CSE439 Fall 2025 Week 9: Beginning Shor's Algorithm

First, let us review the quantum innards of Simon's Algorithm. The first Hadamard transform on the "x" half of the $2n$-dimensional Hilbert space, and then the reversible form F_f, produce the "functional superposition" of f, which---using our vector indexing notation---is defined by

$$\mathbf{u}(xy) \;= \frac{1}{\sqrt{N}} \text{ if } f(x) = y, \text{ else } 0.$$

The second Hadamard transform of the "x" half then gives $\mathbf{v}(xy) = \frac{1}{\sqrt{N}} \sum_{t \in \{0,1\}^n} (-1)^{x \bullet t} \mathbf{u}(ty)$, where we note that the "Boolean dot product" $\bullet$ actually is the inner product in our vector space over the binary field, i.e., with addition modulo 2. Either $x \bullet t = 0$, in which case the nth-order Hadamard matrix gives $+1$, or $x \bullet t = 1$, giving -1. Putting these together, we have:

$$\mathbf{v}(xy) \;= \frac{1}{N} \sum_{t \in \{0,1\}^n} \begin{cases} (-1)^{x \bullet t} & \text{if } f(t) = y \\ 0 & \text{otherwise.} \end{cases}$$

If f is 1-to-1, then for every y there is exactly one t such that $f(t) = y$, and that is the only nonzero term in the sum. This term has the same magnitude for all x. Note that if we get xy as the result of the measurement, it doesn't mean $f(x) = y$, only that we got amplitude from the t such that $f(t) = y$. Every xy combo thus has magnitude $\frac{1}{N} = \frac{1}{2^n}$. There are $N^2 = 2^{2n}$ such combos, so the total probability is N^2 times $\left(\frac{1}{N}\right)^2$, which is 1. That checks out, and we get a uniformly random x.

In the other case, the hidden string s is different from 0^n and $f(x_1) = f(x_2) \iff x_1 \oplus x_2 = s$ for all $x_1, x_2 \in \{0,1\}^n$, which makes f is 2-to-1 in this very particular manner. Now for any y that we get from the measured output xy, the definition of $\mathbf{u}$ guarantees that y is in the range of f. Hence there are exactly two strings t_1, t_2 such that $f(t_1) = f(t_2) = y$, and those obey $t_1 \oplus t_2 = s$. Thus the only two nonzero terms of the sum, for the given y and (any) x, yield

$$\mathbf{v}(xy) = \frac{1}{N}\left((-1)^{x \bullet t_1} \;+\; (-1)^{x \bullet t_2}\right).$$

Now by the fact that $t_2 = s \oplus t_1$ and $\oplus$ being the same as vector addition in our mod-2 space, we get $(-1)^{x \bullet t_2} = (-1)^{x \bullet (s \oplus t_1)} = (-1)^{x \bullet s \,+\, x \bullet t_1} \;=\; (-1)^{x \bullet t_1}(-1)^{x \bullet s}$, and so

$$\mathbf{v}(xy) = \frac{1}{N}\left((-1)^{x \bullet t_1}\left(1 \;+\; (-1)^{x \bullet s}\right)\right).$$

The nub is that whenever $x \bullet s = 1$, the sum in parentheses *cancels*, giving 0 as the amplitude of $\mathbf{v}(xy)$. Thus the only possible strings x we can get from the measurement are those for which $x \bullet s = 0$. Those are the same as those in the orthogonal subspace $\langle s \rangle^{\perp}$ of s. Because all the nonzero
amplitudes are equal, we get a uniformly distributed member x of that subspace---and this makes the probability analysis of the outer classical part correct. This completes the proof...$\boxtimes$

...provided we trust that the probabilities really do sum to 1. Well, by "quantum conservation" and the laws of Nature, they have to. But it's good to do this final "Simon's sanity check": We have magnitude $\frac{2}{N}$ for each non-canceled amplitude (ignoring the sign). By f being 2-to-1 the range is half the target space, so there are 2^{n-1} possible values y. And any $x \in \langle s \rangle^{\perp}$ is possible (again note: this does not extail $f(x) = y$ from the final measurement), so there are 2^{n-1} of those. So there are $\left(\frac{N}{2}\right)^2 = \frac{N^2}{4}$ combos, each of probability $\frac{4}{N^2}$, so yes the probabilities sum to 1.

