CS 294-2 Spring 2007

# Simon's Algorithm + QFT

2/7/07 Lecture 7

## 0.1

Recall that our basic primitive for designing quantum algorithms is fourier sampling: prepare some quantum state $|\psi\rangle = \sum_x \alpha_x |x\rangle$ on $n$ qubits; perform a Hadamard transform, resulting in the superposition $\sum_x \beta_x |x\rangle$; now measure to sample $x$ with probability $|\beta_x|^2$. The point is that classically it is difficult to simulate the effects of the quantum interference, and therefore to determine for which strings $x$ there is constructive interference and are therefore output with high probability.

Today we will see two introduce two innovations that will prepare us for the quantum algorithm for factoring.

1) A more sophisticated method for preparing the initial superposition $|\psi\rangle = \sum_x \alpha_x |x\rangle$?

2) Generalizing the fourier transform from the Hadamard transform (which is a fourier transform over the group $Z_2^n$ to the group $Z_N$.

## 0.2 Setting up a random pre-image state

Suppose we're given a classical circuit for a $k-1$ function $f: \{0,1\}^n \to \{0,1\}^n$.

We will show how to set up the quantum state $|\phi\rangle = 1/\sqrt{k}\sum_{x:f(x)=a} |x\rangle$. Here $a$ is uniformly random among all $a$ in the image of $f$.

The algorithm uses two registers, both with $n$ qubits. The registers are initialized to the basis state $|0\cdots 0\rangle |0\cdots 0\rangle$. We then perform the Hadamard transform $H_{2^n}$ on the first register, producing the superposition

$$\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle |0\cdots 0\rangle .$$

Then, we compute $f(x)$ through the oracle $C_f$ and store the result in the second register, obtaining the state

$$\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle |f(x)\rangle .$$

The second register is not modified after this step. Thus we may invoke the principle of safe storage and assume that the second register is measured at this point.

Let $a$ be the result of measuring of the second register. Then $a$ is a random element in the range of $f$, and according to rules of partial measurement, the state of the first register is a superposition over exactly those values of $x$ that are consistent with those contents for the second register. i.e.

$$|\phi\rangle = 1/\sqrt{k} \sum_{x:f(x)=a} |x\rangle$$

## 0.3 Simon's Algorithm

Suppose we are given function $2-1$ $f: \{0,1\}^n \to \{0,1\}^n$, specified by a black box, with the promise that there is an $a \in \{0,1\}^n$ with $a \neq 0^n$ such that

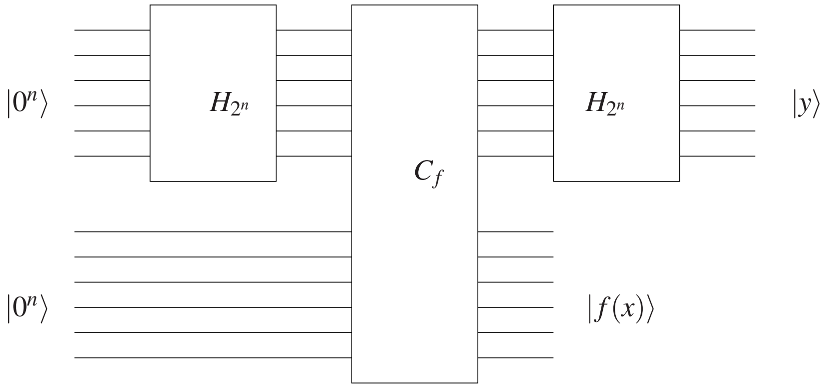


Figure 1: Simon's algorithm

- For all $x$ $f(x+a) = f(x)$.
- If $f(x) = f(y)$ then either $x = y$ or $y = x+a$.

The challenge is to determine $a$. It is intuitively obvious that this is a difficult task for a classical probabilistic computer. We will show an efficient quantum algorithm.

## 0.4 Simon's Algorithm

1. Use $f$ to set up random pre-image state

$$\phi = 1/\sqrt{2}|z\rangle + 1/\sqrt{2}|z+a\rangle$$

where $z$ is a random $n$-bit string.

