CS 593/MA 595 - Introduction to Quantum Computing

Quantum Computation and Quantum Information by *Isaac Chuang, Michael Nielsen*
Student: **Ralph Razzouk**

Homework 7

Problem 1
In this problem, you'll prove the two mathematical facts we needed to know in order for Simon's algorithm to work.

(a) Let A be a finite abelian group. Prove that if $g_1, \ldots, g_l$ are l independently and uniformly randomly chosen elements of A, then the probability that $\langle g_1, g_2, \ldots, g_l \rangle = A$ is at least $1 - \frac{|A|}{2^l}$. [Hint: as an intermediate step, you might use Lagrange's theorem to argue that the probability that $g_{i+1} \notin \langle g_1, \ldots, g_i \rangle$ is at least 1/2 whenever $\langle g_1, \ldots, g_i \rangle \neq A$.]

(b) Let $A = (\mathbb{Z}/2\mathbb{Z})^n$ be an n dimensional vector space over $\mathbb{Z}/2\mathbb{Z}$. Let $s \in A$ be a non-zero element and suppose $g_1, \ldots, g_l \in A$ generate what I called $\langle s \rangle^\perp$, which is defined as

$$\langle s \rangle^\perp = \{a \in A \mid a \cdot s = 0 \bmod 2\},$$

where $a \cdot s$ is the modulo 2 dot product of $a = (a_1, \ldots, a_n)$ and $s = (s_1, \ldots, s_n)$. Prove that s is the unique non-zero solution to the system of equations

$$\begin{aligned} g_1 \cdot x &= 0 \mod 2 \\ g_2 \cdot x &= 0 \mod 2 \\ &\vdots \\ g_l \cdot x &= 0 \mod 2. \end{aligned}$$

Proof. (a) Let A be a finite abelian group and take $g_1, \ldots, g_l$ to be l independently and uniformly chosen elements of A. We will prove this by mathematical induction.

- **For $l = 1$:** We have only one element from A, namely g_1. Then $\mathbb{P}(\langle g_1 \rangle = A) = 1$ since we can construct the group from non-zero elements of the group.
- **For $l = k$:** We now make the inductive step and assume that, for $l = k \geq 1$,

$$\mathbb{P}(\langle g_1, g_2, \ldots, g_k \rangle = A) \geq 1 - \frac{|A|}{2^k}.$$

- **For $l = k + 1$:** We now need to show that

$$\mathbb{P}(\langle g_1, g_2, \ldots, g_k, g_{k+1} \rangle = A) \geq 1 - \frac{|A|}{2^{k+1}}.$$

From the inductive step, we know that $\mathbb{P}(\langle g_1, g_2, \ldots, g_k \rangle = A) \geq 1 - \frac{|A|}{2^k}$. If $\langle g_1, g_2, \ldots, g_k \rangle \neq A$, then $g_{k+1} \notin \langle g_1, g_2, \ldots, g_k \rangle$. Then $\mathbb{P}(g_{k+1} \notin \langle g_1, g_2, \ldots, g_k \rangle) = 1 - \frac{|\langle g_1, g_2, \ldots, g_k \rangle|}{|A|}$ (by Lagrange's Theorem). Thus

$$\begin{aligned} \mathbb{P}(\langle g_1, g_2, \ldots, g_k \rangle = A) &\geq \mathbb{P}(\langle g_1, g_2, \ldots, g_k \rangle = A) \times \mathbb{P}(g_{k+1} \notin \langle g_1, g_2, \ldots, g_k \rangle) \\ &\geq \left(1 - \frac{|A|}{2^k}\right)\left(1 - \frac{|\langle g_1, g_2, \ldots, g_k \rangle|}{|A|}\right) \\ &\geq \left(1 - \frac{|A|}{2^k}\right)\left(\frac{1}{2}\right) \\ &\geq 1 - \frac{|A|}{2^{k+1}}. \end{aligned}$$

