Lecture 20, Tues April 4: Shor, Quantum Fourier Transform

Last time we started in on Shor's algorithm, a quantum algorithm that can factor N into $p \times q$ in polynomial time by reducing the problem to period-finding. Today we'll start to see how to solve period-finding in polynomial time. Before we do, though, a conceptual clarification:

Is Shor's algorithm *provably* faster than any classical algorithm for the same task?

If by "Shor's algorithm," we mean the period-finding core of the algorithm, then the answer is yes. As we saw in the last lecture, there's a provable speedup from $\sim \sqrt{s}$ queries to only O(1).

If, on the other hand, we think of Shor's algorithm as a way to do integer factorization, then the speedup remains conjectural. Indeed it has to, because no one has even proven that P≠NP---and if P=NP, then of course factoring is classically easy.

The way to reconcile these two statements is simply to observe that there are many ways to factor a number besides by reducing factoring to period-finding. In fact, the best known classical factoring algorithms---the Quadratic Field Sieve and Number Field Sieve mentioned in the last lecture---do much better than $\sqrt{s}$ by exploiting additional structure in the factoring problem. The most we can currently prove is that Shor's algorithm achieves an exponential speedup over any classical factoring algorithm *that works via the last lecture's reduction to black-box period-finding*.

We left off last time with our quantum state in the form

$$|r\rangle + |r+s\rangle + |r+2s\rangle + \ldots$$

Now we'll see how to measure this state to extract useful information about the period s.

In science and engineering, any time you have a periodic signal and you're trying to extract its period, there's a single tool that gets called upon…

The Fourier Transform!

There are many types of Fourier transforms: continuous, Boolean, etc. For us, though, the Q-dimensional Fourier transform will be the $Q \times Q$ matrix F_Q defined as follows:

$$\langle i \,|\, F_Q \,|\, j \rangle = \omega^{ij} / \sqrt{Q}, \quad \text{where } \omega = e^{2\pi i/Q} \text{ is a } Q^{\text{th}} \text{ root of unity.}$$

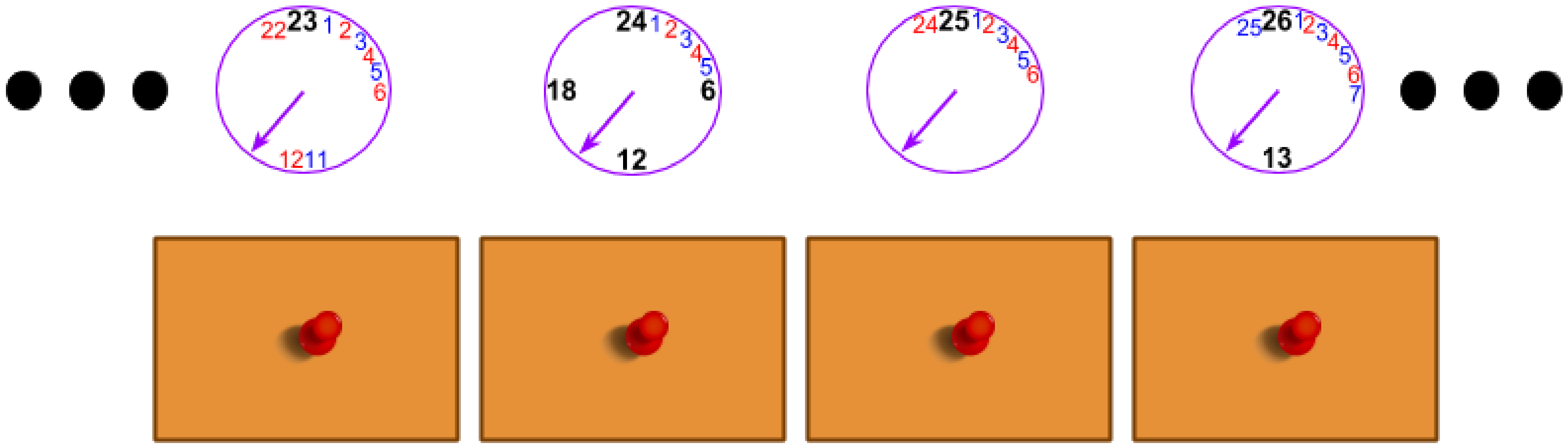
So, the ij^{th} entry of the matrix involves ω raised to the ij^{th} power.

