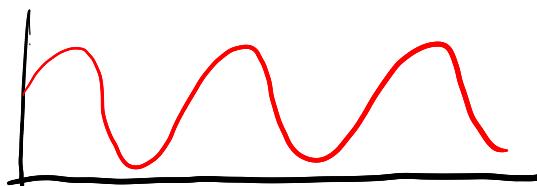


# CMPT 476 Lecture 20

## The Quantum Fourier transform

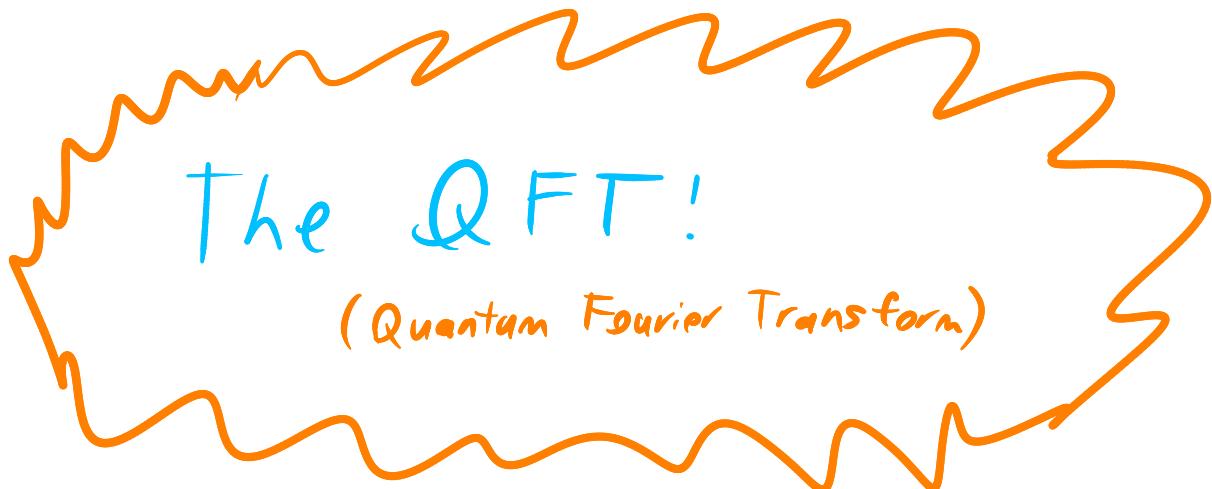
Most of quantum computing is signal processing.



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

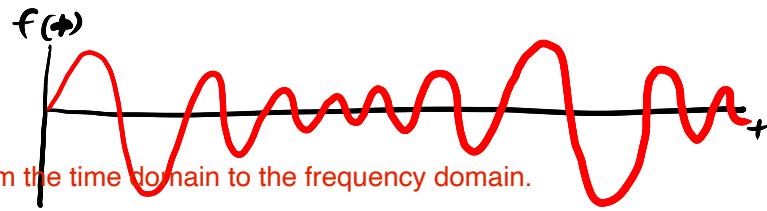
$$f(x) = a^x \bmod N$$

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



# (Fourier transforms & periodicity)

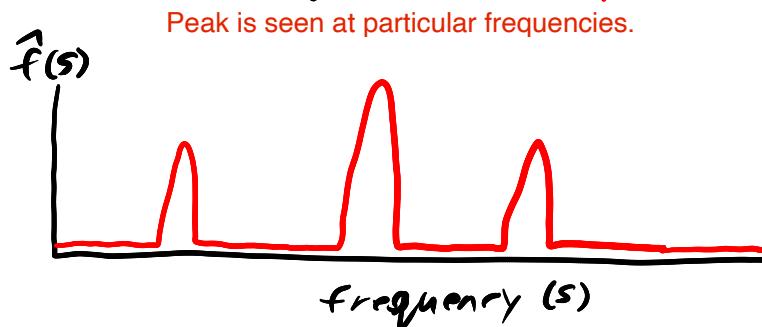
The (discrete) Fourier transform is a tool used in signal processing to translate a signal like this



Fourier transform moves you from the time domain to the frequency domain.

