Quantum Computing 6 – Quantum Fourier Transform and Applications

Angelo Bassi

The Quantum Fourier Transform

The **discrete Fourier transform** takes as input a vector of complex numbers, $x_0, \ldots, x_{N-1}$ where the length N of the vector is a fixed parameter. It outputs the transformed data, a vector of complex numbers $y_0, \ldots, y_{N-1}$, defined by

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi ijk/N}$$

In the **quantum Fourier transform**, we do a DFT on the amplitudes of a quantum state

$$\sum_{j=0}^{N-1} x_j |j\rangle \longrightarrow \sum_{k=0}^{N-1} y_k |k\rangle$$

where the amplitudes y_k are the discrete Fourier transform of the amplitudes x_j. On the **computational basis** it reads

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle$$

It is not obvious from the definition, but this transformation is a **unitary transformation**, and thus can be implemented as the dynamics for a quantum computer.

Since the output amplitudes are a linear combination of the input amplitudes, it is a linear operator. Its form is easy to write down

$$\hat{F} = \sum_{j,k=0}^{N-1} \frac{e^{2\pi i jk/N}}{\sqrt{N}} |k\rangle\langle j|$$

It is easy to check that it is unitary

$$\begin{aligned}
\hat{F}^\dagger \hat{F} &= \frac{1}{N} \sum_{j,k,j',k'} e^{2\pi i (j'k' - jk)/N} |j\rangle\langle j'| \delta_{kk'} \\
&= \frac{1}{N} \sum_{j,k,j'} e^{2\pi i (j'-j)k/N} |j\rangle\langle j'| \\
&= \sum_{j,j'} |j\rangle\langle j'| \delta_{jj'} = \sum_j |j\rangle\langle j| = \hat{I}.
\end{aligned}$$

The Fourier transform lets us define a new basis: $|\tilde{x}\rangle = F\,|x\rangle$, where $\{|x\rangle\}$ is the usual computational basis. This basis has a number of **interesting properties**. Every vector $|\tilde{x}\rangle$ is an equally weighted superposition of all the computational basis states:

$$\begin{aligned}
|\langle \tilde{x} | y \rangle|^2 &= \langle y | \tilde{x} \rangle \langle \tilde{x} | y \rangle = \langle y | \hat{F} | x \rangle \langle x | \hat{F}^\dagger | y \rangle \\
&= \frac{e^{2\pi i xy/N}}{\sqrt{N}} \frac{e^{-2\pi i xy/N}}{\sqrt{N}} = \frac{1}{N}.
\end{aligned}$$

From the point of view of physics, the relationship of this basis to the computational basis is analogous to that between the momentum and position bases of a particle.

Recall that the Hadamard transform could also turn computational basis states into equally weighted superpositions of all states. But it left all amplitudes real, while the amplitudes of $|\tilde{x}\rangle$ are complex.

In **matrix representation**:

$$\mathbf{QFT} = \frac{1}{\sqrt{N}}\begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix} \qquad \omega = e^{2\pi i/N}$$

In particular, for **1 qubit**:

$$\mathbf{QFT} = H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

while, for **2 qubits**:

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

Circuit for the Quantum Fourier Transform

At this point we specialize to the case of **n qubits**, so the dimension is **$N = 2^n$**.

We have seen that the quantum Fourier transform is a unitary operator. Therefore, by our earlier results, there is a quantum circuit which implements it. However, **there is no guarantee that this circuit will be efficient**! A general unitary requires a circuit with a number of gates exponential in the number of bits.

Very fortunately, in this case an efficient circuit does exist. (Fortunately, because the Fourier transform is at the heart of the most impressive quantum algorithms!)

Binary expression: $j \rightarrow j_1 j_2 \dots j_n$

$$j = j_1 2^{n-1} + j_2 2^{n-2} + \cdots + j_n$$

For example, for n = 2 we have $j = 2 j_1 + j_2$, therefore

$0 \rightarrow 00$
$1 \rightarrow 01$
$2 \rightarrow 10$
$3 \rightarrow 11$

as usual. We also consider the **binary fraction**

$$0.j_1 j_2 \dots j_n = j_1/2 + j_2/4 + \cdots + j_n/2^n = j/2^n$$

The key insight into designing a circuit for the Fourier transform is to notice that it can be written in a **product form**

$$|j_1, \ldots, j_n\rangle \rightarrow \frac{\left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i 0.j_1 j_2\cdots j_n}|1\rangle\right)}{2^{n/2}}$$

The proof is the following

$$\begin{aligned}
|j\rangle &\rightarrow \frac{1}{2^{n/2}}\sum_{k=0}^{2^n-1} e^{2\pi i jk/2^n}|k\rangle \\
&= \frac{1}{2^{n/2}}\sum_{k_1=0}^{1}\ldots\sum_{k_n=0}^{1} e^{2\pi i j\left(\sum_{l=1}^{n} k_l 2^{-l}\right)}|k_1\ldots k_n\rangle \\
&= \frac{1}{2^{n/2}}\sum_{k_1=0}^{1}\ldots\sum_{k_n=0}^{1}\bigotimes_{l=1}^{n} e^{2\pi i j k_l 2^{-l}}|k_l\rangle \\
&= \frac{1}{2^{n/2}}\bigotimes_{l=1}^{n}\left[\sum_{k_l=0}^{1} e^{2\pi i j k_l 2^{-l}}|k_l\rangle\right] \\
&= \frac{1}{2^{n/2}}\bigotimes_{l=1}^{n}\left[|0\rangle + e^{2\pi i j 2^{-l}}|1\rangle\right] \\
&= \frac{\left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i 0.j_1 j_2\cdots j_n}|1\rangle\right)}{2^{n/2}}
\end{aligned}$$

In going from the third to the fourth line, we used the identity

$$\begin{aligned}
\sum_{k_1=0}^{1}\cdots\sum_{k_n=0}^{1}\bigotimes_{l=1}^{n} f_{k_l} &= \sum_{k_1=0}^{1}\cdots\sum_{k_n=0}^{1} f_{k_1} f_{k_2}\ldots f_{k_n} \\
&= \bigotimes_{l=1}^{n}\sum_{k_l=0}^{1} f_{k_l} = \sum_{k_1=0}^{1} f_{k_1}\sum_{k_2=0}^{1} f_{k_2}\cdots\sum_{k_n=0}^{1} f_{k_n}
\end{aligned}$$

