Zhian Jia (贾治安)

Quantum computation theory in a nutshell

August 1, 2025

Contents

Part I
Mathematical and physical preliminaries

Part II
Quantum computation theory

1 Classical computation

> A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine.
>
> *Alan Turing*

This chapter presents the essential foundations of classical computation theory, serving as necessary preparation for the study of quantum computation. We review the core concepts of computability and complexity in the classical setting, including formal models of computation—most notably the Turing machine—and the basic framework of computational complexity theory (such as the classes P and NP). By establishing these classical notions with precision, we lay the groundwork required to appreciate both the power and the limitations of classical computers, thereby providing a solid conceptual baseline from which the principles of quantum computing can be introduced and contrasted in subsequent chapters.

To reason precisely about computability and complexity, we require formal mathematical models of the computational process. Several equivalent models exist for classical computation; the best known are the Turing machine and the Boolean circuit (or logic-gate) model. Although Turing machines play a central historical and conceptual role in computability theory, the Boolean circuit model is particularly well-suited for generalization to quantum computing. For this reason, we will primarily adopt the circuit model in what follows.

§ 1.1 Basics of classical computation

Classical computers operate on strings of classical bits, each of which is definitively either 0 or 1. At its core, a classical computation amounts to evaluating a deterministic function: given an input bit string of length n, the computer produces an output bit string of length m, uniquely determined by that input.

Mathematically, the device computes a function

$$f : \{0,1\}^n \to \{0,1\}^m, \tag{1.1}$$

for some fixed integers n and m. (For convenience, we will write $\mathbb{B} := \{0,1\}$ when helpful.)

Since an m-bit output may be viewed as m separate single-bit outputs, any such function decomposes into m individual *Boolean functions*, each mapping to a single bit:

$$f : \{0,1\}^n \to \{0,1\}. \tag{1.2}$$

In this (somewhat informal) sense, Boolean functions serve as the basic building blocks of classical computation.

A natural generalization allows both the input and output to be bit strings of arbitrary finite length. We denote the set of all finite binary strings by

$$\mathbb{B}^* = \bigcup_{n=0}^{\infty} \mathbb{B}^n = \bigcup_{n=0}^{\infty} \{0,1\}^n, \tag{1.3}$$

where $\{0,1\}^0 = \{\varepsilon\}$ consists solely of the empty string ε. In this more flexible setting, a computation is a function

$$f : \mathbb{B}^* \to \mathbb{B}^*. \tag{1.4}$$

As we shall see, not every such function is computable.

In this section, we introduce the standard model of computation—the Turing machine—and review several foundational notions from classical computation theory. A central insight of the field is that all reasonable models of computation, including Turing machines and (appropriately size- and depth-bounded) Boolean circuits, are computationally equivalent: they define precisely the same class of computable functions, differing only in their resource requirements such as time, space, or circuit size.

1.1.1 Turing machine

Although Turing machines will not be used directly in these notes, it is helpful to record a precise definition for completeness. The Turing machine remains the foundational model of classical computation. At a physical level, every modern computer is effectively a universal Turing machine—not a Boolean circuit—so the Turing machine provides the correct conceptual basis for understanding computation.

A Turing machine may be pictured as a meticulous clerk working along an unbounded strip of paper, carrying out a calculation step by step according to a fixed and finite list of instructions.

To begin, we introduce a finite alphabet. For example, if we wish to describe addition, the alphabet may consist of the digits $0, 1, \ldots, 9$ together with the symbol $+$. A computation starts by writing a number, followed by the symbol $+$, and then another number. The clerk then inspects the symbol beneath the pencil, consults the rule associated with the current internal state and that symbol, and performs the prescribed action: replace the symbol with another, move one square to the left or right, and update the internal state.

We now idealize the unbounded strip of paper into an *infinite tape*. The tape is divided into discrete cells, each holding a single symbol—often just a $0, 1$ or a blank in the simplest formulations. The clerk can view only one cell at a time. At each step, they execute the appropriate rule with complete mechanical obedience, without judgment or improvisation. A typical instruction might be: "If you see a '1', erase it and move left." The entire computation unfolds through such deterministic, local operations.

In this way, the Turing machine abstracts the familiar act of performing a calculation on paper. It distills computation to its essential components: a finite instruction set, a symbol under inspection, and a potentially infinite tape serving as memory. Despite its austere structure, this model captures precisely the class of functions that can be computed by any physically realizable device.

Definition 1.1 (Turing Machine). A *Turing machine* is a mathematical model of computation consisting of an infinite tape extending to the right, a read-write head that can move along the tape, and a finite control unit. Formally, a Turing machine is a tuple $\mathfrak{M} = (Q, \Sigma, \delta)$, where the components are defined as follows:

- $Q = \{q_0, q_1, \ldots, q_n\}$ is a finite, nonempty set of *states*, where $q_0 \in Q$ is the *initial state*; there is a subset $H \subseteq Q$ called the set of *accepting* (or *halting*) states;
- $\Sigma = \{0, 1, \ldots, d-1\}$ (with $d \geq 2$) is the finite *input alphabet* (not containing the special symbols below);
- Γ is the finite *tape alphabet*, with $\Sigma \cup \{\triangleright, \square\} = \Gamma$ and $\triangleright, \square \notin \Sigma$. The symbol $\square$ is called the *blank symbol* (the only symbol that may appear infinitely often on the tape), and $\triangleright$ is the *left-end marker*;
- $\delta\colon (Q \setminus H) \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$ is a (possibly partial) function called the *transition function*. For a non-accepting state q, current tape symbol a, and $\delta(q, a) = (q', a', D)$, the machine in state q reading a:
  - overwrites a with a',
  - moves its head one cell *left* if $D = -1$, *right* if $D = +1$, or stays put if $D = 0$,
  - enters state q'.

  The head is not permitted to move left of the $\triangleright$ symbol.

See Figure 1.1 for an illustration of a Turing machine. The tape is infinite to the right and initially contains the input string from $\Sigma^* = \bigcup_{n\geq 0} \Sigma^n$ (the set of all finite strings over Σ), followed by infinitely many blanks. The leftmost cell contains a special marker $\triangleright$, and the read–write head is positioned on the first symbol of the input (or on $\triangleright$ if the input is empty). We may regard the tape contents as an infinite sequence $s = s_0 s_1 s_2 \cdots$, where each s_i is a symbol of Γ, possibly the blank $\square$.

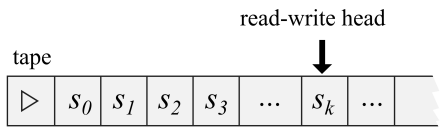


Fig. 1.1 The core components of a Turing machine: a semi-infinite tape (infinite to the right) divided into cells holding symbols from the tape alphabet, and a read–write head that moves left or right, reading the current symbol and writing a new one according to the transition function.

A *configuration* of a Turing machine is a triple $\{s, p, q\}$, where $s = s_0 s_1 s_2 \cdots$ denotes the (infinite) tape contents, $p \in \mathbb{Z}_{\geq 0}$ is the position of the read–write head, and $q \in Q$ is the current internal state. At each step, the Turing machine updates its configuration according to the transition function δ as follows:

- It reads the symbol in the p-th tape cell, say $s_p = a$.
- It evaluates the transition function $\delta(q, a) = (q', a', D)$. If q' belongs to the halting set of states, the machine halts. If (q, a) is not in the domain of the partial function δ, the machine also halts.
- It writes the symbol a' into cell p, and moves the head one step in the direction specified by $D \in \{-1, 0, +1\}$. Thus the new configuration becomes

$$\{\, s_0 s_1 \cdots s_{p-1}\, a'\, s_{p+1} s_{p+2} \cdots \,,\; p + D,\; q' \,\}.$$

A remarkable and foundational insight in computation theory—often referred to as the *Church–Turing thesis* in its informal form, and rigorously substantiated by the equivalence of formal models—is the following:

All reasonable models of classical computation are computationally equivalent.

Note that a Turing machine is, in fact, an abstract model of an algorithm that you may recognize from everyday calculations. However, this model is deliberately primitive and mechanical: it captures the bare essentials of computation rather than the conveniences of high-level programming.

Exercise 1.1. Design a Turing machine that adds two binary numbers.

Definition 1.2 (Computable Function). Let Σ be a finite alphabet and $X \subseteq \Sigma$. A (partial) function $f\colon X^* \dashrightarrow X^*$ is said to be *computable* (or *Turing-computable*) if there exists a Turing machine $\mathfrak{M}$ such that:

- For every input $s \in X^*$ on which $f(s)$ is defined, $\mathfrak{M}$ halts on input s and outputs exactly $\psi_{\mathfrak{M}}(s) = f(s)$.
- For every input $s \in X^*$ on which $f(s)$ is undefined, $\mathfrak{M}$ either halts and explicitly signals "undefined" or runs forever (both conventions appear in the literature).

We then say that $\mathfrak{M}$ *computes* f, and we write $\psi_{\mathfrak{M}} = f$ on the domain of f.

Not every function $f\colon X^* \to X^*$ (or even partial function $f\colon X^* \dashrightarrow X^*$) is computable. In fact, the overwhelming majority are uncomputable. This is an immediate consequence of cardinality: The set of *all* functions $X^* \to X^*$ has cardinality $2^{\aleph_0}$ (the power of the continuum). In contrast, the set of all Turing machines is countable, because each Turing machine can be encoded as a finite string over a fixed finite alphabet, and the set of all finite strings is countable. Since there is no surjection from a countable set onto a set of cardinality $2^{\aleph_0}$, only countably many functions can be Turing-computable.

The most famous example of an uncomputable function is the *halting problem*. Define

$$\mathrm{Halt}\colon X^* \times X^* \dashrightarrow \{0,1\} \tag{1.5}$$

$$(\langle\mathfrak{M}\rangle, x) \mapsto \begin{cases} 1 & \text{if Turing machine } \mathfrak{M} \text{ halts on input } x, \\ 0 & \text{if } \mathfrak{M} \text{ loops forever on } x. \end{cases} \tag{1.6}$$

Alan Turing proved in 1936, using an elegant diagonalization argument, that no Turing machine can compute Halt. This negative result is one of the cornerstones of computability theory and marks the discovery of the first inherently uncomputable problem.

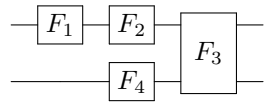
§ 1.2 Classical logic gates and circuit model

More precisely, every function that can be computed by a Turing machine can also be computed by a Boolean (or logic-gate) circuit family of appropriate size and depth, and vice versa. The standard models—including deterministic and nondeterministic Turing machines, random-access machines, lambda calculus, recursive functions, counter machines, and unbounded Boolean circuit families—all recognize exactly the same class of computable functions. Differences between models appear only when we account for resource usage (time, space, or circuit size), giving rise to the rich landscape of computational complexity theory.

Because of this equivalence, we are free to choose the model that is most convenient for the task at hand. While Turing machines remain the historical and conceptual cornerstone of computability, the Boolean circuit model often proves more natural when extending ideas to the quantum setting. Both models

will appear in this chapter, but the circuit perspective will take center stage in later discussions of quantum computation.

Let us now introduce the circuit model for classical computation. A circuit consists of wires and gates, where wires represent classical bit strings and gates represent classical operations. The gates are usually drawn as boxes, and the circuit is typically read from left to right. For example, a two-bit circuit is represented as



(1.7)

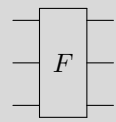
where two wires represent two bits and $F_1, \ldots, F_4$ are gates.

More formally, a gate can be defined as follows.

Definition 1.3. Classical gate An n-input, m-output **classical logic gate** (or simply, gate) is an m-component Boolean function

$$F : \{0,1\}^n \to \{0,1\}^m. \tag{1.8}$$

It can be represented by a box with m input wires and n output wires. For example:



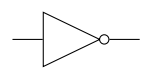
(1.9)

1.2.0.1 Single-bit gate

Notice that there are only 4 single-bit Boolean function $f : \{0,1\} \to \{0,1\}$[1]: (i) the identity function; (ii) $f \equiv 0$; (iii) $f \equiv 1$; (iv) the bit-flipping operation, which is the most interesting case. Thus it has its own name, `NOT` gate:

$$\text{NOT}: \quad a \mapsto \bar{a} = 1 \oplus a. \tag{1.10}$$

Diagrammatically:



(1.11)

[1] Since there are 2 possible outputs for each input 0 and 1, there are a total of $2 \times 2 = 4$ possibilities.

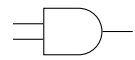
1.2.0.2 Two-bit input and single-bit output gate

For gates with a two-bit input and one-bit output, there are more possible gates. The number of such gates equals the number of Boolean functions $f : \mathbb{B}^2 \to \mathbb{B}$ (simple calculation gives 2^4). Among these gates, there are two basic gates that play a crucial role in the study of logic and computation: the `AND` and `OR` gates:

- The operation of the `AND` gate is as follows:

$$\text{AND}: \quad (a, b) \mapsto a \wedge b = ab. \tag{1.12}$$

  Diagrammatically:

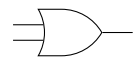


(1.13)

  When both input bits are 1, it outputs 1; otherwise, it outputs 0.
- The operation of the `OR` gate is as follows:

$$\text{OR}: \quad (a, b) \mapsto a \vee b = a \oplus b \oplus ab. \tag{1.14}$$

  Diagrammatically:



(1.15)

  When both input bits are 0, it outputs 0; otherwise, it outputs 1.

Notice that `AND`, `OR` and `NOT` are independent of each other, meaning that no one of them can be obtained by composing the other two (try to prove this yourself!). The composition of `AND` with `NOT` is called `NAND`; similarly, the composition of `OR` with `NOT` is called `NOR`.

Other important two-input gates are `XOR` and `XNOR`:

- `XOR`: when the two input values differ it outputs 1, otherwise 0:

$$\text{XOR}: \quad (a, b) \mapsto a \oplus b. \tag{1.16}$$

- `XNOR`: when the two input values are equal it outputs 1, otherwise 0:

$$\text{XNOR}: \quad (a, b) \mapsto \overline{a \oplus b} = 1 \oplus a \oplus b. \tag{1.17}$$

1.2.0.3 Classical cloning operation

Besides the above 1-input 1-output or 2-input 1-output gates, there are two other gates that we will use later: the cloning gate and the swap gate.

The swap gate:

$$\text{SWAP}: \quad (a, b) \mapsto (b, a). \tag{1.18}$$

The cloning (fan-out) gate (classical):

$$\text{CLONE}: \quad a \mapsto (a, \ldots, a). \tag{1.19}$$

A complicated swap operation can be decomposed into the composition of two-bit swaps. The cloning gate in classical computation is usually taken for granted; in the quantum case the no-cloning theorem forbids copying arbitrary unknown states.

Theorem 1.1 (No-cloning theorem). *In the quantum setting, there is no cloning gate that can copy an arbitrary unknown state perfectly.*

Proof. There are two common proofs: one based on unitarity of the cloning map and another based on linearity. Here we present the standard unitarity argument.

Assume there exists a unitary U such that for any state $|\psi\rangle$,

$$U\big(|\psi\rangle \otimes |0\rangle\big) = |\psi\rangle \otimes |\psi\rangle. \tag{1.20}$$

For two (generally non-orthogonal) states $|\psi_1\rangle$ and $|\psi_2\rangle$ we would have

$$U\big(|\psi_1\rangle \otimes |0\rangle\big) = |\psi_1\rangle \otimes |\psi_1\rangle, \tag{1.21}$$

$$U\big(|\psi_2\rangle \otimes |0\rangle\big) = |\psi_2\rangle \otimes |\psi_2\rangle. \tag{1.22}$$

Taking inner products of the two left-hand sides and using unitarity gives

$$\langle\psi_1|\psi_2\rangle = \langle\psi_1|\psi_2\rangle^2, \tag{1.23}$$

which implies $\langle\psi_1|\psi_2\rangle \in \{0, 1\}$, a contradiction for arbitrary states. Hence such a universal cloning unitary cannot exist.

- `AND` gate: $\text{AND} : (a, b) \mapsto a \wedge b$.
- `OR` gate: $\text{OR} : (a, b) \mapsto a \vee b$.
- `NAND` gate: $\text{NAND} : (a, b) \mapsto 1 \oplus ab$.
- `NOR` gate: $\text{NOR} : (a, b) \mapsto 1 \oplus a \oplus b \oplus ab$.
- `XOR` gate: $\text{XOR} : (a, b) \mapsto a \oplus b$.
- `XNOR` gate: $\text{XNOR} : (a, b) \mapsto 1 \oplus a \oplus b$.
- `SWAP` gate: $\text{SWAP} : (a, b) \mapsto (b, a)$.
- `CLONE` gate: $\text{CLONE} : a \mapsto (a, \ldots, a)$.

1.2.1 Universal gate set

Theorem 1.2 (Universal gate). *The gate set* $\{\mathrm{NOT}, \mathrm{AND}, \mathrm{OR}, \mathrm{CLONE}\}$ *is universal: any Boolean function* $f : \{0,1\}^n \to \{0,1\}^m$ *can be implemented by a circuit composed of these gates.*

Exercise 1.2. Try to prove Theorem 1.2 yourself or consult a standard textbook on classical computation; it is not difficult.

Hint: Examining examples of decomposing a general Boolean function into compositions of these gates will provide you with some intuition.

1.2.2 Binary aritheoremetics

§ 1.3 Classical circuit model of computation

1.3.1 Classical logic gates

By an n-input m-output classical logic gate, we mean an m-component Boolean function

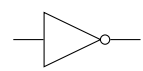
$$f : \mathbb{B}^n \to \mathbb{B}^m. \tag{1.24}$$

The study of these gates is crucial in mathematical logic and computation theory.