From Simon to Shor

The premise that f has a "hidden string" s is what makes the quantum part tick. Note that the proof analysis does not require us to *know* the string s, only to know tnat there is one. The outer classical part eventually enables to compute what s is---with high probability of success, that is. It is unknown whether there is a polynomial-expected-time quantum algorithm, able to query F_f repeatedly as Simon's does, that can distinguish a *general* 2-to-1 function f apart from 1-to-1 cases. This is widely disbelieved, but there is a technically-related problem that is "on the bubble": whether two given undirected graphs G_1 and G_2 are **isomorphic** (meaning: they are structurally the same graph but maybe with differently-numbered nodes). I believe this so-called **Graph Isomorphism** (**GI**) Problem belongs to BQP simply by my belief that **GI** is in classical P.

On the classical side, even if we are told that s exists, it is still a needle in a 2^n-sized haystack. With the Deutsch-Jozsa problem, we proved that a deterministic classical algorithm---when limited to querying for values $f(x)$ on individual strings x only---requires $\Omega\left(2^n\right)$ such queries (and hence requires $\Omega\left(2^n\right)$ time) in worst case to distinguish the "constant" and "balanced" cases. But if we choose queries at random and keep getting the same answer, we can be pretty sure after awhile that "constant" applies---while getting two different answers instantly rules out "constant". Indeed, the expected time for our randomized algorithm is basically $O(n)$. The anti-classical point of Simon's problem is that he proved that not even a randomized algorithm---again making individual-x queries only---can do better than $\Omega\left(2^n\right)$ expected time to unearth the value of s.

This leaves open whether a classical algorithm that is allowed to make "superposed queries" via linear combinations can succeed here. That is to say, the 2^n binary strings x become basis vectors $\mathbf{e}_x$ in N-dimensional space as in the quantum case. Any vector $\mathbf{w}$ can be written as

$$\mathbf{w} = \sum_x a_x \mathbf{e}_x,$$

where we don't necessarily have to restrict the linear-combination coefficients a_x to be 0 or 1; they could be arbitrary real or even complex numbers. The difference is that when we query "$f(\mathbf{w})$", we get not a legal quantum superposition state but the classical linear combination

$$\mathbf{w}' = \sum_x a_x f(\mathbf{e}_x),$$

which is a long vector too. When f is 2-to-1, we get some duplication in this vector, but it can get "mishmashed away" by other values being added on top. Another factor is that we need to use queries $\mathbf{w}$ that can be specified without explicitly writing out 2^n terms---whereas, we justified that the simple functional superposition $\sum_x |x\rangle|f(x)\rangle$ can be prepared by a linear-sized quantum circuit. My gut-feeling is that for those queries $\mathbf{w}$ that can be similarly succinctly specified in $O(n)$ or $n^{O(1)}$ time, the gain for a classical algorithm using linear algebra is minimal. But this also is not proven---the issues are related to the "Algebrization" barrier to resolving the P versus NP question.

Peter Shor, in 1993, was perhaps the first to realize that the feature broadly represented by the hidden string s could be tied to a problem that has resisted solving for over 2,000 years:

Factoring: Given a natural number M, find a prime number p that divides M.

This is more than the problem of telling whether M itself is prime. *That* was placed into deterministic time roughly $O(n^6)$ almost 25 years ago, where $n \sim \log_2 M$ is the number of bits when M is written in binary. [Improving the "6" to match lower exponents long-known for classical randomized algorithms has proved to be a thorny problem unto itself.] The Sieve of Eratosthenes works in time $O(\sqrt{M}) = O(2^{n/2})$ time. There are (classically randomized) algorithms that provably work in roughly $O(2^{n^{1/3}})$ time, and some that possibly work in $O(2^{n^{1/4}})$ or maybe even $O(2^{n^{1/5}})$ time, but those still count as exponential time. There are substantial theoretical and experiential reasons for believing that time $2^{n^{o(1)}}$, let alone polynomial time $2^{O(\log n)} = n^{O(1)}$, is impossible for classical (randomized) algorithms---and that this applies even when $M = pq$ is a product of just two primes. [Mind-you (1): the reasons are weaker than those for believing P $\neq$ NP, because the decision version of the problem---

FACT: Given a natural number M and a number k, is there a prime number $p < k$ that divides M?

---belongs to NP but is not NP-complete unless something mighty close to P = NP actually happens. The two versions are polynomial-time equivalent by classical binary search over k. Mind-you (2): both Lipton and I believe that **FACT** is actually in P, but never-mind why we think that.] It was hence a big shock when Shor proved:

Theorem: **Factoring** belongs to BQP, indeed has quantum circuits of size $\widetilde{O}(n^2)$ that give high probability of finding a prime factor.

