Undergraduate computational physics projects on quantum computing

D. Candela

Citation: American Journal of Physics **83**, 688 (2015); doi: 10.1119/1.4922296
View online: https://doi.org/10.1119/1.4922296
View Table of Contents: https://aapt.scitation.org/toc/ajp/83/8
Published by the American Association of Physics Teachers

Undergraduate computational physics projects on quantum computing

D. Candela[a)]
Physics Department, University of Massachusetts, Amherst, Massachusetts 01003

(Received 30 December 2014; accepted 22 May 2015)

Computational projects on quantum computing suitable for students in a junior-level quantum mechanics course are described. In these projects students write their own programs to simulate quantum computers. Knowledge is assumed of introductory quantum mechanics through the properties of spin 1/2. Initial, more easily programmed projects treat the basics of quantum computation, quantum gates, and Grover's quantum search algorithm. These are followed by more advanced projects to increase the number of qubits and implement Shor's quantum factoring algorithm. The projects can be run on a typical laptop or desktop computer, using most programming languages. Supplementing resources available elsewhere, the projects are presented here in a self-contained format especially suitable for a short computational module for physics students.
[http://dx.doi.org/10.1119/1.4922296]

I. INTRODUCTION

Students in an introductory quantum mechanics class are naturally fascinated with the field of quantum computation. For example, many have heard that a practicable quantum computer could defeat current methods for data encryption. It is valuable for physics students to see how quantum computation works based on the quantum mechanics they have learned: quantum measurements, multiparticle wave functions, and spin angular momentum.

This article describes a set of computational physics projects in which students write their own programs to simulate quantum computers. The projects here have been developed as a short, self-contained module or enrichment for a regular quantum mechanics course, as opposed to a full course on quantum information. All of the programming projects have been tested and some typical output is shown. To create projects that actually work and can be carried out by the target students, simplified approaches to some technical issues (not widely available in the literature, if at all) are given here.[1]

A. Classroom experience

At the University of Massachusetts, these projects have been carried out by students in a junior-level quantum mechanics course for physics majors. Some of the students in this course took an enrichment module on computational quantum mechanics. In this module students wrote their own programs, using their programming languages of preference. At weekly meetings algorithms were discussed and students presented results and discussed difficulties they were encountering. Before working on the quantum computing projects described here, the students had written programs to simulate the wave mechanics of a particle in one dimension.[2]

The prior programming experience of the students varied widely. All students had taken a required one-semester course on computational physics using MATLAB. Some students had no programming experience beyond this one required course, while others were experienced programmers. The programming platforms chosen by the students were, in order of decreasing popularity: Python (with NumPy), MATLAB, C++, and Mathematica. Their choices were driven mainly by the availability of peer help from other students.

Template or example programs were *not* provided to the students, as a goal of the computational module was to force the students to think deeply about how to simulate a quantum system numerically on a classical computer.[3] If the goal were instead to enable students to explore a wide range of quantum computations unburdened by the underlying classical computations, then packages of C++ routines, graphical interfaces, and even quantum programming languages have been developed that better serve that purpose.[4,5]

B. Quantum computing paradigm

The paradigm for quantum computing explored in these projects is the *quantum gate array* or *quantum circuit* (Fig. 1). The quantum gate array can be used to carry out Grover's search algorithm, Shor's factoring algorithm, and other calculations such as the solution of systems of linear equations.[6] A different paradigm not covered here has shown promise recently: adiabatic quantum computing.[7] This would be a valuable future extension.

C. Straightforward and more challenging projects

Many interesting projects can be carried out with $N = 3$ qubits, and the earlier projects all use $N = 3$. The 8-element state vectors and 8×8 matrices can be written out explicitly, leaving some of the thornier computational issues to the later projects. Using $N = 3$, students can explore one-qubit gates

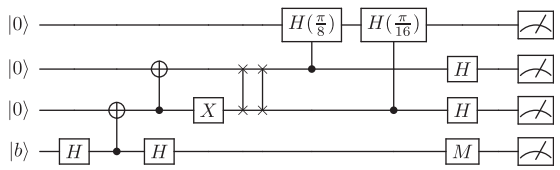

Fig. 1. Example of a quantum gate array computation. Four qubits are used ($N = 4$), shown by the four horizontal lines. The diagram is read from left to right. First the qubit register is initialized to the state $|\Psi\rangle = |000b\rangle$. (Here the quantum state $|b\rangle$ is varied to provide input data to the calculation. For many other quantum calculations, the initialization state $|\Psi\rangle$ is fixed.) Next a sequence of quantum gate operations is carried out, denoted by the various symbols between the left and right sides of the diagram. Finally the qubits are measured, as indicated by the voltmeter symbols on the right side. This quantum computer was built by Cai *et al.*, Ref. 6, using photons as the qubits and used to solve a system of linear equations.

like the Hadamard gate and the phase-shift gate and two-qubit gates like the CNOT. Also, Grover's quantum search can be fully implemented with $N=3$. A student completing the $N=3$ projects should have learned quite a lot about quantum computing, multiparticle quantum states, and quantum measurement.

The projects with $N>3$ are more challenging. As N increases the matrices become too large to enter manually and must be computer-generated. In this way, an inexpensive computer can handle up to about $N=12$ qubits. If the additional step is taken of storing the matrices in sparse form, it is possible to reach $N \approx 20$.

The last project described is Shor's factoring algorithm. Shor's algorithm is much more complicated to understand and program than Grover's algorithm. To help students over this hurdle, the smallest possible Shor's-algorithm computation (using $N=7$ to factor 15) is described in great detail. Finally, information is provided so ambitious students can try Shor's algorithm with more qubits, enabling factoring of numbers up to about 100.

D. Computer projects, programming hints, and exercises

This article is structured as eight programming projects, each preceded by explanatory material. The projects build on each other, with the exception of Project 6 (sparse matrices) which is not required for the later projects. The Appendix has programming hints based on questions that arose when the projects were tried. A few exercises intended for students are also given. They are not difficult, but they require key conceptual points to be absorbed.

E. Sources for further information

A comprehensive source on all aspects of quantum information is the classic text by Nielsen and Chuang.[8] A number of shorter texts intended for science and engineering students have appeared.[9–14] Among the most convenient resources for students carrying out a short computational module are the Wikipedia articles on quantum computing,[15] Grover's algorithm,[16] and Shor's algorithm.[17] All students should learn about the difficulties of building physical quantum computers; Ref. 18 is an accessible article current as of late 2014.

II. THE *N*-QUBIT REGISTER

The *bit* is the smallest unit of classical information, with two possible values (0 or 1). In a classical computer, a physical system like a tiny capacitor is used to store each bit, with different values of the charge on the capacitor used to represent 0 and 1. For a quantum computer, the bit is replaced by the *qubit*, stored by a quantum system with two basis vectors in its Hilbert space. An example would be a spin-1/2 particle like an electron, when only the spin can change.[19] The general quantum state of a spin-1/2 system is

$$|\Psi\rangle = a|\uparrow\rangle + b|\downarrow\rangle, \tag{1}$$

where $|\uparrow\rangle$ and $|\downarrow\rangle$ are the states with $S_z = \pm\hbar/2$. The complex amplitudes a and b obey the normalization condition $|a|^2 + |b|^2 = 1$. The S_z states are used to represent 0 and 1:

$$|0\rangle = |\uparrow\rangle, \quad |1\rangle = |\downarrow\rangle. \tag{2}$$

The significance of a superposition state like $(|\uparrow\rangle + |\downarrow\rangle)/\sqrt{2}$ or $(|\uparrow\rangle + i|\downarrow\rangle)/\sqrt{2}$ is less obvious.

An N-qubit register is N of these 2-state systems, considered together to be one quantum system. Quantum mechanics tells us that a system of N particles should be described by a single joint quantum state $|\Psi\rangle$. For example, a 3-qubit register has $2^N = 8$ basis states,

$$\begin{aligned} |000\rangle &= |\uparrow\rangle|\uparrow\rangle|\uparrow\rangle \\ |001\rangle &= |\uparrow\rangle|\uparrow\rangle|\downarrow\rangle \\ |010\rangle &= |\uparrow\rangle|\downarrow\rangle|\uparrow\rangle \\ &\vdots \\ |111\rangle &= |\downarrow\rangle|\downarrow\rangle|\downarrow\rangle. \end{aligned} \tag{3}$$

In the basis state $|001\rangle$, qubits 1 and 2 have $S_z = +\hbar/2$, while qubit 3 has $S_z = -\hbar/2$. The general quantum state of a 3-qubit register is a superposition of the eight basis states,

$$|\Psi\rangle = a|000\rangle + b|001\rangle + \cdots + g|110\rangle + h|111\rangle, \tag{4}$$

where the complex amplitudes satisfy $|a|^2 + |b|^2 + \cdots + |h|^2 = 1$ for normalization.[20] The quantum state will be stored in the computer as a column vector[21] of eight complex numbers:

$$|\Psi\rangle = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}. \tag{5}$$

Exercise: Compute the largest N for which the quantum state of an N-qubit register can be stored on a 2014-era laptop computer with 4 GB $= 4 \times 10^9$ bytes of RAM. Assume each complex amplitude $a, b\ldots$ requires 16 bytes of storage. How does the amount of memory necessary change if the number of qubits is doubled?

III. THE QUANTUM GATE ARRAY COMPUTER

The type of quantum computation that will be simulated in these projects is shown in Fig. 1. First the N-qubit register is initialized to a definite quantum state. Next, a series of quantum gate operations is applied to the register. Typically, each gate operates on only a few (1–3) of the qubits. Finally, the qubits are measured. To simulate a quantum computer, it is necessary to carry out each of the three stages (initialization, quantum gate operations, and measurements) on a quantum state represented in the computer in the form of Eq. (5).