Notice that there are only 4 single-bit Boolean function $f : \mathbb{B} \to \mathbb{B}$: (i) the identity function; (ii) $f \equiv 0$; (iii) $f \equiv 1$; (iv) the bit-flipping operation, `NOT`:

$$\mathrm{NOT}: \quad a \mapsto \bar{a} = 1 \oplus a. \tag{1.25}$$

Diagrammatically, it's denoted as



(1.26)

1.3.2 Universal logic gates

Exercise 1.3 (Universal gates). Prove that the following statements about universal gates hold:

- Gate NAND is universal.
- Gate NOR is universal.
- Toffoli gate is universal for reversible classical computation.

§ 1.4 Complexity classes

As every nontrivial computational task requires at least reading the entire input strings, it is natural for us to count the number of basic operations as a function of the input length.

2 Basics of quantum computation

> If you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy.
>
> *Richard Feynman*

The idea of using a quantum–mechanical device to perform computation dates back to the pioneering work of Yuri I. Manin (1980), Paul Benioff (1980), Richard Feynman (1982, 1985), and several others. A major turning point came with the introduction of explicit computational models. In 1985, David Deutsch proposed the quantum Turing machine, and a few years later, in 1989, introduced the quantum circuit model—an equivalent but far more convenient formulation. This circuit model, building on Feynman's original insights, has since become the standard framework for quantum computation.

Our aim here is to outline several basic ingredients of quantum computing, providing a foundation upon which further study can be built.

We begin with a brief review of some standard notions from classical computation. We then describe the framework of quantum computation using the circuit model. After that, a number of representative quantum Algorithms are presented to illustrate how quantum systems can process information and, in certain cases, outperform classical methods. It is worth noting, however, that the translation of classical ideas into quantum Algorithms does not always shed light on the underlying reasons for their effectiveness.

§ 2.1 Quantum computation and quantum circuit model

We are now prepared to formulate a mathematical model of a quantum computer. We extend the classical circuit model of computation to the *quantum circuit model*. Essentially, a quantum computer is a quantum circuit consist-

ing of preparing the input state, applying the quantum circuit operations, and performing measurements to obtain results. Quantum mechanics is inherently probabilistic, so the outcomes appear with corresponding probabilities, and we select the most probable results.

To summarize, we have the following rough mathematical definition of a quantum computer:

Definition 2.1. Quantum computer A quantum computer is a physical realization of the quantum circuit model. To compute a function $f : \mathbb{B}^n \to \mathbb{B}^m$, there are three basic steps:

- Encode the classical information, such as bit strings, into quantum states and prepare the input state.
- Set up the quantum circuit that realizes the function f, and implement the quantum circuit on the input state.
- Perform quantum measurements to read out the results.

Note that, similar to classical computation, there exist several models for quantum computation, such as the quantum Turing machine, one-way quantum computation, magic state computation, and the quantum circuit model. All these models have the same computational power. In these notes, we will use the quantum circuit model for illustration, as it is more intuitive and convenient.

2.1.1 Encoding classical information into quantum states

Classical information is represented by bit strings such as

$$00100110,$$

but each qubit is represented on the Bloch sphere. A natural question arises: how do we encode classical information into quantum states?

The simplest method is *basis encoding*, which replaces the classical bit 0 with $|0\rangle$ and the classical bit 1 with $|1\rangle$:

$$|00100110\rangle.$$

For a more general dataset $\mathcal{X} \subset \{0,1\}^n$ (or $\mathcal{X} \subset \mathbb{R}^n$), we can encode data via a bijective map

$$\vec{x} \mapsto \psi_{\vec{x}}, \quad \forall \vec{x} \in \mathcal{X}, \tag{2.1}$$

where $\psi_{\vec{x}}$ is a pure state[1]. The map should be one-to-one, such that $\psi_{\vec{x}}$ and $\psi_{\vec{y}}$ can be distinguished via quantum operations.

[1] It is also possible to encode classical information into mixed states, but we will not cover that here.

This problem is crucial in quantum machine learning. Below we describe two more encoding approaches.

- **Angle encoding:** For an n-bit string $\vec{x} \in \mathcal{X}$, choose an n-qubit Hilbert space $\mathcal{H} = (\mathbb{C}^2)^{\otimes n}$ and map $\vec{x} = (x_1, x_2, \ldots, x_n)$ to the basis

$$\vec{x} \mapsto |\vec{x}\rangle = \bigotimes_{i=1}^{n} (\cos(x_i)|0\rangle + \sin(x_i)|1\rangle). \tag{2.2}$$

  Since $x_i = 0, 1$, this reduces to $|x_i\rangle$. The formula becomes useful when x_i are real numbers.
- **Amplitude encoding:** Introduce a feature map $\vec{f} : \mathcal{X} \to \mathbb{R}^N$ and encode classical data into an N-dimensional feature Hilbert space:

$$\vec{x} \mapsto |\psi_{\vec{x}}\rangle = \frac{1}{\|\vec{f}(\vec{x})\|_2} \sum_i f_i(\vec{x})|i\rangle, \tag{2.3}$$

  where $\|\vec{f}(\vec{x})\|_2 = \left(\sum_i f_i(\vec{x})^2\right)^{1/2}$ and $i = 1, \ldots, N$. A common choice is $f_i(\vec{x}) = x_i$, also called wavefunction encoding.

2.1.2 Quantum circuit

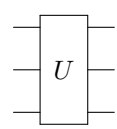
A quantum circuit consists of quantum wires representing quantum states and quantum gates representing unitary operations. Measurements are usually performed at the final step.

The quantum circuit model was proposed by David Deutsch in 1985[2], and its notation appeared in Richard Feynman's paper[3].

Definition 2.2. Quantum gates An n-qubit gate is a unitary operator

$$U : (\mathbb{C}^2)^{\otimes n} \to (\mathbb{C}^2)^{\otimes n},$$

which maps an n-qubit state to another n-qubit state. It is represented as a box with n input and n output wires. For example:



(2.4)

[2] David Deutsch. Quantum theory, the Church–Turing principle, and the universal quantum computer. Proc. R. Soc. Lond. A, 400:97–117, 1985.

[3] Richard Feynman. Simulating physics with computers. Int. J. Theor. Phys., 21(6/7):467–488, 1982.

The basic building blocks of quantum circuits are single-qubit and two-qubit gates. These suffice for universal quantum computation.

2.1.2.1 Single-qubit gates

The Pauli gates are crucial single-qubit gates:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Theorem 2.1 (label=theorem:SU2). *Single-qubit unitary and $SU(2)$ group A general single-qubit gate is a 2×2 unitary matrix*

$$U = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad U^\dagger U = U U^\dagger = I.$$

All single-qubit gates form the unitary group $U(2)$. Any $U \in U(2)$ can be related to $V \in SU(2)$ by a phase factor: $V = e^{-i\alpha} U$ with $\det V = 1$. A general element of $SU(2)$ can be written as

$$U = \begin{pmatrix} a & b \\ -b^* & a^* \end{pmatrix}, \quad a, b \in \mathbb{C}, \quad |a|^2 + |b|^2 = 1, \tag{2.5}$$

and equivalently as a linear combination of Pauli operators:

$$U = tI + ixX + iyY + izZ, \quad t^2 + x^2 + y^2 + z^2 = 1. \tag{2.6}$$

Rotation operators about a direction $\vec{n}$ are

$$R_{\vec{n}}(\theta) = \exp\left(-i\theta \frac{\vec{n} \cdot \vec{\sigma}}{2}\right) = \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2} \vec{n} \cdot \vec{\sigma}. \tag{2.7}$$

Special cases:

$$R_x(\theta) = e^{-i\theta X/2} = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix}, \tag{2.8}$$

$$R_y(\theta) = e^{-i\theta Y/2} = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}, \tag{2.9}$$

$$R_z(\theta) = e^{-i\theta Z/2} = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}. \tag{2.10}$$

Exercise 2.1. (1) Use $\sigma_i \sigma_j = \delta_{ij} I + i \varepsilon_{ijk} \sigma_k$ to show

$$(\vec{a} \cdot \vec{\sigma})(\vec{b} \cdot \vec{\sigma}) = \vec{a} \cdot \vec{b} I + i(\vec{a} \times \vec{b}) \cdot \vec{\sigma}.$$

(2) Deduce $(\vec{n}\cdot\vec{\sigma})^2 = I$ for $|\vec{n}| = 1$ and show

$$\exp\left(-i\theta\frac{\vec{n}\cdot\vec{\sigma}}{2}\right) = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}\vec{n}\cdot\vec{\sigma}.$$

The Hadamard gate:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \quad H|0\rangle = |+\rangle, \quad H|1\rangle = |-\rangle.$$

Phase gate and T-gate:

$$S = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}, \quad T = \begin{pmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{pmatrix}.$$

Theorem 2.2 (Single-qubit unitary decomposition). *An arbitrary single-qubit unitary gate can be written as*

$$U = e^{i\alpha}R_{\vec{n}}(\theta), \quad \textit{or equivalently } U = e^{i\alpha}R_z(\beta)R_y(\gamma)R_z(\zeta),$$

with real parameters $\alpha, \theta, \beta, \gamma, \zeta$ and unit vector $\vec{n}$.

2.1.2.2 Two-qubit gates

The controlled-U gate:

$$C(U) = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U.$$

1. CNOT gate:

$$\mathsf{CNOT} = \quad, \quad |a\rangle \otimes |b\rangle \mapsto |a\rangle \otimes |a \oplus b\rangle.$$

2. Controlled-Z gate:

$$C(Z) = .$$

3. SWAP gate:

$$\mathsf{SWAP} = \sum_{a,b} |a\rangle\langle b| \otimes |b\rangle\langle a|, \quad \mathsf{SWAP}|\psi\rangle \otimes |\varphi\rangle = |\varphi\rangle \otimes |\psi\rangle.$$

2.1.3 Universal quantum gate

Definition 2.3. Universal gate set A set of quantum gates $\mathcal{U} = \{U_1, \ldots, U_m\}$ is universal if, for any n-qubit unitary $V \in U(2^n)$ and $\varepsilon > 0$, there exists a composition $U \in \mathbf{Circ}(\mathcal{U}; n)$ and phase φ such that

$$\|U - e^{i\varphi}V\| \leq \varepsilon.$$

Two-level unitary transformations in a fixed basis are sufficient for universality:

Theorem 2.3 (Two-level gates are universal). *Any $U \in U(\mathcal{H})$ can be decomposed as a product of at most $\frac{n(n-1)}{2}$ two-level unitary transformations.*

Theorem 2.4 (Universal gate set). *The set consisting of all single-qubit gates together with the CNOT gate is universal.*

Theorem 2.5 (Universal gate set with discrete gates). *The CNOT, Hadamard H, and T-gates form a universal gate set.*

3 Quantum Circuit model

§ 3.1 Generalities of quantum computation

3.1.1 Encoding information with quantum states

Let us now give some crucial examples for encoding classical information into quantum states.

Basic encoding

Amplitude encoding

Repeated amplitude encoding

Product encoding

§ 3.2 Quantum circuit model

3.2.1 Encoding information with qubits

3.2.2 Unitary transform as quantum gates

$$C(U) = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U \tag{3.1}$$

likewise we use notation $C^k(U)$ to denote the $(k+1)$-controlled U gate

$$C^k(U) = (I - |1\rangle\langle 1|^{\otimes k}) \otimes I + |1\rangle\langle 1|^{\otimes k} \otimes U. \tag{3.2}$$

$$\bar{C}(U) = |1\rangle\langle 1| \otimes I + |0\rangle\langle 0| \otimes U \tag{3.3}$$

One-qubit unitary transform

Two-qubit unitary transform

***N*-qubit unitary transform**

Some special quantum gates

§ 3.3 Universal quantum gates

3.3.1 The notion of a universal gate set

For a given collection of quantum gates,

$$\mathcal{U} = \{U_1, \cdots, U_n\}, \tag{3.4}$$

using the quantum circuit model, we could compose these gates in many different ways, each composition will result in a unitary operator. The set of unitary operators obtained from this gate set will be denoted as

$$\mathbf{Circ}(\mathcal{U}; n) = \{U = F_{\text{circ}}(U_1, \cdots, U_n) | U_1, \cdots, U_n \in \mathcal{U}\}. \tag{3.5}$$

Definition 3.1. A set of quantum gates $\mathcal{U} = \{U_1, \cdots, U_m\}$ (not necessarily finite) is called universal if, for any $n \geq n_0$, the unitary transform $V \in U(2^n)$ can be approximated with arbitrary accuracy and up to an overall phase by quantum circuit constructed with this gate set. More precisely, $\forall V \in U(2^n)$ and $\forall \varepsilon > 0$, there exist a unitary transform $U \in \mathbf{Circ}(\mathcal{U}; n)$ and a phase factor $e^{i\varphi}$ such that

$$\|U - e^{i\varphi} V\| \leq \varepsilon.$$

That is, for all $n \geq 1$, $\mathbf{Circ}(\mathcal{U}; n)$ is dense in $U(2^n)$.

Note that the norm in the above definition can be chosen as any reasonable one, here and below, I will use the sup norm $\|\cdot\|_{\text{sup}}$.

3.3.2 Two-level unitary transformations are universal

When dealing with the problems related to the universal quantum gates, it is useful to introduce the notion of *two-level unitary transformation* in a given

basis. Consider a d-dimensional Hilbert space $\mathcal{H}$ with standard basis $|0\rangle, \cdots, |d-1\rangle$, we call a unitary transformation U a two-level unitary transformation[1] if it can be decomposed as a direct sum $U = U^{(2)} \oplus I^{(d-2)}$, where $U^{(2)}$ is a unitary transformation acting on the two-dimensional space spanned by two basis states $|i\rangle$ and $|j\rangle$. Notice that the definition depends on the choice of the basis. In general, a two-level unitary transformation in basis $\mathcal{B}_1$ will not be a two-level unitary transformation in basis $\mathcal{B}_2$.

The set of all unitary transformations on $\mathcal{H}$ is denoted as $U(\mathcal{H})$. There is a useful result about the decomposition of arbitrary unitary transformation in $U(\mathcal{H})$ as a product of two-level unitary transformations.

Proposition 3.1. *Any unitary transformation U in $U(\mathcal{H})$ can be decomposed as a product of two-level unitary transformations $U_1, \cdots, U_k$ with $k \leq \frac{n(n-1)}{2}$, where $n = \dim \mathcal{H}$.*

Proof. Since the basis is fixed, we can prove this in matrix form. Suppose that $U = (u_{ij})$ is a $n \times n$ matrix,what we want to do is make the entries of the first column be $1, 0, \cdots, 0$. To this end, we will construct some two-level unitary matrices $U_1, \cdots, U_n$ such that $U_n \cdots U_2 U_1 U = I$, thus $U = U_1^\dagger U_2^\dagger \cdots U_n^\dagger$. If $u_{2,1} = 0$, set $U_1 = I$; if $u_{2,1} \neq 0$, set

$$U_1 = \begin{pmatrix} \frac{u_{11}^*}{\sqrt{|u_{11}|^2+|u_{21}|^2}} & \frac{u_{21}^*}{\sqrt{|u_{11}|^2+|u_{21}|^2}} \\ \frac{u_{21}}{\sqrt{|u_{11}|^2+|u_{21}|^2}} & \frac{-u_{11}}{\sqrt{|u_{11}|^2+|u_{21}|^2}} \end{pmatrix}^{(1,2)} \oplus I^{(n-2)} = U^{(1,2)} \oplus I^{(n-2)},$$

where $U^{(1,2)}$ is a unitary matrix acting on the space spanned by the 1st and 2nd basis states. Then matrix $U_1 U$ will be of the form

$$U_1 U = \begin{pmatrix} u'_{11} & u'_{12} & \cdots & u'_{1n} \\ 0 & u'_{22} & \cdots & u'_{2n} \\ u'_{31} & u'_{32} & \cdots & u'_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ u'_{n1} & u'_{n2} & \cdots & u'_{nn} \end{pmatrix}$$

We now construct U_2 in the same spirit. If $u'_{31} = 0$, set $U_2 = I$; if $u'_{31} \neq 0$, set U_2 as

$$U_2 = \begin{pmatrix} \frac{u'^*_{11}}{\sqrt{|u'_{11}|^2+|u'_{31}|^2}} & \frac{u'^*_{31}}{\sqrt{|u'_{11}|^2+|u'_{31}|^2}} \\ \frac{u'31}{\sqrt{|u'_{11}|^2+|u_{31}|^2}} & \frac{-u'_{11}}{\sqrt{|u'_{11}|^2+|u'_{31}|^2}} \end{pmatrix}^{(1,3)} \oplus I^{(n-2)} = U^{(1,3)} \oplus I^{(n-2)},$$

[1] This is different from the notion of a unitary transformation acting non-trivially on one qubit (recall the fact and one qubit is a two-level system), which means that U can be decomposed as a tensor product $U = U^{(2)} \otimes I^{(2^{n-1})}$. Thus, only for $\dim \mathcal{H} = 2$, two notions are the same.

Then the first column of U_2U_1U will be $(u''_{11}, 0, 0, u''_{41}, \cdots, u''_{n1})^T$. Similarly, we can construct $U_3, \cdots, U_{n-1}$ such that

$$V = U_{n-1}\cdots U_1U = \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ 0 & v_{22} & \cdots & v_{2n} \\ 0 & v_{32} & \cdots & v_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & v_{n2} & \cdots & v_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & v_{22} & \cdots & v_{2n} \\ 0 & v_{32} & \cdots & v_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & v_{n2} & \cdots & v_{nn} \end{pmatrix} = I^{(1)} \oplus U',$$

where the second equality is from the unitarity of V. With this dimension reduction process and by induction on the rank of matrix U, we complete the proof. Since in each dimension reduction process $I^{(n-d)} \oplus U^{(d)} \to I^{(n-d+1)} \oplus U^{(d-1)}$ (d to $d-1$ dimensions), at most $n-1$ two-level unitary matrices are needed, thus the total number of two-level unitary matrices appeared in the decomposition is less than or equal to $(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$. □

3.3.3 Examples of universal gates

§ 3.4 Other quantum computing models

3.4.1 Measurement-only quantum computation

4 Quantum simulation

Part III
Quantum Algorithms

5 Foundational Quantum Algorithms

> Quantum computers cannot compute any function which is not Turing-computable, but they do provide new modes of computation for many classes of problem.
>
> *David Deutsch and Richard Jozsa, "Rapid solution of problems by quantum computation"*

In this chapter, we introduce several foundational quantum Algorithms that illustrate the principles of quantum computation and the sources of their computational advantage. We begin with the *Deutsch-Jozsa Algorithm*, which demonstrates the ability of a quantum computer to solve certain global properties of functions exponentially faster than any classical deterministic Algorithm. Next, we study the *Bernstein-Vazirani Algorithm*, which generalizes this idea to efficiently recover hidden linear structures. We then discuss *Simon's Algorithm*, which provides an exponential speedup for a problem with a hidden XOR mask, and forms the conceptual foundation for Shor's factoring Algorithm.

