

Zhian Jia (贾治安)

QUANTUM COMPUTATION THEORY IN A NUTSHELL

Aug 1, 2025

Contents

Part I Mathematical and physical preliminaries

Part II Quantum computation theory

1	Classical computation	5
1.1	Basics of classical computation	5
1.1.1	Classical logic gates and circuit model	6
1.1.2	Universal gate set	9
1.1.3	Binary arithmetics	10
1.2	Classical circuit model of computation	10
1.2.1	Classical logic gates	10
1.2.2	Universal logic gates	10
1.3	Complexity classes	11
2	Basics of quantum computation	13
2.1	Quantum computation and quantum circuit model	13
2.1.1	Encoding classical information into quantum states	14
2.1.2	Quantum circuit	15
2.1.3	Universal quantum gate	18
3	Quantum Circuit model	19
3.1	Generalities of quantum computation	19
3.1.1	Encoding information with quantum states	19
3.2	Quantum circuit model	19
3.2.1	Encoding information with qubits	19
3.2.2	Unitary transform as quantum gates	19
3.3	Universal quantum gates	20
3.3.1	The notion of a universal gate set	20
3.3.2	Two-level unitary transformations are universal	20
3.3.3	Examples of universal gates	22
3.4	Other quantum computing models	22
3.4.1	Measurement-only quantum computation	22

4 Quantum simulation	23
-----------------------------------	----

Part III Quantum algoritheorems

5 Foundational Quantum Algoritheorems	27
5.1 Deutsch-Jozsa algoritheorem	28
5.1.1 Deutsch-Jozsa Algoritheorem: $n = 1$ Case	29
5.1.2 Deutsch-Jozsa Algoritheorem	31
5.2 Bernstein-Vazirani Algoritheorem	35
5.3 Simon's algoritheorem	38
6 Grover Search Algoritheorem	45
6.1 Grover's Search Algoritheorem	45
6.1.1 Problem Statement	45
6.1.2 Quantum Oracle	46
6.1.3 Grover Iteration	46
6.1.4 Steps of Grover's Algoritheorem	47
6.1.5 Mathematics of Amplitude Amplification	48
6.1.6 Quantum Circuit of Grover's Algoritheorem	49
6.1.7 Advantages and Limitations	49
7 Quantum Integral Transforms with Applications	51
7.1 Quantum Discrete Integral Transform	51
7.2 Quantum Fourier Transform	54
7.2.1 The inverse quantum Fourier transform	59
7.3 Quantum Phase Estimation Algoritheorem	60
8 Shor's Factoring algoritheorem	65
8.1 Quantum Order-Finding Algoritheorem	65
8.1.1 Preliminaries of Modular Arithmetical	66
8.1.2 Quantum Order-Finding Algoritheorem	70
8.1.3 Recovering the Order r from the Measured Phase	72
8.2 Shor's Algoritheorem	74
8.2.1 RSA Cryptosystem (Rivest-Shamir-Adleman)	74
8.2.2 Shor's Algoritheorem	74
8.2.3 Why Does This Work?	75
8.3 Shor's Factoring Algoritheorem	76
8.3.1 Outline of Shor's Algoritheorem	76
8.3.2 Shor's Algoritheorem	76
8.3.3 Example of Shor's Algoritheorem	78
8.3.4 Quantum Speedup	78
9 Harrow-Hassidim-Lloyd algoritheorem	81

Part IV Quantum machine learning

Index	85
--------------------	----

Part I

Mathematical and physical preliminaries

Part II

Quantum computation theory

1

Classical computation

The best way to learn a new topic
is to write a book about it.

*By Serge Lang
Quoted in the article “Serge Lang,
1927-2005” written by Jay
Jorgenson and Steven G. Krantz*

§ 1.1 Basics of classical computation

Classical computers deal with classical bit strings, which consists of 0s and 1s. The computation is just the evaluation of a function: given an n -bit input, it produces an m -bit output uniquely determined by that input. In other words, it computes the value of the function¹

$$f : \{0, 1\}^n \rightarrow \{0, 1\}^m,$$

for a specified n -bit argument x . A function with an m -bit output is equivalent to m functions, each producing a one-bit output. Therefore, we can say that the fundamental task performed by a computer is the evaluation of

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

which is usually called *Boolean function*.

With this concept of computation in mind, we can consider how to model the computation process. There are numerous models for computation, including the Turing machine and the circuit model. In classical computation, we often focus on the Turing machine. However, for the convenience of generalizing to

¹ We will also use \mathbb{B} to represent the binary set $\{0, 1\}$ whenever convenient.

quantum computation, we will focus on the circuit model. It is important to note that different models of computation are equivalent. More crucially:

Theorem 1.1. *The circuit model and the Turing machine are equivalent for classical computation.*

Although we won't discuss the Turing machine in detail, let us provide a rigorous definition for completeness. We won't use the Turing machine to discuss computation in these lecture notes, but it is important to stress that our computers are built based on the Turing machine rather than the circuit model. This is the central concept for classical computation.

Definition 1.1 (k -tape Turing machine). A k -tape Turing machine M is a triple $M = (\Gamma, Q, \delta)$ where

- $\Gamma = \{0, 1, \dots, d-1, \triangleright, \square\}$ is a finite set called the tape symbol alphabet of the Turing machine, where $0, 1, \dots, d-1$ are input symbols, $\triangleright \in \Gamma$ marks the left-hand end of the tape and \square is the blank symbol (the only symbol allowed to occur infinitely often on the tape at any step during computation);
- $Q = \{q_0, q_1, \dots, q_n\}$ is a finite set whose elements are called states of the Turing machine; q_0 is a special element called the initial state and there exists a subset $F \subset Q$ consisting of the states accepted by the machine (called halt states, final states or accepting states) in which the Turing machine will eventually halt;
- $\delta : (Q \setminus F) \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{-1, 0, +1\}$ is a partial function² called the transition function, where -1 (resp. 0 , resp. $+1$) represents the left shift (resp. stand still, resp. right shift) of the read-write tape-head of the Turing machine.

For interested readers, please refer to Chapter 3 of Nielsen and Chuang's book for a brief introduction.