Setup of Shor's Algorithm

In general, a **period** of a function f is a value r such that for all x,

$$f(x + r) = f(x).$$

The string s of the "promise property" in Simon's algorithm actually obeys this definition, even though it is a vector not a scalar. When Peter Shor read Simon's paper, he conceptualized that the final Hadamard transform *amplified* the periodic structure in the form of peaks and troughs of waves. The "trough" is how having $a \bullet s = 1$ made the two terms in the amplitude cancel, whereas having $a \bullet s = 0$ made them add with the same sign and hence concentrate the resulting probabilities on those cases.

Now, ahem, converting *periodic* structure into *peaks* is really the job of the *Fourier transform*, not the Hadamard transform. And the Fourier transform does this with numeric data, not just binary-string data. Shor conceptualized that replacing the final Hadamard transform with the **quantum Fourier transform** (**QFT**) might allow a similar concentration that makes a numeric period r emerge. And there is one such function and period of pre-eminent interest in cryptography... Incidentally, the QFT on n qubits is just the same as the ordinary Discrete Fourier Transform (DFT) on vectors of length $N = 2^n$. The circumstance that the QFT can be applied with $\widetilde{O}(n^2)$ quantum effort---so the theory of quantum circuits tells us---is what makes the difference.

Periodic Functions

The important example of a periodic function is **modular exponentiation**:

$$f_a(x) = a^x \bmod M.$$

Here a is a number in $\{0, 1, \ldots, M-1\}$ that is **relatively prime** to M. This means that a does not share a prime divisor with M. When $M = pq$ is the product of two different primes p and q, this simply means that a is not divisible by p or by q. If a and M did share a divisor p, then a^x would always be a multiple of p, and $a^x \bmod M$ is also a multiple of p because p divides M too. So you would not get all of the possible values modulo M. When a is relatively prime to M, what you always get is a number relatively prime to M. This is worth spelling out more than the text does:

Definition: $G_M = \{1\} \cup \{a : 1 < a < M \text{ and } a \text{ is relatively prime to } M\}$.

Theorem: G_M forms a **group** under multiplication.

A **group** is a set G with a distinguished element 1 together with an operation $\circledcirc$ that satisfies the following axioms:

- For all $g \in G$, $g \circledcirc 1 = 1 \circledcirc g = g$.
- For all $g \in G$ there is a unique $h \in G$ such that $gh = 1$ and $hg = 1$. We write $h = g^{-1}$.

For example, the $n \times n$ unitary matrices U form a group with $U^{-1} = U^*$. Well, the numbers in modular arithmetic form groups that are simpler to understand.

When $M = pq$ is a product of two primes, the size of G_M is exactly $(p-1)(q-1)$. (The general name for the size of G_M is the **totient** function of M, devised by and often named for the mathematician Leonhard Euler.) The consequence of G_M being a group that we need is:

Corollary: For all $a \in G_M$ there is a positive integer r such that $a^r \equiv 1 \bmod M$.

The least such r is exactly the period of $f_a(x)$ that we want to find. It always divides $|G_M|$, so when $M = pq$ we get that r divides $(p-1)(q-1)$. You might think this should narrow down possibilities, but:

- We don't actually get the value $m = (p-1)(q-1)$ factored for us---we don't even know m because we don't know how to factor $M =: pq$ to begin with.
- Compared to the number n of bits or digits of M, which is the complexity parameter we care about, the range of numbers less than m we might have to check is exponential in n.
- By the way, the number x in a^x can be exponential in n, so it looks like it takes too long to compute $f_a(x)$ to begin with. However, by **iterated squaring modulo** M we can compute the following values in $\widetilde{O}(n^2)$ time: $a_1 = a^2 \bmod M$, $a_2 = a_2^2 \bmod M = a^4 \bmod M$, $a_3 = a_2^2 \bmod M = a^8 \bmod M$, $a_4 = a_3^2 \bmod M = a^{16} \bmod M$, and so on up to $a_{n-1} = a_{n-2}^2 \bmod M = a^{n-1} \bmod M$. Then we need only multiply together those a_i such that x as a binary number includes 2^i. This needs only $2n$ multiplications and mod-M reductions of n-bit numbers, so it is doable in $\widetilde{O}(n^2)$ time using an $\widetilde{O}(n)$-time integer multiplication algorithm. (Or we can say $O(n^3)$ time using the simple multiplication algorithm. The **RSA cryptosystem** uses modular exponentiation too---and this time is largely why your credit card needed a chip.)

