Shor’s Algorithm
A Simple Implementation with Q#

DUAN Yuxuan, YAN Ge, ZHU Lizhen

August 8, 2018

Contents

1 Introduction

If computers that you build are quantum,
Then spies everywhere will all want 'em.
Our codes will all fail,
And they'll read our email,
Till we get crypto that's quantum, and daunt 'em.
- Jennifer and Peter Shor

Shor's Algorithm, named after mathematician Peter Shor, is a quantum algorithm for integer factorization formulated in 1994. Informally, it solves the following problem: given an integer N, find its prime factors.

Integer factorization is believed to be a hard problem. The most efficient classical factorization algorithm known as *general number field sieve* achieves an asymptotic runtime exponential in $d^{1/3}$, where d is the number of digits of the integer N we are about to factorize. In contrast, Shor's Algorithm has runtime polynomial in d.

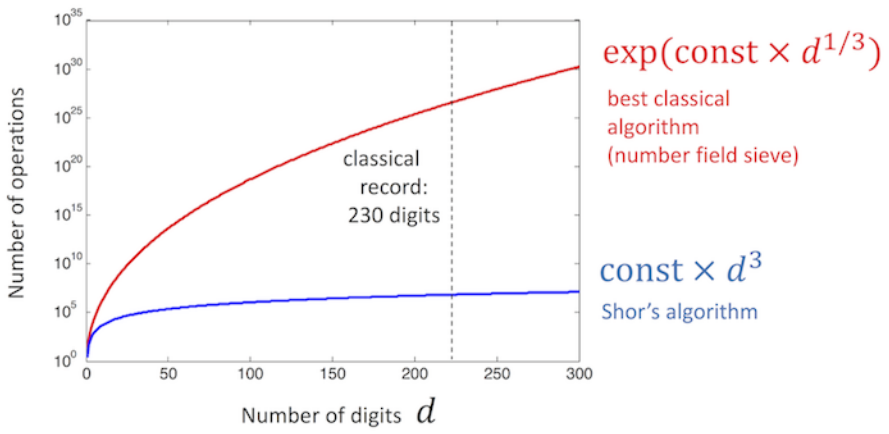


Figure 1: classical vs quantum factorization algorithm

As the quotation at the beginning of this section says, if quantum computers are successfully built, Shor's Algorithm makes it possible to hack *RSA*, a prevalent encryption algorithm based on large integer factorization, which will inevitably cause a revolution on cryptography.

So, we are going to simulate this subtle algorithm with *Microsoft Quantum Development Kit* and its programming language *Q#*.

2 Design & Analysis

2.1 Outline

Our project includes the following modules:

- A classical framework of Shor's Algorithm
- A quantum framework of *Quantum Order Finding Algorithm*
- A *Phase Estimation* procedure.
- An *Inverse Quantum Fourier Transform* procedure
- A test program, embedded in the project.

Our project can:

- Factorize an integer which can be written in the product of two distinct primes, within the range $[2, 128]$.
- Show the process of Shor's Algorithm by output.

2.2 Shor's Algorithm

The main idea of Shor's Algorithm is illustrated in Figure 2 below.

