Appendix

Background, conventions, and notation

This book aims to be as modular as possible, where each numbered chapter and section can be read on its own—as such, specific notational definitions, technical terms, and acronyms are redefined the first time they appear in each numbered section.

Nevertheless, the mathematical presentation throughout this book does assume familiarity with certain concepts, techniques, and conventions that are ubiquitous within the field of theoretical quantum information science. This book is targeting an interdisciplinary community of researchers; thus, the assumed conventions and understanding of "common knowledge" will vary from reader to reader, depending on one's background and experience. In order to make the material as widely accessible as possible, here we collect some of the concepts and notational choices that are commonly used throughout this book.

For readers interested in a more complete introduction to the field and its standard conventions, we recommend the following resources:

- The definitive reference in the field of quantum computation is the book by Nielsen and Chuang [801]. Other classic textbooks include those by Kitaev, Shen, and Vyalyi [627] and Kaye, Laflamme, and Mosca [606]. A similarly general set of topics is covered in the lecture notes of Preskill [845].
- Several sets of more recent lecture notes have a specific focus on quantum algorithms, for example, by de Wolf [1051], Childs [276], and Lin [687]. See also the review article on quantum algorithms by Montanaro [774].
- Some online resources include a website containing lecture notes on quantum algorithms for data analysis and quantum machine learning by Luongo [729], the Pennylane codebook [123], and the quantum algorithm zoo [586].

A.1 Quantum systems and bra-ket notation

Basic concepts from linear algebra are an essential prerequisite for understanding quantum computation and thus for much of the technical discussion in this book. We adopt bra-ket notation to denote quantum states and the linear algebraic objects they correspond to. The state of a quantum system, such as a collection of qubits, is labeled by a *ket*, such as $|\psi\rangle$—this object corresponds to a vector in a finite-dimensional vector space. For instance, the state of a single qubit is an element of the 2D complex vector space $\mathbb{C}^2$, and can be represented by a length-2 column vector or by a *superposition* over orthonormal basis states, denoted $|0\rangle$ and $|1\rangle$:

$$|\psi\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \alpha_0|0\rangle + \alpha_1|1\rangle, \qquad \alpha_0, \alpha_1 \in \mathbb{C}\,.$$

The state of a system of n qubits is an element of the 2^n-dimensional complex vector space $\mathbb{C}^{2^n}$—the tensor product of the n individual 2D vector spaces—and can be represented in any of the following equivalent ways:

$$|\psi\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{2^n-1} \end{pmatrix} = \sum_{x=0}^{2^n-1} \alpha_x|x\rangle = \sum_{x=0}^{2^n-1} \alpha_x|x_0\rangle \otimes |x_1\rangle \otimes \cdots \otimes |x_n\rangle, \qquad \alpha_x \in \mathbb{C}\ \forall x\,,$$

where $x_i \in \{0, 1\}$ denotes the i-th bit of the integer x when x is written in binary (leading zeros are added such that x has n digits, and x_0 corresponds to the most significant bit). The orthonormal basis $|0\rangle, |1\rangle, \ldots, |2^n - 1\rangle$ is called the *computational basis*. An n-qubit quantum state is said to be a *product state* if it can be written as a tensor product of 2D states on each of the n systems

$$|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle, \qquad |\phi_i\rangle \in \mathbb{C}^2\ \forall i\,,$$

and it is said to be *entangled* if it cannot be written as a product state.

For each quantum state $|\psi\rangle$ corresponding to a column vector as above, we denote its Hermitian adjoint (i.e., the complex conjugate of its transpose) by the *bra* $\langle\psi|$, which corresponds to the row vector

$$\langle\psi| = \begin{pmatrix} \alpha_0^* & \alpha_1^* & \cdots & \alpha_{2^n-1}^* \end{pmatrix},$$

where α_x^* denotes the complex conjugate of α_x. A bra $\langle\phi| = \sum_x \beta_x^* \langle x|$ and ket $|\psi\rangle = \sum_x \alpha_x|x\rangle$ together form a braket

$$\langle\phi|\psi\rangle = \sum_{x=0}^{2^n-1} \beta_x^* \alpha_x \in \mathbb{C}\,,$$

which is simply the standard Hermitian inner product between vectors $|\phi\rangle$ and $|\psi\rangle$. The *norm* of the state $|\psi\rangle = \sum_x \alpha_x |x\rangle$ refers to the standard Euclidean vector norm, or 2-norm of the vector, given by

$$\| |\psi\rangle \| = \sqrt{\langle\psi|\psi\rangle} = \sqrt{\sum_{x=0}^{2^n-1} |\alpha_x|^2}\,.$$

A state for which $\| |\psi\rangle \| = 1$ is said to be *normalized*; in this book, kets are usually (but not always) normalized.

