Last class we discussed the classical part of Shor's algorithm for integer factorization: a poly-time reduction to period finding over $\mathbb{Z}_N$. Specifically, finding the period (also called order) of $f(x) = a^x \mod N$.

Today we'll discuss Shor's period finding sub-routine, which modifies Simon's algorithm to work over $\mathbb{Z}_2^n$ rather than $\mathbb{Z}_2$, using an exponentially faster Fourier Transform than the classical FFT called... The QFT! (Quantum Fourier Transform)
The (discrete) Fourier transform is a tool used in signal processing to translate a signal like this

\[ f(t) \]

into a sum of oscillating components (sine and cosine) with different frequencies (periods)

\[ \hat{f}(s) \]

Very roughly, the Fourier transform allows one to approximate the constituent periods of a function \( f \) given samples \( f(t) \). So we know intuitively that it should give some information about the period. The question will be how to extract it...

(The discrete Fourier transform)

Let \( f : \mathbb{Z}_N \rightarrow \mathbb{C} \). The discrete Fourier transform of \( f \) is \( \hat{f} : \mathbb{Z}_N \rightarrow \mathbb{C} \) defined as

\[
\hat{f}(y) = \sum_{x=0}^{N-1} f(x) \cdot e^{-2\pi i \frac{xy}{N}}
\]

If we think of \( f \) as a length \( N \) vector of complex numbers, then the DFT sends
Ex.

We've already seen a Fourier transform: \( \hat{\Psi}(\xi) \). Let \( \Psi = [\hat{\Psi}] \in C^2 \). Define \( \Psi : \mathbb{Z}_N \rightarrow C \) as

\[ \Psi(x) = \langle x \xi \rangle \]  
(i.e. the coefficient of \( |x\rangle \))

The Fourier transform of \( \Psi \) is

\[ \hat{\Psi}(\omega) = \sum_{\xi \in \mathbb{Z}_N} \Psi(\xi) e^{-2\pi i \omega \xi} \]

\[ = \sum_{\xi \in \mathbb{Z}_N} \Psi(\xi) (-1)^{\omega \xi} \]

Now \( |\Psi\rangle = \begin{bmatrix} \Psi(0) \\ \Psi(1) \end{bmatrix} = \begin{bmatrix} \sum_{\xi \in \mathbb{Z}_N} \Psi(\xi) \\ \sum_{\xi \in \mathbb{Z}_N} \Psi(\xi) (-1)^{\omega \xi} \end{bmatrix} = \begin{bmatrix} \alpha + \beta \\ \alpha - \beta \end{bmatrix} \)

which is the hadamard transform applied to \( |\Psi\rangle \!

(\text{unnormalized})

The inverse Fourier transform

The inverse Fourier transform maps \( \hat{f} \rightarrow f \), but it's expression as a function of \( \hat{f} \) makes the interpretation of the DFT as breaking a signal into a sum of oscillators more clear:

\[ f(x) = \frac{1}{N} \sum_{\omega=0}^{N-1} \hat{f}(\omega) e^{2\pi i \omega x / N} \]

\[ \text{period} \]  
\[ \text{Fourier coefficient} \]
Before we move onto generalizing the Hadamard gate to a Fourier transform on $\mathbb{Z}_N$, let's think about how we might use it to find periods. Suppose

$$f(x) = e^{\frac{2\pi i 3x}{N}}$$

If we graph (the real part of) $f$ on $\mathbb{Z}_N$ we get

![Graph of $f(x)$](image)

Period = $\frac{N}{3}$

Taking the Fourier transform over $\mathbb{Z}_N$ gives

$$\hat{f}(y) = \frac{1}{N} \sum_{y=0}^{N-1} f(y) e^{-\frac{2\pi i 3y}{N}}$$

We know intuitively that $\hat{f}(3) = N$ and $\hat{f}(y) = 0$ everywhere else, so sampling the Fourier coefficients should give us 3 which would allow us to compute the period $\frac{N}{3}$

![Frequency domain](image)

In general,

$$f(x) \xrightarrow{\text{DFT}_N} \hat{f}(y)$$

\* Sampling the Fourier coefficients (almost) gives you the period!
The Quantum Fourier Transform on \( n \) qubits is the unitary \( n \)-qubit transformation:

\[
\text{QFT}_n: |x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{y \in \mathbb{Z}_2^n} e^{2\pi i xy} |y\rangle
\]

Viewing a state vector \( |f\rangle \) as a function \( f: \mathbb{Z}_2^n \rightarrow \mathbb{C} \) where \( f(x) = \langle x | f \rangle \):

\[
\text{QFT}_n(|f\rangle) = \frac{1}{\sqrt{2^n}} \sum_y \left[ \sum_x f(x)e^{2\pi i xy} \right] |y\rangle
\]

\[
= \frac{1}{\sqrt{2^n}} \sum_y f(y) |y\rangle
\]