Nevertheless, if we *do* find the period r---for a "good" value a which we stand a fine chance of picking at random from G_M---then it was known long before Peter Shor found his algorithm in 1993 that we can go on to find p and q by classical efficient means.

Theorem: There is a **classical** randomized algorithm that, when provided a ***function oracle*** $g(M, a)$ = some integer multiple of the period of $f_a \bmod M$, finds a factor of M in expected polynomial time. That is, **Factoring** is in BPP^g.

That proof is the entire content of Chapter 12. Lipton and I bundled this up into a separate chapter so that instructors would have the freedom to skip it, as we'll do for the time being. (2024: It was in a replacement lecture done online via Zoom.) So we can focus on the task of finding r (or at least a multiple of r) via *quantum means*.

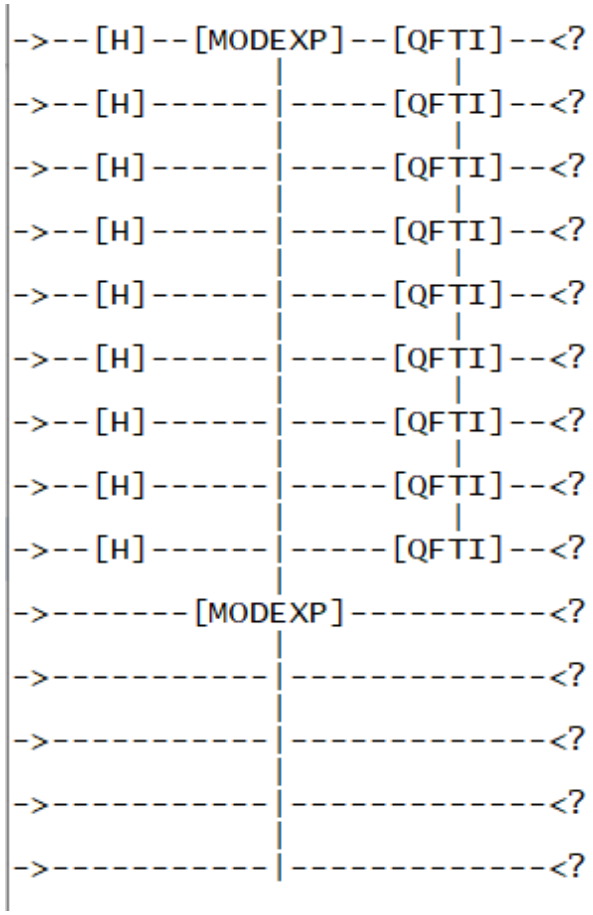
Shor's Theorem: We can build $\widetilde{O}(n^2)$-sized quantum circuits that given $M = pq$ and randomly-chosen $a \in G_M$ allow sampling values w via quantum measurement that with $O(n)$ samples give high probability of calculating (some integer multiple of) r.

One Other Useful Fact: The values $f(x), f(x+1), \ldots, f(x+r-1)$, when r is the **least** period, are all distinct when $f = f_a$ is eponentiation modulo M. [*Why* Shor's algorithm needs this property, even when measurement doesn't give you the least r, is still mysterious.]

Steps of Shor's Algorithm

1. Given M, use classical randomness to guess a number a between 2 and $M-1$.
2. Use Euclid's algorithm to find $\gcd(a, M)$. If it gives a number $c > 1$, then "ka-ching!"---we got a divisor of M. Since both c and M/c are below $M/2$, we can recursively factor both of them.
3. If it gives $\gcd(a, M) = 1$, then we know $a \in G_M$. In the important $M = pq$ case, this had probability $\frac{(p-1)(q-1)}{pq}$ and so was pretty likely anyway. By the way, Euclid's algorithm also gives you a number b such that $ab = 1 \bmod M$. But it doesn't give you this b as a power of a (to wit, as $b = a^{r-1} \bmod M$), which is what you'd need to get r.
4. To give some slack, we choose a number $Q = 2^\ell \approx M^2$ and expand the domain of $f_a(x)$ to include x in the interval up to $Q-1$, not just up to $M-1$. The range is still 1 to $M-1$. So our domain is x in the range 0 to $2^\ell - 1$, which uses $\ell \approx 2n$ bits. This gives us quadratically many "ripples" of the period, which in turn helps the trigonometric analysis in the body of the proof.
5. The quantum circuit begins with ℓ-many Hadamard gates, followed by a quantum implementation of the $n^{O(1)}$ classical gates needed to compute modular exponentiation. This produces the functionally superposed quantum state
$$\Phi_f = \frac{1}{\sqrt{Q}} \sum_{x \in \{0,1\}^\ell} |x f_a(x)\rangle.$$
6. Apply the QFT (or its inverse) to the first ℓ qubits.
7. Then *measure* the whole result. Curiously, we ignore what happens in the "$f_a(x)$" portion of the circuit. The fact that those final n qubits were entangled with the first ℓ qubits is enough. So we let our output w in the "x-space" be the first ℓ bits of the measured result over the binary standard basis.

