Quantum Computing on Simon's Periodicity Algorithm

Joseph Chang, Dec. 16th, 2020, PHYS 411 project

Abstract

Simon's periodicity algorithm is a quantum computing algorithm that suppresses the time efficiency of classical computing algorithms. The difference between a quantum computer and a classical computer is that a quantum computer uses qubits to harness the properties of superposition and entanglement to achieve quantum parallelism. This paper first walks through the essential background knowledge of Simon's algorithm or other quantum algorithms. The background knowledge includes time complexity, classical gates, quantum gates, and some mathematical notations. The problem of a function f that contains a hidden string C is defined. Then, a classical solution with an example and a quantum solution with an example is shown. This paper does not go through the time complexity analysis of algorithms. Proofs of analysis on time complexity can be found in Loceff's textbook, chapter 18. Simon's periodicity algorithm does not have a real-world application, but it demonstrates the power of quantum computing and provides techniques used in other applicable quantum algorithms.

Introduction

In theoretical computer science, a famous set is called the NP-complete set. NP-complete is a set of problems difficult for a computer to solve. These problems took exponential time to solve; even if those problems have solutions, they can't be solved in time if the input domain is large. Simon's algorithm does not have obvious applications in the real world. However, it shows the power of quantum computing because it exponentially speeds up the solution. The complexity of the problem is determined by the string length n of the domain. A classical approach must cover more than half of the domain to guarantee a solution. Therefore, the time bound of the classical solution is an exponential time $T(n) = O(2^n + 1)$. However, the upper time bound of the quantum solution is polynomial $T(n) = O(n^3)$. The technique used in this algorithm can also be applied to algorithms with real-world applications, such as Shor's algorithm, which solves the factor of large numbers in polynomial time. The table below shows the order of time complexity and why it is crucial to have polynomial time solutions to problems.

	n	$n\,log(n)$	n^3	2^n	$n!$
$n = 100$	$< 1s$	$< 1s$	$< 1s$	$10^{17} yrs$	$very\ long$
$n = 10000$	$< 1s$	$< 1s$	$12\ days$	$very\ long$	$very\ long$

The time length in the table is an approximation of time cost to solve the problem with a given time complexity. Time complexity gives a time scale to run an algorithm in the worst-case scenario. The algorithm is generalized to run input with different lengths n. Generally, a longer input size means a longer time to run. The table above shows how time would grow for a time complexity expression according to an input size n.

Background Knowledge

Context

This section does not contain the main context of the paper. However, this section covers the essential tools and mathematical expressions for the main context of this paper, starting with section: the problem (page 6). If the reader is familiar with quantum computing, this section can be skipped.

Logic gate

In classical computers, logic gates are built using transistors. Logic gates are the basic computation unit of logical calculation. Widely used gates include NOT, AND, OR, and XOR. The NOT gate takes a 1-bit input and produces a 1-bit output, and the rest takes a 2-bit input and produces a 1-bit output.

For example,
The symbol of NOT gate is ~.
The NOT gate operator as

$$\sim |0\rangle = |1\rangle$$
$$\sim |1\rangle = |0\rangle$$

A complete table of these gates are shown below:

x	y	$NOT\ x\ (\sim x)$	$x\ AND\ y\ (x \wedge y)$	$x\ OR\ y\ (x \vee y)$	$x\ XOR\ y\ (x \oplus y)$
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

The table above shows the classical behavior of logic gates and classical bits. Logic gates and bits can be quantized. The operator can be represented as a matrix, and the inputs and outputs bits can be written as tensor product states. Plus, a bit does not need to be 0 or 1; it can be a superposition of 0 and 1. However, there are some constraints to quantum logic gates. A linear optical device must be unitary; this infers any computation using quantum logic gates are reversible.

Set {}

The bracket notation means a set of elements. For example, a set of fruits can be {apple, banana, cherry}. The superscript lies outside of the bracket and labels the dimension of the set. For example, $\{0,\ 1\}^2$ means the same as $\{00,\ 01,\ 10,\ 11\}$ and $\{0,\ 1\}^3$ means the same as $\{000,\ 001,\ 010,\ 011,\ 100,\ 101,\ 110,\ 111\}$.