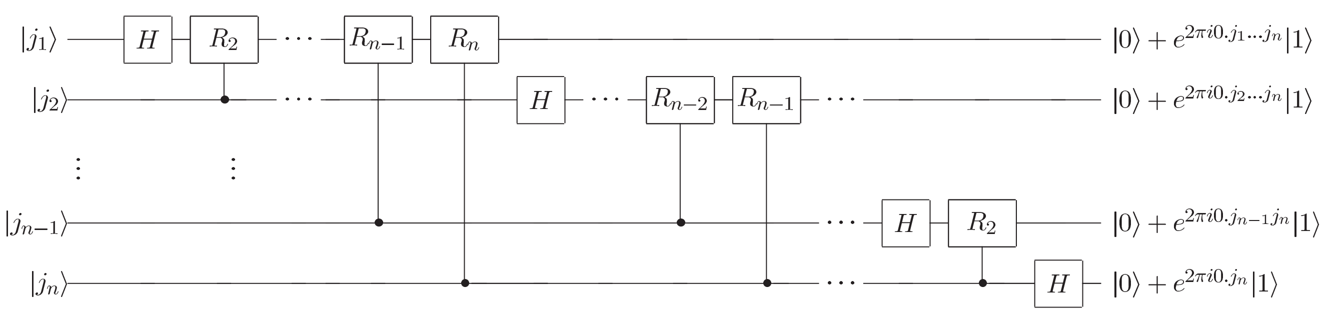
The unitary

$$|0,1\rangle \rightarrow (|0\rangle \pm \exp(i\theta)|1\rangle)/\sqrt{2}$$

is a Hadamard followed by a Z-rotation by θ. In the expression above the rotation depends on the values of the other bits. So **we should expect to be able to build the Fourier transform out of Hadamard and controlled-phase rotation gates.** Define the rotation

$$R_k \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$$

The circuit implementing the QFT then is



Let us see if it works. Applying the Hadamard gate to the first bit produces the state

$$\frac{1}{2^{1/2}}\left(|0\rangle + e^{2\pi i 0.j_1}|1\rangle\right)|j_2 \ldots j_n\rangle$$

since $e^{2\pi i\, 0.j_1} = -1$ when $j_1 = 1$, and is +1 otherwise. Applying the controlled-R_2 gate produces the state

$$\frac{1}{2^{1/2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2}|1\rangle\right)|j_2 \ldots j_n\rangle$$

We continue applying the controlled-R_3, R_4 through R_n gates, each of which adds an extra bit to the phase of the co-efficient of the first $|1\rangle$. At the end of this procedure we have the state

$$\frac{1}{2^{1/2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right)|j_2 \ldots j_n\rangle$$

Next, we perform a similar procedure on the second qubit. The Hadamard gate puts us in the state

$$\frac{1}{2^{2/2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_2}|1\rangle\right)|j_3 \ldots j_n\rangle$$

and the controlled-R_2 through R_{n-1} gates yield the state

$$\frac{1}{2^{2/2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_2 \ldots j_n}|1\rangle\right)|j_3 \ldots j_n\rangle$$

We continue in this fashion for each qubit, giving a final state

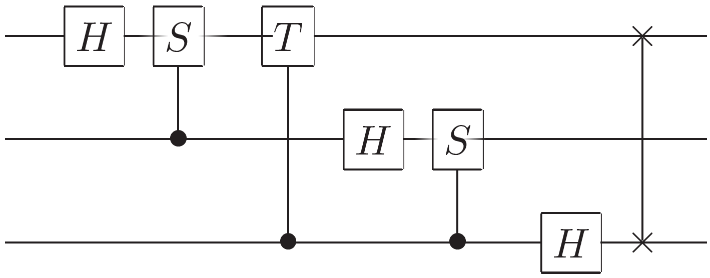
$$\frac{1}{2^{n/2}}\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \ldots j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_2 \ldots j_n}|1\rangle\right)\ldots\left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right)$$

If necessary, swap operations omitted from the Figure for clarity, can be used to reverse the order of the qubits. After the swap operations, the state of the qubits is

$$\frac{1}{2^{n/2}}\left(|0\rangle + e^{2\pi i 0.j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_{n-1} j_n}|1\rangle\right)\ldots\left(|0\rangle + e^{2\pi i 0.j_1 j_2 \cdots j_n}|1\rangle\right)$$

Box 5.1: Three qubit quantum Fourier transform

For concreteness it may help to look at the explicit circuit for the three qubit quantum Fourier transform:



Recall that S and T are the phase and $\pi/8$ gates (see page xxiii). As a matrix the quantum Fourier transform in this instance may be written out explicitly, using $\omega = e^{2\pi i/8} = \sqrt{i}$, as

$$\frac{1}{\sqrt{8}}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 & 1 & \omega^4 \\ 1 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ 1 & \omega^6 & \omega^4 & \omega^2 & 1 & \omega^6 & \omega^4 & \omega^2 \\ 1 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix}. \tag{5.19}$$

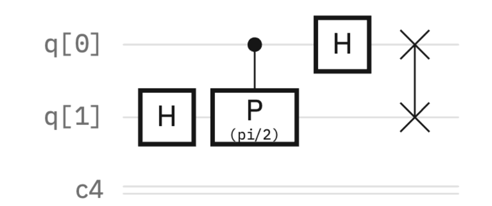
How many gates does this circuit use? We start by doing a Hadamard gate and $n-1$ conditional rotations on the first qubit – a total of n gates. This is followed by a Hadamard gate and $n-2$ conditional rotations on the second qubit, for a total of $n + (n-1)$ gates. Continuing in this way, we see that $n+(n-1)+\cdots+1 = n(n+1)/2$ gates are required plus the gates involved in the swaps. At most $n/2$ swaps are required, and each swap can be accomplished using three CNOT gates. Therefore, **this circuit provides a $\Theta(n^2)$ algorithm for performing the quantum Fourier transform**.

In contrast, the best classical algorithms for computing the discrete Fourier transform on 2^n elements are algorithms such as the **Fast Fourier Transform (FFT),** which compute the discrete Fourier transform using **$\Theta(n2^n)$ gates**.

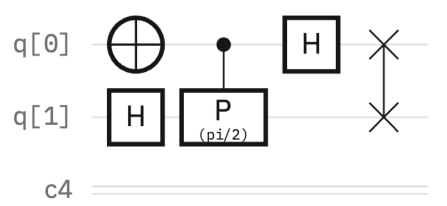
But remember, we cannot measure particular values of the amplitudes! The quantum Fourier Transform is useful only as a piece of another algorithm.

Exercise on IBM Quantum Composer.