The initialization step is simple. Most often the N-qubit register will be initialized to one of its 2^N basis states, with all N qubits in states of definite S_z. This is represented by having one of the 2^N amplitudes $a, b, c\ldots$ equal to 1, with the rest of the amplitudes equal to zero. For example, the initial state might be chosen to be

$$|\Psi\rangle = |011\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{6}$$

Another type of state encountered in quantum computing is the "cat state"

$$|\Psi\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1/\sqrt{2} \end{bmatrix}. \tag{7}$$

This highly nonclassical state is named for Schrödinger's cat, because it is an equal superposition of two very different classical states.

IV. MEASUREMENTS OF THE *N*-QUBIT REGISTER

Usually one measures S_z for each of the N qubits. Each measurement gives $S_z = \pm\hbar/2$. Therefore, one is sure to find the register to be in one of the 2^N basis states of Eq. (3), no matter what the state $|\Psi\rangle$ was before the measurement. The *probability* of measuring each basis state is given by the square magnitude of the corresponding amplitude, for example

$$P(|110\rangle) = |\langle 110|\Psi\rangle|^2 = |g|^2. \tag{8}$$

If the result of the measurement is $|110\rangle$ then quantum mechanics tells us that $|\Psi\rangle$ *collapses* to the basis state $|110\rangle$, and subsequent measurements will always give $|110\rangle$. Therefore, quantum algorithms must be cleverly devised so that a single measurement of the N-qubit register gives the needed results.

A. Programming project 1: Simulate measurement of the N-qubit register

In this section, the first and last parts of the quantum gate array computer are set up: Initialization of the qubit register (left side of a diagram similar to Fig. 1) and measurement of the qubits (right side of the diagram). In later sections quantum gates will be added. For the first part of this article, the number of qubits is $N = 3$, so there will be three horizontal lines in the diagrams for computations. Here is what the computer program must do:

(1) Allocate a column vector with $2^N = 8$ complex entries and set it to the desired initial state $|\Psi\rangle$ [Eq. (5)]. Some initial states to try are listed below.
(2) Produce a result by making a simulated measurement of S_z for the three qubits. The result will be *random*, but the probability of getting each possible result must follow the quantum-mechanical law of Eq. (8). The result will always be one of the eight basis states $|000\rangle \ldots |111\rangle$ and should be printed out in this form (see Appendix for hints). For example, a result of $|101\rangle$ shows that the first and third qubits were measured to be 1, while the second qubit was measured to be zero.
(3) Repeat step 2 many times to see how variable the results are.[22,23] It is helpful to make the program list what the most frequent results are.

Once the program is working, try the following:

- Set the initial state $|\Psi\rangle$ to one of the basis states, for example, $|011\rangle$ [Eq. (6)]. With this initial state, every measurement should give the result $|011\rangle$.
- Set the initial state to the cat state, Eq. (7). Now, at random, either all of the qubits should be 0 or all of the qubits should be 1. Thus, each qubit is equally likely to be measured 0 or 1, but all three qubits are always measured to have the same value.
- Set the initial state to an equal superposition of all 2^N basis states,

$$|\Psi\rangle = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}. \tag{9}$$

This type of state will prove important later for Grover's algorithm (Sec. VI) and Shor's algorithm (Sec. IX). With this state the measured value for each qubit is both random and uncorrelated with the other qubits. Thus, all possible results from $|000\rangle$ to $|111\rangle$ should occur, each with equal frequency to within statistical fluctuations.

V. QUANTUM GATES

Classical computers are made up of logic gates with names like AND, OR, and NOT, connected by wires that carry the states of classical bits from the output of one gate to the input of the next (Fig. 2). The quantum gate array computer (Fig. 1) is similar: quantum gates (symbols in the central part of Fig. 1) take quantum signals in from the left and produce outputs to the right. However, there are important differences:

- Each horizontal line in Fig. 1 represents a qubit as time proceeds, not a wire in space. Therefore, the quantum gates are actually a series of operations carried out one after another in time.

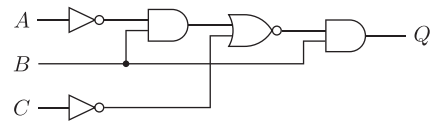

Fig. 2. A classical logic circuit. The open shapes are logic gates (here two NOTs, two ANDs and a NOR) and the lines are wires carrying logic signals from one gate to another. This circuit takes Boolean inputs A, B, and C and produces Boolean output Q.

- The operations carried out by the quantum gates must *not* collapse the state of the system the way a measurement would.

A physical system that is not measured or otherwise disturbed obeys Schrödinger's equation,

$$i\hbar\frac{d|\Psi\rangle}{dt} = \hat{\mathcal{H}}|\Psi\rangle. \tag{10}$$

In a physical quantum computer,[6,24] the Hamiltonian $\hat{\mathcal{H}}$ must vary with time so that Eq. (10) results in the desired quantum gate operations. Applying Schrödinger's equation over a time period is always equivalent to applying a *unitary operator* to $|\Psi\rangle$. This gives the following important rule: Every quantum gate operation is carried out by applying some unitary operator to the state vector,[25]

$$|\Psi\rangle \Leftarrow \hat{U}|\Psi\rangle, \tag{11}$$

where $\hat{U}^\dagger\hat{U} = \hat{U}\hat{U}^\dagger = \hat{I}$ (the definition of "unitary") and $\hat{I}$ is the identity operator.

A. The Hadamard gate

One of the most commonly used quantum gates is the *Hadamard gate*, symbolized by a box with an H in it. There are four Hadamard gates in Fig. 1. A Hadamard gate acts on a single qubit and is described by the matrix

$$\hat{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{12}$$

Note that the symbol $\hat{H}$ is used in this paper for a Hadamard gate, not the Hamiltonian $\hat{\mathcal{H}}$.

Exercise: Show that $\hat{H}$ is unitary. The adjoint $\hat{H}^\dagger$ of a matrix is found by transposing it and then taking the complex conjugate.

The basis states are

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{13}$$

Applying a Hadamard gate to either basis state gives a superposition of *both* basis states, for example,

$$\hat{H}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle). \tag{14}$$

A Hadamard gate can also take a superposition of the basis states and "put it back together" into a single basis state, for example,

$$\hat{H}\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |1\rangle. \tag{15}$$

Exercise: Verify Eqs. (14) and (15) by matrix multiplication.

B. The phase shift gate

Another important one-qubit gate is the *phase shift gate*. The matrix for this gate is

$$\hat{R}_\theta = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}. \tag{16}$$

Usually θ has a specified value like $\pi/4$ or π, and the symbol for this gate is a box with θ in it. Consider the result of applying a phase-shift gate to a superposition of the basis states:

$$\text{If } |\Psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix}, \quad \text{then } \hat{R}_\theta|\Psi\rangle = \begin{bmatrix} a \\ e^{i\theta}b \end{bmatrix}. \tag{17}$$

This shows that the probabilities $|a|^2$ and $|b|^2$ for finding the qubit in the two basis states are unchanged by the phase-shift gate, but the relative *phase* of the two amplitudes is changed. Although a quantum phase cannot be measured directly, the phase shift in Eq. (17) can have an effect if further quantum gate operations are applied to the qubit before it is measured.

C. Applying a gate to the *N*-qubit register

The N-qubit register has 2^N basis states, so a quantum gate is represented by a $2^N \times 2^N$ unitary matrix. How is this larger matrix computed from the 2×2 matrix of Eq. (12) or Eq. (16)? The answer is written formally as a *tensor product*, denoted by $\otimes$. With $N = 3$ qubits the operation that applies a Hadamard gate to qubit 1 is

$$\hat{H}^{(1)} = \hat{H}\otimes\hat{I}\otimes\hat{I}. \tag{18}$$

The superscript on $\hat{H}^{(1)}$ indicates that the gate operates on qubit 1. It is computed as the tensor product of the Hadamard operator with two copies of the identity operator. Conceptually, the identity operators "do nothing" to qubits 2 and 3. Similarly, the operators that apply a Hadamard gate to qubit 2 and qubit 3 are, respectively,

$$\hat{H}^{(2)} = \hat{I}\otimes\hat{H}\otimes\hat{I}, \tag{19}$$

$$\hat{H}^{(3)} = \hat{I}\otimes\hat{I}\otimes\hat{H}. \tag{20}$$

It is not necessary at this point to fully understand the tensor-product notation of Eqs. (18)–(20) (discussed further in Sec. VIII). It will suffice to have the following matrices for a Hadamard gate operating on qubit 1, 2, or 3:

$$\hat{H}^{(1)} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}, \tag{21}$$

$$\hat{H}^{(2)} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}, \tag{22}$$

$$\hat{H}^{(3)} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}. \quad (23)$$

Comparing these with Eq. (12), the matrix elements of $\hat{H}$ are distributed around these larger matrices following a definite pattern. Similarly, matrices can be written for a phase-shift gate on qubit 1, 2, or 3. Here is the matrix for a phase-shift gate on qubit 3:

$$\hat{R}_\theta^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{i\theta} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & e^{i\theta} \end{bmatrix}. \quad (24)$$

The matrix for $\hat{R}_\theta^{(3)}$ follows the same pattern as $\hat{H}^{(3)}$, with four copies of the original 2×2 matrix arrayed along the diagonal of the 8×8 matrix.