2. Perform a Hadamard transform $H^{\otimes n}$.

After step 2 we obtain a superposition

$$\sum_{y \in \{0,1\}^n} \alpha_y |y\rangle$$

where

$$\alpha_y = \frac{1}{\sqrt{2}} \frac{1}{2^{n/2}} (-1)^{y \cdot z} + \frac{1}{\sqrt{2}} \frac{1}{2^{n/2}} (-1)^{y \cdot (z \oplus a)} = \frac{1}{2^{(n+1)/2}} (-1)^{y \cdot z} \left[1 + (-1)^{y \cdot a}\right].$$

There are now two cases. For each $y$, if $y \cdot a = 1$, then $\alpha_y = 0$, whereas if $y \cdot a = 0$, then

$$\alpha_y = \frac{\pm 1}{2^{(n-1)/2}}.$$

So when we observe the first register, with certainty we'll see a $y$ such that $y \cdot a = 0$. Hence, the output of the measurement is a random $y$ such that $y \cdot a = 0$. Furthermore, each $y$ such that $y \cdot a = 0$ has an equal probability of occurring. Therefore what we've managed to learn is an equation

$$y_1 a_1 \oplus \cdots \oplus y_n a_n = 0 \tag{1}$$

where $y = (y_1, \ldots, y_n)$ is chosen uniformly at random from $\{0,1\}^n$. Now, that isn't enough information to determine $a$, but assuming that $y \neq 0$, it reduces the number of possibilities for $a$ by half.

It should now be clear how to proceed. We run the algorithm over and over, accumulating more and more equations of the form in (1). Then, once we have enough of these equations, we solve them using Gaussian elimination to obtain a unique value of $a$. But how many equations is enough? From linear algebra, we know that $a$ is uniquely determined once we have $n-1$ linearly independent equations—in other words, $n-1$ equations

$$\begin{aligned} y^{(1)} \cdot a &\equiv 0 \,(\mathrm{mod}\, 2) \\ &\vdots \\ y^{(n-1)} \cdot a &\equiv 0 \,(\mathrm{mod}\, 2) \end{aligned}$$

such that the set $\left\{y^{(1)}, \ldots, y^{(n-1)}\right\}$ is linearly independent in the vector space $Z_2^n$. Thus, our strategy will be to lower-bound the probability that any $n-1$ equations returned by the algorithm are independent.

Suppose we already have $k$ linearly independent equations, with associated vectors $y^{(1)}, \ldots, y^{(k)}$. The vectors then span a subspace $S \subseteq Z_2^n$ of size $2^k$, consisting of all vectors of the form

$$b_1 y^{(1)} + \cdots + b_k y^{(k)}$$

with $b_1, \ldots, b_k \in \{0, 1\}$. Now suppose we learn a new equation with associated vector $y^{(k+1)}$. This equation will be independent of all the previous equations provided that $y^{(k+1)}$ lies *outside* of $S$, which in turn has probability at least $(2^n - 2^k)/2^n = 1 - 2^{k-n}$ of occurring. So the probability that any $n$ equations are independent is exactly the product of those probabilities.

$$\left(1 - \frac{1}{2^n}\right) \times \left(1 - \frac{1}{2^{n-1}}\right) \times \cdots \times \left(1 - \frac{1}{4}\right) \times \left(1 - \frac{1}{2}\right).$$

Can we lower-bound this expression? Trivially, it's at least

$$\prod_{k=1}^{\infty} \left(1 - \frac{1}{2^k}\right) \approx 0.28879;$$

the infinite product here is related to something in analysis called a q-series. Another way to look at the constant $0.28879\ldots$ is this: it is the limit, as $n$ goes to infinity, of the probability that an $n \times n$ random matrix over $Z_2$ is invertible.

But we don't need heavy-duty analysis to show that the product has a constant lower bound. We use the inequality $(1-a)(1-b) = 1 - a - b + ab > 1 - (a+b)$, if $a, b \in (0, 1)$. We just need to multiply the product out, ignore monomials involving two or more $\frac{1}{2^k}$ terms multiplied together (which only increase the product), and observe that the product is lower-bounded by

$$\left[1 - \left(\frac{1}{2^n} + \frac{1}{2^{n-1}} + \cdots + \frac{1}{4}\right)\right] \cdot \frac{1}{2} \geq \frac{1}{4}.$$

We conclude that we can determine $a$ with constant probability of error after repeating the algorithm $O(n)$ times. So the number of queries to $f$ used by Simon's algorithm is $O(n)$. The number of computation steps, though, is at least the number of steps needed to solve a system of linear equations, and the best known upper bound for this is $O\left(n^{2.376}\right)$, due to Coppersmith and Winograd.

## 0.5 Classical solution

We are going to prove that any probabilistic algorithm needs an exponential time to solve this problem. Suppose that $a$ is chosen uniformly at random from $\{0,1\}^n - \{0^n\}$. Now consider a classical probabilistic algorithm that's already made $k$ queries, to inputs $x_1, \ldots, x_k$. We want to know how much information the algorithm could have obtained about $a$, given those queried pairs $(x_i, f(x_i))$.