1.1.1 Classical logic gates and circuit model

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



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

More formally, a gate can be defined as follows.

Definition 1.2. Classical gate An n -input, m -output **classical logic gate** (or simply, gate) is an m -component Boolean function

$$F : \{0, 1\}^n \rightarrow \{0, 1\}^m. \quad (1.2)$$

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



1.1.1.1 Single-bit gate

Notice that there are only 4 single-bit Boolean function $f : \{0, 1\} \rightarrow \{0, 1\}$ ³: (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:

$$\text{NOT} : \quad a \mapsto \bar{a} = 1 \oplus a. \quad (1.4)$$

Diagrammatically:



1.1.1.2 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 $f : \mathbb{B}^2 \rightarrow \mathbb{B}$ (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:

³ Since there are 2 possible outputs for each input 0 and 1, there are a total of $2 \times 2 = 4$ possibilities.

$$\text{AND} : (a, b) \mapsto a \wedge b = ab. \quad (1.6)$$

Diagrammatically:



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

- The operation of the OR gate is as follows:

$$\text{OR} : (a, b) \mapsto a \vee b = a \oplus b \oplus ab. \quad (1.8)$$

Diagrammatically:



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:

$$\text{XOR} : (a, b) \mapsto a \oplus b. \quad (1.10)$$

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

$$\text{XNOR} : (a, b) \mapsto \overline{a \oplus b} = 1 \oplus a \oplus b. \quad (1.11)$$

1.1.1.3 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:

$$\text{SWAP} : (a, b) \mapsto (b, a). \quad (1.12)$$

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

$$\text{CLONE} : a \mapsto (a, \dots, a). \quad (1.13)$$

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 1.2 (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 $|\psi\rangle$,

$$U(|\psi\rangle \otimes |0\rangle) = |\psi\rangle \otimes |\psi\rangle. \quad (1.14)$$

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

$$U(|\psi_1\rangle \otimes |0\rangle) = |\psi_1\rangle \otimes |\psi_1\rangle, \quad (1.15)$$

$$U(|\psi_2\rangle \otimes |0\rangle) = |\psi_2\rangle \otimes |\psi_2\rangle. \quad (1.16)$$

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

$$\langle\psi_1|\psi_2\rangle = \langle\psi_1|\psi_2\rangle^2, \quad (1.17)$$

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: $\text{CLONE} : a \mapsto (a, \dots, a)$.

1.1.2 Universal gate set

Theorem 1.3 (Universal gate). The gate set $\{\text{NOT}, \text{AND}, \text{OR}, \text{CLONE}\}$ is universal: any Boolean function

$f : \{0,1\}^n \rightarrow \{0,1\}^m$ can be implemented by a circuit composed of these gates.

Exercise 1.1. Try to prove Theorem 1.3 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.

1.1.3 Binary arithmetics

§ 1.2 Classical circuit model of computation

1.2.1 Classical logic gates

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

$$f : \mathbb{B}^n \rightarrow \mathbb{B}^m. \quad (1.18)$$

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

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

$$\text{NOT} : a \mapsto \bar{a} = 1 \oplus a. \quad (1.19)$$

Diagrammatically, it's denoted as



1.2.2 Universal logic gates

Exercise 1.2 (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.

§ 1.3 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 algoritheorems 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 algoritheorems 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 $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, 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 state¹. 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 $\vec{x} = (x_1, x_2, \dots, x_n)$ to the basis

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

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 $\vec{f}: \mathcal{X} \rightarrow \mathbb{R}^N$ and encode classical data into an N -dimensional feature Hilbert space:

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

where $\|\vec{f}(\vec{x})\|_2 = (\sum_i f_i(\vec{x})^2)^{1/2}$ and $i = 1, \dots, N$. 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 1985², and its notation appeared in Richard Feynman's paper³.

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

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

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 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*

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

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

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

and equivalently as a linear combination of Pauli operators:

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

Rotation operators about a direction \vec{n} are

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

Special cases:

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

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

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

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

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

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:

$$\text{CNOT} = \begin{array}{c} \text{---} \\ \text{---} \end{array} \bigoplus, \quad |a\rangle \otimes |b\rangle \mapsto |a\rangle \otimes |a \oplus b\rangle.$$

2. Controlled-Z gate:

$$C(Z) = \begin{array}{c} \text{---} \\ \text{---} \end{array} \bullet.$$

3. SWAP gate:

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

2.1.3 Universal quantum gate

Definition 2.3. Universal gate set A set of quantum gates $\mathcal{U} = \{U_1, \dots, U_m\}$ 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.*

§ 3.1 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*

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

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

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

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

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,

$$\mathcal{U} = \{U_1, \dots, U_n\}, \quad (3.4)$$

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

$$\mathbf{Circ}(\mathcal{U}; n) = \{U = F_{\text{circ}}(U_1, \dots, U_n) \mid U_1, \dots, U_n \in \mathcal{U}\}. \quad (3.5)$$

Definition 3.1. A set of quantum gates $\mathcal{U} = \{U_1, \dots, U_m\}$ (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 $|0\rangle, \dots, |d-1\rangle$, we call a unitary transformation U a two-level unitary transformation¹ 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, \dots, 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 $1, 0, \dots, 0$. To this end, we will construct some two-level unitary matrices U_1, \dots, 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 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 $U_2 U_1 U$ will be $(u''_{11}, 0, 0, u''_{41}, \dots, u''_{n1})^T$. Similarly, we can construct U_3, \dots, U_{n-1} such that

$$V = U_{n-1} \cdots U_1 U = \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ 0 & v_{22} & \cdots & v_{2n} \\ 0 & v_{32} & \cdots & v_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & v_{n2} & \cdots & v_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & v_{22} & \cdots & v_{2n} \\ 0 & v_{32} & \cdots & v_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & v_{n2} & \cdots & v_{nn} \end{pmatrix} = I^{(1)} \oplus U',$$

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 $I^{(n-d)} \oplus U^{(d)} \rightarrow I^{(n-d+1)} \oplus U^{(d-1)}$ (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 $(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$. \square

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 algoritheorems

5 Foundational Quantum Algoritheorems

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

Along the way, we also introduce several basic techniques and tools that recur throughout quantum algoritheorem design, including the use of the Hadamard transform, phase kickback, and quantum parallelism. These tools provide the building blocks for more advanced algoritheorems 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 algoritheorems 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 computa-

tional problems more efficiently, and will be prepared to study more sophisticated algoritheorems in the remainder of this text.