The above corresponds to the case for *pure* quantum states; in some instances, we consider the more general case that the state of the quantum system is *mixed*—that is, it is drawn from a probabilistic ensemble of multiple pure quantum states. In this case, an n-qubit quantum state is represented by a $2^n \times 2^n$ matrix called a *density matrix*, typically denoted by a lowercase Greek letter such as ρ. A matrix ρ is a valid quantum state if it is Hermitian and positive semidefinite. Furthermore, it is a normalized quantum state if it satisfies $\mathrm{tr}(\rho) = 1$. In this language, a pure state $|\psi\rangle$ corresponds to the rank-1 Hermitian matrix $|\psi\rangle\langle\psi|$ given by the outer product of the vector with itself.

Linear transformations of an n-qubit quantum system correspond to $2^n \times 2^n$ matrices, called *operators*. Given an operator M, there is always a singular value decomposition (SVD)

$$M = \sum_{i=0}^{2^n-1} \sigma_i |w_i\rangle\langle v_i|\,,$$

where the *singular values* σ_i are non-negative real numbers, and each of the sets $\{|w_i\rangle\}$ and $\{|v_i\rangle\}$ are orthonormal bases for the vector space. If M is Hermitian, then we may call M an *observable*, and in this case, M is guaranteed to have an eigenvalue decomposition

$$M = \sum_{i=0}^{2^n-1} \lambda_i |\psi_i\rangle\langle\psi_i|, \qquad M|\psi_i\rangle = \lambda_i |\psi_i\rangle \;\forall i$$

for which the eigenvalues $\lambda_0, \lambda_1, \ldots, \lambda_{2^n-1}$ are real, and the eigenvectors (also known as *eigenstates*) $|\psi_0\rangle, |\psi_1\rangle, \ldots, |\psi_{2^n-1}\rangle$ form an orthonormal set.

Many end-to-end problems solved by quantum algorithms boil down to estimating the expectation value of an observable, which correspond to a physical property of the system. Given an observable M and a mixed state ρ, the expectation value of M is given by $\mathrm{tr}(M\rho)$. For a pure state $\rho = |\psi\rangle\langle\psi|$, this reduces to $\langle\psi|M|\psi\rangle$.

An important observable is the *Hamiltonian*, which corresponds to the energy of the physical system. The Hamiltonian generates time evolution of the

state; that is, denoting the state at time t by $|\psi(t)\rangle$ and the time-dependent Hamiltonian by $H(t)$, the state obeys the time-dependent Schrödinger equation

$$\mathrm{i}\frac{\mathrm{d}|\psi(t)\rangle}{\mathrm{d}t} = H(t)|\psi(t)\rangle\,.$$

Here we have set the physical constant $\hbar$ to 1, which is the typical convention in the literature that we cite. The specification of an initial state $|\psi(0)\rangle$ uniquely determines $|\psi(t)\rangle$ for all other t. If $H(t) = H$ is independent of t, this is given exactly by the matrix exponential

$$|\psi(t)\rangle = \mathrm{e}^{-\mathrm{i}Ht}|\psi(0)\rangle\,, \tag{A.1}$$

and if $H(t)$ is time dependent, it is given by a time-ordered matrix exponential.

The eigenvalues of the Hamiltonian are often called the *energies*. The eigenstate corresponding to the minimal eigenvalue is called the *ground state*, and its eigenvalue is called the *ground state energy*; the eigenstates corresponding to larger energies are called *excited states*. In actual quantum systems like atomic nuclei, molecules, and materials, the system's lower energies—and especially its ground state and ground state energy—often determine its key properties; the higher excited states are rarely populated due to energy exchange with the environment favoring lower energy levels. As such, many of the relevant end-to-end problems for which quantum computing may be helpful relate to computing ground state energies and other properties of low-energy states.