If you have a sum of three sound waves, then the Fourier transform will allow you to distinguish the three waves.

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...

How many samples do we need for the Fourier transform to give us accurate information?

## (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

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

Where does the  $\frac{xy}{N}$  come from?  
What does it mean?

This is a  $n$ th root of unity.  
What does this mean?

if  $N=8$ , then  $e^{\lambda} \{ \text{the rest of the expression} \} = e^{\lambda} \{-1/4\pi i xy\}$ , which means it would be an eighth root.

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

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

Since the DFT is a linear transformation, we should be able to apply it on a length  $2^n$  vector. With this, we should be able to implement it on a quantum computer.

We've already seen a Fourier transform:  $H$ !

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

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

The Fourier transform of  $\Phi$  is

$$\begin{aligned} \hat{\Phi}(y) &= \sum_{x \in \mathbb{Z}_2} \Phi(x) e^{-2\pi i \frac{xy}{2}} \\ &= \sum_{x \in \mathbb{Z}_2} \Phi(x) (-1)^{xy} \end{aligned}$$

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

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

(The inverse Fourier transform)

The inverse Fourier transform maps  $\hat{f} \rightarrow f$ , 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:

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

Nth roots of unity scaled by the fourier coefficient

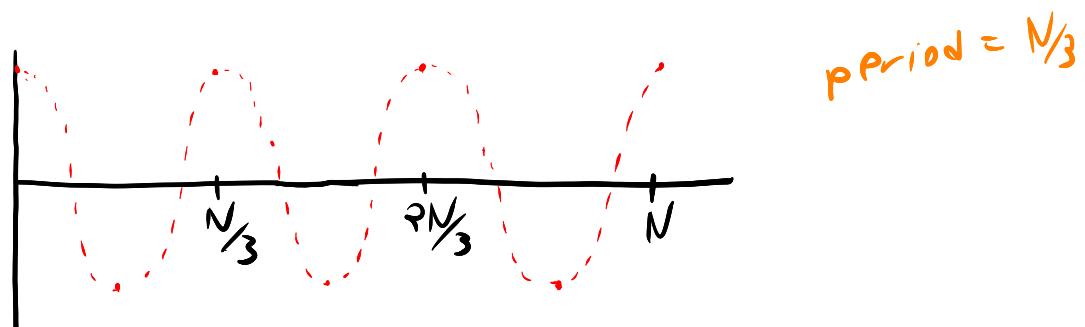
Fourier coefficient

Ex.

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

$$f(x) = e^{2\pi i \frac{3x}{N}} \quad \text{function which is periodic in } N/3$$