§ 5.1 Deutsch-Jozsa algoritheorem

The Deutsch-Jozsa algoritheorem, 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 algoritheorems. 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 algoritheorems revisited”. Both papers are highly readable, and we encourage interested readers to consult them directly.

As a deterministic quantum algoritheorem, the Deutsch-Jozsa algoritheorem 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 algoritheorems. Studying this algoritheorem offers valuable insight into how quantum algoritheorems are structured and why they can outperform classical methods.

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

Definition 5.1 (Problem of the Deutsch-Jozsa Algoritheorem).

Consider a function

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

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 $f : \{0, 1\} \rightarrow \{0, 1\}$. 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 Algoritheorem: $n = 1$ Case*

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

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

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. \square

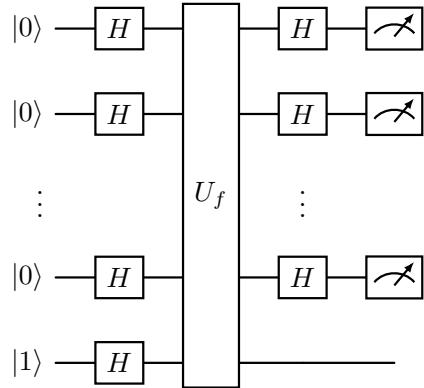


Fig. 5.1 Quantum circuit for the Deutsch-Jozsa algoritheorem. The top n qubits encode the input of the function f , and the single bottom qubit encodes the output of f .

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

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

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

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

$$|0 \oplus z\rangle - |1 \oplus z\rangle = (-1)^z(|0\rangle - |1\rangle), \quad (5.2)$$

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

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

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

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

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

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

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

$$p(0) = \frac{1}{4} \left| 1 + (-1)^{f(0) \oplus f(1)} \right|^2, \quad p(1) = \frac{1}{4} \left| 1 - (-1)^{f(0) \oplus f(1)} \right|^2. \quad (5.6)$$

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 algoritheorem 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 Algoritheorem

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

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle, \quad (5.7)$$

where $x = x_1 x_2 \cdots x_n$ is an n -bit string, and we write

$$|x\rangle := |x_1\rangle |x_2\rangle \cdots |x_n\rangle. \quad (5.8)$$

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 algoritheorem is shown in Figure 5.1. The algoritheorem 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 $|1\rangle$:

$$|0\rangle^{\otimes n} \otimes |1\rangle. \quad (5.9)$$

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

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

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

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle. \quad (5.11)$$

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

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

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

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

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

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

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

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

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

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

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

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

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),

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

the state becomes

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

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

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

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

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

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

$$\pm |0\rangle^{\otimes n}. \quad (5.22)$$

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 algoritheorem.

- 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

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

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

2. The identity

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

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 $x = (x_1, \dots, x_n)$ and use the single-qubit identities

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

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 $z = (z_1, \dots, z_n)$. 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):

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

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 $(H^{\otimes n})^2 = I^{\otimes n}$. \square

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 algoritheorem.

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

§ 5.2 Bernstein-Vazirani Algoritheorem

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

The algoritheorem treats the following problem:

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

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

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

$$x \cdot s = (0 \times 0) \oplus (0 \times 1) \oplus (1 \times 1) = 1. \quad (5.25)$$

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

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

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

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

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

$$\text{maj}(x \oplus y) \neq \text{maj}(x) \oplus \text{maj}(y). \quad (5.27)$$

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

$$\text{maj}(110) = 1, \quad \text{maj}(011) = 1, \quad \text{maj}(101) = 1,$$

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

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

$$10 \cdots 0, \quad 010 \cdots 0, \quad \dots, \quad 0 \cdots 01.$$

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

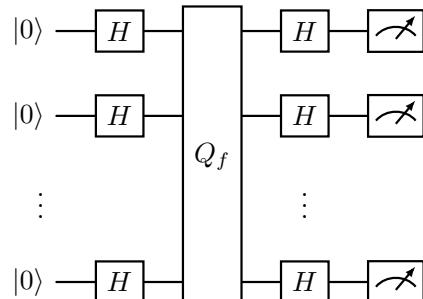


Fig. 5.2 Quantum circuit for the Bernstein–Vazirani algoritheorem.

$$\begin{aligned} 10 \cdots 0 \cdot s &= s_1, \\ 010 \cdots 0 \cdot s &= s_2, \\ &\vdots \\ 0 \cdots 01 \cdot s &= s_n. \end{aligned}$$

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 algoritheorem, 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

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

We then apply the quantum oracle U_f encoding f :

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle. \quad (5.29)$$

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

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

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

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

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

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

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 algoritheorem 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 algoritheorem

Simon's algoritheorem is a quantum algoritheorem designed to find a hidden string s in a particular class of functions, achieving an exponential speedup over any classical algoritheorem. 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 algoritheorem holds historical significance as it was the first to demonstrate an exponential quantum advantage over the best known classical algoritheorem, revealing how quantum computation can surpass classical methods in well-defined computational tasks. The algoritheorem was proposed by Daniel R. Simon in 1994. Simon later recounted the story behind his algoritheorem in a blog article¹:

I submitted my algoritheorem 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 algoritheorem structure to concrete problems like factoring and discrete logarithms. 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 algoritheorem served as a precursor to Peter Shor's celebrated factoring algoritheorem. It laid the conceptual foundation for many of the seminal advances in quantum computation that followed. The problem addressed by Simon's algoritheorem can be rigorously stated as follows:

Definition 5.3 (Simon's Problem). Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ 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:

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

- *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 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 $f : \{0, 1\}^3 \rightarrow \{0, 1\}^3$ 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

$$s = x_i \oplus x_j. \quad (5.33)$$

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 $\binom{k}{2}$ pairs of inputs. Hence, the probability of finding at least one collision is approximately $p_{\text{suc}} \sim \binom{k}{2}/(2^n - 1)$. 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.