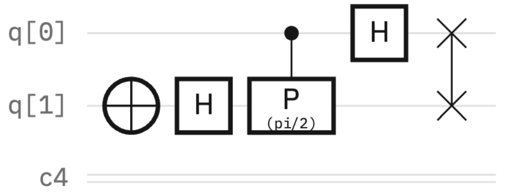
N = 2



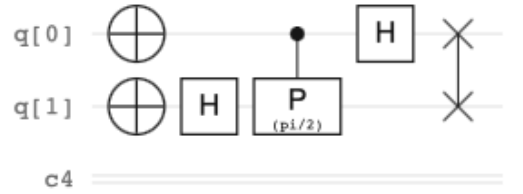
Opuput state
[0.5+0j, 0.5+0j, 0.5+0j, 0.5+0j]



Opuput state
[0.5+0j, 0+0.5j,-0.5+0j, 0-0.5j]



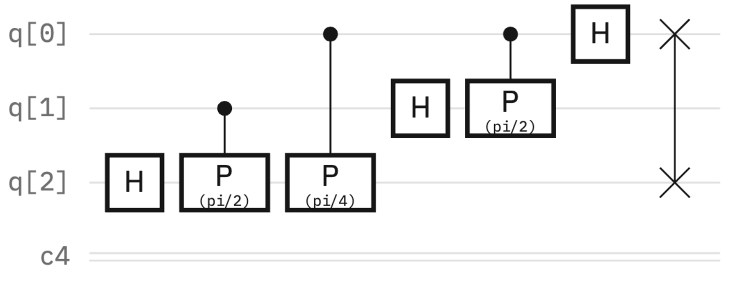
Opuput state
[0.5+0j,-0.5+0j, 0.5+0j,-0.5+0j]



Opuput state
[0.5+0j, 0-0.5j, -0.5+0j, 0+0.5j]

This corresponds to the matrix seen before

N = 3

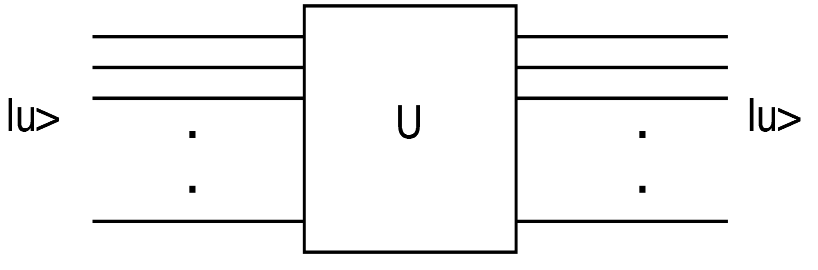


Phase estimation algorithm

The problem: Suppose we are given a unitary operator U on n qubits, which has a known eigenstate |u⟩ with an unknown eigenvalue $e^{2\pi i\phi}$, with ϕ in [0,1). We wish to find the phase ϕ with some precision.

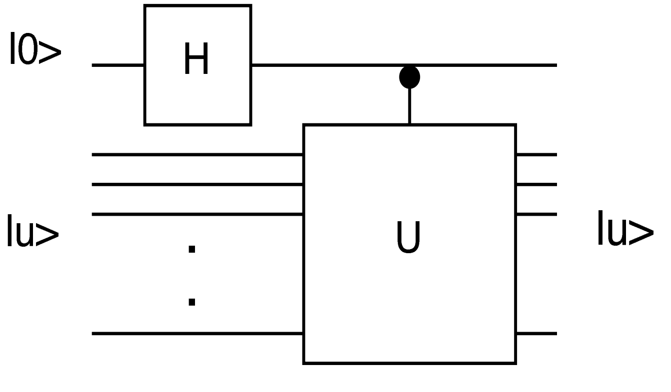
By itself, the phase estimation algorithm is a solution to a rather artificial problem. But this solution turns out to be useful as a piece of several other algorithms, to solve much more natural and important problems.

The first thing we might try is to prepare n q-bits in the state |u⟩, and carry out the unitary transformation U on them:



Is there a measurement on the bits which will give us information about the phase ϕ? The answer, of course, is no: Û just produces an **overall phase** on the state, with no observable consequences.

We need to generate relative phases, which can be measured. This can be done in the following way.



The initial state changes as follows:

$$\begin{aligned} |0\rangle|u\rangle &\to \frac{1}{\sqrt{2}}[|0\rangle + |1\rangle]|u\rangle \\ &\to \frac{1}{\sqrt{2}}[|0\rangle|u\rangle + |1\rangle e^{2\pi i\varphi}|u\rangle] = \frac{1}{\sqrt{2}}[|0\rangle + e^{2\pi i\varphi}|1\rangle]|u\rangle \end{aligned}$$

Now we have a **relative phase** among the two qubits of the computational basis, which can be measured for example by first applying an Hadamard gate to the first qubit, whose state then becomes:

$$\frac{1+e^{2\pi i\varphi}}{2}|0\rangle + \frac{1-e^{2\pi i\varphi}}{2}|1\rangle$$

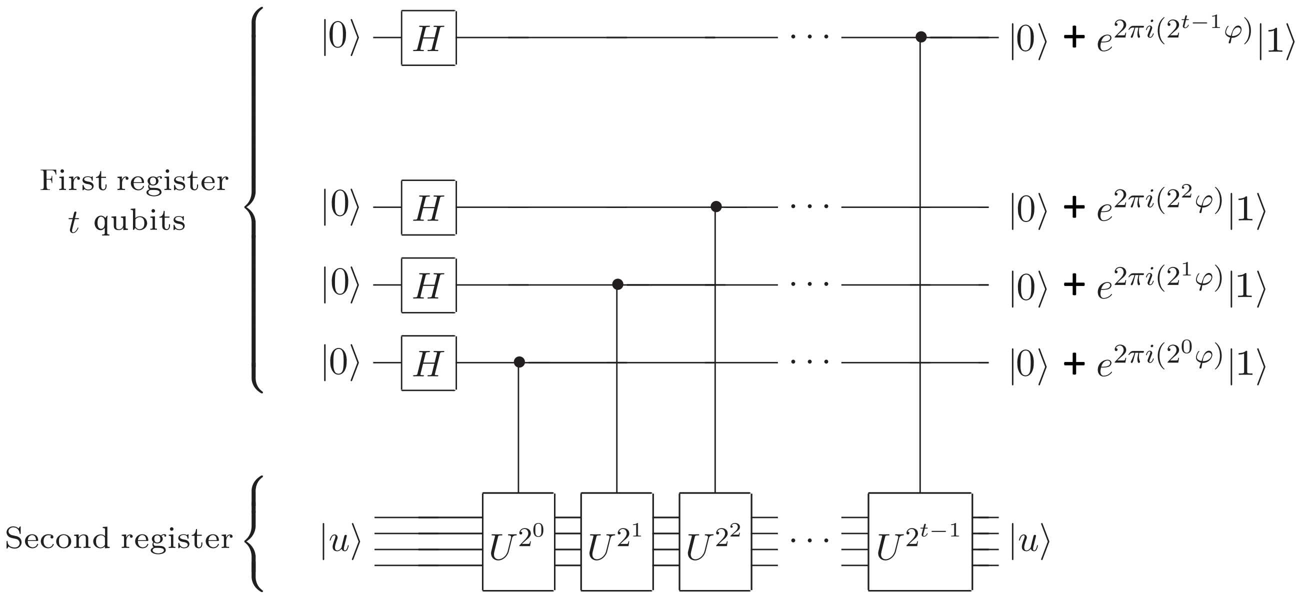
and by making a measurement on the computational basis we have the output probabilities:

$$\mathbb{P}[0] = \frac{1+\cos 2\pi\varphi}{2}, \qquad \mathbb{P}[1] = \frac{1-\cos 2\pi\varphi}{2}$$

We can run this circuit several times to recover the phase ϕ. Unfortunately, the convergence of this algorithm is very slow. After N repetitions, the accuracy in the estimate of the phase ϕ is $N^{-1/2}$. If we wish to know ϕ with m bits accuracy, then $N^{1/2} = 2^m$, which means $N = 2^{2m}$: the number of repetitions grows exponentially with the number of bits of accuracy.

