Lecture 8: Quantum Fourier transform and its applications

Rajat Mittal

IIT Kanpur

We will now look at some of the basic and fundamental algorithms in quantum computing. We have already seen Deutsch-Jozsa, Bernstein-Vazirani, and Simon's algorithm. They were the first few quantum algorithms which showed significant speed-up through quantum computation. These algorithms solved simple and slightly contrived problems, just to show that quantum computation can do better than the classical computation.

Let us take a hint from them. They are all on functions with domain $\mathbb{Z}_2^n$ and use Hadamard transform as an essential tool. Specifically, Simon's algorithm used Hadamard transform to find period in a function on $\mathbb{Z}_2^n$.

The straightforward idea would be to see if we can do Fourier transform (quantumly) on different groups. Indeed, we will learn about Fourier transform on Z_n, called *discrete Fourier transform*. Then, we will see quantum implementation of this Fourier transform, called *quantum Fourier transform (QFT)*. This will allows us to develop two very important applications of QFT, phase estimation and Shor's algorithm.

1 Fourier transform

Our first subroutine/algorithm will be a quantum version of the famous classical algorithm called *discrete Fourier transform*. To see the quantum version, we need to first learn: what is Fourier transform, and then see the classical algorithm for it.

You might have heard of *Fourier transform* as the conversion of a function from time domain to frequency domain. It has applications in analysis of differential equations, spectroscopy, quantum mechanics and signal processing.

Note 1. Even here, the "Fourier transform" of periodic functions is special, concentrated on specific frequencies. We will see that Fourier transform will allow us to find period of a periodic function in the cases mentioned below.

Instead, we will introduce Fourier transforms as a general tool to analyze functions over Abelian groups (mostly finite).

Note 2. Remember that an Abelian group is a commutative group, i.e., $\forall g_1, g_2 \in G : g_1 g_2 = g_2 g_1$.

Note 3. If you are not familiar with group theory, you can directly skip to the next subsection.

In general, Fourier transform acts on functions over these Abelian groups (e.g., real or complex numbers) and gives representation of those functions in the *character basis*. Readers are encouraged to look at the definitions of character, group theory and Fourier analysis over an Abelian group. The next few paragraphs will use facts from the basics of character theory and can be found in any standard text on group theory.

Characters:

Given a function $f : G \to \mathbb{C}$ on an *Abelian group* G, there is a standard representation of the function as a $\mathbb{C}^{|G|}$ vector (as a vector of complex numbers with length $|G|$).

Note 4. If you are confused about a general group, take G to be $\mathbb{Z}_n$, the group of remainders modulo n.

Exercise 1. What is this representation?

The representation is simply the value of the function on different elements of G, i.e., as a vector in $\mathbb{C}^G$.

Let us consider special functions of this kind. A *character* χ of G is a function $\chi : G \to \mathbb{C} - \{0\}$, s.t.,

$$\chi(g_1)\chi(g_2) = \chi(g_1 g_2) \quad \forall g_1, g_2 \in G$$

Note 5. We remove 0 from $\mathbb{C}$ to make it a group under multiplication.

Exercise 2. Can you think of a simple function which is a character?

The function $\chi_0(g) = 1$, for all $g \in G$, is known as the trivial character and is denoted by χ_0.

Exercise 3. Show that $\chi(e) = 1$, where e is the identity element?

For any element g in a finite group G, there exists an $n \leq |G|$ such that $g^n = e$, where e is the identity element. You can show (as an exercise) that $|\chi(g)| = 1$, for any g in G.

Exercise 4. What are the characters for $\mathbb{Z}_2$?

Properties of characters:

These properties of characters are well-known. You are encouraged to read any standard text on group representations for proofs (or try them yourselves). Below, χ will denote a character and G is a finite Abelian group.

Observe that a character χ can be viewed as a vector over complex numbers with length $|G|$.

- For a non-trivial character χ, $\sum_{g \in G} \chi(g) = 0$. If χ is trivial, then this sum is $|G|$.
- The product of two characters is a character.
- Any two distinct characters of G are orthogonal to each other (as vectors in $\mathbb{C}^{|G|}$).
- There are exactly $|G|$ many distinct characters for G.
- With appropriate normalization, the characters of G form an orthonormal basis of the space $\mathbb{C}^{|G|}$.

Note 6. It is known that the set of characters of G form a group under multiplication, called $\hat{G}$. We already saw that the size of $\hat{G}$ and G is same, not just that, they are actually isomorphic. In other words, we can index characters of G with elements of G (though this indexing is not unique).

Using these properties, we can define *the Fourier transform* over an Abelian group G. The Fourier transform is basically the *basis change operator* from the standard basis $e_1, e_2, \cdots, e_{|G|}$ to the normalized *character basis* $\chi_1, \chi_2, \cdots, \chi_{|G|}$. In other words, Fourier transform matrix is the unitary matrix which takes the standard basis $e_1, e_2, \cdots, e_{|G|}$ to the normalized *character basis* $\chi_1, \chi_2, \cdots, \chi_{|G|}$.

Any function f in $\mathbb{C}^{|G|}$ can be represented in Fourier basis as,

$$f = \sum_{i=1}^{|G|} \hat{f}(i)\chi_i.$$

The coefficients $\hat{f}(i)$ is called the i-th Fourier coefficient of f. Taking inner product with χ_i,

$$\hat{f}(i) = \frac{1}{n} \sum_g \chi_i * (g) f(g) := \langle \chi_i | f \rangle.$$

Note 7. We have defined $\langle f | \chi_i \rangle$ slightly differently, with a normalization. This makes χ_i's orthonormal.

Exercise 5. Why is the character basis different from any other orthonormal basis?