D. Programming project 2: First full quantum computations

Set up and run the four quantum computations shown in Fig. 3. In Fig. 3(a), $|\Psi\rangle$ is initialized to the state $|000\rangle$, then a Hadamard gate is applied to qubit 2 by multiplying $|\Psi\rangle$ on the left by $\hat{H}^{(2)}$:

$$|\Psi\rangle \Leftarrow \hat{H}^{(2)}|\Psi\rangle. \quad (25)$$

Equation (22) gives the matrix for $\hat{H}^{(2)}$. A more complex computation is performed by successively multiplying the quantum state on the left by operators for the specified gates. For example, the gates in Fig. 3(b) are carried out by the operation

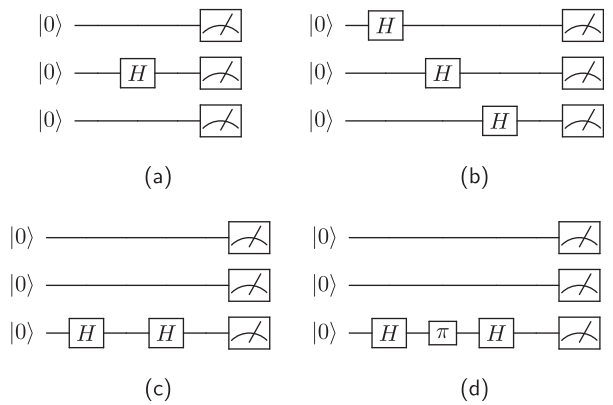

Fig. 3. Four different three-qubit calculations to try.

$$|\Psi\rangle \Leftarrow \hat{H}^{(3)}\hat{H}^{(2)}\hat{H}^{(1)}|\Psi\rangle. \quad (26)$$

The gates are applied to $|\Psi\rangle$ in the same order shown in the quantum circuit diagram.[26] They appear in reverse order in Eq. (26) because each successive matrix is applied on the left. As before, the computation is repeated to see what is definite and what varies randomly.

- In Fig. 3(a), a Hadamard gate is applied to qubit 2. From Eq. (14), this puts qubit 2 in an equal superposition $|0\rangle$ and $|1\rangle$. Therefore, the result of the calculation should vary randomly between the two possibilities, $|000\rangle$ and $|010\rangle$.
- In Fig. 3(b), Hadamard gates are applied to qubits 1, 2, and 3. This leaves each of the three qubits equally likely to be in the states $|0\rangle$ and $|1\rangle$, with no correlations between the qubits. Knowing the state of qubit 1, for example, gives no information about the states of qubits 2 and 3. The result of the calculation should vary randomly between all eight possibilities, $|000\rangle, |001\rangle, \ldots, |111\rangle$. Putting the N-qubit register into an equal superposition of all 2^N basis states by applying a Hadamard gate to each qubit is one of the important building blocks of many quantum algorithms.
- In Fig. 3(c), two successive Hadamard gates are applied to the same qubit. As shown above, the Hadamard gate splits a single quantum amplitude into two but it can also put two quantum amplitudes back together into one. In this case, the second Hadamard undoes the effect of the first one, so the result of the calculation should always be $|000\rangle$.
- In Fig. 3(d), again two Hadamard gates are applied to the same qubit. But now the phase-shift gate $\hat{R}_\theta$ with $\theta = \pi$ is applied between the two Hadamard gates, shown by a box with π in it. As in the previous case, the result of the calculation is perfectly definite, but now the result is always $|001\rangle$. The net effect of the three gates has been to flip qubit 3 from $|0\rangle$ to $|1\rangle$. Equations (14) and (15) show why this works. This computation shows that using a phase-shift gate to change the phase of quantum amplitudes can indeed change the results of the calculation.

With these programs, a complete quantum computer has been simulated. Section VI shows how, with a few additions, Grover's quantum search can be carried out.

VI. GROVER'S QUANTUM SEARCH ALGORITHM

Consider a classical database like a phone book with D names, each followed by a phone number. If a phone number is given, how many entries of the phone book must be looked at to find the corresponding name? Looking through the phone book one might be lucky and find the phone number on the first try, or unlucky and need to look at all D entries. The *average* number of entries consulted will be halfway between these extremes, $D/2$.

Here is another classical search problem: A logical function of N Boolean (T or F, meaning True or False) variables like Fig. 2 is specified, and it is known that the function is T only for one specific combination of the inputs (this is true for Fig. 2). To find this combination among the $D = 2^N$ possibilities for the inputs, one goes through the D possibilities until the one is found that yields

T. Again, on average it is necessary to go through $D/2$ of the possibilities.

Using Grover's quantum search algorithm,[8,16,27] the average number of times the database must be consulted is reduced from $D/2$ to $(\pi/4)\sqrt{D}$. This is an enormous savings if the size D of the database is large. However, there is a catch: The database or logic function must be put in the form of a *quantum oracle*. A classical oracle returns a one-bit answer (yes or no) in response to a question ("Is the 437th entry in the phone book 545–0111?" or "Is the logic function T for inputs TFTFFTFT?"). A *quantum* oracle must accept a quantum superposition of questions, and return the corresponding quantum superposition of answers.

By using a superposition, it is possible to ask all possible questions of the oracle simultaneously. It might seem that a quantum oracle only needs to be consulted once to find the correct question, but measuring the qubits collapses the quantum state so the complete superposition cannot be determined. Nevertheless, Grover showed how to get the needed information from a quantum oracle in far fewer tries than would be needed classically.

A. Grover's algorithm:Details

Figure 4 shows the quantum circuit for Grover's algorithm. The size of the database that can be searched is $D = 2^N$, where N is the number of qubits as usual. The diagram is drawn for $N = 3 \Rightarrow D = 8$. In addition to Hadamard gates, an operator $\hat{O}$ for the quantum oracle and a special operator $\hat{J}$ are needed. The oracle knows what the right question is to give a "yes" or T response, and it functions as follows:

- If the oracle is given one of the $D - 1$ wrong questions, it returns its input state unchanged. For example, say the right question (represented as a binary number) is 110. Then, since 100 is not the right question, $\hat{O}|100\rangle = |100\rangle$.
- Conversely, if the oracle is given the right question, it returns its input multiplied by -1. Continuing the example above, this means $\hat{O}|110\rangle = -|110\rangle$.

Therefore the quantum oracle matrix is like the identity matrix, except that it has -1 as the diagonal element for the correct question. When that question is 110, the matrix is

$$\hat{O} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{27}$$

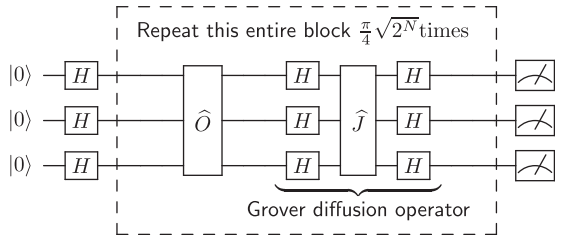

Fig. 4. Quantum circuit diagram for Grover's search algorithm.

since $|110\rangle$ is the seventh basis vector.[28] The oracle $\hat{O}$ is clearly unitary: when multiplied by its adjoint it gives the identity matrix. The operator $\hat{J}$ in Fig. 4 is like the oracle, except that the -1 element is always in the first position:

$$\hat{J} = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{28}$$

Here is how Grover's algorithm works. The first set of Hadamard gates in Fig. 4 creates an equal superposition of all 2^N basis states. This superposition is passed to the oracle, effectively asking all 2^N questions at the same time. The oracle flips the sign of the amplitude for the correct question. The "Grover diffusion operator" in Fig. 4 is designed to convert this *phase* difference, which is unmeasurable, into a *magnitude* difference that will show up when the qubits are measured.[16] It might seem better to use an oracle that encodes its output in a directly usable form,

$$\hat{O}' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{29}$$

If $\hat{O}'$ is applied to a superposition of all basis states, the result is immediately the basis state for the correct question $|110\rangle$. Unfortunately, a matrix like $\hat{O}'$ is not unitary and so does not represent a possible operation in a quantum computer.

B. Programming project 3: Implement Grover's quantum search

Implement the Grover search algorithm for $N = 3$ qubits, as diagrammed[26] in Fig. 4. The needed matrices are given above. The optimum number of repetitions should be close to $(\pi/4)\sqrt{2^N}$ but of course it must be an integer.

- Set the oracle so the correct question is 110 as above, and with the optimum number of repetitions of the oracle plus Grover diffusion block. Run the computation many times. The measured result should be $|110\rangle$ more than 90% of the time.[16]
- Change the oracle so a different question is correct.
- Change the number of repetitions. The percentage of time that the result is correct should decrease with either more or fewer repetitions.

VII. GATES OPERATING ON MORE THAN ONE QUBIT

In Fig. 1, there are six gates that connect *two* horizontal lines. Each of these gates therefore operates simultaneously on two qubits. Consider the second two-qubit gate in Fig. 1, a "controlled NOT" or CNOT gate. This gate has a solid dot on qubit 3 (numbering qubits from 1 at the top) connected by a link to a $\oplus$ symbol on qubit 2. The NOT function changes $|0\rangle$ to $|1\rangle$, and $|1\rangle$ to $|0\rangle$. This CNOT gate applies a NOT function to qubit 2 if and only if qubit 3 is in the state $|1\rangle$. That is, qubit 3 *controls* whether or not a NOT function is applied to the qubit 2. Some other types of controlled gates can be seen in Fig. 1. In each case, a solid dot on the controlling qubit is linked to a symbol indicating a conditional action on another qubit. The matrix for a CNOT gate is

$$\hat{C}_{\text{NOT}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{30}$$

Exercise: Verify that this matrix carries out the CNOT function of Fig. 5(a). It helps to write the four basis vectors $|00\rangle \ldots |11\rangle$ down the side and across the top of the matrix.