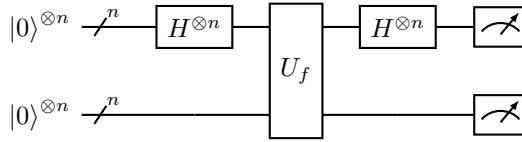


Fig. 5.3 Quantum circuit for the Simon’s algoritheorem. The top n qubits hold the input register, and the bottom n qubits are the ancillary register used for the phase oracle U_f .

However, Simon’s algoritheorem 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 algoritheorem:

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

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 algoritheorem 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 algoritheorem’s steps:

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:

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

The state now becomes

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

3. Query the Oracle

Use the oracle U_f to compute $f(x)$ for each x :

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle. \quad (5.36)$$

After applying the oracle, the state becomes

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

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

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

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

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

- *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

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

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,

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

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,

$$z_1 \cdot s = 0,$$

$$z_2 \cdot s = 0,$$

$$\vdots$$

$$z_m \cdot s = 0,$$

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

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

Let us consider a special example.

§ 6.1 Grover's Search Algoritheorem

Grover's search algoritheorem is one of the cornerstone algoritheorems in quantum computation, providing a quadratic speedup for unstructured search problems. The search algoritheorem for an unstructured database is crucial in various applications. For example, it can be used to find the lowest-priced product, determine the shortest path to a destination via transfers, and more.

This Grover's algoritheorem is proposed by Lov Grover in 1996. Classical algoritheorems require $O(N)$ queries to search an unsorted database of N elements, but Grover's algoritheorem reduces this to $O(\sqrt{N})$ using quantum parallelism and amplitude amplification.

6.1.1 Problem Statement

Problem statement of Grover search algoritheorem The goal of Grover's algoritheorem is to search for a unique item (or marked element) in an unsorted database. Given an oracle function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, where $f(x) = 1$ for the marked item x_{marked} and $f(x) = 0$ for all other items, the task is to find x_{marked} using fewer queries to the oracle than classical algoritheorems would require.

We will assume that the search space contains $N = 2^n$ possible items. Classical search requires $O(N)$ oracle queries, since at the worst case, we need to query the oracle N times to find the marked item. The Grover's algoritheorem reduces this to $O(\sqrt{N})$.

6.1.2 Quantum Oracle

Recall that the classical oracle is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$, where $f(x) = 1$ for the marked item x_{marked} and $f(x) = 0$ for all other items. We can generalize this to a quantum oracle U_f .

The oracle is a quantum operation U_f that flips the sign of the amplitude of the marked state. It acts as follows:

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

For the marked state x_{marked} , $f(x_{\text{marked}}) = 1$, so the phase of this state is flipped:

$$U_f |x_{\text{marked}}\rangle = -|x_{\text{marked}}\rangle$$

For all other states $x \neq x_{\text{marked}}$, $f(x) = 0$, so their amplitudes remain unchanged.

Exercise 6.1. This quantum oracle is related to the one defined in Eq. (5.7) by setting $|y\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. Check it by yourself.

6.1.3 Grover Iteration

The algorithm operates by repeatedly applying a process known as *Grover iteration* or *Grover operator* to amplify the amplitude of the marked state. Notice that if there are M marked items, we can introduce

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x:\text{unmarked}} |x\rangle \quad (6.1)$$

which is the equal superposition state of all unmarked items; and

$$|\beta\rangle = \frac{1}{\sqrt{M}} \sum_{x:\text{marked}} |x\rangle. \quad (6.2)$$

The crucial fact is that the equal superposition state of all items

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle \quad (6.3)$$

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

$$|\psi\rangle = \frac{\sqrt{N-M}}{\sqrt{N}} |\alpha\rangle + \frac{\sqrt{M}}{\sqrt{N}} |\beta\rangle. \quad (6.4)$$

$|\beta\rangle$ is our target state and we need to amplify the amplitude for $|\beta\rangle$.

$$\cos \frac{\theta}{2} = \frac{\sqrt{N-M}}{\sqrt{N}}, \quad \sin \frac{\theta}{2} = \frac{\sqrt{M}}{\sqrt{N}} \quad (6.5)$$

which is useful for us to understand Grover search geometrically. Hereinafter, for simplicity, we will assume $M = 1$, viz., there is only one marked item. We can set

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.
2. **Amplitude Amplification (Grover diffusion operator):** Apply the Grover operator D , which amplifies the probability amplitude of the marked state by inverting all amplitudes about their average. The diffusion operator can be expressed as:

$$D = 2|\psi\rangle\langle\psi| - I$$

where $|\psi\rangle$ is the equal superposition state $\frac{1}{\sqrt{N}}\sum_x|x\rangle$ and I is the identity operator.

Due to the fact that $|\psi\rangle$ in the plane spanned by $|\alpha\rangle$ and $|\beta\rangle$, the Grover iteration can be illustrated as a geometric transformation, see Fig. ?? We can introduce the Grover iteration operator as

$$G = DU_f = (2|\psi\rangle\langle\psi| - I)U_f. \quad (6.6)$$

Thus from the equal superposition state $|\psi\rangle$, one Grover iteration rotates $3\theta/2$ towards our target state $|\beta\rangle$.

Exercise 6.2. Check that $G|\psi\rangle$ works as that illustrated in Fig. ???. Also try to show that the Grover iteration can be regarded as a rotation matrix in the basis of $|\alpha\rangle$ and $|\beta\rangle$.

From Fig. ???, it's clear that θ in Eq. (6.5) must be small to reach the high precision. This means that M/N must be small. The Grover search can reach a high precision for large database.

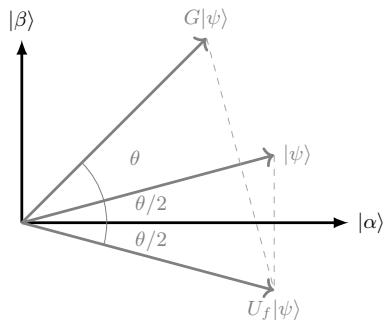


Fig. 6.1 Illustration of the Grover iteration.