Hint: Look at the entry-wise multiplication of character basis vectors.

These Fourier coefficients capture properties of our function which might not be evident in the standard representation. For instance, let us look at the first Fourier coefficient, one corresponding to the trivial representation. By definition,

$$\hat{f}(1) = \langle \chi_1 | f \rangle = \frac{1}{|G|} \sum_x f(x). \tag{1}$$

In other words, first Fourier coefficient tells us about the summation of function value at all points.

Deutsch Jozsa algorithm as a Fourier transform: Remember Deutsch's problem, we want to find whether a function $f : \{0,1\}^n \to \{0,1\}$ is balanced or constant. Clearly, sum of all function values when f is constant is either 0 or 2^n, it is 2^{n-1} for the balanced case.

A much better separation condition can be obtained by looking at the function $(-1)^{f(x)}$ (this is similar to moving to range $\{-1,1\}$). If f is balanced then $\sum_x (-1)^{f(x)} = 0$; if f is constant then $\left|\sum_x (-1)^{f(x)}\right| = 2^n$.

So, to solve Deutsch's problem, we do Fourier transform of $(-1)^{f(x)}$ and look at the Fourier coefficient of the trivial representation.

Exercise 6. Which Abelian group should we take Fourier transform over?

Given the definition of the function, it is clear that we need Fourier transform over group $\mathbb{Z}_2^n$. To find it, we first find the Fourier transform over $\mathbb{Z}_2$.

Notice that there are two characters of the group $\mathbb{Z}_2$.

$$\chi_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ and } \chi_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

To get the Fourier transform over $\mathbb{Z}_2$, we put χ_i's in columns and multiply by an appropriate constant to normalize.

Exercise 7. What is the Fourier transform for $\mathbb{Z}_2$?

The change of basis matrix from standard basis to χ_1, χ_2 is our old friend, Hadamard matrix. So, the Fourier transform over $\mathbb{Z}_2^n$ is just $H^{\otimes n}$. A rigorous proof of the previous line requires some work, readers are encouraged to do it.

Exercise 8. Convince yourself that Deutsch-Jozsa algorithm is basically a Fourier transform over $\mathbb{Z}_2^n$.

1.1 Discrete Fourier transform

The *discrete Fourier transform* (DFT) is the Fourier transform over the group $\mathbb{Z}_n$. It is a linear transformation on functions of the type $f : \mathbb{Z}_n \to \mathbb{C}$.

Exercise 9. Can you find all characters of the group $\mathbb{Z}_n$? Hint: use the fact that $\mathbb{Z}_n$ is cyclic.

Remember, this function f can be represented by a vector $x \in \mathbb{C}^n$. The DFT is given by the DFT matrix F ($n \times n$ matrix),

$$F_{jk} = \omega^{jk}.$$

Here $\omega = e^{2\pi i/n}$ is the n^{th} root of unity and j, k range from 0 to $n-1$. So the matrix looks like,

$$F_n = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ 1 & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}$$

If the co-ordinates of x are $x_0, x_1, \cdots, x_{n-1}$ where $x_i = f(i)$. The DFT outputs n complex numbers $y_0, y_1, \cdots, y_{n-1}$, s.t.,

$$y_k = \frac{1}{\sqrt{n}} \sum_{0 \le j \le n-1} \omega^{jk} x_j.$$

Exercise 10. Show that F_n is a unitary matrix. What is the inverse of F_n?

1.2 Quantum discrete Fourier transform

The classical discrete Fourier transform takes a vector x to its image y. Suppose we consider x as a quantum state $|x\rangle$ (co-ordinates being the amplitudes), then the *quantum Fourier transform (QFT)* takes this state to $|y\rangle$ (similar to quantum Fourier transform over $\mathbb{Z}_2^n$).

We will show a circuit for quantum discrete Fourier transform using 1 qubit gates and controlled unitaries on 1 qubit. Since any 1 qubit or controlled unitaries on 1 qubit can be performed efficiently using any universal gate set (Solovay-Kitaev theorem [3]), this circuit is efficient in the usual sense.

To specify the action of the Fourier transform in terms of the basis $|0\rangle, |1\rangle, \cdots, |n-1\rangle$,

$$|j\rangle \to \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \omega^{jk} |k\rangle,$$

where $\omega = e^{2\pi i/n}$ is the n-th root of unity.

Exercise 11. Show that it is a valid quantum operation.

Exercise 12. Will implementing QFT imply that we can do DFT?

It turns out that QFT can be done in poly-log(n) operations, much better than even FFT. Though, in this case, we don't have direct access to the amplitudes of the state. So, having an algorithm for QFT does not imply that we have a fast algorithm for DFT. The output required in the two problems is different. They are definitely related and we will show some surprising results using QFT in later lectures (phase estimation, Shor's algorithm).

Exercise 13. Is the Fourier transform matrix Hermitian?

Again, we assume $n = 2^k$ for simplicity (for other n's, the algorithm is slightly different and uses phase estimation, refer [1]). So, the basis states are k bit strings $|j\rangle = |j_1 j_2 \cdots j_k\rangle$. QFT has a very useful *product* representation (the representation gives us the intuition to find the circuit for QFT).