Given any one-qubit gate $\hat{U}$, it is possible to construct a two-qubit gate in which qubit 1 controls whether or not the operation $\hat{U}$ is applied to qubit 2. This "controlled-U" gate has the symbol shown in Fig. 5(b) and has the matrix[29]

$$\hat{C}_U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{0,0} & U_{0,1} \\ 0 & 0 & U_{1,0} & U_{1,1} \end{bmatrix}. \tag{31}$$

In the $N = 3$ computer any of the three qubits can be the controlling bit and either of the remaining two qubits can be the controlled bit, so there are six possible CNOT gates. It will suffice to write out two of the six possibilities, first with qubit 2 controlling qubit 3,

$$\hat{C}^{(2,3)}_{\text{NOT}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \tag{32}$$

then with qubit 2 controlling qubit 1:

$$\hat{C}^{(2,1)}_{\text{NOT}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{33}$$

The algorithm used to generate these matrices is given in Sec. VIII.

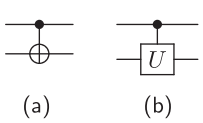

Fig. 5. (a) Symbol for a CNOT gate, where the first bit is the controlling bit. (b) Symbol for the general controlled-U gate. The CNOT is a controlled-U with U = NOT.

A. Programming project 4: Computations using CNOT gates

Try the quantum computations shown in Fig. 6, using the CNOT operations of Eqs. (32) and (33).

(a) In this diagram, there is a Hadamard gate on qubit 2, before qubit 2 is used to control qubit 3. The Hadamard gate puts qubit 2 in a superposition of $|0\rangle$ and $|1\rangle$ [Eq. (14)]. When this superposition is used to control qubit 3, that qubit will also end up in a superposition. When the computation is carried out multiple times, the output should vary randomly between $|000\rangle$ and $|011\rangle$. The measured values for qubits 2 and 3 are both random, but they are correlated with each other. This is called an entangled state of qubits 2 and 3.

(b) This computation creates a cat state. A Hadamard gate is used to put qubit 2 into a superposition of $|0\rangle$ and $|1\rangle$. Then two CNOTs are used to correlate qubits 3 and 1 with qubit 2. The measured results should vary randomly between $|000\rangle$ and $|111\rangle$, as they did with the cat state of Eq. (7).

(c) This duplicates an earlier calculation. First a Hadamard gate puts qubit 2 into a superposition of $|0\rangle$ and $|1\rangle$. Then a second Hadamard gate converts the superposition back into the pure state $|0\rangle$. With this computation, the measured result is always $|000\rangle$.

(d) This is like the previous diagram, but now a CNOT has been used to observe the state of qubit 2 when it is in a superposition of $|0\rangle$ and $|1\rangle$. Qubit 3 is flipped by the CNOT to agree with the state of qubit 2, and when qubit 3 is measured it reveals what the state of qubit 2 was between the two Hadamard gates. Observing the state of qubit 2 destroys the quantum coherence of the

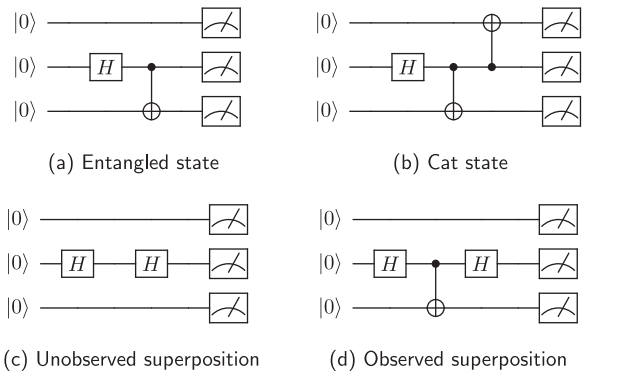

Fig. 6. Quantum computations using CNOT gates. See text for details.

superposition, so the second Hadamard gate cannot put qubit 2 back into the pure state $|0\rangle$. The measured results for qubits 2 and 3 will be random and uncorrelated (equal probabilities for observing $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$).

The CNOT gate in Fig. 6(d) does not "do anything" to qubit 2, which always ends up in the same state it started in. Thus, the destruction of coherence cannot be attributed to a direct action of the CNOT on the thing being observed, qubit 2; rather, it is built into the way quantum mechanics describes the states of multiparticle systems.

Optional exercise: In Fig. 6(d), the state of qubit 2 is not really observed until the qubits are measured. Figure out how to undo the observation of qubit 2 before the qubits are measured by adding one more gate, and verify that with this modification the results are the same as for Fig. 6(c).

Figure 6(d) also illustrates a peculiarity of quantum computation. Classically the only effect of the CNOT would be on qubit 3. In a quantum calculation, the CNOT might be used specifically for its effect on qubit 2.

This concludes the projects using $N=3$ qubits. In Sections VIII and IX the number, the number of qubits is increased, resulting in more challenging programming tasks. This is a necessary prelude to trying Shor's factoring algorithm, which requires at least $N=7$.

VIII. INCREASING THE NUMBER OF QUBITS

The challenge here is to make N as large as possible. It is addressed in two stages. In Sec. VIII A, the techniques used above are extended to $N>3$ simply by using bigger vectors and matrices. On a 2012-era personal computer, it was possible to reach $N=12$ before the matrices became too large to store in the computer. In Sec. VIII C, sparse-matrix techniques are used to increase N beyond this limit by taking advantage of the many zeros in the matrices.

A. Matrices for gates with an arbitrary number of qubits

Now the computer program must generate the $2^N \times 2^N$ matrices for gates, and a more compact notation is needed to represent these matrices. Let p and q be binary digits, each 0 or 1. A 2×2 matrix $\hat{M}$ can be represented in the form $M_{p,q}$,

$$\hat{M} = \begin{bmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{bmatrix}. \tag{34}$$

Exercise: Show that the 2×2 identity matrix is $I_{p,q}=\delta_{pq}$, the Kronecker delta.

Using $N=3$ to illustrate, let $[pqr]$ be a 3-digit binary number, with each of p, q, and r equal to 0 or 1. Using this notation, the tensor product of Eq. (19) can be translated as follows:

$$\hat{H}^{(2)} = \hat{I}\otimes\hat{H}\otimes\hat{I} \iff H^{(2)}_{[pqr],[p'q'r']} = \delta_{pp'}H_{q,q'}\delta_{rr'}. \tag{35}$$

The Hadamard matrix operates on the indices q,q' corresponding to the second qubit, while Kronecker deltas give the identity operators for all the other qubits.

Here is an example: Let us assume that matrix indices start from zero,[29] so the element in the seventh row and fifth column of $\hat{H}^{(2)}$ is $H^{(2)}_{6,4}$. Using Eq. (35), this element is computed as

$$H^{(2)}_{6,4} = H^{(2)}_{[110],[100]} = \delta_{11}H_{1,0}\delta_{00} = H_{1,0} = \frac{1}{\sqrt{2}}, \tag{36}$$

in agreement with Eq. (22). Notice how 6 and 4 have been written in binary as 110 and 100.

Exercise: Use the bottom line of Eq. (35) to write out the 8×8 matrix for $\hat{H}^{(2)}$ and check that it agrees with Eq. (22). It helps to write $[pqr]$ ranging from [000] to [111] down the side of the 8×8 matrix, and $[p'q'r']$ ranging from [000] to [111] across the top.

Shor's algorithm will also require two-qubit gates. Consider a CNOT gate in an $N=3$ system in which qubit 2 controls qubit 1, as used in Fig. 6(b). The equivalent of Eq. (35) is

$$C^{(2,1)}_{\mathrm{NOT}[pqr],[p'q'r']} = C_{\mathrm{NOT}[qp],[q'p']}\delta_{rr'}. \tag{37}$$

As with Eq. (35), we have Kronecker deltas for every qubit that the CNOT does *not* operate on, in this case only the third qubit with indices r,r'. Since the second qubit with indices q,q' is the controlling qubit for the CNOT, it is used as the *first* index for the C_{NOT} matrix. For example, the element in the seventh row and third column of $\hat{C}^{(2,1)}_{\mathrm{NOT}}$, which is[29] $C^{(2,1)}_{\mathrm{NOT}\,6,2}$, is

$$\begin{aligned} C^{(2,1)}_{\mathrm{NOT}\,6,2} &= C^{(2,1)}_{\mathrm{NOT}[110],[010]} \\ &= C_{\mathrm{NOT}[11],[10]}\delta_{00} = C_{\mathrm{NOT}\,3,2} = 1, \end{aligned} \tag{38}$$

in agreement with Eq. (33). As above, 6 and 2 have been written in binary as 110 and 010, but now after the correct bits have been picked out according to Eq. (37) the binary numbers 11 and 10 are converted back to 3 and 2 to index the $\hat{C}_{\mathrm{NOT}}$ matrix, Eq. (30).

B. Programming project 5: Handle an arbitrary number of qubits

Program Grover's search for an arbitrary number of qubits N. Check that the optimum number of Grover iterations is $(\pi/4)\sqrt{2^N}$, for $N>3$. Then increase N as much as possible.

Table I shows results obtained on a 2012-era personal computer with 8 GB of RAM. Each additional qubit increased both the memory needed and the time to complete the calculation by a factor of about 4. With 13 qubits, the memory of the PC was exceeded.

C. Using sparse matrices

Many of the elements in the matrices of Eqs. (21)–(28) are zero. Matrices like this that have only a small fraction of nonzero elements are called *sparse*. It is wasteful of computer memory and time to store and compute with all these zeros. Some computer languages have sparse-matrix functions built in. This section explains how to implement sparse matrices without built-in functions, but either route can be pursued to increase N.