XOR ⊕

A mathematical notation ⊕ is used in this paper. This is the symbol of the bitwise XOR gate. The definition for this function is the following:

$$\oplus : \{0,1\}^N \times \{0,1\}^N = \{0,1\}^N$$
$$X \oplus Y = x_1 \oplus y_1, x_2 \oplus y_2, ..., x_N \oplus y_N$$

Where X and Y are strings of N bits.

$$X = x_1, x_2, ..., x_N = x_1 x_2 ... x_N$$
$$Y = y_1, y_2, ..., y_N = y_1 y_2 ... y_N$$

⊕ for N = 1 is a standard XOR gate

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

$\langle \quad , \quad \rangle$

Another mathematical notation is $\langle \quad , \quad \rangle$. This function **is not** an inner product of vectors. The definition for this function is the following:

$$\langle \quad , \quad \rangle : \{0,1\}^N \times \{0,1\}^N = \{0,1\}$$
$$\langle X, Y \rangle = \langle x_1 x_2 ... x_N, y_1 y_2 ... y_N \rangle$$
$$= (x_1 \wedge y_1) \oplus (x_2 \wedge y_2) \oplus ... \oplus (x_N \wedge y_N)$$

Where ∧ is a standard AND gate

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

cNOT

For example, quantum gate cNOT (Controlled NOT gate) is similar to the classical XOR gates; however, it is both one-to-one and onto. It is a bijection function. Bijection means the function is reversible such that an element of range can always be mapped to a unique element in the domain.

$$cNOT(\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle) = \alpha|00\rangle + \beta|01\rangle + \delta|10\rangle + \gamma|11\rangle$$

Input	Output
0 0	0 0
0 1	0 1
1 0	1 1
1 1	1 0

The first column of output remains the value of the first column of input. The second column of output is the XOR result from both columns of the input.

The matrix form of cNOT is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

With a similar technique; all classical gates can be built into quantum as a reversible gate. However, the true power of quantum computing comes from harnessing the property of superpositions and quantum parallelism. Quantum parallelism means we don't apply a logic gate on 1 or 2 qubits, as in classical computing, we apply a quantum logic gate on N qubits.

Hadamard

An important quantum gate is called the Hadamard gate. Hadamard gate turns a discrete state of n qubits into a superposition with equal probability across all bases. It is a generalized even beam splitter. In 1 qubit, Hadamard gate is an even beam splitter represented as

$$H^{\otimes 1} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Where

$$H^{\otimes 1}|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

For N qubits, Hadamard acts like:

$$H^{\otimes N}|0\rangle = \frac{\sum_{x\in\{0,1\}^N} |x\rangle}{\sqrt{2^N}}$$

In matrix form, the matrix can be generalized using XOR $\oplus$ and AND $\wedge$ operator:

$$H^{\otimes 1} = \frac{1}{\sqrt{2}}\begin{pmatrix}(-1)^{0\wedge 0} & (-1)^{0\wedge 1}\\ (-1)^{1\wedge 0} & (-1)^{1\wedge 1}\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$

To get Hadamard N= 2, we can simply do tensor product on Hadamard of N=1

$H^{\otimes 2} = H^{\otimes 1} \otimes H^{\otimes 1}$

The result is the following, notice that the XOR gate $\oplus$ comes from multiplication between (−1)'s power e.g. $(-1)^{0\wedge 1} * (-1)^{1\wedge 0} = (-1)^{0\wedge 1\oplus 1\wedge 0}$

$$H^{\otimes 2} = \frac{1}{\sqrt{4}}\begin{pmatrix}(-1)^{0\wedge 0\oplus 0\wedge 0} & (-1)^{0\wedge 0\oplus 0\wedge 1} & (-1)^{0\wedge 1\oplus 0\wedge 0} & (-1)^{0\wedge 1\oplus 0\wedge 1}\\ (-1)^{0\wedge 0\oplus 1\wedge 0} & (-1)^{0\wedge 0\oplus 1\wedge 1} & (-1)^{0\wedge 1\oplus 1\wedge 0} & (-1)^{0\wedge 1\oplus 1\wedge 1}\\ (-1)^{1\wedge 0\oplus 0\wedge 0} & (-1)^{1\wedge 0\oplus 0\wedge 1} & (-1)^{1\wedge 1\oplus 0\wedge 0} & (-1)^{1\wedge 1\oplus 0\wedge 1}\\ (-1)^{1\wedge 0\oplus 1\wedge 0} & (-1)^{1\wedge 0\oplus 1\wedge 1} & (-1)^{1\wedge 1\oplus 1\wedge 0} & (-1)^{1\wedge 1\oplus 1\wedge 1}\end{pmatrix}$$