A smarter solution is provided by the following algorithm.

The **first stage** of the algorithm is:



(normalization factors $2^{-1/2}$ have been omintted)

How we choose t depends on two things: the number of digits of accuracy we wish to have in our estimate for ϕ, and with what probability we wish the phase estimation procedure to be successful.

The final state of the first register is easily seen to be

$$\frac{1}{2^{t/2}}\left(|0\rangle + e^{2\pi i 2^{t-1}\varphi}|1\rangle\right)\left(|0\rangle + e^{2\pi i 2^{t-2}\varphi}|1\rangle\right)\dots\left(|0\rangle + e^{2\pi i 2^{0}\varphi}|1\rangle\right)$$

$$= \frac{1}{2^{t/2}}\sum_{k_1=0}^{1}\dots\sum_{k_t=0}^{1} e^{2\pi i\varphi(k_1 2^{t-1}+k_2 2^{t-2}+\dots+k_t 2^0)}|k_1 k_2 \dots k_t\rangle = \frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi i\varphi k}|k\rangle\,.$$

Suppose ϕ may be expressed exactly in t bits, as $\phi = 0.\phi_1 \dots \phi_t$. Then

$$\begin{aligned} e^{2\pi i\phi} &= e^{2\pi i 0.\phi_1\dots\phi_t}. \\ e^{4\pi i\phi} &= e^{2\pi i\phi_1.\phi_2\dots\phi_t} = e^{2\pi i\phi_1 + 2\pi i 0.\phi_2\dots\phi_t} = e^{2\pi i 0.\phi_2\dots\phi_t} \\ e^{2^j\pi i\phi} &= e^{2\pi i 0.\phi_j\dots\phi_t}. \end{aligned}$$

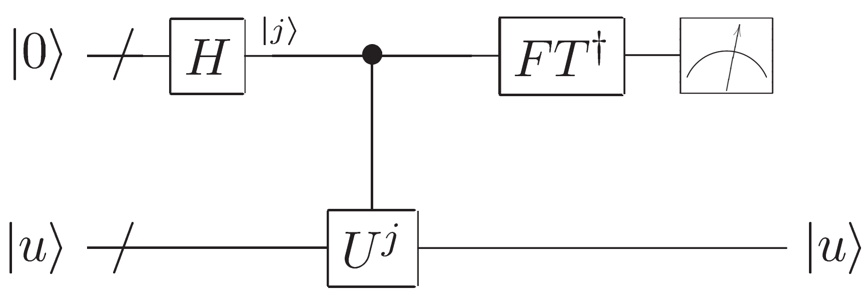
The output state can be rewritten as

$$\frac{1}{2^{t/2}}\left(|0\rangle + e^{2\pi i 0.\varphi_t}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.\varphi_{t-1}\varphi_t}|1\rangle\right)\ldots\left(|0\rangle + e^{2\pi i 0.\varphi_1\varphi_2\cdots\varphi_t}|1\rangle\right)$$

which is the Quantum Fourier Transform of the state $|\varphi_1 \cdots \varphi_t\rangle$

Therefore the **second stage** of the algorithm is to apply the inverse Quantum Fourier Transform to the output of the first state, and one recovers exactly the bits of the binary fraction for ϕ.

The full phase estimation algorithm is:



The above analysis applies to the ideal case, where ϕ can be written exactly with a t bit binary expansion. What happens when this is not the case?

Applying the inverse QFT to the state in the previous page produces the state

$$\frac{1}{2^t}\sum_{k,l=0}^{2^t-1} e^{\frac{-2\pi i k l}{2^t}} e^{2\pi i \varphi k}|l\rangle = \frac{1}{2^t}\sum_{k,l=0}^{2^t-1} e^{-\frac{2\pi i}{2^t}(l-2^t\varphi)k}|l\rangle$$

We see again that if $\phi = 0.\phi_1 \ldots \Phi_t$, then $2^t\,\phi$ is an integer and the sum over k returns a Kronecker delta, forcing it to be equal to $2^t\,\phi$. The final state is $|\phi_1 \cdots \phi_t\rangle = |2^t\phi\rangle$

If this is not the case, the coefficient associated to the state |l> is:

$$= \frac{1}{2^t}\sum_{k=0}^{2^t-1} e^{-\frac{2\pi i}{2^t}(l-2^t\varphi)k} = \frac{1}{2^t}\frac{1-e^{2\pi i(l-2^t\varphi)}}{1-e^{2\pi i(l-2^t\varphi)/2^t}}$$

And its square modulus is

$$\frac{1}{2^{2t}}\frac{1-\cos[2\pi(l-2^t\varphi)]}{1-\cos[2\pi(l-2^t\varphi)/2^t]}$$

The above function is sharply peaked around the closest l to 2^t ϕ. More precisely, it can be shown (see Nielsen & Chuang) that to successfully obtain ϕ accurate to n bits with probability of success at least 1 − ε, we choose

$$t = n + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right)\right\rceil$$

The number of qubits needed to run the algorithm with the desired accuracy grows linearly. Assuming that the controlled-U^{2j} unitaries are given by oracles (and hence free), the complexity of the algorithm is basically that of the Quantum Fourier Transform, $O(t^2)$. We have an exponential advantage with respect to the naïve algorithm we first tried.

However, if we have to perform circuits for the controlled-U^{2j} unitaries, than things change. Even if we have an efficient circuit for controlled-U, we need efficient circuits for all the controlled-U^{2j} gates as well; just repeating the controlled-U 2^j times will make the complexity exponential.

There is another somewhat artificial assumption as well. It is assumed that we don't know the eigenvalue $e^{2\pi i\phi}$, but that we can prepare the eigenvector |u⟩. While this may sometimes be true, in most cases it will not be.

The phase estimation algorithm then is:

Inputs: (1) A black box wich performs a controlled-U^j operation, for integer j, (2) an eigenstate $|u\rangle$ of U with eigenvalue $e^{2\pi i\varphi_u}$, and (3) $t = n + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right)\right\rceil$ qubits initialized to $|0\rangle$.

Outputs: An n-bit approximation $\widetilde{\varphi_u}$ to φ_u.

Runtime: $O(t^2)$ operations and one call to controlled-U^j black box. Succeeds with probability at least $1 - \epsilon$.

Procedure:

1.	$\|0\rangle\|u\rangle$	initial state
2.	$\rightarrow \frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}\|j\rangle\|u\rangle$	create superposition
3.	$\rightarrow \frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}\|j\rangle U^j\|u\rangle$	apply black box
	$= \frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1}e^{2\pi i j\varphi_u}\|j\rangle\|u\rangle$	result of black box
4.	$\rightarrow \|\widetilde{\varphi_u}\rangle\|u\rangle$	apply inverse Fourier transform
5.	$\rightarrow \widetilde{\varphi_u}$	measure first register

Example 1. Consider the unitary operator (X gate)

$$U = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

with eigenstate $|+\rangle = \frac{1}{\sqrt{2}}[|0\rangle + |1\rangle]$ and we know the relative eigenvalue is a phase $\lambda = e^{2\pi i\theta}$.