The $2^N \times 2^N$ matrix for a one-qubit gate [Eqs. (21)–(23)] has at most two nonzero elements per row. Similarly, the

Table I. Grover's algorithm with full matrices.

qubits N	Memory to store matrices	Time to compute matrices	Time to do Grover calculation
10	104 MB	53 s	4.3 s
11	450 MB	3.9 min	37 s
12	1.96 GB	17.6 min	3.8 min
13	**Failed**	…	…

$2^N \times 2^N$ matrix for a two-qubit gate has at most four nonzero elements per row; CNOT gates [Eqs. (32)–(33)] have only one nonzero element per row. Therefore, the Hadamard matrix of Eq. (21) can be stored as

$$\hat{H}^{(1)} = \begin{bmatrix} \left(0, \frac{1}{\sqrt{2}}\right), \left(4, \frac{1}{\sqrt{2}}\right) \\ \left(1, \frac{1}{\sqrt{2}}\right), \left(5, \frac{1}{\sqrt{2}}\right) \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ \left(3, \frac{1}{\sqrt{2}}\right), \left(7, -\frac{1}{\sqrt{2}}\right) \end{bmatrix}. \tag{39}$$

The first row of Eq. (39) indicates that the nonzero elements in row 0 are $1/\sqrt{2}$ in column 0 and $1/\sqrt{2}$ in column 4.[29] (It may be more convenient to store the integer column indices and complex matrix elements in two separate arrays.) A diagonal matrix can be stored even more compactly, if desired, since it is known that the column index equals the row index for the nonzero elements.

D. Programming project 6: Use sparse matrices to increase *N*

Write functions to generate sparse matrix structures, or figure out how to use built-in sparse matrix functions. Then carry out Grover's algorithm with N as large as possible.

Table II shows results obtained with the same PC as used for Table I. Both the amount of memory used and the time needed to compute the matrices were drastically less than when full matrices were used. Extrapolating from this table, the storage limit of 8 GB would be reached at $N = 21$ and the calculation would take a week to finish.

Table II. Grover's algorithm with sparse matrices.

qubits N	Memory to store matrices	Time to compute matrices	Time to do Grover calculation
10	1.82 MB	0.172 s	3.6 s
11	4.4 MB	0.48 s	11.4 s
12	9.9 MB	1.54 s	34 s
13	22 MB	5.4 s	1.78 min
14	46 MB	23 s	5.4 min

IX. SHOR'S QUANTUM FACTORING ALGORITHM

Pick two large prime numbers P_1, P_2, and compute their product $C = P_1 P_2$; for example, $241 \times 683 = 164603$. Commonly used methods of data encryption depend on the fact that it takes a lot of computer power to find the factors P_1, P_2 if only C is known (using bigger numbers than this example, of course). Shor's algorithm[17,30] can factor a large composite (i.e., not prime) number C in far fewer steps than any known classical algorithm, potentially rendering some current encryption methods useless.

A. Shor's algorithm step by step

To state Shor's algorithm, these two concepts are needed:

- The *greatest common divisor* of two integers $\gcd(p, q)$ is the largest integer that divides both p and q with zero remainder. For example, $\gcd(12, 18) = 6$ and $\gcd(12, 35) = 1$.
- The *modular congruence* $p \equiv q (\mathrm{mod}\ C)$ means $p - q$ is an integer multiple of C. For example, $35 \equiv 13 (\mathrm{mod}\ 11)$. This can also be written $p(\mathrm{mod}\ C) = q(\mathrm{mod}\ C)$ where "$p(\mathrm{mod}\ C)$" means the *remainder* when p is divided by C. So $35(\mathrm{mod}\ 11) = 2$, and $13(\mathrm{mod}\ 11) = 2$.

Given a composite integer C, the following steps will find a nontrivial factor of C (nontrivial means other than 1 or C):[31]

(1) Check that C is odd and not a power of some smaller integer. If C is even or a power, then a factor of C has been found and we are done.
(2) Pick any integer a in the range $1 < a < C$.
(3) Find $\gcd(a, C)$. If by luck the gcd is greater than 1, then a factor of C has been found (the gcd) and we are done.
(4) Find the smallest integer $p > 1$ such that $a^p \equiv 1(\mathrm{mod}\ C)$.
(5) If p is odd, or if p is even and $a^{p/2} \equiv -1(\mathrm{mod}\ C)$, go back to step 2 and pick a different a.
(6) The numbers $P_\pm = \gcd(a^{p/2} \pm 1, C)$ are nontrivial factors of C.

Example: $C = 15$

(1) 15 is odd and is not a power of a smaller integer, so proceed.
(2) Arbitrarily pick $a = 7$.
(3) $\gcd(7, 15) = 1$, so proceed.
(4) Try p's starting with 2:
   - $7^2 = 49$ and $49(\mathrm{mod}\ 15) = 4 \neq 1$.
   - $7^3 = 343$ and $343(\mathrm{mod}\ 15) = 13 \neq 1$.
   - $7^4 = 2401$ and $2401(\mathrm{mod}\ 15) = 1$, so the result of this step is $p = 4$.
(5) $p = 4$ is even, and $7^{4/2} = 49$. Since $49 \not\equiv -1(\mathrm{mod}\ 15)$, it is not necessary to go back and pick a different a.
(6) $P_+ = \gcd(50, 15) = 5$ and $P_- = \gcd(48, 15) = 3$ are the sought-for factors of 15.

One should imagine implementing Shor's algorithm for a very large number C. A *classical* computer can quickly determine if C is a power, and quickly compute the gcd of two large numbers using Euclid's algorithm. So the only thing for which a quantum computer is needed is step 4.

B. The quantum part of Shor's algorithm: Period-finding

Step 4 is called "period finding," because the function $f(x) = a^x(\mathrm{mod}\ C)$ is periodic with period p. In the example $C = 15$, $a = 7$, $f(x)$ has period 4:

$$
\begin{aligned}
f(0) &= 7^0 \pmod{15} = 1\\
f(1) &= 7^1 \pmod{15} = 7\\
f(2) &= 7^2 \pmod{15} = 4\\
f(3) &= 7^3 \pmod{15} = 13\\
f(4) &= 7^4 \pmod{15} = 1\\
f(5) &= 7^5 \pmod{15} = 7\\
f(6) &= 7^6 \pmod{15} = 4\\
&\vdots
\end{aligned}
\tag{40}
$$

Figure 7 shows the quantum circuit used to find p. The qubit register is divided into two parts, the x-register with L qubits initialized to the state $|0\ldots0\rangle$ and the f-register with M qubits initialized to the state $|0\ldots01\rangle$. The steps in the calculation are:

(1) Apply a Hadamard gate to each of the L qubits in the x-register, as indicated by the box with $H^{\otimes L}$. This puts the x-register in a superposition of all 2^L possible x values.
(2) Based on the value in the x-register, multiply the f-register by $a^x \pmod C$. This leaves the f-register containing $f(x)$.
(3) Measure the f-register. It doesn't matter if the f-register is measured at this point, or later when the x-register is measured, or never, but the explanation of Fig. 7 is simpler if the f-register is measured as shown.
(4) Perform an inverse quantum Fourier transform (IQFT) on the x-register. Since the Fourier transform of a function has peaks at the frequencies present in the function, the IQFT will make it possible to find the period = 1/frequency of $f(x)$. The IQFT is discussed in more detail below.
(5) Measure the output $\tilde{x}$ of the IQFT. Shor proved that the measured $\tilde{x}/2^L$ equals approximately s/p, where s is an unknown integer. This information is used to find p. For example, if $\tilde{x}/2^L = 0.32 \approx \frac{1}{3} = \frac{2}{6} = \frac{3}{9}\cdots$ it can be guessed that p is one of 3, 6, 9,…. These are checked to see which one satisfies $a^p \equiv 1 \pmod C$ and is therefore the true period p.

C. Shor's algorithm with seven qubits

The smallest numbers satisfying step 1 of the algorithm (odd composite integers that are not powers of another integer) are $C = 15, 21, 33, 35, 39, \ldots$. To date, physical quantum computers using Shor's algorithm have factored the numbers 15 and 21. Vandersypen *et al.*[24] made physical quantum computers with $N = 7$ using nuclear spins as the qubits. They used Shor's algorithm to factor $C = 15$ using $L = 3$ and $M = 4$.

Figure 8 shows the quantum circuit for the $N = 7$ Shor's-algorithm quantum computation, based on the design of Vandersypen *et al.* In addition to elements like Hadamard gates familiar from earlier projects there are two new elements:

- There are three *controlled phase shift gates*, all in the IQFT block. These are phase shift gates as used earlier, but now controlled by another qubit.
- There are three quantum gates using the x-register bits ℓ_2, ℓ_1, ℓ_0 to control the f-register bits m_3, m_2, m_1, m_0. The gate controlled by bit ℓ_k conditionally multiplies the f-register by $a^{2^k} \pmod{15}$. Together the three gates multiply the f-register by $a^{\ell_0+2\ell_1+4\ell_2} \pmod{15} = a^x \pmod{15} = f(x)$, as required for step 2 of the period-finding calculation.[32]

D. Programming project 7: Implement Shor's algorithm with seven qubits

Set up the quantum computation diagrammed in Fig. 8 and use it to factor $C = 15$. The computation should work for all values of a that pass steps 2 and 3 of Shor's algorithm as listed in Table III. The measured values of the seven qubits labeled on the right side of Fig. 8 should be used to print out the two numbers $\tilde{x}/2^L = (\tilde{x}_2\tilde{x}_1\tilde{x}_0)/2^L$ and $f = (f_3f_2f_1f_0)$. For example, if the measured output is $|0110100\rangle$ then $f = 0100_2 = 4$ from the last four qubits and $\tilde{x} = 110_2 = 6$ from the first three qubits in reversed order, so $\tilde{x}/2^L = 6/8 = 0.75$. Here's what should happen:

- The measured f should always be one of the p different values listed in Table III. For example, when $a = 7$, f should vary randomly among 1, 7, 4, and 13. This is just a check, as the f value is not used.
- The measured $\tilde{x}/2^L$ should come out close to s/p for some integer s. This is the actual output of the program, used to find the period p.

