# Implementing the Quantum Fourier Transform

Raul-Martin Rebane

May 7, 2020

*Note: This lab is not as difficult as it may appear on just scrolling through and looking at the formulas. The bit arithmetic in exponents just makes everything look much more complicated than it really is.*

## 1 Preliminaries

### 1.1 Bit arithmetic and swapping

We're going to be doing some bit arithmetic in this lab, so it's good to get a small refresher on doing some bit arithmetic. In base 2 a number $y = 10110_2$ is $y_1 \cdot 2^4 + y_2 \cdot 2^3 + y_3 \cdot 2^2 + y_4 \cdot 2^1 + y_5 \cdot 2^0$ where $y_n$ represents the $n$th bit of $y$ from the left. Note that in general for an $n$ bit string, the largest exponent will be $2^{n-1}$ and the smallest will be $2^0$.

We can split the bitstrings apart the same way we do with numbers. For instance I can say $1234 = 12 \cdot 100 + 34$, and similarly I can do with bits. $y = y_1y_2y_3y_4y_5 = y_1y_2 \cdot 2^4 + y_3y_4y_5$.

We can shift a bitstring to the left by multiplying it with 2, the same way that multiplying a number by 10 shifts all digits to the left and appends a 0. $1234 \cdot 10 = 12340$, $10110_2 \cdot 2 = 101100_2$. We'll also need a swapping operation $\overline{y} = 01101$ which just reverses the order of all the bits.

In our proof we'll need to express the following product. You can refer back to this section later to understand why we're doing this, but currently we want to show that we can express a product as a certain sum. Namely we're interested in expressing the value

$$\overline{bx} \cdot cy$$

Where $b$ and $c$ are 1 bit length, and $x, y$ are $n-1$ bit length. Note that $cy$ does **not** in this case denote multiplication, but the concatenation of a bitstring. In this case $c$ is the first bit of the bitstring $cy$. All multiplications involving a bitstring

are denoted with a $\cdot$ operator. If you see two bitstrings next to each other then this is a concatenation.

In $\overline{bx}$ the $b$ goes to the end because of the swap, and so we have

$$\overline{bx} \cdot cy = \overline{x}b \cdot cy$$

Now we can split the bitstring as we did above, and then use distributivity. This is no different than expressing $12 \cdot 34 = (10+2) \cdot (30+4) = 10 \cdot 30 + 10 \cdot 4 + 2 \cdot 30 + 2 \cdot 4$. It is just unusual to have to multiply like this.

$$\overline{bx} \cdot cy = \overline{x}b \cdot cy = (\overline{x} \cdot 2 + b) \cdot (c \cdot 2^{n-1} + y) = 2^n \cdot \overline{x} \cdot c + 2 \cdot \overline{x} \cdot y + 2^{n-1} \cdot b \cdot c + b \cdot y$$

In our proof we will be interested in reaching the product on the left, and we will do so by obtaining the sum on the right.

## 1.2 Roots of unity

The roots of unity that we're dealing with are going to be the $N = 2^n$ roots of unity, which gives us some nice properties. Remember that the $n$th root of unity was a complex number $z$ such that $z^n = 1$. A $n$-th primitive root of unity is one that is not a $k$-th root of unity, for any positive $k < n$. So we can't say that our $2^8$th primitive root of unity is 1, for instance, as it is also the 4th root of unity for example.

We find the primitive roots of unity by dividing the unit circle up into $n$ sections, as multiplying elements on the unit circle only adds the angle between them. So if we have $z$ that is $\frac{1}{n}$ of a full turn of the unit section, then $z^n$ does the full turn and brings us back to 1. The general formula for expressing this is $e^{\frac{2\pi i}{n}}$. Since we're only dealing with powers of 2, we define a shorthand $\omega_n = e^{\frac{2\pi i}{2^n}}$.