For N=n case, we can order the row, and column base on tensor product

$$row = \{0,1\}^n$$
$$column = \{0,1\}^n$$
$$matrix\ index = (-1)^{\langle row,\ column\rangle}$$

The problem

Assume there exists a function f with a pattern C. The function operates by taking an input of N bits and producing an input of N bits. Each bit is either 0 or 1. The function has a pattern C such that the output of the function from two inputs, X and Y strings, are the same if and only if one input string is the same as another input string XOR pattern C. The function f acts as a black box to us. We can collect outputs by sending inputs to the function. The goal is to find the pattern C in string form.

$$f : \{0,1\}^N \rightarrow \{0,1\}^N$$
$$f(X) = f(Y) \quad \text{iff} \quad X = C \oplus Y$$
$$X \in \{0,1\}^N \quad Y \in \{0,1\}^N \quad C \in \{0,1\}^N$$

Due to the pattern C, the function f is either two-to-one or one-to-one. If pattern C is $\{0\}^N$, then f is one-to-one because $X = \{0\}^N \oplus Y$ infers $X = Y$. If pattern C is not $\{0\}^N$, then f is two-to-one because $X = C \oplus Y$ infers $X \neq Y$ when $C \neq \{0\}^N$ and $X = C \oplus Y$ also infers $Y = C \oplus X$. However, for the sake of the quantum algorithm, we always assume the function f is two-to-one such that C is never $\{0\}^N$.

Example for N=2

$$f(|00\rangle) = |01\rangle$$
$$f(|01\rangle) = |01\rangle$$
$$f(|10\rangle) = |11\rangle$$
$$f(|11\rangle) = |11\rangle$$

For this example, $C = |01\rangle$ because:

$$f(|00\rangle) = f(|01\rangle) \rightarrow |00\rangle = C \oplus |01\rangle$$
$$C \oplus |00\rangle = C \oplus C \oplus |01\rangle$$
$$C = |01\rangle$$

Classical solution

To solve the problem using a classical approach, we need to provide an input set that covers more than half of the domains for string length N, and we need to compare all the output of given inputs. If no two inputs denote the same output, then C is, $\{0\}^N$ and the function f is one-to-one. If two inputs denote the same output, we can solve C by the following procedure:

We found two inputs denote the same output

$$f(X) = f(Y) \quad X, Y \in \{0,1\}^N$$

We know that if two outputs are the same, they must have a relation with pattern C.

$$X = C \oplus Y$$

We can solve C because X and Y are the known input we fed to the function.

$$X \oplus Y = (C \oplus Y) \oplus Y$$
$$= C \oplus (Y \oplus Y) = C \oplus \{0\}^N = C$$

The domain size of the function is N^2. In the worst-case scenario, we need to cover just more than half of the domain, so we need to test $N^2/2 + 1$ times.

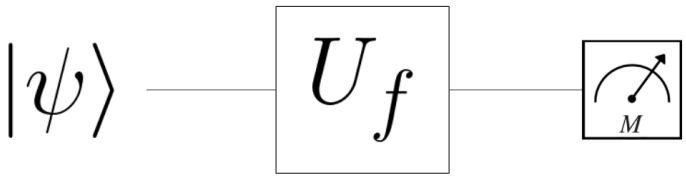
Using the example given in the problem:

$$f(|00\rangle) = |01\rangle$$
$$f(|01\rangle) = |01\rangle$$
$$f(|10\rangle) = |11\rangle$$
$$f(|11\rangle) = |11\rangle$$

The procedure is the following:

1. N=2, so we need to test $N^2/2 + 1 = 3$ times.
2. Send $|00\rangle$ to f and get $|01\rangle$.
3. Send $|01\rangle$ to f and get $|01\rangle$.
4. Send $|10\rangle$ to f and get $|11\rangle$.
5. Using sorting and matching algorithms, we discover inputs, $|00\rangle$ and $|01\rangle$ both denote an output of $|01\rangle$.
6. Solve $C = |00\rangle \oplus |01\rangle = |01\rangle$.