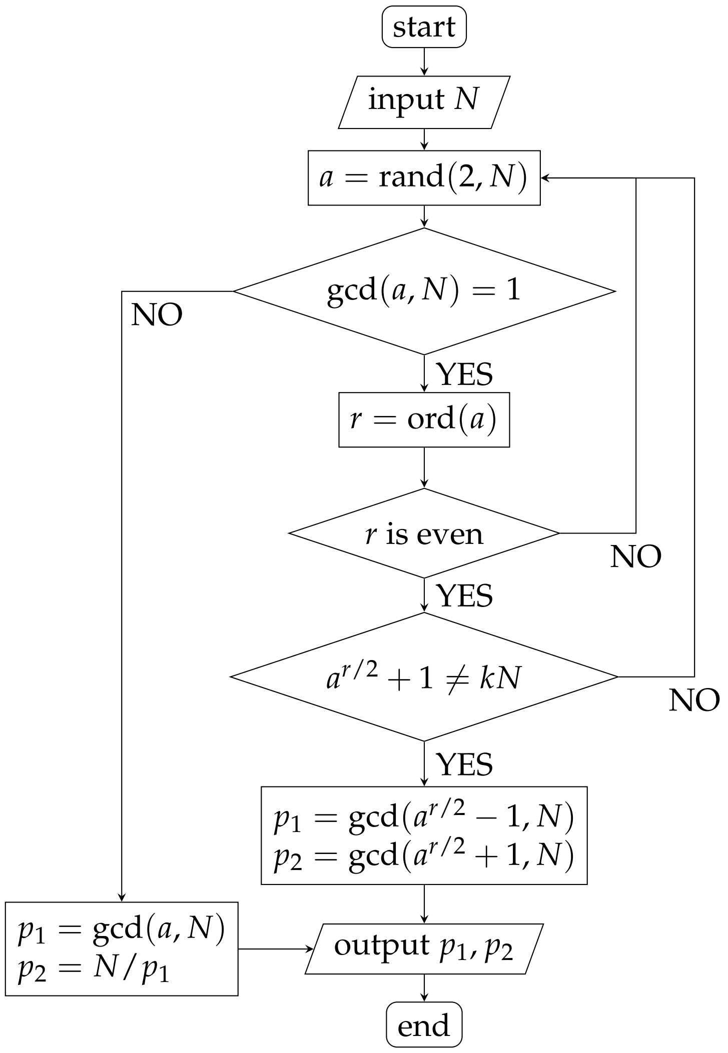


Figure 2: Shor's Algorithm

N is the integer we are factorizing which satisfies $N = p_1 \cdot p_2$, where p_1 and p_2 are two distinct primes.

Here is a brief proof of Shor's Algorithm:

1. a is the integer we randomly choose within $[2, N]$, if $x = \gcd(a, N) \neq 1$, then x must be a prime factor of N. So we can get

$$\begin{aligned} p_1 &= \gcd(a, N) \\ p_2 &= N/p_1 \end{aligned}$$

2. Otherwise, we compute r, the order of a using Quantum Order Finding Algorithm. If r is even, we just choose another a, till r is an even number.

3. Next, if $a^{r/2} + 1 \equiv 0 \pmod{N}$, we choose another a as well.

4. For $r = \text{ord}(a)$, then

$$a^{r/2} - 1 \not\equiv 0 \pmod{N}$$

5. Now we have

$$a^r - 1 = (a^{r/2} + 1)(a^{r/2} - 1) \equiv 0 \pmod{N}$$

so

$$\begin{aligned} (a^{r/2} + 1) &\equiv 0 \pmod{p_1} \\ (a^{r/2} - 1) &\equiv 0 \pmod{p_2} \end{aligned}$$

6. Finally, we can get p_1 and p_2 by

$$\begin{aligned} p_1 &= \gcd(a^{r/2} + 1, N) \\ p_2 &= \gcd(a^{r/2} - 1, N) \end{aligned}$$

2.3 Quantum Order Finding Algorithm

So far, there is no efficient classical order finding algorithm. However, using some quantum technique, we can solve this problem much faster.

Let's assume that we are finding the order of x with regard to N, and the order is r.

We can build an operator U_x which satisfies

$$U_x|y\rangle = |xy \bmod N\rangle$$

and $y \in \{0,1\}^L$ is the binary representation of a decimal integer. (As we take $L = 7$ in our project, we can therefore factorize integers less than 128)

The eigenstates of U_x can be written as

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

Although we cannot know a single eigenstate as $|u_s\rangle$ depends on r, we find that

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

in which

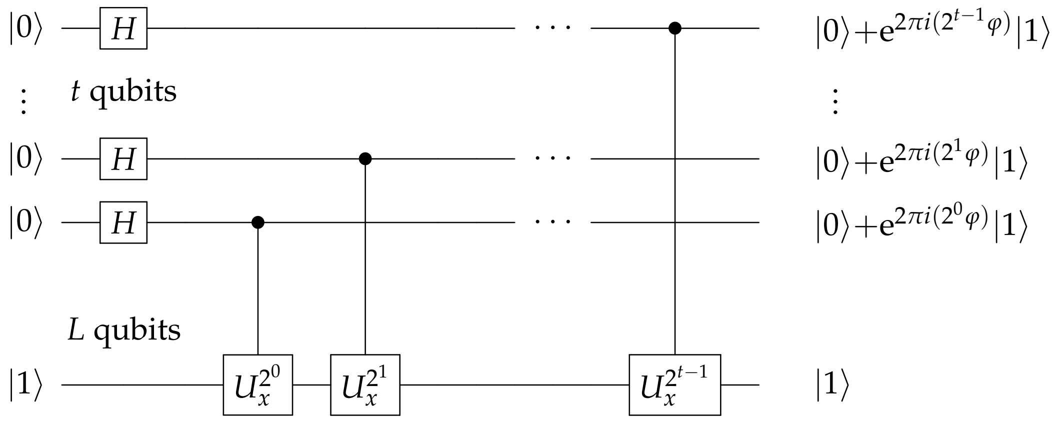
$$\sum_{s=0}^{r-1}\exp\left[\frac{-2\pi isk}{r}\right] = \begin{cases} r & (k=0) \\ 0 & (\text{otherwise}) \end{cases}$$