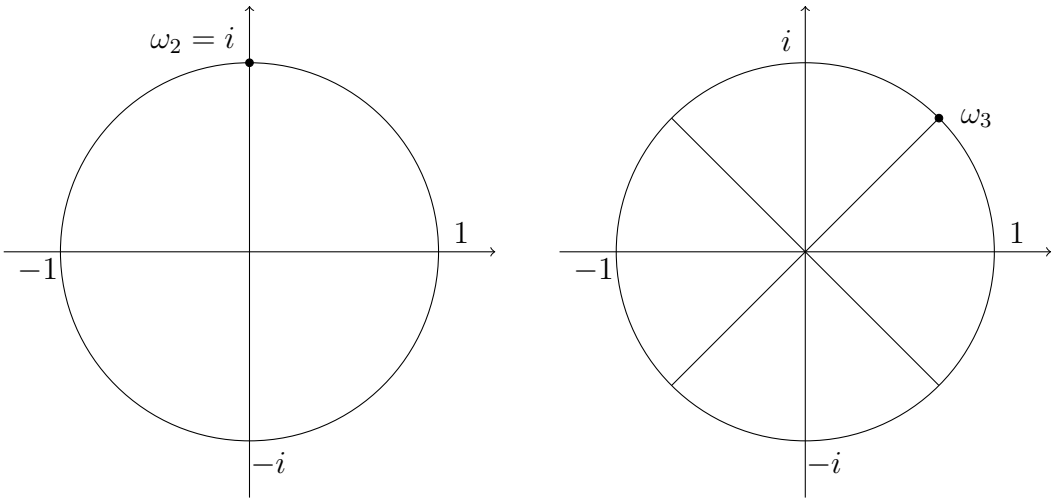


Figure 1: Roots of unity for $N = 2^2$ giving $\omega_2$ and $N = 2^3$ giving $\omega_3$.

Note that when dealing with $\omega_n$'s, we have a nice new property emerging. Namely that $\omega_{n+1}^2 = \omega_n$. This is quite intuitive if you think about dividing up the unit circle - in $\omega_{n+1}$ we're halving each sector, so we need to take two steps of $\omega_{n+1}$ to do one step of $\omega_n$. This holds in the general case too.

$$\omega_n = e^{\frac{2\pi i}{2^n}} = e^{\frac{2\pi i \cdot 2^k}{2^n \cdot 2^k}} = (e^{\frac{2\pi i}{2^{n+k}}})^{2^k} = \omega_{n+k}^{2^k}$$

This is something that we will need in the later proof, as we'll express things such as $\omega_2 = \omega_n^{2^{n-2}}$. It follows from above (take $k = n - 2$).

# 2 Implementing QFT

## 2.1 Our target

On a $N = 2^n$ system we want to implement QFT which is defined as

$$QFT|x\rangle \to \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{x \cdot y} |y\rangle$$

where the $xk$ stands for integer multiplication of the numbers represented by bitstrings $x$ and $k$. First let's look at a special case of a 1-qubit system. What we would want then is

$$QFT|x\rangle \to \frac{1}{\sqrt{2}} (\omega_1^{x \cdot 0}|0\rangle + \omega_1^{x \cdot 1}|1\rangle)$$

And since we only have two basis states we can even explicitly compute what each basis state is mapped to. Notice that $\omega_1 = -1$

$$QFT|0\rangle \to \frac{1}{\sqrt{2}} (\omega_1^{0 \cdot 0}|0\rangle + \omega_1^{0 \cdot 1}|1\rangle) = \frac{1}{\sqrt{2}} ((-1)^0|0\rangle + (-1)^0|1\rangle) = |+\rangle$$
$$QFT|1\rangle \to \frac{1}{\sqrt{2}} (\omega_1^{1 \cdot 0}|0\rangle + \omega_1^{1 \cdot 1}|1\rangle) = \frac{1}{\sqrt{2}} ((-1)^0|0\rangle + (-1)^1|1\rangle) = |-\rangle$$

Conveniently, it turns out that we can implement QFT on a 1-qubit system using just a Hadamard gate. To extend to larger systems, we'll use a recursive approach, where we assume the existence of a $n-1$ qubit QFT gate and use that to implement an $n$ qubit one.