So the QFT is the DFT on a state vector. We can visualize such a state vector as:

\[
\begin{array}{c}
\boxed{\text{QFT}}
\end{array}
\]

Note that the inverse QFT is:

\[
\text{QFT}_n^T: |y\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{Z}_2^n} e^{-2\pi i xy} |x\rangle
\]

Example:

What does a Fourier matrix look like?

\[
\text{QFT}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix}
| & |
| & |
\end{bmatrix}
\]

\[
\text{QFT}_4 = \frac{1}{2} \begin{bmatrix}
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\end{bmatrix}
\]
Implementation of the QFT

Classically, the best known algorithm for the DFT runs in time \( O(N \log N) \) where \( N \) is the dimension of the vector. Shor's insight was that the QFT can be computed in \( O(n^2) \) when \( N=2^n \), i.e., quadratic in the number of qubits.

To see how, let \( n=4 \) and recall that in binary,
\[
y = 8y_3 + 4y_2 + 2y_1 + y_0
\]
So expanding the product \( x \cdot y \) in binary we get
\[
x \cdot y = y_3 (8x) + y_2 (4x) + y_1 (2x) + y_0 x
\]
Applying this to the QFT 1-bit state we have
\[
\frac{1}{\sqrt{2^4}} \sum_y e^{i \frac{2\pi}{2^4} xy} |y\rangle = \frac{1}{4} \sum_{y_0 \ldots y_3} w_16^{y_0 y_3} (8x) + y_2 (4x) + y_1 (2x) + y_0 x |y_0 \ldots y_3\rangle
\]
\[
= \frac{1}{4} \left( \sum_{y_0} w_16^{y_0 x} |y_0\rangle \right) \otimes \left( \sum_{y_1} w_16^{y_1 (2x)} |y_1\rangle \right) \otimes \left( \sum_{y_2} w_16^{y_2 (4x)} |y_2\rangle \right) \otimes \left( \sum_{y_3} w_16^{y_3 (8x)} |y_3\rangle \right)
\]
\[
= \left( \frac{10^2 + w_16^{1} 117}{\sqrt{12}} \right) \otimes \left( \frac{10^2 + w_16^{2} 117}{\sqrt{2}} \right) \otimes \left( \frac{10^2 + w_16^{4} 117}{\sqrt{4}} \right) \otimes \left( \frac{10^2 + w_16^{8} 117}{\sqrt{8}} \right)
\]
Note that this is a separable (or unentangled) state, so we can (kind of) proceed bit by bit.
Our goal is

\[
\begin{align*}
&x_0 & \quad & \frac{1}{\sqrt{2}} (10 \oplus x_0 11) \\
&x_1 & \quad & \frac{1}{\sqrt{2}} (10 \oplus x_0 11) \\
&x_2 & \quad & \frac{1}{\sqrt{2}} (10 \oplus x_0 11) \\
&x_3 & \quad & \frac{1}{\sqrt{2}} (10 \oplus x_0 11)
\end{align*}
\]

First observe that \( \omega_{16}^x = (-1)^x \) since \( \omega_{16}^8 = (-1) \).

Moreover, \( x = 8x_3 + 4x_2 + 2x_1 + x_0 \), so

\[
(-1)^x = (-1)^{x_0}
\]

This tells us that the high-order bit \( y_3 \) of the QFT is just \( \frac{1}{\sqrt{2}} (10 \oplus (-1)^{x_0} 11) = H(x_0)! \)

So, we know one gate (and a qubit reordering)

\[
\begin{align*}
&x_0 & \quad & H \\
&x_1 & \quad & x_1 \\
&x_2 & \quad & x_2 \\
&x_3 & \quad & x_3 \\
& & & \frac{1}{\sqrt{2}} (10 \oplus (-1)^{x_0} 11)
\end{align*}
\]

We can do the same with the next bit

\[
\omega_{16}^{y_3} = (-1)^{y_3} = (-1)^{8x_3 + 4x_2 + 2x_1 + x_0} = (-1)^{x_1 + x_0} = (-1)^{x_1}(-1)^{x_0}
\]

\[
\frac{1}{\sqrt{2}} (10 \oplus (-1)^{x_1}(-1)^{x_0} 11) = \begin{cases} 
H(1x_1) \text{ if } x_0 = 0 \\
S H(1x_1) \text{ if } x_1 = 1
\end{cases}
\]

Recall that \( S = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \)}
Now, this is exactly an H gate followed by an S gate if \( x_0 = 1 \), i.e. a \textbf{controlled-S} gate.