My own quantum circuit simulator draws an ASCII picture of the Shor circuit, here for $M = 21 = 3*7$ (where I guessed $a = 5$), which gave $\ell = 9$ since $2^9 = 512$ is the next power of 2 after $M^2 = 441$:



But there isn't any more to the quantum circuitry than that. It's all simply: compute a giant functional superposition and apply QFT (or its inverse) to it.

The analysis establishes that with pretty good probability already in one shot, the x part of the measured output xy reveals the period r by a followup classical means. (Note: we do not generally gate $y = f_a(x)$ here.) And with initial good probability over the choice of a, the resulting value r unlocks the key to factoring M. We will focus on understanding why the measured x has much to do with the period r to begin with. The basic point---which has been known for centuries---is that the Fourier transform converts *periodic data* to *peaked data*. Here is how the simple quantum circuit above applies this fact.

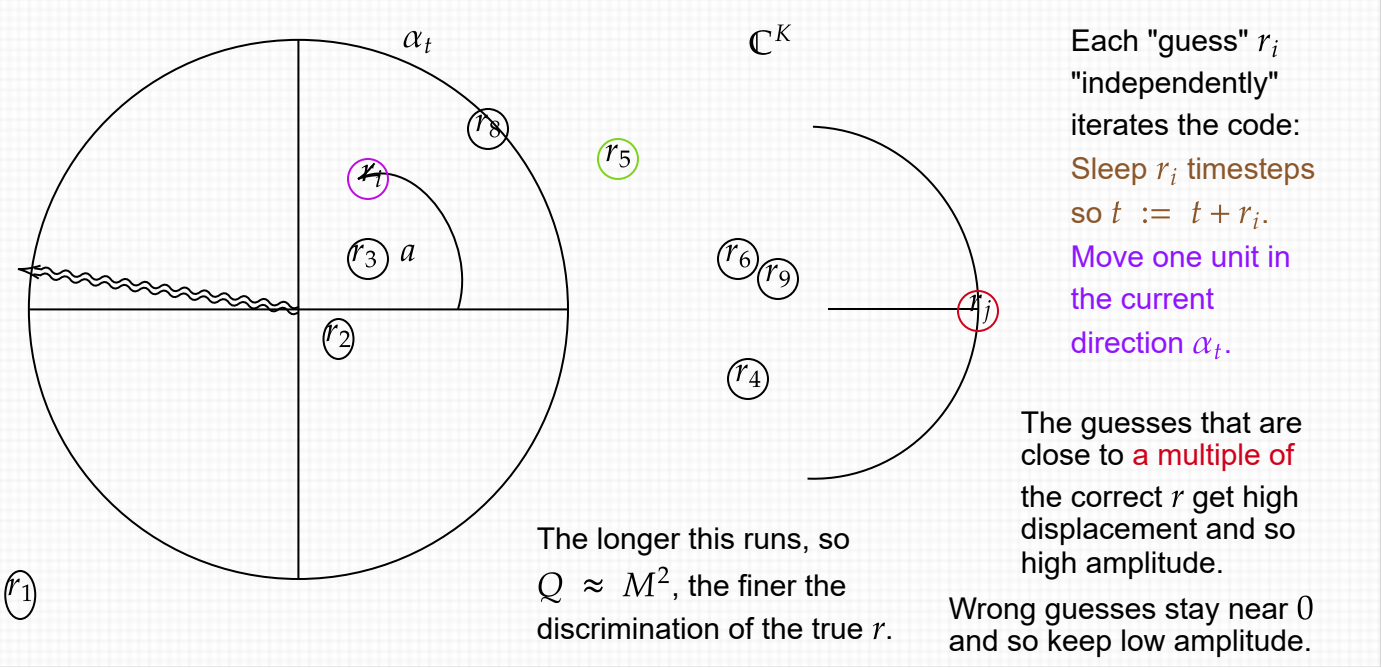
[Thursday's lecture will pick up here---please read the Aaronson blog post if you can. I will do a "clumsy animation" using MathCha. My own quantum simulator is in the folder `/shared/projects/regan/QCSAT`, executable code `qci`, on the CSE machines `turing` or `cerf`. I will introduce it on Thursday too.]