so

$$\begin{aligned}\frac{1}{r}\sum_{k=0}^{r-1}\left(\sum_{s=0}^{r-1}\exp\left[\frac{-2\pi isk}{r}\right]\right)|x^k \bmod N\rangle &= \frac{1}{r}\cdot r|x^0 \bmod N\rangle \\ &= |1\rangle\end{aligned}$$

Therefore, if we give $|1\rangle$ to the Phase Estimation procedure as the second register, the outcome we measure will be the phase $\varphi = s/r$ of one of the eigenvalues, which we may get r as a probable order using an algorithm called *The Continued Fraction Expansion*.

2.4 Phase Estimation



In this procedure we use the circuit above to estimate the phase φ. We have already gotten the U_x as Chapter 2.3 defined.

The quantum phase estimation procedure uses two registers. The first register contains t qubits initially in state $|0\rangle$, where t is set with regard to the precision we demand (In our project we take $t = 5$). The second register begins in the state $|1\rangle$, which is L qubits.

The second stage of the phase estimation procedure is to apply the inverse quantum Fourier transform on the first register of the first stage. This stage can be done in $\Theta(t^2)$ steps.

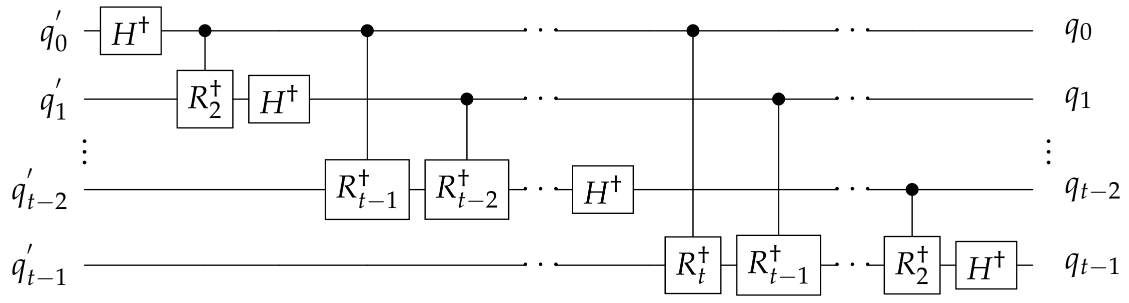
The final stage of the phase estimation algorithm is to read out the state of the first register by doing a measurement in the computational basis. When applying the controlled-$U_x^{2^j}$ to the qubits, operate U_x on $|1\rangle$ for 2^j times is very time consuming. From the definition of the controlled-U_x gate we know the following fact:

$$\begin{aligned} U_x^j|y\rangle &= |x^j y \bmod N\rangle \\ &= U_{x^j}|y\rangle \end{aligned}$$

Therefore, we can calculate the 2 to the k^{th} power in advance instead of applying the controlled-U gate 2^k times.

Summarizing, the phase estimation algorithm allows one to estimate the phase φ, which indicates some information of the order r we are computing.

2.5 Inverse Quantum Fourier Transform



$$R_x = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^x} \end{bmatrix}$$

This is Inverse Quantum Fourier Transform. The order of operation is reverse to the Quantum Fourier transform. $q'_{n-1}, \ldots q'_0$ are the outcome of the first stage of Phase Estimation. First we need to swap the qubits (the i^{th} qubit with the $(t-i-1)^{th}$) in order to let the input of Inverse Quantum Fourier Transform be in the right order.