One complication is that actual quantum systems in nature typically do not directly correspond to a collection of two-level qubit systems. Instead, they are modeled as discrete or continuous systems with a larger (possibly infinite) number of levels. However, the states of these systems are still described by vectors in a well-defined vector space. For example, the position of an electron in 3D space is given by an element of the vector space of square integrable functions on $\mathbb{R}^3$—in this context, the state vector $|\psi\rangle$ is often called the *wavefunction*, a term that is sometimes also used in discrete situations as well. The position of η particles in 3D space has 3η continuous degrees of freedom, and states correspond to square integrable functions on $\mathbb{R}^{3\eta}$. However, particles found in nature, such as electrons, are indistinguishable, and quantum mechanics dictates that the corresponding wavefunctions must either be antisymmetric (if the particles are fermions) or symmetric (if they are bosons) under particle exchange, which restricts the accessible vector space. For generic multiqubit systems, no such symmetry is naturally imposed. Fermionic and bosonic systems are the subjects of quantum algorithms for chemistry, condensed matter physics, and nuclear and particle physics—to simulate these and other non-qubit systems on a quantum computer, algorithmic choices must be made on

how to embed the relevant vector space into a tensor product of qubit systems. It may also be required to truncate the (possibly infinite-dimensional) vector space, incurring errors in the calculation.

A.2 The quantum circuit model

We follow standard convention and work in the quantum circuit model of quantum computation. In this paradigm, quantum computations process the information in quantum states by applying a sequence of unitary operators to the state, known as *gates*, which generalize classical Boolean logic gates—unitarity ensures that the norm of the state is preserved by the operator being applied. A single-qubit gate is given by a 2×2 unitary matrix. There are a few essential examples that occur throughout the book.

- The Pauli matrices (which are both unitary and Hermitian):

$$\sigma_x = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \sigma_y = Y = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \qquad \sigma_z = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

- The Hadamard gate H and the phase gate S:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad S = \sqrt{Z} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix},$$

- The T gate:

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

A k-qubit gate is given by a $2^k \times 2^k$ unitary matrix, and some of the essential multiqubit gates include the 2-qubit controlled NOT (CNOT) gate and the 3-qubit Toffoli gate

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \mathrm{TOFFOLI} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Given as input a 2-qubit state $|x_0 x_1\rangle$ with $x_0, x_1 \in \{0, 1\}$, the CNOT gate flips the value of x_1 conditioned on ("controlled on") x_0 being 1; the first qubit is

the control and the second is the target. The Toffoli gate is a doubly controlled NOT gate in the sense that it flips the third qubit controlled on both the first and second qubits being set to 1.

When a k-qubit gate described by the $2^k \times 2^k$ matrix V acts on a subset of qubits in an n-qubit system (with $n > k$), the gate enacted on the n-qubit system is given by a tensor product of V with the identity matrix. For example, if a single-qubit X gate acts on the second qubit of an n-qubit system, then the full $2^n \times 2^n$ unitary operator U for the gate may be decomposed as

$$U = I \otimes X \otimes I \otimes I \otimes \cdots \otimes I\,,$$

where I is the 2×2 identity matrix.

Given a fixed discrete (i.e., finite) set of gates, we may consider the set of all gates generated by the discrete gate set—that is, gates that can be formed by multiplying a sequence of gates drawn from the discrete gate set. A gate set is *universal* if it generates a dense subset of the set of all n-qubit unitary operators on the system, or equivalently, if any n-qubit unitary may be approximated to arbitrary precision by a product of gates drawn from the generating set.

For an n-qubit system, the discrete gate set formed from single-qubit gates $\{X, Y, Z, H, S\}$ on each of the n qubits combined with 2-qubit CNOT gates between any pair of qubits generates the *Clifford* group, which is a finite group that is not universal. Importantly, quantum computations on n qubits involving only Clifford gates can be efficiently simulated on a classical computer; thus, significant quantum computational speedups cannot be achieved using only Clifford gates. By adding either the T gate or the Toffoli gate to the generating set, the gate set becomes universal. The Clifford + T gate set is the most common discrete gate set considered in compilations of quantum algorithms. The Toffoli gate can be exactly decomposed into Clifford gates and T gates.

