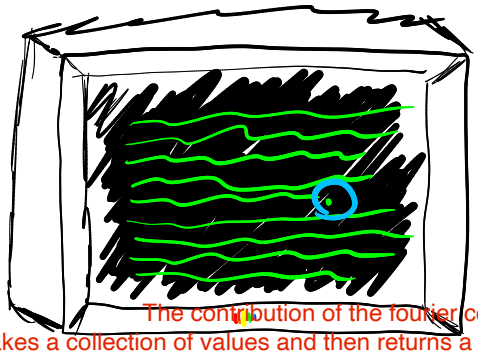
CMPT 476 Lecture 21

Quantum period finding



The contribution of the fourier coefficient:
fourier transformation takes a collection of values and then returns a collection of functions that sum to that joined value.
transform applied on something that has already been affected by the fourier transform will return you the same thing (almost)
since you already computed the sum of the functions which get that value.
The fourier coefficient tells you how important that function's value is for finding the period.

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)

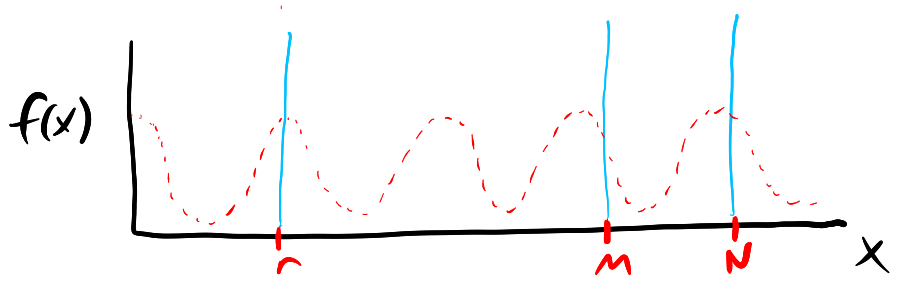
Input: A function $f: \mathbb{Z}_N \to \mathbb{Z}_N$

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

Goal: Find the period r.

What does it mean to extend the function?
Translate the period?

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.

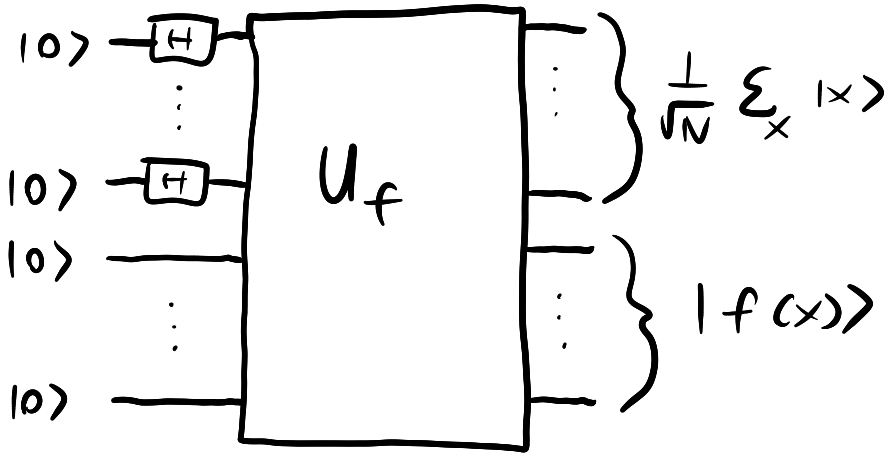
(Shor's period finding algorithm)

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 U_f to get $\frac{1}{\sqrt{N}} \sum_x |x\rangle |f(x)\rangle$

Oracle for f

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,

Comparing the first n registers of two different states, they will differ by a factor of the period.