Here's some useful intuition for how it works:

In grad school, you can easily fall into a 26-hour-per-day cycle. So one day you wake up at 8am, the next day you wake up at 10am, then 12pm, and so forth.

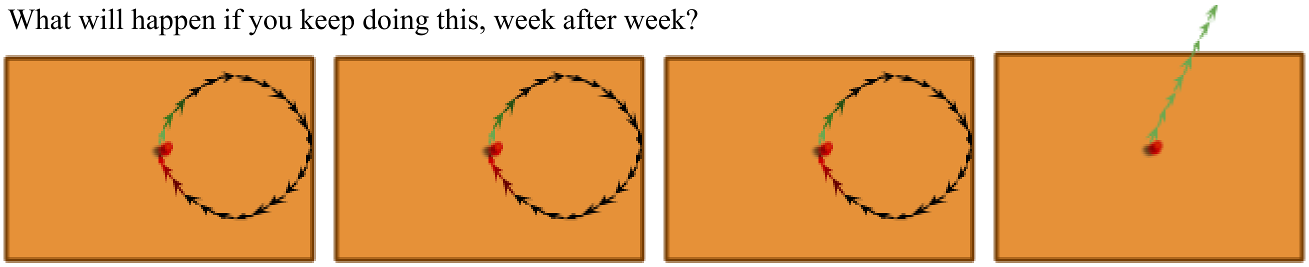
Suppose you've fallen into such a cycle, and you want to figure out how long the cycle is, without doing any complicated calculations like subtraction.

What you can do is install a series of clocks in your room, each tracking "days" of different lengths. So you'd have a 23-hour clock, a 24-hour clock, a 25-hour clock, etc. In addition, install a bulletin board below each clock and place a single thumbtack in the center.



Now: every time you wake up, for each clock, move the thumbtack one inch in the direction that the hour hand points.

What will happen if you keep doing this, week after week?



The thumbtack corresponding to the 26-hour clock will always move in the same direction.

This is constructive interference!

And the same can be said for the 13-hour clock (as well as the 2- and 1-hour clocks).

All the others, the 23-hour clock, the 24-hour clock, etc, will have the thumbtack move around, but sometimes one way, sometimes another way, so that it eventually returns to the origin.

The Quantum Fourier Transform is essentially this, but with quantum-mechanical amplitudes instead of thumbtacks.

There are two questions we need to answer here:

1. How do we implement the Quantum Fourier Transform using a small quantum circuit?

   Since it's a $Q \times Q$ matrix, it's not obvious whether we can do it using a circuit with polylog(Q) gates.

2. Once we've applied the QFT and measured, how do we make sense of the outcome?

   Complications can arise because the period s doesn't divide Q (in all likelihood).

To answer the first question, let's look at some examples.

$F_2 = H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \end{pmatrix}$ because it's $\begin{pmatrix} \omega^{0*0} & \omega^{0*1} \end{pmatrix}$

$$\begin{pmatrix} 1 & -1 \end{pmatrix} \qquad \begin{pmatrix} \omega^{1*0} & \omega^{1*1} \end{pmatrix}$$

$$F_4 = \tfrac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \qquad \text{i.e.} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^2 & 1 & \omega^2 \\ 1 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix}$$

For F_8 you'd have
$$\frac{1}{2\sqrt{2}}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix}$$

keep in mind that you can simplify

$\omega^2 = i$

$\omega^4 = -1$

We could design an algorithm to apply these matrices by brute force, but there's a better way. This method is related to one of the most widely used classical algorithms…

The FFT – Fast Fourier Transform

Suppose we have a vector of length *Q*, and we want to apply a *Q*×*Q* matrix *A* to it. In general this takes ~Q^2 operations. However, *if* we know that *A* is the Fourier transform, then the FFT lets us apply it in only O(*Q* log *Q*) steps, by exploiting regularities in the Fourier matrix.