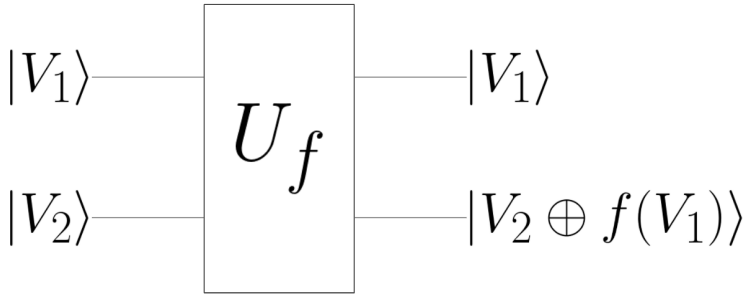
The quantum circuit for this classical solution is represented as



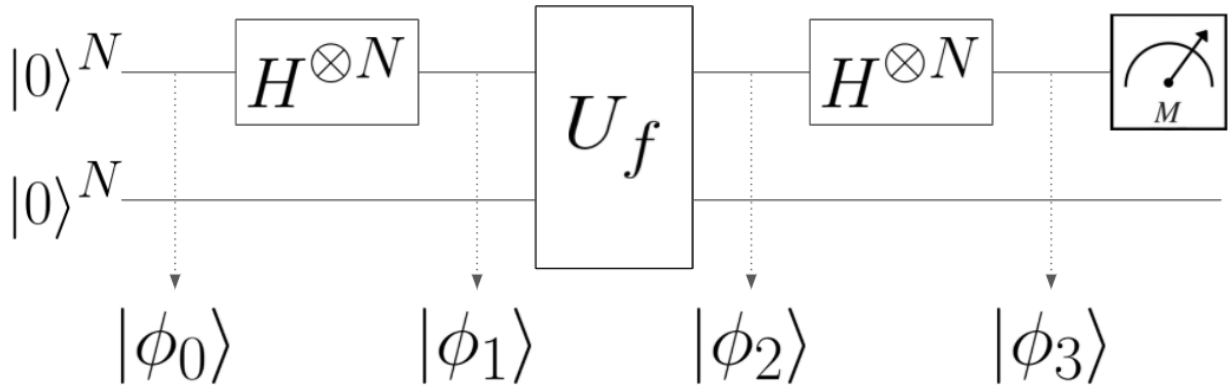
We set $|\psi\rangle$ to be $N^2/2 + 1$ different combinations of $\{0, 1\}^N$.

Quantum computing solution

The first thing we want to do is to generalize the function f such that it is reversible and can be built physically as a linear optical device. It takes two input states $|V_1\rangle$ and $|V_2\rangle$, where $|V_1\rangle, |V_2\rangle \in \{0,1\}^N$. The gate f keeps the top as $|V_1\rangle$, but change the bottom as $|V_2 \oplus f(V_1)\rangle$.



Now, the quantum circuit for Simon's algorithm uses both the reversible function gate f and the Hadamard gate that was introduced in the background section. The circuit for Simon's algorithm is as follows:



At the beginning, we have an initial state $|\phi_0\rangle = |00...0\rangle \otimes |00...0\rangle$

Then, we apply the Hadamard gate to the top state using the operator $H^{\otimes N} \otimes \mathbb{I}$ that leads to the state

$$
\begin{aligned}
|\phi_1\rangle &= (H^{\otimes N} \otimes \mathbb{I})|\phi_0\rangle \\
&= (H^{\otimes N} \otimes \mathbb{I})(|00...0\rangle \otimes |00...0\rangle) \\
&= \frac{\sum_{x \in \{0,1\}^N} |x\rangle}{\sqrt{2^N}} \otimes |00...0\rangle \\
&= \frac{\sum_{x \in \{0,1\}^N} |x, 0\rangle}{\sqrt{2^N}}
\end{aligned}
$$