6.1.4 Steps of Grover's Algoritheorem

The following is a detailed procedure for implementing Grover search:

1. Initialization:

- Start with n qubits in the state $|0\rangle^{\otimes n}$.
- Apply Hadamard gates $H^{\otimes n}$ to create an equal superposition of all possible states:

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

2. Grover Iterations:

- Apply the Grover iteration $O(\sqrt{N})$ times. Each iteration consists of:
 - Oracle Application:** Apply the oracle U_f , flipping the sign of the marked state.
 - Amplitude Amplification:** Apply the diffusion operator D , amplifying the marked state's amplitude.

3. Measurement:

After $O(\sqrt{N})$ iterations, the probability amplitude of the marked state will be close to 1. Measure the state to obtain x_{marked} with high probability.

6.1.5 Mathematics of Amplitude Amplification

Suppose the number of marked item is $M = 1$, let us analyze the iteration process more carefully. The key to Grover's algoritheorem lies in how the amplitudes of the states evolve under each Grover iteration. Initially, all states have equal amplitude $\frac{1}{\sqrt{N}}$. After applying the oracle and the diffusion operator, the amplitude of the marked state increases while the amplitude of the unmarked states decreases.

Each Grover iteration performs a rotation in a two-dimensional subspace spanned by the equal superposition of all states and the marked state. The angle of rotation is $\theta \approx 2 \sin^{-1} \left(\frac{1}{\sqrt{N}} \right)$. After $k = O(\sqrt{N})$ iterations, the state vector is rotated close to the marked state, maximizing the probability of measuring x_{marked} .

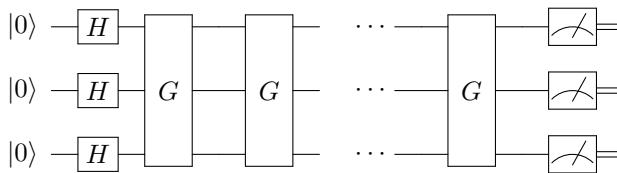
The optimal number of Grover iterations is approximately $\frac{\pi}{4}\sqrt{N}$. If the number of iterations exceeds this, the amplitudes will begin to rotate away from the marked state, reducing the success probability.

6.1.6 Quantum Circuit of Grover's Algoritheorem

The quantum circuit for Grover's algoritheorem consists of:

- **Hadamard Gates:** Apply $H^{\otimes n}$ to initialize the superposition state.
- **Grover iteration G :** Implement the Grover iteration many times, each Grover iteration consists of
 - **Oracle U_f :** Implement the oracle that flips the phase of the marked state.
 - **Diffusion Operator:** Apply the Grover diffusion operator, typically constructed using Hadamard gates, Z gates, and controlled operations.
- **Measurement:** Measure the qubits to obtain the marked state.

The following is an example of three-qubit circuit for Grover search:



Each iteration consists of applying the oracle U_f followed by the diffusion operator D .

6.1.7 Advantages and Limitations

- **Quadratic Speedup:** Grover's algoritheorem provides a quadratic speedup over classical search algoritheorems, requiring only $O(\sqrt{N})$ queries to the oracle.
- **Generality:** It works for any unstructured search problem where the oracle function can be defined.
- **Limitation:** Grover's algoritheorem only provides a quadratic speedup, so for very large databases, the improvement may not be sufficient to surpass classical methods.
- **Multiple Solutions:** If there are multiple marked elements, the algoritheorem still works but the number of iterations must be adjusted accordingly. The optimal number of iterations decreases as the number of marked elements increases.

Grover's Search Algoritheorem is a powerful tool in the quantum computing repertoire, illustrating how quantum mechanics can provide computational speedups for problems like unstructured search. Its quadratic speedup, while modest compared to the exponential gains offered by algoritheorems like Shor's

factoring algoritheorem, demonstrates the potential of quantum algoritheorems for solving problems more efficiently than classical algoritheorems.

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 necessarily have to be equal.

In the previous chapter, we discussed quantum algoritheorems that rely on query oracles. In this section, we shift our focus to algoritheorems 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 algoritheorem.

These circuit-based algoritheorems 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 algoritheorems, 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 include the Fourier transform, Laplace transform, Mellin transform, and many others, are indispensable across mathematics, physics, engineering, signal processing, quantum mechanics, and beyond. These transformations often reveal 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):

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

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

$$y_i = \sum_{j=0}^{N-1} K_{ij} x_j. \quad (7.1)$$

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 $j = 0, \dots, N - 1$. 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)

$$|j\rangle \xrightarrow{\text{QDIT}} U_K |j\rangle = \sum_{k=0}^{N-1} K_{kj} |k\rangle. \quad (7.2)$$

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:

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

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

$$|j\rangle \xrightarrow{\text{QDIT}} U_K |j\rangle = \sum_{k=0}^{N-1} K_{kj} |k\rangle. \quad (7.3)$$

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 algoritheorem, 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

$$U_{\text{QFT}} = K_{jk} = \frac{1}{\sqrt{N}} e^{2\pi i \frac{jk}{N}} = \frac{\omega_N^{jk}}{\sqrt{N}} \quad (7.4)$$

is unitary. You may find the following formula useful:

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

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

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

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

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 algoritheorems 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 algoritheorems, including Shor's algoritheorem 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 $|j\rangle \mapsto U_{\text{QFT}}|j\rangle$ with $j = 0, \dots, N-1$, 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:

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

or equivalently, for a general quantum state,

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

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

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

This implies that

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

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

$$k \times 0.j_1 j_2 \cdots j_n. \quad (7.11)$$

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

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

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

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

We then drop the integer part when $k_s = 1$ for all $s = 1, \dots, n$ 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

$$U_{\text{QFT}}|j\rangle = \frac{(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle) \cdots (|0\rangle + e^{2\pi i 0.j_1j_2\cdots j_n}|1\rangle)}{\sqrt{N}}. \quad (7.13)$$