Along the way, we also introduce several basic techniques and tools that recur throughout quantum Algorithm design, including the use of the Hadamard transform, phase kickback, and quantum parallelism. These tools provide the building blocks for more advanced Algorithms in subsequent chapters.

Quantum computing provides a fundamentally new paradigm for processing information. Unlike classical computers, which manipulate bits that take definite values 0 or 1, quantum computers manipulate quantum bits, or qubits, which can exist in coherent superpositions of states. This allows quantum Algorithms to explore computational pathways simultaneously, opening the door to significant speedups for certain problems.

By the end of this chapter, the reader will gain an understanding of how quantum superposition and interference can be harnessed to solve computational problems more efficiently, and will be prepared to study more sophisticated Algorithms in the remainder of this text.

§ 5.1 Deutsch-Jozsa Algorithm

The Deutsch-Jozsa Algorithm, introduced by David Deutsch and Richard Jozsa in their 1992 paper "Rapid solution of problems by quantum computation", marked an important early milestone in the development of quantum Algorithms. The formulation familiar to us today—featuring the standard quantum circuit model and oracle—was later refined in 1998 by Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca in their paper "Quantum Algorithms revisited". Both papers are highly readable, and we encourage interested readers to consult them directly.

As a deterministic quantum Algorithm, the Deutsch-Jozsa Algorithm has limited direct practical applications. Its significance lies in demonstrating, for the first time, that quantum computation can achieve an exponential speedup over all *deterministic* classical Algorithms. Studying this Algorithm offers valuable insight into how quantum Algorithms are structured and why they can outperform classical methods.

It is worth noting that this exponential separation holds only when compared with deterministic classical Algorithms. If classical randomness is allowed, such an exponential advantage no longer appears. The first quantum Algorithm to establish an exponential speedup over *any* classical stochastic Algorithm is Simon's Algorithm, which we will encounter later in this chapter.

Definition 5.1 (Problem of the Deutsch-Jozsa Algorithm). Consider a function

$$f : \{0,1\}^n \to \{0,1\}$$

that accepts n-bit binary inputs and outputs either 0 or 1. It is guaranteed that f is either *constant* (producing the same output for all inputs) or *balanced* (outputting 1 for exactly half of the inputs and 0 for the other half). The objective is to determine whether f is constant or balanced by querying the oracle.

For example, consider a one-bit input function $f : \{0,1\} \to \{0,1\}$. The function is constant if $f(0) = f(1)$, and balanced if $f(0) \neq f(1)$ (equivalently, $f(0) \oplus f(1) = 1$, where $\oplus$ denotes addition modulo 2). Determining whether f is constant or balanced is thus equivalent to evaluating $f(0) \oplus f(1)$. This can also be viewed as a quantum XOR game, if the result is 0, then f is constant; otherwise, f is balanced.

5.1.1 *Deutsch-Jozsa Algorithm: $n = 1$ Case*

Since this is the first quantum Algorithm we encounter, let us begin with the simplest example, namely the $n = 1$ case of the Deutsch-Jozsa Algorithm, to gain some intuition about how quantum Algorithms work.

Classically, we would need to evaluate f twice to solve the problem, namely compute $f(0)$ and $f(1)$. There is no way around this. Quantum mechanically, we can do better. To begin, we need to "promote" the function f to a unitary operation U_f, defined by

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle. \qquad (5.1)$$

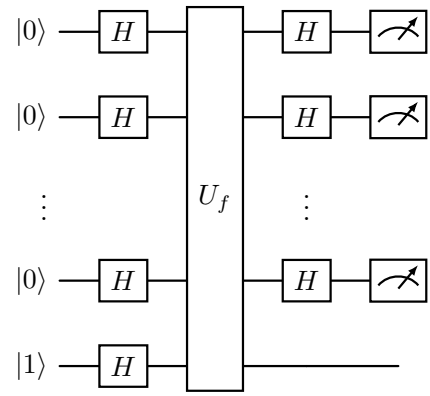


Fig. 5.1 Quantum circuit for the Deutsch-Jozsa Algorithm. The top n qubits encode the input of the function f, and the single bottom qubit encodes the output of f.

We do not concern ourselves here with how to implement U_f in an actual quantum circuit. Instead, we treat it as a black box that can be used freely. Such a black box is called a *quantum oracle*, and we will encounter this concept frequently in the following sections.

Exercise 5.1. As a simple practice, prove that U_f defined above is a unitary operator for both balanced and constant functions f.

Proof. Consider the action of U_f on the four computational basis states:

$$\begin{aligned}
|0\rangle|0\rangle &\mapsto |0\rangle|0 \oplus f(0)\rangle,\\
|0\rangle|1\rangle &\mapsto |0\rangle|1 \oplus f(0)\rangle,\\
|1\rangle|0\rangle &\mapsto |1\rangle|0 \oplus f(1)\rangle,\\
|1\rangle|1\rangle &\mapsto |1\rangle|1 \oplus f(1)\rangle.
\end{aligned}$$

Notice that $0 \oplus f(0) \neq 1 \oplus f(0)$ regardless of the value of $f(0)$, and similarly $0 \oplus f(1) \neq 1 \oplus f(1)$. Hence, the images of the basis states under U_f form an orthonormal basis, which shows that U_f is indeed unitary. □

Exercise 5.2. Show that for the balanced function $f(0) = 0$ and $f(1) = 1$, the oracle U_f acts as a CNOT gate.

With the quantum oracle U_f, the Deutsch–Jozsa Algorithm works as follows. We prepare two qubits: one for the input of f and one for the output. They are initialized in the state $|0\rangle|1\rangle$. Then we perform the following steps:

1. Apply Hadamard gates to both qubits: $(H \otimes H)\,|0\rangle|1\rangle$.

2. Apply the oracle: $(U_f \otimes I)\big((H \otimes H)\,|0\rangle|1\rangle\big)$.
3. Apply a Hadamard gate to the first qubit: $(H \otimes I)\,(U_f \otimes I)\big(H \otimes H\,|0\rangle|1\rangle\big)$.
4. Measure the first qubit in the computational basis.

Let us examine what happens step by step. After step 1 and 2, we obtain

$$\begin{aligned}|0\rangle|1\rangle &\xrightarrow{H\otimes H} |+\rangle|-\rangle = \frac{1}{2}\Big(|0\rangle(|0\rangle - |1\rangle) + |1\rangle(|0\rangle - |1\rangle)\Big)\\ &\xrightarrow{U_f\otimes I} \frac{1}{2}|0\rangle\Big(|0\oplus f(0)\rangle - |1\oplus f(0)\rangle\Big) + |1\rangle\Big(|0\oplus f(1)\rangle - |1\oplus f(1)\rangle\Big).\end{aligned}$$

Next, we use a simple but important trick: we rewrite the action of the oracle so that the value of $f(x)$ appears as a phase. The key identity is

$$|0\oplus z\rangle - |1\oplus z\rangle = (-1)^z\big(|0\rangle - |1\rangle\big), \tag{5.2}$$

valid for $z = 0, 1$. The state after applying the oracle can be rewritten as

$$\begin{aligned}&\frac{1}{2}\Big(|0\rangle(|0\oplus f(0)\rangle - |1\oplus f(0)\rangle) + |1\rangle(|0\oplus f(1)\rangle - |1\oplus f(1)\rangle)\Big)\\ &= \frac{1}{2}\Big((-1)^{f(0)}|0\rangle(|0\rangle - |1\rangle) + (-1)^{f(1)}|1\rangle(|0\rangle - |1\rangle)\Big)\\ &= (-1)^{f(0)}\frac{1}{2}\Big(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle\Big)(|0\rangle - |1\rangle).\end{aligned} \tag{5.3}$$

Ignoring the second qubit and the global phase, the first qubit is in the state

$$\frac{1}{\sqrt{2}}\Big(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle\Big). \tag{5.4}$$

Our goal is to determine $f(0) \oplus f(1)$, which is encoded in the relative phase $(-1)^{f(0)\oplus f(1)}$ of the first qubit. To read out this information, we apply a Hadamard gate to the first qubit and then measure it in the computational basis. This is an example of an *interferometric measurement scheme*, a standard method for extracting the coefficient of a state in a given basis; you will encounter this technique in many other contexts as well.

After applying the Hadamard, the state of the first qubit becomes

$$\begin{aligned}&\frac{1}{2}\Big(|0\rangle + |1\rangle + (-1)^{f(0)\oplus f(1)}|0\rangle - (-1)^{f(0)\oplus f(1)}|1\rangle\Big)\\ =&\frac{1}{2}\Big(\big(1 + (-1)^{f(0)\oplus f(1)}\big)|0\rangle + \big(1 - (-1)^{f(0)\oplus f(1)}\big)|1\rangle\Big).\end{aligned} \tag{5.5}$$

The probabilities of measuring the first qubit in the computational basis are then

$$p(0) = \frac{1}{4}\big|1 + (-1)^{f(0)\oplus f(1)}\big|^2, \qquad p(1) = \frac{1}{4}\big|1 - (-1)^{f(0)\oplus f(1)}\big|^2. \tag{5.6}$$

Hence, if $f(0) \oplus f(1) = 0$ (the function is constant), we will certainly measure $|0\rangle$. If $f(0) \oplus f(1) = 1$ (the function is balanced), we will measure $|1\rangle$. In other words, a single query to the quantum oracle suffices to determine with certainty whether f is constant or balanced. In contrast, a classical Algorithm would require evaluating f twice.

What happens here is a manifestation of *quantum parallelism*: in step 2, the oracle effectively evaluates f on both possible inputs simultaneously, encoding the results in a superposition. The final Hadamard gate then transforms this information into measurable probability amplitudes, allowing us to extract the answer efficiently.

5.1.2 Deutsch-Jozsa Algorithm

Now let us turn to the general case in which f has an n-bit input and one output bit. As in the $n = 1$ case, the classical oracle must be replaced by a quantum oracle U_f that coherently evaluates the function. By definition, the quantum oracle acts as

$$U_f\,|x\rangle|y\rangle \;=\; |x\rangle\,|y \oplus f(x)\rangle, \tag{5.7}$$

where $x = x_1x_2\cdots x_n$ is an n-bit string, and we write

$$|x\rangle := |x_1\rangle|x_2\rangle\cdots|x_n\rangle. \tag{5.8}$$

This means we use the canonical encoding that maps each bit string to a corresponding computational basis state.

Exercise 5.3. Prove that quantum oracle U_f is a unitary operation.

The quantum circuit for the Deutsch–Jozsa Algorithm is shown in Figure 5.1. The Algorithm proceeds as follows:

1. **Initial State:** Prepare an n-qubit input register in the state $|0\rangle^{\otimes n}$ and a single output qubit in the state $|1\rangle$:

$$|0\rangle^{\otimes n} \otimes |1\rangle. \tag{5.9}$$

2. **Hadamard Transform:** Apply the Hadamard gate to each qubit:

$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \tag{5.10}$$

3. **Oracle Query:** Next, we apply the quantum oracle U_f that encodes the function f:

$$U_f\,|x\rangle|y\rangle = |x\rangle\,|y \oplus f(x)\rangle. \tag{5.11}$$

After applying the oracle, the combined state of the input and output registers becomes

$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle), \tag{5.12}$$

where we have used the identity (Eq. (5.2))

$$|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle = (-1)^{f(x)}(|0\rangle - |1\rangle). \tag{5.13}$$

Notice that the output qubit is no longer entangled with the input register. This means we can safely ignore it, leaving the first register in the state

$$|\psi_f\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle. \tag{5.14}$$

Let us pause to understand the structure of this state in different cases.

(1) Constant functions. Suppose $f(x)$ is constant. If $f(x) = 0$ for all x, then

$$|\psi_f\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle, \tag{5.15}$$

which is just an equal superposition of all basis states. On the other hand, if $f(x) = 1$ for all x, we have

$$|\psi_f\rangle = -\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle. \tag{5.16}$$

Up to an overall minus sign, the state is again an equal superposition of all basis states. The key point is that for constant functions, all amplitudes have the same magnitude and phase (up to a global sign).

(2) Balanced functions. Now suppose f is balanced. This means that exactly half of the inputs, say $x \in A$, satisfy $f(x) = 0$, while the other half, $x' \in A'$, satisfy $f(x') = 1$, where

$$\{0,1\}^n = A \cup A', \quad A \cap A' = \emptyset, \quad |A| = |A'| = 2^{n-1}.$$

In this case, we can decompose $|\psi_f\rangle$ as

$$|\psi_f\rangle = \frac{1}{\sqrt{2}}\left(|\psi_f^{(0)}\rangle - |\psi_f^{(1)}\rangle\right), \tag{5.17}$$

where $|\psi_f^{(0)}\rangle$ and $|\psi_f^{(1)}\rangle$ are equal superpositions of the basis states in A and A', respectively. Unlike the constant case, the amplitudes here have relative

phases, which will play a crucial role in distinguishing balanced from constant functions.

4. **Second Hadamard Transform:** We now apply $H^{\otimes n}$ to the input register. Using the standard identity (see Exercise 5.4 for a proof),

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z\in\{0,1\}^n} (-1)^{x\cdot z}|z\rangle, \tag{5.18}$$

the state becomes

$$\frac{1}{2^n} \sum_{z\in\{0,1\}^n} \left(\sum_{x\in\{0,1\}^n} (-1)^{f(x)+x\cdot z} \right) |z\rangle. \tag{5.19}$$

Let us denote the coefficient of $|z\rangle$ by

$$c_z = \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{f(x)+x\cdot z}. \tag{5.20}$$

Consider the special case $z = 0$, where all components of z are zero. Then $x \cdot z = 0$ for all x, so the coefficient reduces to

$$c_0 = \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}. \tag{5.21}$$

If f is constant, all terms in the sum are the same, giving $c_0 = \pm 1$. The final state is then

$$\pm\, |0\rangle^{\otimes n}. \tag{5.22}$$

On the other hand, if f is balanced, half of the terms are $+1$ and half are -1, so they cancel exactly, giving $c_0 = 0$. This difference in amplitude for $|0\rangle^{\otimes n}$ is what allows us to distinguish constant functions from balanced ones in the Deutsch-Jozsa Algorithm.

5. **Measurement:** Measure the n-qubit input register. The outcome reveals the type of function:

   - If f is constant, interference ensures only $|0\rangle^{\otimes n}$ appears.
   - If f is balanced, destructive interference prevents $|0\rangle^{\otimes n}$ from appearing.

   Measuring $|0\rangle^{\otimes n}$ indicates a constant function; any other outcome indicates a balanced function.

Exercise 5.4. The following are two crucial results of Hadamard gates that will be used frequently:

1. Prove that

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle, \tag{5.23}$$

where $|x\rangle = |x_1\rangle|x_2\rangle\cdots|x_n\rangle$.

2. The identity

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z\in\{0,1\}^n} (-1)^{x\cdot z}|z\rangle \tag{5.24}$$

is a basic and frequently used fact. Here we use the convention $x \cdot z = x_1 z_1 \oplus \cdots \oplus x_n z_n$ (dot product modulo 2), and the exponent $(-1)^{x\cdot z}$ is interpreted in the usual way ($(-1)^0 = 1$, $(-1)^1 = -1$). Prove (5.24) and verify that applying $H^{\otimes n}$ again returns $|x\rangle$.

Proof. 1. This is easy. You can rigorously prove it using mathematical induction.

2. Write $x = (x_1, \ldots, x_n)$ and use the single-qubit identities

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \qquad H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{|0\rangle + (-1)^1|1\rangle}{\sqrt{2}}.$$

Hence for each bit $x_i \in \{0, 1\}$,

$$H|x_i\rangle = \frac{|0\rangle + (-1)^{x_i}|1\rangle}{\sqrt{2}}.$$

Taking the n-fold tensor product gives

$$H^{\otimes n}|x\rangle = \bigotimes_{i=1}^{n} \frac{|0\rangle + (-1)^{x_i}|1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2^n}} \sum_{z\in\{0,1\}^n} (-1)^{\sum_i x_i z_i}|z\rangle,$$

where the sum is over all n-bit strings $z = (z_1, \ldots, z_n)$. Noting that $\sum_i x_i z_i \equiv x \cdot z \pmod 2$, we obtain (5.24).

To verify that $H^{\otimes n}$ is its own inverse on computational-basis states, apply $H^{\otimes n}$ to the right-hand side of (5.24):

$$\begin{aligned} H^{\otimes n}\Big(\frac{1}{\sqrt{2^n}} \sum_{z\in\{0,1\}^n} (-1)^{x\cdot z}|z\rangle\Big) &= \frac{1}{2^n} \sum_{z\in\{0,1\}^n} \sum_{w\in\{0,1\}^n} (-1)^{x\cdot z + w\cdot z}|w\rangle \\ &= \sum_{w\in\{0,1\}^n} \left(\frac{1}{2^n} \sum_{z\in\{0,1\}^n} (-1)^{(x\oplus w)\cdot z}\right)|w\rangle. \end{aligned}$$

The inner sum equals 1 when $w = x$ and 0 otherwise (orthogonality of characters over $\mathbb{F}_2^n$), so the result is $|x\rangle$. Equivalently, note $H^2 = I$ on each qubit, so $(H^{\otimes n})^2 = I^{\otimes n}$. □

Exercise 5.5. For the case $n = 2$, write down all possible Boolean functions (two constant functions and six balanced functions), and calculate the final state of the input register for the Deutsch-Jozsa Algorithm.