The Intuition

The following intuitive explanation famously comes from Scott Aaronson, https://www.scottaaronson.com/blog/?p=208 .

Recall: $\mathbf{QFT}_\ell[x,u] = \omega^{xu}$, where x and u are treated as numbers not strings and ω is a principal 2^ℓ root of unity, wlog. $\omega = e^{2\pi i/Q}$. The angles of multiples and powers of ω are what we consider as values in $[0, Q-1]$.

Let r stand for the true period of f. Then r is at most the size of the group G_M, minus one,so in particular, $r < M$. Let a be any element of the group G_M of size $(p-1)(q-1)$. Then we will picture a as a "crazy clock" that jumps a units *counter*-clockwise at each time step.



With fairly high probability, measurement---followed by figuring needed to get the guessed r_i from the measurement---yields a multiple of r. The true r is the least of the multiples. It is individually the most likely value returned and is also returned with reasonable probability. A non-least r might work anwyay. We can tell whether r works by seeing if the classical part gives us p or q, else we just try the quantum process again.

Heading into the analysis, however, we need to say exactly what the measured string w actually represents. In general, the angle α represented by a (when we actually use the complex plane to model the "crazy clock") will not be a whole-number fraction of the circle. But let us first suppose it is. Then the smallest period r (i.e., the true period) will go exactly once around the circle and back to angle α as represented by a. So suppose r_i is a correct guess of r. Then with high probability, the output w

of the measurement has the same angle α. Since angles add when we multiply complex numbers, this means $r\alpha$ takes us once around the circle. This in turn means that α is the *reciprocal* of r with regard to the circle. So w would be close to this reciprocal.

In the general case, we have to go some number t times around the circle before we get exactly back to a. That is, we have $r\alpha = t$ with respect to the circle. So $\alpha = \frac{t}{r}$ times whatever number Q represents the extent of once-around-the-circle in the units we are using. This finally means that w should be close to $\frac{tQ}{r}$ in these units. The w needs to be close enough to pull one final switcharoo: We don't know what t is either, but from $w \approx \frac{tQ}{r}$ we get $r \approx t\frac{Q}{w}$. Since r has to be an integer, we just need to find a t that multiply the fraction $\frac{Q}{w}$ into being real close to an integer. It turns out this will work when the additive error in the measured w relative to the "true amplifying direction" $\frac{tQ}{r}$ is at most ± 0.5 in the circle's units. Choosing Q high enough makes those units fine enough for this to work. The "analysis of the quantum part" tells how often the measured w is close enough to be "good." (As was the case with Simon's algorithm, the text re-uses the letter "x" to denote the particular string from the "x-space" that was obtained in the measurement.)

Details of Shor's Algorithm

The top-down goal is to find a number X such that $X^2 \equiv 1$ modulo M but X is not $\equiv 1$ or $\equiv -1$ modulo M. Then $X^2 - 1 = (X-1)(X+1)$ is a multiple of M but neither factor is zero. When $M = pq$ with p, q prime, this means p and q each divide one or both factors. We need to split them across the factors, so that $\gcd(X-1, M)$ and/or $\gcd(X+1, M)$ will find p and q as opposed to just giving M back again. Thus we want to guess a such that:

1. The period r of a is even, so that $r/2$ is defined;
2. $X = a^{r/2} \not\equiv M - 1$ modulo M.
3. Either $X - 1$ or $X + 1$ is a multiple of one of p, q **but not both**.

If our value of a fails either of these, we just try again from the start of guessing $a < M$.

Our treatment (blog post and chapter 12) also desires r to be a multiple of $p-1$ or $q-1$. It can be shown that many a give this "helpful" property, which requires $r \geq \sqrt{(p-1)(q-1)} \approx \sqrt{M}$.

(It is not clear whether we show this. It could be an exercise: Consider numbers r that divide a product mn of two nearly-equal composite numbers. Conditioned on $r \geq \min\{m, n\}$, give a lower bound for the proportion that are a multiple of m or a multiple of n. Note that m and n need not be themselves relatively prime; $p-1$ and $q-1$ are both even, for instance. It would still need to be argued that most a

give such an r. But I am not sure that the "helpful" property is needed either.)