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

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

$$\begin{aligned} U_{\text{QFT}}|j\rangle &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k / N} |k\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{k_1=0}^1 \cdots \sum_{k_n=0}^1 e^{2\pi i j (\sum_{l=1}^n k_l 2^{-l})} |k_1 \cdots k_n\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{k_1=0}^1 \cdots \sum_{k_n=0}^1 \bigotimes_{l=1}^n e^{2\pi i j k_l 2^{-l}} |k_l\rangle \\ &= \frac{1}{\sqrt{N}} \bigotimes_{l=1}^n \left[\sum_{k_l=0}^1 e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right] \\ &= \frac{1}{\sqrt{N}} \bigotimes_{l=1}^n \left[|0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \right] \\ &= \frac{(|0\rangle + e^{2\pi i 0.j_n}|1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle) \cdots (|0\rangle + e^{2\pi i 0.j_1j_2\cdots j_n}|1\rangle)}{\sqrt{N}}. \end{aligned}$$

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

For convenience, let us denote

$$\begin{aligned} |q_1\rangle &= \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.j_1j_2\cdots j_n}|1\rangle), \\ |q_2\rangle &= \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.j_2j_3\cdots j_n}|1\rangle), \\ &\vdots \\ |q_n\rangle &= \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.j_n}|1\rangle). \end{aligned} \quad (7.14)$$

Using this notation, we have

$$U_{\text{QFT}}|j_1\rangle|j_2\rangle\cdots|j_n\rangle = |q_n\rangle|q_{n-1}\rangle\cdots|q_1\rangle. \quad (7.15)$$

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

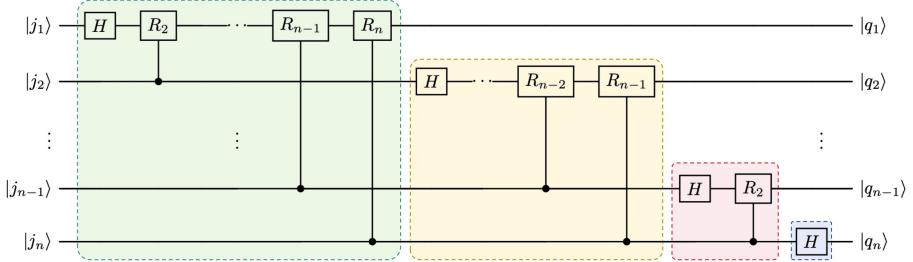


Fig. 7.1 The quantum circuit for quantum Fourier transform.

$$|j_1\rangle|j_2\rangle\cdots|j_n\rangle \longrightarrow |q_1\rangle|q_2\rangle\cdots|q_n\rangle, \quad (7.16)$$

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

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

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 contexts; here is yet another useful representation:

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

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

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

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))

$$|j_n\rangle \rightarrow |q_n\rangle. \quad (7.20)$$

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 $e^{2\pi i 0.0j_n}$, so that the final phase factor is $e^{2\pi i 0.j_{n-1}j_n}$. The state of this quantum wire becomes

$$|j_{n-1}\rangle \rightarrow |q_{n-1}\rangle. \quad (7.21)$$

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 $e^{2\pi i 0.0j_{n-1}}$, and the controlled- R_3 gate controlled by j_n contributes $e^{2\pi i 0.00j_n}$. 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

$$|j_{n-2}\rangle \rightarrow |q_{n-2}\rangle. \quad (7.22)$$

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_{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:

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

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

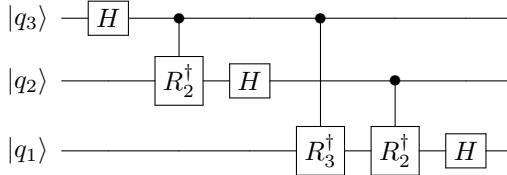
Exercise 7.4. Verify that $U_{\text{QFT}}^\dagger = U_{\text{QFT}}^{-1}$ using Eq. (7.5) in Exercise 7.1.

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

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

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 algoritheorems.

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 Algoritheorem

A key application of the quantum Fourier transform is the *Quantum Phase Estimation* algoritheorem. This algoritheorem allows one to estimate the phase (or eigenvalue) of a unitary operator and serves as a central component of many quantum algoritheorems, including Shor's factoring algoritheorem 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 $0 \leq \theta < 1$. Expressing θ in binary gives

$$\theta = 0.\theta_1\theta_2\theta_3\dots = \sum_{k=1}^{\infty} \theta_k 2^{-k}, \quad (7.25)$$

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 algoritheorem aims to determine the phase θ in the eigenvalue equation

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

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,

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

Then

$$U^{2^l}|\psi\rangle = e^{2\pi i 0.\theta_1\theta_2\dots\theta_n \times 2^l}|\psi\rangle = e^{2\pi i \theta_1\dots\theta_l \cdot \theta_{l+1}\dots\theta_n}|\psi\rangle, \quad (7.27)$$

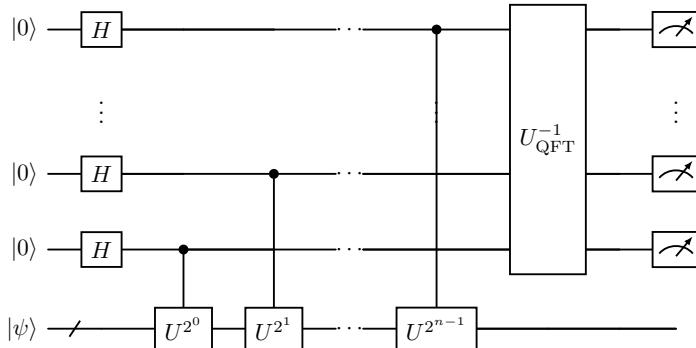


Fig. 7.2 The quantum circuit for quantum phase estimation algoritheorem.

where the integer part of $\theta_1 \cdots \theta_l \cdot \theta_{l+1} \cdots \theta_n$ 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):

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

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

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

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 algoritheorem 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 algoritheorem 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

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

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 (QFT^\dagger) on the first register to extract the phase information:

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

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 algoritheorem estimates the phase θ with a high probability of success. If θ can be exactly represented with n bits, the algoritheorem will output the correct result with probability 1. If θ cannot be exactly represented, the algoritheorem 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 algoritheorem is a key quantum algoritheorem that estimates the phase corresponding to an eigenvalue of a unitary operator. It is widely used in algoritheorems for factoring, quantum simulations, and quantum chemistry.