*Note: In the practice we had a small mistake where we had $2^{n-2}$ instead of $2^{n-1}$ and I said I must have missed a two somewhere by arithmetic mistake. The mistake was here, as I mistakenly wrote $(-1) = \omega_2$ rather than $\omega_1$.*

## 2.2 Swapping

To make our life a little easier, we'll have the $QFT_n$ gate be slightly different, where the input first goes through bit flipping.

$$\overline{QFT}_n|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n} \omega_n^{\bar{x} \cdot y} |y\rangle$$

Notice that $\overline{QFT}_1 = QFT_1$ since swapping a bitstring of length 1 does nothing. Another thing to notice is that if we achieve implementing $\overline{QFT}_n$ then we can

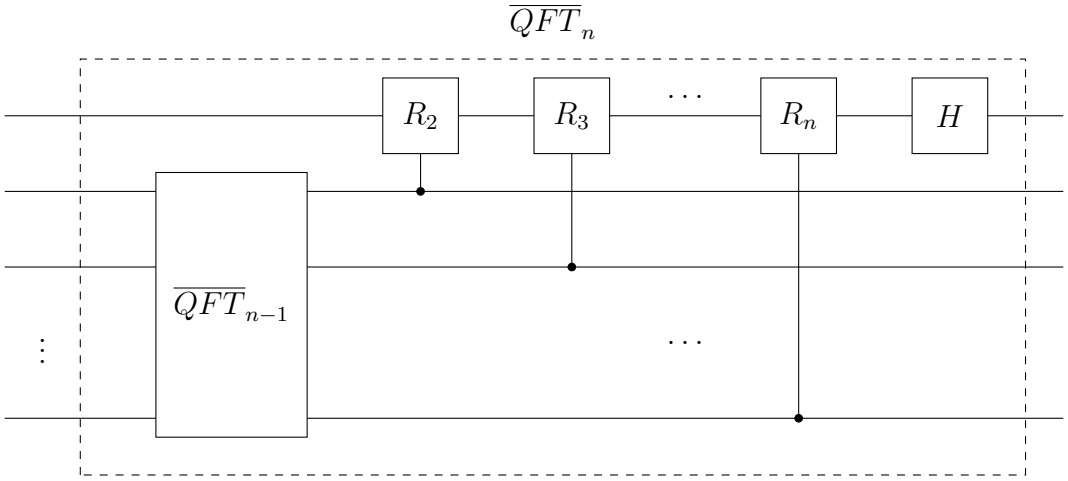


Figure 2: The construction of $\overline{QFT}_n$

make a $QFT_n$ by first flipping the input with a unitary that swaps qubits around $U|x\rangle \to |\overline{x}\rangle$ and then applying $\overline{QFT}_n$.

$$|x\rangle \Rightarrow U|x\rangle = |\overline{x}\rangle \Rightarrow \overline{QFT}_n|\overline{x}\rangle = \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n} \omega_n^{\overline{\overline{x}}\cdot y}|y\rangle = \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n} \omega_n^{x\cdot y}|y\rangle$$

$$|x\rangle \to \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n} \omega_n^{x\cdot y}|y\rangle$$

The above maps $|x\rangle$ to exactly what is needed in $QFT_n$. All that is really happening here is that flipping the input twice (once in U, once in the equation of $\overline{QFT}$) gets us back to normal.

## 2.3 Induction

So now that we've seen how it's enough for us to construct $\overline{QFT}_n$ and we know that a Hadamard gate is $\overline{QFT}_1$, if we can show how to create a $\overline{QFT}_n$ using only a $\overline{QFT}_{n-1}$ then we have successfully implemented QFT!

We will split out input into two parts, so we're dealing with $|bx\rangle = |b\rangle \otimes |x\rangle$ where $b$ is just a bit and $x$ is a $n-1$ length bitstring.

As we've now split the input, it is helpful to redefine our target ($\overline{QFT}_n$) in a way where the input is also split.

$$\overline{QFT}_n|bx\rangle = \frac{1}{\sqrt{2^n}} \sum_{c\in\{0,1\}, y\in\{0,1\}^{n-1}} \omega_n^{\overline{bx}\cdot cy}|cy\rangle = \frac{1}{\sqrt{2^n}} \omega_n^{b\cdot c\cdot 2^{n-1}+2\cdot\overline{x}\cdot y+b\cdot y+\overline{x}\cdot c\cdot 2^n}|cy\rangle$$

In the above equation, we're using the identity that we defined in subsection 1.1 where we talked about integer multiplication of $\overline{bx} \cdot cy$.

