POLYNOMIAL-TIME ALGORITHMS FOR PRIME FACTORIZATION AND DISCRETE LOGARITHMS ON A QUANTUM COMPUTER*

PETER W. SHOR†

Abstract. A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems which are generally thought to be hard on a classical computer and which have been used as the basis of several proposed cryptosystems. Efficient randomized algorithms are given for these two problems on a hypothetical quantum computer. These algorithms take a number of steps polynomial in the input size, e.g., the number of digits of the integer to be factored.

Key words. algorithmic number theory, prime factorization, discrete logarithms, Church's thesis, quantum computers, foundations of quantum mechanics, spin systems, Fourier transforms

AMS subject classifications. 81P10, 11Y05, 68Q10, 03D10

1. Introduction. One of the first results in the mathematics of computation, which underlies the subsequent development of much of theoretical computer science, was the distinction between computable and non-computable functions shown in papers of Church [1936], Post [1936], and Turing [1936]. The observation that several apparently different definitions of what it meant for a function to be computable yielded the same set of computable functions led to the proposal of Church's thesis, which says that all computing devices can be simulated by a Turing machine. This thesis greatly simplifies the study of computation, since it reduces the potential field of study from any of an infinite number of potential computing devices to Turing machines. Church's thesis is not a mathematical theorem; to make it one would require a precise mathematical description of a computing device. Such a description, however, would leave open the possibility of some practical computing device which did not satisfy this precise mathematical description, and thus would make the resulting theorem weaker than Church's original thesis.

With the development of practical computers, it became apparent that the distinction between computable and non-computable functions was much too coarse; computer scientists are now interested in the exact efficiency with which specific functions can be computed. This exact efficiency, on the other hand, was found to be too precise a quantity to work with easily. The generally accepted compromise between coarseness and precision distinguishes efficiently from inefficiently computable functions by whether the length of the computation scales polynomially or superpolynomially with the input size. The class of problems which can be solved by algorithms having a number of steps polynomial in the input size is known as P.

For this classification to make sense, it must be machine-independent. That is, the question of whether a function is computable in polynomial time must be independent of the type of computing device used. This corresponds to the following quantitative version of Church's thesis, which has been called the "Strong Church's Thesis" by

*This paper is a substantially expanded version of a paper that appeared in the Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, Nov. 20–22, 1994, IEEE Computer Society Press, Los Alamitos, CA, pp. 124–134. It will appear in *SIAM Journal on Computing,* 1997. Copyright by the Society for Industrial and Applied Mathematics.

†AT&T Labs – Research, Room 2D-149, 600 Mountain Ave., Murray Hill, NJ 07974.

Vergis, Steiglitz, and Dickinson [1986] and which makes up half of the "Invariance Thesis" of van Emde Boas [1990].

THESIS 1.1 (Quantitative Church's Thesis). *Any physical computing device can be simulated by a Turing machine in a number of steps polynomial in the resources used by the computing device.*

Readers who are not comfortable with Turing machines may think instead of digital computers having an amount of memory that grows linearly with the length of the computation, as these two classes of computing machines can efficiently simulate each other. In statements of this thesis, the Turing machine is sometimes augmented with a random number generator, as it has not yet been determined whether there are pseudorandom number generators which can efficiently simulate truly random number generators for all purposes.

There are two escape clauses in the above thesis. One of these is the word "physical." Researchers have produced machine models that violate the above quantitative Church's thesis, but most of these have been ruled out by some reason for why they are not physical, that is, why they could not be built and made to work. The other escape clause in the above thesis is the word "resources," the meaning of which is not completely specified above. There are generally two resources which limit the ability of digital computers to solve large problems: time (computational steps) and space (memory). There are more resources pertinent to analog computation; some proposed analog machines that seem able to solve NP-complete problems in polynomial time have required exponentially precise parts or an exponential amount of energy. (See Vergis et al. [1986] and Steiglitz [1988]; this issue is also implicit in the papers of Canny and Reif [1987] and Choi et al. [1995] on three-dimensional shortest paths.)

For quantum computation, in addition to space and time there is also a third potentially important resource: precision. For a quantum computer to work, at least in any currently envisioned implementation, it must be able to make changes in the quantum states of objects (e.g., atoms, photons, or nuclear spins). These changes can clearly not be perfectly accurate, but must contain some small amount of inherent imprecision. If this imprecision is constant (i.e., it does not depend on the size of the input), then it is not known how to compute any functions in polynomial time on a quantum computer that cannot also be computed in polynomial time on a classical computer with a random number generator. However, if we let the precision grow polynomially in the input size (so the number of *bits* of precision grows logarithmically in the input size), we appear to obtain a more powerful type of computer. Allowing the same polynomial growth in precision does not appear to confer extra computing power to classical mechanics, although allowing exponential growth in precision may [Hartmanis and Simon 1974, Vergis et al. 1986].

As far as we know, what precision is possible in quantum state manipulation is dictated not by fundamental physical laws but by the properties of the materials from which and the architecture with which a quantum computer is built. It is currently not clear which architectures, if any, will give high precision, and what this precision will be. If the precision of a quantum computer is large enough to make it more powerful than a classical computer, then in order to understand its potential it is important to think of precision as a resource that can vary. Treating the precision as a large constant (even though it is almost certain to be constant for any given machine) would be comparable to treating a classical digital computer as a finite automaton — since any given computer has a fixed amount of memory, this view is technically correct; however, it is not particularly useful.

Because of the remarkable effectiveness of our mathematical models of computation, computer scientists have tended to forget that computation is dependent on the laws of physics. This can be seen in the statement of the quantitative Church's thesis in van Emde Boas [1990], where the word "physical" in the above phrasing is replaced by the word "reasonable." It is difficult to imagine any definition of "reasonable" in this context which does not mean "physically realizable," i.e., that this computing machine could actually be built and would work.

Computer scientists have become convinced of the truth of the quantitative Church's thesis through the failure of all proposed counter-examples. Most of these proposed counter-examples have been based on the laws of classical mechanics; however, the universe is in reality quantum mechanical. Quantum mechanical objects often behave quite differently from how our intuition, based on classical mechanics, tells us they should. It thus seems plausible that the natural computing power of classical mechanics corresponds to that of Turing machines,[1] while the natural computing power of quantum mechanics might be greater.

The first person to look at the interaction between computation and quantum mechanics appears to have been Benioff [1980, 1982a, 1982b]. Although he did not ask whether quantum mechanics conferred extra power to computation, he showed that reversible unitary evolution was sufficient to realize the computational power of a Turing machine, thus showing that quantum mechanics is computationally at least as powerful as classical computers. This work was fundamental in making later investigation of quantum computers possible.