We also know that this must come before we prepare the high-order bit, since when we do we "lose \( x_0 \). So now we have

\[
\begin{align*}
X_0 & \quad \text{H} \quad X_2 \\
X_1 & \quad \text{H} \quad S \\
X_2 & \\
X_3 & \quad \frac{1}{\sqrt{2}} \left( |0\rangle + (-1)^{x_0} |1\rangle \right) \\
& \quad \frac{1}{\sqrt{2}} \left( |0\rangle + (-1)^{x_0} |1\rangle \right)
\end{align*}
\]

Repeating with the next bit,

\[
W_{16}^{2X} = W_8^{8X_3 + 4X_2 + 2X_1 + X_0} = (-1)^{X_2 \cdot X_1 \cdot X_0} W_8
\]

Same story here — H \( (x_2) \) followed by phase rotations of 1 and \( W_8 = e^{\alpha \tau i/8} \) conditional on \( X_1 \) and \( X_0 \), respectively. Recalling that

\[
T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\alpha \tau i/8} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & W_8 \end{bmatrix}
\]

We have

\[
\begin{align*}
X_0 & \quad \text{H} \\
X_1 & \quad \text{H} \quad S \\
X_2 & \quad \text{H} \quad S \quad T \\
X_3 & \\
& \quad \frac{1}{\sqrt{2}} \left( |0\rangle + (-1)^{x_0} |1\rangle \right) \\
& \quad \frac{1}{\sqrt{2}} \left( |0\rangle + (-1)^{x_0} |1\rangle \right)
\end{align*}
\]
Finally, $w_{16}^x = (-1)^{x_3} i^{x_2} w_8^x w_{16}^{x_0}$. Denoting

$$\sqrt{T} = \begin{bmatrix} 1 & 0 \\ 0 & w_{16} \end{bmatrix}$$

and following the pattern,

And that's $QFT_{16}$

(QFT$_{2^n}$)

Let $R_k = \begin{bmatrix} 1 & 0 \\ 0 & w_{2^k} \end{bmatrix}$. Then QFT$_{2^n}$ can be implemented as (ignoring the final reordering)

(Complexity of the QFT)

We first have $n$ gates, followed by a QFT$_{2^{n-1}}$, so

$$n + (n-1) + (n-2) + \cdots + 1 = \sum_{i=1}^{n} i \in O(n^2)$$
A note on small angle controlled rotations

The naive complexity analysis of the QFT hides a very important issue: the complexity of implementing the controlled rotations \( C-R_k \). To be capable of implementing any QFT, we would theoretically need an infinite gate set available to our computer. Given the finite gate set assumption, these \( C-R_k \) gates will generally need to be approximated.

**Problem: How to approximate a 2-qubit gate?**

Well in general we want to decompose it into CNOT and single qubit gates, then approximate. In this case

\[
\begin{align*}
R_k & \approx \begin{bmatrix} R_{k+1} & 0 \\ 0 & R_{k+1} \end{bmatrix} \\
& = \begin{bmatrix} R_{k+1} & 0 \\ 0 & R_{k+1} \end{bmatrix}
\end{align*}
\]

To see why, observe that for \( x, y \in \{0, 1\} \)

\[
2xy = x + y - (x \oplus y)
\]

This is another Fourier Transform! (just on \( \mathbb{Z}_2 \)). Now the effect of \( C-R_k \) on \( |x\rangle |y\rangle \) is

\[
C-R_k |x\rangle |y\rangle = W^x_{2k} |x\rangle (y) \langle y| \\
= W^x_{2k+1} |x\rangle |y\rangle \\
= W^x_{2k+1} W^y_{2k+1} W^{-(x \oplus y)}_{2k+1} |x\rangle |y\rangle
\]
Noting again that \( R_{k+1}(x) = w_{3k+1}(x) \), we just need to apply 3 phase rotations on the basis vectors \( |x\rangle \), \( |y\rangle \), and \( |x\otimes y\rangle \) which can be prepared with a CNOT gate and then uncomputed after the rotation.

Assuming each of these gates takes \( O(\log^3(1/\varepsilon)) \) gates to approximate, this brings the effective complexity of the QFT closer to

\[
O(n^2 \log^3(1/\varepsilon))
\]

An alternative is to note instead that

\[
R_k \approx I \quad \text{as} \quad k \to \infty
\]

and instead drop any \( R_k \) gates with \( k \geq k_0 \). In practice we need to do both and more optimizations to implement the QFT efficiently.

(A note on the reordering)

In principle we could perform a final qubit reordering to implement the QFT exactly with \( \lfloor 3n/2 \rfloor \) swap gates, but in practice we usually don't need to — when compiling an algorithm to a circuit it's just as easy to re-index all gates that follow. That is, rather than doing

\[
\text{you may as well do}
\]

\[
\text{U} \\
\text{x} \\
\text{U} \\
\text{y}
\]

\[
\text{x} \\
\text{y} \\
\text{U} \\
\text{x}
\]