Classically, if we want to solve the problem brute-forcely, we need to check half of the inputs, thus evaluate f for 2^{n-1} times at the worst case. The Algorithm solves the problem with only one evaluation of the oracle, compared to the exponentially growing number of evaluations required by a classical Algorithm. The quantum Algorithm achieves this exponential speedup by leveraging quantum superposition and interference.

§ 5.2 Bernstein-Vazirani Algorithm

The Bernstein–Vazirani Algorithm is one of the earliest and clearest examples of a quantum Algorithm that can dramatically outperform any classical Algorithm for a well-defined problem. It can be viewed as a application of the Deutsch-Jozsa Algorithm. The Algorithm was first introduced by Ethan Bernstein and Umesh Vazirani in their 1993 paper, and later revisited in their 1997 paper.

The Algorithm treats the following problem:

Definition 5.2 (Bernstein–Vazirani problem). Let f be a function from n-bit strings to a single bit,

$$f : \{0,1\}^n \to \{0,1\},$$

such that there exists a secret bit string $s \in \{0,1\}^n$ with the property that, for every input $x \in \{0,1\}^n$,

$$f(x) = x \cdot s,$$

where the dot product is defined by

$$x \cdot s = x_1 s_1 \oplus x_2 s_2 \oplus \cdots \oplus x_n s_n,$$

with $x = x_1 x_2 \cdots x_n$ and $s = s_1 s_2 \cdots s_n$. The task is to determine the secret string s using as few queries to f as possible.

Exercise 5.6. 1. For the 3-bit string $s = 011$, compute $x \cdot s$ for all $x \in \{0,1\}^3$. As an example, take $x = 001$. Then

$$x \cdot s = (0 \times 0) \oplus (0 \times 1) \oplus (1 \times 1) = 1. \tag{5.25}$$

The function defined by $f(x) := x \cdot s$ is an example of a function satisfying the Bernstein–Vazirani property.

2. Show that the majority function

$$\mathrm{maj}(x) = \begin{cases} 1, & \text{if } x \text{ contains at least two 1's,} \\ 0, & \text{otherwise,} \end{cases}$$

is a counterexample: it does not satisfy the Bernstein–Vazirani property.

Hint. Any function $f(x) = x \cdot s$ is linear over $\mathbb{F}_2$, i.e. it obeys

$$f(x \oplus y) = f(x) \oplus f(y) \quad \text{for all } x, y. \tag{5.26}$$

To disprove that maj has the Bernstein–Vazirani form, find explicit x, y for which

$$\mathrm{maj}(x \oplus y) \neq \mathrm{maj}(x) \oplus \mathrm{maj}(y). \tag{5.27}$$

For instance, take $x = 110$ and $y = 011$. Then $x \oplus y = 101$, while

$$\mathrm{maj}(110) = 1, \qquad \mathrm{maj}(011) = 1, \qquad \mathrm{maj}(101) = 1,$$

so $\mathrm{maj}(110) \oplus \mathrm{maj}(011) = 0 \neq 1 = \mathrm{maj}(101)$, showing the required contradiction. □

Classically, the problem can be solved by considering a complete basis of bit strings,

$$10\cdots0, \quad 010\cdots0, \quad \cdots, \quad 0\cdots01.$$

Evaluating the function $f(x) := x \cdot s$ on these strings yields

$$\begin{aligned} 10\cdots0 \cdot s &= s_1, \\ 010\cdots0 \cdot s &= s_2, \\ &\vdots \\ 0\cdots01 \cdot s &= s_n. \end{aligned}$$

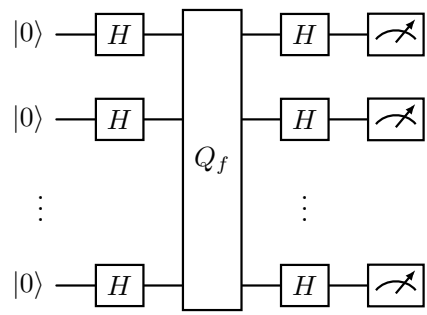


Fig. 5.2 Quantum circuit for the Bernstein-Vazirani Algorithm.

Hence, the secret string s can be determined with n queries. Since the bits of s are independent, no classical strategy can do better. Remarkably, a quantum computer can solve the same problem with far fewer queries.

From Eq. (5.24), we observe that an inner product appears in the phase factor $(-1)^{x \cdot z}$. If we can prepare the corresponding state, applying Hadamard gates directly yields s. This state naturally appears just before the final Hadamard transforms in the Deutsch–Jozsa Algorithm, so the same circuit can be used to solve the Bernstein–Vazirani problem with only a single query.

To implement this, we prepare an n-qubit input register in the state $|0\rangle^{\otimes n}$ and a single auxiliary qubit in the state $|1\rangle$. Applying Hadamard gates to all $n+1$ qubits yields

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes \frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big). \tag{5.28}$$

We then apply the quantum oracle U_f encoding f:

$$U_f\,|x\rangle|y\rangle = |x\rangle\,|y \oplus f(x)\rangle. \tag{5.29}$$

After acting on $|\psi\rangle$, the combined state becomes

$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle \otimes \frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big). \tag{5.30}$$

On the input register, this action is equivalently described by the operator Q_f:

$$Q_f \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle. \tag{5.31}$$

Recall that $f(x) := x \cdot s$, then apply quantum oracle Q_f given

$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot s}|x\rangle. \tag{5.32}$$

Then apply Hadamard gates $H^{\otimes n}$ will give $|s\rangle$, see Exercise 5.4. Via measuring the final state, we can readout s.

The quantum circuit of Bernstein-Vazirani Algorithm is completely the same as that for Deutsch-Jozsa, a simpler version is given in Figure 5.2 based on Q_f.

§ 5.3 Simon's Algorithm

Simon's Algorithm is a quantum Algorithm designed to find a hidden string s in a particular class of functions, achieving an exponential speedup over any classical Algorithm. In this setting, the hidden string s appears in a bitwise addition relation that plays a role analogous to the period of a function. For this reason, the problem is sometimes referred to as the *period-finding problem*.

Simon's Algorithm holds historical significance as it was the first to demonstrate an exponential quantum advantage over the best known classical Algorithm, revealing how quantum computation can surpass classical methods in well-defined computational tasks. The Algorithm was proposed by Daniel R.

Simon in 1994. Simon later recounted the story behind his Algorithm in a blog article[1]:

> I submitted my Algorithm to a theoretical computer science conference (STOC 1993), but it was rejected. However, Peter Shor was on the program committee for that conference, and immediately saw the potential (that I had had absolutely no inkling of, to be honest) for applying the same general Algorithm structure to concrete problems like factoring and discrete logaritheorems. After trying unsuccessfully to persuade the committee to accept my paper, he got to work on his own, and submitted it to the next major theoretical computer science conference (FOCS 1993)—in parallel with mine, resubmitted. We had in fact agreed to merge the papers if only his was accepted, but the committee fortunately agreed to accept them both, giving us each credit for our respective portions of the resulting achievement.

From this account, we see that Simon's Algorithm served as a precursor to Peter Shor's celebrated factoring Algorithm. It laid the conceptual foundation for many of the seminal advances in quantum computation that followed. The problem addressed by Simon's Algorithm can be rigorously stated as follows:

Definition 5.3 (Simon's Problem). Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function satisfying the *periodicity condition*: there exists a string $s \in \{0,1\}^n$ such that

$$f(x) = f(y) \quad \text{if and only if} \quad y = x \oplus s$$

for all $x, y \in \{0,1\}^n$, where $\oplus$ denotes bitwise addition modulo two (bitwise `XOR`).

This implies that f can only take one of the following two forms:

- *One-to-One Case*: If $s = 0$, then f is one-to-one.
- *Two-to-One Case*: If $s \neq 0$, then for any distinct $x, y \in \{0,1\}^n$,

$$f(x) = f(y) \quad \text{if and only if} \quad y = x \oplus s,$$

so f is two-to-one.

The task is to determine the hidden string s using as few evaluations of f as possible, where f is implemented as a black box (oracle).

The secret period bit string s is also referred to as a *mask*—a term commonly used in computer science—and, since it is applied to the inputs via bitwise XOR, it is often called an *XOR mask*. It is important to distinguish between periodicity under ordinary addition and that under bitwise addition modulo two, as this

[1] Grant Salton, Daniel Simon, and Cedric Lin, *Exploring Simon's Algorithm with Daniel Simon*, October 11, 2021.

difference can be subtle for first-time learners. For real-valued periodic functions, if T is a period, then $2T = T + T$ is also a period. Likewise, if we treat a bit string s as a binary number, then when s is a period, sums such as $s+s$, $s+s+s$, and so forth are also periods. In contrast, for functions defined using bitwise addition modulo two, any bit string s satisfies $s \oplus s = 0$. This property leads to an important consequence: if a nonzero period exists in this setting, it must be unique.

Example 5.1. Consider the three-bit string $s = 011$. We can compute the bitwise addition $x \oplus s$ for all $x \in \{0,1\}^3$ as follows:

$$\begin{aligned}
000 \oplus 011 &= 011 \\
001 \oplus 011 &= 010 \\
010 \oplus 011 &= 001 \\
011 \oplus 011 &= 000 \\
100 \oplus 011 &= 111 \\
101 \oplus 011 &= 110 \\
110 \oplus 011 &= 101 \\
111 \oplus 011 &= 100
\end{aligned}$$

A function $f : \{0,1\}^3 \to \{0,1\}^3$ satisfying Simon's property with period $s = 011$, according to the definition above, must be two-to-one. That is, its outputs are paired as follows:

$$\begin{aligned}
f(000) = f(011) &= z_1 \\
f(001) = f(010) &= z_2 \\
f(100) = f(111) &= z_3 \\
f(101) = f(110) &= z_4
\end{aligned}$$

where z_1, z_2, z_3, z_4 are all distinct. For example, one possible assignment is $z_1 = 000$, $z_2 = 001$, $z_3 = 010$, $z_4 = 011$. This illustrates that the function naturally pairs inputs that differ by the secret string $s = 011$.

In the classical setting, we can find the secret XOR mask s by identifying a *collision*—a pair of inputs that produce the same output. More precisely, we query the oracle by sending a bit string x_1 as input and observing the corresponding output $f(x_1)$. We then repeat this process for another input x_2, obtaining $f(x_2)$, and continue in this manner for additional queries. By keeping a record of the function values, we eventually find a pair of input strings x_i and x_j such that $f(x_i) = f(x_j)$. According to the defining property of Simon's problem, this equality implies that the two inputs differ by the secret string, i.e., $x_j = x_i \oplus s$. Using the property that any bit string satisfies $x \oplus x = 0$, we can verify that

$$s = x_i \oplus x_j. \tag{5.33}$$

Thus, the secret string s can be recovered.

One may now ask how many queries are required to solve Simon's problem—that is, to determine the secret XOR mask s. We have seen that s can be obtained by finding a *collision pair*, two distinct inputs x_i and x_j such that $f(x_i) = f(x_j)$. The question, then, becomes: how many queries does it take to find such a pair?

A straightforward approach is to query the oracle for every possible input. Since f is defined on bit strings of length n, there are 2^n possible inputs in total. By exhaustively querying all of them, we are guaranteed to find a matching pair and thus recover s. However, this brute-force strategy requires up to 2^n queries. In fact, because f is a two-to-one function, it is sufficient to check at most half of the inputs in the worst case. Therefore, we require no more than 2^{n-1} queries to find a collision and determine s.

Can we do better than brute-force querying? The answer is yes, if we query f with random inputs—but the improvement is limited. By reasoning similar to that used in the *birthday paradox* (which considers the probability that at least two people in a group of n share the same birthday), we can draw an analogy to Simon's problem. Roughly speaking, the total number of bit strings, 2^n, is analogous to the total number of days in a year, and the number of queries corresponds to the number of people chosen. More precisely, suppose we query one input x and then another input y. The probability that they form a collision is $p[f(x) = f(y)] = 1/(2^n - 1)$, since there are 2^n possible bit strings and we have already queried one. If we query k inputs, there are $\binom{t}{2}$ pairs of inputs. Hence, the probability of finding at least one collision is approximately $p_{\text{suc}} \sim \binom{t}{2}/(2^n - 1)$. Assuming a reasonable success probability, for example $p_{\text{suc}} \geq 2/3$, a simple calculation then shows that we expect to find a collision after roughly $t = O(2^{n/2})$ queries.

This quadratic improvement arises from the probabilistic nature of collisions: as we collect outputs of f, the likelihood of encountering a repeated value increases roughly with the square of the number of queries.

However, Simon's Algorithm performs significantly better, achieving an *exponential speedup* over classical approaches. It requires the use of a quantum oracle U_f which is given to us as a black box (an "oracle"), which has a similar form to the oracle used in the Deutsch-Jozsa Algorithm:

$$U_f \left|x\right\rangle \left|y\right\rangle = \left|x\right\rangle \left|y \oplus f(x)\right\rangle .$$

Here, both $f(x)$ and y are n-qubit states, and the bitwise addition $\oplus$ indicates that they are not treated as ordinary binary numbers, but rather as bit strings under modulo-two addition.

Simon's Algorithm mainly consists of two parts: (i) The *quantum part*, implemented by the quantum circuit shown in Figure 5.3, which is run $t = O(n)$ times. (ii) The *classical postprocessing*, which operates on the measurement outcomes and scales efficiently in n, used to determine the secret XOR mask s.

Here's a detailed breakdown of the Algorithm's steps:

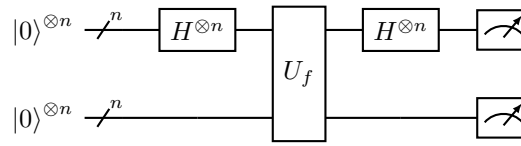


Fig. 5.3 Quantum circuit for the Simon's Algorithm. The top n qubits hold the input register, and the bottom n qubits are the ancillary register used for the phase oracle U_f.

1. **Initial Setup**
   Prepare two quantum registers:
   - An n-qubit input register initialized to $|0\rangle^{\otimes n}$.
   - An n-qubit output register also initialized to $|0\rangle^{\otimes n}$.

   The initial state of the system is $|0\rangle^{\otimes n}|0\rangle^{\otimes n}$.
2. **Apply Hadamard Gates**
   Apply the Hadamard gate H to all qubits in the input register to create a superposition of all possible input states:

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle. \tag{5.34}$$

   The state now becomes

$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|0\rangle^{\otimes n}. \tag{5.35}$$

3. **Query the Oracle**
   Use the oracle U_f to compute $f(x)$ for each x:

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle. \tag{5.36}$$

   After applying the oracle, the state becomes

$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle|f(x)\rangle. \tag{5.37}$$

   It is clear that this oracle introduce entanglement to the output state.
4. **Apply Hadamard Gates**
   Apply the Hadamard gate H to all qubits in the input register. This transforms the state to

$$\frac{1}{2^n}\sum_{z\in\{0,1\}^n}\sum_{x\in\{0,1\}^n}(-1)^{x\cdot z}|z\rangle|f(x)\rangle \tag{5.38}$$

where we used Eq. (5.24).

5. **Measure the Output Register**
   Measure the output register and obtain the value $f(x) = w$. There are two possible cases:

   - *One-to-One Case*: If f is one-to-one, there is only a single x such that $f(x) = w$. The post-measurement state becomes

   $$\frac{1}{\sqrt{2^n}}\sum_{z}(-1)^{x\cdot z}|z\rangle|f(x) = w\rangle. \tag{5.39}$$

   - *Two-to-One Case*: If f is two-to-one, there exist x and $y = x \oplus s$ such that $f(x) = f(y) = w$. The post-measurement state is

   $$\frac{1}{\sqrt{2^n}}\sum_{z}\frac{1}{\sqrt{2}}\left[(-1)^{x\cdot z} + (-1)^{y\cdot z}\right]|z\rangle|f(x) = w\rangle. \tag{5.40}$$

   Notice that this measurement step is not strictly necessary, as the result itself is not used in subsequent computations; however, we include it here because it simplifies the analysis and helps illustrate the underlying idea.

6. **Generate Linear Equations**
   The relationships derived from the measurements create linear equations involving the secret string s.
   If f is one-to-one, measuring the state in Eq. (5.39) for the input register yields a uniformly random bit string z. If f is two-to-one, however, the measurement returns a random bit string z satisfying $x \cdot z = y \cdot z \pmod 2$, since otherwise the amplitude $(-1)^{x\cdot z} + (-1)^{y\cdot z}$ cancels out. Explicitly,

   $$(-1)^{x\cdot z} + (-1)^{y\cdot z} = \begin{cases} 0, & x\cdot z \neq y\cdot z, \\ \pm 2, & x\cdot z = y\cdot z, \end{cases} \tag{5.41}$$

   so only the basis states $|z\rangle$ satisfying $x \cdot z = y \cdot z \pmod 2$ have nonzero amplitude. Since $y = x \oplus s$, we find that

   $$x\cdot z = (x\oplus s)\cdot z \pmod 2 \;\Rightarrow\; s\cdot z = 0 \pmod 2.$$

   Thus, each measurement produces a random bit string z orthogonal to s.

7. **Repeat the Process $O(n)$ Times**
   Repeat steps 1–6 about n times. Each iteration yields a new, independent equation of the form $z \cdot s = 0$ involving the hidden string s. Since s has n

independent components, only n such linear equations are required to determine it. This results in an exponential reduction in the number of queries needed.

8. **Classical Postprocessing: Solve the System of Equations**
   Once a sufficient number of equations have been collected,

$$\begin{aligned} z_1 \cdot s &= 0, \\ z_2 \cdot s &= 0, \\ &\vdots \\ z_m \cdot s &= 0, \end{aligned}$$

   use classical linear algebra methods (such as Gaussian elimination) to solve for the hidden string s.