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

so we can write the resulting state as

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

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

normalization factor

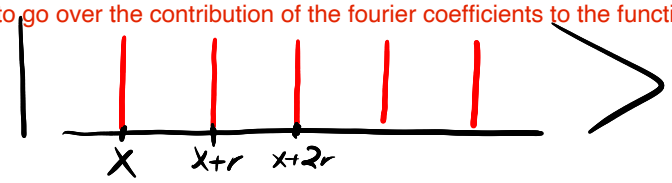
↑ how many periods fit in N

$L \approx N/r$

The QFT on a bit string of length n is equivalent to doing the n Hadamard gates.

We can visualize this state as

Applying the discrete fourier transform on this graph will give you the peaks, the non-zero fourier coefficients. Ask professor to go over the contribution of the fourier coefficients to the function.



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

QFT_N

(we hope)

More formally,

Substitute x + kr for x in the expression of the QFT.

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

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

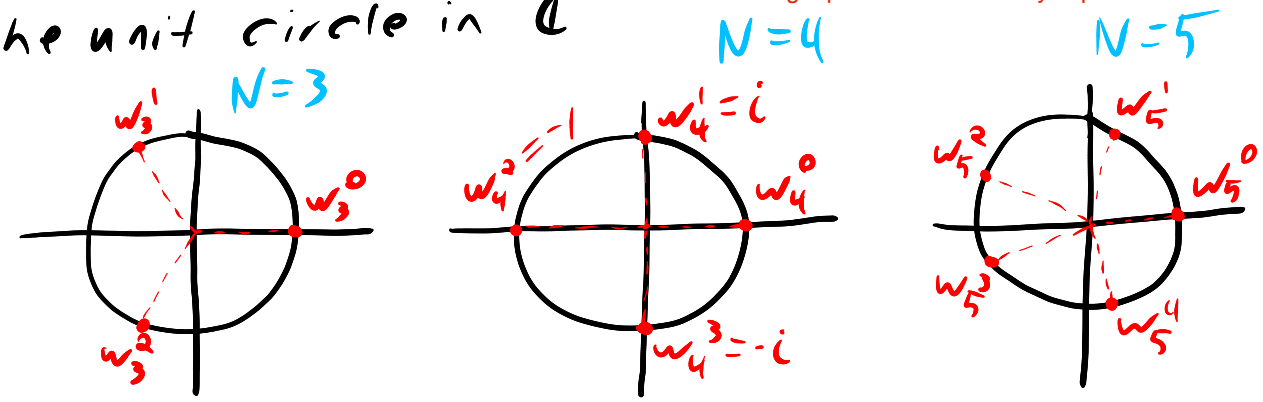
(Roots of unity)

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

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

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 $\mathbb{C}$

Is a root being a primitive root of unity important?



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

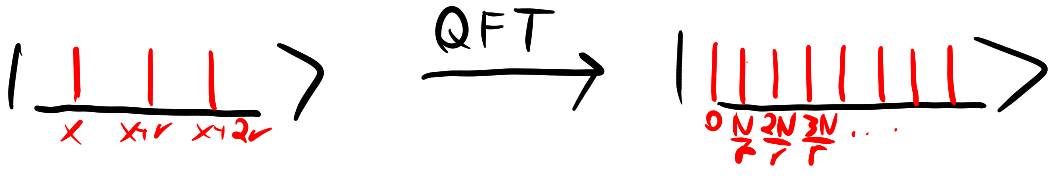
$$\frac{1}{\sqrt{L \cdot N}} \cdot \sum_{k=0}^{L-1} \omega_N^{xz + kr} = \frac{\omega^{xz}}{\sqrt{L \cdot N}} \sum_{k=0}^{L-1} \omega_N^{krz}$$

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

L is the number of periods that fit into N.

$x \mapsto \omega_N^{kx}$

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



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

For the roots of unity, if they are not raised to an integer multiple of M, then they will destructively interfere.
If it is raised to an integer multiple of M, then it will constructively interfere.

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

How do we have that these two are equivalent?

If z = hM

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

Do we obtain L since the exponent above the root is a multiple of L, so for each root we will always obtain 1?

