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

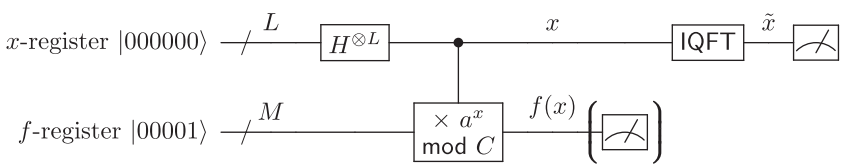

Fig. 7. The quantum circuit for the period-finding part of Shor's algorithm. The top line marked with a slash and L denotes a group of L qubits. Similarly the bottom line denotes a group of M qubits.

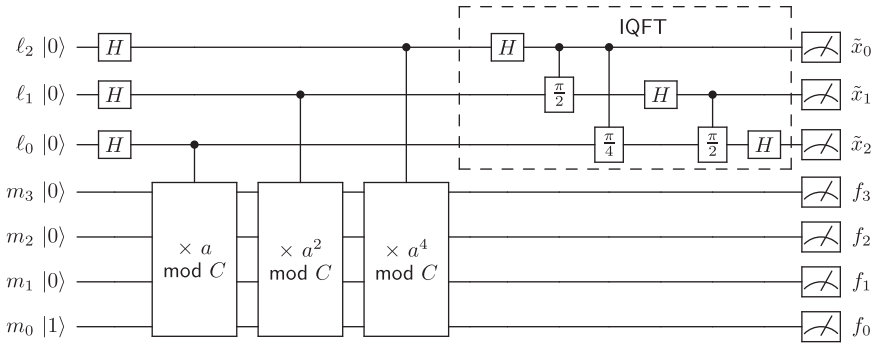

Fig. 8. Quantum circuit to implement Shor's algorithm with $N = 7$ qubits.

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

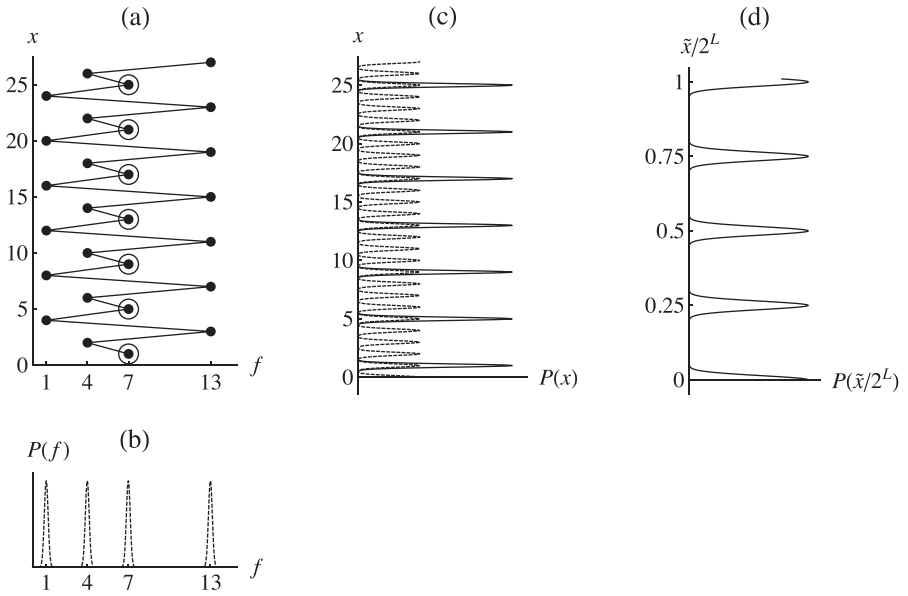

Fig. 9. How the quantum period-finding calculation works. See text for details.

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

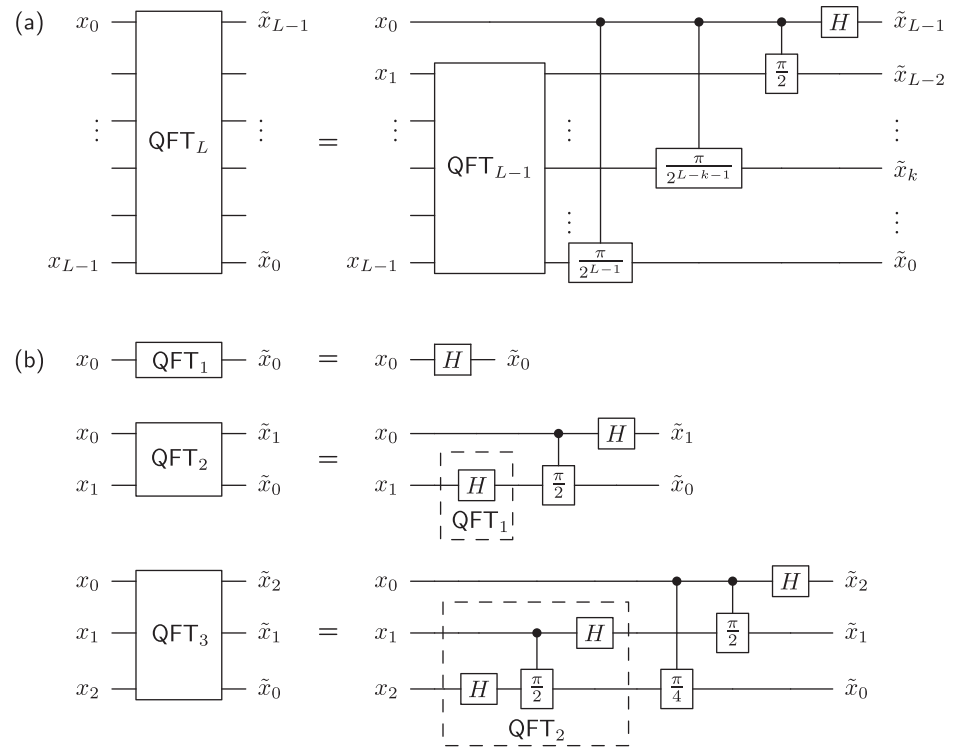

Fig. 10. (a) Recursive definition of the quantum Fourier transform (QFT) for L qubits in terms of the QFT for $L-1$ qubits. See Ref. 8 or 33 for a derivation. (b) QFT's for $L=1$, 2, and 3 from the recursive definition.

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)…

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

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.

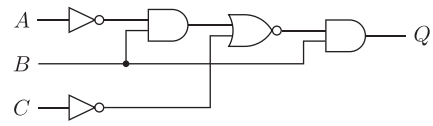

Fig. 2. A classical logic circuit. The open shapes are logic gates (here two NOTs, two ANDs and a NOR) and the lines are wires carrying logic signals from one gate to another. This circuit takes Boolean inputs A, B, and C and produces Boolean output Q.