CMPT 476 Lecture 20

The Quantum Fourier transform

Last class we discussed the classical part of Shor's algorithm for integer factorization: a poly-time probabilistic reduction to period finding over $\mathbb{Z}_N$ ~ specifically, finding the period (also called order) of

$$f(x) = a^x \bmod N$$

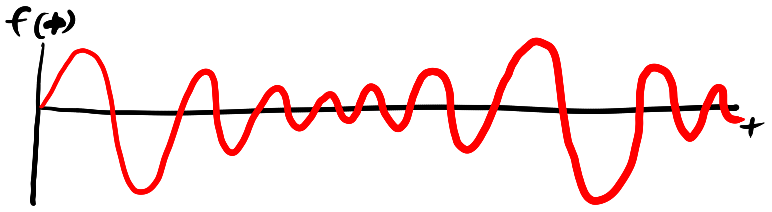
Today we'll discuss Shor's period finding sub-routine, which modifies Simon's algorithm to work over $\mathbb{Z}_{2^n}$ rather than $\mathbb{Z}_2^n$, using an exponentially faster Fourier Transform than the classical FFT called...

The QFT!

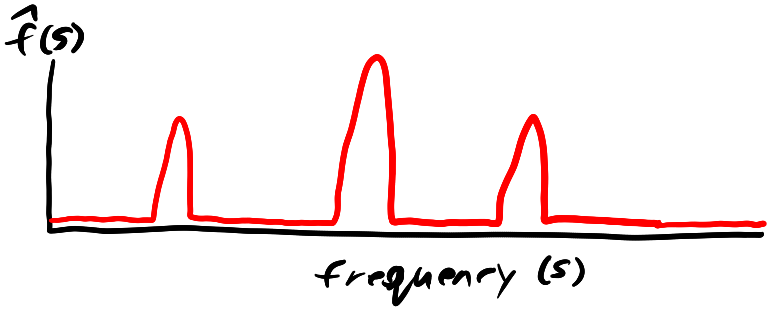
(Quantum Fourier Transform)

(Fourier transforms & periodicity)

The (discrete) Fourier transform is a tool used in signal processing to translate a signal like this



into a sum of oscillating components (sin & cos) with different frequencies (periods)



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 $f : \mathbb{Z}_N \to \mathbb{C}$. The discrete Fourier transform of f is $\hat{f} : \mathbb{Z}_N \to \mathbb{C}$ defined as

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

If we think of f as a length N vector of complex numbers, then the DFT sends

$$\begin{bmatrix} f(0) \\ f(1) \\ \vdots \\ f(N-1) \end{bmatrix} \xrightarrow{DFT} \begin{bmatrix} \hat{f}(0) \\ \hat{f}(1) \\ \vdots \\ \hat{f}(N-1) \end{bmatrix}$$

Ex.

We've already seen a Fourier transform: H!

Let $|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \in \mathbb{C}^2$. Define $\psi : \mathbb{Z}_2 \to \mathbb{C}$ as

$$\psi(x) = \langle x|\psi\rangle \quad \text{(i.e. the coefficient of } |x\rangle)$$

The fourier transform of ψ is

$$\hat{\psi}(y) = \sum_{x \in \mathbb{Z}_2} \psi(x)\, e^{-2\pi i \cdot \frac{xy}{2}}$$

$$= \sum_{x \in \mathbb{Z}_2} \psi(x)\,(-1)^{xy}$$

Now $|\hat{\psi}\rangle = \begin{bmatrix} \hat{\psi}(0) \\ \hat{\psi}(1) \end{bmatrix} = \begin{bmatrix} \sum_x \psi(x) \\ \sum_x \psi(x)(-1)^x \end{bmatrix} = \begin{bmatrix} \alpha + \beta \\ \alpha - \beta \end{bmatrix}$

which is the hadamard transform applied to $|\psi\rangle$!
(unnormalized)

(The inverse Fourier transform)

The inverse Fourier transform maps $\hat{f} \to f$, but its expression as a function of $\hat{f}$ makes the interpretation of the DFT as breaking a signal into a sum of oscillators more clear:

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

$\frac{N}{y}$ period

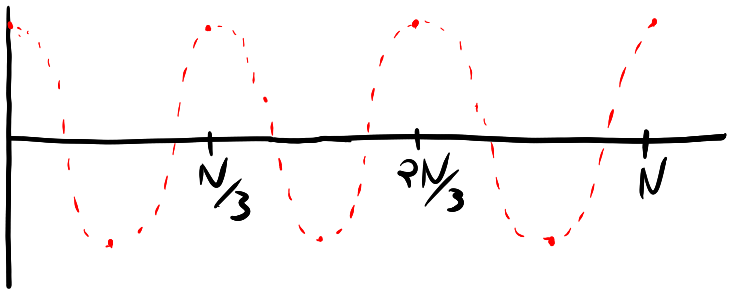
$\hat{f}(y)$: Fourier coefficient

Ex.