Thus, with proof by induction, we have

$$\mathbb{P}(\langle g_1, g_2, \ldots, g_k \rangle = A) \geq 1 - \frac{|A|}{2^k}.$$

(b) For $1 \leq i \leq l$, s satisfies the system of equations given by $\langle s \rangle^{\perp} = \{g_i \in A \mid g_i \cdot s \equiv 0 \bmod 2\}$, where $g_i \in A$. Let x be a non-zero solution, then $x \cdot s \equiv 0 \bmod 2$ and $x \in \langle s \rangle^{\perp}$ is orthogonal to s. Any element in x can be expressed as a linear combination of the elements $g_1, g_2, \ldots, g_l$ as

$$x = \sum_{i=1}^{l} c_i g_i,$$

where $c_i \in \mathbb{Z}/2\mathbb{Z}$. We then have

$$\begin{aligned} x \cdot s &= \left(\sum_{i=1}^{l} c_i g_i \right) \cdot s \\ &= \sum_{i=1}^{l} c_i \left(g_i \cdot s \right) \\ &\equiv 0 \bmod 2. \end{aligned}$$

Thus, x satisfies the equation if and only if s does, but since s must be a solution, then any other solution must be orthogonal to it, which is a contradiction.

Therefore, s is the unique non-zero solution to system of equations defined.

∎

Problem 2
List all of the numbers $1 \leq x \leq 100$ such that Shor's factoring algorithm actually needs to use a quantum computer in order to find a factor.

Proof. Needing a quantum computer to find a factor of a number N is essentially when we do not have an efficient classical algorithm to find a factor of N. The two classically efficient algorithms for finding a factor of a number N are if N is even or if the number N is some power of a unique prime, *i.e.* $N = p^r$, like 2^6, 7^3, etc. Thus, removing all even numbers, primes, and powers of primes (and 1 trivially) from the list, we are left with

$$\{15, 21, 33, 35, 39, 45, 51, 55, 57, 63, 65, 69, 75, 77, 85, 87, 91, 93, 95, 99\}.$$

∎

Problem A4.17
(Reduction of order-finding to factoring) We have seen that an efficient order-finding algorithm allows us to factor efficiently. Show that an efficient factoring algorithm would allow us to efficiently find the order modulo N of any x co-prime to N.

Proof. Suppose x and N are coprime with $N = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}$. By the Chinese Remainder Theorem, we can identify $\mathbb{Z}/N\mathbb{Z}$ with a sum of cyclic groups of prime power order. Our goal is to find the smallest r such that

$$x^r \equiv 1 \pmod{N}.$$

Suppose we have an efficient factoring algorithm and let p_1, p_2, $\ldots$, p_n be the prime factors of N as above. Then, by Euler's theorem

$$x^{\phi(N)} \equiv 1 \pmod{N},$$

where $\phi(N)$ is the Euler totient function which returns the number of positive integers up to N that are relatively prime with N, and is given by

$$\phi(N) = N \prod_{p|N} \left(1 - \frac{1}{p}\right),$$

where the product is over the distinct prime numbers dividing N. Notice that if N is prime, then every number less than N is clearly relatively prime with N, and thus $\phi(N) = N - 1$. Additionally, the Euler totient function is multiplicative, so if $gcd(m, n) = 1$, then $\phi(mn) = \phi(m)\phi(n)$.
Then,

$$\begin{aligned}
\phi(N) &= \phi\left(p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}\right) \\
&= \phi\left(p_1^{a_1}\right) \phi\left(p_2^{a_2}\right) \cdots \phi\left(p_n^{a_n}\right) \\
&= p_1^{a_1 - 1}(p_1 - 1) p_2^{a_2 - 1}(p_2 - 1) \cdots p_n^{a_n - 1}(p_n - 1) \\
&= \prod_i p_i^{a_i - 1}(p_i - 1) \\
&= \prod_i \phi\left(p_i^{a_i}\right).
\end{aligned}$$