Feynman [1982,1986] seems to have been the first to suggest that quantum mechanics might be computationally more powerful than Turing machines. He gave arguments as to why quantum mechanics is intrinsically computationally expensive to simulate on a classical computer. He also raised the possibility of using a computer based on quantum mechanical principles to avoid this problem, thus implicitly asking the converse question: by using quantum mechanics in a computer can you compute more efficiently than on a classical computer? The first to ask this question explicitly was Deutsch [1985, 1989]. In order to study this question, he defined both quantum Turing machines and quantum circuits and investigated some of their properties.

The question of whether using quantum mechanics in a computer allows one to obtain more computational power was more recently addressed by Deutsch and Jozsa [1992] and Berthiaume and Brassard [1992, 1994]. These papers showed that there are problems which quantum computers can quickly solve exactly, but that classical computers can only solve quickly with high probability and the aid of a random number generator. However, these papers did not show how to solve any problem in quantum polynomial time that was not already known to be solvable in polynomial time with the aid of a random number generator, allowing a small probability of error; this is the characterization of the complexity class BPP, which is widely viewed as the class of efficiently solvable problems.

Further work on this problem was stimulated by Bernstein and Vazirani [1993]. One of the results contained in their paper was an oracle problem (that is, a problem involving a "black box" subroutine that the computer is allowed to perform but for which no code is accessible) which can be done in polynomial time on a quantum

[1] See Vergis et al. [1986], Steiglitz [1988], and Rubel [1989]. I believe that this question has not yet been settled and is worthy of further investigation. In particular, turbulence seems a good candidate for a counterexample to the quantitative Church's thesis because the non-trivial dynamics on many length scales appear to make it difficult to simulate on a classical computer.

Turing machine but which requires super-polynomial time on a classical computer. This result was improved by Simon [1994], who gave a much simpler construction of an oracle problem which takes polynomial time on a quantum computer but requires *exponential* time on a classical computer. Indeed, while Bernstein and Vaziarni's problem appears contrived, Simon's problem looks quite natural. Simon's algorithm inspired the work presented in this paper.

Two number theory problems which have been studied extensively but for which no polynomial-time algorithms have yet been discovered are finding discrete logarithms and factoring integers [Pomerance 1987, Gordon 1993, Lenstra and Lenstra 1993, Adleman and McCurley 1994]. These problems are so widely believed to be hard that several cryptosystems based on their difficulty have been proposed, including the widely used RSA public key cryptosystem developed by Rivest, Shamir, and Adleman [1978]. We show that these problems can be solved in polynomial time on a quantum computer with a small probability of error.

Currently, nobody knows how to build a quantum computer, although it seems as though it might be possible within the laws of quantum mechanics. Some suggestions have been made as to possible designs for such computers [Teich et al. 1988, Lloyd 1993, 1994, Cirac and Zoller 1995, DiVincenzo 1995, Sleator and Weinfurter 1995, Barenco et al. 1995b, Chuang and Yamomoto 1995], but there will be substantial difficulty in building any of these [Landauer 1995, 1996, Unruh 1995, Chuang et al. 1995, Palma et al. 1996]. The most difficult obstacles appear to involve the decoherence of quantum superpositions through the interaction of the computer with the environment, and the implementation of quantum state transformations with enough precision to give accurate results after many computation steps. Both of these obstacles become more difficult as the size of the computer grows, so it may turn out to be possible to build small quantum computers, while scaling up to machines large enough to do interesting computations may present fundamental difficulties.

Even if no useful quantum computer is ever built, this research does illuminate the problem of simulating quantum mechanics on a classical computer. Any method of doing this for an arbitrary Hamiltonian would necessarily be able to simulate a quantum computer. Thus, any general method for simulating quantum mechanics with at most a polynomial slowdown would lead to a polynomial-time algorithm for factoring.

The rest of this paper is organized as follows. In §2, we introduce the model of quantum computation, the *quantum gate array,* that we use in the rest of the paper. In §3 and §4, we explain two subroutines that are used in our algorithms: reversible modular exponentiation in §3 and quantum Fourier transforms in §4. In §5, we give our algorithm for prime factorization, and in §6, we give our algorithm for extracting discrete logarithms. In §7, we give a brief discussion of the practicality of quantum computation and suggest possible directions for further work.

2. Quantum computation. In this section we give a brief introduction to quantum computation, emphasizing the properties that we will use. We will describe only *quantum gate arrays,* or *quantum acyclic circuits,* which are analogous to acyclic circuits in classical computer science. This model was originally studied by Yao [1993] and is closely related to the quantum computational networks discussed by Deutsch [1989]. For other models of quantum computers, see references on quantum Turing machines [Deutsch 1985, Bernstein and Vazirani 1993, Yao 1993] and quantum cellular automata [Feynman 1986, Margolus 1986, 1990, Lloyd 1993, Biafore 1994]. If they are allowed a small probability of error, quantum Turing machines and quantum gate arrays can compute the same functions in polynomial time [Yao 1993]. This may also be true for the various models of quantum cellular automata, but it has not yet been proved. This gives evidence that the class of functions computable in quantum polynomial time with a small probability of error is robust, in that it does not depend on the exact architecture of a quantum computer. By analogy with the classical class BPP, this class is called BQP.

Consider a system with n components, each of which can have two states. Whereas in classical physics, a complete description of the state of this system requires only n bits, in quantum physics, a complete description of the state of this system requires $2^n - 1$ complex numbers. To be more precise, the state of the quantum system is a point in a 2^n-dimensional vector space. For each of the 2^n possible classical positions of the components, there is a basis state of this vector space which we represent, for example, by $|011\cdots0\rangle$ meaning that the first bit is 0, the second bit is 1, and so on. Here, the *ket* notation $|x\rangle$ means that x is a (pure) quantum state. (Mixed states will not be discussed in this paper, and thus we do not define them; see a quantum theory book such as Peres [1993] for their definition.) The *Hilbert space* associated with this quantum system is the complex vector space with these 2^n states as basis vectors, and the state of the system at any time is represented by a unit-length vector in this Hilbert space. As multiplying this state vector by a unit-length complex phase does not change any behavior of the state, we need only $2^n - 1$ complex numbers to completely describe the state. We represent this superposition of states by

$$\sum_{i=0}^{2^n-1} a_i |S_i\rangle, \tag{2.1}$$

where the amplitudes a_i are complex numbers such that $\sum_i |a_i|^2 = 1$ and each $|S_i\rangle$ is a basis vector of the Hilbert space. If the machine is measured (with respect to this basis) at any particular step, the probability of seeing basis state $|S_i\rangle$ is $|a_i|^2$; however, measuring the state of the machine projects this state to the observed basis vector $|S_i\rangle$. Thus, looking at the machine during the computation will invalidate the rest of the computation. General quantum mechanical meaurements, i.e., POVM's [see Peres 1993], can be considerably more complicated than the case of projection onto the canonical basis to which we restrict ourselves in this paper. This does not greatly restrict our model of computation, since measurements in other reasonable bases, as well as other local measurements, can be simulated by first using quantum computation to perform a change of basis and then performing a measurement in the canonical basis.