Simon's Algorithm leverages quantum principles, particularly superposition and interference, to solve the problem in $O(n)$ evaluations of f. Unlike the Deutsch-Jozsa and Grover Algorithms, Simon's Algorithm requires some classical data processing after measurement.

6 Grover's Search Algorithm

In this chapter, we introduce the quantum search algorithm.

§ 6.1 Grover's Search Algorithm

Grover's search algorithm, also called the quantum search algorithm, is a cornerstone of quantum computation, providing a quadratic speedup for so-called unstructured search problems. It was introduced by Lov K. Grover in 1996. Whereas classical algorithms require $O(N)$ queries to search an unsorted database of N elements, Grover's algorithm achieves the same task with only $O(\sqrt{N})$ queries by exploiting quantum parallelism and amplitude amplification.

Searching an unstructured database is a fundamental task in many practical applications, such as finding the lowest-priced product, determining the shortest path to a destination via transfers, and solving other optimization problems. We begin with the simplest case of a single marked item and later consider the general case with multiple marked items.

Definition 6.1 (Unstructured search problem). The goal of Grover's algorithm is to locate the *marked element* in an unsorted database. Given an oracle function

$$f : \{0,1\}^n \to \{0,1\},$$

where $f(x) = 1$ for the marked item x' and $f(x') = 0$ for all other items, the task is to find marked elements x''s using fewer oracle queries than would be required classically.

The core of Grover's algorithm is the *Grover iteration*. The performance of the algorithm depends on the ratio M/N, where M is the number of marked items and N is the total number of items. For convenience, we assume the search

space contains $N = 2^n$ possible items (which is convenient for us to encode the items in to a n-qubit space by mapping bit string to the computation basis $x = x_1x_2\cdots x_n \mapsto |x\rangle = |x_1\rangle|x_2\rangle\cdots|x\rangle$). A classical search requires $O(N)$ oracle queries, since in the worst case the marked item might be the last one checked. Grover's algorithm, by contrast, accomplishes the search with only $O(\sqrt{N})$ queries.

Recall that the classical oracle is a function $f : \{0,1\}^n \to \{0,1\}$, with $f(x) = 1$ for the marked item x_{marked} and $f(x) = 0$ for all other items. We can generalize this to a quantum oracle U_f, which is a unitary operation that flips the sign of the amplitude of the marked state:

$$U_f|x\rangle = (-1)^{f(x)}|x\rangle, \tag{6.1}$$

where $|x\rangle = |x_1\rangle|x_2\rangle\cdots|x_n\rangle$ corresponds to the binary string x.

This construction is the same as the oracles encountered in the Deutsch–Jozsa and Bernstein–Vazirani algorithms. By introducing an ancillary qubit initialized in the state $|-\rangle$, we define the oracle

$$\tilde{U}_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle, \tag{6.2}$$

so that

$$\tilde{U}_f|x\rangle(|0\rangle - |1\rangle) = (-1)^{f(x)}|x\rangle(|0\rangle - |1\rangle) = (U_f|x\rangle)(|0\rangle - |1\rangle). \tag{6.3}$$

Dropping the ancillary qubit then recovers Eq. (6.1).

In particular, for the marked state x', $f(x') = 1$, so its phase is flipped:

$$U_f|x'\rangle = -|x'\rangle. \tag{6.4}$$

For all unmarked elements, $f(x) = 0$, so their phase remain unchanged.

6.1.1 Grover Iteration

The algorithm proceeds by repeatedly applying a procedure known as the *Grover iteration* or *Grover operator* to amplify the amplitude of the marked state. As we shall see, each Grover iteration corresponds to a rotation within a two-dimensional plane. The rotation angle at each step depends on the ratio M/N between the number of marked items and the total number of items.

Suppose the search space contains $N = 2^n$ items, of which M are marked. Define the states

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x:\ \text{unmarked}} |x\rangle, \tag{6.5}$$

representing an equal superposition of all unmarked items, and

$$|\beta\rangle = \frac{1}{\sqrt{M}} \sum_{x:\ \text{marked}} |x\rangle, \tag{6.6}$$

representing an equal superposition of all marked items.

The uniform superposition over all items,

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x} |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle = H^{\otimes n}|0\rangle^{\otimes n}, \tag{6.7}$$

can be expressed as a superposition of $|\alpha\rangle$ and $|\beta\rangle$:

$$|\psi\rangle = \frac{\sqrt{N-M}}{\sqrt{N}}|\alpha\rangle + \frac{\sqrt{M}}{\sqrt{N}}|\beta\rangle. \tag{6.8}$$

We assume that M/N is small. In this regime, the coefficient $c_\alpha = \frac{\sqrt{N-M}}{\sqrt{N}}$ corresponding to the unmarked states is much larger than the coefficient $c_\beta = \frac{\sqrt{M}}{\sqrt{N}}$ associated with the marked states. Consequently, the initial state $|\psi\rangle$ is predominantly aligned along $|\alpha\rangle$, and the Grover iteration rotates the state vector slowly toward $|\beta\rangle$, allowing gradual amplitude amplification of the marked state.

It is convenient to introduce an angle θ defined by

$$\cos\frac{\theta}{2} = \frac{\sqrt{N-M}}{\sqrt{N}}, \qquad \sin\frac{\theta}{2} = \frac{\sqrt{M}}{\sqrt{N}}. \tag{6.9}$$

This angle quantifies the initial projection of $|\psi\rangle$ onto the marked subspace and will be useful for understanding Grover iteration geometrically.

Each Grover iteration consists of two main steps:

1. **Oracle Query (Phase Flip):** Apply the oracle U_f, which flips the phase of the marked state:
$$U_f|x\rangle = (-1)^{f(x)}|x\rangle. \tag{6.10}$$
Geometrically, this can be viewed as a reflection about the state $|\alpha\rangle$. For any state in the two-dimensional subspace spanned by $|\alpha\rangle$ and $|\beta\rangle$,
$$|\varphi\rangle = a|\alpha\rangle + b|\beta\rangle, \tag{6.11}$$
the action of the oracle gives
$$|\varphi'\rangle = U_f|\varphi\rangle = a|\alpha\rangle - b|\beta\rangle, \tag{6.12}$$
showing that the $|\beta\rangle$ component is reflected.
2. **Amplitude Amplification (Grover Diffusion Operator):** Apply the diffusion operator
$$D = 2|\psi\rangle\langle\psi| - I, \tag{6.13}$$

where $|\psi\rangle$ is the uniform superposition over all items as defined in Eq. (6.7). This operation can be interpreted as a reflection about $|\psi\rangle$ in the plane spanned by $|\alpha\rangle$ and $|\beta\rangle$. To see this, define

$$|\psi^{\perp}\rangle = -c_{\beta}|\alpha\rangle + c_{\alpha}|\beta\rangle, \tag{6.14}$$

where c_{α} and c_{β} are the coefficients of $|\psi\rangle$ in the $|\alpha\rangle$, $|\beta\rangle$ basis, so that $|\psi^{\perp}\rangle$ is orthogonal to $|\psi\rangle$ in the plane. Any state in this plane can be expressed as

$$|\chi\rangle = a|\psi\rangle + b|\psi^{\perp}\rangle. \tag{6.15}$$

Acting with D yields

$$|\chi'\rangle = D|\chi\rangle = a|\psi\rangle - b|\psi^{\perp}\rangle, \tag{6.16}$$

confirming that D performs a reflection about $|\psi\rangle$.

The combined Grover iteration operator is defined as

$$G = DU_f = (2|\psi\rangle\langle\psi| - I)\,U_f. \tag{6.17}$$

Within the two-dimensional subspace spanned by $|\alpha\rangle$ and $|\beta\rangle$, consider an initial state

$$|\Phi\rangle = \cos\frac{\theta}{2}\,|\alpha\rangle + \sin\frac{\theta}{2}\,|\beta\rangle, \tag{6.18}$$

whose angle from $|\alpha\rangle$ is $\theta/2$. Applying the oracle U_f first reflects the state about $|\alpha\rangle$, and subsequently applying D reflects the resulting state about $|\psi\rangle$. The composition of these two reflections produces a net rotation by an angle $3\theta/2$ toward $|\beta\rangle$.

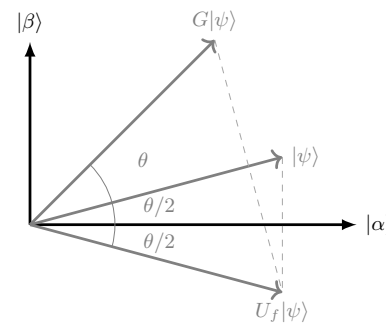


Fig. 6.1 Illustration of a Grover iteration. Each iteration rotates the state vector in the two-dimensional plane spanned by $|\alpha\rangle$ and $|\beta\rangle$, gradually increasing the amplitude of the marked state. Notice for Grover's search, the initial state is uniform superposition over all items $|\psi\rangle$.

In Grover's search, the initial state is the uniform superposition $|\psi\rangle$, which lies at an angle $\theta/2$ from $|\alpha\rangle$. Each application of G therefore increases this angle by θ, gradually amplifying the amplitude of $|\beta\rangle$ and increasing the probability of measuring the marked state.

Exercise 6.1. Verify that $G|\psi\rangle$ performs the rotation illustrated in Fig. 6.1. Show explicitly that the Grover iteration acts as a rotation matrix in the basis $\{|\alpha\rangle, |\beta\rangle\}$.

From Fig. 6.1 and Eq. (6.9), it is clear that θ becomes small when the ratio M/N is small. Consequently, each Grover iteration corresponds to a small but

precise rotation, making the algorithm particularly powerful for large databases with relatively few marked items.

6.1.2 Grover's Search Algorithm

The detailed procedure of Grover's search algorithm is given below.

1. **Initialization**
   - Prepare n qubits in the state $|0\rangle^{\otimes n}$.
   - Apply the Hadamard transform $H^{\otimes n}$ to obtain the equal superposition

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle. \tag{6.19}$$

2. **Grover iterations** Apply the Grover iteration operator G for $O(\sqrt{N})$ rounds. Each round consists of
   a. **Oracle** U_f, flipping the phase of the marked state.
   b. **Diffusion operator** D, performing amplitude amplification.
3. **Measurement** After approximately $O(\sqrt{N})$ iterations, the amplitude of the marked state becomes close to one. Measuring the qubits yields marked item x' with high probability.

To analyze the behaviour of the algorithm, let us assume for simplicity that there is exactly one marked item, $M = 1$. Initially all basis states have amplitude $\frac{1}{\sqrt{N}}$. After applying the oracle and the diffusion operator, the amplitude of the marked state increases, while the amplitudes of unmarked states decrease.

Each Grover iteration acts as a rotation in the two-dimensional subspace spanned by the unmarked-state superposition $|\alpha\rangle$ and the marked state $|\beta\rangle$. The rotation angle is

$$\theta \approx 2 \sin^{-1}\left(\frac{1}{\sqrt{N}}\right). \tag{6.20}$$

After k iterations, the state becomes

$$G^k|\psi\rangle = \cos\left(\frac{2k+1}{2}\theta\right)|\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right)|\beta\rangle. \tag{6.21}$$

Thus repeated applications of G rotate the initial state toward the marked state. The optimal number of iterations is

$$k_{\text{opt}} \approx \frac{\pi}{4}\sqrt{N}, \tag{6.22}$$

which maximizes the probability of obtaining $|\beta\rangle$ upon measurement. Applying G more times causes an overshoot and reduces the success probability.

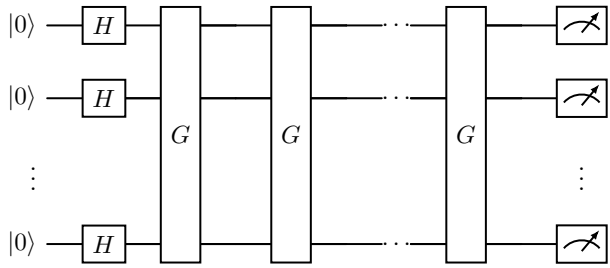


Fig. 6.2 Quantum circuit for Grover's search algorithm.

The quantum circuit for Grover's Algorithm consists of the following components.

- **Hadamard layer** The transform $H^{\otimes n}$ generates the initial equal superposition.
- **Grover iteration** G Each iteration is built from
  - the oracle U_f, performing a phase flip of the marked state
  - the diffusion operator D, a reflection about $|\psi\rangle$ implemented via Hadamard gates, Z gates and controlled operations
- **Measurement** Measuring the qubits yields the marked item with high probability.

Grover's search demonstrates how interference and amplitude amplification enable quantum mechanics to outperform classical strategies for unstructured search. Although the speedup is more modest than exponential, it remains a foundational example of quantum computational advantage.

7 Quantum Integral Transforms with Applications

> Beautiful mathematics eventually tends to be useful, and useful mathematics eventually tends to be beautiful.
>
> *From Meyer, Carl (2000) Matrix analysis and applied linear algebra*

Integral transforms such as Fourier transform, Laplace transform and so on, are powerful tools, they are ubiquitous in mathematics, engineering, physics and many other scientific areas. Mathematically, an integral transform T is an operator which maps a function f into another function Tf, and two functions do not necess.

In the previous chapter, we discussed quantum Algorithms that rely on query oracles. In this section, we shift our focus to Algorithms that do not primarily use query oracles but instead transform the problem into a quantum circuit. Key examples include the Quantum Fourier Transform, Quantum Phase Estimation, and Shor's Algorithm.

These circuit-based Algorithms are not only vital for their computational power but also for their wide-reaching applications in fields like cryptography, optimization, and machine learning. By challenging long-held assumptions about computational complexity, they provide new frameworks for addressing real-world problems. As we examine the inner workings of these Algorithms, we encourage you to experiment with practical implementations using Qiskit or actual quantum hardware, such as IBM's quantum computers.

§ 7.1 Quantum Discrete Integral Transform

The *integral transform* is a powerful mathematical tool that maps a function from its original domain into a new function space through an integration oper-

ation (or summation in the discrete case). Typical examples includs the Fourier transform, Laplace transform, Mellin transform, and many others, are indispensable across mathematics, physics, engineering, signal processing, quantum mechanics, and beyond. These transformation often reveals properties of the original function—such as frequency content, growth behavior, or analytic structure—that are more easily analyzed or manipulated in the transformed space. In most cases, the original function can be recovered exactly via an inverse transform.

Mathematically, an integral transform $\mathcal{T}$ is an operator that takes a function f and produces a new function $\mathcal{T}[f]$, which is often denoted by $\hat{f}$, $\tilde{f}$, or f^*, and in some contexts may simply be written as f with a different variable. A general integral transform is expressed using an *integral kernel* $K(p, x)$ as

$$\mathcal{T}[f](x) = \int_{-\infty}^{\infty} K(x, p)\, f(p)\, dp,$$

provided the integral exists. The form of this integral equation is the origin of the term "integral transform."

Among the vast family of integral transforms, the *Fourier transform* stands out for its unparalleled elegance and ubiquity. Any reader who has studied quantum mechanics will already know it intimately. In the natural units where $\hbar = 1$, physicists typically define the position-space wave function $\psi(x)$ from its momentum-space counterpart $\psi(p)$ by the *inverse* Fourier transform (chosen to avoid an overall minus sign in the exponent):

$$\psi(x) = \mathcal{F}[\psi](x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ipx} \psi(p)\, \mathrm{d}p.$$

The kernel of the inverse transform is simply the position–momentum overlap:

$$K(p, x) = \frac{e^{ipx}}{\sqrt{2\pi}} = \langle x|p\rangle,$$

where $|x\rangle$ and $|p\rangle$ are the usual position and momentum eigenstates. This single transformation lies at the very heart of quantum mechanics. It converts differential equations into algebraic ones, turns convolutions into products, and recasts purely local position-space information as global momentum content. Momentum-space methods often render seemingly intractable operator equations manifestly solvable.

The discussion that follows extends these classical ideas into the realm of quantum theory through the notion of *quantum integral transforms*. Since we are concerned with a discrete qubit system, our focus will be on *discrete integral transforms*. Recall that a function $f(x)$ can be regarded as a sequence of values indexed by x over a continuous domain. The corresponding integral transform $\mathcal{T}[f]$ may thus be viewed as a matrix transformation with continuous indices.

In the discrete setting, this analogy becomes exact: an integral transform is precisely realized as a matrix transform.

Definition 7.1 (Discrete integral transform). Let $K_{ij} =: K(i,j)$ be a matrix with indices $i,j = 0,\dots,N-1$, and let $\vec{x} = (x_0,\dots,x_{N-1})^T$ denote a vector, which can be regarded as a discrete function $x(i) = x_i$ defined on the domain $\{0,1,\dots,N-1\}$. The *discrete integral transform* is then defined by

$$y(i) = [Kx](i) = \sum_{j=0}^{N-1} K(i,j)\, x(j),$$

which, in matrix form, can be written as

$$y_i = \sum_{j=0}^{N-1} K_{ij}\, x_j. \tag{7.1}$$

Throughout, we assume that all vectors are column vectors, so that matrices act on them from the left. And we can assume K to be invertible, viz., $\det K \neq 0$.

For convenience in extending the discrete integral transform to the quantum realm, we assume that K is unitary. In this case, the inverse discrete integral transform is given by the Hermitian conjugate $K^\dagger$ of K.

Consider a Hilbert space $\mathbb{C}^N$ with basis states $|0\rangle, |1\rangle, \dots, |N-1\rangle$. Since the discrete integral transform is a linear map, it suffices to specify its action on the basis elements $|j\rangle$ with $j = 0,\cdots,N-1$. Given a discrete integral transform kernel K_{ij}, the corresponding *quantum discrete integral transform* acts on the basis states as (here we write $U_K = K$ to emphasize unitarity)

$$|j\rangle \overset{\text{QDIT}}{\longrightarrow} U_K|j\rangle = \sum_{k=0}^{N-1} K_{kj}\,|k\rangle. \tag{7.2}$$

The quantum discrete integral transform can thus be regarded as the quantum analogue of the discrete integral transform; indeed, as we shall see, they are essentially the same operation.