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

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

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



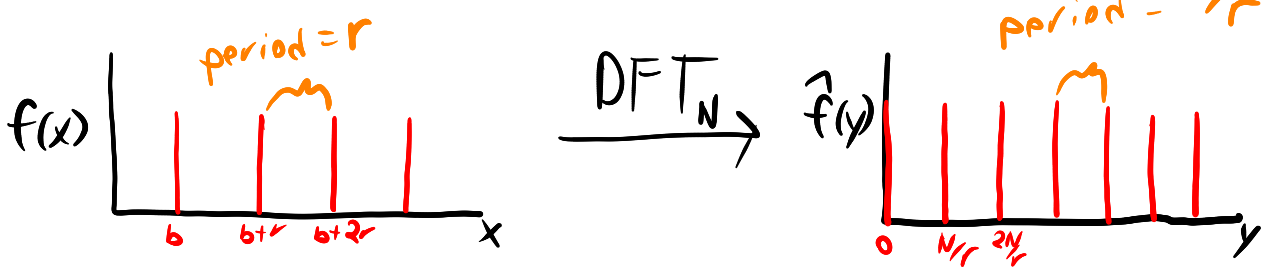
period = $N/3$

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

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

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 $N/3$

In general,



∴ Sampling the Fourier coefficients (almost) gives you the period!

(The Quantum Fourier Transform)

The **Quantum Fourier Transform** on n qubits is the unitary n-qubit transformation

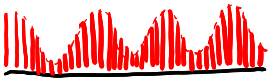
$$QFT_{2^n}: |x\rangle \longmapsto \frac{1}{\sqrt{2^n}} \sum_{y \in \mathbb{Z}_{2^n}} e^{2\pi i \frac{xy}{2^n}} |y\rangle$$

integer multiplication

Viewing a state vector $|f\rangle$ as a function $f: \mathbb{Z}_{2^n} \to \mathbb{C}$ where $f(x) = \langle x|f\rangle$

$$\begin{aligned} QFT_{2^n}(|f\rangle) &= \frac{1}{\sqrt{2^n}} \sum_y \left[\sum_x f(x) e^{2\pi i \frac{xy}{2^n}} \right] |y\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_y \hat{f}(y) |y\rangle \\ &= |\hat{f}\rangle \end{aligned}$$

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

$$|\ \text{  }\ \rangle \xrightarrow{QFT} |\ \text{  }\ \rangle$$

Note that the **inverse** QFT is

$$QFT_{2^n}^{+}: |y\rangle \longmapsto \frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{Z}_{2^n}} e^{-2\pi i \frac{xy}{2^n}} |x\rangle$$

Ex.

What does a Fourier matrix look like?

$$QFT_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$QFT_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

(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$|x\rangle$ state we have

$$\frac{1}{\sqrt{2^n}} \sum_y e^{2\pi i \frac{xy}{2^n}} |y\rangle$$

$$w_{16} = e^{2\pi i/16}$$

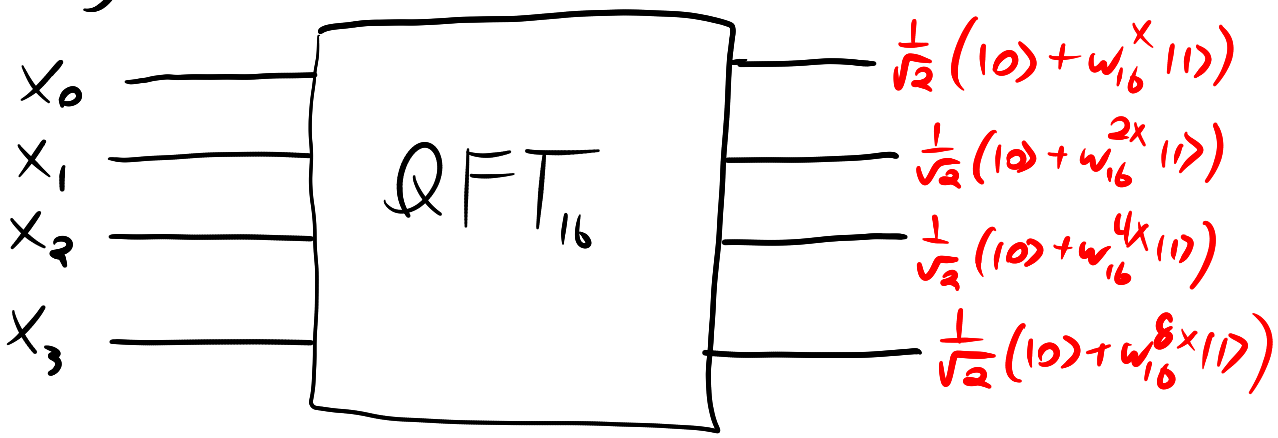
$$= \frac{1}{4} \sum_{y_0 \cdots y_3} w_{16}^{y_3(8x) + y_2(4x) + y_1(2x) + y_0 x} |y_0 \cdots y_3\rangle$$

