Shor's Algorithm

Pranish Bhagat

August 2021

Abstract

Although any integer number has a unique decomposition into a product of primes, finding the prime factors is believed to be a hard problem. In fact, the security of our online transactions rests on the assumption that factoring integers with a thousand or more digits is practically impossible. This assumption was challenged in 1995 when Peter Shor proposed a polynomial-time quantum algorithm for the factoring problem. Shor's algorithm is arguably the most dramatic example of how the paradigm of quantum computing changed our perception of which problems should be considered tractable. In this project we briefly summarize some basic facts about factoring, highlight the main ingredients of Shor's algorithm, and illustrate how it works by using a toy factoring problem.

1 Introduction

Shor's algorithm is a polynomial-time quantum factoring algorithm, and one of the first quantum algorithms that provided an exponential speedup for a useful, real-life problem. The problem it solves is to factor integers, and classically, the best known algorithms take super-polynomial time. Quantum factorization, on the other hand, is much faster. It requires both classical pre-processing and a quantum circuit, but both parts take only polynomial time, so the overall algorithm is polynomial as well. The full algorithm is as follows:

1.1 Shor's Algorithm

Let us say the number we are trying to factor is N, and it is an n-bit number. For simplicity's sake, the algorithm will only find a single factor of N (or determine if N is prime), and then, we can recursively run it at most n times to find all prime factors. The factor-finding algorithm goes like so:

1. Check if N is prime. This can be done through any number of primality tests in polynomial time.

2. Choose a random number a, $1 < a < N - 1$. If $gcd(a, N) > 1$, return $gcd(a, N)$. The gcd can be calculated in polynomial time with Euclid's algorithm.

3. Use the quantum order-finding algorithm to find the order of a mod N. The order of a is the smallest positive integer r such that $a^r \equiv 1(modN)$.

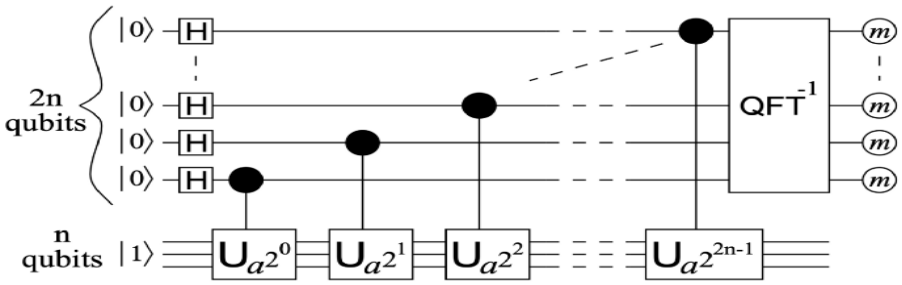


Figure 1: Order Finding Algorithm

4. Because $a^r \equiv 1(modN), a^{r/2} - 1$ should have a common factor with N. Of course, that wouldn't be the case if r is odd, or if this common factor is N itself. If so, we would have to go back to step 2. Otherwise, we can return $gcd(a^{r/2} - 1, N)$.

The meat of this algorithm is the order-finding circuit, which classically would have taken exponential time, but is only polynomial on a quantum computer. The circuit above is one way to do order-finding, but it has a few disadvantages. The first is that it requires 3n qubits in total, which is a problem as qubits are a scarce resource. Another is that the U gates in the diagram all have to be hard-coded: they would only work for a specific a and N, and they are not generalizable.

2 Methodology

2.1 Better Order-Finding

It requires only $2n + 3$ qubits to find the order of an n-bit number, and it has the advantage of being entirely general. There are a few components that allow it to be this way. The first is:

2.1.1 The Adder Gate

This gate adds a to the phase of the input. To use this gate to actually add two numbers, what you would need to do is encode the first number b in binary, then apply a Quantum Fourier Transform (QFT) to it. The adder gate solely consists of R_z gates, which are a rotation by a given angle about the Pauli Z axis.Using this adder gate, we can construct:

2.1.2 The Modular Adder Gate

This gate adds a to b and takes the result modulo N. It uses both the adder gate, and the inverse of the adder gate, which would be a subtract gate. In the diagram below, the adder gate is represented with a black bar on the right, and the subtract gate is represented with a black bar on the left. With this gate, we can make:

2.1.3 The Multiplier Gate

This gate acts on two registers, $|x> |b>$, and takes them to $|x> |b + ax(modN)>$. It works by considering each of the bits of x separately, and adding a times that bit to b. The modular addition gates require $n + 2$ qubits, and the x register is n qubits, so in total, the multiplier gate has $2n + 2$ qubits. With this multiplier gate, we can finally construct:

2.1.4 The General Unitary Operator

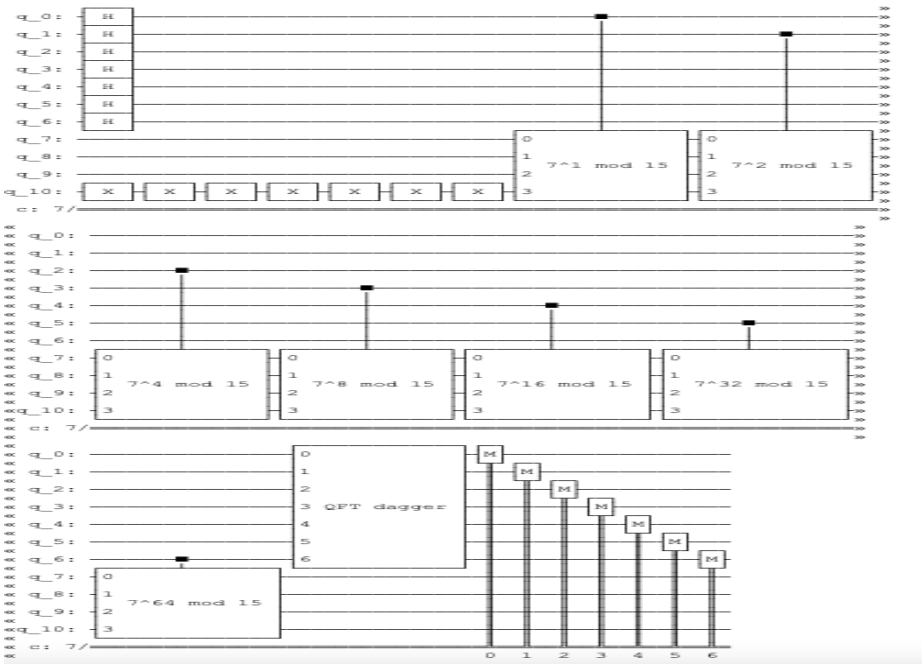
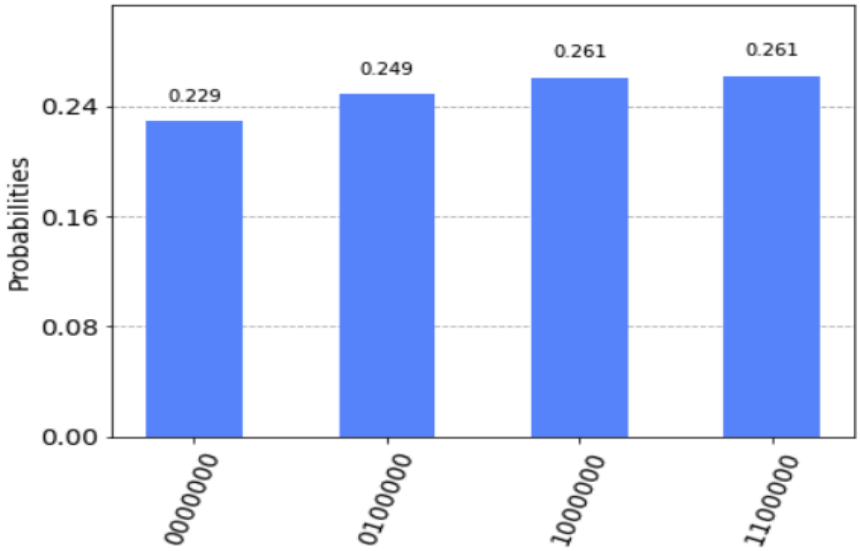


Figure 2: Circuit obtained from the experiment

This is the operator $U(a)$ which is used in the order finding circuit. Its behavior is to take an input $|x>$ and multiply it by a to get $|ax \bmod N>$. We are almost there with the multiplier gate: if $|b>=|0>$, then we have $|ax \bmod N>$ in the second register. The above circuit is all we need to get the full gate.

3 Results

I ran the $2n+3$ circuit with $N = 15$ and $a = 2$. So, it was attempting to find the order of 2 mod 15. Because $N = 15$, that means $n = log(N) = 4$. So, the circuit will use $2 * 4 + 3 = 11$ qubits, and will output $2 * 4 = 8$ bits of measurement. Here is the histogram of results for running the order finding circuit on IBM's QASM Simulator with 2048 shots: The 8 bits this circuit outputs is not r, the



order, but rather $(2^{2n)*s/r}$, where s could be any number from 0 to $r1$. From the histogram, we see that the output of the circuit could be one of four states: $0, 64, 128$, or 192. If we divide these by 2^{2n}, then we can see what r would be for each of these states: The four output states correspond to the fractions $\frac{0}{1}$,

	Phase	Fraction	Guess for r
0	0.25	1/4	4
1	0.50	1/2	2
2	0.00	0/1	1
3	0.75	3/4	4

Figure 4: Possible values of r

$\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$. So, there are three possible guesses we could make for r. The

following histogram shows the probability we make each of those guesses. The actual order of 2 mod 15 is 4. So, that means if we run the order finding circuit, there is a 0.5 chance of getting the right answer. This is actually perfectly fine, because we can run the circuit numerous times, and we only have to get the right answer once. In fact, no matter the a and N, there is always at least a 0.5 chance of the circuit outputting the correct r, barring any errors.

4 Conclusion

Ultimately, I believe that this circuit has more use in its generality than in its small amount of qubits. Due to its length, even in the simplest of cases, quantum error correction would have to be applied. The smallest distance-3 codes require 7 qubits, so the total number of qubits would be multiplied by at least that number.

However, the good part about this method is that with enough qubits and with good error correction, this implementation could find the order of any number modulo any other number, albeit in $O(n^4)$ time.

The length of the circuit could certainly be more optimized. A constant factor could be removed simply by ordering the qubits in the right way, as CNOTs only work between adjacent qubits. Also, if we look at the histogram of results, we can see that only the first two measurements were really used: the rest were all zero. Therefore, we could shorten the circuit by only computing those first two values (first $log(n)$ values in the general case).

However, this implementation is a good first step, and could one day be the key to much larger factorization. Modern encryption schemes are dependent on the fact that factoring is a hard problem, but with this algorithm, breaking RSA is only a matter of time.