Exercise 14. Before you see the following equations. Try applying F_4 on a 2-qubit basis state. Try to write the output as a tensor product of two qubits. How would you implement it? You can use the following elementary gates and H.

$$R_l = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^l} \end{pmatrix},$$

Given the exercise above, we can similarly factorize the output of QFT on a basis state.

$$
\begin{aligned}
F_{2^k}|j_1 j_2 \cdots j_k\rangle &= F_{2^k}|j\rangle \\
&= \frac{1}{2^{k/2}} \sum_{l=0}^{2^k-1} \omega^{jl}|l\rangle \\
&= \frac{1}{2^{k/2}} \sum_{l_1}\sum_{l_2}\cdots\sum_{l_k} \omega^{j(\sum_{h=1}^{k} l_h 2^{k-h})}|l_1 l_2 \cdots l_k\rangle \\
&= \frac{1}{2^{k/2}} \sum_{l_1}\sum_{l_2}\cdots\sum_{l_k} \bigotimes_{h=1}^{k} \omega^{j l_h 2^{k-h}}|l_h\rangle \\
&= \frac{1}{2^{k/2}} \bigotimes_{h=1}^{k} \left(\sum_{l_h} \omega^{j l_h 2^{k-h}}|l_h\rangle \right) \\
&= \frac{1}{2^{k/2}} \bigotimes_{h=1}^{k} \left(|0\rangle + \omega^{j 2^{k-h}}|1\rangle \right) \\
&= \frac{1}{2^{k/2}} \bigotimes_{h=1}^{k} \left(|0\rangle + (\omega^{2^k})^{j 2^{-h}}|1\rangle \right)
\end{aligned}
\tag{2}
$$

Notice that $\omega^{2^k} = 1$, hence we can say,

$$
|j_1 j_2 \cdots j_k\rangle \xrightarrow{F_{2^k}} \frac{1}{2^{k/2}}(|0\rangle + 1^{0.j_k}|1\rangle)(|0\rangle + 1^{0.j_{k-1}j_k}|1\rangle)\cdots(|0\rangle + 1^{0.j_1 j_2 \cdots j_k}|1\rangle).
$$

Note 8. $0.j_1 j_2 \cdots j_k$ means the expression $\sum_{l=1}^{k} j_l 2^{-l}$, as in the usual binary notation.

$$
|j_1 j_2 \cdots j_k\rangle \xrightarrow{F_{2^k}} \frac{1}{2^{k/2}}(|0\rangle + (-1)^{j_k}|1\rangle)(|0\rangle + (-1)^{j_{k-1}} i^{j_k}|1\rangle)\cdots(|0\rangle + (e^{2\pi i/2})^{j_1}(e^{2\pi i/4})^{j_2}\cdots(e^{2\pi i/2^k})^{j_k}|1\rangle).
$$

Here we used $1 = e^{2\pi i}$ to represent the roots of unity succinctly.

Exercise 15. Convince yourself that the above product representation is correct.

This product representation is the *main* step in implementing QFT.
Using this product representation and the elementary gates,

$$
R_l = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^l} \end{pmatrix},
$$

we can derive the circuit for quantum Fourier transform.

Exercise 16. What are the gates R_1, R_2, R_3?

Consider the first qubit of the transformed state, $(|0\rangle + e^{2\pi i 0.j_k}|1\rangle)$. If j_k is 0 then we get $|+\rangle$ state, else $|-\rangle$ state. That means, just apply a Hadamard to j_k to get this qubit.

Exercise 17. Show that a Hadamard on j_k will produce the desired state.

Similarly the second qubit, $(|0\rangle + e^{2\pi i 0.j_{k-1}j_k}|1\rangle)$, has a relative phase of $(-1)^{j_{k-1}}$ multiplied by i^{j_k}. To produce $(|0\rangle + e^{2\pi i 0.j_{k-1}j_k}|1\rangle)$, we can apply a Hadamard to j_{k-1} (to get a relative phase of $(-1)^{j_{k-1}}$), and then a controlled R_2 on j_{k-1} with control bit j_k (to get a relative phase of i^{j_k}). Continuing this process will give us the required circuit. The only thing is that the order of qubits are flipped. We can use the SWAP operation (swap the two qubits) to get the correct order.

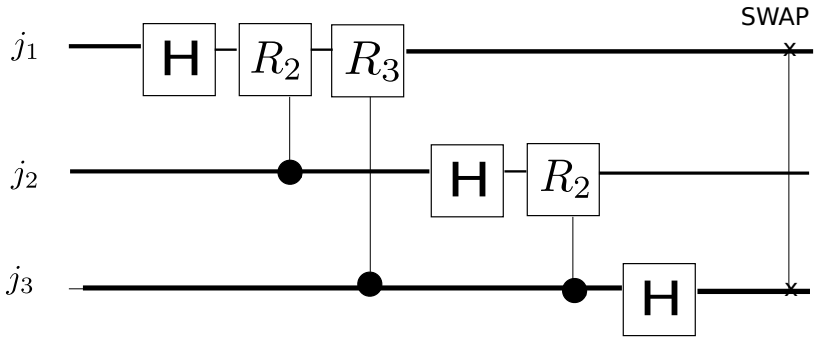


Fig. 1. QFT for $k = 3$

Exercise 18. Show that the circuit 1 gives QFT for $k = 3$.

Note 9. The operations on $(k-1)$-th qubit needs to be applied before the operations on k-th qubit.

It is easy to swap qubits using the CNOT operation. Have you seen it earlier? Try to construct a circuit for swapping qubits.

Exercise 19. Give the QFT circuit for general k.

Exercise 20. What is the number of operations in our implementation of QFT in terms of n?

Exercise 21. QFT can be used to find period of a function with domain $\mathbb{Z}_n$, this is called *period finding*. Read more about it, the algorithm is very similar to Simon's algorithm.

2 Phase estimation

Another very useful subroutine, in quantum computing, is known as *phase estimation*. Given a unitary U ($UU^* = I$), we know that all its eigenvalues have norm 1. Since any complex number can be written as $re^{2\pi i\theta}$, all eigenvalues of U should be of the form $e^{2\pi i\theta}$ for some θ. To determine the eigenvalue, it is enough to find this θ, called *the phase of the eigenvalue* or *eigenphase*.