In this language, a quantum computation consists of the initialization of a quantum state (typically an n-qubit product state such as $|0\rangle^{\otimes n}$), the application of a prespecified sequence of gates, and finally, a measurement of the n qubits in the computational basis. If the normalized n-qubit initial state is $|\psi_0\rangle$ and the sequence of gates is $U_1, \ldots, U_\ell$, then the quantum state prior to the measurement is given by $|\phi\rangle = U_\ell U_{\ell-1} \cdots U_1 |\psi_0\rangle$. The measurement then produces an outcome x with probability equal to $|\langle x|\phi\rangle|^2$. This procedure can be depicted in a *quantum circuit* diagram, such as Fig. A.1.

When performing a *resource estimate* of a quantum algorithm in the Clifford + T gate set, the key quantities are the total number of qubits and gates that appear in the associated quantum circuit diagram. Occasionally, one is also interested in the circuit *depth*. For example, the circuit in Fig. A.1 acts on 3 qubits and has a total gate count equal to 12. The circuit depth is 6 since the

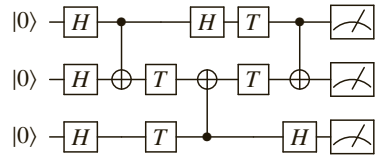

Figure A.1 Example of a quantum circuit with gates drawn from the Clifford + T gate set. Time flows from left to right. The two qubit gates are CNOT gates with the control indicated by the symbol • and the target indicated by the symbol ⊕.

gates can be *parallelized* into 6 sequential layers. When working in the Clifford + T gate set, it is common to ignore the Clifford gates and only count the non-Clifford gates, that is, the T gates (or the Toffoli gates). The main reason for this is that non-Clifford gates are more difficult to implement than Clifford gates in many (but not all) schemes for fault-tolerant quantum computing. The circuit in Fig. A.1 has a T-count of 4 and a T-depth of 2, since only 2 of the 6 layers contain T gates.

A quantum algorithm for a certain computational problem is a procedure that takes as input an instance of the computational problem and determines a quantum circuit (or multiple quantum circuits), as well as a procedure to convert the measurement result(s) into the answer to the computational problem. Since measurement outcomes are random, the answer need only be correct with high probability.

A.3 Noise in quantum gates and the NISQ era

The quantum circuit model is an idealized and abstract depiction of a quantum computer. Actual quantum computers attempt to realize this model by using physical 2-level quantum systems as qubits—several options can be considered including the electronic states of ions or neutral atoms, the spin states of electrons, the polarization states of photons, and the number of excitations in superconducting electrical circuits. Gates are applied by turning on and off external fields. While experimental control of these quantum systems has improved dramatically over time, one cannot expect gates to be performed perfectly. Much of the work in theoretical quantum information science deals with how to characterize, detect, and correct the errors that occur in noisy quantum computations.

Specifically, methods for fault-tolerant quantum computation have been developed, whereby a quantum computation can be accomplished correctly using

faulty components, provided that the noise in the system meets certain conditions. The ideal quantum circuit is referred to as the *logical* circuit, composed of logical gates and logical qubits. Each logical qubit is realized using a larger number of noisy *physical qubits*, and each logical gate requires the action of multiple faulty *physical gates*. A resource estimate for the ideal logical circuit can be converted into a resource estimate for the actual physical quantum computer, a calculation that depends on the specifics of the hardware and the fault-tolerance scheme being utilized.

Due to the resource overhead required by fault-tolerant quantum computation, researchers have also investigated the question of whether noisy quantum computers can solve interesting problems without correcting the errors. The era of quantum computing where quantum devices of tens or hundreds of qubits exist, but large-scale fault-tolerant quantum computation is not yet possible, has been referred to as the noisy intermediate-scale quantum (NISQ) era. Generally speaking, algorithms for NISQ-era quantum computers should possess a certain resilience to the inevitable occurrence of errors in the computation, often by restricting to quantum circuits with a limited gate count or gate depth. While the focus of this book is on quantum algorithms for fault-tolerant quantum computers and logical resource estimates, we comment in passing on NISQ algorithms for many of the tasks.