In order to use a physical system for computation, we must be able to change the state of the system. The laws of quantum mechanics permit only unitary transformations of state vectors. A unitary matrix is one whose conjugate transpose is equal to its inverse, and requiring state transformations to be represented by unitary matrices ensures that summing the probability over all possible outcomes yields 1. The definition of quantum circuits (and quantum Turing machines) only allows *local* unitary transformations; that is, unitary transformations on a fixed number of bits. This is physically justified because, given a general unitary transformation on n bits, it is not at all clear how one could efficiently implement it physically, whereas two-bit transformations can at least in theory be implemented by relatively simple physical systems [Cirac and Zoller 1995, DiVincenzo 1995, Sleator and Weinfurter 1995, Chuang and Yamomoto 1995]. While general n-bit transformations can always be built out of two-bit transformations [DiVincenzo 1995, Sleator and Weinfurter 1995, Lloyd 1995, Deutsch et al. 1995], the number required will often be exponential in n [Barenco et al. 1995a]. Thus, the set of two-bit transformations form a set of building blocks for quantum circuits in a manner analogous to the way a universal set of classical gates (such as the AND, OR and NOT gates) form a set of building blocks for classical circuits. In fact, for a universal set of quantum gates, it is sufficient to take all one-bit gates and a single type of two-bit gate, the controlled NOT gate (also called the XOR or parity gate), which negates the second bit if and only if the first bit is 1.

Perhaps an example will be informative at this point. A quantum gate can be expressed as a truth table: for each input basis vector we need to give the output of the gate. One such gate is:

$$\begin{aligned} |00\rangle &\to |00\rangle \\ |01\rangle &\to |01\rangle \\ |10\rangle &\to \tfrac{1}{\sqrt{2}}(|10\rangle + |11\rangle) \\ |11\rangle &\to \tfrac{1}{\sqrt{2}}(|10\rangle - |11\rangle). \end{aligned} \tag{2.2}$$

Not all truth tables correspond to physically feasible quantum gates, as many truth tables will not give rise to unitary transformations.

The same gate can also be represented as a matrix. The rows correspond to input basis vectors. The columns correspond to output basis vectors. The (i, j) entry gives, when the ith basis vector is input to the gate, the coefficient of the jth basis vector in the corresponding output of the gate. The truth table above would then correspond to the following matrix:

$$\begin{array}{c|cccc|} & |00\rangle & |01\rangle & |10\rangle & |11\rangle \\ |00\rangle & 1 & 0 & 0 & 0 \\ |01\rangle & 0 & 1 & 0 & 0 \\ |10\rangle & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ |11\rangle & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{array}\,. \tag{2.3}$$

A quantum gate is feasible if and only if the corresponding matrix is unitary, i.e., its inverse is its conjugate transpose.

Suppose our machine is in the superposition of states

$$\tfrac{1}{\sqrt{2}}|10\rangle - \tfrac{1}{\sqrt{2}}|11\rangle \tag{2.4}$$

and we apply the unitary transformation represented by (2.2) and (2.3) to this state. The resulting output will be the result of multiplying the vector (2.4) by the matrix (2.3). The machine will thus go to the superposition of states

$$\tfrac{1}{2}(|10\rangle + |11\rangle) - \tfrac{1}{2}(|10\rangle - |11\rangle) \;=\; |11\rangle. \tag{2.5}$$

This example shows the potential effects of interference on quantum computation. Had we started with either the state $|10\rangle$ or the state $|11\rangle$, there would have been a chance of observing the state $|10\rangle$ after the application of the gate (2.3). However, when we start with a superposition of these two states, the probability amplitudes for the state $|10\rangle$ cancel, and we have no possibility of observing $|10\rangle$ after the application of the gate. Notice that the output of the gate would have been $|10\rangle$ instead of $|11\rangle$ had we started with the superposition of states

$$\frac{1}{\sqrt{2}}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle \tag{2.6}$$

which has the same probabilities of being in any particular configuration if it is observed as does the superposition (2.4).

If we apply a gate to only two bits of a longer vector (now our circuit must have more than two wires), for each basis vector we apply the transformation given by the gate's truth table to the two bits on which the gate is operating, and leave the other bits alone. This corresponds to multiplying the whole state by the tensor product of the gate matrix on those two bits with the identity matrix on the remaining bits. For example, applying the transformation represented by (2.2) and (2.3) to the first two bits of the basis vector $|110\rangle$ yields the vector $\frac{1}{\sqrt{2}}(|100\rangle - |110\rangle)$.

A quantum gate array is a set of quantum gates with logical "wires" connecting their inputs and outputs. The input to the gate array, possibly along with extra work bits that are initially set to 0, is fed through a sequence of quantum gates. The values of the bits are observed after the last quantum gate, and these values are the output. This model is analogous to classical acyclic circuits in theoretical computer science, and was previously studied by Yao [1993]. As in the classical case, in order to compare quantum gate arrays with quantum Turing machines, we need to make the gate arrays a *uniform* complexity class. In other words, because we use a different gate array for each size of input, we need to keep the designer of the gate arrays from hiding non-computable (or hard to compute) information in the arrangement of the gates. To make quantum gate arrays uniform, we must add two requirements to the definition of gate arrays. The first is the standard uniformity requirement that the design of the gate array be produced by a polynomial-time (classical) computation. The second uniformity requirement should be a standard part of the definition of analog complexity classes; however, since analog complexity classes have not been as widely studied, this requirement is not well known. The requirement is that the entries in the unitary matrices describing the gates must be computable numbers. Specifically, the first $\log n$ bits of each entry should be classically computable in time polynomial in n [Solovay 1995]. This keeps non-computable (or hard to compute) information from being hidden in the bits of the amplitudes of the quantum gates.

3. Reversible logic and modular exponentiation. The definition of quantum gate arrays gives rise to completely reversible computation. That is, knowing the quantum state on the wires leading out of a gate tells uniquely what the quantum state must have been on the wires leading into that gate. This is a reflection of the fact that, despite the macroscopic arrow of time, the laws of physics appear to be completely reversible. This would seem to imply that anything built with the laws of physics must be completely reversible; however, classical computers get around this by dissipating energy and thus making their computations thermodynamically irreversible. This appears impossible to do for quantum computers because superpositions of quantum states need to be maintained throughout the computation. Thus, quantum computers necessarily have to use reversible computation. This imposes extra costs when doing classical computations on a quantum computer, which can be necessary in subroutines of quantum computations.

Because of the reversibility of quantum computation, a deterministic computation is performable on a quantum computer only if it is reversible. Luckily, it has already been shown that any deterministic computation can be made reversible [Lecerf 1963,

TABLE 3.1
Truth tables for Toffoli and Fredkin gates.

