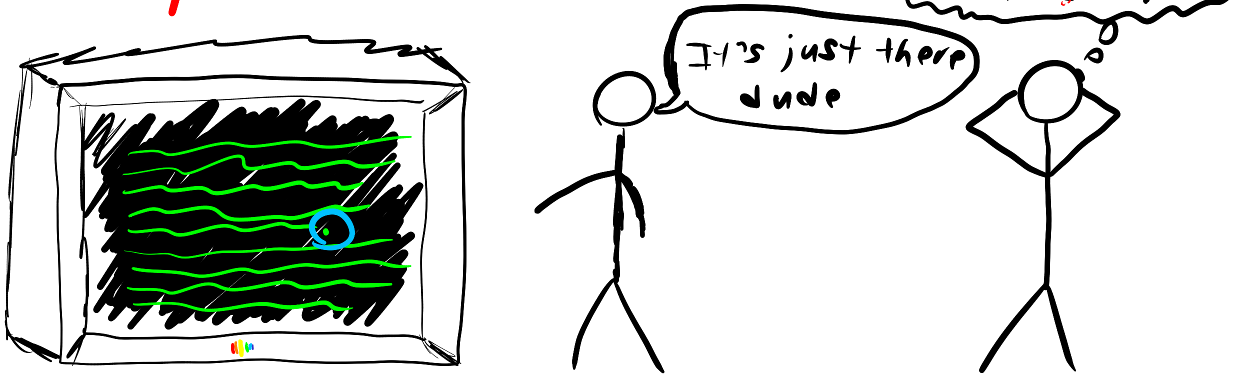
CMPT 476 Lecture 21

Quantum period finding



As we've been discussing for the last few days, the quantum part of Shor's factoring algorithm is really a period-finding algorithm using an efficient (Quantum) Fourier transform

$$QFT_{2^n} : |x\rangle \longmapsto \frac{1}{\sqrt{2^n}} \sum_{y \in \mathbb{Z}_{2^n}} \omega_{2^n}^{xy} |y\rangle$$

Today we'll see how this period finding algorithm works. First, let's define the problem of period finding in general.

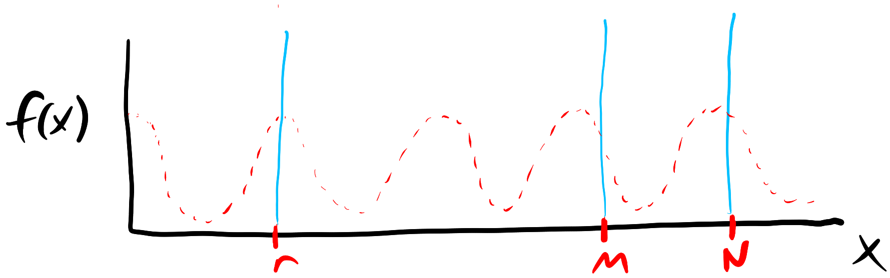
(The period finding problem)

Input: A function $f: \mathbb{Z}_N \to \mathbb{Z}_N$

Promise: f is periodic with $f(x) = f(x+r) = f(x+2r) = \cdots$

Goal: Find the period r.

Technically we'll only solve the problem for $N = 2^n$, due to the restriction of the QFT to Fourier transforms over $\mathbb{Z}_{2^n}$, but generally it doesn't matter as we can usually extend $f: \mathbb{Z}_M \to \mathbb{Z}_M$ to $N \geq M$ by translating the fundamental period $[0, \ldots, r-1]$