A.4 Big-O notation

Analyses of (classical or quantum) algorithms often focus on how the computational cost, also referred to as the *complexity*, scales with the size of the input. Inputs to a computational problem are assigned an integer size n—for example, the number of digits in a number one wishes to factor—and resource metrics such as the qubit count, gate count, and circuit depth are expressed as functions of n. Often, of primary interest is the asymptotic scaling of these complexities with n. To facilitate this, we adopt big-O notation. Given two positive-valued real functions $f(n)$ and $g(n)$, we use the following definitions:

- $f(n) = O(g(n))$ if there exist n_0, c, such that $f(n) \leq cg(n)$ whenever $n \geq n_0$.
- $f(n) = \Omega(g(n))$ if there exist n_0, c, such that $f(n) \geq cg(n)$ whenever $n \geq n_0$.
- $f(n) = \Theta(g(n))$ if $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$.
- $f(n) = \widetilde{O}(g(n))$ if there exists c such that $f(n) = O(g(n) \cdot \log^c(g(n)))$.
- $f(n) = \text{poly}(n)$ if there exists c such that $f(n) = O(n^c)$.
- $f(n) = \text{polylog}(n)$ if there exists c such that $f(n) = O(\log(n)^c)$.

Above, n_0 and c are always constants, independent of n. Intuitively, O, Ω, and Θ are used to indicate that the asymptotic growth rate of $f(n)$ is upper bounded, lower bounded, and exactly equal to that of $g(n)$, respectively. Tildes are added to suppress logarithmic factors and simplify the expressions.

We also occasionally utilize little-o notation, which has the following definitions:

- $f(n) = o(g(n))$ if *for any* constant c there exists n_0 for which $f(n) \leq cg(n)$ whenever $n \geq n_0$.
- $f(n) = \omega(g(n))$ if *for any* constant c there exists n_0 for which $f(n) \geq cg(n)$ whenever $n \geq n_0$.

Thus, little-o and little-ω communicate instances where the growth rate of $f(n)$ is *strictly* smaller than $g(n)$ and larger than $g(n)$, respectively.

While big-O and little-o notation carries the formal mathematical definitions above, in some contexts this notation is utilized in a less mathematically precise fashion. For example, big-O is occasionally used simply to indicate that constant prefactors have been omitted or that a certain quantity is roughly of the same order as another. The expression $O(1)$ is often used as a placeholder for an unspecified constant, even when there is not a well-defined growing parameter n. Meanwhile, the expression $o(1)$ is used for functions $f(n)$ that approach 0 as $n \to \infty$. The usage of Ω is often chosen to add emphasis to the fact that a certain quantity is a *lower bound* for another, even when Θ would also have been mathematically appropriate.

We also employ big-O notation for functions of multiple independent parameters. For example, if the input is an $m \times n$ matrix, we might be interested in the complexity dependence on both m and n. Another common scaling parameter is the target precision ϵ to which a certain quantity should be estimated by the quantum algorithm. Smaller ϵ typically incurs greater resources, and thus we wish to compute how the complexity scales with growing $1/\epsilon$. When two multivariate functions f and g are monotonically nondecreasing in all of the scaling parameters, there is little ambiguity about how to extend the definitions. For example:

- $f(n, m) = O(g(n, m))$ if there exist n_0, c, such that $f(n, m) \leq cg(n, m)$ whenever $n, m \geq n_0$.
- $f(n, m) = \text{poly}(n, m)$ if there exists c such that $f(n, m) = O((nm)^c)$.

Ambiguity may arise if it is possible for the function to decrease when certain scaling parameters are increased; in this case, the limiting behavior of the function can depend on the rates at which the parameters grow relative to one

another. Additional context about the range or relationship of the scaling parameters may be required to understand what is being communicated by the big-O notation.

Big-O notation enables a determination of the magnitude of a quantum speedup. Suppose the complexity of the quantum algorithm is $Q(n)$ and the complexity of the classical algorithm is $C(n)$.

- We say that the quantum algorithm has an *exponential speedup* if $Q(n) = O(\log(C(n)))$.
- We say that the quantum algorithm has a *polynomial speedup* of degree d if $Q(n) = O(C(n)^{1/d})$. If $Q(n) = \widetilde{O}(C(n)^{1/d})$, we say that the speedup is *essentially* (or *nearly*) degree-d, and often we drop these qualifiers for ease of discussion.
- If $Q(n)$ and $C(n)$ meet the criterion for a polynomial speedup for all $d \geq 1$ but not the criterion for an exponential speedup, then we say the speedup is *superpolynomial*.

