Sevag Gharibian

Lecture 9: The Quantum Fourier Transform

"... in mathematics you don't understand things. You just get used to them."
— John von Neumann.

It is hard to understate the impact of the *Fourier transform (FT)* on our everyday lives. Did you listen to an MP3 sound file this week? You used the Fourier Transform. Download JPEG photos of your friends off the internet? You used the Fourier Transform. Get an MRI recently? You guessed it — *Fourier Transform.* So what is the Fourier transform, and why is it so useful?

In a nutshell, the Fourier Transform allows us to efficiently decompose a signal (say, a sound wave) into its constituent frequencies. So, for example, that MP3 file which compresses your favorite song down to a few megabytes — it roughly works by applying the Fourier Transform to break down the song into its constituent frequencies, and then removing or *filtering out* the frequencies whose absence you are unlikely to notice (e.g. very high or low frequencies). Moreover, what makes this so applicable in practice is that the Fourier transform has a speedy near-linear time implementation. In particular, the Discrete Fourier transform (DFT) of a vector $|\psi\rangle \in \mathbb{C}^N$ can be implemented via the Fast Fourier Transform (FFT) algorithm using [1] $O(N \log N)$ field operations over $\mathbb{C}$. These two properties (signal decomposition and highly efficient implementation) are the one-two punch which makes the Fourier Transform so ubiquitous in everyday life.

It is thus perhaps fitting that, in the quantum setting, the *Quantum* Fourier Transform (QFT) underpins one of the greatest achievements of the field — Shor's quantum factoring algorithm. The primary focus of this lecture is hence to introduce the QFT, its implementation, and various applications. With these concepts under our belt, we will be ready to tackle the quantum factoring algorithm.

Now, we should be clear that broadly speaking, the topic of Fourier transforms is a bit confusing (this is roughly what the quote atop this lecture is alluding to). There are, in fact, *four* Fourier transforms (at least from the perspective of signal processing), depending on whether the input signal is continuous or discrete, finite/periodic or infinite/aperiodic. These go by the names of Fourier Series (continuous, finite), Discrete Fourier Transform (discrete, finite), Fourier Transform (continuous, infinite), and Discrete-Time Fourier Transform (discrete, infinite). (The interested reader is referred to [Kul02] for further details.) Here, when we talk about the FFT or QFT, we are referring to fast classical and quantum *implementations*, respectively, of the *DFT*. Thus, our "signals" are discrete and finite, encoded as vectors $|\psi\rangle \in \mathbb{C}^N$. The DFT, in turn, is a linear opeator mapping $\mathbb{C}^N$ to $\mathbb{C}^N$, representable as an $N \times N$ complex matrix. For this reason, we begin with a general class of matrices from which the DFT arises — the *Vandermonde* matrices.

1 From Vandermonde matrices to the Discrete Fourier Transform

Recall that given any complex number $c \in \mathbb{C}$, one can define *geometric sequence* $c^0, c^1, c^2, \ldots, c^{N-1}$. The strong structure of such sequences gives rise to nice properties — for example, the *sum* of the sequence entries, i.e. the *geometric series* $\sum_{k=0}^{N-1} c^k$, has a closed form. What happens if we embed such sequences as rows of a matrix? We might naively expect to get a structured matrix whose properties can, like the geometric series, be analyzed. Such matrices are called *Vandermonde* matrices.

Vandermonde matrices. As a naive first attempt for $N = 3$, fix any $c \in \mathbb{C}$, and consider Vandermonde matrix

$$M_1 = \begin{pmatrix} 1 & c & c^2 \\ 1 & c & c^2 \\ 1 & c & c^2 \end{pmatrix}.$$

[1] Note that the naive implementation of the DFT is $O(N^2)$, which is essentially unusable in practice.

This is not very interesting — since all rows are the same, M is just a rank 1 embedding of the original sequence $(1, c, c^2)$. But something interesting happens when we pick a *different* c for each row. For any distinct $c_1, c_2, c_3 \in \mathbb{C}$, the matrix

$$M_2 = \begin{pmatrix} 1 & c_1 & c_1^2 \\ 1 & c_2 & c_2^2 \\ 1 & c_3 & c_3^2 \end{pmatrix}.$$

turns out to have linearly independent rows. Thus, it is full rank (and hence invertible, assuming M_2 is square).