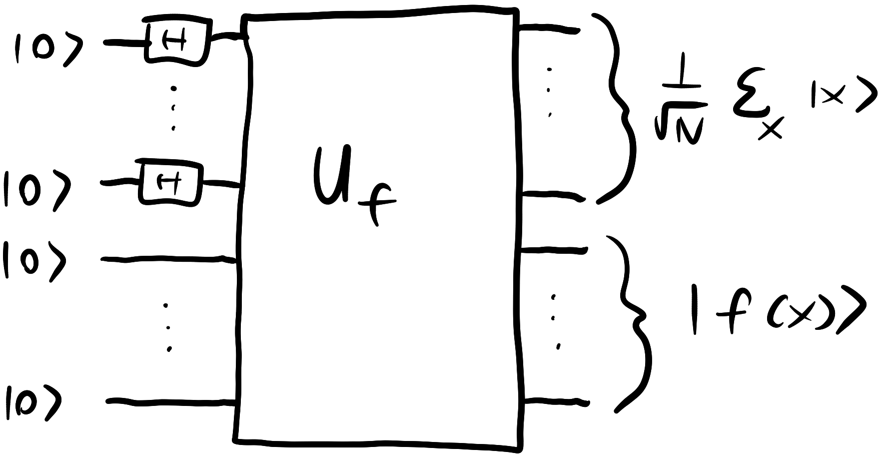
In the case of modular exponentiation $a^x \bmod M$, if $x \geq M$, then $a^x \bmod M = a^y \bmod M$ where $x = qM + y$. The mathematics is the same (mostly) if $N \neq 2^n$ so we'll just speak in general terms of N.

(Shor's period finding algorithm)

Shor's algorithm for finding periods in $\mathbb{Z}_N$ starts off identically to Simon's for $\mathbb{Z}_2^n$

1. Create the uniform superposition $\frac{1}{\sqrt{N}} \sum_{x \in \mathbb{Z}_N} |x\rangle$
2. Apply $\underbrace{U_f}_{\text{Oracle for } f}$ to get $\frac{1}{\sqrt{N}} \sum_x |x\rangle |f(x)\rangle$

As a circuit, we know what this looks like:



As in Simon's algorithm, we'll pretend to measure $|f(x)\rangle$ to get a particular value y. Now,

$$y = f(x) = f(x+r) = f(x+2r) = \cdots$$

so we can write the resulting state as

$$\frac{1}{\sqrt{L}} \sum_{x \mid f(x)=y} |x\rangle|y\rangle = \frac{1}{\sqrt{L}} \sum_{k=0}^{L-1} |x+kr\rangle\,|y\rangle$$

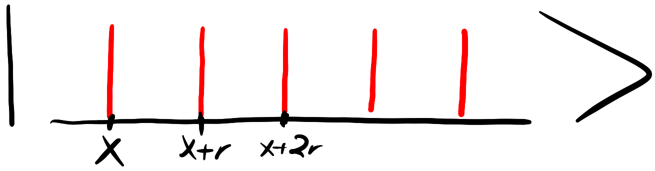
normalization factor

s.t. $f(x)=y$

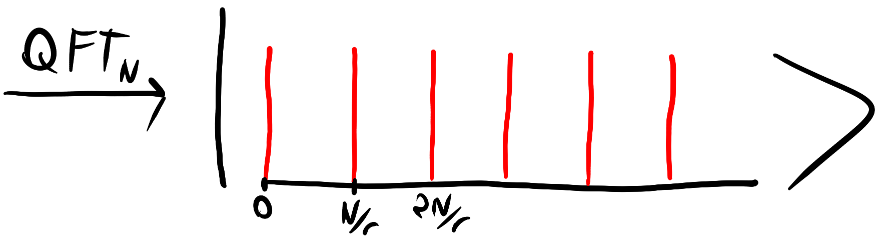
how many periods fit in N

$L \approx N/r$

We can visualize this state as



Now, what happens if we apply the QFT_N to the periodic state $\frac{1}{\sqrt{L}}\left(|x\rangle + |x+r\rangle + \cdots + |x+(L-1)r\rangle\right)$?