Here are some examples:

- If $Q(n) = 3n$ and $C(n) = 2n^3$, there is a degree-3, or *cubic*, polynomial speedup.
- If $Q(n) = n2^{n/4}$ and $C(n) = 2^n$, there is a nearly degree-4, or *quartic*, polynomial speedup.
- If $Q(n) = n^2$ and $C(n) = \mathrm{e}^{n^{1/3}}$, then there is a superpolynomial speedup.
- If $Q(n) = 10n$ and $C(n) = 2^{n/1000}$, then there is an exponential speedup.

End-to-end analyses should ideally also assess the constant prefactors that are omitted when using big-O notation (and polylogarithmic prefactors when using big-$\widetilde{O}$), as these can still contribute significantly to the outlook of a certain application if they are especially large.

A.5 Complexity theory background

Occasionally, we make reference to concepts and results from complexity theory. Complexity theory aims to classify different computational problems based on the quantity of computational resources required to solve them. These computational complexities are often categorized solely based on whether they scale polynomially or superpolynomially with the size of the input. If the complexity of an algorithm is polynomial in the relevant parameter, it is called *efficient*.

Let x be an instance of a computational problem, which is associated with an integer length n. The desired output of the problem on input x is denoted by $M(x)$. If $M(x) \in \{0, 1\}$ is a single bit, the problem is called a *decision problem*. This enables a definition of the following important complexity classes:

- The set P contains decision problems where $M(x)$ can be computed by a deterministic classical algorithm with time complexity poly(n).
- The set BPP contains decision problems where $M(x)$ can be computed with high probability by a randomized classical algorithm (i.e., an algorithm that can make coin flips) with time complexity poly(n).
- The set BQP contains decision problems where $M(x)$ can be computed with high probability by a quantum algorithm with gate complexity poly(n).

To arrive at a precise form for the time or gate complexity, one must specify a particular computational model; in the case of the quantum circuit model, one also needs to specify a gate set, such as Clifford + T. However, for the purpose of these complexity classes, these details are generally unimportant, as they do not change which problems are in P and BQP.

Complexity theory also defines classes of problems for which the solutions are efficient to verify, even if they are not efficient to compute. Specifically, we may fix a verification algorithm with time complexity poly(n) that computes a function $M'(x, y)$ of two inputs, where x has size n and y has size poly(n). We say that y is a witness for x if $M'(x, y) = 1$. The verification algorithm can also be a quantum algorithm, in which case y can be a quantum state $|y\rangle$, which acts as the initial state for a quantum circuit in the quantum circuit model. A state $|y\rangle$ is a witness for x if the quantum verification algorithm produces output $M'(x, |y\rangle) = 1$ with high probability.

- The set NP contains decision problems for which there exists a deterministic classical verification algorithm where, on input x, there exists a witness y if and only if $M(x) = 1$.
- The set QMA contains decision problems for which there exists a quantum verification algorithm where, on input x, there exists a witness $|y\rangle$ if and only if $M(x) = 1$.

The most famous outstanding open question in complexity theory is whether P = NP, that is, whether or not there exist problems that cannot be solved efficiently, but for which solutions can be verified efficiently given a witness. It is widely believed that P ≠ NP. The prototypical example of a problem in NP that is believed not to be in P is the Boolean satisfiability problem. Here, the input is a Boolean formula φ (referred to as x above, and specified by a description of length n). The formula φ maps an input string z consisting of m

bits to an output bit $\varphi(z) \in \{0, 1\}$, where $n = \text{poly}(m)$. Given z, the output bit $\varphi(z)$ can be evaluated in time complexity $\text{poly}(n)$, and the question is whether there exists a z for which $\varphi(z) = 1$, in which case we call φ satisfiable. This problem is efficient to verify since whenever $\varphi(z) = 1$, the string z acts as a witness to the fact that φ is satisfiable. However, there are 2^m possible inputs z, so without a witness, it naively requires trying all possible inputs to determine if φ is satisfiable, a procedure which has superpolynomial-in-n time complexity.