$$= \frac{1}{4} \left(\sum_{y_0} w_{16}^{y_0 x} |y_0\rangle\right) \otimes \left(\sum_{y_1} w_{16}^{y_1(2x)} |y_1\rangle\right) \otimes \left(\sum_{y_2} w_{16}^{y_2(4x)} |y_2\rangle\right) \otimes \left(\sum_{y_3} w_{16}^{y_3(8x)} |y_3\rangle\right)$$

$$= \left(\frac{|0\rangle + w_{16}^{x}|1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle + w_{16}^{2x}|1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle + w_{16}^{4x}|1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle + w_{16}^{8x}|1\rangle}{\sqrt{2}}\right)$$

Note that this is a separable (or unentangled) state, so we can (kind of) proceed bit by bit.

Our goal is



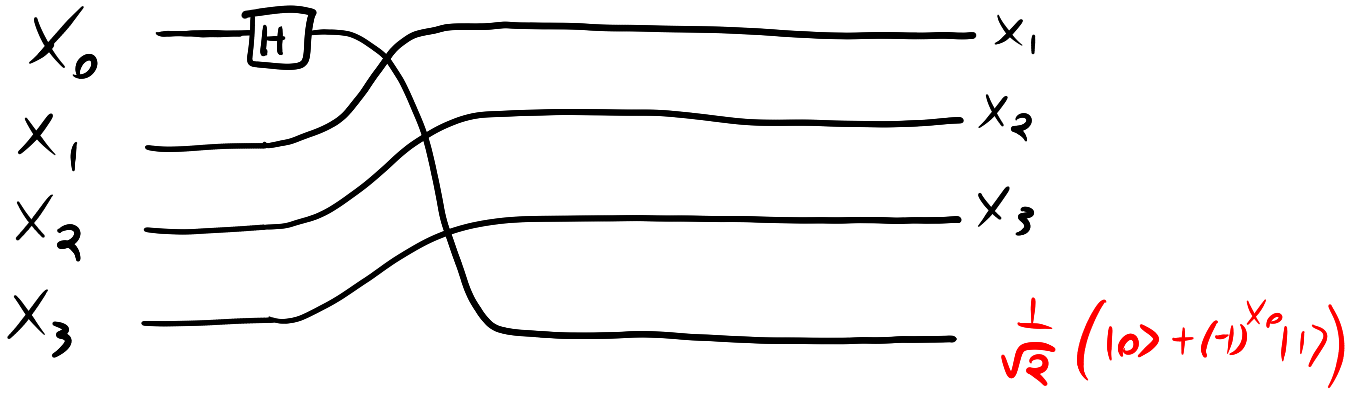
First observe that $w_{16}^{8x} = (-1)^x$ since $w_{16}^{8} = (-1)$.

Moreover, $x = 8x_3 + 4x_2 + 2x_1 + x_0$, so

$$(-1)^x = (-1)^{8x_3 + 4x_2 + 2x_1 + x_0} = (-1)^{x_0}$$

This tells us that the high-order bit y_3 of the QFT is just $\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{x_0}|1\rangle) = H|x_0\rangle$!