(we hope)

More formally,

$$QFT_N\left(\frac{1}{\sqrt{L}}\sum_{k=0}^{L-1}|x+kr\rangle\right) = \frac{1}{\sqrt{L}}\sum_{k=0}^{L-1}\left[\frac{1}{\sqrt{N}}\sum_{z=0}^{N-1}\omega_N^{(x+kr)\cdot z}\,|z\rangle\right]$$

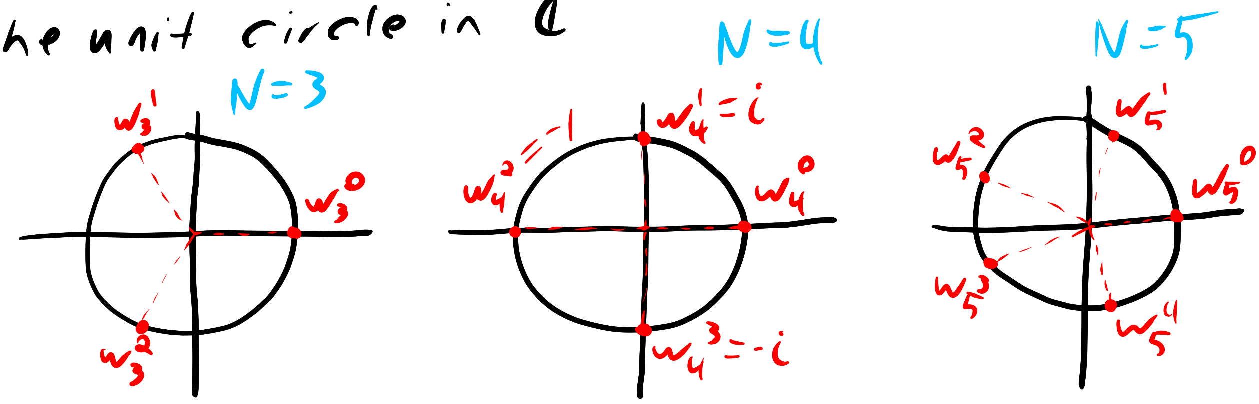
To determine which z's have non-zero amplitude, we'll need some theory of roots of unity.

(Roots of unity)

The N^{th} roots of unity are the complex numbers

$$e^{2\pi i \frac{k}{N}} = w_N^k, \quad k = 0, 1, \ldots, N-1$$

Note that $w_N^N = 1$. If k and N are co-prime, then w_N^k is a primitive root of unity. The N^{th} roots of unity correspond to equally spaced points on the unit circle in $\mathbb{C}$



Note that $w_N^k = w_{N/k}$ whenever k divides N — e.g. $w_4^2 = e^{2\pi i/4 \cdot 2} = e^{2\pi i/2} = w_2$ as above.

An important fact about roots of unity which will drive the interference we need to find periods is given below:

(Sums of N^{th} roots of unity)

Let w_N^k be an N^{th} root of unity and $w_N^k \neq 1$. Then

$$\sum_{i=0}^{N-1} w_N^{k \cdot i} = 0$$

In other words, if we sum up all the N^{th} roots of unity, they all point in opposing directions on the unit circle and end up cancelling out.

On the other hand, if $k = Nm$ for $m \in \mathbb{Z}$, then

$$\sum_{i=0}^{N-1} \omega_N^{k \cdot i} = \sum_{i=0}^{N-1} \omega_N^{Nmi} = \sum_{i=0}^{N-1} 1^{mi} = N$$

So like in Simon's algorithm, when

$$(x+kr)\cdot z \equiv 0 \bmod N$$

we'll get constructive interference, and every other case will give destructive interference.

Let's do the analysis

(Intereference analysis)

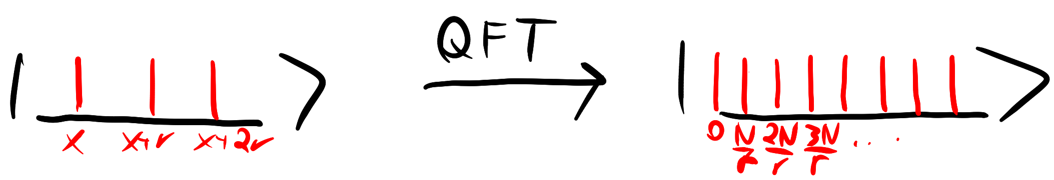
Consider some particular $z \in \mathbb{Z}_{2^n}$. The state $|z\rangle$ in the output of the period finding algorithm has amplitude

$$\frac{1}{\sqrt{L \cdot N}} \cdot \sum_{k=0}^{L-1} \omega_N^{xz + kr} = \frac{\omega^{xz}}{\sqrt{L \cdot N}} \sum_{k=0}^{L-1} \omega_N^{krz}$$