We will start with a few basic assumptions so that the idea of phase estimation is clear; later we will see how to take care of them.

- Assume that $\theta = 0.j_1j_2\cdots j_k$ in the binary representation.

  Exercise 22. Why can we assume that $\theta \leq 1$?

- We are given the eigenvector as a quantum state $|u\rangle$.

The phase estimation subroutine, given a unitary U and its eigenvector $|u\rangle$, finds the phase of the eigenvalue corresponding to the eigenvector $|u\rangle$. To be precise, the algorithm will take the eigenvector $|u\rangle$ as input, and it needs the ability to perform controlled $U^{2^i}(i \leq k)$ operations; using those, it determines the corresponding eigenphase.

Exercise 23. What is the connection with QFT?

Assume that $\theta = 0.j_1 j_2 \cdots j_k$, otherwise this could be the appproximation of θ to k bits. Remember the action of QFT on a basis state $|j\rangle$,

$$|j_1 j_2 \cdots j_k\rangle \xrightarrow{F_{2^k}} \frac{1}{2^{k/2}}(|0\rangle + e^{2\pi i 0.j_k}|1\rangle)(|0\rangle + e^{2\pi i 0.j_{k-1}j_k}|1\rangle)\cdots(|0\rangle + e^{2\pi i 0.j_1 j_2 \cdots j_k}|1\rangle).$$

From the way we defined θ, the previous equation becomes

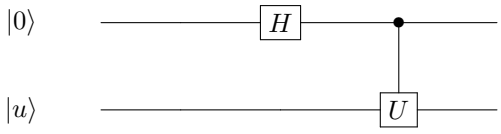
$$|2^k\theta\rangle \xrightarrow{F_{2^k}} \frac{1}{2^{k/2}}(|0\rangle + e^{2\pi i(2^{k-1}\theta)}|1\rangle)(|0\rangle + e^{2\pi i(2^{k-2}\theta)}|1\rangle)\cdots(|0\rangle + e^{2\pi i\theta}|1\rangle).$$

The main idea for phase estimation is: we can create the right hand side using the unitary and its eigenvector, then applying inverse of QFT will give us θ.

Exercise 24. Why does the inverse of QFT exist?

Exercise 25. How can you create the last qubit of right hand side, $\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\theta}|1\rangle)$.

We again use the fact that the phase on a state $e^{2\pi i\theta}|a\rangle|b\rangle$ can be absorbed in $|a\rangle$ or $|b\rangle$. We can create the phase of $e^{2\pi i\theta}$ using the eigenvector $|u\rangle$ and unitary U, and absorb in the associated qubit. The circuit would be,



How about the second last qubit? We need a phase of $e^{2\pi i(2\theta)}$.

Exercise 26. What unitary operator should we use? Draw the circuit like we did for last qubit.

2.1 Circuit for phase estimation

We will start with the state $|0, u\rangle$, where the first part of the register holds k qubits and second register holds the eigenvector $|u\rangle$. Then we will apply Hadamard on the first part and obtain,

$$\frac{1}{2^{k/2}}\sum_{l=1}^{2^k}|l, u\rangle.$$

Exercise 27. What other operation can we use instead of applying Hadamard?

To start with, we will also assume that we have the ability to perform U^l for all $l \leq 2^k = n$ (instead of just controlled U^{2^k}). This will give us another way to obtain the phase estimation circuit.

Later we will show that controlled $U^{2^i}(i \leq k)$ operators can be used to perform U^l for all $l \leq 2^k$. In the end we will get the same circuit as from the intuition given in the previous section.

Now we can perform the operation $|l, u\rangle \to |l\rangle U^l|u\rangle$. Notice that this can be done classically on the basis states and hence can be done quantumly.

This gives us the state,

$$\frac{1}{2^{k/2}}\sum_{l=1}^{2^k}|l\rangle U^l|u\rangle = \frac{1}{2^{k/2}}\sum_{l=1}^{2^k}e^{2\pi i\theta l}|l, u\rangle.$$

Exercise 28. What can be done to recover θ now?

Some thought shows that the first part of the register is the Fourier transform of $2^k\theta$. Hence applying inverse Fourier transform, we get the state $|2^k\theta\rangle$.

If we are only given the controlled versions of U^{2^l} where $l \leq k$, then how can we achieve the same phase estimation? Notice that l now varies only up to k. Essentially, we are given the power to apply $U, U^2, U^4 ..., U^{2^k}$.

The simple idea is to break any integer $0 \leq h \leq 2^k$ as powers of 2. Then using the controlled version, we can apply U^h. The following circuit has been taken from [3].

Exercise 29. Show that the circuit for phase estimation (Figure 2) works.

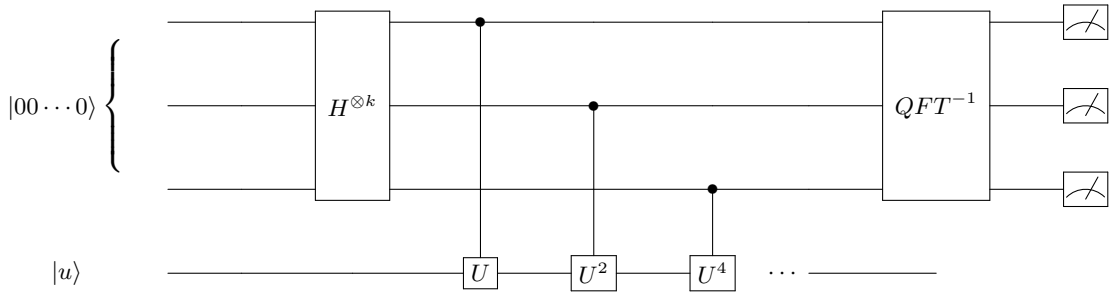


Fig. 2. Circuit for phase estimation

Let us see how to take care of the assumptions we made, there are only k bits in the expansion of θ and we have the eigenvector as a quantum state $|u\rangle$.