On the one hand, there might be a pair of inputs $x_i, x_j$ (with $1 \le i, j \le k$) such that $f(x_i) = f(x_j)$. In this case, the algorithm already has enough information to determine $a$: $a = x_i \oplus x_j$.

On the other hand, suppose no such pair $f(x_i), f(x_j)$ exists. Then the queried $f(x_i)$'s are distinct and $a$ is none of $\binom{k}{2}$ values $x_i \oplus x_j$.

The probability that the next query will succeed is at most

$$\frac{k}{2^n - 1 - \binom{k}{2}}$$

because there are at least $2^n - 1 - \binom{k}{2}$ possible values of u for choosing at the $(k+1)$-th query. And $f(x_{k+1})$ should be equal to one of the prior observed $f(x_i)$, $i \in [1,k]$.

Taking the sum over all $k \in \{1, \ldots, m\}$. We get

$$\sum_{k=1}^{m} \frac{k}{2^n - 1 - \binom{k}{2}} \le \sum_{k=1}^{m} \frac{k}{2^n - k^2} \le \frac{m^2}{2^n - m^2}$$

In order to have an constant probability, we must choose $m = \Omega(2^{n/2})$. Hence, any deterministic algorithm has to run in exponential time to get a correct answer with probability larger than a constant.

# Fourier transform on $\mathbf{Z}_N$

Let $f$ be a complex-valued function on $\mathbf{Z}_N$. Then its Fourier transform is

$$\hat{f}(t) = \frac{1}{\sqrt{N}} \sum_{x \in \mathbf{Z}_N} f(x) w^{xt}$$

where $w = \exp(2\pi i/N)$. Let $B_1$ be the standard basis for $\mathscr{C}^{Z_N}$ consisting of vectors $f_i(j) = \delta_{i,j}$. In the standard basis the matrix for the Fourier transform is

$$FT_N = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & w^3 & \cdots & w^{N-1} \\ 1 & w^2 & w^4 & w^6 & \cdots & w^{2N-2} \\ 1 & w^3 & w^6 & w^9 & \cdots & w^{3N-3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{N-1} & w^{2N-2} & w^{3N-3} & \cdots & w^{(N-1)(N-1)} \end{pmatrix}$$

where $i, j$'th entry of $FT_N$ is $w^{ij}$.

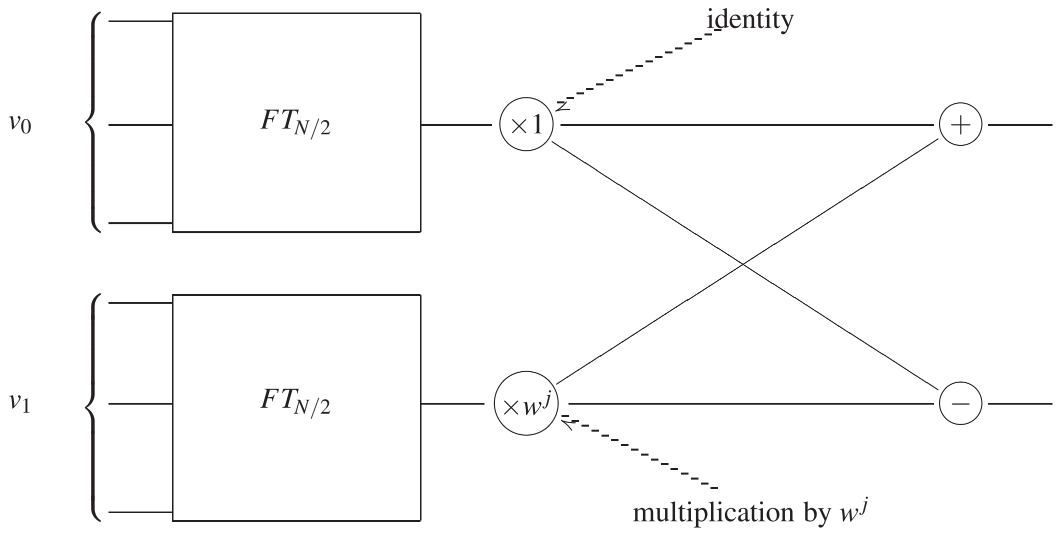


Figure 2: A circuit for classical fast Fourier transform

# Classical fast Fourier transform

Straightforward multiplication of the vector $f$ by $FT_N$ would take $\Omega(N^2)$ steps because multiplication of $f$ by each row requires $N$ multiplications. However, there is an algorithm known as fast Fourier transform (FFT) that performs Fourier transform in $O(N\log N)$ operations.