Exercise. Suppose above that $c_1 = c_2$, but $c_1 \neq c_3$. Is M_2 full rank? What if we add a fourth row with entries 1, c_4, c_4^2, for c_4 distinct from c_1, c_2, c_3?

To help give intuition as to the usefulness of Vandermonde matrices, one important application (which we revisit at the end of this section) is evaluating polynomials $p : \mathbb{C} \mapsto \mathbb{C}$ at the points (in the case of M_2) $c_1, c_2, c_3 \in \mathbb{C}$.

Exercise. Consider polynomial $p(x) = 4x^2 - 3x + 1$. Encoding the coefficients of p as vector $|\psi\rangle = (1, -3, 4)^T$, show that the ith coordinate of $M_2|\psi\rangle$ contains $p(c_i)$.

Exercise. Using the previous exercise, prove the Interpolation Theorm, which states: Any set of $d + 1$ point-value pairs $\{(c_i, p(c_i))\}_{i=1}^{d+1}$ identifies a unique degree-d polynomial p, assuming all c_i are distinct.

Returning to the DFT. Since we can choose *any* complex values for the $\{c_i\}$, let us choose the "quintessential" set of N complex numbers, the *Nth roots of unity*. These are generated by taking powers of the "principal" Nth root of unity, $\omega_N := e^{2\pi i/N}$, i.e.

$$\omega_N^0, \omega_N^1, \ldots, \omega_N^{N-1}.$$

For example, the principal 4th root of unity is $\omega_4 = e^{\pi/2} = i$, and the 4th roots are $(1, i, i^2 = -1, i^3 = -i)$.

Exercise. Draw the 4th roots of unity on the complex unit circle. More generally, what do the Nth roots of unity look like on the circle?

Exercise. Why are there only N Nth-roots of unity? More precisely, why is sequence $(\omega_N^0, \omega_N^1, \ldots, \omega_N^{N-1}, \omega_N^N)$ redundant?

Plugging the Nth roots of unity into a Vandermonde matrix, we finally arrive at the order-N DFT:

$$\mathrm{DFT}_N = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & (\omega_N^2)^2 & \cdots & (\omega_N^2)^{N-1} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & (\omega_N^{N-1})^2 & \cdots & (\omega_N^{N-1})^{N-1} \end{pmatrix}.$$

Exercise. Why is the first row of DFT_N all ones?

Exercise. What is DFT_2? DFT_3?

Since the roots of unity ω_N^k are distinct, the properties of Vandermonde matrices tell us DFT_N (which is square) is invertible. Can we write down its inverse? Remarkably, as shown in the exercise above, DFT_2 is just (up to scaling) the 2×2 Hadamard matrix we have grown quite fond of. And this is no coincidence — not only is DFT_N invertible, it is actually unitary. Thus, its inverse is simply $\mathrm{DFT}_N^\dagger$ (up to scaling).

Exercise. Prove that the rows of DFT_N are orthogonal. You may find it useful to look up the closed formula for geometric series, which also applies to complex numbers.

Exercise. Prove that a square matrix is unitary if and only if its rows (equivalently, columns) form an orthonormal set. Conclude that, up to scaling, DFT_N is unitary.

Exercise. What is the correct scaling required to make DFT_N unitary?

Since DFT_N is unitary, we can in principle apply it as a quantum gate to a quantum state! In other words, given $|\psi\rangle \in \mathbb{C}^N$, Nature allows the mapping $\mathrm{DFT}_N|\psi\rangle \in \mathbb{C}^N$. The only problem is that for an n-qubit system, the dimension $N = 2^n$ is exponential in n. So the question is: *Can we implement* DFT_N *via a quantum circuit of size polynomial in the number of qubits,* n*?* In the next section, we show the answer is *yes*. It is this quantum implementation of the (normalized) DFT which is henceforth denoted the *Quantum Fourier Transform (QFT)*.

Exercise. The fastest classical implementation of the DFT is the Fast Fourier Transform (FFT), which requires $O(N \log N)$ time. We shall show in Section 2 that the quantum implementation of the DFT, the QFT, requires only time $O(\mathrm{polylog}(N))$. Thus, on the face of it, it looks like one quantum computers exponentially speed up computation of the DFT. Why is this a misleading claim? (Hint: How does the output of the QFT differ from that of the FFT?))

Finally, back to polynomial evaluation. Before closing this section, let us return to our aside into polynomial evaluation.