We can encode a classical complex vector $\vec{x}$ into a normalized quantum state

$$|\psi_{\vec{x}}\rangle = \frac{1}{\|\vec{x}\|} \sum_{j=0}^{N-1} x_j |j\rangle,$$

a procedure known as *amplitude encoding* in quantum data encoding, where $\|\vec{x}\|$ is L^2 norm. Applying the unitary U_K then realizes the discrete integral transform:

$$U_K|\psi_{\vec{x}}\rangle = \frac{1}{\|\vec{x}\|}\sum_{j=0}^{N-1} x_j\, U_K|j\rangle = \frac{1}{\|\vec{x}\|}\sum_{k=0}^{N-1}\Big(\sum_{j=0}^{N-1} K_{kj}x_j\Big)|k\rangle = \frac{1}{\|\vec{y}\|}\sum_{k=0}^{N-1} y_k|k\rangle,$$

where $y_k = \sum_{j=0}^{N-1} K_{kj}x_j$ is the discrete integral transform of $\vec{x}$. Since U_K is unitary, it preserves norms, so $\|\vec{x}\| = \|\vec{y}\|$, and the final quantum state $|\psi_{\vec{y}}\rangle$ is also normalized.

Definition 7.2 (Quantum discrete integral transform). The *quantum discrete integral transform* associated with a unitary kernel K_{kj} is the unitary operator $U_K = (K_{kj})$ acting as

$$|j\rangle \overset{\text{QDIT}}{\longrightarrow} U_K|j\rangle = \sum_{k=0}^{N-1} K_{kj}\,|k\rangle. \tag{7.3}$$

To carry out the discrete integral transform $y_k = \sum_{j=0}^{N-1} K_{kj}x_j$, one first prepares the normalized state $|\psi_{\vec{x}}\rangle = \sum_{j=0}^{N-1} x_j|j\rangle/\|\vec{x}\|$, then applies U_K to obtain $U_K|\psi_{\vec{x}}\rangle = |\psi_{\vec{y}}\rangle$, and finally measures in the computational basis $\{|k\rangle\}$ to extract the amplitudes y_k.

To implement a quantum discrete integral transform Algorithm, one must design a quantum circuit that realizes the unitary operator U_K. This step is generally highly nontrivial and requires careful thought and insightful design. Moreover, the structure of the quantum circuit depends sensitively on the choice of kernel K; different kernels typically lead to very different circuit constructions.

§ 7.2 Quantum Fourier Transform

In this section, we focus on the *quantum Fourier transform (QFT)*, which is the most fundamental and widely used example of a quantum discrete integral transform. The corresponding unitary matrix is $U_{\text{QFT}} = (K_{jk})$, where the kernel matrix is

$$K_{jk} = \frac{1}{\sqrt{N}} e^{2\pi i \frac{jk}{N}} = \frac{\omega_N^{jk}}{\sqrt{N}}$$

where $\omega_N = e^{\frac{2\pi i}{N}}$ is the N-th root of unit. We will construct a quantum circuit that efficiently implements the quantum Fourier transform.

Exercise 7.1. Prove that the kernel matrix

$$U_{\mathrm{QFT}} = K_{jk} = \frac{1}{\sqrt{N}} e^{2\pi i \frac{jk}{N}} = \frac{\omega_N^{jk}}{\sqrt{N}} \tag{7.4}$$

is unitary. You may find the following formula useful:

$$\sum_{a=0}^{N-1} \omega_N^{a(b-c)} = N\,\delta_{b,c}, \tag{7.5}$$

where $b, c \in \{0, \cdots, N-1\}$.

The *discrete Fourier transform* (DFT) is defined as

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i \frac{jk}{N}} x_j, \tag{7.6}$$

where x_j and y_k are complex numbers. which is ubiquitous in fields such as digital signal processing, speech recognition, image analysis, and data compression. It can transform a problem into some other problem for which the solution is easier to found.

The most efficient classical Algorithms for computing the discrete Fourier transform (DFT) on N elements are based on the fast Fourier transform (FFT), which has a computational complexity of $O(N \log N)$. In quantum computation, the DFT is implemented via the quantum Fourier transform, which can be realized on a quantum computer using only $O((\log N)^2)$ elementary gates. This exponential speedup over classical FFTs is a central feature of several landmark quantum Algorithms, including Shor's Algorithm for integer factorization and the quantum phase estimation subroutine.

As explained in Section 7.1, in quantum computing we consider the mapping of basis states $|j\rangle \mapsto U_{\mathrm{QFT}}|j\rangle$ with $j = 0, \ldots, N-1$, so that the focus is on the indices of $\vec{x}$ and $\vec{y}$. The quantum Fourier transform (QFT) is thus a unitary map on the computational basis states:

$$|j\rangle \;\mapsto\; \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i jk/N} |k\rangle, \tag{7.7}$$

or equivalently, for a general quantum state,

$$|\psi_{\vec{x}}\rangle = \sum_{j=0}^{N-1} x_j |j\rangle \;\mapsto\; |\psi_{\vec{y}}\rangle = \sum_{k=0}^{N-1} y_k |k\rangle. \tag{7.8}$$

The amplitude y_k after applying the unitary U_{QFT} corresponds to the discrete Fourier transform of $\vec{x}$ given in Eq. (7.6).

We will now assume that $N = 2^n$ is the number of basis states, then we can work in an n-qubit space $(\mathbb{C}^2)^{\otimes n}$. To understand the quantum Fourier transform effectively, it is essential to have a solid grasp of binary number operations, since we will treat the basis label of $|j\rangle = |j_1 j_2 \cdots j_n\rangle$ as a binary number. Suppose we represent an integer j in binary as $j = j_1 j_2 \cdots j_n$, which corresponds to

$$j = j_1 2^{n-1} + j_2 2^{n-2} + \cdots + j_n 2^0. \tag{7.9}$$

This implies that

$$\frac{j}{N} = 0.j_1 j_2 \cdots j_n = \frac{j_1}{2} + \frac{j_2}{2^2} + \cdots + \frac{j_n}{2^n}. \tag{7.10}$$

Essentially, the QFT exploits the binary representation of j in the phase factor $e^{2\pi i k j/N}$, which involves

$$k \times 0.j_1 j_2 \cdots j_n. \tag{7.11}$$

We only need to consider the fractional part, as the integer part contributes a factor of 1 to the exponential.

Now consider $k = k_1 k_2 \cdots k_n$ as a binary number, so that

$$k = k_1 2^{n-1} + k_2 2^{n-2} + \cdots + k_n 2^0. \tag{7.12}$$

Calculating the multiplication with $j/N = 0.j_1 j_2 \cdots j_n$, we have

$$\begin{aligned}
k_n 2^0 \times 0.j_1 j_2 \cdots j_n &= \begin{cases} 0, & k_n = 0, \\ 0.j_1 j_2 \cdots j_n, & k_n = 1, \end{cases} \\
k_{n-1} 2^1 \times 0.j_1 j_2 \cdots j_n &= \begin{cases} 0, & k_{n-1} = 0, \\ j_1.j_2 \cdots j_n, & k_{n-1} = 1, \end{cases} \\
&\vdots \\
k_1 2^{n-1} \times 0.j_1 j_2 \cdots j_n &= \begin{cases} 0, & k_1 = 0, \\ j_1 j_2 \cdots j_{n-1}.j_n, & k_1 = 1. \end{cases}
\end{aligned}$$

We then drop the integer part when $k_s = 1$ for all $s = 1, \cdots, n$ since their contributions for exponential factor are all one. This means that for the s-th qubit k_s, it contributes to the exponential $e^{2\pi i k j/N}$ as $1 = e^0$ if $k_s = 0$, and as $e^{2\pi i 0.j_{n-s+1} \cdots j_n}$ if $k_s = 1$ (note $0.j_{n-s+1} \cdots j_n \simeq 0.j_1 j_2 \cdots j_n \times 2^{n-s}$ up to discarding the integer parts).

Thus, the quantum Fourier transform on $|j\rangle$ can be written as

$$U_{\mathrm{QFT}}|j\rangle = \frac{(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle)\cdots(|0\rangle + e^{2\pi i 0.j_1 j_2\cdots j_n}|1\rangle)}{\sqrt{N}}. \tag{7.13}$$

Designing a quantum circuit to realize this transformation is the next step in constructing the quantum Fourier transform Algorithm.

Exercise 7.2. The Eq. (7.13) can also be derived from the following calculation:

$$\begin{aligned}
U_{\mathrm{QFT}}\,|j\rangle &= \frac{1}{\sqrt{N}}\sum_{k=0}^{N-1} e^{2\pi ijk/N}\,|k\rangle \\
&= \frac{1}{\sqrt{N}}\sum_{k_1=0}^{1}\cdots\sum_{k_n=0}^{1} e^{2\pi ij\left(\sum_{l=1}^{n} k_l 2^{-l}\right)}\,|k_1\cdots k_n\rangle \\
&= \frac{1}{\sqrt{N}}\sum_{k_1=0}^{1}\cdots\sum_{k_n=0}^{1}\bigotimes_{l=1}^{n} e^{2\pi ijk_l 2^{-l}}\,|k_l\rangle \\
&= \frac{1}{\sqrt{N}}\bigotimes_{l=1}^{n}\left[\sum_{k_l=0}^{1} e^{2\pi ijk_l 2^{-l}}\,|k_l\rangle\right] \\
&= \frac{1}{\sqrt{N}}\bigotimes_{l=1}^{n}\left[|0\rangle + e^{2\pi ij2^{-l}}\,|1\rangle\right] \\
&= \frac{\left(|0\rangle + e^{2\pi i 0.j_n}\,|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}\,|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i 0.j_1 j_2\cdots j_n}\,|1\rangle\right)}{\sqrt{N}}.
\end{aligned}$$

Take $n = 3$ as an example to check the expression for both derivations.

For convenience, let us denote

$$\begin{aligned}
|q_1\rangle &= \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\, 0.j_1 j_2\cdots j_n}|1\rangle\right), \\
|q_2\rangle &= \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\, 0.j_2 j_3\cdots j_n}|1\rangle\right), \\
&\vdots \\
|q_n\rangle &= \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\, 0.j_n}|1\rangle\right).
\end{aligned} \tag{7.14}$$

Using this notation, we have

$$U_{\mathrm{QFT}}\,|j_1\rangle|j_2\rangle\cdots|j_n\rangle = |q_n\rangle|q_{n-1}\rangle\cdots|q_1\rangle. \tag{7.15}$$

The reversed order of the indices in $|q_l\rangle$ is intentional.

The product form of the state after applying the quantum Fourier transform naturally guides the design of a quantum circuit that implements U_{QFT}, as illustrated in Figure 7.1. The circuit produces the output state

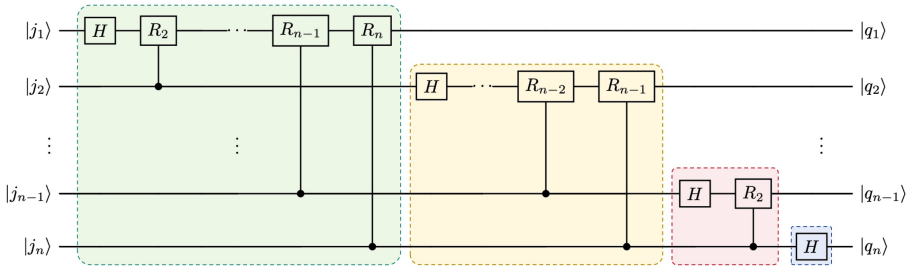


Fig. 7.1 The quantum circuit for quantum Frourier transform.

$$|j_1\rangle|j_2\rangle\cdots|j_n\rangle \longrightarrow |q_1\rangle|q_2\rangle\cdots|q_n\rangle, \tag{7.16}$$

after which a sequence of SWAP gates is applied to reverse the qubit order. To construct this circuit, we employ the Hadamard gate together with a series of controlled rotation gates $C(R_l)$, where each rotation gate R_l takes the form

$$R_l = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^l} \end{pmatrix}. \tag{7.17}$$

Although we will not use it here, notice that $R_1 = Z$ is the Pauli-Z operator, which flips the phase of the $|1\rangle$ basis.

We have already encountered several equivalent expressions for the Hadamard operator that play crucial roles in different context; here is yet another useful representation:

$$H|x\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i\, 0.x}|1\rangle\right). \tag{7.18}$$

The action of the controlled-R_l gate, denoted $C(R_l)$, is as follows:

$$\begin{aligned} |b\rangle \otimes |\psi\rangle \xrightarrow{C(R_l)} & \begin{cases} |0\rangle \otimes (\alpha|0\rangle + \beta|1\rangle), & b = 0, \\ |1\rangle \otimes (\alpha|0\rangle + \beta e^{2\pi i/2^l}|1\rangle), & b = 1, \end{cases} \\ & = |b\rangle \otimes \left(\alpha|0\rangle + \beta e^{2\pi i\, \frac{b}{2^l}}|1\rangle\right), \end{aligned} \tag{7.19}$$

where $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Thus, the controlled rotation $C(R_l)$ injects the control qubit's binary value b into the phase of the target qubit, contributing a factor of $e^{2\pi i\, \frac{b}{2^l}}$. With the above preparation, let us now examine the quantum circuit in Figure 7.1.

For the bottom quantum wire $|j_n\rangle$, the Hadamard operator gives (see Eq. (7.18))

$$|j_n\rangle \;\rightarrow\; |q_n\rangle. \tag{7.20}$$

For $|j_{n-1}\rangle$, the Hadamard gate contributes a phase factor $e^{2\pi i 0.j_{n-1}}$, and the controlled-R_2 gate controlled by j_n contributes $e^{2\pi i 0.0 j_n}$, so that the final phase factor is $e^{2\pi i\, 0.j_{n-1}j_n}$. The state of this quantum wire becomes

$$|j_{n-1}\rangle \;\rightarrow\; |q_{n-1}\rangle. \tag{7.21}$$

For $|j_{n-2}\rangle$, the Hadamard gate contributes a phase factor $e^{2\pi i 0.j_{n-2}}$, the controlled-R_2 gate controlled by j_{n-1} contributes $e^{2\pi i 0.0 j_{n-1}}$, and the controlled-R_3 gate controlled by j_n contributes $e^{2\pi i 0.00 j_n}$. The final phase factor is thus $e^{2\pi i\, 0.j_{n-2}j_{n-1}j_n}$, and the state of this quantum wire becomes

$$|j_{n-2}\rangle \;\rightarrow\; |q_{n-2}\rangle. \tag{7.22}$$

This pattern continues similarly for the remaining qubits. We see that the quantum circuit in Figure 1 realizes the quantum Fourier transform.

Exercise 7.3. Take $n = 3$ as an example the check the quantum states after the quantum Fourier transform.

7.2.1 The inverse quantum Fourier transform

The inverse quantum Fourier transform $U_{\mathrm{QFT}}^{\dagger}$ is the Hermitian conjugate of the quantum Fourier transform. It reverses the action of the QFT by converting phase-encoded states back into computational basis states:

$$U_{\mathrm{QFT}}^{\dagger}|k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{-2\pi i \frac{jk}{N}} |j\rangle, \tag{7.23}$$

where we use the convention $[U_{\mathrm{QFT}}]_{ab} = e^{2\pi i ab/N}/\sqrt{N}$.

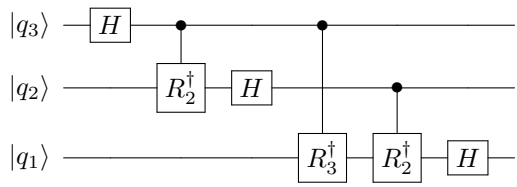
Exercise 7.4. Verify that $U_{\mathrm{QFT}}^{\dagger} = U_{\mathrm{QFT}}^{-1}$ using Eq. (7.5) in Exercise 7.1.

The inverse QFT transforms a phase-encoded superposition back into a definite computational basis state:

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{2\pi i x\theta/N} |x\rangle \;\mapsto\; |\theta\rangle, \tag{7.24}$$

where the phase factor $e^{2\pi i x\theta/N}$ carries the information about an unknown parameter θ. Applying the inverse quantum Fourier transform thus yields a state sharply peaked around the binary representation of θ, so that a measurement provides an accurate estimate of θ. This mechanism underlies many applications of the quantum Fourier transform, including quantum phase estimation and related Algorithms.

Exercise 7.5. The quantum circuit implementing the inverse quantum Fourier transform is simply the Hermitian conjugate of the circuit shown in Figure 7.1. Note that $C(R_l)^\dagger = C(R_l^\dagger)$ and $H^\dagger = H$. Verify that the three-qubit circuit takes the following form:



Try to generalize this construction and draw the circuit for the n-qubit case.

§ 7.3 Quantum Phase Estimation Algorithm

A key application of the quantum Fourier transform is the *Quantum Phase Estimation* Algorithm. This Algorithm allows one to estimate the phase (or eigenvalue) of a unitary operator and serves as a central component of many quantum Algorithms, including Shor's factoring Algorithm and quantum simulation.

Consider a unitary matrix U. Since $U^\dagger U = I$, all eigenvalues of U have the form $e^{2\pi i\theta}$ with $0 \le \theta < 1$. Expressing θ in binary gives

$$\theta = 0.\theta_1\theta_2\theta_3\cdots = \sum_{k=1}^{\infty} \theta_k\, 2^{-k}, \tag{7.25}$$

where each $\theta_k \in \{0, 1\}$. The task is to estimate θ to n binary digits of precision.

More precisely, the quantum phase estimation Algorithm aims to determine the phase θ in the eigenvalue equation

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle, \tag{7.26}$$

where U is the given unitary operator, $|\psi\rangle$ is an eigenstate of U that can be prepared as an input state, and θ is the phase to be determined. We further assume that controlled-U^{2^l} operations can be implemented for arbitrary non-negative integer l. The goal is to estimate θ to the desired number of digits with high probability.