Chapter 12 does handle the argument in property 3, given that r is "helpful"---which also subsumes issue 1 since $p-1$ and $q-1$ are even. Issue 2 is handled by a random argument.

We will see that the closer r is to $\sqrt{M}$ as opposed to being order-of M, the more challenging for a potential classical simulation of Shor's algorithm.

Another thing to observe is that when M is a Blum integer, meaning p and q are both congruent to 3 modulo 4, then $(p-1)(q-1)$ is divisible by 4 but no higher even number. There are always four square roots of 1 modulo $M = pq$, so we need to argue that the a's such that $a^{r/2}$ is one of the good ones are as plentiful as the bad ones. (Note that r depends only on a.) Here is an example for the smallest Blum integer: $21 = 3*7$. The quadratic residues are:

$1:1, 2:4, 3:9, 4:16, 5:4, 6:15, 7:7, 8:1, 9:18, 10:16,$
$20:1, 19:4, 18:9, 17:16, 15:15, 14:7, 13:1, 12:18, 11:16$

Now $(p-1)(q-1) = 12$. The numbers $Y = 8-1, 8+1, 13+1$, and $13-1$ all give a factor via $\gcd(21, Y)$.

$a = 1$: $r = 1$; of course doesn't work.
$a = 2$: $2, 4, 8, 16, 11, 1$. Works
$a = 4$: $16, 1$ (period 3 is odd)
$a = 5$: $4, 20, 16, 17, 1$; doesn't work because $20 \equiv -1$.
$a = 8$: $8^2 \equiv 1$. Period $r = 2$ is "helpful" and $8^{r/2} = 8$ is not -1. So works.
$a = 10$: $16, 13, 4, 19, 1$. Works
The other values are mirror images.

A more interesting Blum integer IMHO is $77 = 7*11$. Then $(p-1)(q-1) = 60$. "Helpful" means the period is a multiple of 6 or of 10. Note: $34^2 = 1156 = 77*15 + 1$ is a nontrivial square root of 1 and $43^2 = 1849 = 77*24 + 1$ is the other one. Does 2 work?

$2:4, 8, 16, 32, 64, 51, 25, 50, 23, 46, 15, 30, 60, 43, 9, 18, 36, 72, 67, 57, 37, 74,$ etc.: yes.

The next question is whether it is OK for the quantum part to obtain a multiple $r' = br$ of a helpful r. If b is even than certainly not, because $a^{r'/2}$ will be 1. But if b is odd---? In any event, we can obviate this question because we can single out the minimum r with sufficiently high probability.

The key auxiliary technical notion is a number x that is "good" to help find r.

11.2 Good Numbers

Let Q be a power of two, $Q = 2^\ell$, such that $M^2 \le Q < 2M^2$. Say an integer x in the range $0, 1, \ldots, Q-1$ is **good** provided there is an integer t relatively prime to the period r such that

$$tQ - xr = k, \quad \text{where} \quad -r/2 \le k \le r/2. \tag{11.1}$$

The key part is the multiple t of Q being relatively prime to r.

LEMMA 11.1 There are $\Omega\left(\frac{r}{\log\log r}\right)$ good numbers.

Proof. The key insight is to think of equation (11.1) as an equation modulo r. Then it becomes

$$tQ \equiv k \bmod r,$$

where $-r/2 \le k \le r/2$. But as t varies from 0 to $r-1$, the value of k can be arranged to be always in this range, so the only constraint on t is that it must be relatively prime to r. The number of values t that are relatively prime to r defines Euler's *totient* function, which is denoted by $\phi(r)$. Note that for each value of t there is a different value of x, so counting ts is the same as counting xs. Thus, the lemma reduces to a lower bound on Euler's function. But it is known that

$$\phi(z) = \Omega\left(\frac{z}{\log\log z}\right).$$

Indeed, the constant in Ω approaches $e^{-\gamma}$, where $\gamma = 0.5772156649\ldots$ is the famous Euler-Mascheroni constant. In any event, this proves the lemma. □

The general drift is that a good x gives a good chance of finding r exactly, by purely classical means. Of note:

If r is close to M, then by choosing Q close to M rather than M^2, we would stand a good chance of finding a good x just by picking about $\log \ell$-many of them classically at random. However, this does not help when r is smaller. The genius of Shor's algorithm is that the quantum Fourier transform can be used to drive amplitude toward good numbers in all cases.

This makes $r \approx M^{1-\epsilon}$ where $0 < \epsilon < 1$ the "vat" of hard cases: too sparse to guess at random. For the quantum part, however, we need $Q > rM$. Just to finish off the classical part:

LEMMA 11.7 If x is good, then in classical polynomial time, we can determine the value of r.