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

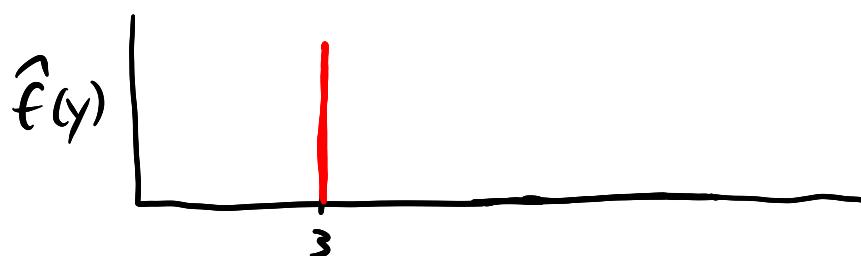


Taking the Fourier transform over  $\mathbb{Z}_N$  gives

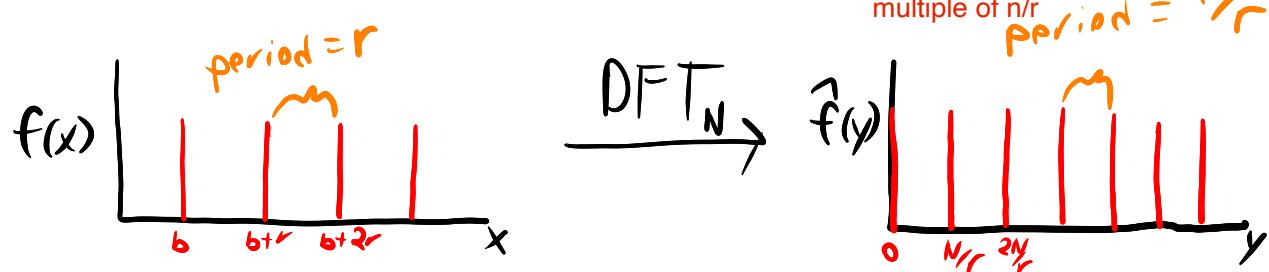
$$f(x) = e^{2\pi i \frac{3x}{N}} = \frac{1}{N} \sum_{y=0}^{N-1} \hat{f}(y) e^{2\pi i \frac{yx}{N}}$$

How do we know this intuitively?

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  $\frac{N}{3}$



In general,



$\therefore$  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

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

integer multiplication

Viewing a state vector  $|f\rangle$  as a function  $f: \mathbb{Z}_2^n \rightarrow \mathbb{C}$  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

$$|\text{red bars}\rangle \xrightarrow{\text{QFT}} |\text{black bars}\rangle$$

Note that the inverse QFT is

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

Ex.

What does a Fourier matrix look like?

$$QFT_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \rightarrow \begin{bmatrix} i^{1 \cdot 0} \\ i^{1 \cdot 1} \\ i^{1 \cdot 2} \\ i^{1 \cdot 3} \end{bmatrix}$$

$$QFT_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \rightarrow \begin{bmatrix} i^{2 \cdot 0} \\ i^{2 \cdot 1} \\ i^{2 \cdot 2} \\ i^{2 \cdot 3} \end{bmatrix}$$

## (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.

Shor showed that the quantum algorithm can be done in  $O((\log(n))^2)$

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

$$x \cdot y = y_3(8x) + y_2(4x) + y_1(2x) + y_0x$$

Applying this to the  $\text{QFT}|x\rangle$  state we have

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

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

$$= \frac{1}{4} \sum_{y_0 \dots y_3} w_{16}^{y_3(8x) + y_2(4x) + y_1(2x) + y_0x} |y_0 \dots y_3\rangle$$

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

$$\otimes \left( \sum_{y_3} w_{16}^{y_3(8x)} |y_3\rangle \right)$$

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

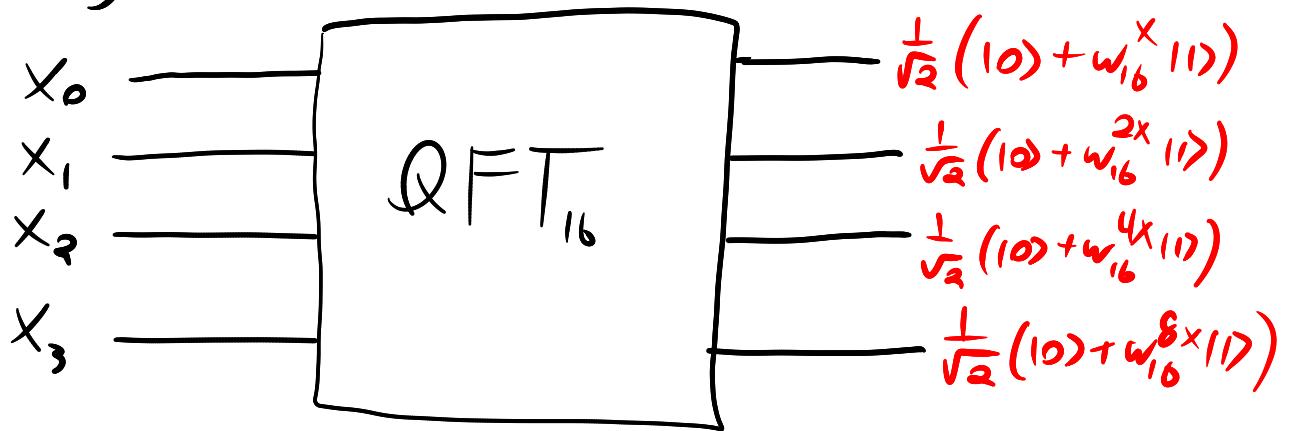
Note that this is a separable (or unentangled) state, so we can (kind of) proceed bit by bit.