We first run the smaller $\overline{QFT}_{n-1}$ on the $x$ part and just apply the definition.

$$(I \otimes \overline{QFT}_{n-1})|bx\rangle \rightarrow \frac{1}{\sqrt{2^{n-1}}} \sum_{y=0}^{2^{n-1}} \omega_{n-1}^{\overline{x} \cdot y} |by\rangle$$

Now we will do the controlled rotations, where we rotate our first bit by $R_n$ is the $n$th bit is 1. The rotation $R_n$ itself is defined as

$$R_n = \begin{pmatrix} 1 & 0 \\ 0 & \omega_n \end{pmatrix}$$
$$R_n|0\rangle = |0\rangle$$
$$R_n|1\rangle = \omega_n|1\rangle$$

Notice that since $1 = \omega_n^0$, we can also write $R_n|b\rangle = \omega_n^b|b\rangle$. And because the $y_n$ bit controls whether we even do the rotation, we do a logical conjunction between $b$ and $y_{n-1}$ (both have to be 1 for the gate to do something), which for bits is a multiplication! So when expressing the controlled gate $C_n$ where the second bit is the control bit, the mapping looks like

$$C_n|00\rangle \rightarrow |00\rangle \ \textit{Control bit is 0, no rotation}$$
$$C_n|10\rangle \rightarrow |10\rangle \ \textit{Control bit is 0, no rotation}$$
$$C_n|01\rangle \rightarrow |01\rangle \ \textit{No rotation as } R_n|0\rangle = |0\rangle$$
$$C_n|11\rangle \rightarrow \omega_n|11\rangle$$
$$C_n|ab\rangle \rightarrow \omega_n^{a \cdot b}|ab\rangle$$

After so after we apply the first rotation gate $R_2$, our state will be

$$\frac{1}{\sqrt{2^{n-1}}} \sum_{y=0}^{2^{n-1}} \omega_2^{b \cdot y_1} \cdot \omega_{n-1}^{\overline{x} \cdot y} |by\rangle$$

Now we can express $\omega_2$ as $\omega_n$ to the power of something like in subsection 1.2.

$$\omega_2 = e^{\frac{2\pi i}{2^2}} = e^{\frac{2\pi i \cdot 2^{n-2}}{2^n}} = (e^{\frac{2\pi i}{2^n}})^{2^{n-2}} = (\omega_n)^{2^{n-2}}$$

And because we have $\omega_2^{b \cdot y_1} = \omega_2^{b \cdot y_1 \cdot 2^{n-2}}$ you can think of the multiplication of $2^{n-2}$ as effectively shifting one of those bits (let's say $y_1$) $n-2$ steps to the left. So $b$ is multiplied with a number where its bit representation is $n-1$ bits long and it starts with $y_1$.

Doing this with $R_3$ with the second bit $y_2$ gives us the coefficient $\omega_n^{b\cdot y_2\cdot 2^{n-3}}$, which is the $y_2$ bit shifted $n-3$ positions to the left. And in our state

$$\frac{1}{\sqrt{2^{n-1}}}\sum_{y=0}^{2^{n-1}}\omega_n^{b\cdot y_1\cdot 2^{n-2}}\cdot\omega_n^{b\cdot y_2\cdot 2^{n-3}}\cdot\omega_{n-1}^{\overline{x}\cdot y}|by\rangle$$

notice that we can unite the two new terms as they have the same base for the exponentiation.

$$\omega_n^{b\cdot y_1\cdot 2^{n-2}}\cdot\omega_n^{b\cdot y_2\cdot 2^{n-3}}=\omega_n^{b\cdot y_1\cdot 2^{n-2}+b\cdot y_2\cdot 2^{n-3}}=\omega_n^{b\cdot(y_1\cdot 2^{n-2}+y_2\cdot 2^{n-3})}$$

In the above equation, the sum $(y_1\cdot 2^{n-2}+y_2\cdot 2^{n-3})$ is what you would get if you turned the last $n-2$ bits of $y$ to 0. But if we keep doing this process, we eventually recover the entire $y$ this way.