In this chapter, we discuss the Shor's factoring algoritheorem.

§ 8.1 Quantum Order-Finding Algoritheorem

The order-finding problem is a fundamental problem that plays a central role in several quantum algoritheorems; in particular, it underlies Shor's celebrated factoring algoritheorem. Classically, order-finding is hard—no efficient classical algoritheorem 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 $\text{ord}_N(a)$ or simply r , is the smallest *positive* integer r such that

$$a^r \equiv 1 \pmod{N}. \quad (8.1)$$

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

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 algoritheorem for order-finding exploits the periodic structure of the function

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

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

$$U|y\rangle = |ay \pmod{N}\rangle, \quad (8.3)$$

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

8.1.1 Preliminaries of Modular Arithoermetic

Before presenting the complete quantum algoritheorem, 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, $N = 2, 3, 5, 7, 11, \dots$ 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:

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

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

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, ... 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 arithoermetic. In everyday life we count linearly: 1, 2, 3, ..., in modular arithoermetic 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 :

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

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:

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

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 $\mathbb{Z}_7 := \{0, 1, 2, \dots, 6\}$. Then $1 \equiv 8 \pmod{N}$, $2 \equiv 9 \pmod{N}$. The congruence class $[0]$ consists of $\{0, 7, 14, 21, \dots\}$, the congruence class $[1]$ consists of $\{1, 8, 15, 22, \dots\}$, and $[2]$ consists of $\{2, 9, 16, 23, \dots\}$, 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 $0, 1, \dots, N - 1$:

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

However, not all values of $b = 0, 1, \dots, N - 1$ 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. \square

In Shor's algoritheorem, 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 $k \in \{1, 2, \dots, N-1\}$ 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 if 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 $r = \text{ord}_N(a)$ always satisfies $1 \leq r \leq \phi(N)$.

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

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

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

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

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

\vdots

$$a^{r-1} \not\equiv 1 \pmod{N},$$

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

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

$$a^{r+2} \equiv a^2 \pmod{N},$$

$$a^{r+3} \equiv a^3 \pmod{N}.$$

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 $r = \text{ord}_7(2) = 3$.

8.1.2 Quantum Order-Finding Algoritheorem

The quantum order-finding algoritheorem 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

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

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*

$$U_{a,N}|y\rangle = |ay \bmod N\rangle, \quad y = 0, 1, \dots, N-1, \quad (8.7)$$

is unitary.

Proof. Since $\gcd(a, N) = 1$, multiplication by a modulo N defines a bijection on the set $\{0, 1, \dots, N-1\}$. Thus $U_{a,N}$ merely permutes the computational basis $\{|0\rangle, |1\rangle, \dots, |N-1\rangle\}$. 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 $0 \leq y, y' < N$, 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. \square

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 $s = 0, 1, \dots, r-1$, define the state*

$$|u_s\rangle := \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i sk/r} |a^k \bmod N\rangle. \quad (8.8)$$

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

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

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

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

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

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

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

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

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

This holds for every $s = 0, 1, \dots, r - 1$. \square

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

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

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

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

The output of the phase estimation algoritheorem is thus

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

represented as binary numbers. A classical algoritheorem 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 algoritheorem, the quantum Fourier transform yields a phase $\theta \approx s/r$, where $s \in \{0, 1, \dots, r - 1\}$ 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

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

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 $q \leq N$, 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 $\leq N$ 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: $13 \leq 21$. 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 $\theta \approx 7/12 = [0; 1, 2, 1, 2]$).

Algoritheorem 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 Algoritheorem

8.2.1 RSA Cryptosystem (Rivest-Shamir-Adleman)

$N = pq$ where p, q are primes.

Euler function: $\phi(N) = (p - 1)(q - 1)$

1. Choose e with $\gcd(e, \phi(N)) = 1$.

Public key: (e, N)

2. **Extended Euclidean Algoritheorem:** If $\gcd(x, y) = 1$, $\exists d$ such that:

$$xd \equiv 1 \pmod{y}$$

Since $\gcd(e, \phi(N)) = 1$:

Secret key: d where $ed \equiv 1 \pmod{\phi(N)}$

3. **Encryption:** $a \mapsto b = a^e \pmod{N}$

4. **Decryption:** $b \mapsto b^d \equiv a^{ed} \equiv a \pmod{N}$

If for arbitrary $N = pq$ we can find p, q , then we can directly break RSA.

8.2.2 Shor's Algoritheorem

Problem:

- **Input:** $N = pq$
- **Output:** p, q

Shor's Algoritheorem:

Classically transform the factoring problem into an order-finding problem.

1. Pick arbitrary $1 < a < N$
2. Calculate $\gcd(a, N)$:
 - (i) If $1 < \gcd(a, N) < N$, then $\gcd(a, N) = p$ or q . Done!
 - (ii) If $\gcd(a, N) = 1$, go to step 3.
3. Find order r of a modulo N .
4. If r is odd, go back to step 1 and choose different a .
5. If r is even, calculate $a^{r/2} \pmod{N}$:

- (i) If $a^{r/2} \equiv -1 \pmod{N}$, go back to step 1.
- (ii) If $a^{r/2} \not\equiv \pm 1 \pmod{N}$, go to step 6.

6. Calculate:

$$p = \gcd(a^{r/2} - 1, N)$$

$$q = \gcd(a^{r/2} + 1, N)$$

8.2.3 Why Does This Work?

If r is even:

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

So $N \mid (a^{r/2} - 1)(a^{r/2} + 1)$, which means:

$$pq \mid (a^{r/2} - 1)(a^{r/2} + 1)$$

Since p and q are prime numbers, we have several cases:

1. $p \mid (a^{r/2} - 1)$ and $q \mid (a^{r/2} + 1)$
2. $p \mid (a^{r/2} + 1)$ and $q \mid (a^{r/2} - 1)$
3. $pq \mid (a^{r/2} - 1)$
4. $pq \mid (a^{r/2} + 1)$

Cases 1 and 2 give us the factors. Cases 3 and 4 are impossible:

Proof (Case 3 is impossible). Suppose $a^{r/2} - 1 \equiv 0 \pmod{N}$. Then:

$$a^{r/2} \equiv 1 \pmod{N}$$

This contradicts the definition of r (the order should be $r/2$, not r).