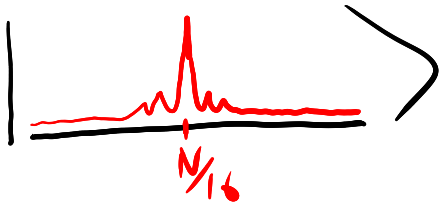
So if we take a few samples we would get

$$m_1 L, m_2 L, m_3 L, \ldots$$

How are we obtaining these samples?

With high probability, $m_1, m_2, m_3, \ldots$ share no common primes, so $GCD(m_1 L, m_2 L, m_3 L, \ldots) = L = N/r$ 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 not divide N

A function that is 15 periodic is similar to a function that is 16 periodic.
A fourier transform is a sequence of samples.
If we decompose our function into a collection of nth roots of unity, they will be powers of two.

Well, we saw constructive interference when

Fourier transform tells you where the peaks are.
A function that is n/k periodic has peaks where the n/k period would be, but larger peaks in other locations as well.

$z = mL = mN/r$ 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...

We need to find the closest integer to N/r.
Epsilon is at most 1/2 since N/r can be at most 1/2 away from an integer.

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

Does it matter if we add or subtract epsilon since in either case we do not have an exact period?
No?

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

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

The first term disappears since we are multiplying by one?
Yes. You have an nth root of unity raised to a multiple of n.

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

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

Taking the square root of an nth root of unity gives you a 2nth root of unity.

$$\sum_{k=0}^{L-1} w_N^{kr \cdot \frac{1}{2}} = \sum_{k=0}^{L-1} w_{2N}^{kr}$$

How am I obtaining a 1 here?

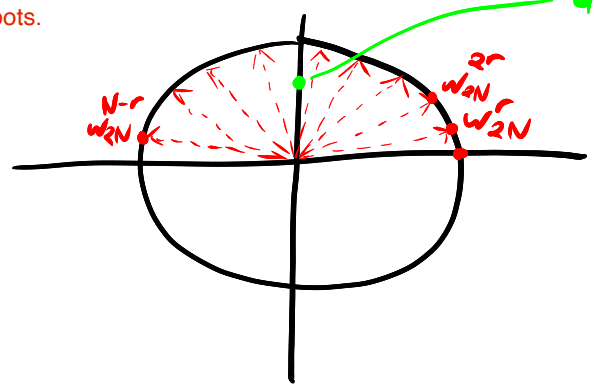
$$= 1 + w_{2N}^{r} + \cdots + w_{2N}^{(L-1)r} \approx w_{2N}^{N-r}$$

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

Could you please explain how this sums to -1?

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

We sum up only the first half of the roots of unity since we only get the first L roots and we have 2N roots.



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

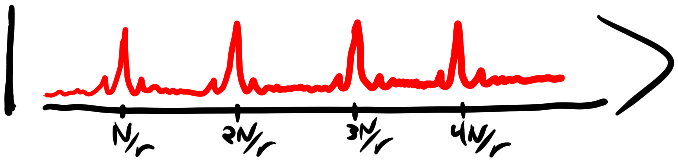
The probability of measuring this z is hence

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

If $2N = r$, then we get an integer multiple of n. We have approximately r of these periods. Ask professor to elaborate on this.

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:

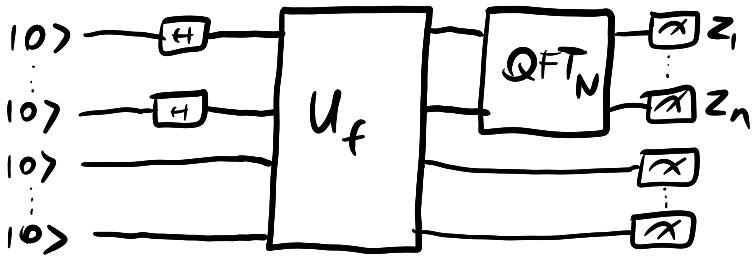
We can pump up the success probability by our choice of $N = 2^n$ where n is the number of qubits. If $r \leq M$ then choosing $N \gg M$ can make N/r closer to an integer and boost the chances of success. How much?