$$\frac{1}{\sqrt{2^{n-1}}}\sum_{y=0}^{2^{n-1}}\omega_n^{b\cdot y_1\cdot 2^{n-2}}\cdot\omega_n^{b\cdot y_2\cdot 2^{n-3}}\cdot\ldots\cdot\omega_n^{b\cdot y_{n-1}}\cdot\omega_{n-1}^{\overline{x}\cdot y}|by\rangle=$$
$$\frac{1}{\sqrt{2^{n-1}}}\sum_{y=0}^{2^{n-1}}\omega_n^{b\cdot y_1\cdot 2^{n-2}+b\cdot y_2\cdot 2^{n-3}+\ldots+b\cdot y_{n-1}}\cdot\omega_{n-1}^{\overline{x}\cdot y}|by\rangle=$$
$$\frac{1}{\sqrt{2^{n-1}}}\sum_{y=0}^{2^{n-1}}\omega_n^{b\cdot y}\cdot\omega_{n-1}^{\overline{x}\cdot y}|by\rangle$$

We'll also move $\omega_n^{b\cdot y}\cdot\omega_{n-1}^{\overline{x}\cdot y}$ to a $\omega_n$ base. Remember that $\omega_{n-1}=\omega_n^2$ and thus our state is

$$\frac{1}{\sqrt{2^{n-1}}}\sum_{y=0}^{2^{n-1}}\omega_n^{b\cdot y}\cdot\omega_n^{2\cdot\overline{x}\cdot y}|by\rangle=$$
$$\frac{1}{\sqrt{2^{n-1}}}\sum_{y=0}^{2^{n-1}}\omega_n^{b\cdot y+2\cdot\overline{x}\cdot y}|by\rangle$$

Notice that we have all the terms of our target now that don't involve $c$. To introduce $c$ we'll run the bit $b$ through a Hadamard gate. Remember that from the 1-qubit case we know that

$$H|x\rangle=\frac{1}{\sqrt{2}}\sum_{c\in\{0,1\}}\omega_1^{b\cdot c}|c\rangle$$

$$(H\otimes I)\frac{1}{\sqrt{2^{n-1}}}\sum_{y=0}^{2^{n-1}}\omega_n^{b\cdot y+2\cdot\overline{x}\cdot y}|by\rangle=\frac{1}{\sqrt{2^{n-1}}}\frac{1}{\sqrt{2}}\sum_{c,y}\omega_1^{b\cdot c}\omega_n^{b\cdot y+2\cdot\overline{x}\cdot y}|cy\rangle$$

And since $\omega_1 = \omega_n^{2^{n-1}}$, we can add that to the $\omega_n$ base as well.

$$\frac{1}{\sqrt{2^n}} \sum_{c,y} \omega_1^{b \cdot c} \omega_n^{b \cdot y + 2 \cdot \overline{x} \cdot y} |cy\rangle = \frac{1}{\sqrt{2^n}} \sum_{c,y} \omega_n^{b \cdot c \cdot 2^{n-1} + b \cdot y + 2 \cdot \overline{x} \cdot y} |cy\rangle$$

Now the only summand that is missing is $\overline{x} \cdot c \cdot 2^n$. But remember that $\omega_n^{2^n} = 1$ as it does a complete circle. And thus we can say $1 = 1^a = \omega_n^{2^n \cdot a}$, and we can add in the last term.

$$\begin{aligned}
&\frac{1}{\sqrt{2^n}} \sum_{c,y} \omega_n^{b \cdot c \cdot 2^{n-1} + b \cdot y + 2 \cdot \overline{x} \cdot y} \cdot 1 |cy\rangle = \\
&\frac{1}{\sqrt{2^n}} \sum_{c,y} \omega_n^{b \cdot c \cdot 2^{n-1} + b \cdot y + 2 \cdot \overline{x} \cdot y} \cdot \omega_n^{2^n \cdot \overline{x} \cdot c} |cy\rangle = \\
&\frac{1}{\sqrt{2^n}} \sum_{c,y} \omega_n^{b \cdot c \cdot 2^{n-1} + b \cdot y + 2 \cdot \overline{x} \cdot y + 2^n \cdot \overline{x} \cdot c} |cy\rangle = \\
&\frac{1}{\sqrt{2^n}} \sum_{c,y} \omega_n^{\overline{bx} \cdot cy} |cy\rangle
\end{aligned}$$

And thus we have now successfully achieved the desired mapping, and shown that we can implement $\overline{QFT}_n$ by using $\overline{QFT}_{n-1}$, a Hadamard gate and some controlled rotations. And since $\overline{QFT}_1$ is just the Hadamard gate, we now know how to fully implement a $\overline{QFT}_n$ and thus a $QFT_n$.