Toffoli Gate

INPUT			OUTPUT		
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Fredkin Gate

INPUT			OUTPUT		
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1

Bennett 1973]. In fact, reversible classical gate arrays (or reversible acyclic circuits) have been studied. Much like the result that any classical computation can be done using NAND gates, there are also universal gates for reversible computation. Two of these are Toffoli gates [Toffoli 1980] and Fredkin gates [Fredkin and Toffoli 1982]; these are illustrated in Table 3.1.

The Toffoli gate is just a controlled controlled NOT, i.e., the last bit is negated if and only if the first two bits are 1. In a Toffoli gate, if the third input bit is set to 1, then the third output bit is the NAND of the first two input bits. Since NAND is a universal gate for classical gate arrays, this shows that the Toffoli gate is universal. In a Fredkin gate, the last two bits are swapped if the first bit is 0, and left untouched if the first bit is 1. For a Fredkin gate, if the third input bit is set to 0, the second output bit is the AND of the first two input bits; and if the last two input bits are set to 0 and 1 respectively, the second output bit is the NOT of the first input bit. Thus, both AND and NOT gates are realizable using Fredkin gates, showing that the Fredkin gate is universal.

From results on reversible computation [Lecerf 1963, Bennett 1973], we can efficiently compute any polynomial time function $F(x)$ as long as we keep the input x in the computer. We do this by adapting the method for computing the function F non-reversibly. These results can easily be extended to work for gate arrays [Toffoli 1980, Fredkin and Toffoli 1982]. When AND, OR or NOT gates are changed to Fredkin or Toffoli gates, one obtains both additional input bits, which must be preset to specified values, and additional output bits, which contain the information needed to reverse the computation. While the additional input bits do not present difficulties in designing quantum computers, the additional output bits do, because unless they are all reset to 0, they will affect the interference patterns in quantum computation. Bennett's method for resetting these bits to 0 is shown in the top half of Table 3.2. A non-reversible gate array may thus be turned into a reversible gate array as follows. First, duplicate the input bits as many times as necessary (since each input bit could be used more than once by the gate array). Next, keeping one copy of the input around, use Toffoli and Fredkin gates to simulate non-reversible gates, putting the extra output bits into the RECORD register. These extra output bits preserve enough of a record of the operations to enable the computation of the gate array to be reversed. Once the output $F(x)$ has been computed, copy it into a register that has been preset to zero, and then undo the computation to erase both the first OUTPUT register and the RECORD register.

To erase x and replace it with $F(x)$, in addition to a polynomial-time algorithm for F, we also need a polynomial-time algorithm for computing x from $F(x)$; i.e., we

TABLE 3.2
Bennett's method for making a computation reversible.

INPUT	- - - - - -	- - - - - -	- - - - - -
INPUT	OUTPUT	RECORD(F)	- - - - - -
INPUT	OUTPUT	RECORD(F)	OUTPUT
INPUT	- - - - - -	- - - - - -	OUTPUT
INPUT	INPUT	RECORD(F^{-1})	OUTPUT
- - - - - -	INPUT	RECORD(F^{-1})	OUTPUT
- - - - - -	- - - - - -	- - - - - -	OUTPUT

need that F is one-to-one and that both F and F^{-1} are polynomial-time computable. The method for this computation is given in the whole of Table 3.2. There are two stages to this computation. The first is the same as before, taking x to $(x, F(x))$. For the second stage, shown in the bottom half of Table 3.2, note that if we have a method to compute F^{-1} non-reversibly in polynomial time, we can use the same technique to reversibly map $F(x)$ to $(F(x), F^{-1}(F(x))) = (F(x), x)$. However, since this is a reversible computation, we can reverse it to go from $(x, F(x))$ to $F(x)$. Put together, these two stages take x to $F(x)$.

The above discussion shows that computations can be made reversible for only a constant factor cost in time, but the above method uses as much space as it does time. If the classical computation requires much less space than time, then making it reversible in this manner will result in a large increase in the space required. There are methods that do not use as much space, but use more time, to make computations reversible [Bennett 1989, Levine and Sherman 1990]. While there is no general method that does not cause an increase in either space or time, specific algorithms can sometimes be made reversible without paying a large penalty in either space or time; at the end of this section we will show how to do this for modular exponentiation, which is a subroutine necessary for quantum factoring.

The bottleneck in the quantum factoring algorithm; i.e., the piece of the factoring algorithm that consumes the most time and space, is modular exponentiation. The modular exponentiation problem is, given n, x, and r, find $x^r \pmod n$. The best classical method for doing this is to repeatedly square of $x \pmod n$ to get $x^{2^i} \pmod n$ for $i \le \log_2 r$, and then multiply a subset of these powers $\pmod n$ to get $x^r \pmod n$. If we are working with l-bit numbers, this requires $O(l)$ squarings and multiplications of l-bit numbers $\pmod n$. Asymptotically, the best classical result for gate arrays for multiplication is the Schönhage–Strassen algorithm [Schönhage and Strassen 1971, Knuth 1981, Schönhage 1982]. This gives a gate array for integer multiplication that uses $O(l \log l \log\log l)$ gates to multiply two l-bit numbers. Thus, asymptotically, modular exponentiation requires $O(l^2 \log l \log\log l)$ time. Making this reversible would naïvely cost the same amount in space; however, one can reuse the space used in the repeated squaring part of the algorithm, and thus reduce the amount of space needed to essentially that required for multiplying two l-bit numbers; one simple method for reducing this space (although not the most versatile one) will be given later in this section. Thus, modular exponentiation can be done in $O(l^2 \log l \log\log l)$ time and $O(l \log l \log\log l)$ space.

While the Schönhage–Strassen algorithm is the best multiplication algorithm discovered to date for large l, it does not scale well for small l. For small numbers, the best gate arrays for multiplication essentially use elementary-school longhand multiplication in binary. This method requires $O(l^2)$ time to multiply two l-bit numbers, and thus modular exponentiation requires $O(l^3)$ time with this method. These gate arrays can be made reversible, however, using only $O(l)$ space.

We now give the method for constructing a reversible gate array that takes only $O(l)$ space and $O(l^3)$ time to compute $(a, x^a \pmod n)$ from a, where a, x, and n are l-bit numbers and x and n are relatively prime. This case, where x and n are relatively prime, is sufficient for our factoring and discrete logarithm algorithms. A detailed analysis of essentially this method, giving an exact number of quantum gates sufficient for factoring, was performed by Beckman et al. [1996].

The basic building block used is a gate array that takes b as input and outputs $b + c \pmod n$. Note that here b is the gate array's input but c and n are built into the structure of the gate array. Since addition $\pmod n$ is computable in $O(\log n)$ time classically, this reversible gate array can be made with only $O(\log n)$ gates and $O(\log n)$ work bits using the techniques explained earlier in this section.