N is far greater than M, but by how much?

(Just kidding. The number theorists say ≈ 1)

(Finding the period)

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



We cannot take the GCD of multiple of the z values since none of them are the actual period and are approximations. We used continued fractions for these.

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 $\frac{436}{100} = 4 + \frac{36}{100}$

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

← take reciprocal so that we can recursively expand...

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

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

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

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

(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

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

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.

Expanding out a/(2^n) will give you exactly b/c in O(n) time.

(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

M is the modulus we wish to factor.

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

I understand this expression, but I am lost on how the following sentence follows from this.

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

If we choose N strategically, taking the sample of n which differs by a factor of 1/2 will lead to Z/N satisfying the conditions of the theorem.

Taking convergents will allow us to find m and r since we can get it exactly in O(n) time.
However, if m and r reduce, then we do not obtain m and r but their simplified forms.

How do we know when to truncate?

When did we obtain that 1/(2N) is equivalent to 1/(2M^2)?

Well, we know $\left|\frac{z}{N} - \frac{q_i}{b_i}\right| \leq \left|\frac{1}{2N}\right| = \left|\frac{1}{2M^2}\right|$ when $\frac{q_i}{b_i} = \frac{n}{r}$, and $r \leq n$. Now, suppose $\frac{q_j}{b_j}$ is another distinct

We need to take a few samples to compute r and then take their lowest common multiple.

Why do I want to take the lowest common multiple?

Taking the LCM will help you discover r since all the reduced values will have the least common multiple of r.

convergent with these properties. Then

$$\left|\frac{1}{2M^2}\right| + \left|\frac{1}{2M^2}\right| \geq \left|\frac{z}{N} - \frac{q_i}{b_i}\right| + \left|\frac{q_j}{b_j} - \frac{z}{N}\right|$$

$$\geq \left|\frac{q_j}{b_j} - \frac{q_i}{b_i}\right|$$

$$= \left|\frac{q_j b_i - q_i b_j}{b_j b_i}\right|$$

$$\geq \left|\frac{1}{b_j b_i}\right| \quad \text{since } \frac{q_i}{b_i} \neq \frac{q_j}{b_j}$$

$$> \left|\frac{1}{M^2}\right| \quad \text{a contradiction}$$

It turns out that the convergent $\frac{q_i}{b_i} = \frac{n}{r}$ 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 😊.

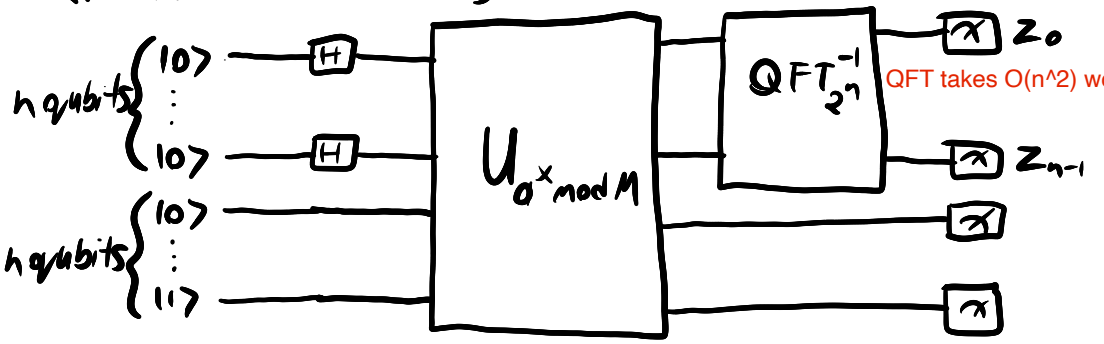
Let's recap the full algorithm first:

Shor's algorithm (KLM's analysis)

Given integers M & a, to find r such that

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

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



QFT takes O(n^2) work

3. Run continued fractions on $\frac{z}{2^n}$ until $\left|\frac{z}{2^n} - \frac{a}{b}\right| \leq \frac{1}{2 \cdot 2^n}$.
   If no such a & b is found, output FAIL

If z is not equal to some integer multiple of r, then the step fails.

