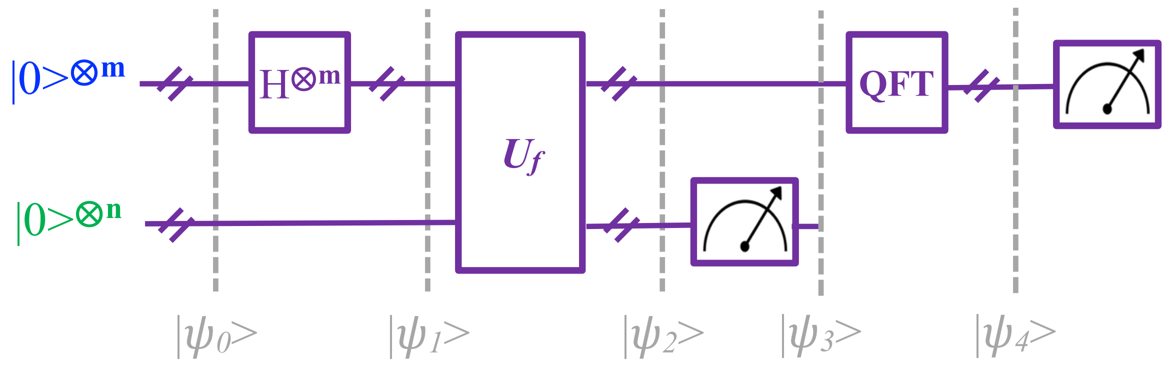
$$|x_0 > + |x_0 + r > + |x_0 + 2r > + \cdots + |x_0 + (M/r - 1)r >$$

where *f(x$_0$)=F*. Note that the problem is done in the "quantum world" because if we can "see" this state, we can easily "see" r. Unfortunately, this is not the case in "our world" because all we can do is measure, and once we do, the state will collapse to one of the terms, and one term does not determine r. Cloning the state is not allowed, and repeating the entire process two times (in the hope of seeing two terms and determining r from their difference) doesn't help because the two attempts may collapse to a different *F*, and thus a different x_0.

We therefore need to manipulate the state before we measure. As in the Apple Gedanken, we add a "merge" on the upper output (a quantum Fourier transform) to allow the components to interfere and expose the period.

7. Period Finding: The Algorithm

The input consists of m+n qubits, m in the upper register and n in the lower, all set to |0>. We create a superposition in the upper input thru Hadamard gates. The gate's lower output is then measured. The upper output undergoes a quantum Fourier transform before we arrive at the final upper output, which gets measured.



$$|\psi_0> = |000 \ldots 000> \otimes |000 \ldots 000>$$

$$|\psi_1> = \frac{|0> + |1>}{\sqrt{2}} \otimes \frac{|0> + |1>}{\sqrt{2}} \otimes \ldots \otimes \frac{|0> + |1>}{\sqrt{2}} \ldots \otimes |000 \ldots 000>$$

$$|\psi_1> = \frac{1}{\sqrt{M}} \sum_{k=0}^{M-1} |k> \otimes |000 \ldots 000>$$

$$|\psi_2> = \frac{1}{\sqrt{M}} \sum_{k=0}^{M-1} |k> \otimes |f(k)>$$

If we now measure the lower register of |*ψ₂*>, |*f(k)*> will collapse to some value *F* and the entangled upper register |k> will collapse to values compatible with *F*, i.e. values of x that map to *F*. Since *f* is periodic with period r, there are M/r such values in [0,M-1] (as they are r apart):

$$|\psi_3> = \frac{1}{\sqrt{M/r}} \sum_{k=0}^{M/r-1} |x_0 + kr> \otimes |F>, \text{where } f(x_0) = F$$

As indicated in the previous section, performing a measurement of the upper register at this stage is not helpful because we need at least two terms to discern the period r, and repeating the whole process is not guaranteed to have the same x_0. That's why we resort to Fourier:

The Fourier transform is a well-established technique for detecting and exposing periodicity. The most common application involves a signal that changes in time in a periodic way. Taking the transform of that signal leads to a function that peaks around the periods of the signal, thus exposing them. The Quantum Fourier Transform (QFT) is a linear, unitary gate which, acting on an m-qubit input |x>, it produces the following m-qubit output (with $M=2^m$):

$$QFT\,|x> = \frac{1}{\sqrt{M}} \sum_{y=0}^{M-1} \phi_y |y> \quad \text{where } \phi_y = e^{2\pi i x y/M}$$

Applying this to the upper m qubits in ψ_3 (i.e. to the $|x_0 + kr>$ ket) yields:

$$|\psi_4> = \frac{1}{\sqrt{M/r}} \times \frac{1}{\sqrt{M}} \sum_{y=0}^{M-1} \phi_y |y> = \frac{\sqrt{r}}{M} \sum_{y=0}^{M-1} \phi_y |y>$$

$$\text{where } \phi_y = \sum_{k=0}^{M/r-1} e^{2\pi i (x_0+kr)y/M} = e^{2\pi i x_0 y/M} \sum_{k=0}^{M/r-1} e^{2\pi i k r y/M}$$

How does QFT expose periodicity? The QFT output ψ_4 looks like a superposition of M terms, but upon closer examination, we can see that many of them are zero (interfered destructively). The only non-zero terms (constructive interference) are those that correspond to the period r of ψ_3.

Indeed, the terms in which y is a multiple of M/r, the coefficient ϕ_y = M/r (because 2πikry/M in the final sum would be a multiple of 2πi, which leads to a sum of 1's, which yields M/r). Clearly, there are r such terms in the sum, and hence:

$$|\psi_4> = \frac{\sqrt{r}}{M}\left(\sum_{y|M/r} \frac{M}{r} |y> + \sum_{y\nmid M/r} \phi_y |y> \right) = \frac{1}{\sqrt{r}} \sum_{y|M/r} |y> + \frac{\sqrt{r}}{M} \sum_{y\nmid M/r} \phi_y |y>$$

Note that the first term is normalized, and since ψ_4 is normalized, the last term must be zero.

If we now measure ψ_4, it will collapse randomly and uniformly to one of the y states in which y is some multiple of M/r, which gives us some information about r. To learn more, we repeat the entire process to get another multiple of M/r. After collecting enough distinct multiples, we should be able to compute M/r and thus r.

The above discussion glanced over a point: what if M/r is not an integer, i.e. the chosen range for x does not contain a whole number of periodic runs? In this case, the Fourier transform will have small spreads around multiples of M/r rather than being sharply peaked, and hence, the obtained multiples of M/r will be approximate. It can be shown that choosing $N^2 \le M < 2N^2$ will ensure that the measured state will, with high probability, be within ½ of a multiple of M/r.

Finally, we need to estimate the number of times we need to repeat the entire process before the period can be determined. The problem is that each measurement gives a multiple of M/r, not M/r itself. For example, if M/r = 7 and the measurements yielded 14 and 21 then it is clear that M/r must be 7. But if the measurement yielded 14 and 28 then M/r could be 2 or 7. It can be shown that by using continued fractions, one can determine M/r, and thus r, unambiguously after at most lg(lgr) steps. The oracle complexity of our algorithm is thus O(lglgN) (because r is less than N), which is O(lgN), which is linear in the number of bits in N.

- Argue that the intermediate measurement of the lower register is not needed.
  Hint: See the appendix in the notes on Simon's Algorithm.
- Show that terms in ψ_4 in which y is not a multiple of M/r vanish (i.e. have $\phi_y = 0$).
  Hint: This was shown above using a normalization argument. This question asks you to show it directly.

Remarks

- This algorithm was proposed by Peter Shor c. 1994.
- All pre and post quantum processing is done on a classical computer.
- The complexity of common classical factoring algorithms is $O(2^n)$, where n is the number of bits in the number to be factored.
- The query complexity of Shor's algorithm is O(n).
- The complexity of Shor's algorithm (not just query complexity) is $O(n^3)$ per iteration. This is estimated as shown below.
- Computing GCD takes O(n).
- The Uf oracle takes $O(n^3)$. Better exponentiation algorithms reduce this to $O(n^2 \text{lgnlglgn})$.
- The QFT gate takes $O(n^2)$. It can be built out of $O(n^2)$ 1/2-qubit gates.