Most of the time, it is not possible to know the number of digits in the binary expansion of θ beforehand. What can be done in this case? If we want to approximate θ up to k bits of accuracy, using the same circuit with $k + f(\epsilon)$ qubits instead of k qubits will give us the answer with probability $1 - \epsilon$. Here, ϵ should be treated as a parameter and $f(\epsilon)$ is some function of ϵ. The details can be found in Nielsen and Chuang [3].

Suppose we don't have the eigenvector $|u\rangle$. If the same procedure is done over $|\psi\rangle = \sum_i \alpha_i |u_i\rangle$, we will get the phase corresponding to $|u_i\rangle$ with probability $|\alpha_i|^2$.

Exercise 30. Prove the above assertion.

Exercise 31. What will be the output of phase estimation on Z gate with $|u\rangle = |+\rangle$ state?

3 Factorization: Shor's algorithm

Shor's algorithm for integer factorization is one of the leading results in the field of quantum computing; it works by noticing that factorization can be reduced to order finding. We will see both steps, reduction to order finding and then the quantum algorithm for order finding.

We will introduce the two problems first.

Factorization: As one would guess, the problem is to find a non-trivial factor of a composite number n. Notice that the input size is $\log n$ and hence we are looking for algorithms which run in polylog(n) time. Since the number of factors are at most $\log n$, we can find all factors of n by applying this algorithm at most $\log n$ times.

Order finding over a group G: Given an element g in a group G, find the order of g in G (smallest r, s.t., $g^r = 1$). It turns out that order finding is a period finding problem over $\mathbb{Z}$, you will prove this in the assignment.

Group $\mathbb{Z}$ is not finite (unlike $\mathbb{Z}_n$), still we can use QFT. Remember that QFT can be used to find period of a function with domain $\mathbb{Z}_n$. We will use QFT to find period of a function with domain $\mathbb{Z}$. In section 3.2, the quantum algorithm for order finding will be described.

3.1 Extra reading: Reduction to order-finding

In this section, we will reduce the factorization of n to order-finding in the group $\mathbb{Z}_n^*$. The group $\mathbb{Z}_n^*$ denotes the set of all elements of $\mathbb{Z}_n$ coprime to n with multiplication as operation. In other words, we are simply asking, given n and a (coprime to n), what it the smallest r such that $a^r = 1 \mod n$.

Exercise 32. What is the order of 2 in $\mathbb{Z}_7^*$? What is its order in $\mathbb{Z}_{16}^*$?

Notice a subtlety here, factorization reduces to order finding over $\mathbb{Z}_n^*$ (a finite group). Though, order finding over a group (including $\mathbb{Z}_n^*$) is same as solving HSP over $\mathbb{Z}$ (an infinite group).

Our aim in this section is to reduce factorization to order finding over $\mathbb{Z}_n^*$. We will first get rid of the trivial cases. It can be easily checked if the number is even or if $n = m^k$ (take the square root, cubic root etc. up to $\log n$).

Exercise 33. How can we find square root (and other roots) efficiently?

This allows us to assume that the input to the factorization problem, n, is a number of type kk', where k and k' are co-prime and odd. We are interested in finding a non-trivial factor of n (not 1 or n). As mentioned earlier, we can repeat the procedure to find the complete factorization. The reduction shows that if we can solve order finding on $\mathbb{Z}_n^*$, then we can find a non-trivial factor of n (where n is of above type).

The basic idea of the reduction is to find a non-trivial square root b of 1, i.e.,

$$b: \quad b^2 = 1 \mod n, \; b \neq \pm 1 \mod n$$

Look at the possible square roots of $1 \mod n$, i.e., b for which $b^2 = 1 \mod n$. Clearly there are two trivial solutions, $b = \pm 1 \mod n$. Though, if there exists a square root $b \neq \pm 1 \mod n$, then $b^2 - 1$ is divisible by n and $b \pm 1$ is not. In this case, $gcd(b \pm 1, n)$ will give non-trivial factors of n.

The reduction from factorization to order-finding basically searches for such a non-trivial square root b. The algorithm takes a random $a < n$ and finds its order r. If r is even and $a^{r/2} \neq \pm 1 \mod n$, we have found a non-trivial square root.

Exercise 34. What if a is not in $\mathbb{Z}_n^*$?

Exercise 35. Look at Algorithm 1, and convince yourself that it will output a non-trivial factor.

We are only left to prove that this algorithm works with high probability. That is equivalent to showing that there are enough a's such that,

- order r of a is even,
- and $b = a^{r/2} \neq \pm 1 \mod n$.

Exercise 36. Can it happen that $a^{r/2} = 1 \mod n$?

Note 10. The quantum algorithm for factorization is a randomized algorithm; hence, it is enough to show: the number of *good* a's is a constant fraction of the number of total a's.

Following theorem proves that there are enough number of good a's. It follows from Chinese remaindering and standard number theory arguments. A casual reader can directly skip to the algorithm for order finding.