After the swap, the input will be:

$$\begin{aligned} q'_0 &= |0\rangle + e^{2\pi i(2^{t-1}\varphi)}|1\rangle \\ q'_1 &= |0\rangle + e^{2\pi i(2^{t-2}\varphi)}|1\rangle \\ &\vdots \\ q'_{t-1} &= |0\rangle + e^{2\pi i(2^{0}\varphi)}|1\rangle \end{aligned}$$

After the transform, we can measure the first register and get the outcome of Phase Estimation:

$$\begin{aligned} q_0 &= |\varphi_0\rangle \\ q_1 &= |\varphi_1\rangle \\ &\vdots \\ q_{t-1} &= |\varphi_{t-1}\rangle \end{aligned}$$

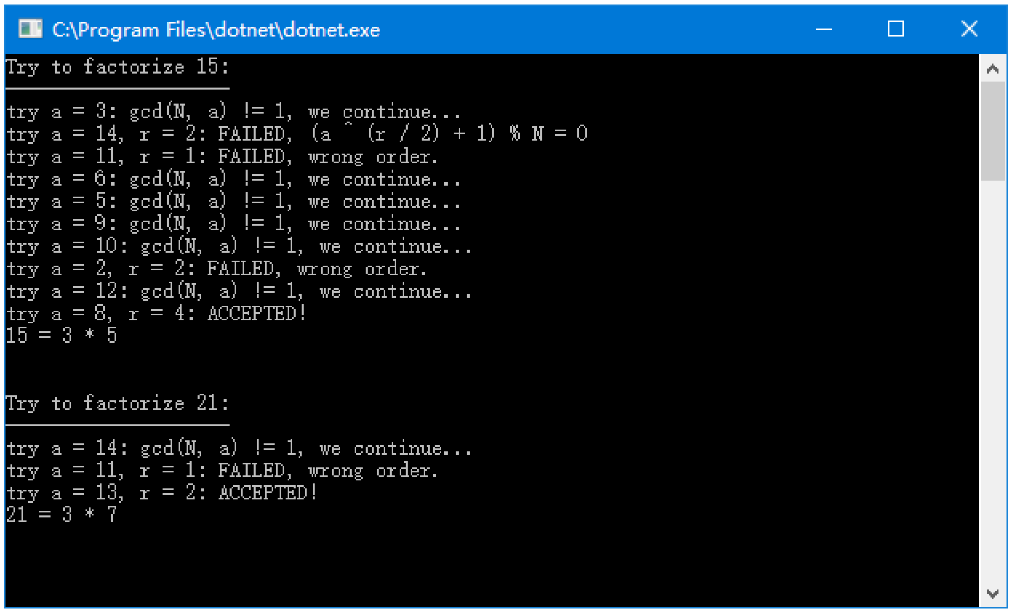
where we $\varphi = 0.\varphi_0\varphi_1 \ldots \varphi_{t-1}$, the decimal representation of φ.

Inverse Quantum Fourier Transform uses $O(n^2)$ operations.

3 Test

3.1 Basic Test

We have fixed the bugs in our program and have tried to run it with $N = 15$ and $N = 21$, which are the only two integers in range $(2, 31)$ as the coursework required.



```
Try to factorize 15:
--------------------
try a = 3: gcd(N, a) != 1, we continue...
try a = 14, r = 2: FAILED, (a ^ (r / 2) + 1) % N = 0
try a = 11, r = 1: FAILED, wrong order.
try a = 6: gcd(N, a) != 1, we continue...
try a = 5: gcd(N, a) != 1, we continue...
try a = 9: gcd(N, a) != 1, we continue...
try a = 10: gcd(N, a) != 1, we continue...
try a = 2, r = 2: FAILED, wrong order.
try a = 12: gcd(N, a) != 1, we continue...
try a = 8, r = 4: ACCEPTED!
15 = 3 * 5


Try to factorize 21:
--------------------
try a = 14: gcd(N, a) != 1, we continue...
try a = 11, r = 1: FAILED, wrong order.
try a = 13, r = 2: ACCEPTED!
21 = 3 * 7
```

Figure 3: the output while factorizing 15 and 21

As the outcome showed, factorization was completed successfully!

3.2 Further Trial

As our program support the factorization of integer greater than 31 and less than 128. We have tried some larger input N. However, the simulation of Shor's Algorithm is quite slow. So we only tested $N = 33, 35, 39$, and all the three trials succeeded.

4 Reference

- *Quantum Computation and Quantum Information*, Michael A. Nielsen, Isaac L. Chuang
- *Shor's Algorithm - IBM Q Experience Documentation*, IBM Research and the IBM QX team
- *Microsoft Quantum Development Kit Preview*, Keith Heston et al.

Figure 4: Members (left to right): YAN Ge, DUAN Yuxuan, ZHU Lizhen