Now suppose that we express θ in n binary digits,

$$\theta = 0.\theta_1\theta_2\cdots\theta_n.$$

Then

$$U^{2^l}|\psi\rangle = e^{2\pi i\, 0.\theta_1\theta_2\cdots\theta_n\times 2^l}|\psi\rangle = e^{2\pi i\,\theta_1\cdots\theta_l.\theta_{l+1}\cdots\theta_n}|\psi\rangle, \tag{7.27}$$

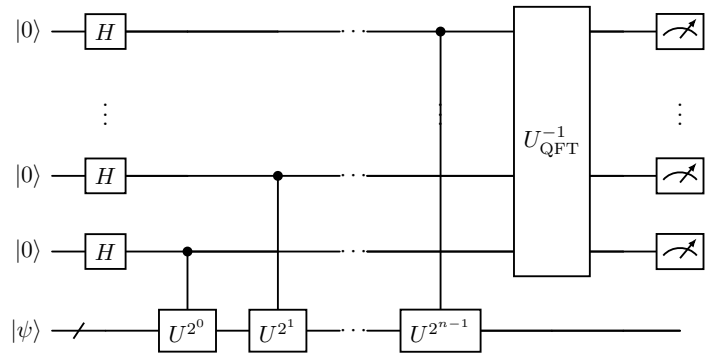


Fig. 7.2 The quantum circuit for quantum phase estimation Algorithm.

where the integer part of $\theta_1 \cdots \theta_l.\theta_{l+1} \cdots \theta_n$ can be omitted, as it contributes only a trivial phase factor of 1.

For the controlled unitary $C(U^{2^l})$, its action on $|+\rangle|\psi\rangle$ is (with $|+\rangle$ the control qubit):

$$\begin{aligned} C(U^{2^l})\frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big)|\psi\rangle &= \frac{1}{\sqrt{2}}|0\rangle \otimes |\psi\rangle + e^{2\pi i 0.\theta_{l+1}\cdots\theta_n}|1\rangle \otimes |\psi\rangle \\ &= \frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.\theta_{l+1}\cdots\theta_n}|1\rangle\big) \otimes |\psi\rangle. \end{aligned} \tag{7.28}$$

It is convenient to view this control qubit as representing the $(l+1)$-th digit of a binary number

$$k = k_1 k_2 \cdots k_n = k_1 \times 2^0 + k_2 \times 2^1 + \cdots + k_n \times 2^{n-1}.$$

With this notation, the above expression can be rewritten as

$$C(U^{2^l})\frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big)|\psi\rangle = \left(\frac{1}{\sqrt{2}}\sum_{k_{l+1}=0}^{1} e^{2\pi i \theta\, k_{l+1} 2^l}|k_{l+1}\rangle\right)|\psi\rangle. \tag{7.29}$$

This expression for the first qubit is familiar from our earlier discussion of the quantum Fourier transform, and it naturally suggests that the quantum Fourier transform should be applied here. The eigenstate $|\psi\rangle$ therefore remains unchanged and can be reused in subsequent steps.

The quantum phase estimation Algorithm uses two quantum registers as shown in Figure 7.2:

- First register (phase record): Contains n qubits, which will store the estimated phase.

- Second register (eigenstate): Contains the eigenstate $|\psi\rangle$, which is an eigenstate of the unitary operator U.

The quantum phase estimation Algorithm consists of the following steps:

1. **Prepare Initial State**
   The system starts with two registers:
   - The first register is initialized in the state $|0\rangle^{\otimes n}$.
   - The second register is initialized in the eigenstate $|\psi\rangle$ of U.

   The initial state is: $|0\rangle^{\otimes n} \otimes |\psi\rangle$
2. **Apply Hadamard Gates**
   Apply Hadamard gates to each qubit in the first register, thereby creating a uniform superposition over all possible computational basis states:

$$\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \otimes |\psi\rangle.$$

   Here, $k = k_1 k_2 \cdots k_n$ denotes the binary representation of the integer k, with the bits encoded from the bottom to the top qubit in the first register.
3. **Apply Controlled-U^{2^l} Operations**
   For each qubit k_l in the first register, apply the controlled-U^{2^l} operation. This step entangles the first and second registers, yielding the state

$$\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \otimes U^k |\psi\rangle.$$

   Since $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, this becomes

$$\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi ik\theta} |k\rangle \otimes |\psi\rangle.$$

   At this stage, the second register is no longer needed, as all phase information is now encoded in the first register.
4. **Apply the Inverse Quantum Fourier Transform (QFT)**
   Apply the inverse quantum Fourier transform ($\text{QFT}^\dagger$) on the first register to extract the phase information:

$$\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi ik\theta} |k\rangle$$

5. **Measure the First Register**

Finally, measure the qubits in the first register. The measurement will yield the binary representation of the phase θ with an n-bit approximation. The output will be an n-bit approximation of the phase θ, such that:

$$\theta \approx 0.\theta_1\theta_2 \dots \theta_n$$

where $\theta_1, \theta_2, \dots, \theta_n$ are the binary digits of θ.

Exercise 7.6. For the above three qubit QPE, calculate the explicit output state and show that they give the estimated phase.

The QPE Algorithm estimates the phase θ with a high probability of success. If θ can be exactly represented with n bits, the Algorithm will output the correct result with probability 1. If θ cannot be exactly represented, the Algorithm will output the closest n-bit approximation with a high probability.

Quantum Phase Estimation (QPE) has many applications and, like the Quantum Fourier Transform (QFT), is often used as a subroutine. The Quantum Phase Estimation Algorithm is a key quantum Algorithm that estimates the phase corresponding to an eigenvalue of a unitary operator. It is widely used in Algorithms for factoring, quantum simulations, and quantum chemistry.

8 Shor's Factoring Algorithm

In this chapter, we discuss the Shor's factoring algorithm which can be used to break the RSA (Rivest–Shamir–Adleman) public-key cryptosystem.

§ 8.1 Quantum Order-Finding Algorithm

The order-finding problem is a fundamental problem that plays a central role in several quantum Algorithms; in particular, it underlies Shor's celebrated factoring Algorithm. Classically, order-finding is hard—no efficient classical Algorithm is known—but quantum computers can solve it efficiently. Efficiently determining the order of an integer modulo N allows integers to be factored in polynomial time, which would compromise RSA encryption, as we will discuss in the subsequent sections.

The order-finding problem is defined rigorously as follows:

Definition 8.1 (Order-Finding Problem). Let $N \in \mathbb{Z}_+$ be a positive integer, and let $a \in \mathbb{Z}_N^\times$ be an integer coprime to N (that is, the greatest common divisor $\gcd(a, N)$ of a and N equals one, $\gcd(a, N) = 1$)[1]. The *order of a modulo N*, denoted $\mathrm{ord}_N(a)$ or simply r, is the smallest *positive* integer r such that

$$a^r \equiv 1 \pmod{N}. \tag{8.1}$$

The *order-finding problem* is: given a and N, compute $r = \mathrm{ord}_N(a)$.

At first glance, this definition may appear somewhat abstract. Do not worry—shortly, we will unpack each of the mathematical ingredients and illustrate their meaning with concrete examples.

[1] The notation $\mathbb{Z}_N^\times$ denotes the set of invertible elements in $\mathbb{Z}_N$ under multiplication—those integers that are coprime to N.

The quantum Algorithm for order-finding exploits the periodic structure of the function

$$f_{a,N}(x) \equiv a^x \pmod{N}, \tag{8.2}$$

which is periodic with period exactly r. By preparing a quantum superposition over all possible exponents and applying the quantum phase estimation Algorithm to the unitary operator

$$U|y\rangle = |ay \pmod{N}\rangle, \tag{8.3}$$

we obtain an efficient quantum procedure that succeeds with high probability.

8.1.1 Preliminaries of Modular Aritheoremetic

Before presenting the complete quantum Algorithm, let us first review a few essential results from number theory. This subsection may appear somewhat technical compared to others; if you prefer not to delve into the details, simply remember that for coprime integers a and N, there exists a positive order r, and the function in Eq. (8.2) is periodic.

An integer $N > 1$ is called a *prime number* if it has no nontrivial divisors; for example, $N = 2, 3, 5, 7, 11, \ldots$ are prime numbers. Otherwise, N is called a *composite number*. We use $a \mid b$ to denote that b is divisible by a, for instance $3 \mid 12$, $2 \mid 6$. Similarly, $a \nmid b$ means that a does not divide b, for example $3 \nmid 14$.

Every integer $N > 1$ can be uniquely expressed as a product of prime powers:

$$N = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_m^{e_m}, \tag{8.4}$$

where $p_1, \ldots, p_m$ are distinct prime numbers and $e_1, \ldots, e_m$ are positive integers. This result is known as the *Fundamental Theorem of Aritheoremetic*.

To obtain the prime factorization of a given number N, one typically follows these steps:

1. Divide N by 2 repeatedly until the quotient is no longer divisible by 2. Let e_2 be the number of times we divide by 2, and denote the resulting quotient by $N_2 = N/2^{e_2}$, for which $2 \nmid N_2$.
2. For the quotient N_2, proceed with the next prime, 3, and divide repeatedly until the quotient is no longer divisible by 3. Let e_3 be the number of divisions, and denote the resulting quotient by $N_3 = N_2/3^{e_3}$.
3. Continue in this manner with successive primes $5, 7, 11, \ldots$ until the quotient becomes 1.

Exercise 8.1. Decompose $N = 84$ into its prime factors: $84 = 2^2 \times 3 \times 7$.

The key operation we will use is modular aritheoremetic. In everyday life we count linearly: 1, 2, 3, ..., in modular aritheoremetic we count on a circle.

Take a clock that shows only 12 hours. When the hand reaches 12, it jumps back to 0. Mathematically, we say time is measured modulo 12. This simple idea—wrapping around after a fixed modulus N—is the crucial in number theory and its application in modern public-key cryptography.

Definition 8.2 (Congruence). Fix an integer $N \geq 2$, called the *modulus*. Two integers a and b are *congruent modulo* N if their difference is a multiple of N:

$$a \equiv b \pmod{N} \quad \overset{\text{def}}{\iff} \quad N \mid (a-b) \quad \iff \quad \exists\, k \in \mathbb{Z} \text{ such that } a = b + kN.$$

We read $a \equiv b \pmod{N}$ as "a is congruent to b modulo N".

The congruence class of a is usually represented by its smallest non-negative residue:

$$a \mod N \in \mathbb{Z}_N := \{0, 1, 2, \ldots, N-1\}.$$

It is also convenient to use $[a]$ to denote congruence class whenever there is not ambiguity of modulus.

Example 8.1. Let $N = 7$, we have $\mathbb{Z}_7 := \{0, 1, 2, \ldots, 6\}$. Then $1 \equiv 8 \pmod{N}$, $2 \equiv 9 \pmod{N}$. The congruence class $[0]$ consists of $\{0, 7, 14, 21, \cdots\}$, the congruence class $[1]$ consists of $\{1, 8, 15, 22, \cdots\}$, and $[2]$ consists of $\{2, 9, 16, 23, \cdots\}$, etc.

The order-finding problem considers the modular exponentiation for given integers a and N:

$$a^x \equiv b \pmod{N}. \tag{8.5}$$

The remainder b can take values in $0, 1, \ldots, N-1$:

$$\begin{aligned} a &\equiv 0 \pmod{N}, \\ a &\equiv 1 \pmod{N}, \\ &\vdots \\ a &\equiv N-1 \pmod{N}. \end{aligned}$$

However, not all values of $b = 0, 1, \ldots, N-1$ are necessarily attainable. For example, let $a = 2$ and $N = 6$. We find that

$$\begin{aligned} 2^0 &\equiv 1 \pmod{6}, \\ 2^1 &\equiv 2 \pmod{6}, \\ 2^2 &\equiv 4 \pmod{6}, \\ 2^3 &\equiv 2 \pmod{6}, \\ 2^4 &\equiv 4 \pmod{6}, \\ 2^5 &\equiv 2 \pmod{6}, \\ &\vdots \end{aligned}$$

We observe a repeating pattern. In this case, the residues 0, 3, and 5 never appear.

Note also that $a^0 \equiv 1 \pmod{N}$ for any integers a and N. Hence, the *order* of a modulo N is defined as the smallest positive integer r such that $a^r \equiv 1 \pmod{N}$. However, we must still ensure that such an order actually exists. In fact, if $\gcd(a, N) = d > 1$, then no positive integer r satisfies $a^r \equiv 1 \pmod{N}$. This can be seen in the previous example with $a = 2$ and $N = 6$, where the sequence never returns to 1 modulo 6.

Exercise 8.2. Let $N > 1$ and $\gcd(a, N) = d > 1$. Then the only integer r satisfying

$$a^r \equiv 1 \pmod{N}$$

is $r = 0$.

Proof. Suppose there exists an integer r such that $a^r \equiv 1 \pmod{N}$. Then N divides $a^r - 1$, so in particular

$$d \mid (a^r - 1).$$

Since $d \mid a$, we also have $d \mid a^r$, and therefore

$$d \mid (a^r - 1) \implies d \mid 1,$$

which contradicts $d > 1$. Hence no such $r \neq 0$ exists.

For $r = 0$, the convention $a^0 = 1$ gives

$$a^0 = 1 \equiv 1 \pmod{N}$$

trivially. Thus $r = 0$ is the only solution. □

In Shor's Algorithm, we either assume $\gcd(a, N) = 1$, or first compute the greatest common divisor classically, which already yields a nontrivial factor of N if $d > 1$.

Another useful concept we will use is *Euler's Totient function*: $\varphi(N)$ counts the number of integers $k \in \{1, 2, \ldots, N-1\}$ coprime to N. For example, $\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(3) = 2$, $\varphi(4) = 2$, etc. For fixed N, consider $\mathbb{Z}_N$, we call $a \in \mathbb{Z}_N$ invertible is there is an integer c such that $ac \equiv 1 \pmod{N}$.

Proposition 8.1 (Bézout's Identity). *Let a and b be integers with greatest common divisor $d = \gcd(a, b)$. Then there exist integers x and y such that*

$$ax + by = d.$$

Moreover, the integers of the form $az + bt$ are exactly the multiples of d.

The set $\mathbb{Z}_N$ is not a group under multiplication modulo N in general. We denote by $(\mathbb{Z}_N^\times, \times)$ the set of invertible elements, which indeed forms a group.

By Bézout's identity, $\mathbb{Z}_N^\times$ consists of all integers coprime to N, and this group has order $\phi(N)$, where ϕ denotes Euler's totient function.

From a group-theoretic perspective, for any $a \in \mathbb{Z}_N^\times$, there exists a positive integer r such that $a^r \equiv 1 \pmod{N}$. Since a must be invertible, it follows that $\gcd(a, N) = 1$. In the multiplicative group $\mathbb{Z}_N^\times$, the order of any element divides the order of the group, that is,

$$r \mid |\mathbb{Z}_N^\times| = \phi(N).$$

Theorem 8.1 (Euler's Theorem). *If* $\gcd(a, N) = 1$*, then*

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

and hence the order of a *modulo* N *divides* $\phi(N)$*.*

By Euler's theorem, for integers a and N with $\gcd(a, N) = 1$, there always exists a positive integer r such that

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

the order $r = \mathrm{ord}_N(a)$ always satisfies $1 \leq r \leq \phi(N)$.

Since in the cyclic subgroup generated by $[a] \in \mathbb{Z}_N^\times$ we have

$$\{[a^0], [a^1], [a^2], \ldots, [a^{r-1}]\}, \quad \text{with } [a^r] = [a^0] = [1],$$

the sequence $a^n \bmod N$ is periodic with period equal to the order r:

$$\begin{aligned}
a^0 &\equiv 1 \pmod{N},\\
a^1 &\equiv a \pmod{N},\\
&\vdots\\
a^{r-1} &\not\equiv 1 \pmod{N},\\
a^r &\equiv 1 \pmod{N},\\
a^{r+1} &\equiv a \pmod{N},\\
a^{r+2} &\equiv a^2 \pmod{N},\\
a^{r+3} &\equiv a^3 \pmod{N}.
\end{aligned}$$

Thus, the sequence repeats with period r.

For example set $a = 2$ and $N = 7$ we have $f_{a,N}(x) = 2^x \pmod{7}$ as:

$$
\begin{aligned}
2^0 &\equiv 1 \pmod 7 \\
2^1 &\equiv 2 \pmod 7 \\
2^2 &\equiv 4 \pmod 7 \\
2^3 &\equiv 8 \equiv 1 \pmod 7 \\
2^4 &\equiv 16 \equiv 2 \pmod 7 \\
2^5 &\equiv 32 \equiv 4 \pmod 7 \\
2^6 &\equiv 64 \equiv 1 \pmod 7 \\
2^7 &\equiv 128 \equiv 2 \pmod 7 \\
2^8 &\equiv 256 \equiv 4 \pmod 7
\end{aligned}
$$

we see the order is $r = \mathrm{ord}_7(2) = 3$.

8.1.2 Quantum Order-Finding Algorithm

The quantum order-finding Algorithm can be divided into two main steps:

1. Encode the order into the phase of an eigenvalue of a suitable unitary operator, thereby reducing the order-finding problem to a quantum phase estimation problem.
2. Recover the order r from the estimated phase φ using the method of continued fractions.

Let us first examine how to encode the order into the phase of a unitary operator.

Fix integers a and N such that $\gcd(a, N) = 1$. Define the unitary operator $U_{a,N}$ acting on computational basis states $|y\rangle$ by

$$U_{a,N} |y\rangle = |ay \bmod N\rangle , \quad y = 0, 1, \ldots, N-1. \tag{8.6}$$

We will see that the order can be map to the phase of an eigenvalue of $U_{a,N} |y\rangle$

Proposition 8.2. *Let $N \geq 2$ and let a be an integer coprime to N. The operator $U_{a,N}$ on $\mathbb{C}^N$ defined by*

$$U_{a,N} |y\rangle = |ay \bmod N\rangle , \quad y = 0, 1, \ldots, N-1, \tag{8.7}$$

is unitary.