For this project it is necessary to compute $2^7 \times 2^7$ matrices for the new gates in Fig. 8:

Controlled phase-shift gates: The 4×4 controlled phase-shift gate is given by Eq. (31) with the phase shift matrix [Eq. (16)] used as the U submatrix. Then the full 128×128 matrix is constructed using an equation like Eq. (37).

Gates to compute $f(x)$: The 128×128 matrices for the gates that compute $f(x)$ are *permutation matrices*, which means that each column is all 0's except for one 1, and the 1 is in a different row in each column [like Eqs. (32)–(33)]. Using $C = 15$:

(1) Compute $A_0 = a \pmod{15}$, $A_1 = a^2 \pmod{15}$, and $A_2 = a^4 \pmod{15}$.
(2) Start with the first controlled gate in Fig. 8, controlled by ℓ_0. In each column $k = 0\ldots127$ of the matrix,[29] the

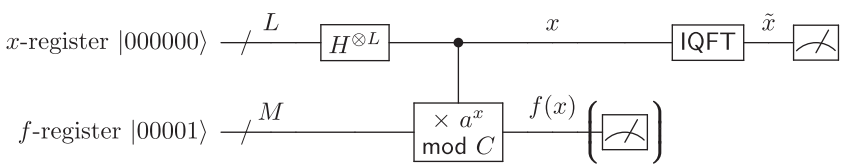

Fig. 7. The quantum circuit for the period-finding part of Shor's algorithm. The top line marked with a slash and L denotes a group of L qubits. Similarly the bottom line denotes a group of M qubits.

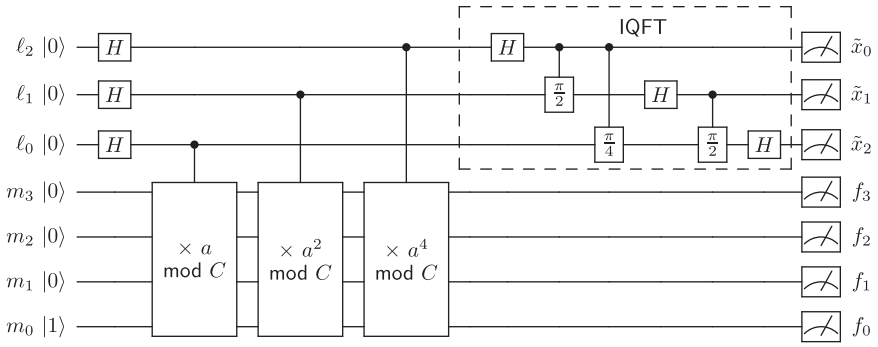

Fig. 8. Quantum circuit to implement Shor's algorithm with $N = 7$ qubits.

entries are all 0's except for a 1 in some row j. To compute j:

(a) Write the column k as a 7-bit binary number $k = \ell_2\ell_1\ell_0 m_3 m_2 m_1 m_0$ as on the left side of Fig. 8.
(b) If $\ell_0 = 0$, then $j = k$. This puts the 1 in column k on the diagonal, as in an identity matrix. (Conceptually if $\ell_0 = 0$, the gate does nothing to the f-register.)
(c) If $\ell_0 = 1$, then express the four-bit binary number $m_3 m_2 m_1 m_0$ as an integer f. If $f \geq 15$ (i.e., if $f \geq C$), then again set $j = k$.
(d) If $\ell_0 = 1$ and $f < 15$, compute a new f value $f' = A_0 f \pmod{15}$. Write this in binary as $f' = m'_3 m'_2 m'_1 m'_0$. Then j is given by the 7-bit binary number $j = \ell_2\ell_1\ell_0 m'_3 m'_2 m'_1 m'_0$. [Conceptually if $\ell_0 = 1$ then the f-register is multiplied by $A_0 \pmod{15}$.]

(3) Use the same procedure to compute the other two controlled gates, substituting ℓ_1, ℓ_2 for ℓ_0 and A_1, A_2 for A_0.

Optional exercise: (a) Show that a permutation matrix as defined above is unitary. (b) (subtle) Show that the procedure given above results in a permutation matrix, and therefore a unitary matrix.

This is all that is needed to try out Shor's algorithm. Table IV shows the results of 100 quantum measurements when this was tried with $a = 7$. The measured $\tilde{x}/2^L$ values were always close to $\frac{0}{4}, \frac{1}{4}, \frac{2}{4}$, or $\frac{3}{4}$. Since these are multiples of $\frac{1}{4}$, they give a guess $p = 4$ for the period (the correct answer).

E. How the quantum part of Shor's algorithm works

In Figs. 7 and 8, computations are carried out on the f-register, then no use is made of this register. A similar situation occurred earlier with Fig. 6(d): using qubit 2 to control qubit 3 affected the measured value of qubit 2. Similarly, using the x-register to control the f-register affects the measured value of the x-register. Figure 9 shows how this works.

- Figure 9(a) represents the quantum state $|\Psi\rangle$ after the Hadamard operation on the x-register and the multiplication of the f-register by $f(x)$. Due to the Hadamard operation, every possible x value is equally likely. Due to the controlled-multiply operation, for every x value the f-register contains one of the p possible $f(x)$ values. (In Fig. 9(a), these are 1, 7, 4, 13.) Thus, $|\Psi\rangle$ at this point in the calculation is an equal superposition of many terms, each represented by one of the solid dots in Fig. 9(a). The dashed curves in Figs. 9(b) and 9(c) show that if the f-register is measured, the result will be 1, 4, 7, or 13, while if the x-register is measured, the result will be any integer.
- Now measure the f-register, and assume the result happens to be 7. According to basic quantum principles, a measurement causes the state $|\Psi\rangle$ to *collapse* so that it agrees with the measured value. Therefore after measuring $f = 7$ only the dots with $f = 7$ [circled in Fig. 9(a)] remain in the superposition.
- After measuring f, all possible values of the x-register are no longer equally likely. Rather, $P(x)$ is nonzero only for the x's in the circled dots, so $P(x)$ has become a comb of peaks separated by the period $p = 4$ [solid curve in Fig. 9(c)].
- The IQFT takes the Fourier transform of this comb-like function. Because the function has period 4, its Fourier transform will contain the frequency $\omega = 1/4$ and its harmonics $\omega = 2/4$ and 3/4. In a discrete FT, the frequencies are read out as $\omega = \tilde{x}/2^L$. Therefore when the x-register is measured the probability will have peaks at $\tilde{x}/2^L = s/4 = s/p$ for all integer harmonic numbers s, which is the claimed result.

Table III. Usable a values for $C = 15$.

a	p	$f(x)$ values
2	4	1, 2, 4, 8
4	2	1, 4
7	4	1, 7, 4, 13
8	4	1, 8, 4, 2
11	2	1, 11
13	4	1, 13, 4, 7
14	2	1, 14

Table IV. Shor's-algorithm results for $N = 7$, $C = 15$, $a = 7$.

$\tilde{x}$	$\tilde{x}/2^L$	Measurements with this result
0	0.0	27
1	0.125	0
2	0.25	25
3	0.375	0
4	0.5	30
5	0.625	0
6	0.75	18
7	0.875	0

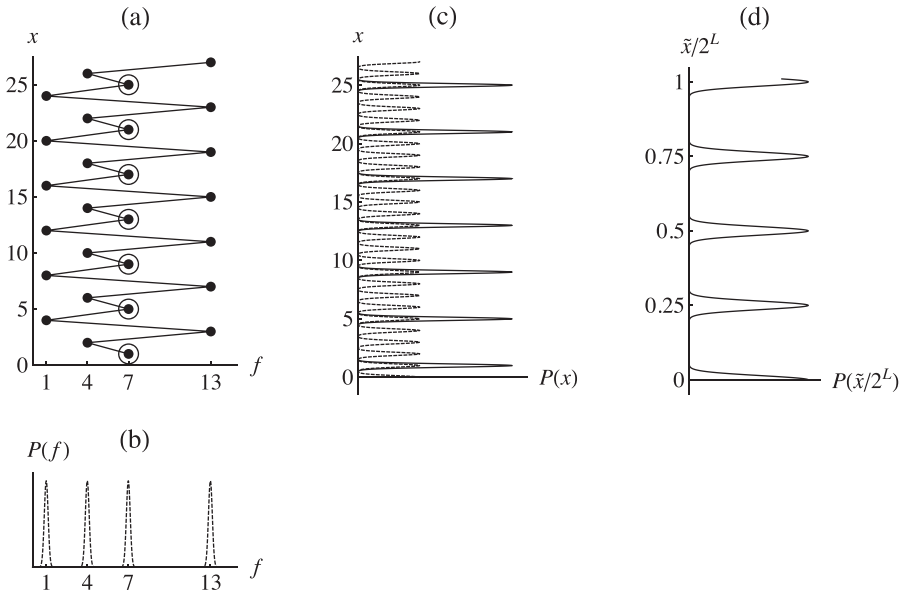

Fig. 9. How the quantum period-finding calculation works. See text for details.

More on the IQFT: In a classical discrete FT or IFT on an input register with L values, the FT is applied to L input numbers giving L output numbers. By contrast, here there are 2^L different x values represented by the 2^L quantum amplitudes in the superposition in the x-register, and the IQFT operates on all 2^L numbers at once. For the large prime numbers used in encryption, L could be several thousand and a classical Fourier transform on 2^L numbers (or even storing 2^L numbers classically) is completely impossible.