To do the analysis, it's helpful to first see what happens if $N = rM$ for some $M \in \mathbb{Z}$.
In particular, the period is some multiple of N.
Intuitively, since the Fourier Transform decomposes a function into a sum of N/k-periodic functions

$$x \longmapsto \omega_N^{kx}$$

Our function should be representable exactly by sums of r-periodic functions $x \longmapsto \omega_N^{kx}$, $N/k = rM$.
This is the picture we informally saw last class:



Case 1: r divides N (i.e $N = rM$)

Then $L = N/r = M$ and $\omega_N^{krz} = \omega_M^{kz} = \omega_L^{kz}$, so

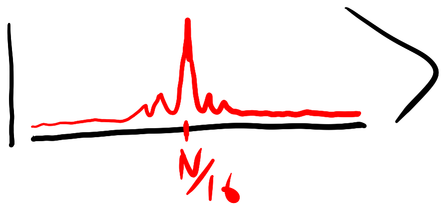
$$\sum_{k=0}^{L-1} \omega_L^{k\cdot z} = \begin{cases} L & \text{if } z = mL \quad \text{for some } m \in \mathbb{Z} \\ 0 & \text{otherwise} \end{cases}$$

So if we take a few samples we would get

$$m_1 L, m_2 L, m_3 L, \dots$$

With high probability, $m_1, m_2, m_3, \dots$ share no common primes, so $GCD(m_1 L, m_2 L, m_3 L, \dots) = L = N/r$ and we're done since $r = N/L$.

If r does not divide N, then things are more interesting. In particular, ω_N^{kx} is not exactly r-periodic for any k, so the Fourier Transform decomposes the function into a sum of terms which are not quite r-periodic. How? Well intuitively if the period is 15, that's almost a period of 16, so the Fourier spectrum will have a spike near $N/16$ (if $N = 2^n$) along with smaller spikes nearby to shift the frequency a tiny bit:



Case 2: r does <u>not</u> divide N

Well, we saw constructive interference when

$$z = mL = m\,N/r$$

for some integer m, so intuitively the same should happen when $z \approx m\,N/r$. If we really want to be sure we should do some back of the envelope calculations...

Suppose $m\,N/r = z - \varepsilon$ for some $m, z \in \mathbb{Z}$. Then

$$\sum_{k=0}^{L-1} w_N^{krz} = \sum_{k=0}^{L-1} w_N^{kr(m N/r + \varepsilon)}$$

$$= \sum_{k=0}^{L-1} w_N^{kmN}\, w_N^{kr\varepsilon}$$

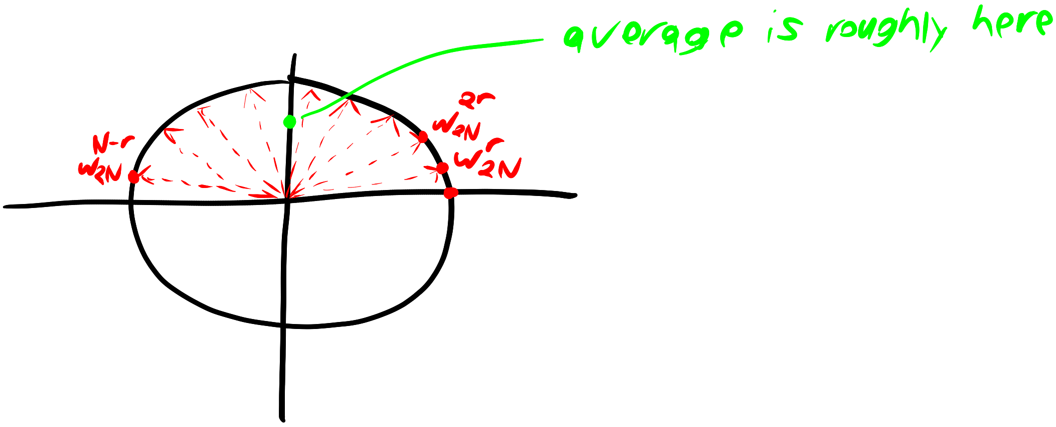
$$= \sum_{k=0}^{L-1} w_N^{kr\varepsilon}$$