Check if n is even or of the form $n = m^k$;
Pick an a, s.t., $gcd(a, n) = 1$ (else we have already found a non-trivial factor of n) ;
for $i = 1, \cdots$ **do**
    Find the order of a and call it r (use the quantum algorithm for order-finding) ;
    if *r is odd or* $a^{r/2} = -1 \mod n$ **then**
        Pick another a co-prime to n ;
    else
        Found $b = a^{r/2} \neq \pm 1 \mod n$, square root of 1 ;
        Find the non-trivial factors from $gcd(b \pm 1, n)$;
        Break;
    end
end

Algorithm 1: Algorithm for factorization using order-finding

Theorem 1. *Suppose n is a product of two co-prime numbers $k, k' > 1$. For a randomly chosen a, the probability that a has an even order r and $a^{r/2} \neq -1 \mod n$ is at least $1/4$.*

Exercise 37. Convince yourself that Theorem 1 shows that our reduction is complete.

Before we prove the theorem, we need to know few properties of the numbers of the form $q = p^k$.

- $\mathbb{Z}_q^*$ is cyclic (Theorem 2 in Sec. 10).
- It is easy to calculate Euler function $\phi(q)$ (Read about Euler's totient function ϕ, if you don't know it).

  Exercise 38. Show, $\phi(q) = p^{k-1}(p-1)$.

- Lemma 1 proven below.

Let $m := \phi(q) = p^{k-1}(p-1)$ denote the size of group $\mathbb{Z}_q^*$. We introduce the notation, $p_2(z)$, the highest power of 2 that divides any number z. In other words, $z = 2^{p_2(z)}k$, where k is an odd number.

Lemma 1. *For a random element from $\mathbb{Z}_q^*$, its order r satisfies $p_2(r) = p_2(m)$ with probability exactly $1/2$.*

Proof. We know that $\mathbb{Z}_q^*$ is cyclic. Let g be a generator of $\mathbb{Z}_q^*$. In other words, a random element of $\mathbb{Z}_q^*$ is g^t where $1 \leq t \leq m$. The order of g^t is,

$$\text{ord}(g^t) = \frac{m}{gcd(m, t)}.$$

If t is odd, then $\gcd(m, t)$ is odd and $p_2(r) = p_2(m)$. Similarly, if t is even then $gcd(m, t)$ is even and $p_2(r) < p_2(m)$. Since half the t's are odd and half even, lemma follows.

□

We are ready to prove Theorem 1.

Proof of Theorem 1. Consider the prime factorization $n = p_1^{i_1} \cdots p_s^{i_s}$. By Chinese remainder theorem,

$$\mathbb{Z}_n^* \cong \mathbb{Z}_{p_1^{i_1}}^* \times \cdots \times \mathbb{Z}_{p_s^{i_s}}^*.$$

So, to randomly chose a, it is equivalent to pick random $a_1, \cdots, a_s$ from the respective $\mathbb{Z}_{p^i}^*$'s. Say, r_j are the orders of a_j modulo $p_j^{i_j}$. Then by Chinese remainder theorem, r is the LCM of r_j's. The following claim captures the *bad a's* in terms of $p_2(r_j)$.

Claim. Suppose the order r of a is odd or $a^{r/2} = -1 \mod n$. Then, $p_2(r_j)$ is same for all j.

Proof. The order is odd iff all r_j's are odd.

Otherwise, if $a^{r/2} = -1 \mod p_j^{i_j}$ then none of r_j divide $r/2$ (p_j's are odd). All r_j's divide r but not $r/2$, so $p_2(r_j)$ is the same (equal to $p_2(r)$). □

From lemma 1, with half the probability, the order r_j of a_j will be such that $p_2(r_j) = p_2(\phi(p_j)) =: l_j$. Call the case when $p_2(r_j) = l_j$ as the *first* case and other the *second* case (they happen with probability 1/2). We need to pick a_i's such that all $p_2(r_j)$'s are not same. The table below summarizes the situation.

n	$p_1^{i_1}$	$p_2^{i_2}$	$\cdots$	$p_s^{i_s}$
$p_2(m)$	l_1	l_2	$\cdots$	l_s
a	a_1	a_2	$\cdots$	a_s
r	$p_2(r_1) = l_1 \mid p_2(r_1) \neq l_1$	$p_2(r_2) = l_2 \mid p_2(r_2) \neq l_2$	$\cdots$	$p_2(r_s) = l_s \mid p_2(r_s) \neq l_s$

The last row denotes that for exactly half of the a_j's, $p_2(r_j) = l_j$.

Notice that l_j's only depend on n. If all l_j are equal, pick a_1's from first case and a_2's from the second case. If they are unequal, say $l_1 \neq l_2$, then pick a_1, a_2 from the first case.

In either scenario, r_j's can't be all equal, implying r is even and $a^{\frac{r}{2}} \neq 1 \mod n$ (by claim). Since we have only fixed at most 2 cases out of s, the probability is at least 1/4.

□

Hence the reduction from factorization to order finding is complete.

$\mathbb{Z}^*_{p^k}$ is cyclic The proof of the following theorem is given for completeness. Interested readers can take a look.

Theorem 2. *If $n = p^k$ for some power k of an odd prime p then $G = \mathbb{Z}^*_n$ is cyclic.*

Note 11. This is not true for even prime, $\mathbb{Z}^*_8$ is not cyclic.

Exercise 39. Find out where did we use the fact that p is odd.

Proof. Assume that $t = p^{k-1}(p-1)$, the order of the group G.

We know that $\mathbb{Z}^*_p$ is cyclic [2], and hence have a generator g. We will use g to come up with a generator of G. First notice that,

$$(g+p)^{p-1} = g^{p-1} + (p-1)g^{p-2}p \neq g^{p-1} \mod p^2.$$

So either $(g+p)^{p-1}$ or g^{p-1} is not $1 \mod p^2$. We can assume the latter case, otherwise replace g by $g+p$ in the argument below.

So $g^{p-1} = 1 + k_1 p$ where $p \nmid k_1$. So using binomial theorem,

$$g^{p(p-1)} = (1 + k_1 p)^p = 1 + k_2 p^2,$$

where $p \nmid k_2$.