After that, we apply function gate U_f

$$
\begin{aligned}
|\phi_2\rangle &= U_f|\phi_1\rangle \\
&= U_f \frac{\sum_{x\in\{0,1\}^N} |x, 0\rangle}{\sqrt{2^N}} \\
&= \frac{\sum_{x\in\{0,1\}^N} |x, 0 \oplus f(x)\rangle}{\sqrt{2^N}} \\
&= \frac{\sum_{x\in\{0,1\}^N} |x, f(x)\rangle}{\sqrt{2^N}}
\end{aligned}
$$

Then, we apply Hadamard gate again

$$
\begin{aligned}
|\phi_3\rangle &= (H^{\otimes N} \otimes \mathbb{I})|\phi_2\rangle \\
&= (H^{\otimes N} \otimes \mathbb{I}) \frac{\sum_{x\in\{0,1\}^N} |x, f(x)\rangle}{\sqrt{2^N}} \\
&= \frac{\sum_{x\in\{0,1\}^N} \sum_{z\in\{0,1\}^N} (-1)^{\langle z,x\rangle}|z\rangle}{\sqrt{2^N}\times\sqrt{2^N}} \otimes \sum_{x\in\{0,1\}^N} |f(x)\rangle \\
&= \frac{\sum_{x\in\{0,1\}^N} \sum_{z\in\{0,1\}^N} (-1)^{\langle z,x\rangle}|z, f(x)\rangle}{2^N}
\end{aligned}
$$

Due to the property of pattern string C, $|z, f(x)\rangle = |z, f(x \oplus C)\rangle$.
The coefficient for $|z, f(x \oplus C)\rangle$ is

$$
\begin{aligned}
&\frac{(-1)^{\langle z,x\rangle} + (-1)^{\langle z,x\oplus C\rangle}}{2} \\
&= \frac{(-1)^{\langle z,x\rangle} + (-1)^{\langle z,x\rangle\oplus\langle z,C\rangle}}{2} \\
&= \frac{(-1)^{\langle z,x\rangle} + (-1)^{\langle z,x\rangle}(-1)^{\langle z,C\rangle}}{2}
\end{aligned}
$$

If $\langle z, C\rangle = 1$, the coefficient is 0.

If $\langle z, C\rangle = 0$, the coefficient is ± 1.
As a result, when measuring $|\varphi_3\rangle$, $\langle z, C\rangle = 0$ gives equal non-zero probability, and $\langle z, C\rangle = 1$ gives zero probability on the corresponding basis vector. From this set of equations, we can solve string C. According to Loceff's textbook section 18.12, a classical approach to solving the equation is called Gaussian Elimination. The GE algorithm cost $O(n^3)$ is the upper bound of Simon's algorithm. To obtain the quantum state of the upper output, we need to measure many times based on the size of N. However, the time complexity to perform measurements is way below the cubic polynomial cost of solving the system using Gaussian Elimination.

Simon's algorithm Example

Again, take the example introduced earlier:

$$f(|00\rangle) = |01\rangle$$
$$f(|01\rangle) = |01\rangle$$
$$f(|10\rangle) = |11\rangle$$
$$f(|11\rangle) = |11\rangle$$

N=2

$$|\phi_0\rangle = |00\rangle \otimes |00\rangle.$$

$$|\phi_1\rangle = \frac{\sum_{x\in\{0,1\}^2} |x, 0\rangle}{\sqrt{2^2}}$$
$$= \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \otimes |00\rangle$$

$$|\phi_2\rangle = \frac{\sum_{x\in\{0,1\}^2} |x, f(x)\rangle}{2}$$
$$= \frac{1}{2}(|00\rangle \otimes |01\rangle + |01\rangle \otimes |01\rangle + |10\rangle \otimes |11\rangle + |11\rangle \otimes |11\rangle)$$

$$|\phi_3\rangle = \frac{\sum_{x\in\{0,1\}^2} \sum_{z\in\{0,1\}^2} (-1)^{\langle z,x\rangle} |z\rangle \otimes |f(x)\rangle}{4}$$