Now $L \approx N/r$ as before and $\varepsilon = 1/2$ in the worst case since N/r is at most 0.5 away from an integer z. So

$$\sum_{k=0}^{L-1} w_N^{kr\cdot\frac{1}{2}} = \sum_{k=0}^{L-1} w_{2N}^{kr} = 1 + w_{2N}^{r} + \cdots + w_{2N}^{(L-1)r} \approx w_{2N}^{N-r}$$

$$= 1 + w_{2N}^{r} + \cdots + w_{2N}^{N-r}$$

Since $w_{2N}^{N} = -1$, we're more or less summing up the roots of unity of a half-plane (in the worst case)



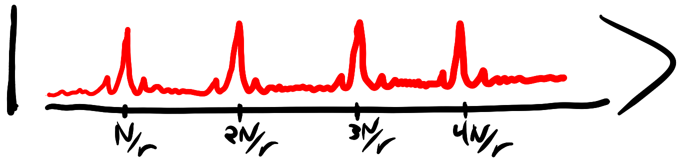
Now, the $\overbrace{\text{center of mass}}^{=\text{average}}$ of a half ring with radius 1 is $\frac{2}{\pi}$, so $\frac{1}{L}\sum_{k=0}^{L-1} w_{2N}^{kr} \approx \frac{2}{\pi} \approx 0.64i$

The probability of measuring this z is hence

$$\left|\frac{1}{\sqrt{LN}}\sum_{k=0}^{L-1} w_{2N}^{kr}\right|^2 \approx \frac{L}{N}\,0.64^2 \approx \frac{1}{r}\,0.41$$

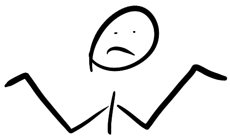
Since we have about r many values of z, we have a good 40% chance of measuring $z \approx m\,N/r$.

The Fourier spectrum in this case looks a bit like this:



Note:

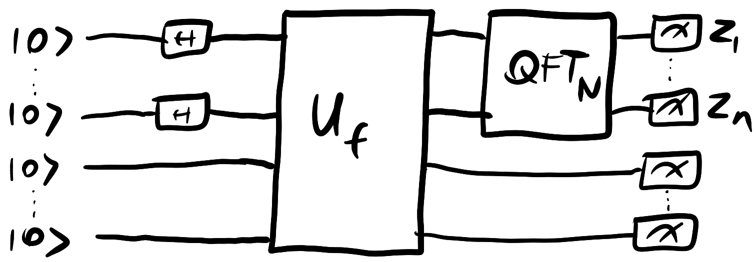
We can pump up the success probability by our choice of $N = 2^n$ where n is the number of qubits. If $r \le M$ then choosing $N \gg M$ can make N/r closer to an integer and boost the chances of success. How much?



(Just kidding. The number theorists say ≈ 1)

(Finding the period)

So where are we now? We know that by running this circuit



the bitstring $z = z_n z_{n-1} \cdots z_1$ is the floor or ceiling of $m\,N/r$ with at least 40% approximate probability.

Assuming we got lucky, we still don't know r.

To find r, we'll have to use continued fractions.