In our presentation of FFT we shall restrict ourselves to the case $N = 2^n$. Let $B_2$ be a basis for $\mathscr{C}^{\mathbf{Z}_N}$ consisting of vectors

$$f_i(j) = \begin{cases} \delta_{2i,j}, & i \in \{0,1,\ldots,N/2-1\}, \\ \delta_{2i-N+1,j}, & i \in \{N/2, N/2+1,\ldots,N-1\}, \end{cases}$$

i.e., the vectors of the standard basis sorted by the least-significant bit. Then as a map from $B_2$ to $B_1$ the Fourier transform has the matrix representation

$$\begin{array}{c c} \text{bit }\# & \begin{array}{cc} \quad 2k \quad & \quad 2k+1 \end{array} \\ \begin{array}{c} j \\ j+N/2 \end{array} & \left(\begin{array}{c|c} w^{2jk} & w^{2jk}w^j \\ \hline w^{2jk} & w^{2jk}w^j \end{array}\right) \end{array} = \begin{pmatrix} FT_{N/2} & w^j FT_{N/2} \\ FT_{N/2} & -w^j FT_{N/2} \end{pmatrix}.$$

Hence,

$$\left(\begin{array}{c|c} w^{2jk} & w^{2jk}w^j \\ \hline w^{2jk} & w^{2jk}w^j \end{array}\right) \left(\begin{array}{c} v_0 \\ \hline v_1 \end{array}\right) = \begin{pmatrix} FT_{N/2}v_0 + w^j FT_{N/2}v_1 \\ FT_{N/2}v_0 - w^j FT_{N/2}v_1 \end{pmatrix}.$$

This representation gives a recursive algorithm for computing the Fourier transform in time $T(N) = 2T(N/2) + O(N) = O(N\log N)$. As a circuit the algorithm can be implemented as

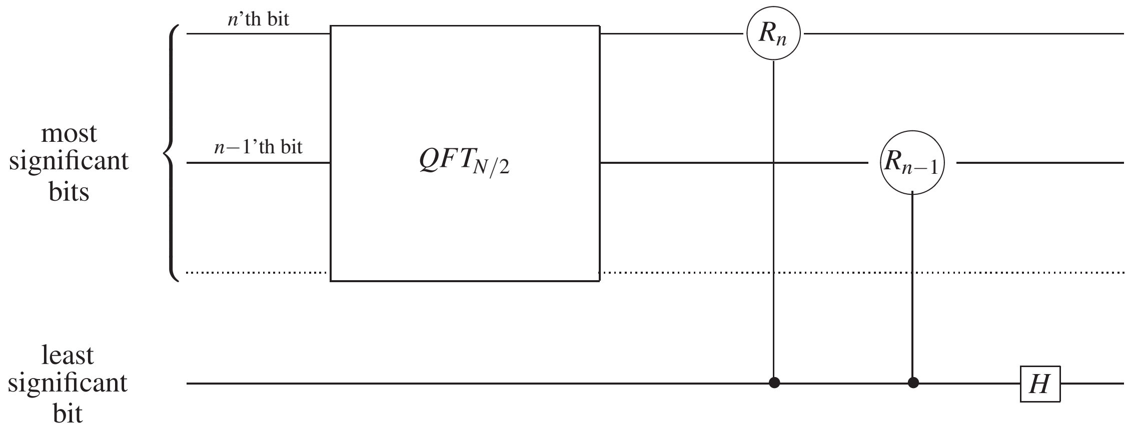


Figure 3: Circuit for quantum Fourier transform

# Quantum Fourier transform

Let $N = 2^n$. Suppose a quantum state $\alpha$ on $n$ qubits is given as $\sum_{j=0}^{N-1} \alpha_j |j\rangle$. Let the Fourier transform of $\phi$ be $FT_N|\phi\rangle = \sum_{j=0}^{N-1} \beta_j |j\rangle$ where

$$FT_N \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{N-1} \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{N-1} \end{pmatrix}.$$

The map $FT_N = |\alpha\rangle \mapsto |\beta\rangle$ is unitary (see the proof below), and is called the quantum Fourier transform (QFT). A natural question arises whether it can be efficiently implemented quantumly. The answer is that it can be implemented by circuit of size $O(\log^2 N)$. However, this does not constitute an exponential speed-up over the classical algorithm because the result of quantum Fourier transform is a superposition of states which can be observed, and any measurement can extract at most $n = \log N$ bits of information.