Proof (Case 4 is impossible). Suppose $a^{r/2} + 1 \equiv 0 \pmod{N}$. Then:

$$a^{r/2} \equiv -1 \pmod{N}$$

This is checked in step 5, and we reject such cases.

Therefore:

$$p = \gcd(a^{r/2} - 1, N)$$

$$q = \gcd(a^{r/2} + 1, N)$$

§ 8.3 Shor's Factoring Algoritheorem

Shor's algoritheorem is a groundbreaking quantum algoritheorem for integer factorization, , it was proposed by Peter Shor in 1994. It finds the prime factors of a large composite integer exponentially faster than the best-known classical algoritheorems, posing a significant threat to classical cryptographic systems (like RSA), which rely on the difficulty of factorization.

8.3.1 *Outline of Shor's Algoritheorem*

Shor's algoritheorem combines quantum computing with classical number theory, particularly by leveraging the properties of periodic functions to solve the factorization problem. It consists of two main parts:

1. **Classical Part:** Reduce the factoring problem to finding the period of a periodic function.
2. **Quantum Part:** Use a quantum algoritheorem (Quantum Fourier Transform) to efficiently determine the period.

Problem Statement of Shor's factoring Given a composite integer N , the goal is to find its prime factors, i.e., find integers p and q such that $N = p \cdot q$.

8.3.2 *Shor's Algoritheorem*

The factorization problem is reduced to a problem of finding the period of a periodic function. The key insight is based on the following steps (They will be explained in more details during the lecture):

8.3.2.1 **Step 1: Choose a Random Number a**

Pick a random number a such that $1 < a < N$. Then, compute the greatest common divisor (GCD) of a and N , i.e., $\gcd(a, N)$.

- If $\gcd(a, N) \neq 1$, then you've already found a non-trivial factor of N , so the algoritheorem terminates.
- If $\gcd(a, N) = 1$, proceed to the next step. This ensures that a and N are coprime.

8.3.2.2 Step 2: Find the Period r

The next goal is to find the period r of the function $f(x) = a^x \bmod N$, where r is the smallest integer such that:

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

This means that the sequence $a^x \bmod N$ is periodic with period r . Finding r is the core of the quantum part of Shor's algoritheorem.

8.3.2.3 Step 3: Quantum Fourier Transform to Find r

To find r , the algoritheorem leverages the Quantum Fourier Transform (QFT). This is where the quantum speedup comes in. The QFT is used to efficiently find the period r by performing the following quantum steps:

1. Initialize two quantum registers:

- The first register is initialized to an equal superposition of all possible states $|x\rangle$ where $x \in \{0, 1, \dots, 2^n - 1\}$.
- The second register is initialized to the value $|1\rangle$, representing the current value of the modular exponentiation.

2. Apply modular exponentiation:

Apply the function $f(x) = a^x \bmod N$ to the second register, entangling it with the first. This results in the state:

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle |a^x \bmod N\rangle$$

3. Perform the Quantum Fourier Transform:

Perform the QFT on the first register. This efficiently extracts the period r from the periodic function.

4. Measure the first register:

Measuring the first register yields information that, with high probability, allows you to deduce the period r .

8.3.2.4 Step 4: Use the Period r to Find the Factors

Once you have the period r , the next task is to use it to find factors of N . If r is even, then the following holds:

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

Thus, N divides the product $(a^{r/2} - 1)(a^{r/2} + 1)$. If neither $a^{r/2} - 1$ nor $a^{r/2} + 1$ is divisible by N , then compute:

$$\gcd(a^{r/2} - 1, N) \quad \text{and} \quad \gcd(a^{r/2} + 1, N)$$

At least one of these GCD computations will yield a non-trivial factor of N .

8.3.2.5 Step 5: Repeat If Necessary

If the result from Step 4 does not yield factors, the algoritheorem can be repeated with a different randomly chosen a .

8.3.3 Example of Shor's Algoritheorem

Let's work through an example where $N = 15$ and we want to factor it using Shor's algoritheorem.

1. **Choose a random number:** Let $a = 7$.

$$\text{Compute } \gcd(7, 15) = 1$$

Since $\gcd(7, 15) = 1$, proceed to the next step.

2. **Find the period r :** Using a quantum computer, we apply modular exponentiation to find the period of $f(x) = 7^x \pmod{15}$. The result is $r = 4$, as $7^4 \equiv 1 \pmod{15}$.
3. **Use the period to find the factors:**

- Since $r = 4$ is even, compute $7^{r/2} - 1 = 7^2 - 1 = 48$.
- Compute the GCD: $\gcd(48, 15) = 3$, which is a 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 factored $15 = 3 \times 5$.

8.3.4 Quantum Speedup

Shor's algoritheorem provides an exponential speedup over classical algoritheorems. Classical factoring algoritheorems, like the general number field sieve, run in sub-exponential time. In contrast, Shor's algoritheorem runs in polynomial time, specifically:

$$O((\log N)^2 \log \log N \log \log \log N)$$

where N is the integer to be factored. This makes it feasible to factor large integers that are used in cryptographic protocols (e.g., 2048-bit RSA) on a quantum computer.

Shor's algoritheorem is one of the most important quantum algoritheorems because of its potential to break widely used cryptographic systems. By exploiting quantum parallelism and the quantum Fourier transform, it efficiently solves the integer factorization problem, which is intractable for classical computers. The development of practical, large-scale quantum computers could revolutionize cryptography, making Shor's algoritheorem a key focus of both theoretical and applied quantum computing research.

9

Harrow-Hassidim-Lloyd algoritheorem

Part IV

Quantum machine learning

Index

discrete Fourier transform, 54
discrete integral transform, 53

quantum discrete integral transform, 54