The goal is to find the phase, with 1 bit precision.

1. The initial state is:

$$|0\rangle|+\rangle$$

2. We apply a Hadamard to the first qubit to create the superposition:

$$\frac{1}{\sqrt{2}}[|0\rangle + |1\rangle]|+\rangle$$

3. We apply the controlled-U gate once: the state remains unchanged

4. The inverse Fourier transform, which amounts to an Hadamard, brings the state back to >

$$|0\rangle|+\rangle$$

5. Measuring the first register gives 0, from which we learn that the phase, in binary fraction, is 0.0 (to 1 bit accuracy).

This is correct: we know that the eigenvalue is 1, therefore the phase is 0.

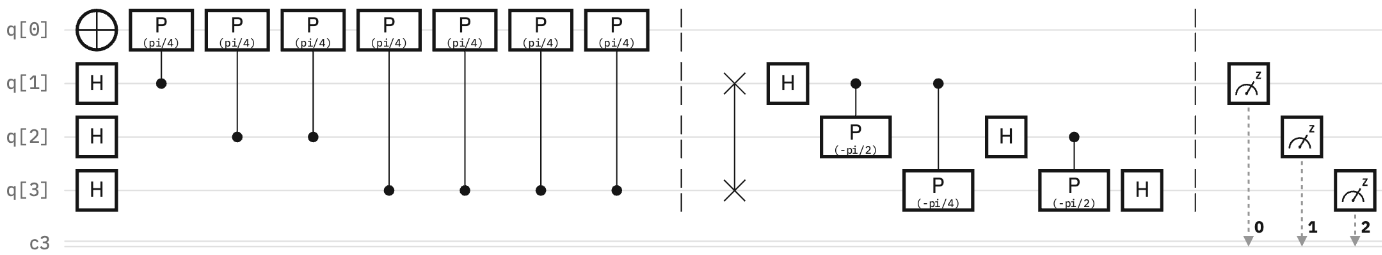
Exercise 1. Consider the unitary operator (T gate)

$$T = \begin{bmatrix} 1 & 0 \\ 0 & \exp(i\pi/4) \end{bmatrix}$$

with eigenstate |1>. Write down the circuit to find the associated eigenvalue (which is assumed to be a phase) with 3 bit precision.

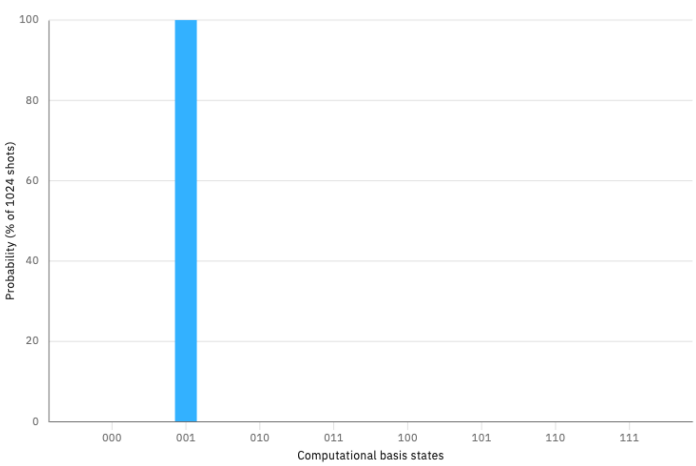
Solution: since the associated eigenvalue is $e^{i\pi/4}$, the desired phase is 1/8, which corresponds to the binary fraction 0.001. Therefore the algorithm returns 001, corresponding to the exact value of the phase.

On the **IBM Quantum Composer**, it looks as follows

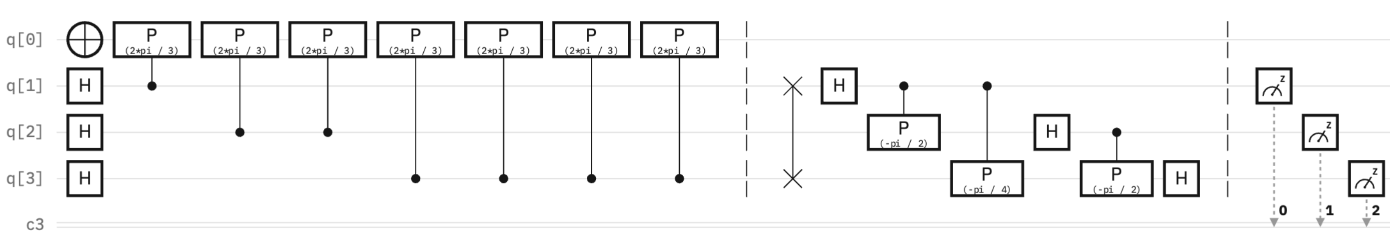


Remember that qubits are ordered from bottom to top.
(use “freeform alignment” to place the gate as desired)

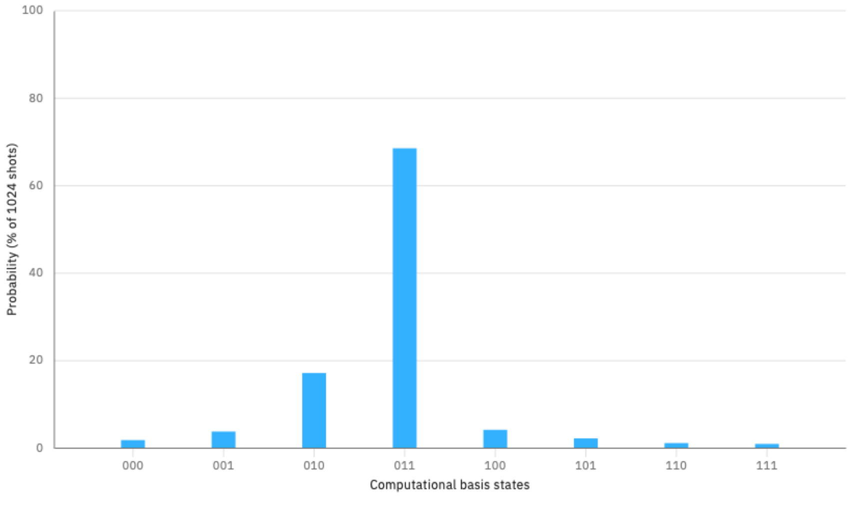
The **output probabilities** are



Exercise 2. Consider the same case as before, with a phase ϕ = 1/3 = 0,33333 (not binary fraction). With 3-bit precision the algorithm is similar to the one before:



The output probabilities are

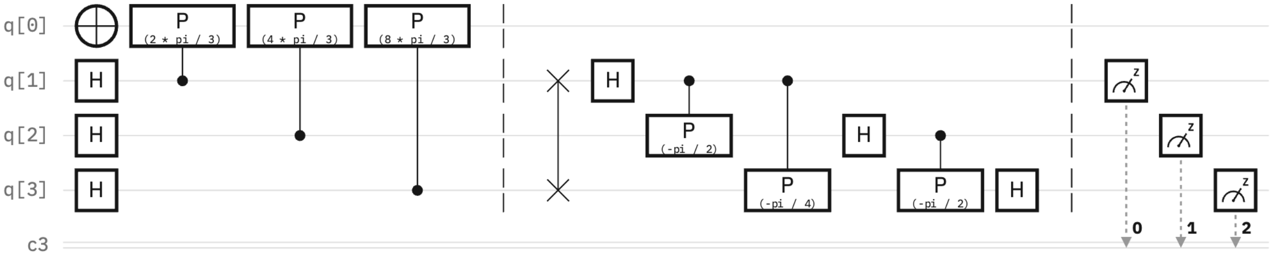


Most likely outcome: 011 → binary fraction 0.011 corresponding to a phase ϕ = 1/4 + 1/8 = 0,375 → 0,042 difference from exact result (off by 13%).