Exercise. Using the FFT, argue that an arbitrary polynomial of degree $N-1$ can be evaluated at all Nth roots of unity in just $O(N \log N)$ time. What would be the naive runtime for this if we do not appeal to the FFT?

Exercise. Given two degree $N-1$ polynomials p and q, suppose we use the FFT (as in the previous exercise) to evaluate p and q at the Nth roots of unity, i.e. to obtain two lists of point-value pairs

$$\begin{aligned} &(1, p(1)), \quad (\omega_N, p(\omega_N)), \quad \ldots \quad , (\omega_N^{N-1}, p(\omega_N^{N-1})), \text{ and} \\ &(1, q(1)), \quad (\omega_N, q(\omega_N)), \quad \ldots \quad , (\omega_N^{N-1}, q(\omega_N^{N-1})). \end{aligned}$$

Using the fact that for any $(x, p(x))$ and $(x, q(x))$, the tuple $(x, p(x)q(x))$ denotes the value of the product of p and q on input x, show that the coefficients of polynomial $pq(x) := p(x)q(x)$ can be computed in time $O(N \log N)$. (Hint: You will also need the Interpolation Theorem.) How does this compare with the naive runtime for multiplying out two degree $N-1$ polynomials p and q (i.e. without using the FFT)?

Exercise. Use the FFT-based algorithm you derived in the previous exercise to compute the product of polynomials $p(x) = x^2 - x + 1$ and $q(x) = 2x^2 + 3x$. (Hint: Since $N = 3$ in this case, you can do all computations by hand.)

2 The Quantum Fourier Transform (QFT)

Suppose we have an n-qubit system. We now show how to implement the unitary operation $\frac{1}{\sqrt{N}}\mathrm{DFT}_N$ for $N = 2^n$ via $\mathrm{polylog}(N) = \mathrm{poly}(n)$ 1- and 2-qubit gates. To avoid having to repeat the normalization factor each time, we henceforth define $\mathrm{QFT}_N := \frac{1}{\sqrt{N}}\mathrm{DFT}_N$.

The N=4 case. We already know that $\mathrm{QFT}_2 = H$ (i.e. when $n = 1$), so what is QFT_4 ($n = 2$)? Recall that any linear map is completely specified by its action on a basis, such as the standard basis. Thus, since $\mathrm{QFT}_4|j\rangle$ extracts the jth column of QFT_4, we have:

$$\mathrm{QFT}_4|j\rangle \quad = \quad \frac{1}{2}\sum_{k=0}^{3} e^{2\pi i j\frac{k}{4}}|k\rangle, \tag{1}$$

where we have intentionally grouped $k/4$ together for the following reason. Recall from your programming courses that, typically, division of two integers a/b is not "cheap", but division by a power of 2, $a/2^k$, *is* cheap — it simply shifts over the bit representation of a to the right by k positions. For example, translating $3/2$ to binary means we shift the bit representation of 3, 11_2, to the right one position to obtain 1.1_2, which should be interpreted as $1 \cdot 2^1 + 1 \cdot 2^{-1}$.

Exercise. What is the binary representation of $14/29$? Expand it out in terms of powers of 2.

Rewriting $|k\rangle$ in binary as $|k_1k_2\rangle$, we thus have

$$\begin{aligned}
\mathrm{QFT}_4|j\rangle &= \frac{1}{2}\sum_{k=0}^{3} e^{2\pi i j(0.k_1k_2)}|k_1k_2\rangle && (2)\\
&= \frac{1}{2}\sum_{k=0}^{3} e^{2\pi i j\left(k_1\cdot 2^{-1}+k_2\cdot 2^{-2}\right)}|k_1k_2\rangle && (3)\\
&= \frac{1}{2}\left(\sum_{k_1=0}^{1} e^{2\pi i j k_1\cdot 2^{-1}}|k_1\rangle\right)\left(\sum_{k_2=0}^{1} e^{2\pi i j k_2\cdot 2^{-2}}|k_2\rangle\right) && (4)\\
&= \frac{1}{2}\left(|0\rangle + e^{2\pi i\frac{j}{2}}|1\rangle\right)\left(|0\rangle + e^{2\pi i\frac{j}{2^2}}|1\rangle\right) && (5)\\
&= \frac{1}{2}\left(|0\rangle + e^{2\pi i(0.j_2)}|1\rangle\right)\left(|0\rangle + e^{2\pi i(0.j_1j_2)}|1\rangle\right). && (6)
\end{aligned}$$

Exercise. In Equation (6), why can we omit the bit j_1 in the phase $e^{2\pi i(0.j_2)}$?