The technique we use for computing $x^a \pmod n$ is essentially the same as the classical method. First, by repeated squaring we compute $x^{2^i} \pmod n$ for all $i < l$. Then, to obtain $x^a \pmod n$ we multiply the powers $x^{2^i} \pmod n$ where 2^i appears in the binary expansion of a. In our algorithm for factoring n, we only need to compute $x^a \pmod n$ where a is in a superposition of states, but x is some fixed integer. This makes things much easier, because we can use a reversible gate array where a is input, but where x and n are built into the structure of the gate array. Thus, we can use the algorithm described by the following pseudocode; here a_i represents the ith bit of a in binary, where the bits are indexed from right to left and the rightmost bit of a is a_0.

```
power := 1
for i = 0 to l−1
    if ( a_i == 1 ) then
        power := power * x^(2^i)   (mod n)
    endif
endfor
```

The variable a is left unchanged by the code and $x^a \pmod n$ is output as the variable *power*. Thus, this code takes the pair of values $(a, 1)$ to $(a, x^a \pmod n)$.

This pseudocode can easily be turned into a gate array; the only hard part of this is the fourth line, where we multiply the variable *power* by $x^{2^i} \pmod n$; to do this we need to use a fairly complicated gate array as a subroutine. Recall that $x^{2^i} \pmod n$ can be computed classically and then built into the structure of the gate array. Thus, to implement this line, we need a reversible gate array that takes b as input and gives $bc \pmod n$ as output, where the structure of the gate array can depend on c and n. Of course, this step can only be reversible if $\gcd(c, n) = 1$, i.e., if c and n have no common factors, as otherwise two distinct values of b will be mapped to the same value of $bc \pmod n$. Fortunately, x and n being relatively prime in modular exponentiation implies that c and n are relatively prime in this subroutine.

We will show how to build this gate array in two stages. The first stage is directly analogous to exponentiation by repeated multiplication; we obtain multiplication from repeated addition $\pmod n$. Pseudocode for this stage is as follows.

```
result := 0
for i = 0 to l−1
    if ( b_i == 1 ) then
        result := result + 2^i c   (mod n)
    endif
endfor
```

Again, $2^i c \pmod n$ can be precomputed and built into the structure of the gate array.

The above pseudocode takes b as input, and gives $(b, bc \pmod n)$ as output. To get the desired result, we now need to erase b. Recall that $\gcd(c, n) = 1$, so there is a $c^{-1} \pmod n$ with $c\,c^{-1} \equiv 1 \pmod n$. Multiplication by this c^{-1} could be used to reversibly take $bc \pmod n$ to $(bc \pmod n), bcc^{-1} \pmod n)) = (bc \pmod n), b)$. This is just the reverse of the operation we want, and since we are working with reversible computing, we can turn this operation around to erase b. The pseudocode for this follows.

```
for i = 0 to l−1
    if ( result_i == 1 ) then
        b := b − 2^i c^{-1}   (mod n)
    endif
endfor
```

As before, $result_i$ is the ith bit of $result$.

Note that at this stage of the computation, b should be 0. However, we did not set b directly to zero, as this would not have been a reversible operation and thus impossible on a quantum computer, but instead we did a relatively complicated sequence of operations which ended with $b = 0$ and which in fact depended on multiplication being a group $\pmod n$. At this point, then, we could do something somewhat sneaky: we could measure b to see if it actually is 0. If it is not, we know that there has been an error somewhere in the quantum computation, i.e., that the results are worthless and we should stop the computer and start over again. However, if we do find that b is 0, then we know (because we just observed it) that it is now exactly 0. This measurement thus may bring the quantum computation back on track in that any amplitude that b had for being non-zero has been eliminated. Further, because the probability that we observe a state is proportional to the square of the amplitude of that state, doing the modular exponentiation and measuring b every time that we know it should be 0 may result in a higher probability of overall success than the same computation done without the repeated measurements of b. This is the *quantum watchdog* (or *quantum Zeno*) effect [Peres 1993], and whether it is applicable in this setting depends on the error model for our quantum circuits. The argument above does not actually show that repeated measurement of b is indeed beneficial, because there is a cost (in time, if nothing else) of measuring b. Before this is implemented, then, it should be checked with analysis or experiment that the benefit of such measurements exceeds their cost. In general, I believe that partial measurements such as this one are a promising way of trying to stabilize quantum computations.

Currently, Schönhage–Strassen is the algorithm of choice for multiplying very large numbers, and longhand multiplication is the algorithm of choice for small numbers. There are also multiplication algorithms which have asymptotic efficiencies between these two algorithms, and which are superior for intermediate length numbers [Karatsuba and Ofman 1962, Knuth 1981, Schönhage et al. 1994]. It is not clear which algorithms are best for which size numbers. While this is known to some extent for classical computation [Schönhage et al. 1994], using data on which algorithms work better on classical computers could be misleading for two reasons: First, classical computers need not be reversible, and the cost of making an algorithm reversible depends on the algorithm. Second, existing computers generally have multiplication for 32- or 64-bit numbers built into their hardware, and this tends to increase the optimal changeover points. To further confuse matters, some multiplication algorithms can take better advantage of hardwired multiplication than others. In order to program quantum computers most efficiently, work thus needs to be done on the best way of implementing elementary arithmetic operations on quantum computers. One tantalizing fact is that the Schönhage–Strassen fast multiplication algorithm uses the fast Fourier transform, which is also the basis for all the fast algorithms on quantum computers discovered to date. It is thus tempting to speculate that integer multiplication itself might be speeded up by a quantum algorithm; if possible, this would result in a somewhat faster asymptotic bound for factoring on a quantum computer, and indeed could even make breaking RSA on a quantum computer asymptotically faster than encrypting with RSA on a classical computer.

4. Quantum Fourier transforms. Since quantum computation deals with unitary transformations, it is helpful to know how to build certain useful unitary transformations. In this section we give a technique for constructing in polynomial time on quantum computers one particular unitary transformation, which is essentially a discrete Fourier transform. This transformation will be given as a matrix, with both rows and columns indexed by states. These states correspond to binary representations of integers on the computer; in particular, the rows and columns will be indexed beginning with 0 unless otherwise specified.

This transformation is as follows. Consider a number a with $0 \le a < q$ for some q. We will perform the transformation that takes the state $|a\rangle$ to the state

$$\frac{1}{q^{1/2}} \sum_{c=0}^{q-1} |c\rangle \exp(2\pi i a c / q). \tag{4.1}$$

That is, we apply the unitary matrix whose (a, c) entry is $\frac{1}{q^{1/2}} \exp(2\pi i a c/q)$. This Fourier transform is at the heart of our algorithms, and we call this matrix A_q.