(Continued fractions)

Continued fractions are an old, old technique related to Euclidean division in the GCD algorithm. The idea is to expand a fractional number x/y by repeatedly separating the integer & fractional parts, giving

$$\frac{x}{y} = a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cdots}}}$$

Ex.

Suppose we want to expand $\frac{436}{100} \approx \sqrt{19}$.

First we take $\frac{436}{100} = 4 + \frac{36}{100}$

$$= 4 + \cfrac{1}{\frac{100}{36}}$$

take reciprocal so that we can recursively expand...

Now $\frac{100}{36} = 2 + \frac{28}{36} = 2 + \cfrac{1}{\frac{36}{28}}$

And $\frac{36}{28} = 1 + \frac{8}{28} = 1 + \cfrac{1}{28/8}$

And $\frac{28}{8} = 3 + \frac{4}{8} = 3 + \cfrac{1}{8/4} = 3 + \frac{1}{2}$

So $$\frac{436}{100} = 4 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{3 + \cfrac{1}{2}}}}$$

(Convergents)

If we truncate the continued fraction expansion of $\frac{x}{y}$ after k levels, the resulting fraction $\frac{a}{b}$ approximates $\frac{x}{y}$ and is called a convergent.

Ex.

The convergents of $\frac{436}{100}$ are

$$4,\quad 4+\frac{1}{2}=\frac{9}{2},\quad 4+\frac{1}{2+\frac{1}{1}}=\frac{13}{3},\quad 4+\frac{1}{2+\frac{1}{1+\frac{1}{3}}}=\frac{48}{11}$$

Note that the denominator increases with each convergent, which is intuitively what allows each convergent to better approximate the intended fraction. Moreover, as the next theorem shows, given appropriate error bounds, we can always find a given approximation as a convergent.

Theorem

Let $\left|\frac{a}{2^n}-\frac{b}{c}\right| \leq \frac{1}{2c^2}$ for some $a,b,c,n \in \mathbb{Z}$. Then the continued fractions algorithm for $\frac{a}{2^n}$ converges to $\frac{a}{2^n}$ in $O(n)$ steps, and $\frac{b}{c}$ is a convergent in this sequence.

(Back to period finding)

So how does this help? Well suppose we got some z which is the closest integer to mN/r. Then if $N \geq 2r^2$ which we can get by choosing (for modular exponentiation with modulus M) $N=2^n \geq M^2$, we have

$$\left|\frac{z}{N}-\frac{m}{r}\right| \leq \left|\frac{mN/r+\frac{1}{2}}{N}-\frac{m}{r}\right| = \left|\frac{1}{2N}\right| \leq \left|\frac{1}{2r^2}\right|$$

So if we compute continued fractions of $\frac{z}{N}$, we'll eventually find m & r.

How do we know when to truncate?

Well, we know $\left|\frac{z}{N} - \frac{a_i}{b_i}\right| \le \left|\frac{1}{2N}\right| = \left|\frac{1}{2M^2}\right|$ when $\frac{a_i}{b_i} = \frac{m}{r}$, and $r < m$. Now, suppose $\frac{a_j}{b_j}$ is another <u>distinct</u> convergent with these properties. Then

$$\begin{aligned}
\left|\frac{1}{2M^2}\right| + \left|\frac{1}{2M^2}\right| &\ge \left|\frac{z}{N} - \frac{a_i}{b_i}\right| + \left|\frac{a_j}{b_j} - \frac{z}{N}\right| \\
&\ge \left|\frac{a_j}{b_j} - \frac{a_i}{b_i}\right| \\
&= \left|\frac{a_j b_i - a_i b_j}{b_j b_i}\right| \\
&\ge \left|\frac{1}{b_j b_i}\right| \quad \text{since } \frac{a_i}{b_i} \ne \frac{a_j}{b_j} \\
&> \left|\frac{1}{M^2}\right| \quad \text{a contradiction}
\end{aligned}$$

It turns out that the convergent $\frac{a_i}{b_i} = \frac{m}{r}$ satisfying these conditions will be the last convergent with $b_i < N$, so just take the second last convergent.