A quantum circuit for quantum Fourier transform is where $R_K$ is the controlled phase shift by angle $2\pi/2^K$ whose matrix is

$$R_K = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{2\pi/2^K} \end{pmatrix}.$$

In the circuity above the quantum Fourier transform on $n-1$ bits corresponds to two Fourier transforms on $n-1$ bits in the figure **??**. The controlled phase shifts correspond to multiplications by $w^j$ in classical circuit. Finally, the Hadamard gate at the very end corresponds to the summation.

# Properties of Fourier transform

- $FT_N$ is unitary. Proof: the inner product of the $i$'th and $j$'th column of $FT_N$ where $i \neq j$ is

$$\frac{1}{N}\sum_{k\in\mathbf{Z}_N} w^{ik}\overline{w^{jk}} = \frac{1}{N}\sum_{k\in\mathbf{Z}_N} w^{ik-jk} = \frac{1}{N}\sum_{k\in\mathbf{Z}_N} (w^{i-j})^k = \frac{1}{N}\frac{w^{N(i-j)}-1}{w^{i-j}-1} = \frac{1}{N}\frac{1-1}{w^{i-j}-1}$$

which is zero because $w^{i-j} \neq 1$ due to $i \neq j$. The norm of $i$'th column is

$$\sqrt{\frac{1}{N}\sum_{k\in\mathbf{Z}_N} w^{ik}\overline{w^{ik}}} = \sqrt{\frac{1}{N}\sum_{k\in\mathbf{Z}_N} 1} = 1.$$

- $FT_N^{-1}$ is $FT_N$ with $w$ replaced by $w^{-1}$. Proof: since $FT$ is unitary we have $F_N^{-1} = FT_N^*$. Since $FT_N$ is symmetric and $\bar{w} = w^{-1}$, the result follows.

- Fourier transform sends translation into phase rotation, and vice versa. More precisely, if we let the translation be $T_l\colon \left|x\right\rangle \mapsto \left|x+l \pmod N\right\rangle$ and rotation by $P_k\colon \left|x\right\rangle \mapsto w^{kx}\left|x\right\rangle$, then $FT_N P_l P_k = P_l T_{-k} FT_N$. Proof: by linearity it suffices to prove this for a vector of the form $\left|x\right\rangle$. We have

$$FT_N T_l P_k\left|x\right\rangle = FT_N w^{kx}\left|x+l \pmod N\right\rangle = \frac{1}{\sqrt{N}} w^{kx}\sum_{y\in\mathbf{Z}_N} w^{y(x+l)}\left|y\right\rangle$$

and by making the substitution $y = y' - k$

$$\begin{aligned} &= \frac{1}{\sqrt{N}} w^{y'x}\sum_{y'\in\mathbf{Z}_N} w^{(y'-k)l}\left|y'-k\right\rangle = \frac{1}{\sqrt{N}} P_l T_{-k}\sum_{y'\in\mathbf{Z}_N} w^{xy}\left|y'\right\rangle \\ &= P_l T_{-k} FT_N\left|x\right\rangle. \end{aligned}$$

Corollary: $FT_N$ followed by Fourier sampling is equivalent to $T_l FT_N$ followed by Fourier sampling.

- Suppose $r \mid N$. Let $\left|\phi\right\rangle = \frac{1}{\sqrt{N/r}}\sum_{j=0}^{N/r-1}\left|jr\right\rangle$. Then $FT_N\left|\phi\right\rangle = \frac{1}{\sqrt{r}}\sum_{i=0}^{r-1}\left|i\frac{N}{r}\right\rangle$. Proof: the amplitude of $\left|i\frac{N}{r}\right\rangle$ is

$$\frac{1}{\sqrt{N}}\frac{1}{\sqrt{N/r}}\sum_{j=0}^{N/r-1} w^{(jr)(iN/r)} = \frac{\sqrt{r}}{N}\sum_{j=0}^{N/r-1} 1 = \frac{1}{\sqrt{r}}$$

Since $FT_N$ is unitary, the norm of $FT_N\left|\phi\right\rangle$ has to be equal to the norm of $\left|\phi\right\rangle$ which is 1. However the orthogonal projection of $FT_N\left|\phi\right\rangle$ on the space spanned by vectors of the form $\left|i\frac{N}{r}\right\rangle$ has norm 1. Therefore $FT_N\left|\phi\right\rangle$ lies in that space.

If we apply the corollary above to $\left|\phi\right\rangle$ we conclude that the result of Fourier sampling of $T_l\left|\phi\right\rangle = \frac{\sqrt{r}}{\sqrt{N}}\sum_{j=0}^{N/r-1}\left|jr+l\right\rangle$ is a random multiples of $N/r$.