<u>So what regularity *is* there in the Fourier matrix?</u>

Look at
$$F_4 = \tfrac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

If we swap the second and third columns, we get

$$\begin{array}{rl} & \tfrac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} \\ \text{This quadrant looks like H…} & \phantom{\tfrac{1}{2}}\begin{pmatrix} 1 & -1 & i & -i \end{pmatrix} \\ & \phantom{\tfrac{1}{2}}\begin{pmatrix} 1 & 1 & -1 & -1 \end{pmatrix} \\ \text{This one too…} & \phantom{\tfrac{1}{2}}\begin{pmatrix} 1 & -1 & -i & i \end{pmatrix} \end{array}$$

In fact, we can rewrite the whole matrix as

$$\begin{pmatrix} H & AH \\ H & -AH \end{pmatrix} \qquad \text{for } A = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \text{ [the phase shift]}$$

You can do the same procedure for F_8

$$\frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix}$$

Moving all the odd columns to the left, and the even columns to the right, and simplifying the ω's, we get

$$\frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i & \omega^1 & \omega^3 & \omega^5 & \omega^7 \\ 1 & -1 & 1 & -1 & i & -i & i & -i \\ 1 & -i & -1 & i & \omega^3 & \omega & \omega^7 & \omega^5 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & i & -1 & -i & \omega^5 & \omega^7 & \omega & \omega^3 \\ 1 & -1 & 1 & -1 & -i & i & -i & i \\ 1 & -i & -1 & i & \omega^7 & \omega^5 & \omega^3 & \omega^1 \end{pmatrix}$$

Which, again, we can now rewrite as

$$\begin{pmatrix} F_4 & AF_4 \\ F_4 & -AF_4 \end{pmatrix}$$

This time $A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \omega & 0 & 0 \\ 0 & 0 & \omega^2 & 0 \\ 0 & 0 & 0 & \omega^3 \end{pmatrix}$

You can work out why it happens on your own, but it turns out that we can define F_Q in terms of this nesting recurrence:

$$\boldsymbol{F_Q} = \frac{1}{\sqrt{2}} \begin{pmatrix} F_{Q/2} & AF_{Q/2} \\ F_{Q/2} & -AF_{Q/2} \end{pmatrix}$$

Notice that applying F_Q takes linear time plus twice however much time it takes to apply $F_{Q/2}$, so solving the recurrence, the overall running time of the FFT algorithm is O(Q log Q).

In the quantum case, we're actually interested in the unitary transformation

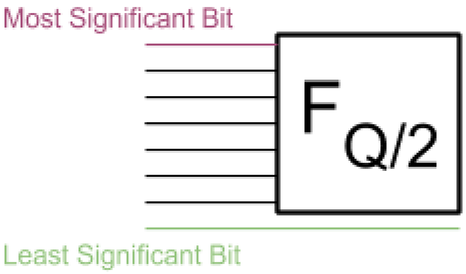
$|\Psi\rangle \rightarrow F_Q|\Psi\rangle$ where $|\Psi\rangle$ is a quantum state of $\log_2 Q$ qubits.

In one sense, our new goal is less ambitious: we don't need the result vector written explicitly in memory anywhere; it only needs to be encoded implicitly in a vector of amplitudes. In another sense, though, our new goal is more ambitious, since we now want to apply a Fourier transform in time polynomial in only log Q.

So how do we do it?

We can use the same recursion that we used for the FFT, *plus* the additional observation that that recursion behaves "linearly" with respect to quantum states.

Let's think of our log(Q) qubits as representing an integer from 0 to Q-1 in binary notation. And let's order the bits of that integer from most to least significant. Then we can try applying the circuit $F_{Q/2}$ to the "first half" of the number: in other words, to all but the least significant bit.