In particular, if the order of x is r, then r must divide $\phi\left(p_i^{a_i}\right)$. Since we have an efficient factoring algorithm, then all we have to do is find a factorization of $p_i - 1$.
Suppose $p_i - 1$ is a product of prime powers $q_j^{b_j}$, then

$$\phi\left(p_i^{a_i}\right) = p_i^{a_i - 1}(p_i - 1) = p_i^{a_i - 1} \prod_j q_j^{b_j}.$$

Iterating through all the divisors of $\phi\left(p_i^{a_i}\right)$, we find the smallest r such that $x^r \equiv 1 \pmod{p_i^{a_i}}$. This last part can be done efficiently since the powers a_i and b_j are relatively smaller than both x and N.

■

Problem 5.13
Prove (5.44). (*Hint:* $\sum_{s=0}^{r-1} \exp(-2\pi isk/r) = r\delta_{k0}$.) In fact, prove that

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \mathrm{e}^{2\pi isk/r} \left|u_s\right\rangle = \left|x^k \bmod N\right\rangle.$$

Proof. Starting with the left hand side of Equation (5.44), we have

$$\begin{aligned}
\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \left|u_s\right\rangle &= \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \left[\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \mathrm{e}^{\frac{2\pi isk}{r}} \left|x^k \bmod N\right\rangle\right] \\
&= \frac{1}{r} \sum_{s=0}^{r-1} \sum_{k=0}^{r-1} \mathrm{e}^{\frac{-2\pi isk}{r}} \left|x^k \bmod N\right\rangle \\
&= \frac{1}{r} \sum_{k=0}^{r-1} \sum_{s=0}^{r-1} \mathrm{e}^{\frac{-2\pi isk}{r}} \left|x^k \bmod N\right\rangle \\
&= \frac{1}{r} \sum_{k=0}^{r-1} r\delta_{k0} \left|x^k \bmod N\right\rangle \\
&= \sum_{k=0}^{r-1} \delta_{k0} \left|x^k \bmod N\right\rangle \\
&= |1\rangle.
\end{aligned}$$

Additionally, we have

$$\begin{aligned}
\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1} \mathrm{e}^{\frac{2\pi i s k}{r}} \left|u_s\right\rangle &= \frac{1}{\sqrt{r}}\sum_{s=0}^{r-1} \mathrm{e}^{\frac{2\pi i s k}{r}} \left[\frac{1}{\sqrt{r}}\sum_{k=0}^{r-1} \mathrm{e}^{\frac{2\pi i s k}{r}} \left|x^k \bmod N\right\rangle\right] \\
&= \frac{1}{r}\sum_{s=0}^{r-1}\sum_{k'=0}^{r-1} \mathrm{e}^{\frac{2\pi i s(k-k')}{r}} \left|x^{k'} \bmod N\right\rangle \\
&= \frac{1}{r}\sum_{k'=0}^{r-1}\sum_{s=0}^{r-1} \mathrm{e}^{\frac{2\pi i s(k-k')}{r}} \left|x^{k'} \bmod N\right\rangle \\
&= \frac{1}{r}\sum_{k'=0}^{r-1} r\delta_{kk'} \left|x^{k'} \bmod N\right\rangle \\
&= \sum_{k'=0}^{r-1} \delta_{kk'} \left|x^{k'} \bmod N\right\rangle \\
&= \left|x^k \bmod N\right\rangle .
\end{aligned}$$

■

Problem 5.16
For all $x \geq 2$ prove that $\int_x^{x+1} 1/y^2 \, \mathrm{d}y \geq 2/3x^2$. Show that

$$\sum_q \frac{1}{q^2} \leq \frac{3}{2}\int_2^\infty \frac{1}{y^2}\,\mathrm{d}y = \frac{3}{4}$$

and thus that (5.58) holds.