Now comes the key observation: Applying the Hadamard to single qubit standard basis state $|j\rangle \in \mathbb{C}^2$ yields

$$H|j\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i(0.j)}|1\rangle\right). \tag{7}$$

Exercise. Convince yourself that Equation (7) holds.

Thus, returning to attempting to implement DFT_4, we might naively try

$$H_1 \otimes H_2|j\rangle = H_1 \otimes H_2|j_1\rangle|j_2\rangle = \frac{1}{2}\left(|0\rangle + e^{2\pi i(0.j_1)}|1\rangle\right)\left(|0\rangle + e^{2\pi i(0.j_2)}|1\rangle\right). \tag{8}$$

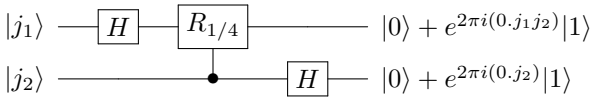
This is *almost* what we want (up to swapping the two output registers), except we are missing the bit j_2 in the phase $\exp(2\pi i(0.j_1 j_2))$ of Equation (6). Inserting a phase $\theta \in \mathbb{R}$, however, is easily accomplished (at least in theory) via gate

$$R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i\theta} \end{pmatrix}. \tag{9}$$

Exercise. Prove that R_θ is unitary for any $\theta \in \mathbb{R}$. What is its action on input $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$?

Exercise. Consider circuit $(R_{1/4} \otimes I)(H \otimes H)|j\rangle$. Why does this not reproduce $\mathrm{DFT}_4|j\rangle$ (up to swapping the output registers)?

The exercise above highlights that in order to insert phase $\exp(2\pi i(0.0j_2))$ into the *first* term (and hence first qubit) of Equation (8), the gate $R_{1/4}$ must also depend on the *second* qubit state, $|j_2\rangle$. We thus arrive at our QFT_4 circuit by adding a *controlled-*$R_{1/4}$ gate (below, we omit swapping the two output qubits for simplicity):



Exercise. Convince yourself that, up to swapping the output qubits, this is a correct QFT_4 circuit.

The general $N = 2^n$ case. The general $N = 2^n$ case now follows analogously to $N = 4$. Below, we state the action of QFT_N on a standard basis state, followed with the corresponding circuit. Proofs are left as an exercise. First, for standard basis state $|j\rangle \in \mathbb{C}^{2^n}$,

$$\begin{aligned}
\mathrm{QFT}_{2^n}|j\rangle &= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i j \frac{k}{2^n}} |k\rangle && (10)\\
&= \frac{1}{\sqrt{2^n}} \left(|0\rangle + e^{2\pi i(0.j_n)}|1\rangle\right) \left(|0\rangle + e^{2\pi i(0.j_{n-1}j_n)}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i(0.j_1 j_2 \cdots j_n)}|1\rangle\right) && (11)\\
&=: \frac{1}{\sqrt{2^n}} |\phi_n\rangle|\phi_{n-1,n}\rangle \cdots |\phi_{1,\ldots,n}\rangle, && (12)
\end{aligned}$$

where we have defined $|\phi_{k,\ldots,n}\rangle := |0\rangle + e^{2\pi i(0.j_k \cdots j_n)}|1\rangle$ for convenience.

Exercise. Prove Equation (11).

The QFT_N circuit is (up to reordering of output qubits):

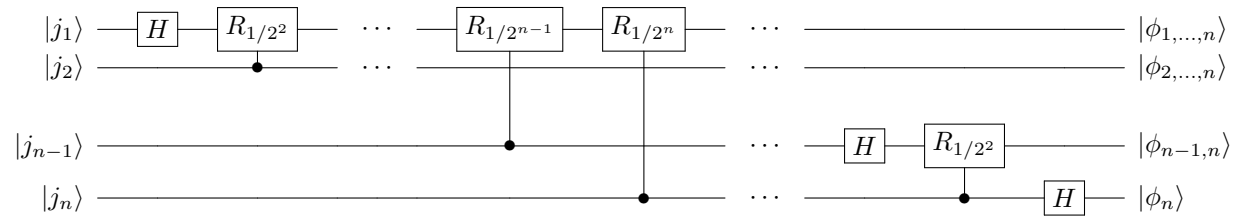
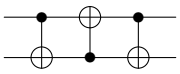


Figure 1: The QFT_N circuit (omitting final reordering of output qubits) acting on n qubits, where $N = 2^n$.