Exercise 40. Continuing this process, show that,

$$g^{p^{e-1}(p-1)} = 1 + k_e p^e,$$

with $p \nmid k_e$.

From the previous exercise $g^t = 1 \mod p^k$ but $g^{t/p} \neq 1 \mod p^k$. The only possible order of g then is $p^{k-1}d$ where d is a divisor of $p-1$ (because the order has to divide t, Lagrange's theorem).

If the order is $p^{k-1}d$, then

$$g^{p^{k-1}d} = 1 \mod p^k = 1 \mod p.$$

But $g^p = g \mod p$ (why?). That implies $g^d = 1 \mod p$. Since $p-1$ is the order of g modulo p (g is the generator), implies $d = p-1$. Hence proved.

□

3.2 Order finding algorithm

Since $\mathbb{Z}$ is not finite, we will solve it by a seemingly different technique (this perspective of factoring algorithm is due to Kitaev). The explanation here is inspired from John Watrous's course notes [4].

You can read about the usual approach (HSP based) from multiple sources, e.g., [3] and [1]. In both the cases, you get lot of multiples of $1/r$ and deduce r from that information (classical postprocessing).

For factorization, we need to solve order finding over group $\mathbb{Z}_n^*$. That means, given an a co-prime to n, we need to find the smallest positive r for which $a^r = 1 \mod n$.

Note 12. Order finding is not a period finding problem over $\mathbb{Z}_n^*$ but on $\mathbb{Z}$.

Let us take a look at the algorithm for order finding using phase estimation. Suppose, we are interested in finding the order of a modulo n (a and n are given and they are coprime).

Exercise 41. How can we efficiently find if two numbers are coprime?

First approach:

Let k be the smallest number, such that, $2^k \geq n$. Consider the Hilbert space $\mathbb{C}^{2^k}$ spanned by $|b\rangle$, where b ranges from 0 to $2^k - 1$. Define the unitary operator,

$$U|b\rangle = |ab \mod n\rangle \text{ for } b \in \mathbb{Z}_n^*.$$

The unitary is not completely specified, but we are only interested in these basis states. The action on the other basis states can be assumed to be identity.

Exercise 42. Show that it is a unitary operator because a is coprime to n. How can we implement this unitary?

Since a has order r, it can be observed that $U^r = I$. From spectral decomposition, the eigenvalues of U have to be rth roots of unity. In other words, possible eigenvalues of U are $e^{2\pi i \frac{s}{r}}$, where s is an integer between 0 and $r-1$. The following exercise explicitly finds the eigenvalues and eigenvectors of U.

Exercise 43. Show that $|u_s\rangle$ is an eigenvector of U with the eigenvalue $e^{2\pi i \frac{s}{r}}$. Where,

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} e^{2\pi i j \frac{s}{r}} |a^j \mod n\rangle.$$

This analysis seems to suggest a very simple algorithm given s and the eigenvector $|u_s\rangle$. We can apply phase estimation on the unitary U with state $|u_s\rangle$. The output will be a good approximation of $\frac{s}{r}$, we can get r from this information. Notice that the powers U^{2^j} can be implemented using repeated squaring. Unfortunately, we don't have state $|u_s\rangle$ for any s.

Modified approach:

The idea would be to apply phase estimation on a superposition of eigenvectors. By linearity, we will get a particular eigenvalue with probability according to the amplitude of the corresponding eigenvector in superposition (we saw this in the phase estimation lecture). It turns out that the state $|1\rangle$ (which we can definitely prepare) can be written as a linear combination of these eigenvectors. In fact, it is an easy exercise to prove that,

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle.$$

Notice that even the amplitude on each eigenvector is same! We will apply phase estimation on $|1\rangle$ for unitary U; we will get a good approximations of $\frac{s}{r}$, with equal probability for all s. How can we obtain r from $\frac{s}{r}$?

We can repeat this phase estimation multiple times and obtain various approximations of s/r for different s (all s occur with equal probability). These actually correspond to different characters trivial on the subgroup $r\mathbb{Z}$. As in all hidden subgroup problems, we need to find our subgroup from these characters. The strategy for finite Abelian group will not work here.

Finishing the algorithm, classical part:

Our remaining task is to find r given multiple approximations to various $\frac{s}{r}$. The next insight is the following theorem about continued fractions (for reference, look at [3]). It shows that given a *good enough* approximation to an $\frac{s}{r}$, where s and r are coprime, there exist a continued fractions algorithm which can recover s and r from this approximation.

Theorem 3 ([3]). *Suppose $\frac{s}{r}$ is a rational number in the lowest form (no common factor between s and r), s.t.,*

$$\left|\frac{s}{r} - \phi\right| \leq \frac{1}{2r^2}.$$

If $r < n$, then s, r can be obtained in poly(L) time from ϕ, where $L = \lceil \log n \rceil$.

Note 13. The continued fractions algorithm is a very beautiful mathematical fact. Unfortunately, it will not be covered in this course due to lack of time. To read more about the continued fractions algorithm, please refer to [3] and references therein.

The consequence of continued fraction theorem is: if we get a *good enough* approximation of s/r (for an s coprime to r), we can find r. Let us take care of these two issues.

Exercise 44. It will be helpful if you remember the details of phase estimation. Please take a look at the notes for phase estimation.

- *Precision*: Since $r \leq n$, it is sufficient to get an approximation which is $\frac{1}{2n^2}$ close. That means, we need the first $2L+1$ bits of phase to be correct (remember $L = \lceil \log n \rceil$). So, we need to run phase estimation on $2L + 1 + f(\epsilon)$ qubits and it will give us phase with probability $1 - \epsilon$ (this follows from our discussion on phase estimation). We can extend U to $\mathbb{C}^{2^{2L+1}}$ by trivial action on other basis states.