Proof. Recall that x being good means that there is a t relatively prime to r so that (by symmetry)

$$xr - tQ = k \qquad \text{where} \qquad -\frac{r}{2} \leq k \leq \frac{r}{2}.$$

Assume that $k \geq 0$; the argument is the same in the case where it is negative. We can divide by rQ and get the equation

$$\left| \frac{x}{Q} - \frac{t}{r} \right| \leq \frac{1}{2Q}.$$

We next claim that r and t are unique. Suppose there is another t'/r'. Then

$$\left| \frac{t}{r} - \frac{t'}{r'} \right| \geq \frac{1}{rr'} \geq \frac{1}{M^2}.$$

But then both fractions are close, which makes Q smaller than M^2, a contradiction.

Because r is unique, it follows that t is too. So we can treat

$$xr - tQ = k$$

as an integer program in a fixed number of variables: the variables are r, t, and two slack variables used to state

$$-r/2 \leq k \leq r/2$$

as two equations. While integer programs are hard in general, for a fixed number of variables they are solvable in polynomial time. This proves the lemma. □

Simulation Interlude

Before we go to this analysis, let's see a brute-force simulation of Shor's algorithm. It pretty much builds the concrete "mazes" for $\ell + n$ qubits and simulates all the legal "Feynman mouse paths" through them. The run of my simulator on $M = 21$ and $a = 5$ succeeded on the second try:

```
About to do try 1 of sampling QFT applied to 1010101011010010100 with status now PROBS_ENUMERA
Sampling with status PROBS_ENUMERATED:
Base probability for conditionals: 0.166015625000
Current: 0 with probability 0.083007813 on rolling 0.325191374; last 0 prob = 0.500000000
Current: 00 with probability 0.055282593 on rolling 0.563273639; last 0 prob = 0.665992647
Current: 001 with probability 0.027659269 on rolling 0.559076137; last 0 prob = 0.499674899
Current: 0010 with probability 0.027418884 on rolling 0.941772811; last 0 prob = 0.991309060
Current: 00101 with probability 0.027183985 on rolling 0.139894580; last 0 prob = 0.008567052
Current: 001010 with probability 0.026380861 on rolling 0.938149097; last 0 prob = 0.970455980
Current: 0010101 with probability 0.025648040 on rolling 0.595421001; last 0 prob = 0.02777850
Current: 00101010 with probability 0.020074378 on rolling 0.114898273; last 0 prob = 0.7826866
Current: 001010101 with probability 0.018908726 on rolling 0.791199151; last 0 prob = 0.058066
sampled output vector: 00101010110100
time cost: 1.23308 milliseconds.

Measured 001010101 as 85 giving 0.166015625
Fractional approximation is 1/6
; Possible period is 6
; Unable to determine factors, we'll try again.
Let's take a free random crack at it without the QFT application...
Fractional approximation is 2/3
; Odd denominator, trying to expand by 2.
; Possible period is 6
; Unable to determine factors, we'll try again.

About to do try 2 of sampling QFT applied to 1010101011010010100 with status now PROBS_ENUMERA
Sampling with status PROBS_ENUMERATED:
Base probability for conditionals: 0.166015625000
Current: 1 with probability 0.083007813 on rolling 0.527169932; last 0 prob = 0.500000000
Current: 10 with probability 0.055282593 on rolling 0.051374227; last 0 prob = 0.665992647
Current: 100 with probability 0.027623324 on rolling 0.277237177; last 0 prob = 0.499674899
Current: 1000 with probability 0.027576410 on rolling 0.189192738; last 0 prob = 0.998301645
Current: 10000 with probability 0.027567765 on rolling 0.562397971; last 0 prob = 0.999686499
Current: 100000 with probability 0.027564179 on rolling 0.523783427; last 0 prob = 0.999869929
Current: 1000000 with probability 0.027562462 on rolling 0.694951445; last 0 prob = 0.99993770
Current: 10000000 with probability 0.027561612 on rolling 0.646817553; last 0 prob = 0.9999691
Current: 100000000 with probability 0.027561188 on rolling 0.353241189; last 0 prob = 0.999984
sampled output vector: 10000000010100
time cost: 1.2329 milliseconds.

Measured 100000000 as 256 giving 0.500000000
Fractional approximation is 1/2
; Possible period is 2
; Success: 21 = 3 * 7
Success after 2 xy sample(s) plus 2 QFT sample(s).
```