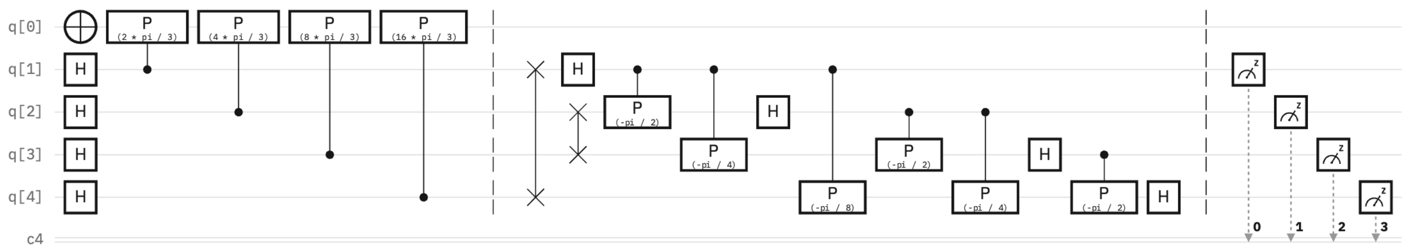
The second most likely outcome is: 010 → binary fraction 0.010 corresponding to a phase ϕ = 1/4 = 0,25 → 0,083 difference from exact result (off by 25%).

The true phase lies in between, closer to the most likely outcome.

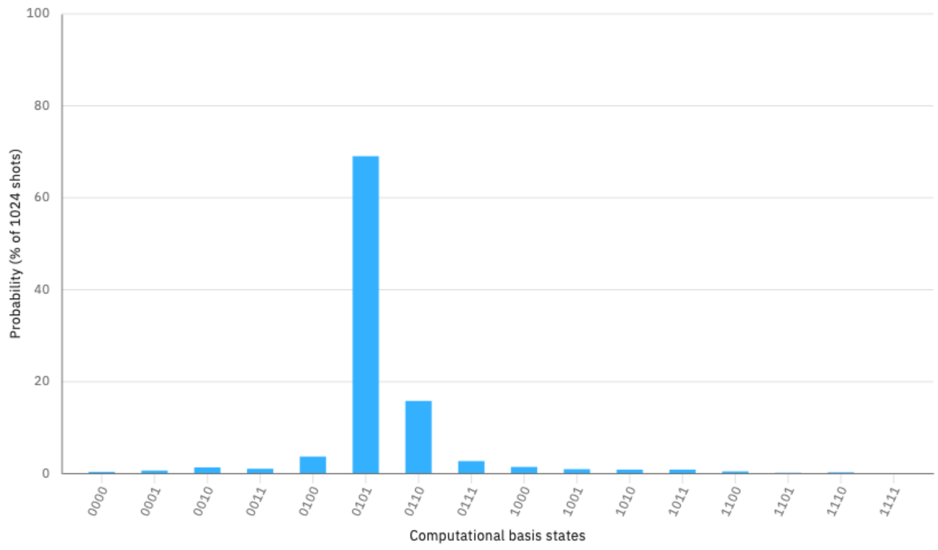
The circuit can be written more shortly as follows



Exercise 3. Same as before, but with 4-bit precision. The circuit is



The output probabilities are

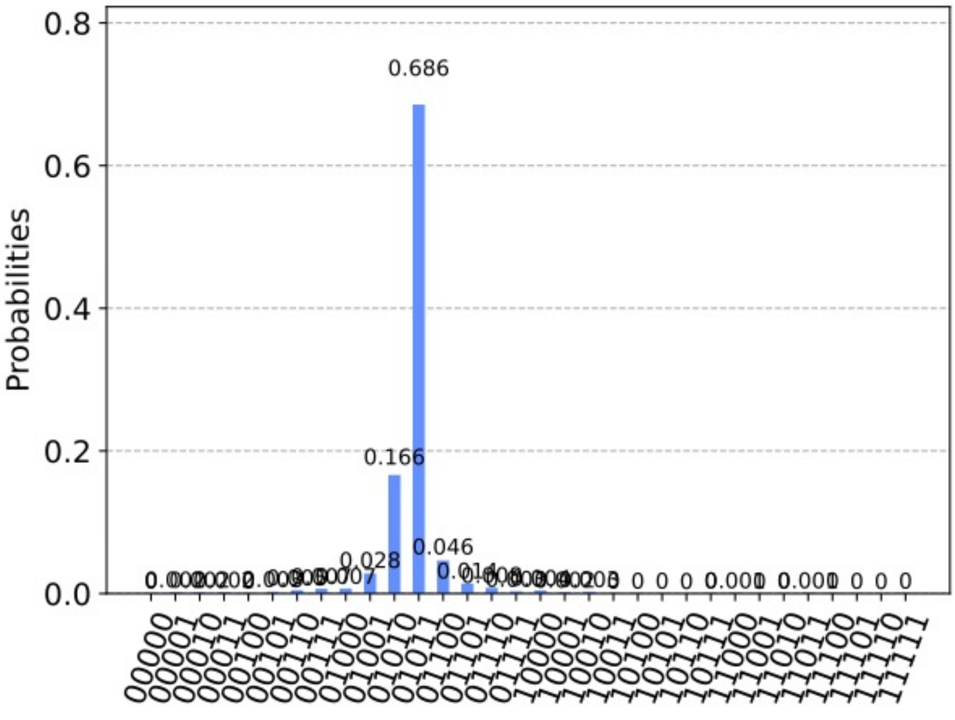


Most likely outcome: 0101 → binary fraction 0.0101 corresponding to a phase ϕ = 1/4 + 1/16 = 0,313 → 0,02 difference from exact result (off by 6%).

The second most likely outcome is: 0110 → binary fraction 0.0110 corresponding to a phase ϕ = 1/4 + 1/8 = 0,375 → 0,042 difference from exact result (off by 13%).