In fact, it has been shown that the Boolean satisfiability problem is NP-hard, a term that means it is as hard as any other problem in NP. Specifically, we can make the following (slightly informal) definitions:

- A problem is NP-hard if the existence of an efficient deterministic classical algorithm for the problem would imply that P = NP.
- A problem is QMA-hard if the existence of an efficient quantum algorithm for the problem would imply that BQP = QMA.
- A problem is BQP-hard if the existence of an efficient randomized classical algorithm for the problem would imply that BPP = BQP.[1]

A problem that is both NP-hard and in NP is called NP-complete. The Boolean satisfiability problem is one example of an NP-complete problem. Similar definitions follow for the terms QMA-complete and BQP-complete.

The conjecture that P $\neq$ NP entails that all NP-hard problems do not admit efficient classical algorithms. Thus, one way to give evidence that a problem cannot be solved in polynomial time is to prove that it is NP-hard. Since it is also widely believed that BQP $\neq$ QMA and that NP $\not\subset$ BQP, showing that a problem is NP-hard or QMA-hard is strong evidence that it does not admit an efficient quantum algorithm. In the search for good quantum algorithms, these hardness results establish limits on what we expect to be possible.

On the other hand, if one can show that a problem is BQP-complete, then this is evidence that the problem exhibits a superpolynomial quantum speedup over the best possible classical algorithm. If it did not, then this would imply that BPP = BQP, which is widely believed to be false.

These complexity-theoretic results are useful guides for navigating quantum algorithms, but it is worth emphasizing that they typically deal with worst-case hardness and may not always be relevant for real-world instances of a problem. For example, preparing the ground state of a Hamiltonian consisting of local terms—which is in a sense a quantum analog of the Boolean satisfiability problem—is well known to be QMA-complete; thus, it is not expected to admit an efficient quantum algorithm in all instances. Nevertheless, for many specific

[1] Technically, due to their probabilistic nature, BPP and BQP should be defined as classes of *promise* problems to correctly formalize the notion of BQP-hard; see [418].

Hamiltonians that arise in nature, we do expect efficient ground state preparation to be possible, and this forms the basis for many proposed applications of quantum computing.

We conclude with a discussion of the concept of *oracles*. A (classical or quantum) algorithm is said to have access to an oracle g if it can *query* the oracle in a black-box fashion, by fixing an n-bit input string z and receiving the corresponding m-bit output string $g(z)$. The *query complexity* of an algorithm is the number of times it requests an output from the oracle. In quantum algorithms, one typically allows the oracle to be queried in superposition, in the sense of performing the unitary map U_g, defined by

$$U_g\left(\sum_{z=0}^{2^n-1}\sum_{w=0}^{2^m-1}\alpha_{z,w}|z\rangle|w\rangle\right)=\sum_{z=0}^{2^n-1}\sum_{w=0}^{2^m-1}\alpha_{z,w}|z\rangle|w\oplus g(z)\rangle\,,$$

where $\oplus$ denotes bitwise addition modulo 2, and the coefficients $\alpha_{z,w}$ are arbitrary complex numbers. The ability to query in superposition may give a quantum algorithm an advantage over classical algorithms that cannot do so.

Oracles play multiple conceptual roles. For one, they enable modular accounting of the costs of an algorithm. For example, in algorithms where oracle calls represent the dominant computational burden, one may count the number of oracle queries made by the algorithm and multiply this by the computational cost (e.g., time complexity or gate complexity) required to implement a single oracle query. In end-to-end analyses, it is important to instantiate all oracles using elementary operations and account for the costs of implementing them in final resource expressions.

In complexity theory, oracles also provide a mechanism for establishing more definitive separations between different models of computation. Oracles are black-box objects, and algorithms may only interact with oracles by querying them on different inputs—this makes it easier to prove a lower bound on the query complexity of an algorithm than to prove a lower bound on the time complexity or gate complexity. For example, there exists an oracle g for which it can be shown that $\mathrm{P} \neq \mathrm{NP}$ *relative to* g, even though $\mathrm{P} = \mathrm{NP}$ remains possible in the non-oracle setting (in fact, there also exist oracles relative to which $\mathrm{P} = \mathrm{NP}$).

Similarly, there are specific computational problems involving oracles, such as Simon's problem, where one can show an unconditional exponential separation between the quantum and classical query complexity required to solve the problem. Relative to this oracle, it holds that $\mathrm{BPP} \neq \mathrm{BQP}$. Such separations do not alone constitute a definitive quantum advantage in end-to-end complex-

ity, but they may capture the core mechanism by which an end-to-end analysis aims to achieve an advantage.