In our factoring and discrete logarithm algorithms, we will use A_q for q of exponential size (i.e., the number of bits of q grows polynomially with the length of our input). We must thus show how this transformation can be done in time polynomial in the number of bits of q. In this paper, we give a simple construction for A_q when q is a power of 2 that was discovered independently by Coppersmith [1994] and Deutsch [see Ekert and Jozsa 1996]. This construction is essentially the standard fast Fourier transform (FFT) algorithm [Knuth 1981] adapted for a quantum computer; the following description of it follows that of Ekert and Jozsa [1996]. In the earlier version of this paper [Shor 1994], we gave a construction for A_q when q was in the special class of smooth numbers having only small prime power factors. In fact, Cleve [1994] has shown how to construct A_q for all smooth numbers q whose prime factors are at most $O(\log n)$.

Take $q = 2^l$, and let us represent an integer a in binary as $|a_{l-1} a_{l-2} \ldots a_0\rangle$. For the quantum Fourier transform A_q, we only need to use two types of quantum gates.

These gates are R_j, which operates on the jth bit of the quantum computer:

$$R_j \;=\; \begin{array}{c|cc|} & |0\rangle & |1\rangle \\ |0\rangle & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ |1\rangle & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{array}, \tag{4.2}$$

and $S_{j,k}$, which operates on the bits in positions j and k with $j < k$:

$$S_{j,k} \;=\; \begin{array}{c|cccc|} & |00\rangle & |01\rangle & |10\rangle & |11\rangle \\ |00\rangle & 1 & 0 & 0 & 0 \\ |01\rangle & 0 & 1 & 0 & 0 \\ |10\rangle & 0 & 0 & 1 & 0 \\ |11\rangle & 0 & 0 & 0 & e^{i\theta_{k-j}} \end{array}, \tag{4.3}$$

where $\theta_{k-j} = \pi/2^{k-j}$. To perform a quantum Fourier transform, we apply the matrices in the order (from left to right)

$$R_{l-1}\, S_{l-2,l-1}\, R_{l-2}\, S_{l-3,l-1}\, S_{l-3,l-2}\, R_{l-3} \ldots R_1\, S_{0,l-1}\, S_{0,l-2} \ldots S_{0,2}\, S_{0,1}\, R_0\,; \tag{4.4}$$

that is, we apply the gates R_j in reverse order from R_{l-1} to R_0, and between R_{j+1} and R_j we apply all the gates $S_{j,k}$ where $k > j$. For example, on 3 bits, the matrices would be applied in the order $R_2 S_{1,2} R_1 S_{0,2} S_{0,1} R_0$. To take the Fourier transform A_q when $q = 2^l$, we thus need to use $l(l-1)/2$ quantum gates.

Applying this sequence of transformations will result in a quantum state $\frac{1}{q^{1/2}} \sum_b \exp(2\pi iac/q)|b\rangle$, where b is the bit-reversal of c, i.e., the binary number obtained by reading the bits of c from right to left. Thus, to obtain the actual quantum Fourier transform, we need either to do further computation to reverse the bits of $|b\rangle$ to obtain $|c\rangle$, or to leave these bits in place and read them in reverse order; either alternative is easy to implement.

To show that this operation actually performs a quantum Fourier transform, consider the amplitude of going from $|a\rangle = |a_{l-1} \ldots a_0\rangle$ to $|b\rangle = |b_{l-1} \ldots b_0\rangle$. First, the factors of $1/\sqrt{2}$ in the R matrices multiply to produce a factor of $1/q^{1/2}$ overall; thus we need only worry about the $\exp(2\pi iac/q)$ phase factor in the expression (4.1). The matrices $S_{j,k}$ do not change the values of any bits, but merely change their phases. There is thus only one way to switch the jth bit from a_j to b_j, and that is to use the appropriate entry in the matrix R_j. This entry adds π to the phase if the bits a_j and b_j are both 1, and leaves it unchanged otherwise. Further, the matrix $S_{j,k}$ adds $\pi/2^{k-j}$ to the phase if a_j and b_k are both 1 and leaves it unchanged otherwise. Thus, the phase on the path from $|a\rangle$ to $|b\rangle$ is

$$\sum_{0\le j<l} \pi a_j b_j + \sum_{0\le j<k<l} \frac{\pi}{2^{k-j}} a_j b_k. \tag{4.5}$$

This expression can be rewritten as

$$\sum_{0\le j\le k<l} \frac{\pi}{2^{k-j}} a_j b_k. \tag{4.6}$$

Since c is the bit-reversal of b, this expression can be further rewritten as

$$\sum_{0\le j\le k<l} \frac{\pi}{2^{k-j}} a_j c_{l-1-k}. \tag{4.7}$$

Making the substitution $l-k-1$ for k in this sum, we obtain

$$\sum_{0 \le j+k < l} 2\pi \frac{2^j 2^k}{2^l} a_j c_k \tag{4.8}$$

Now, since adding multiples of 2π do not affect the phase, we obtain the same phase if we sum over all j and k less than l, obtaining

$$\sum_{j,k=0}^{l-1} 2\pi \frac{2^j 2^k}{2^l} a_j c_k = \frac{2\pi}{2^l} \sum_{j=0}^{l-1} 2^j a_j \sum_{k=0}^{l-1} 2^k c_k, \tag{4.9}$$

where the last equality follows from the distributive law of multiplication. Now, $q = 2^l$ and

$$a = \sum_{j=0}^{l-1} 2^j a_j, \qquad c = \sum_{k=0}^{l-1} 2^k c_k, \tag{4.10}$$

so the expression (4.9) is equal to $2\pi ac/q$, which is the phase for the amplitude $|a\rangle \to |c\rangle$ in the transformation (4.1).

When $k-j$ is large in the gate $S_{j,k}$ in (4.3), we are multiplying by a very small phase factor. This would be very difficult to do accurately physically, and thus it would be somewhat disturbing if this were necessary for quantum computation. In fact, Coppersmith [1994] has shown that one can use an approximate Fourier transform that ignores these tiny phase factors, but which approximates the Fourier transform closely enough that it can also be used for factoring. In fact, this technique reduces the number of quantum gates needed for the (approximate) Fourier transform considerably, as it leaves out most of the gates $S_{j,k}$.

Recently, Griffiths and Niu [1996] have shown that this Fourier transform can be carried out using only one-bit gates and measurements of single bits. Both of these operations are potentially easier to implement in a physical system than two-bit gates. The use of two-bit gates, however, is still required during the modular exponentiation step of the factoring and discrete logarithm algorithms.