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



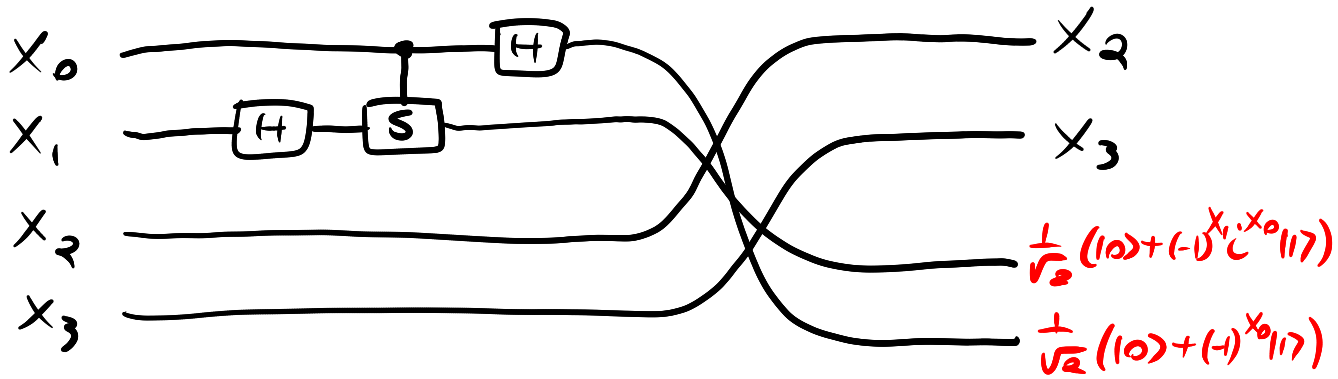
We can do the same with the next bit

$$w_{16}^{4x} = i^x = i^{8x_3 + 4x_2 + 2x_1 + x_0} = i^{2x_1 + x_0} = (-1)^{x_1} i^{x_0}$$

$$\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{x_1} i^{x_0}|1\rangle) = \begin{cases} H|x_1\rangle \text{ if } x_0 = 0 \\ SH|x_1\rangle \text{ if } x_1 = 1 \end{cases}$$

↖ recall that $S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$

Now, this is exactly an H gate followed by an S gate if $x_0 = 1$, i.e. a **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



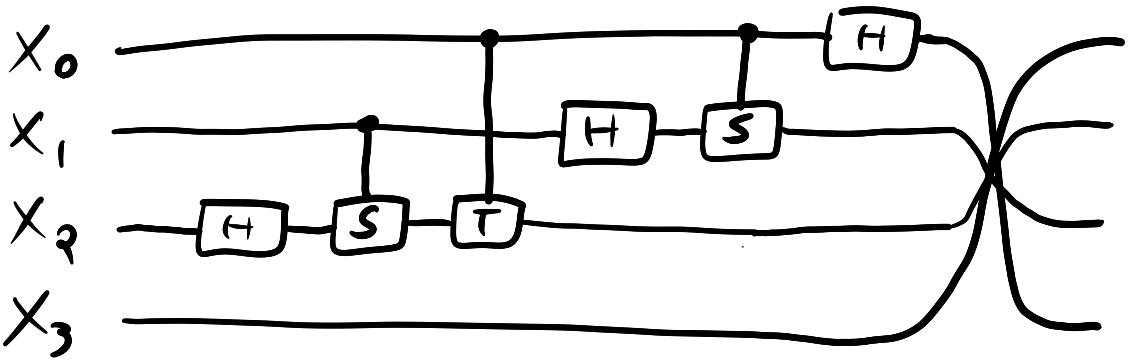
Repeating with the next bit,

$$w_{16}^{2x} = w_8^{8x_3 + 4x_2 + 2x_1 + x_0} = (-1)^{x_2} i^{x_1} w_8^{x_0}$$

Same story here — $H|x_2\rangle$ followed by phase rotatations of i and $w_8 = e^{2\pi i/8}$ conditional on x_1 and x_0, respectively. Recalling that

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/8} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & w_8 \end{bmatrix}$$