  Exercise 45. Do you remember the definition of U? What was its action on basis states?

- *Coprime s*: Can you think of a way to find an s coprime to r? It will again be our usual trick, find lots of s, one of them will be coprime to r. Prime number theorem states that the number of primes less than n are around $O(\frac{n}{\log n})$.

  Exercise 46. Show that the number of s, co-prime to r, are at least $O(\frac{1}{L})$ fraction of total s using prime number theorem.

  If we have $O(L)$ different s (picked randomly), with high probability one of them will be coprime to r. Continued fraction algorithm on that s/r will give us the order r.

Exercise 47. Can you summarize the complete algorithm for factorization now? Try to draw a circuit for it.

Hence the order finding algorithm can be summed up as: repeat the phase estimation on U at least $O(L)$ times. The precision for phase estimation would be $k = 2L + 1 + f(\epsilon)$. The circuit for order finding is given in Fig. 3.

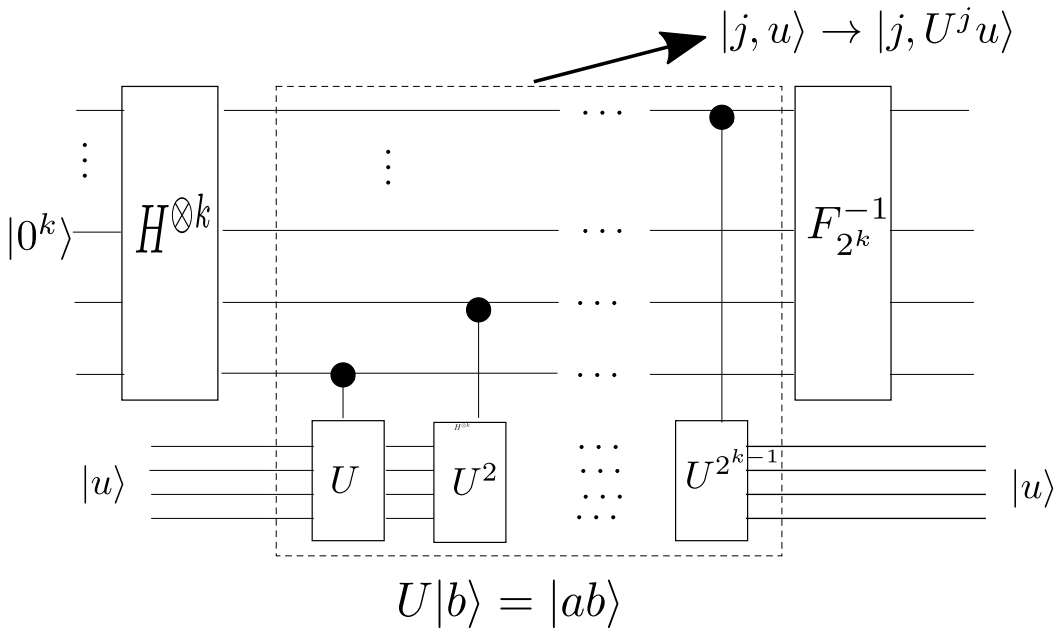


Fig. 3. Order finding algorithm

4 Assignment

Exercise 48. Show that the action of Hadamard is,

$$H^{\otimes k}|i\rangle = \frac{1}{\sqrt{2^k}} \sum_j (-1)^{i.j} |j\rangle,$$

Exercise 49. Read about characters and groups.

Exercise 50. For any element g in a finite group G, there exists an n such that $g^n = e$, where e is the identity element. Show that $|\chi(g)| = 1$ for any g in G.

Exercise 51. What are the characters of $\mathbb{Z}_n$.

Exercise 52. Give a circuit to perform inverse Fourier transform.

Exercise 53. Read about the approximate phase estimation from [3].

Exercise 54. Is there a difference between $H^{\otimes k}$ and QFT on k qubits (2^k dimensional vector)?

Exercise 55. Read about the complexity class BQP.

Exercise 56. **Lemma 2.** *Given a positive number α, let there exists a sequence of rationals $\frac{p_n}{q_n}$, such that, $|q_n\alpha - p_n| \neq 0$ tends to zero. Then, α is irrational.*

Prove the lemma by showing that if $\alpha = a/b$ then $|q_n\alpha - p_n| \geq 1/b$. Using the lemma, show that e is irrational.

Exercise 57. We had claimed that order finding is a period finding problem.

- Show that $g^j \neq g^k$ if $0 \leq j < k < r$. Where r is the order of g.
- Show that order finding is a period finding problem in $\mathbb{Z}$.

Exercise 58. Using Chinese remainder theorem, show that there exists a b, such that, $b^2 = 1 \mod n$ and $b \neq \pm 1 \mod n$. Here n contains at least two distinct primes in its factorization.

Exercise 59. Why is the phase estimation algorithm for order finding, described above, is same as the HSP algorithm.

Exercise 60. This exercise is kind of a sanity check on continued fractions algorithm. Show that there is no other rational number $\frac{s'}{r'}$, where $r' < r$ and

$$\left| \frac{s'}{r'} - \phi \right| \leq \frac{1}{2r^2}.$$

References

1. A. Childs. Quantum algorithms, 2013. `https://www.cs.umd.edu/~amchilds/qa/qa.pdf`.
2. R. Lidl and H. Niederreiter. *Finite Fields*. Cambridge University Press, 1997.
3. M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge, 2010.
4. J. Watrous. Order finding, 2006. `https://cs.uwaterloo.ca/~watrous/QC-notes/QC-notes.10.pdf`.