F. Implementing Shor's algorithm for arbitrary N

By using $N>7$ it is possible to factor numbers C greater than 15. The total number of qubits is $N=L+M$. The f-register holds binary values of $a^x(\text{mod } C)$, so M must satisfy $2^M \geq C$. To be confident[17] of finding the period p, the size L of the x-register should satisfy $2^L \geq C^2$. Table V shows the required L and M. However, as seen above Shor's algorithm can work with smaller L than this table implies: for $C=15$, $L=3 \rightarrow N=7$ was used.

Figure 10 shows how to build the QFT for arbitrary L from Hadamard and controlled phase shift gates. For Shor's algorithm, the inverse quantum Fourier transform (IQFT) is needed. Quantum computing provides a way to invert a function that is not available classically. Since every quantum gate carries out a reversible operation, the QFT can be inverted by reversing the *time order* in which the gates are applied: compare the IQFT in Fig. 8 with the bottom diagram in Fig. 10.

Table V. Numbers C suitable for Shor's-algorithm factoring and numbers of qubits required.

C	L	M	N	C	L	M	N
$15=3\times5$	8	4	12	$57=3\times19$	12	6	18
$21=3\times7$	9	5	14	$63=3\times3\times7$	12	6	18
$33=3\times11$	11	6	17	$65=5\times13$	13	7	20
$35=5\times7$	11	6	17	$69=3\times23$	13	7	20
$39=3\times13$	11	6	17	$75=3\times5\times5$	13	7	20
$45=3\times3\times5$	11	6	17	$77=7\times11$	13	7	20
$51=3\times17$	12	6	18	$85=5\times17$	13	7	20
$55=5\times11$	12	6	18	$87=3\times29$	13	7	20

G. Programming project 8: Implement Shor's algorithm with N > 7

Program Shor's algorithm for arbitrary N, and see how big a number can be factored. Table VI shows a few results obtained in this way. In each case, the L used was smaller than is listed in Table V. In the results for $C=39$, increasing L from 6 to 8 gave measured $\tilde{x}/2^L$ values much closer to the expected values $\frac{1}{6}, \frac{2}{6}, \frac{3}{6}\ldots = 0.1667, 0.3333, 0.5000, \ldots$.

H. Using continued fractions to guess the period

The output of the period-finding calculation is $\tilde{x}/2^L \approx s/p$ where s is an unknown integer, and is used to guess possible values for p. With a physical quantum computer, one would be trying to find p after running the computation just once. Trial values for p are typically derived from $\tilde{x}/2^L$ by using *continued fractions*. Having set up the computer to factor $C=87$ and choosing $a=13$, from Table VI one might get the single measured result $\tilde{x}/2^L = 0.6426$. A continued-fraction expansion[34] of this number is

$$0.6426 = \cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{1+\cfrac{1}{19+\cdots}}}}}} \approx 1, \frac{1}{2}, \frac{2}{3}, \frac{7}{11}, \frac{9}{14}, \frac{178}{277}, \ldots. \tag{41}$$

This gives a sequence of rational numbers closer and closer to 0.6426. From $0.6426 \approx 7/11$ one tries $p = 11, 22\ldots$ and from $0.6426 \approx 9/14$ one tries $p = 14, 28\ldots$ Even for large

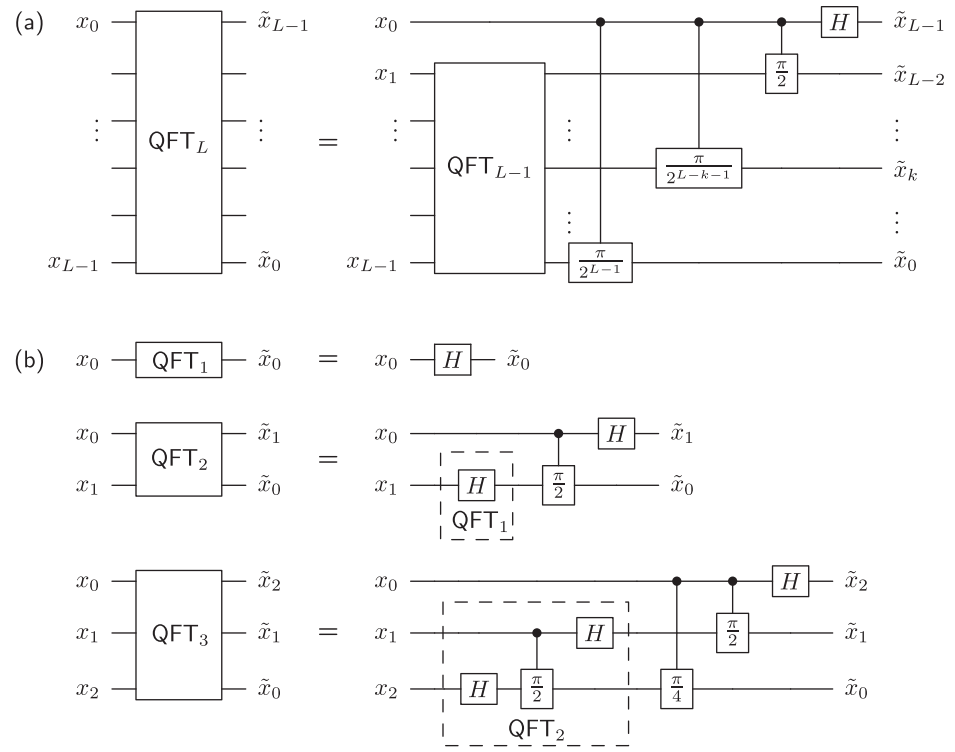

Fig. 10. (a) Recursive definition of the quantum Fourier transform (QFT) for L qubits in terms of the QFT for $L-1$ qubits. See Ref. 8 or 33 for a derivation. (b) QFT's for $L=1$, 2, and 3 from the recursive definition.

C, a classical computer can check which of the trial p's satisfies $a^p \equiv 1 (\mathrm{mod}\, C)$. This gives the correct $p=14$, from which the factors of 87 are computed to be $\gcd(13^{14/2} \pm 1, 87) = 29, 3$.

X. POSTSCRIPT: IS QUANTUM COMPUTATION TRULY POWERFUL?

As the projects in this paper illustrate, the classical resources needed to simulate a quantum calculation grow exponentially with the number of qubits (Tables II and VI). This suggests that a quantum computer should be exponentially more powerful than a classical computer with the same number of bits. However, there are classical algorithms to search databases and factor numbers that are far more efficient than simulating Grover's algorithm and Shor's algorithm, making the classical vs. quantum comparison subtle.[35]

Computer scientists classify problems by the way the classical computation time t increases with the size S of the input (e.g., digits in the number to be factored). The complexity class **P** consists of problems for which t grows no faster than a power of S. The class **NP** includes **P** but also includes more difficult problems thought to require time that grows faster than any power of the input size, for example, $t \propto e^S$. For such problems, even modest input size can result in an impossibly long classical computation time.

Factoring numbers is thought to lie in **NP** but not **P** and thus to be exponentially difficult on a classical computer. Conversely, the time needed to factor a number on a quantum computer using Shor's algorithm is only polynomial in the input size.[36] In this sense, at least, it seems that gate-array quantum computation as described in this article is inherently more powerful than classical computation. Finally, there are other models of quantum computation[15] (adiabatic quantum computing, quantum annealing, quantum emulation…) that may also offer possibilities for exponentially exceeding the capabilities of any classical computer. Realizing these vast computational powers in the real world will depend on continued progress in building physical quantum computers.[18]

APPENDIX: PROGRAMMING HINTS

1. Binary numbers and array indices

In some computer languages (Python, C++,…), array indices start from 0. If the variable `psi` is an 8-element array representing the quantum state $|\Psi\rangle$ in Eq. (5), `psi[0]` $= a$

Table VI. Shor's-algorithm results using sparse matrices.

C	L	M	N	$a \Rightarrow p$	Storage used	Calculation time	Most frequently measured $\tilde{x}/2^L$ values (occurrences in 100 measurements)
39	6	6	12	10 6	8 MB	5.8 s	0.8281 (16), 0.0 (16), 0.5 (15), 0.3281 (14), 0.6719 (11), 0.1719 (8)…
39	8	6	14	10 6	44 MB	72 s	0.5 (18), 0.0 (18), 0.1680 (16), 0.3320 (13), 0.8320 (11), 0.6680 (11)…
87	9	7	16	13 14	198 MB	32 min	0.6426 (9), 0.1426 (9), 0.0723 (8), 0.8574 (6), 0.5 (6), 0.2148 (6)…

and `psi[7]` $= h$. In these languages, to find the basis state represented by `psi[j]`, convert j into a binary number. For example, $j = 6$ corresponds to $|110\rangle$.

In other computer languages (MATLAB, Mathematica,…), array indices start from 1. In these languages, Eq. (5) implies `psi(1)` $= a$ and `psi(8)` $= h$, so to find the basis state represented by `psi(j)`, convert $j - 1$ into a binary number. Now $j = 7$ corresponds to $|110\rangle$.

2. Convention used for numbering qubits

In a gate-array diagram like Fig. 1, qubit 1 is the top line and qubit N is the bottom line. For a basis state like $|011\rangle$, qubit 1 is the leftmost bit (0), and qubit N is the rightmost bit (1). It follows that basis states for an N-qubit register are read from top to bottom in a gate-array diagram. Thus in Fig. 1 the register is initialized to the state $|000b\rangle$.