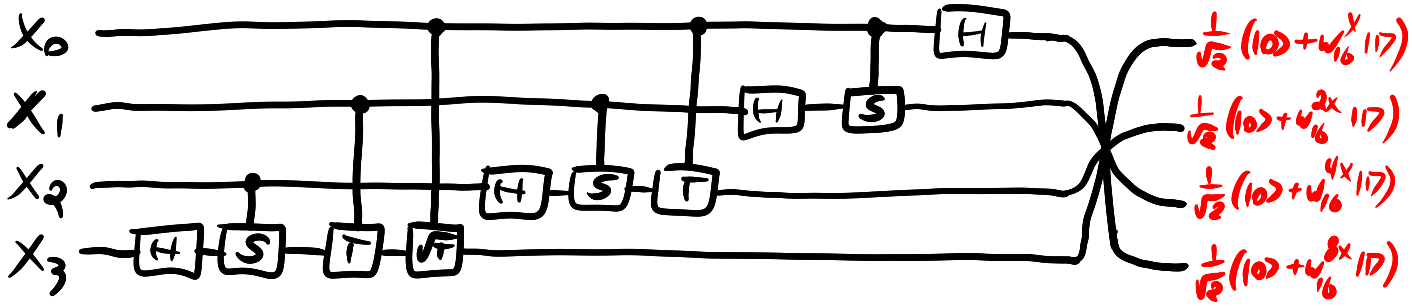
we have



Finally, $w_{16}^x = (-1)^{x_3} i^{x_2} w_8^{x_1} w_{16}^{x_0}$. Denoting

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

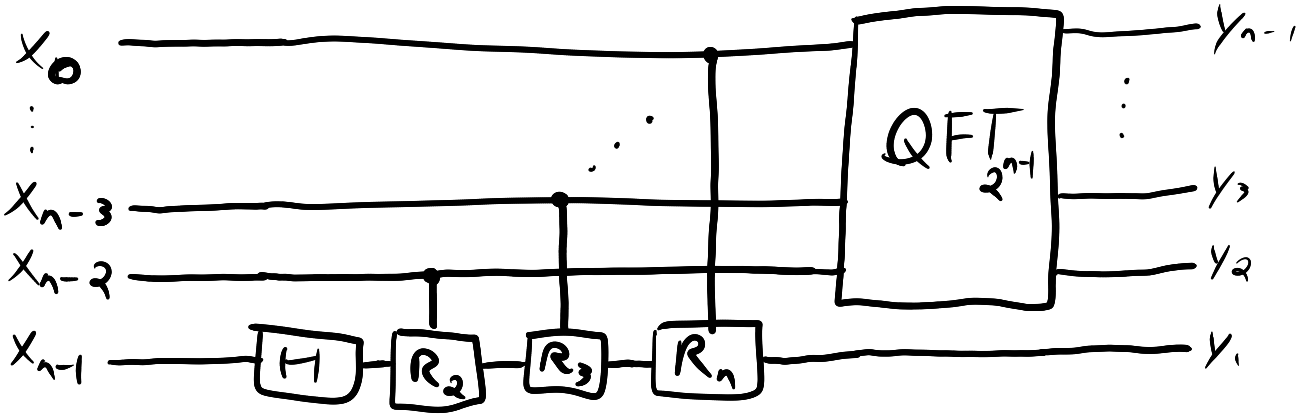
and following the pattern,



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

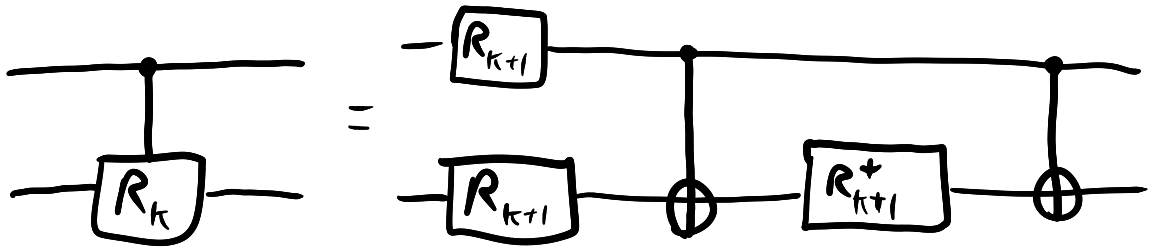
$$n + (n-1) + (n-2) + \cdots + 1 = \sum_{i=1}^{n} i \in O(n^2)$$

(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 $c\text{-}R_k$. 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 $c\text{-}R_k$ 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, <u>then</u> approximate. In this case



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 $c\text{-}R_k$ on $|x\rangle|y\rangle$ is

$$\begin{aligned} c\text{-}R_k|x\rangle|y\rangle &= \omega_{2^k}^{xy}|x\rangle|y\rangle \\ &= \omega_{2^{k+1}}^{2xy}|x\rangle|y\rangle \\ &= \omega_{2^{k+1}}^{x}\omega_{2^{k+1}}^{y}\omega_{2^{k+1}}^{-(x\oplus y)}|x\rangle|y\rangle \end{aligned}$$

Noting again that $R_{k+1}|x\rangle = w_{2^{k+1}}^{x}|x\rangle$, we just need to apply 3 phase rotations on the basis vectors $|x\rangle_1$, $|y\rangle_1$ and $|x \oplus y\rangle$ 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

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

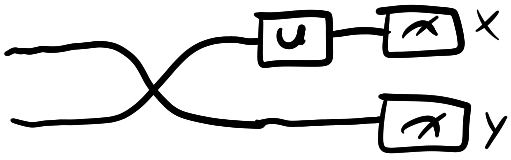
An alternative is to note instead that

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

and instead drop any R_k gates with $k \geq k_{th}$. 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 $\lfloor n/2 \rfloor$ 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



you may as well do