Proof. Since $\gcd(a, N) = 1$, multiplication by a modulo N defines a bijection on the set $\{0, 1, \ldots, N-1\}$. Thus $U_{a,N}$ merely permutes the computational basis $\{|0\rangle , |1\rangle , \ldots , |N-1\rangle\}$. Any permutation of an orthonormal basis extends to a unitary operator, so $U_{a,N}$ is unitary.

Alternatively, suppose $U_{a,N}|y\rangle = U_{a,N}|y'\rangle$. Then

$$ay \equiv ay' \pmod N \implies a(y-y') \equiv 0 \pmod N \implies N \mid a(y-y').$$

Because $\gcd(a,N)=1$, we may cancel a (more precisely, multiply both sides by the modular inverse of a modulo N) to obtain $N \mid (y-y')$. Since $0 \le y, y' < N$, the only possibility is $y - y' = 0$, so $y = y'$. Thus $U_{a,N}$ is injective on a basis of $\mathbb{C}^N$. Being an isometry that maps a basis to a basis, it is unitary. □

Proposition 8.3. *Let r be the order of a modulo N, i.e., the smallest positive integer such that $a^r \equiv 1 \pmod N$. For each $s = 0, 1, \ldots, r-1$, define the state*

$$|u_s\rangle := \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i s k/r} \left|a^k \bmod N\right\rangle. \tag{8.8}$$

For each $s = 0, 1, \ldots, r-1$, $|u_s\rangle$ is an eigenstate of $U_{a,N}$ with eigenvalue $e^{2\pi i s/r}$.

Proof. Apply $U_{a,N}$ to $|u_s\rangle$:

$$\begin{aligned}
U_{a,N}|u_s\rangle &= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i s k/r} U_{a,N} \left|a^k \bmod N\right\rangle \\
&= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i s k/r} \left|a^{k+1} \bmod N\right\rangle \\
&= \frac{1}{\sqrt{r}} \sum_{k=1}^{r} e^{-2\pi i s (k-1)/r} \left|a^k \bmod N\right\rangle \\
&\qquad \text{(reindex the sum by letting } k \mapsto k+1\text{)} \\
&= \frac{1}{\sqrt{r}} \sum_{k=1}^{r} e^{-2\pi i s k/r} e^{2\pi i s/r} \left|a^k \bmod N\right\rangle \\
&= e^{2\pi i s/r} \frac{1}{\sqrt{r}} \sum_{k=1}^{r} e^{-2\pi i s k/r} \left|a^k \bmod N\right\rangle.
\end{aligned}$$

The sum now runs from $k = 1$ to r, but the $k = r$ term is

$$e^{-2\pi i s r/r} |a^r \bmod N\rangle = e^{-2\pi i s} |1 \bmod N\rangle = |1 \bmod N\rangle,$$

while the $k = 0$ term in the original definition of $|u_s\rangle$ is

$$e^0 \left|a^0 \bmod N\right\rangle = |1 \bmod N\rangle.$$

Thus the missing $k = 0$ term is exactly replaced by the $k = r$ term, and we obtain

$$U_{a,N}\,|u_s\rangle = e^{2\pi i s/r}\frac{1}{\sqrt{r}}\sum_{k=0}^{r-1} e^{-2\pi i s k/r}\,\left|a^k \bmod N\right\rangle$$
$$= e^{2\pi i s/r}\,|u_s\rangle\,.$$

This holds for every $s = 0, 1, \ldots, r-1$. □

Example 8.2. Let $N = 5$ and $a = 2$. Then the action of $U_{2,5}$ is

$$U_{2,5}\,|0\rangle = |0\rangle\,,$$
$$U_{2,5}\,|1\rangle = |2\rangle\,,$$
$$U_{2,5}\,|2\rangle = |4\rangle\,,$$
$$U_{2,5}\,|3\rangle = |1\rangle\,,$$
$$U_{2,5}\,|4\rangle = |3\rangle\,.$$

Hence, the matrix representation of $U_{2,5}$ in the computational basis is

$$U_{2,5} = \begin{pmatrix} 1&0&0&0&0\\ 0&0&0&1&0\\ 0&1&0&0&0\\ 0&0&0&0&1\\ 0&0&1&0&0 \end{pmatrix}.$$

Now the procedure becomes straightforward. We apply quantum phase estimation to the operators $U_a^{2^j}$ with the eigenstate $|u_s\rangle$, which yields the phases

$$\theta_s = \frac{s}{r}, \quad s = 0, 1, \ldots, r-1.$$

The output of the phase estimation Algorithm is thus

$$0.\theta_0\theta_1\ldots\theta_n,$$

represented as binary numbers. A classical Algorithm called continues fraction can be applied to obtain the order r.

8.1.3 Recovering the Order r from the Measured Phase

In the order-finding Algorithm, the quantum Fourier transform yields a phase $\theta \approx s/r$, where $s \in \{0, 1, \ldots, r-1\}$ is random and r is the unknown order of a modulo N. Since $r \leq N$, it suffices to find the best rational approximation to θ with denominator at most N.

This is achieved using the **continued-fraction expansion** of θ. Any real number $x \in [0, 1)$ can be written as

$$x = [a_0; a_1, a_2, a_3, \dots] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}, \qquad a_0 = 0,\ a_k \in \mathbb{N}^+.$$

The k**-th convergent** h_k/k_k is the rational

$$\frac{h_k}{k_k} = [a_0; a_1, \dots, a_k].$$

A fundamental theorem guarantees that if $|x - p/q| < 1/(2q^2)$ and $q \le N$, then p/q appears as a convergent of x.

Thus, computing the continued-fraction expansion of the measured θ and inspecting its convergents with denominator $\le N$ reveals the order r.

Example 8.3 (Finding the order of $a = 2$ modulo $N = 21$). The order of 2 modulo 21 is $r = 12$, but suppose the QFT returned

$$\theta \approx 0.1562 \approx \frac{5}{32}.$$

The continued-fraction expansion is

$$\frac{5}{32} = [0; 6, 2, 2, 2].$$

The convergents are:

$$\begin{aligned}
[0] &= \frac{0}{1}, && \text{denominator } 1\\
[0;6] &= \frac{1}{6}, && \text{denominator } 6\\
[0;6,2] &= \frac{2}{13}, && \text{denominator } 13\\
[0;6,2,2] &= \frac{5}{32}, && \text{denominator } 32 > 21.
\end{aligned}$$

Since $32 > 21$, we discard the last convergent and test the previous one: $13 \le 21$. Checking powers of 2 modulo 21 confirms

$$2^{13} = 8192 \equiv 1 \pmod{21},$$

so $r = 13$ is indeed a divisor of the true order 12. Repeating the quantum routine with a different s will eventually yield $r = 12$ directly (e.g. via $\theta \approx 7/12 = [0; 1, 2, 1, 2]$).

Algorithm summary (post-processing):

1. Measure phase $\theta \in [0, 1)$.

2. Compute continued-fraction expansion of θ.
3. List all convergents p/q with $q \leq N$.
4. For each such q, check whether $a^q \equiv 1 \pmod{N}$.
5. The smallest positive q satisfying the congruence is the order r.

§ 8.2 Shor's Algorithm

Shor's algorithm, introduced by Peter Shor in 1994, is the breakthrough quantum algorithm that factors large integers in polynomial time, a task widely believed to be intractable for classical computers. Given an integer N, the algorithm finds its prime factors in $O\big((\log N)^2(\log\log N)(\log\log\log N)\big)$ time on a quantum computer. This is exponentially faster than the best classical algorithms, such as the *general number field sieve*, which runs in sub-exponential time $O\Big(e^{\,c\,(\log N)^{1/3}(\log\log N)^{2/3}}\Big)$, where $c = \frac{1}{3}(92 + 26\sqrt{13})^{1/3}$. It is one of the most famous and influential algorithms in quantum computing. The discovery of Shor's algorithm demonstrated that a sufficiently large quantum computer could break widely used public-key cryptosystems, most notably RSA, whose security relies on the assumed hardness of integer factorization. This insight launched both the global effort to build scalable quantum computers and the rapid development of post-quantum cryptography, which aims to design cryptographic protocols secure even against quantum adversaries.

The algorithm combines classical number theory with a powerful quantum subroutine. It begins by reducing the factoring problem to the task of finding the period r of the function $f(x) = a^x \bmod N$ for a randomly chosen a coprime to N. This period-finding step is solved efficiently using quantum phase estimation. Once an even period r is obtained, the factors of N can be recovered classically as $\gcd(a^{r/2} \pm 1, N)$. The exponential speedup arises from the ability of quantum phase estimation and the quantum Fourier transform to extract the global periodicity in a single parallel operation—an inherently quantum phenomenon with no known efficient classical counterpart.

8.2.1 RSA Cryptosystem

Shor's algorithm was primarily designed to break the RSA public-key cryptosystem. For completeness, we first review the core principles of RSA. The RSA cryptosystem, along with its variants, is one of the most widely deployed methods for securely transmitting messages over public channels, such as the Internet, where it protects confidentiality and enables digital signatures. Its security relies fundamentally on a single, long-standing computational assumption:

that factoring a large composite integer into its prime factors is extraordinarily difficult for classical computers.

It is important to note that testing whether a number is prime is computationally efficient, while factoring a large composite number appears to be prohibitively hard on classical hardware. RSA takes advantage of precisely this asymmetry. Although the primes used to generate the key are easy to verify individually, recovering them from the public modulus is, for classical machines, effectively infeasible. This one-way computational gap is the foundation of RSA security and enables secure encryption and decryption.

The RSA cryptosystem operates as follows:

1. Alice prepares two very large prime numbers p and q, which she keeps secret. She computes $N = pq$ and publishes N. She also chooses a number, called the public exponent (or encoding exponent), $e < N$, which is required to be relatively prime to $(p-1)(q-1)$, that is, $\gcd(e,(p-1)(q-1)) = 1$. Alice then computes the multiplicative inverse of e modulo $(p-1)(q-1)$, namely a number d satisfying

$$de \equiv 1 \pmod{(p-1)(q-1)}. \tag{8.9}$$

   She keeps d secret, thus called secret exponent (or decoding exponent).
2. When Bob wants to send a message to Alice, he first encodes the message as an integer m with $m < N$. He encrypts it as

$$M = m^e \pmod{N}. \tag{8.10}$$

3. Upon receiving M, Alice decrypts it using

$$m \equiv M^d \pmod{N}, \tag{8.11}$$

   where d is the multiplicative inverse of the public exponent e modulo $(p-1)(q-1)$.

To see that the RSA cryptosystem correctly recovers the message m using Eq. (8.11), we need to check

$$m \equiv (m^e)^d \pmod{N}. \tag{8.12}$$

From Eq. (8.9), there is an integer k such that $de = k(p-1)(q-1)+1$. Hence

$$(m^e)^d = m\left[m^{k(q-1)}\right]^{p-1}. \tag{8.13}$$

Suppose m is not a multiple of p. Fermat's little theorem gives

$$\left[m^{k(q-1)}\right]^{p-1} \equiv 1 \pmod{p}. \tag{8.14}$$

If m is a multiple of p, then $m^{de} \equiv 0 \pmod{p}$. Similarly,

$$\left[m^{k(p-1)}\right]^{q-1} \equiv 1 \pmod{q}, \tag{8.15}$$

when m is not a multiple of q, and $m^{de} \equiv 0 \pmod{q}$ if m is a multiple of q.

We now verify Eq. (8.12) by considering several cases. (1) If m is a multiple of both p and q, then it is automatically a multiple of $N = pq$, and Eq. (8.12) holds trivially. (2) If m is a multiple of p but not of q, then

$$m \equiv 0 \pmod{p}, \qquad m^{de} \equiv 0 \pmod{p}, \tag{8.16}$$

and

$$\left[m^{k(p-1)}\right]^{q-1} \equiv 1 \pmod{q}. \tag{8.17}$$

Hence,

$$m^{de} \equiv m \pmod{q}, \qquad m^{de} \equiv m \pmod{p}. \tag{8.18}$$

By the Chinese remainder theorem[2],

$$m^{de} \equiv m \pmod{N}. \tag{8.19}$$

(3) If m is a multiple of q but not of p, the argument is symmetric, giving

$$m^{de} \equiv m \pmod{N}. \tag{8.20}$$

(4) If m is not a multiple of either p or q, then Fermat's little theorem gives

$$m^{de} \equiv m \pmod{p}, \qquad m^{de} \equiv m \pmod{q}. \tag{8.21}$$

Again, by the Chinese remainder theorem,

$$m^{de} \equiv m \pmod{N}. \tag{8.22}$$

This completes the proof of Eq. (8.12).

We see from the above analysis that if one could factor N efficiently, then the RSA system would be completely compromised. Indeed, factoring N reveals the primes p and q, from which one can compute $(p-1)(q-1)$, determine the public exponent e, and then recover the secret exponent d. With d in hand, any encrypted message can be decrypted.

---

[2] We are using a basic consequence of the Chinese remainder theorem: if two integers a and b satisfy $a \equiv b \pmod{p}$ and $a \equiv b \pmod{q}$, with $\gcd(p, q) = 1$, then $a \equiv b \pmod{pq}$. In particular, if p and q are prime, it's easier to see $p \mid (a-b)$ and $q \mid (a-b)$ implies $pq \mid (a-b)$.

8.2.2 Shor's Algorithm

Definition 8.3 (The Factoring Problem). The input to Shor's algorithm is an odd composite integer $N = pq$, where p and q are distinct large primes. The goal is to find the prime factors p and q.

Shor's algorithm is a hybrid quantum-classical algorithm that factors N in polynomial time on a quantum computer. It consists of two main phases:

1. **Classical reduction**: Transform factoring N into an instance of the *order-finding problem*.
2. **Quantum order-finding**: Solve the order-finding problem efficiently using a quantum computer.

The classical reduction (which succeeds with probability $> 1/2$ for random choices) is as follows:

1. Choose a random integer $a \in \{2, 3, \ldots, N-1\}$ and compute $d = \gcd(a, N)$:
   - If $1 < d < N$, then d is a non-trivial factor of N.
     Factoring complete.
   - If $d = 1$, continue.
2. Find the multiplicative *order* r of a modulo N, i.e., the smallest positive integer r such that
$$a^r \equiv 1 \pmod{N}.$$
3. If r is odd, return to step 1 with a new a.
4. If r is even, compute $a^{r/2} \mod N$:
   - If $a^{r/2} \equiv -1 \pmod{N}$, return to step 1.
   - If $a^{r/2} \not\equiv \pm 1 \pmod{N}$, proceed.
5. Compute the factors:
$$p = \gcd(a^{r/2} - 1, N), \qquad q = \gcd(a^{r/2} + 1, N).$$
   At least one of these is a non-trivial factor of N.

The only step that is computationally infeasible on a classical computer is step 2: finding the order r. Shor's breakthrough contribution is a *quantum algorithm* that performs this order-finding step in polynomial time using *quantum phase estimation* on the unitary operator $x \mapsto ax \mod N$.

Once r is obtained quantumly, the remaining classical post-processing immediately yields the prime factors p and q. Thus, on a sufficiently large fault-tolerant quantum computer, Shor's algorithm solves the integer factorization problem efficiently, rendering RSA and related cryptosystems insecure.

Remark 8.1 (Why the Reduction Works). Assume that r is even and that $a^{r/2} \not\equiv -1 \pmod{N}$ (the cases explicitly filtered out by the algorithm). Since r is the order of a modulo N, we have $a^r \equiv 1 \pmod{N}$. This implies

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

so $N \mid (a^{r/2} - 1)(a^{r/2} + 1)$, or equivalently, $pq \mid (a^{r/2} - 1)(a^{r/2} + 1)$.

Because p and q are distinct primes, the prime factors of the product $(a^{r/2} - 1)(a^{r/2} + 1)$ can be distributed in only four logical ways:

1. p divides $a^{r/2} - 1$ and q divides $a^{r/2} + 1$,
2. p divides $a^{r/2} + 1$ and q divides $a^{r/2} - 1$,
3. pq divides $a^{r/2} - 1$ (i.e., N divides $a^{r/2} - 1$),
4. pq divides $a^{r/2} + 1$ (i.e., N divides $a^{r/2} + 1$).

Cases 1 and 2 are precisely what we want: they yield non-trivial factors via the gcd computations $p = \gcd(a^{r/2} - 1, N)$ and $q = \gcd(a^{r/2} + 1, N)$ (or vice versa). Cases 3 and 4, however, must be excluded:

- Case 3 is impossible: If N divides $a^{r/2} - 1$, then $a^{r/2} \equiv 1 \pmod{N}$. But r is the *order* of a modulo N, so r would have to divide $r/2$, which is impossible unless $r = 0$, a contradiction.
- Case 4 is impossible under our assumption: If N divides $a^{r/2} + 1$, then $a^{r/2} \equiv -1 \pmod{N}$. This situation is explicitly detected and rejected in the algorithm (step 5).

Thus, whenever the algorithm reaches step 6, we are guaranteed to be in Case 1 or Case 2, and the two gcd operations recover the prime factors p and q of N.

Example 8.4. Let us illustrate Shor's algorithm with the integer $N = 15$.

1. Choose a random number: Let $a = 7$. Compute $\gcd(7, 15) = 1$. Since $\gcd(7, 15) = 1$, we proceed to the next step.
2. Find the period r: Using a quantum computer, we determine the period of the function $f(x) = 7^x \bmod 15$. The result is $r = 4$, since $7^4 \equiv 1 \bmod 15$.
3. Use the period to find the factors:
   - Since $r = 4$ is even, compute $7^{r/2} - 1 = 7^2 - 1 = 48$.
   - Compute the greatest common divisor: $\gcd(48, 15) = 3$, which is one factor of 15.
   - Similarly, $7^{r/2} + 1 = 7^2 + 1 = 50$, and $\gcd(50, 15) = 5$, which is the other factor.

Thus, we have successfully factored $15 = 3 \times 5$.

9 Hidden subgroup problem

10 Harrow-Hassidim-Lloyd Algorithm

Part IV
Quantum machine learning

Index