Implementing game:

We need to remove one of the bits by using the controlled rotation gates.

If we have an eight root, then with an exponent of 8 or higher evaluates to 1.

Our goal is



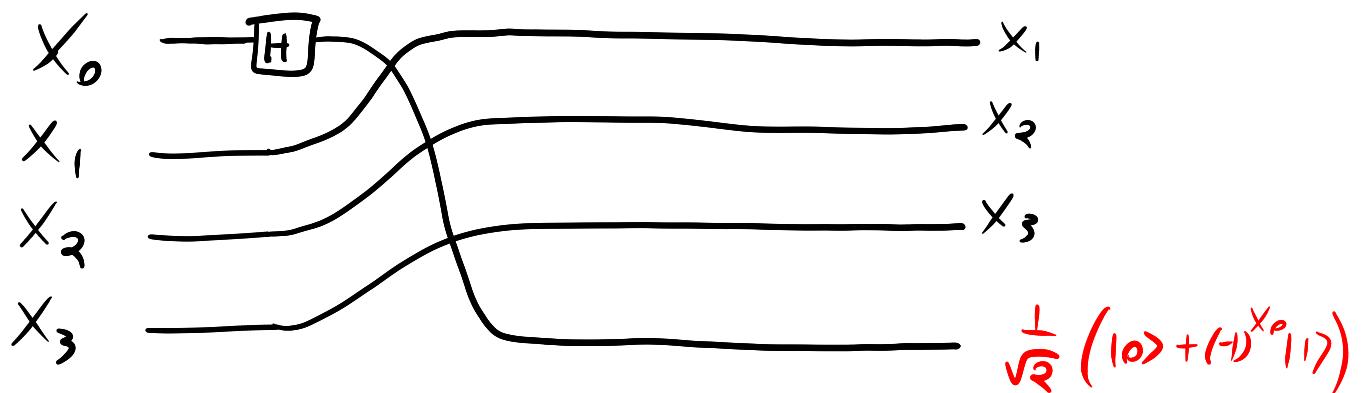
First observe that  $w_{16}^{8x} = (-1)^x$  since  $w_{16}^8 = (-1)$ .

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

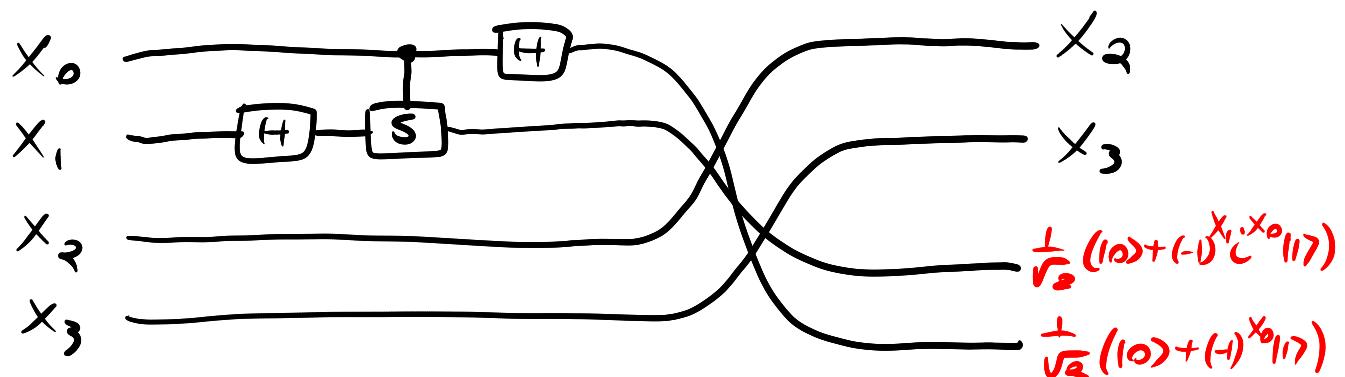
$$w_{16}^{4x} = i^x = i^{8x_3 + 4x_2 + 2x_1 + x_0} = i^{2x_1 + x_0} = (-1)^{x_1} i^{x_0}$$