3. Hints for project 1: Measurement of the N-qubit register

To carry out a quantum measurement on the state $|\Psi\rangle$ in Eq. (5),

(1) Pick a random number r with $0 \le r < 1$ (a built-in function in most languages).
(2) Carry out the following steps (r is fixed during these steps; q is a new variable):
(i) Set $q = |a|^2$. If $r < q$ the result is $|000\rangle$; otherwise proceed to the next step.
(ii) Set $q \Leftarrow q + |b|^2$. If $r < q$ the result is $|001\rangle$; otherwise proceed to the next step.[25]
(iii) Set $q \Leftarrow q + |c|^2$. If $r < q$ the result is $|010\rangle$; otherwise proceed to the next step....(three steps omitted)...
(vii) Set $q \Leftarrow q + |g|^2$. If $r < q$ the result is $|110\rangle$; otherwise the result is $|111\rangle$.

These steps are designed to give Eq. (8): The probability of measuring $|000\rangle$ is $|a|^2$, etc. They can be programmed as above with seven IF statements, or a more elegant program can be devised that works for any N.

4. Hints for projects 2–4: Full quantum computations with $N = 3$ qubits

To carry out Fig. 3(a) (a Hadamard gate applied to qubit 2), multiply the column vector Ψ_k that represents $|\Psi\rangle$ on the left by the matrix for $\hat{H}^{(2)}$ given in Eq. (22):

$$\Psi_j \Leftarrow \sum_{k=0}^{7} H_{j,k}^{(2)} \Psi_k. \tag{A1}$$

This equation[29] carries out the operation shown abstractly by Eq. (25). All of the quantum computations in this paper are done by carrying out multiple operations of the form of Eq. (A1) in the order shown in the quantum gate array diagram.[22,26]

5. Hints for project 5: Using an arbitrary number of qubits N

It is necessary to find or write functions to convert integers into lists of their binary digits, and vice versa. For example, for $N = 7$ the conversions for the integer 23 are

$$23 \iff (0,0,1,0,1,1,1). \tag{A2}$$

Here is a way to implement Eq. (35), to create the $2^N \times 2^N$ matrix $\hat{H}^{(n)}$ that applies the gate $\hat{H}$ to qubit n [in Eq. (35), $N = 3$ and $n = 2$]. The program should be checked by printing out the $N = 3$ Hadamard matrices [Eqs. (21)–(23)].

(i) Start with a $D \times D$ array initialized to all zeros, where $D = 2^N$.
(ii) For each element $H_{j,k}^{(n)}$ with $j = 0 \ldots D-1$ and $k = 0 \ldots D-1$, convert j and k into lists of 1's and 0's giving binary representations of these numbers.[29]
(iii) Find the elements of each list at location n. These are the numbers q, q' in Eq. (35).
(iv) To enforce the Kronecker deltas in Eq. (35), only put a nonzero element in the big array when the bit lists for j and k are equal to each other at all locations *other* than n. When this condition is met, put into the location j, k of the big array the q, q' element of the 1-qubit Hadamard array [Eq. (12)].

6. Hints for project 7: Shor's algorithm with $N = 7$ qubits

For the three gates implementing modular multiplication, print out the sum of each row and the sum of each column, and verify that every sum is 1.

Start without the IQFT, to check that the first part of the calculation is working. The measured x should be random, and the measured f should be $f(x)$. For example, with $a = 7$ when $\tilde{x} = 6 = 110_2$, $x = 011_2 = 3$ so from Eq. (40) f should be $f(3) = 13$.

7. Hints for project 8: Shor's algorithm with $N > 7$ qubits

The recursive definition of the QFT of Fig. 10 must be implemented for arbitrary L, with the gates in reverse order to give the IQFT. Use Fig. 10 to write a routine (two nested FOR loops) to print out and check the *names* of the gates for any L. Then add the actual gates to the computation. The output for $L = 3$ should be:

```
Hadamard on 1
1 controls pi/2 on 2
1 controls pi/4 on 3
Hadamard on 2
2 controls pi/2 on 3
Hadamard on 3
```

[a]Electronic mail: candela@physics.umass.edu

[1]Although tensor products are mentioned, it is not assumed that the reader is familiar with and able to compute tensor products. Therefore, an alternative method is given based on bitwise manipulation of matrix indices (Sec. VIII A). Similarly, a bitwise method is used to compute the modular-arithmetic gates needed for Shor's algorithm (Sec. IX D). A good understanding of binary numbers should suffice for both situations.

[2]With four weekly meetings (one third of the semester) on quantum computing, students completed the three-qubit projects including Grover's algorithm and a variable amount of the more advanced material. Consistently reaching Shor's algorithm would require additional meetings.

[3]Working code for these projects in Mathematica is available by email request to the author.

[4]"List of QC simulators," Quantiki, <http://www.quantiki.org/wiki/List_of_QC_simulators>.

[5]Brian Hayes, "Programming your quantum computer," Am. Sci. **102**, 22–25 (2014).

[6]X.-D. Cai *et al.*, "Experimental quantum computing to solve systems of linear equations," Phys. Rev. Lett. **110**, 230501-1–5 (2013).

[7]Nanyang Xu *et al*., "Quantum factorization of 143 on a dipolar-coupling nuclear magnetic resonance system," Phys. Rev. Lett. **108**, 130501-1–5 (2012).
[8]Michael A. Nielsen and Isaac L. Chuang, *Quantum Computation and Quantum Information* (Cambridge U.P., Cambridge, 2000).
[9]Michel Le Bellac, *A Short Introduction to Quantum Information and Quantum Computation* (Cambridge U.P., Cambridge, 2006).
[10]N. David Mermin, *Quantum Computer Science: An Introduction* (Cambridge U.P., Cambridge, 2007).
[11]Giuliano Benenti, Giulio Casati, and Giuliano Strini, *Principles of Quantum Computation and Information—Vol.1: Basic Concepts* (World Scientific, Singapore, 2004).
[12]Eleanor Rieffel and Wolfgang Polak, *Quantum Computing: A Gentle Introduction* (MIT Press, Cambridge, MA, 2011).
[13]Anirban Pathak, *Elements of Quantum Computation and Quantum Communication* (Taylor and Francis, Boca Raton, FL, 2013).
[14]David McMahon, *Quantum Computing Explained* (Wiley, Hoboken, NJ, 2008).
[15]"Quantum computer," Wikipedia, <http://en.wikipedia.org/wiki/Quantum_computer>.
[16]"Grover's algorithm," Wikipedia, <http://en.wikipedia.org/wiki/Grover's_algorithm>.
[17]"Shor's algorithm," Wikipedia, <http://en.wikipedia.org/wiki/Shor's_algorithm>.
[18]Elizabeth Gibney, "Quantum computer quest," Nature **516**, 24–26 (2014).
[19]The particle could have a spatial wave function $\Psi(x)$ that stays fixed, for example.
[20]Although electrons are identical fermions, it is not necessary to antisymmetrize $|\Psi\rangle$ if the electrons are not allowed to exchange places with one another.
[21]In most computer languages $|\Psi\rangle$ will simply be an eight-element array of complex numbers. In matrix multiplications $|\Psi\rangle$ must be treated as a column vector: when multiplied on the *left* by a square matrix, the result is another column vector.
[22]If desired, the program can compute $|\Psi\rangle$ once just before the measurement and then use this $|\Psi\rangle$ as the input for multiple measurements. This would not be possible in a physical quantum computer since measuring $|\Psi\rangle$ would collapse the quantum state.
[23]The probability for any measurement result can be computed from Eq. (8) without making simulated measurements. However, there is pedagogical value in having the simulated quantum system act like a real quantum system. Seeing the results from repeating the quantum measurement a small number of times is closer to the operation of a physical quantum computer.
[24]Lieven M. K. Vandersypen *et al*., "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance," Nature **414**, 883–887 (2001).
[25]The left-arrow notation, e.g., $a \Leftarrow a+1$, is used to denote the assignment of a new value to the program variable a. In most computer languages this is written a=a+1.
[26]Gates can be moved along their lines as long as the order of operations *on each line* is unchanged. So for Fig. 3(b) the three Hadamard gates could be applied in any order.
[27]Lov K. Grover, "A fast quantum mechanical algorithm for database search," arXiv:quant-ph/9605043 (1996), <http://arxiv.org/abs/quant-ph/9605043>.
[28]One can object that to write Eq. (27), we need to know the answer to the search problem. For a useful application of Grover's algorithm, the oracle would be a quantum calculation that can be constructed without knowing the correct answer. For example, the oracle could implement the logic circuit of Fig. 2 using quantum gates. See Ref. 10, pp. 88–89, for further discussion.
[29]This is written for array indices starting at zero as in Python or C++ (see Appendix).
[30]Peter W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," arXiv:quant-ph/9508027 (1995), <http://arxiv.org/abs/quant-ph/9508027>.
[31]See Ref. 8 or 17 for proof. The branch of mathematics that studies primes, factoring, and other aspects of the integers is called number theory. It is well known from number theory that steps 1–6 factor the number C. Shor's contribution was to prove that step 4 can be done much more quickly on a quantum computer than classically.
[32]In physical quantum computers, gates typically operate on 1–3 qubits at a time, making this part of the period-finding calculation more complicated than the IQFT.
[33]"Quantum Fourier transform," Wikipedia, <http://en.wikipedia.org/wiki/Quantum_Fourier_transform>.
[34]To compute a continued-fraction expansion like Eq. (41), invert the number, subtract off the integer part (which is the next number in the continued fraction), invert the fractional part, subtract off the integer part, and so on.
[35]Troels F. Rønnow *et al*., "Defining and detecting quantum speedup," Science **345**, 420–424 (2014).
[36]This is a simplified description of computational complexity classes. For example, it has not been proven that **NP** is actually a different class than P. Based on Shor's algorithm, factoring is thought to lie in the class **BQP** (larger than **P** but not equal to **NP**) of problems that can be solved with high probability in polynomial time on a quantum computer. See "Computational complexity theory," Wikipedia, <http://en.wikipedia.org/wiki/Computational_complexity_theory>, and Ref. 8, Sec. 3.2.