Proof. We have

$$\begin{aligned}
\int_x^{x+1} \frac{1}{y^2}\,\mathrm{d}y &= -\frac{1}{y}\bigg|_x^{x+1} \\
&= -\frac{1}{x+1} + \frac{1}{x} \\
&= \frac{1}{x(x+1)}.
\end{aligned}$$

Consider $\frac{1}{x(x+1)} - \frac{2}{3x^2} = \frac{x-1}{3x^2(x+1)}$. For values of $x \geq 2$, the right hand side is always positive, which means that

$$\frac{1}{x(x+1)} = \int_x^{x+1} \frac{1}{y^2}\,\mathrm{d}y \geq \frac{2}{3x^2}.$$

Now, we have

$$\begin{aligned}
\frac{3}{2}\int_2^\infty \frac{1}{y^2}\,\mathrm{d}y &= \frac{3}{2}\sum_{q=2}^\infty \int_q^{q+1} \frac{1}{y^2}\,\mathrm{d}y \\
&\geq \frac{3}{2}\sum_{q=2}^\infty \frac{2}{3q^2} \\
&= \sum_{q=2}^\infty \frac{1}{q^2}.
\end{aligned}$$

On the other hand,

$$\begin{aligned}\frac{3}{2}\int_2^{\infty}\frac{1}{y^2}\,\mathrm{d}y &= \frac{3}{2}\left.-\frac{1}{y}\right|_2^{\infty}\\ &= \frac{3}{2}\left(-\frac{1}{\infty}+\frac{1}{2}\right)\\ &= \frac{3}{4}.\end{aligned}$$

Thus,

$$\sum_q \frac{1}{q^2} \leq \frac{3}{2}\int_2^{\infty}\frac{1}{y^2}\,\mathrm{d}y = \frac{3}{4}.$$

Finally, from Equation (5.58), we have

$$1-\sum_{q=2}^{\infty}\frac{1}{q^2} \geq 1-\frac{3}{4} = \frac{1}{4}.$$

Thus, Equation (5.58) holds. ∎

Problem 5.17
Suppose N is L bits long. The aim of this exercise is to find an efficient classical algorithm to determine whether $N = a^b$ for some integers $a \geq 1$ and $b \geq 2$. This may be done as follows:

(a) Show that b, if it exists, satisfies $b \leq L$.

(b) Show that it takes at most $O\left(L^2\right)$ operations to compute $\log_2(N)$, $x = y/b$ for $b \leq L$, and the two integers u_1 and u_2 nearest to 2^x.

(c) Show that it takes at most $O\left(L^2\right)$ operations to compute u_1^b and u_2^b (use repeated squaring) and check to see if either is equal to N.

(d) Combine the previous results to give an $O\left(L^3\right)$ operation algorithm to determine whether $N = a^b$ for integers a and b.

Proof. (a) We have $N = a^b$. Taking the logarithm on both sides, we get $L = b\log(a)$.

- If $a = 1$, then $L = 1$ and $b = 0$.
- If $a \geq 2$, then $\log(a) \geq 1$ and b is a positive integer with $b \leq L$.

(b) To calculate two estimates of $x = \log N/b$, we need $O(1)$ operations to find y, $O(L^2)$ operations to compute $x = y/b$ for a specific b, and $O(1)$ operations to calculate 2^x and find the nearest integers u_1 and u_2.

(c) When taking the square of a number, we roughly multiply the number of digits by two. Considering the $\log_2(b)$ loops, $a \times a$ takes

- $O\left(L^2\right)$ in the first iteration.
- $O\left((2L)^2\right)$ in the second iteration.
- $O\left((4L)^2\right)$ in the third iteration.
- $O\left((2^{k-1}L)^2\right)$ in the kth iteration.

Assuming the number of iterations k is relatively small compared to the number of digits L of N, then we need $O(L^3)$ operations to compute u_1^b and u_2^b using repeated squaring and to check to see if either is equal to N.

(d) We need to do parts (b) and (c) L times, requiring a total of $O(L^3)$ operations.

∎