Introduction to Shor's Algorithm

as taught by Seth Lloyd, notes by Aaron Vontell

(Dated: November 10, 2016)

These notes include an introduction to Shor's algorithm as taught by Seth Lloyd in 8.370 Quantum Computation at MIT during Fall 2016. A more in-depth coverage of the material can be found at `https://en.wikipedia.org/wiki/Shors_algorithm`

I. INTRODUCTION

Prime factorization is a famous and important problem in the fields of mathematics and computer science. The problem is defined as follows: *Given N, which is equal to the product of two primes p and q, find p and q.*

Classically, the best known algorithm to find p and q given N is the number sieve algorithm, which as a run time of about $O(n^{1/3})$. These notes cover the basics of Shor's algorithm, which uses a quantum computer to complete prime factorization in $O((\log N)^2(\log\log N)(\log\log\log N))$

A. Factoring reduction to discrete log problem

It is recognized that the problem of prime factorization can be reduced to the discrete logarithm problem, which is defined as follows: *Given some X, find r such that*

$$x^r \equiv 1 \bmod N$$

or similarly,

$$x^r = bN + 1$$

for some integer b.

B. Algorithm Steps

1. Overall idea

The first step is to pick an integer X (at random) and find the greatest common divisor of X and N.

If we are lucky enough to have picked an X that divides N, then we are done! This means that X is p or q.

However, this is usually not the case. If not, we need to find r with a quantum computer (covered in the next section). Once we find r, if r is odd, then we pick another X and we try these steps again. Otherwise, r being even implies that we have the following:

$$(x^{r/2}+1)(x^{r/2}-1) = bN$$

Note that the first two terms are size $O(N)$, while the right hand side has size $O(N^2)$. With this result, we now know that one of $(x^{r/2}+1)$,$(x^{r/2}-1)$ is p, and the other is q, which we can compute since we now have X and r.

How do we find the greatest common divisors of N, $x^{r/2}+1$, and $x^{r/2}-1$? We use Euclid's algorithm, which is very fast.

2. Finding r

We will now see how to compute r using a quantum computer. First, we pick an integer n such that $N^2 \leq 2^n \leq 2N^2$. Next, using a quantum computer, we construct the following state:

$$\frac{1}{2^{n/2}} \sum_{k=00\ldots0}^{11\ldots1} |k\rangle \otimes |x^k \bmod N\rangle$$

You will notice that this requires computing $x^k \bmod N$. It turns out that we can compute this relatively fast using modular exponentiation and repeated squaring. For instance, if x^k is equal to x^{2^l}, then this takes time on the order of $O(n^3)$. What's even more interesting is that this is actually the limiting part of the algorithm; this is the step that takes the most time.

HW Problem 8.2: Can pq divide only $x^{r/2}+1$ or $x^{r/2}-1$? If not, why not? *This is related to the discussion in class where we may have that $N(N-1) = bN$.*

HW Problem 8.3: Show that modular exponentiation takes time $O(n^3)$. *It is unclear whether he meant for all states, or for just one.*

We have now constructed the state above, but we can note something interesting and useful; the function $x^k \bmod N$ is periodic with period r! Due to the handy quantum fourier transform, we can find the period and in turn find the (smallest possible) value r, something that we would not be able to do classically. This is because of the following:

$$x^{k+r} \equiv x^k x^r \equiv x^k \equiv x^{k+2r} \equiv \ldots \equiv x^{k+lr} \bmod N$$

3. Performing the QFT

We now perform the quantum fourier transform, but on the **first** (i.e. k) register. This results in the following:

$$\frac{1}{2^n}\sum_{j,k=0}^{2^n-1} e^{2\pi ijk/2^n}|j\rangle \otimes |x^k \bmod N\rangle$$

4. Measuring the first register

We now measure the first register. When we have that jlr is close to a multiple of 2^n, we get positive interference. This means that we know $jlr \approx 2^n$ with high probability.

When we measure the first register, we get a value j such that $jr/2^n$ is close to an integer. Rearranging, for some s we have that

$$\frac{j}{2^n} \approx \frac{s}{r}$$

5. The sneaky trick

Given $\frac{j}{2^n}$, we want to approximate it. From above, we have $\frac{j}{2^n} \approx \frac{s}{r}$, and we can find r $(+s)$ using the method of continued fractions. For example, the continued fraction for $\frac{23}{9}$ is

$$\frac{23}{9} = 2+\frac{5}{9} = 2+\frac{1}{\frac{9}{5}} = 2+\frac{1}{1+\frac{4}{5}} = 2+\frac{1}{1+\frac{1}{\frac{5}{4}}} = 2+\frac{1}{1+\frac{1}{1+\frac{1}{4}}}$$

The trick? This method converges very fast, and allows us to calculate r in an efficient manner.

HW Problem 4: Construct the continued fraction expansions of π, e, and the binary number 0.101010. *Calculate pi and e to 6 digits of accuracy.*

6. Conclusion

By doing the steps above, we have found r with high probability. *However, note that calculating this probability is very difficult.* Now that we have found r with good probability, we can also calculate p and q with high probability through the equation in section 1.

HW Problem 5: (Robustness of cont. fraction expansion) Compare the continued fraction expansion of π to the expansion of $\pi + 0.000001$ (decimal)