4. Repeat 2-3 to get another a', b'.
5. Compute $r = LCM(b, b')$.

Ask about how r is computed here.

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 $O(n^3) = O(\log^3 M)$ in the black-box query model.

If you implement strassan multiplication, the issue is with the number of ancillas used.

You want to do simple things on quantum computers so you do not use up more space.

(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

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

However 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
```

Why is the algorithm exponential in 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

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

Text

(Modular exponentiation, bitwise)

```
1. def modExp(a, N, [x_0, x_1, ..., x_{n-1}]):
2.     p = 1
3.     for i in range(0, n):
4.         if x_i = 1
5.             p = p * a^{2^i}
6.     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

Controlled modular multiplier a^x mod n

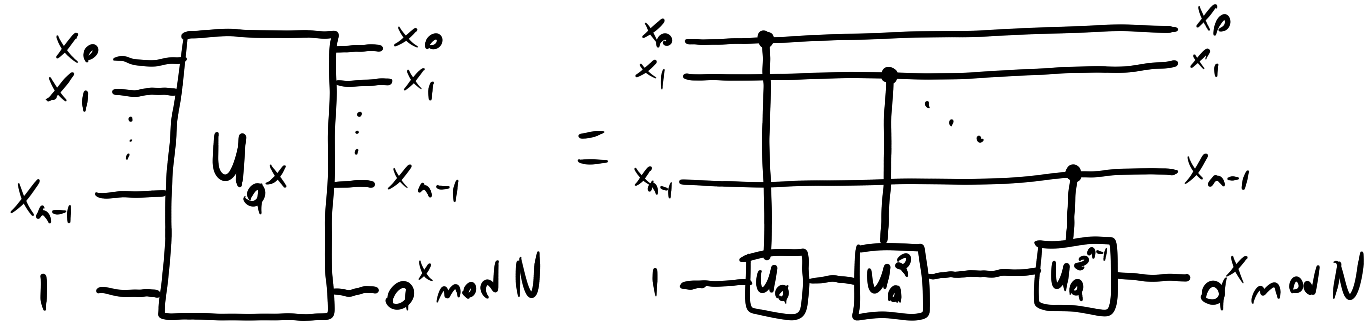
$$a^x a^y \pmod N = ((a^x \bmod N)(a^y \bmod N)) \bmod N$$

We need to do modular exponentiation conditional on the bits of x.

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

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

as



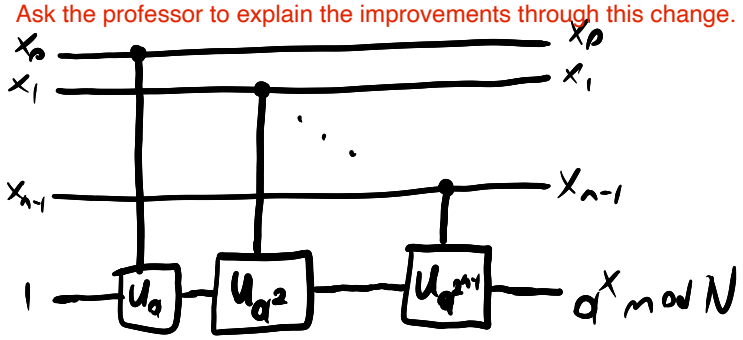
Now, this is technically exponential since we can't (obviously) do repeated squaring of an oracle. Instead, we can observe that

Is this exponential in terms of the number of times you multiply by a?

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

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

I am confused about the benefit brought forth by this modification. Ask the professor to explain the improvements through this change.



To actually complete the algorithm, we would need to further implement modular multiplication reversibly. For now we can satisfy ourselves knowing that reversible computation can efficiently simulate classical computation, but it's important to note that this takes the bulk of the work in Shor's 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