This corresponds to applying the matrix

$$\begin{pmatrix} F_{Q/2} & \\ & F_{Q/2} \end{pmatrix}$$

To get an A on the bottom right quadrant, as in the FFT algorithm, we can then apply a control-A:

$$\begin{pmatrix} F_{Q/2} & \\ & AF_{Q/2} \end{pmatrix}$$

We can implement A using a linear number of gates, because it simply amounts to the following:

```
If the most significant bit is 1, then do a rotation
Else if the second bit is 1, then do a rotation that's half as big
Else if the third bit is 1,          … etc…
```

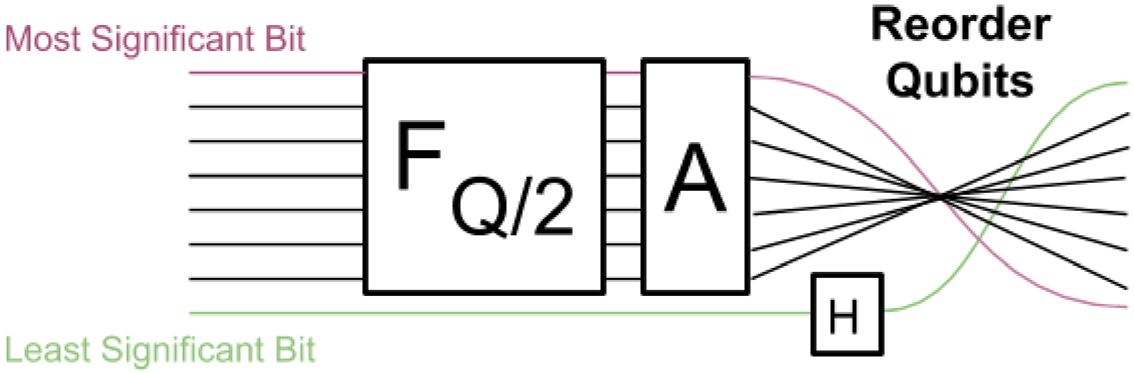
By the time you reach the last couple of bits, the rotation is exponentially small.

When they first learn about Shor's algorithm, some people object that it's "unphysical," since there's no practical way to apply such tiny rotations. But it turns out that the exponentially small rotations *don't matter* for the algorithm---indeed, there's a theorem that says that you can just *omit* these tiny rotations. Doing so even improves the size of the quantum circuit that implements F_Q, from $O(q^2)$ to $O(q \log q)$.

The final step to get the matrix we want is a Hadamard gate.

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} F_{Q/2} & \\ & AF_{Q/2} \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} F_{Q/2} & AF_{Q/2} \\ F_{Q/2} & -AF_{Q/2} \end{pmatrix}$$

So here's our finished quantum circuit:



Note that the reordering of the qubits isn't part of the recursion: it's performed only once, at the very end of the circuit. We'll leave it as an exercise to see why the qubits emerge from the circuit in *reverse* order, with the least significant at the top and the most significant at the bottom. (Hint: it has to do with the least significant, Hadamarded qubit "jumping up" to become the most significant one, at each level of the recursion.)

Okay. So now that we've seen how to implement the Quantum Fourier Transform as a quantum circuit, it's time to answer our second question:
<u>What comes out when we measure, and how can we use it to learn *s*?</u>

We have a state like $\frac{1}{\sqrt{L}}\left(|r\rangle + |r+s\rangle + |r+2s\rangle + \ldots + |r+(L\text{-}1)s\rangle\right)$

And we know that the QFT maps this state to

$$\frac{1}{\sqrt{QL}} \sum_{j=0}^{Q-1} \sum_{l=0}^{L-1} \omega^{(r+ls)j} |j\rangle.$$

What's going on with the above state? Let's start with an easy special case, and only later handle the general case.

The easy case is that s divides Q.
Assuming that, let's answer the question: <u>which *j*'s can be observed, when we measure the above state in the computational basis?</u> This in turn boils down to: <u>for a given *j*, do the various contributions to *j*'s amplitude interfere constructively or destructively?</u>

To answer this question, we can ignore the global phase ω^r and just look at the sum $\sum_{l=0}^{L-1} \omega^{jsl}$.

The key is to identify whether js is a multiple of Q.

- If js **is not** a multiple of Q

Then we have destructive interference because the terms ω^{js}, $(\omega^{js})^2$, $(\omega^{js})^3$, etc. are all pointing in different directions in the complex plane, and they cancel each other out.