(Shor's period finding algorithm for $a^x \bmod M$)

We're just about done with Shor's integer factorization algorithm — we just have to put the pieces together and implement modular exponentiation on a quantum computer ಠ_ಠ.

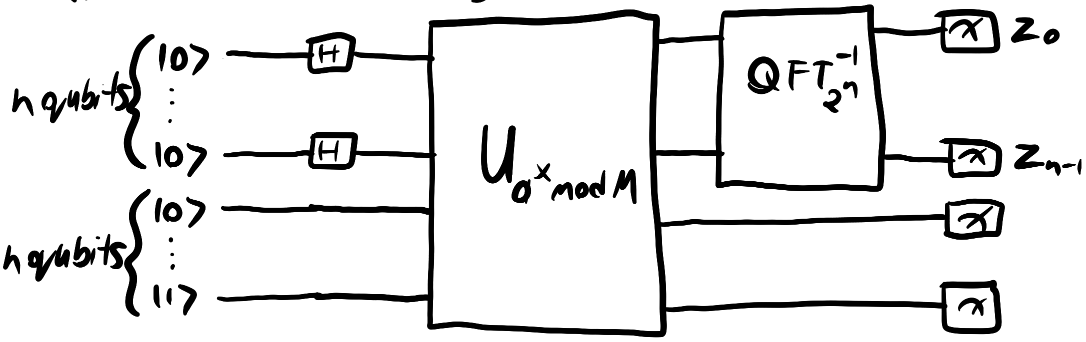
Let's recap the full algorithm first:

Shor's algorithm (KLM's analysis)

Given integers M & a, to find r such that

$$a^r \equiv 1 \bmod M:$$

1. Set $n = \lceil 2\log N \rceil$ (i.e. $2^n \geq 2r^2$)
2. Run the following circuit to get $z = z_0 \cdots z_{n-1}$



3. Run continued fractions on $\frac{z}{2^n}$ until $\left|\frac{z}{2^n} - \frac{a}{b}\right| \leq \frac{1}{2^{n+1}}$.
   If no such a & b is found, output FAIL
4. Repeat 2-3 to get another a', b'.
5. Compute $r = LCM(b, b')$.
6. If $a^r \equiv 1 \bmod M$ return r, else fail.

Theorem

Shor's algorithm outputs the correct period r with probability at least $\frac{384}{\pi^6} > 0.399$ and runs in time $O(n^2) = O(\log^2 M)$ in the black-box query model.

(Construction of the oracle)

Shor's algorithm succeeds where previous ones failed to find a real quantum speed-up by using a concrete function for the oracle - one where knowledge of its implementation doesn't help a classical algorithm to solve the period finding algorithm faster. This is of course the modular exponentiation oracle

$$U_a : |x\rangle|0\rangle \longrightarrow |x\rangle|a^x \bmod N\rangle$$

<u>However</u> now that we have an explicit oracle, we need to do the analysis to be sure it can be implemented efficiently.

(Python implementation)

Here's a naive python implementation

```
1. def modExp(a, N, x):
2.     p = 1
3.     for i in range(0, x):
4.         p = p*a
5.     return p % N
```

There's an obvious problem: x is itself an n-bit number which makes the algorithm exponential in n! In any case, we can still ask how we might implement this on a quantum computer — in particular, how would we do the loop? We only have access to the bits of x and bit-wise controls!