$$= \frac{1}{4} \sum_{x \in \{0,1\}^2} \{(-1)^{\langle 00,x \rangle} |00\rangle \otimes |f(x)\rangle + (-1)^{\langle 01,x \rangle} |01\rangle \otimes |f(x)\rangle$$
$$+(-1)^{\langle 10,x \rangle} |10\rangle \otimes |f(x)\rangle + (-1)^{\langle 11,x \rangle} |11\rangle \otimes |f(x)\rangle\}$$

$$= \frac{1}{4}\{+|00\rangle \otimes |01\rangle + |01\rangle \otimes |01\rangle + |10\rangle \otimes |01\rangle + |11\rangle \otimes |01\rangle$$
$$+|00\rangle \otimes |01\rangle - |01\rangle \otimes |01\rangle + |10\rangle \otimes |01\rangle - |11\rangle \otimes |01\rangle$$
$$+|00\rangle \otimes |11\rangle + |01\rangle \otimes |11\rangle - |10\rangle \otimes |11\rangle - |11\rangle \otimes |11\rangle$$
$$+|00\rangle \otimes |11\rangle - |01\rangle \otimes |11\rangle - |10\rangle \otimes |11\rangle + |11\rangle \otimes |11\rangle\}$$

$$= \frac{1}{2}\{|00\rangle \otimes |01\rangle + |00\rangle \otimes |11\rangle + |10\rangle \otimes |01\rangle - |10\rangle \otimes |11\rangle\}$$

$$= \frac{1}{2}\{|00\rangle \otimes (|01\rangle + |11\rangle) + |10\rangle \otimes (|01\rangle - |11\rangle)\}$$

By measuring the upper output, we get an even distribution between state $|00\rangle$ and $|10\rangle$; this gives 2 equations for N=2

$$\langle 00, C \rangle = 0$$
$$\langle 10, C \rangle = 0$$

These two equations can solve for C, from the second equation:

$$\langle 10, C \rangle = \langle 01, c_1 c_2 \rangle = (1 \wedge c_1) \oplus (0 \wedge c_2) = 0$$
$$(0 \wedge c_2) = 0 \quad \Rightarrow \quad 0 \oplus (1 \wedge c_1) = (1 \wedge c_1) = 0$$
$$c_1 = 0$$

From the first equation:

$$\langle 00, C \rangle = \langle 00, 0 c_2 \rangle = (0 \wedge 0) \oplus (0 \wedge c_2) = 0$$
$$\Rightarrow \quad 0 \oplus (0 \wedge c_2) = (0 \wedge c_2) = 0$$
$$c_2 = 1 \quad or \quad 0$$

We are certain that the function f is not one-to-one, so $C \neq |00\rangle$; therefore, $C = |01\rangle$.

Conclusion

The goal for this topic is to solve a problem that finds the hidden pattern string C from a black box function f. We don't know how the function works, but we can get the output value of the function by entering an input value. This paper shows a classical solution and a quantum solution. Examples of function f, classical solution walkthrough on function f, and Simon's algorithm walkthrough on function f are also present. According to Loceff's time analysis on both the classical algorithm and Simon's quantum algorithm, the time complexity of the classical algorithm is exponential, which is an unsolvable problem if the size of N is large, and the time complexity of the quantum algorithm is a polynomial of $O(n^3)$. This problem cannot be projected onto a real-world application, but it shows the potential of exponential speed-up done by quantum computers. It also has some crucial techniques that construct quantum algorithms of real-world applications, including Shor's algorithm. In theoretical computer science, the exponential speed-up is critical because it turns a set of unsolvable problems into a solvable problem set. Many applicable quantum algorithms have been published, but all of them rely on a stable quantum computer that can entangle a large number of qubits. The greatest challenge of quantum computing is that it is difficult to build a stable quantum computer that runs many qubits.

Reference

Loceff, M. (2015). *A Course in Quantum Computing* (Vol. 1). Los Altos Hills, CA: 2015 Michael Loceff.

Yanofsky, N. S., & Mannucci, M. A. (2018). *Quantum Computing for Computer Scientists*. New York, NY: Cambridge university press.