Exercise. Note the output qubits of the circuit above are in the opposite order than in Equation (12). Given the ability to perform the two-qubit SWAP gate, which maps any bipartite standard basis state $|i\rangle|j\rangle$ to $|j\rangle|i\rangle$, show how to rearrange the output of the circuit to match Equation (12).

Exercise. Prove that the following sequence of CNOT gates implements the two-qubit SWAP gate:



Do you need to explicitly prove it for any choice of product input state $|\psi\rangle|\phi\rangle \in \mathbb{C}^4$?

Exercise. How does the SWAP gate act on each of the four Bell basis states for $\mathbb{C}^4$?

Exercise. Show the circuit of Figure 2, together with your use of postprocessing via SWAP gates, correctly computes QFT_N. This quantum circuit *implementation* of DFT_N is what we call here the QFT_N.

Exercise. What is the size of the circuit in Figure 2, where "size" is defined by the number of 1- and 2-qubit gates? (Hint: Think about arithmetic series.) How much overhead does the final postprocessing via SWAP gates incur, and does this change the overall circuit size estimate for Figure 2? Give your estimates in terms of both N and n.

Let us close this section by again stressing that, up to renormalization, DFT_N and QFT_N perform precisely the same map. However, while the former is an abstract linear map, the latter is an explicit quantum circuit *implementation* of DFT_N which requires $O(\log^2 N) \in O(n^2)$ gates. In contrast, the best known classical implementation for the DFT_N is the FFT, which requires $O(N \log N)$ time. Thus, it *looks* like QFT_N gives an exponential speedup over the FFT. But this is entirely misleading, as the FFT and QFT_N represent their output in entirely different formats — the former gives an explicit list of all N entires of the output vector $|\psi\rangle$, whereas the latter gives $|\psi\rangle$ as a physical quantum state on $\log N$ qubits, whose amplitudes *cannot* in general be individually recovered.

3 Quantum Phase Estimation (QPE)

With QFT_N in hand, we are in principle in a position to discuss Shor's factoring algorithm. However, it will be instructive to take a top-down approach — to start by discussing, at a high level, the *class* of algorithms factoring falls into. Specifically, the heart of the factoring algorithm is a special case of the following general problem.

Definition 1 (Quantum Phase Estimation (QPE))**.**

- *Input: A quantum circuit implementing a unitary operator* $U \in \mathrm{L}\left((\mathbb{C}^2)^{\otimes n}\right)$*, and eigenvector* $|\psi_\theta\rangle$ *satisfying* $U|\psi_\theta\rangle = e^{2\pi i\theta}|\psi_\theta\rangle$*, for some* $\theta \in [0, 1)$*.*
- *Output: Phase* $\theta \in [0, 1)$*.*

Exercise. Prove that a normal operator U is unitary if and only if all of its eigenvalues have form $e^{i\theta}$.

Exercise. How would you solve QPE classically, i.e. given full written matrix and vector descriptions of U and $|\psi\rangle$?

Despite its arguably dull appearance, QPE is an important problem. Not only is it at the heart of the factoring algorithm, but QPE has applications throughout quantum algorithms, from quantum walks to

solving linear systems of equations. We will briefly comment on such applications in Section 3.1. Before then, however, take a moment to break down the statement of Definition 1 in your mind, and allow the inevitable myriad of questions to arise — *Can we solve QPE in general? What if θ has no finite representation? How do we magically get our hands on an eigenvector $|\psi_\theta\rangle$? And what if we cannot compute such an eigenvector efficiently?*

3.1 Applications of QPE

To motivate QPE, we now briefly mention three applications, and give a flavor of how QPE is applied.

1. **Factoring.** As we shall see in a subsequent lecture, factoring classically reduces to the *order-finding* problem: Given $x, N \in \mathbb{Z}^+$, what is the smallest $r \in \mathbb{Z}^+$ such that $x^r \equiv 1 \mod N$? For fixed x, we can define unitary U_x to have action $U_x|y\rangle = |xy \mod N\rangle$. Then, Shor's algorithm applies QPE to U_x, and classically postprocesses the result to find the order r.