Well, how would we do it classically using the binary expansion

$$x = x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \cdots + x_1 \cdot 2 + x_0$$

(Modular exponentiation, bitwise)

```
def modExp(a, N, [x0, x1, ..., x(n-1)]):
    p = 1
    for i in range(0, n):
        if x_i = 1
            p = p * a^(2^i)
    return p % N
```

can be implemented efficiently by repeated squaring now

$$a^{2^i} = \underbrace{(((a^2)^2)^2 \cdots)^2}_{i \text{ times}}$$

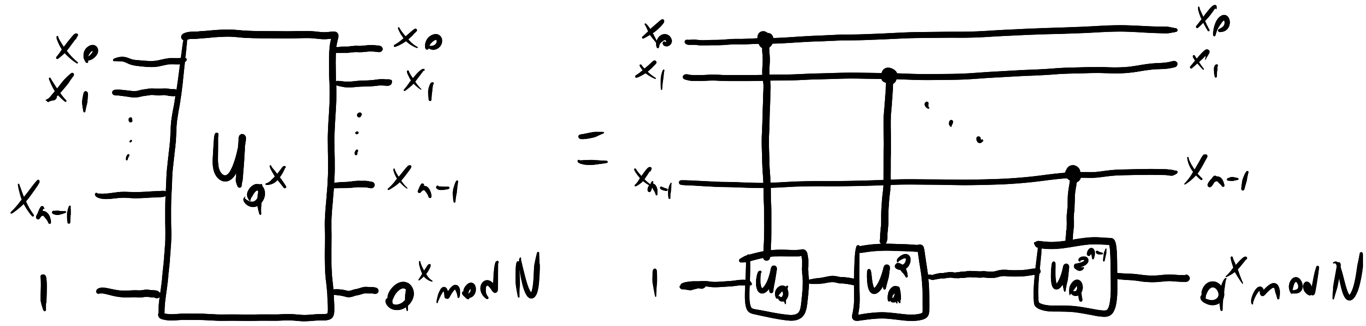
So we have a classical algorithm which is efficient in n, but how to translate it into a quantum algorithm? Well for each bit i of x, we multiply by a^{2^i} if $x_i = 1$ and nothing otherwise — this is just a controlled multiplication by a^{2^i}! Now, for practical purposes it will make more sense to do modular multiplication, which is valid since

$$a^x a^y \pmod N = ((a^x \bmod N)(a^y \bmod N)) \bmod N$$

Let $U_a : |y\rangle \longmapsto |a \cdot y \pmod N\rangle$. Then we can implement the modular exponentiation oracle

$$U_{a^x} : |x\rangle|y\rangle \longmapsto |x\rangle|a^x \cdot y \bmod N\rangle$$

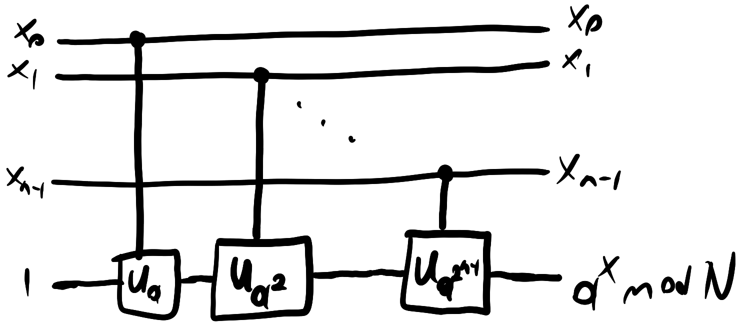
as



Now, this is technically exponential since we can't (obviously) do repeated squaring of an oracle. Instead, we can observe that

$$\overbrace{a\cdot a\cdots a}^{m\text{ times}}\cdot y \bmod N = (a^m \bmod N)\cdot y \bmod N$$

In other words, $U_a^m = U_{a^m \bmod N}$, so we only need to (classically) compute $a^{2^i} \bmod N$ for $i=0,1,\dots,n-1$ and then we have a circuit consisting of n controlled modalar multipliers



To actually complete the algorithm, we would need to further implement modalar multiplication reversibly. For now we can satisfy ourselves knowing that reversible computation can efficiently simalate classical computation, but it's important to note that <u>this takes the bulk of the work in Shor's</u> algorithm, so to really understand the complexity in practical terms we need to carry the analysis all the way through. This is something called resource estimation and is broadly speaking why quantum compilers (like mine)[1,2] exist.

1. github.com/meamy/feynman
2. github.com/softwareQinc/staq