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

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 **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



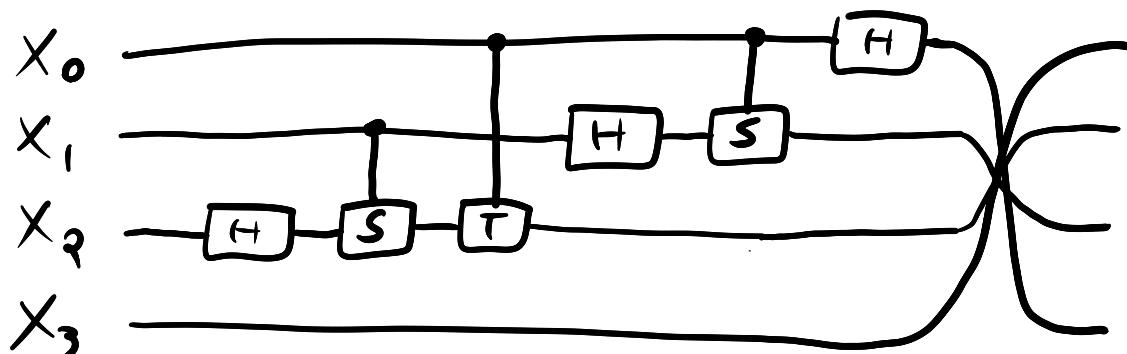
Repeating with the next bit,

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

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

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

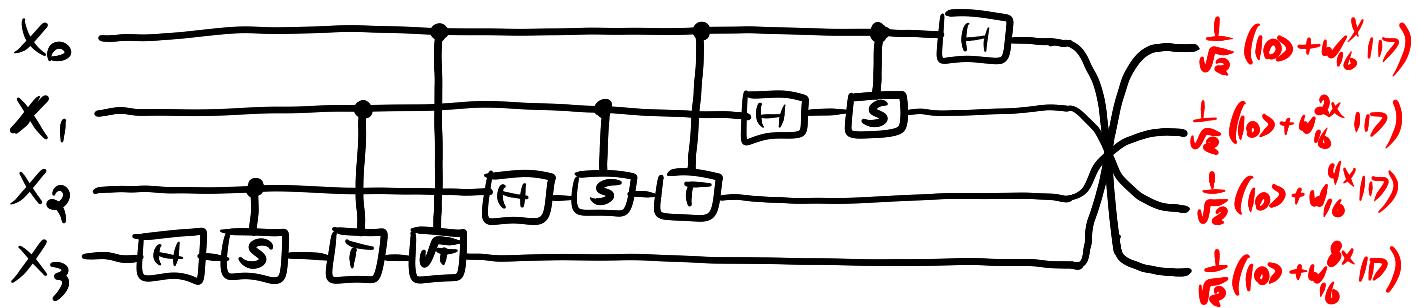
We have



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

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

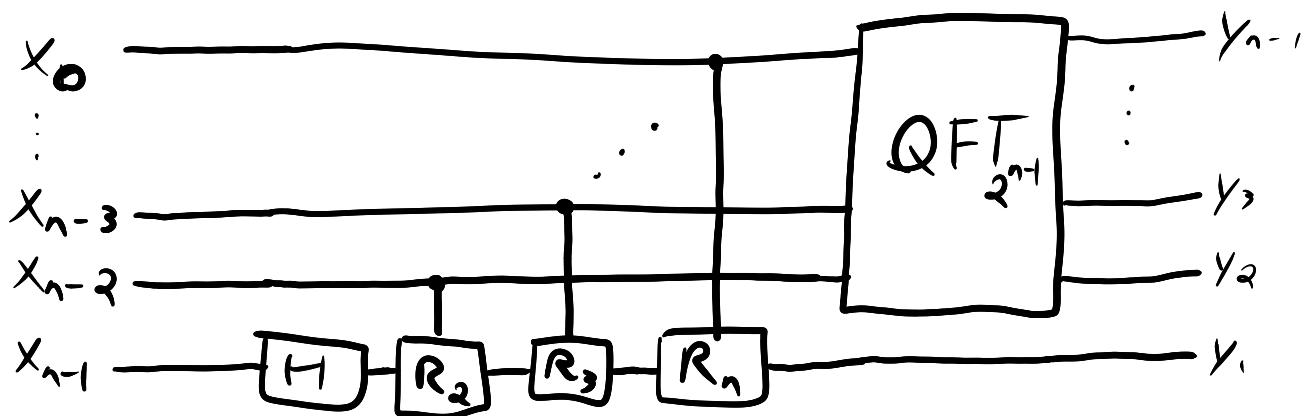
and following the pattern,



And that's  $\text{QFT}_{16}$

$(\text{QFT}_{2^n})$

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



(Complexity of the QFT)

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

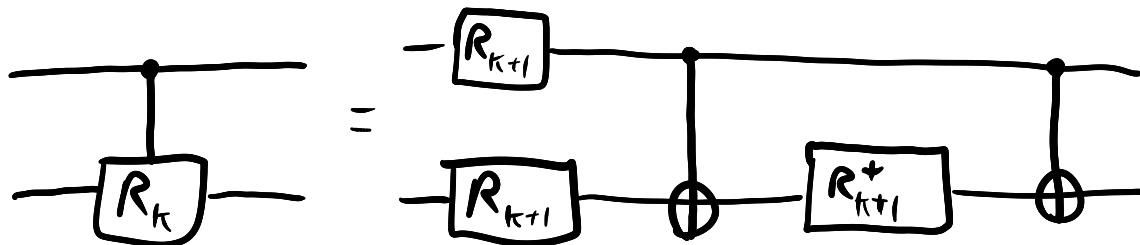
$$n + (n-1) + (n-2) + \dots + 1 = \sum_{i=1}^n i \in O(n^2)$$

## (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



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

$$2^{xy} = x + y - (x \oplus y)$$

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

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

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  $|x\rangle$ ,  $|y\rangle$ , and  $|xy\rangle$  which can be prepared with a CNOT gate and then uncomputed after the rotation.

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

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

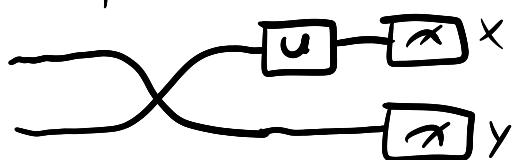
An alternative is to note instead that

$$R_k \approx I \text{ as } k \rightarrow \infty$$

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



Nth root of unity:  
complex number  $z$  such that  $z^N = 1$   
 $(W_n)^k = e^{\{2\pi i k/n\}}$  is also an nth root of unity  
If we take all the nth roots of unity and add them up, we get destructive interference and they all cancel out.

For third root of unity:  
the complex values of the second and third root cancel out.  
The real values of the first and (second and third) root cancel out.

As long as we take a root of unity which is not 1, then summing them up will have us obtain zero.  
Ask professor about this.