2. **Quantum random walks.** A classical *random walk* is exactly what it sounds like — given a graph $G = (V, E)$ and a starting vertex $v \in V$, in each time step we pick a random neighbor of our current vertex and "walk" there. A general framework for implementing *quantum* walks uses QPE as follows. Given a vector $|\psi\rangle$, as a subroutine in the quantum walk, we would like to *reflect* about $|\psi\rangle$, i.e. to apply operator $R_\psi = 2|\psi\rangle\langle\psi| - I$.

   Exercise. Show that R_ψ is indeed a reflection about $|\psi\rangle$, i.e. that $R_\psi|\psi\rangle = |\psi\rangle$, and for any $|\psi^\perp\rangle$ orthogonal to $|\psi\rangle$, $R_\psi|\psi^\perp\rangle = -|\psi^\perp\rangle$.

   One approach for implementing R_ψ, roughly, is to set up a unitary map U for which $|\psi\rangle$ is an eigenvector. Then, to simulate application of R_ψ to an arbitrary state $|\phi\rangle$, one uses QPE to decompose $|\phi\rangle$ into its components: The component proportional to $|\psi\rangle$, and the component orthogonal to $|\psi\rangle$. With this "on-the-fly" decomposition in hand, one can "inject" a phase of $+1$ or -1 as needed, respectively.

3. **Powers of unitaries and linear systems.** Suppose we have access to a circuit implementation of unitary U, but want to instead apply $\sqrt{U}$. Can we do it using just U? QPE gives us an approach for this, based on the following exercise.

   Exercise. If the kth eigenvalue of U is $e^{i\theta_k}$, what is the kth eigenvalue of $\sqrt{U}$?

   Thus, we can use QPE to, roughly, extract the eigenvalue phases of U into a separate register (i.e. as strings $|\theta_k\rangle$), and coherently apply the square root function to "update" or "customize" the eigenvalue phases to $|\sqrt{\theta_k}\rangle$. Undoing the QPE estimation circuit then effectively "reinserts" the eigenvalues back into the operator to simulate $\sqrt{U}$.

   A striking application of this idea of "eigenvalue surgery" is to solve linear systems of equations $A|\psi\rangle = |b\rangle$ (i.e. given A, $|b\rangle$, what is $|\psi\rangle$?). The setup there is more complicated, as the coefficient matrix A need not even be Hermitian (never mind unitary). But the rough premise is similar — to simulate the inverse A^{-1}, one uses QPE to extract the eigenvalues λ_i of A (via some appropriate unitary implementation of A), inverts the λ_i "manually" to $1/\lambda_i$, and then uses *postselection* to "reinsert" the edited eigenvalues "back into A".

3.2 Quantum algorithms for QPE

We now give quantum algorithms for QPE. For simplicity, we assume the desired phase $\theta \in [0, 1)$ can be represented exactly using n bits, i.e. in binary we write $\theta = (0.\theta_1 \cdots \theta_n)_2$. As a result, $2^n\theta = (\theta_1 \cdots \theta_n)_2$. Our aim is, given U and $|\psi_\theta\rangle$, to prepare a quantum n-qubit register containing $|\theta_1 \cdots \theta_n\rangle = |2^n\theta\rangle$.

The algorithm. Recall that Equation (11) says

$$\mathrm{QFT}_{2^n}|\theta\rangle = \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{2\pi i k\frac{\theta}{2^n}}|k\rangle \tag{13}$$

$$= \frac{1}{\sqrt{2^n}}\left(|0\rangle + e^{2\pi i(0.\theta_n)}|1\rangle\right)\left(|0\rangle + e^{2\pi i(0.\theta_{n-1}\theta_n)}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i(0.\theta_1\theta_2\cdots\theta_n)}|1\rangle\right). \tag{14}$$

Thus, if we could prepare the right hand side above, applying $\mathrm{QFT}_N^\dagger$ would recover $|\theta\rangle$, as desired. There are two ways to do this, depending on how much we want to assume about what we can do with U.

More restrictive: Assume we can apply controlled-U^k gates for any integer $k \geq 0$. The following exercise suggests an approach for solving QPE in this setting.

Exercise. Show that for non-negative integer k, $U^k|\psi_\theta\rangle = e^{2\pi i k\theta}|\psi_\theta\rangle$.