5. Prime factorization. It has been known since before Euclid that every integer n is uniquely decomposable into a product of primes. For nearly as long, mathematicians have been interested in the question of how to factor a number into this product of primes. It was only in the 1970's, however, that researchers applied the paradigms of theoretical computer science to number theory, and looked at the asymptotic running times of factoring algorithms [Adleman 1994]. This has resulted in a great improvement in the efficiency of factoring algorithms. Currently the best factoring algorithm, both asymptotically and in practice, is the number field sieve [Lenstra et al. 1990, Lenstra and Lenstra 1993], which in order to factor an integer n takes asymptotic running time $\exp(c(\log n)^{1/3}(\log\log n)^{2/3})$ for some constant c. Since the input n is only $\log n$ bits in length, this algorithm is an exponential-time algorithm. Our quantum factoring algorithm takes asymptotically $O((\log n)^2(\log\log n)$ $(\log\log\log n))$ steps on a quantum computer, along with a polynomial (in $\log n$) amount of post-processing time on a classical computer that is used to convert the output of the quantum computer to factors of n. While this post-processing could in principle be done on a quantum computer, there is no reason not to use a classical computer for this step.

Instead of giving a quantum computer algorithm for factoring n directly, we give a quantum computer algorithm for finding the order r of an element x in the multiplicative group (mod n); that is, the least integer r such that $x^r \equiv 1 \pmod{n}$. It is known that using randomization, factorization can be reduced to finding the order of an element [Miller 1976]; we now briefly give this reduction.

To find a factor of an odd number n, given a method for computing the order r of x, choose a random $x \pmod{n}$, find its order r, and compute $\gcd(x^{r/2}-1, n)$. Here, $\gcd(a,b)$ is the greatest common divisor of a and b, i.e., the largest integer that divides both a and b. The Euclidean algorithm [Knuth 1981] can be used to compute $\gcd(a,b)$ in polynomial time. Since $(x^{r/2}-1)(x^{r/2}+1) = x^r - 1 \equiv 0 \pmod{n}$, the numbers $\gcd(x^{r/2}+1, n)$ and $\gcd(x^{r/2}-1, n)$ will be two factors of n. This procedure fails only if r is odd, in which case $r/2$ is not integral, or if $x^{r/2} \equiv -1 \pmod{n}$, in which case the procedure yields the trivial factors 1 and n. Using this criterion, it can be shown that this procedure, when applied to a random $x \pmod{n}$, yields a non-trivial factor of n with probability at least $1 - 1/2^{k-1}$, where k is the number of distinct odd prime factors of n. A brief sketch of the proof of this result follows.

Suppose that $n = \prod_{i=1}^{k} p_i^{\alpha_i}$ is the prime factorization of n. Let r_i be the order of $x \pmod{p_i^{\alpha_i}}$. Then r is the least common multiple of all the r_i. Consider the largest power of 2 dividing each r_i. The algorithm only fails if all of these powers of 2 agree: if they are all 1, then r is odd and $r/2$ does not exist; if they are all equal and larger than 1, then $x^{r/2} \equiv -1 \pmod{p_i^{\alpha_i}}$ for every i, so $x^{r/2} \equiv -1 \pmod{n}$. By the Chinese remainder theorem [Knuth 1981, Hardy and Wright 1979, Theorem 121], choosing an $x \pmod{n}$ at random is the same as choosing for each i a number $x_i \pmod{p_i^{\alpha_i}}$ at random, where $x \equiv x_i \pmod{p_i^{\alpha_i}}$. The multiplicative group $(\mathrm{mod}\ p^{\alpha})$ for any odd prime power p^α is cyclic [Knuth 1981], so for the odd prime power $p_i^{\alpha_i}$, the probability is at most 1/2 of choosing an x_i having a particular power of two as the largest divisor of its order r_i. Thus each of these powers of 2 has at most a 50% probability of agreeing with the previous ones, so all k of them agree with probability at most $1/2^{k-1}$. There is thus at least a $1-1/2^{k-1}$ probability that the x we choose is good. This argument shows the scheme will work as long as n is odd and not a prime power; finding a factor of even numbers and of prime powers can be done efficiently with classical methods.

We now describe the algorithm for finding the order of $x \pmod{n}$ on a quantum computer. This algorithm will use two quantum registers which hold integers represented in binary. There will also be some amount of workspace. This workspace gets reset to 0 after each subroutine of our algorithm, so we will not include it when we write down the state of our machine.

Given x and n, to find the order of x, i.e., the least r such that $x^r \equiv 1 \pmod{n}$, we do the following. First, we find q, the power of 2 with $n^2 \le q < 2n^2$. We will not include n, x, or q when we write down the state of our machine, because we never change these values. In a quantum gate array we need not even keep these values in memory, as they can be built into the structure of the gate array.

Next, we put the first register in the uniform superposition of states representing numbers $a \pmod{q}$. This leaves our machine in state

$$\frac{1}{q^{1/2}} \sum_{a=0}^{q-1} |a\rangle |0\rangle. \tag{5.1}$$

This step is relatively easy, since all it entails is putting each bit in the first register into the superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

Next, we compute $x^a \pmod n$ in the second register as described in §3. Since we keep a in the first register this can be done reversibly. This leaves our machine in the state

$$\frac{1}{q^{1/2}} \sum_{a=0}^{q-1} |a\rangle |x^a \pmod n\rangle. \tag{5.2}$$

We then perform our Fourier transform A_q on the first register, as described in §4, mapping $|a\rangle$ to

$$\frac{1}{q^{1/2}} \sum_{c=0}^{q-1} \exp(2\pi i a c/q) |c\rangle. \tag{5.3}$$

That is, we apply the unitary matrix with the (a, c) entry equal to $\frac{1}{q^{1/2}} \exp(2\pi i a c/q)$. This leaves our machine in state

$$\frac{1}{q} \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} \exp(2\pi i a c/q) |c\rangle |x^a \pmod n\rangle. \tag{5.4}$$

Finally, we observe the machine. It would be sufficient to observe solely the value of $|c\rangle$ in the first register, but for clarity we will assume that we observe both $|c\rangle$ and $|x^a \pmod n\rangle$. We now compute the probability that our machine ends in a particular state $|c, x^k \pmod n\rangle$, where we may assume $0 \le k < r$. Summing over all possible ways to reach the state $|c, x^k \pmod n\rangle$, we find that this probability is

$$\left| \frac{1}{q} \sum_{a:\, x^a \equiv x^k} \exp(2\pi i a c/q) \right|^2. \tag{5.5}$$

where the sum is over all a, $0 \le a < q$, such that $x^a \equiv x^k \pmod n$. Because the order of x is r, this sum is over all a satisfying $a \equiv k \pmod r$. Writing $a = br + k$, we find that the above probability is

$$\left| \frac{1}{q} \sum_{b=0}^{\lfloor (q-k-1)/r \rfloor} \exp(2\pi i (br + k)c/q) \right|^2. \tag{5.6}$$