Just like the thumbtack's movement coming back to the origin.

- If js **is** a multiple of Q--or equivalently, if $js = kQ$ and $j = kQ/s$ for some integer k.

Then, since ω was a Q^{th} root of unity, the terms ω^{js}, $(\omega^{js})^2$, $(\omega^{js})^3$, ... all point in the same direction in the complex plane, producing constructive interference.

If we repeat this procedure several times, we'll generate a list of such j's, each of which is an integer multiple of Q/s:

$j_1 = k_1Q/s, \qquad j_2 = k_2Q/s, \ldots$

So then we just need to take their GCD to get Q/s itself (with overwhelming probability), from which we can compute s, given our knowledge of Q.

Remember: This is only possible because we assumed that s divides Q.

The harder, general case is that s doesn't divide Q.

In this case, if we calculate the final amplitude for a specific basis state j, then ignoring the global phase ω^r , we'll still get a sum of the form $\sum_{l=0}^{L-1} \omega^{jsl}$. So, how likely we are to observe j will still depend on whether this sum involves constructive or destructive interference.

What changes is that now Q/s isn't an integer--and as a result, neither the constructive nor the destructive interference will be perfect. But we'll see that they're still good enough for the period s to be efficiently recovered.

Let's ask whether j has the form $\lfloor k\frac{Q}{s} \rceil$

for some integer k: in other words, whether it's the nearest integer to some multiple of $\frac{Q}{s}$.

- If $j = \lfloor k\frac{Q}{s} \rceil$

then we'll claim that we'll see *mostly* constructive interference.

- If $j \neq \lfloor k\frac{Q}{s} \rceil$

then we'll claim that we'll see *mostly* destructive interference.

Let's look at the constructive case first.

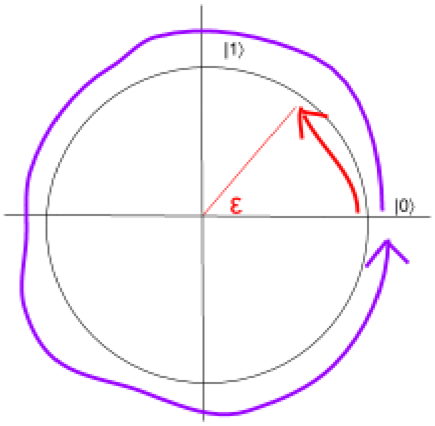
Assume we get a bit lucky, and we have (say) $j = k\frac{Q}{s} + \varepsilon$ where $|\varepsilon| \sim \frac{1}{10}$.

That means that ignoring normalization, the final amplitude of basis state j has the form

$$\sum_{l=0}^{L-1} \omega^{(k\frac{Q}{s}+\varepsilon)sl} = \sum_{l=0}^{L-1} \omega^{kQl}\omega^{\varepsilon sl} .$$

We can go further, and say that the ω^{kQl} term doesn't matter, because

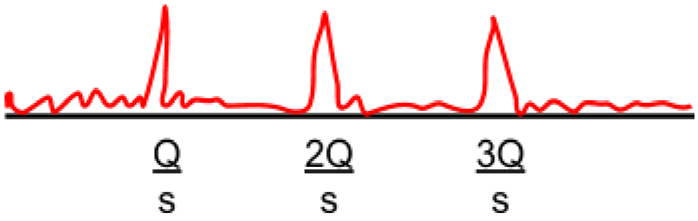
$$\omega^{kQl} = e^{(2\pi i/Q)(kQl)} = e^{2\pi ikl} = 1.$$



So then we're left with the $\omega^{\varepsilon sl}$ part, which amounts to a rotation around an ε fraction of the unit circle.

i.e. mostly constructive interference.

Next suppose *j* isn't the nearest integer to a multiple of *Q/s*. In that case, as we vary *l*, the term ω^{jsl} will loop all the way around the unit circle one or more times. So we'll get mostly destructive interference, except for a small amount of constructive interference from the final rotation.



If you plot the final amplitude as a function of *j*, you get something like the graph on the left.