We see an improvement with respect to the case with 3 bits

Exercise 4. Same as before, but with 5-bit precision. The outcome probabilities are



Most likely outcome: 01011 → binary fraction 0.01011 corresponding to a phase ϕ = 1/4 + 1/16 + 1/32 = 0,344 → 0,011 difference from exact result (off by 3%).

The second most likely outcome is: 01010 → binary fraction 0.01010 corresponding to a phase ϕ = 1/4 + 1/16 = 0,313 → 0,02 difference from exact result (off by 6%).

Again, we see an improvement with respect to the previous cases.

Order finding algorithm

Definition of order: For positive integers x and N , x < N , with no common factors, the order of x modulo N is defined to be the least positive integer, r, such that $x^r = 1 (\mathrm{mod}\ N)$.

Order finding problem: to determine the order for some specified x and N.

Order-finding is believed to be a hard problem on a classical computer. The quantum algorithm for order-finding is just the phase estimation algorithm applied to the unitary operator

$$U|y\rangle \equiv |xy(\mathrm{mod}\ N)\rangle$$

with y ∈ {0, 1, … 2^L-1}, with L to be defined later. Note that here and below, when N ≤ y ≤ 2^L −1, we use the convention that xy(mod N) is just y again. That is, U only acts non-trivially when 0 ≤ y ≤ N − 1.

Example: N = 15, x = 7. Then:

U |0> = |0>
U |1> = |7>
U |2> = |14>
U |3> = |6>
U |4> = |13>
U |5> = |5>
U |6> = |12>
U |7> = |4>

U |8> = |11>
U |9> = |3>
U |10> = |10>
U |11> = |2>
U |12> = |9>
U |13> = |1>
U |14> = |8>
U |15> = |15>

U |16> = |16>
U |17> = |17>
and so on

A simple calculation shows that the states defined by

$$|u_s\rangle \equiv \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i s k}{r}\right] |x^k \bmod N\rangle$$

for integer $0 \leq s \leq r - 1$ are eigenstates of U, since

$$\begin{aligned} U|u_s\rangle &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp\left[\frac{-2\pi i s k}{r}\right] |x^{k+1} \bmod N\rangle \\ &= \exp\left[\frac{2\pi i s}{r}\right] |u_s\rangle \,. \end{aligned}$$

(this because x^r mod N = 1, by definition).

Using the phase estimation procedure allows us to obtain, with high accuracy, the corresponding eigenvalues $\exp^{2\pi is/r}$, from which we can obtain the order r with a little bit more work.

There are three important requirements to be met in order for the algorithm to be efficient:

- We must have efficient procedures to implement a controlled-U^{2j} operation for any integer j.
- We must be able to efficiently prepare an eigenstate $|u_s\rangle$ with a non-trivial eigenvalue.
- We must be able to obtain the desired answer, r, from the result of the phase estimation algorithm, $\phi \approx s/r$.

We analyse the three elements separately.

Implementation of the controlled-U^{2j} operation: modular exponentiation. The following relation holds:

$$
\begin{aligned}
|z\rangle|y\rangle \rightarrow |z\rangle U^{z_t 2^{t-1}} \dots U^{z_1 2^0} |y\rangle \\
= |z\rangle|x^{z_t 2^{t-1}} \times \cdots \times x^{z_1 2^0} y(\mathrm{mod}\ N)\rangle \\
= |z\rangle|x^z y(\mathrm{mod}\ N)\rangle.
\end{aligned}
$$

Thus the sequence of controlled-U^{2j} operations used in phase estimation is equivalent to multiplying the contents of the second register by the modular exponential x^z(mod N), where z is the contents of the first register.

This operation may be accomplished classically using $O(L^3)$ gates. The classical circuit can be transformed into a reversible circuit, which can be translated into a quantum circuit of similar complexity, computing the transformation $|z\rangle|y\rangle \rightarrow |z\rangle|x^z y \text{ (mod N)}\rangle$. *The book of Nakahara & Ohimi (p. 156) explains in detail how to do it.*

Prepare an eigenstate $|u_s\rangle$. Preparing $|u_s\rangle$ requires that we know r, so this is out of the question. Fortunately, there is a clever observation which allows us to circumvent the problem of preparing $|u_s\rangle$, which is that

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

This means that if we prepare the second register in the state |1>, just before measurement the state of the two registers will be:

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

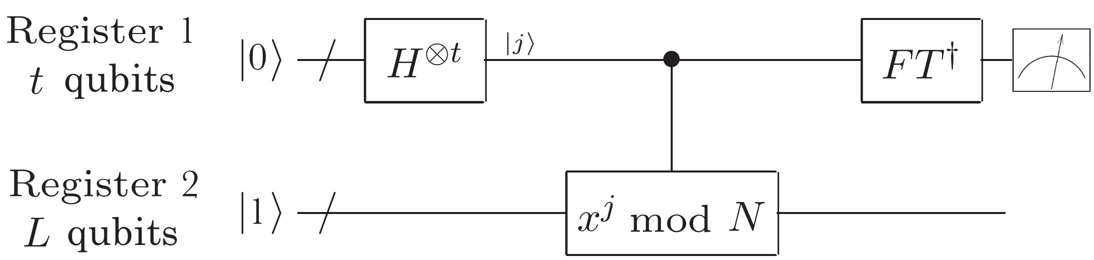
A measurement of the first register will collapse the state of the second register to the eigenstate $|u_s>$, and the first register will end up in the state $|\phi_s>$, from which the phase $\phi \approx s/r$ can be read.

Therefore, if we use

$$t = 2L + 1 + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right)\right\rceil$$

qubits in the first register and prepare the second register in the state $|1\rangle$, which is trivial to construct, it follows that for each s in the range 0 through r − 1, we will obtain an estimate of the phase $\phi \approx s/r$ accurate to 2L + 1 bits, with probability at least $(1 - \varepsilon)/r$.

The order finding algorithm the is:



How to extract r from $\phi \approx s/r$. We only know ϕ to 2L + 1 bits, but we also know a priori that it is a rational number – the ratio of two integers – and if we could compute the nearest such fraction to ϕ we might obtain r.

There is an algorithm which accomplishes this task efficiently, known as the continued fractions algorithm: given ϕ the continued fractions algorithm efficiently produces numbers s′ and r′ with no common factor, such that $s'/r' = s/r$. The number r′ is our candidate for the order. We can check to see whether it is the order by calculating $x^{r'}$ mod N, and seeing if the result is 1. If so, then r′ is the order of x modulo N, and we are done.

Performance. How can the order-finding algorithm fail? There are two possibilities.

First, the phase estimation procedure might produce a bad estimate to s/r. This occurs with probability at most ε, and can be made small with a negligible increase in the size of the circuit.

More seriously, it might be that s and r have a common factor, in which case the number r′ returned by the continued fractions algorithm be a factor of r, and not r itself. Fortunately, there are at least three ways around this problem. Perhaps the most straightforward way is to note that for randomly chosen s in the range 0 through r − 1, it's actually pretty likely that s and r are co-prime, in which case the continued fractions algorithm must return r. Specifically, one can show that by repeating the algorithm 2log(N) times we will, with high probability, observe a phase s/r such that s and r are co-prime, and therefore the continued fractions algorithm produces r, as desired.