But now we claim it is easy to prepare the right-hand side of Equation (13): Prepare an equal superposition over all $|k\rangle$, and then apply the controlled-U^k gate:

$$|0^n\rangle|\psi_\theta\rangle \xmapsto{H^{\otimes n}} \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1}|k\rangle|\psi_\theta\rangle \xmapsto{c-U^k} \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{2\pi i k\theta}|k\rangle|\psi_\theta\rangle = \frac{1}{\sqrt{2^n}}\sum_{k=0}^{2^n-1} e^{2\pi i k\frac{(2^n\theta)}{2^n}}|k\rangle|\psi_\theta\rangle \xmapsto{\mathrm{QFT}_N^\dagger} |2^n\theta\rangle|\psi_\theta\rangle,$$

where the first map applies local Hadamards to the first register, the second the controlled-U^k gates with the first register as control and the second register as target, and the last the inverse QFT_N to the first register.

Less restrictive: Assume we can apply controlled-U^{2^k} gates for any integer $k \geq 0$. Suppose we cannot raise U to *arbitrary* powers, but at least we can raise it to powers of 2. Can we still solve QPE? The answer is *yes* (albeit slightly more involved than the previous case), and is given in the following exercise.

Exercise. Show that by leveraging the expansion in Equation (14) instead of Equation (13), one can still solve QPE, even with the restricted ability of raising U to powers of 2. (Hint: Each choice of a power of 2 will allow you to prepare a specific term in the tensor product on the right hand side of Equation (14).)

Runtime and discussion. There are two major issues we have thus far swept under the rug: What if θ is not representable in n bits, and what is the size of the circuit implementing QPE?

Irrational $\theta \in [0, 1)$. Suppose now θ's binary representation is longer than n bits (potentially infinitely long). Can we at least approximate θ to its n most significant bits? *Yes*, and in fact, the exact same circuit as above works, with the following tweaks. Suppose we wish to compute the n most significant bits of θ's binary expansion with probability of success at least $1 - \epsilon$ for $\epsilon > 0$. Then, one can show that running the same procedure as before suffices, *so long as* we are willing to slightly expand the size of the first register to $n + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ qubits (all initialized to $|0\rangle$ before the circuit starts).

Size of circuit. It is easy to see that the two dominant components of the QPE algorithm are the QFT_N circuit and the controlled-U^k gate. We know the size of the first; but what about the size of the second? Unfortunately, here we hit a snag — to get n bits of precision in our estimate of θ, we need the gate $c-U^{2^n}$. But in general it will be impossible, given an arbitrary U, to compute U^k efficiently if k is superpolynomial in the input size.

Exercise. Suppose given an arbitrary unitary U on n qubits, we can compute U^{2^n} efficiently, i.e. with a quantum circuit of size polynomial in n. Prove that this implies quantum computers can efficiently count the number of solutions to a SAT formula $\phi : \{0,1\}^n \mapsto \{0,1\}$ (formally, the complexity class BQP would contain #P). Since it is believed unlikely that quantum computers can solve NP-hard problems (never mind #P problems), conclude it is unlikely for such a powering circuit for U to exist.

Thus, for arbitrary U, the best we can hope to do is estimate θ within $O(\text{polylog}(n))$ bits of precision (since then the largest power of U we need is polynomial in n).

Exercise. Suppose we approximate $\theta = 0.\theta_1\theta_2\ldots \in [0,1)$ by its first $\lceil \log n \rceil$ bits, i.e. $\widetilde{\theta} = 0.\theta_1 \ldots \theta_{\lceil \log n \rceil}$. What is the largest the additive error $\left|\theta - \widetilde{\theta}\right|$ can be?

The exercise above shows that logarithmically many bits of precision suffices to get inverse polynomial additive error. Note that this suffices in certain applications, since (roughly speaking) polynomial-size quantum circuits cannot distinguish states which are closer than an inverse polynomial in distance (trace distance for density operators, which reduces to Euclidean distance for pure state vectors) anyway.

Conclusion for QPE. We conclude our discussion on QPE by reiterating that, to estimate θ within n bits of precision, we require the controlled-U^{2^k} gate for $k \in O(n)$. The overall runtime of QPE will thus scale with the size of the QFT_N circuit and the powering-of-U circuit, the latter of which generally scales exponentially with k (the precise size is naturally context specific, i.e. depending on which U we have). Second, for the factoring algorithm, being able to raise U to exponentially large powers *will* be important — luckily, as we will see there, the specific U needed will implement a *classical* operation whose precise structure allows such fast exponential powering.

References

[Kul02] S. R. Kulkarni. Chapter 4: Frequency domain and Fourier transforms. `https://www.princeton.edu/~cuff/ele201/kulkarni_text/frequency.pdf`, 2002.