See Nielsen and Chuang for further details.

Note. The quantum state produced in the order-finding algorithm, before the inverse Fourier transform, is

$$|\psi\rangle = \sum_{j=0}^{2^t-1} |j\rangle U^j |1\rangle = \sum_{j=0}^{2^t-1} |j\rangle |x^j \bmod N\rangle$$

if we initialize the second register as $|1\rangle$. The same state is obtained if we replace U^j with a different unitary transform V, which computes

$$V|j\rangle|k\rangle = |j\rangle|k + x^j \bmod N\rangle$$

and start the second register in the state $|0\rangle$. Moreover, V can be constructed also using $O(L^3)$ gates.

The order finding algorithm then is

Inputs: (1) A black box $U_{x,N}$ which performs the transformation $|j\rangle|k\rangle \to |j\rangle|x^j k \bmod N\rangle$, for x co-prime to the L-bit number N, (2) $t = 2L + 1 + \lceil \log(2 + \frac{1}{2\epsilon}) \rceil$ qubits initialized to $|0\rangle$, and (3) L qubits initialized to the state $|1\rangle$.

Outputs: The least integer $r > 0$ such that $x^r = 1 \pmod N$.

Runtime: $O(L^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. $|0\rangle|1\rangle$ — initial state

2. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|1\rangle$ — create superposition

3. $\to \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j \bmod N\rangle$ — apply $U_{x,N}$

$$\approx \frac{1}{\sqrt{r2^t}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{2\pi i s j/r} |j\rangle|u_s\rangle$$

4. $\to \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |\widetilde{s/r}\rangle|u_s\rangle$ — apply inverse Fourier transform to first register

5. $\to \widetilde{s/r}$ — measure first register

6. $\to r$ — apply continued fractions algorithm

Example: Find the order r of x = 7 mod N = 15. We use L = 8 qubits. The initial state is:

$$|\mathbf{00...0}\rangle_8\, |\mathbf{00...0}\rangle_8 = |0\rangle\, |0\rangle$$

Next, we apply the Hadamard transformation to the first register

$$\frac{1}{\sqrt{256}}\Big(\underbrace{|\mathbf{00...0}\rangle_8}_{=0} + ...\, \underbrace{|\mathbf{11...1}\rangle_8}_{=255} \Big) |\mathbf{00...0}\rangle_8$$

The next step is to perform the modular exponentiation. One gets

$$\frac{1}{\sqrt{256}}\Big(\left|0\right\rangle \left|1\right\rangle + \left|1\right\rangle \left|7\right\rangle + \left|2\right\rangle \left|4\right\rangle + \left|3\right\rangle \left|13\right\rangle + \left|4\right\rangle \left|1\right\rangle + \left|5\right\rangle \left|7\right\rangle ... \left|255\right\rangle \left|13\right\rangle \Big)$$

which can be rewritten as

$$\begin{aligned} &= \frac{1}{\sqrt{4}}\left(\frac{\left|0\right\rangle + \left|4\right\rangle + ... + \left|252\right\rangle}{\sqrt{64}}\right)\left|1\right\rangle + \frac{1}{\sqrt{4}}\left(\frac{\left|1\right\rangle + \left|5\right\rangle + ... + \left|253\right\rangle}{\sqrt{64}}\right)\left|7\right\rangle \\ &+ \frac{1}{\sqrt{4}}\left(\frac{\left|2\right\rangle + \left|6\right\rangle + ... + \left|254\right\rangle}{\sqrt{64}}\right)\left|4\right\rangle + \frac{1}{\sqrt{4}}\left(\frac{\left|3\right\rangle + \left|7\right\rangle + ... + \left|255\right\rangle}{\sqrt{64}}\right)\left|13\right\rangle \end{aligned}$$

DA CONTINUARE SU QISKIT

Factoring

The factoring problem turns out to be equivalent to the order-finding problem we just studied, in the sense that a fast algorithm for order-finding can easily be turned into a fast algorithm for factoring. The reduction of factoring to order-finding proceeds in two basic steps.

The **first step** is to show that we can compute a **factor of N** if we can find a non-trivial solution x neq ± 1(mod N) to the equation **x^2 = 1(mod N).**

The **second step** is to show that a **randomly chosen y co-prime to N** is quite likely to have an order r which is even, and such that $y^{r/2}$ neq ± 1(mod N). **Thus x ≡ $y^{r/2}$(mod N) is a non-trivial solution to x^2 = 1(mod N).**

The algorithm runs as follows.

Inputs: A composite number N

Outputs: A non-trivial factor of N.

Runtime: $O((\log\ N)^3)$ operations. Succeeds with probability $O(1)$.

Procedure:

1. If N is even, return the factor 2.

2. Determine whether $N = a^b$ for integers $a \geq 1$ and $b \geq 2$, and if so return the factor a (uses the classical algorithm of Exercise 5.17).

3. Randomly choose x in the range 1 to $N-1$. If $\gcd(x, N) > 1$ then return the factor $\gcd(x, N)$.

4. Use the order-finding subroutine to find the order r of x modulo N.

5. If r is even and $x^{r/2} \neq -1 (\mathrm{mod}\ N)$ then compute $\gcd(x^{r/2} - 1, N)$ and $\gcd(x^{r/2} + 1, N)$, and test to see if one of these is a non-trivial factor, returning that factor if so. Otherwise, the algorithm fails.

Steps 1 and 2 of the algorithm either return a factor, or else ensure that **N is an odd integer with more than one prime factor**. These steps may be performed using O(1) and $O(L^3)$ operations, respectively.

Step 3 either returns a factor, or produces a **randomly chosen element x** of {0, 1, 2, . . . , N − 1}, **co-prime to N**.

Step 4 calls the order-finding subroutine, computing the order r of x modulo N.

Step 5 completes the algorithm, since Theorem 5.3 of Nielsen & Chuang guarantees that with probability at least one-half, r will be even and $x^{r/2}$ neq − 1(mod N), and then Theorem 5.2 of Nielsen & Chuang guarantees that either $\gcd(x^{r/2} - 1, N)$ or $\gcd(x^{r/2} + 1, N)$ is a non-trivial factor of N.

Example: Factoring **N = 15**.

15 is neither even, nor of the form a^b with a greater or equal to 1 and b greater or equal to 2. Therefore steps 1 and 2 do not return anything.

Step 3 requires to pick randomly a number between 1 and d 14. Following the previous example, suppose we choose **x = 7**. It is co-prime to 15.

Step 4 makes use of the order finding algorithm to find the order r of x = 7 mod N = 15. We saw that the output is **r = 4**.

By chance, 4 is even, and more over $x^{r/2}$ mod 15 = 4 differ from -1 mod 15, so the algorithm works. Computing the greatest common divisor $\gcd(x^2 - 1, 15) = 3$ and $\gcd(x^2 + 1, 15) = 5$ tells us that **15 = 3×5**.