We can ignore the term of $\exp(2\pi i k c/q)$, as it can be factored out of the sum and has magnitude 1. We can also replace rc with $\{rc\}_q$, where $\{rc\}_q$ is the residue which is congruent to $rc \pmod q$ and is in the range $-q/2 < \{rc\}_q \le q/2$. This leaves us with the expression

$$\left| \frac{1}{q} \sum_{b=0}^{\lfloor (q-k-1)/r \rfloor} \exp(2\pi i b \{rc\}_q/q) \right|^2. \tag{5.7}$$

We will now show that if $\{rc\}_q$ is small enough, all the amplitudes in this sum will be in nearly the same direction (i.e., have close to the same phase), and thus make the sum large. Turning the sum into an integral, we obtain

$$\frac{1}{q} \int_0^{\left\lfloor \frac{q-k-1}{r} \right\rfloor} \exp(2\pi i b \{rc\}_q/q)\, db + O\left(\frac{\lfloor (q-k-1)/r \rfloor}{q} \left(\exp(2\pi i \{rc\}_q/q) - 1 \right) \right). \tag{5.8}$$

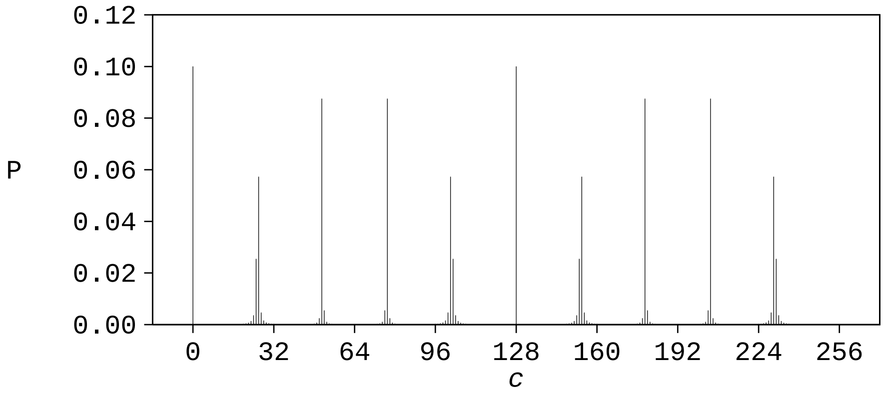

FIG. 5.1. *The probability* P *of observing values of c between* 0 *and* 255*, given $q = 256$ and $r = 10$.*

If $|\{rc\}_q| \le r/2$, the error term in the above expression is easily seen to be bounded by $O(1/q)$. We now show that if $|\{rc\}_q| \le r/2$, the above integral is large, so the probability of obtaining a state $|c, x^k \pmod n\rangle$ is large. Note that this condition depends only on c and is independent of k. Substituting $u = rb/q$ in the above integral, we get

$$\frac{1}{r}\int_0^{\frac{r}{q}\left\lfloor\frac{q-k-1}{r}\right\rfloor} \exp\left(2\pi i \frac{\{rc\}_q}{r} u\right) du. \tag{5.9}$$

Since $k < r$, approximating the upper limit of integration by 1 results in only a $O(1/q)$ error in the above expression. If we do this, we obtain the integral

$$\frac{1}{r}\int_0^1 \exp\left(2\pi i \frac{\{rc\}_q}{r} u\right) du. \tag{5.10}$$

Letting $\{rc\}_q/r$ vary between $-\frac{1}{2}$ and $\frac{1}{2}$, the absolute magnitude of the integral (5.10) is easily seen to be minimized when $\{rc\}_q/r = \pm\frac{1}{2}$, in which case the absolute value of expression (5.10) is $2/(\pi r)$. The square of this quantity is a lower bound on the probability that we see any particular state $|c, x^k \pmod n\rangle$ with $\{rc\}_q \le r/2$; this probability is thus asymptotically bounded below by $4/(\pi^2 r^2)$, and so is at least $1/3r^2$ for sufficiently large n.

The probability of seeing a given state $|c, x^k \pmod n\rangle$ will thus be at least $1/3r^2$ if

$$\frac{-r}{2} \le \{rc\}_q \le \frac{r}{2}, \tag{5.11}$$

i.e., if there is a d such that

$$\frac{-r}{2} \le rc - dq \le \frac{r}{2}. \tag{5.12}$$

Dividing by rq and rearranging the terms gives

$$\left|\frac{c}{q} - \frac{d}{r}\right| \le \frac{1}{2q}. \tag{5.13}$$

We know c and q. Because $q > n^2$, there is at most one fraction d/r with $r < n$ that satisfies the above inequality. Thus, we can obtain the fraction d/r in lowest terms by rounding c/q to the nearest fraction having a denominator smaller than n. This fraction can be found in polynomial time by using a continued fraction expansion of c/q, which finds all the best approximations of c/q by fractions [Knuth 1981, Hardy and Wright 1979, Chapter X].

The exact probabilities as given by (5.7) for an example case with $r = 10$ and $q = 256$ are plotted in Fig. 5.1. The value $r = 10$ could occur when factoring 33 if x were chosen to be 5, for example. Here q is taken smaller than 33^2 so as to make the values of c in the plot distinguishable; this does not change the functional structure of $\mathrm{P}(c)$. Note that with high probability the observed value of c is near an integral multiple of $q/r = 256/10$.

If we have the fraction d/r in lowest terms, and if d happens to be relatively prime to r, this will give us r. We will now count the number of states $|c, x^k \pmod n\rangle$ which enable us to compute r in this way. There are $\phi(r)$ possible values of d relatively prime to r, where ϕ is Euler's totient function [Knuth 1981, Hardy and Wright 1979, §5.5]. Each of these fractions d/r is close to one fraction c/q with $|c/q - d/r| \le 1/2q$. There are also r possible values for x^k, since r is the order of x. Thus, there are $r\phi(r)$ states $|c, x^k \pmod n\rangle$ which would enable us to obtain r. Since each of these states occurs with probability at least $1/3r^2$, we obtain r with probability at least $\phi(r)/3r$. Using the theorem that $\phi(r)/r > \delta/\log\log r$ for some constant δ [Hardy and Wright 1979, Theorem 328], this shows that we find r at least a $\delta/\log\log r$ fraction of the time, so by repeating this experiment only $O(\log\log r)$ times, we are assured of a high probability of success.

In practice, assuming that quantum computation is more expensive than classical computation, it would be worthwhile to alter the above algorithm so as to perform less quantum computation and more postprocessing. First, if the observed state is $|c\rangle$, it would be wise to also try numbers close to c such as $c \pm 1$, $c \pm 2$, …, since these also have a reasonable chance of being close to a fraction qd/r. Second, if $c/q \approx d/r$ where d and r have a common factor, this factor is likely to be small. Thus, if the observed value of c/q is rounded off to d'/r' in lowest terms, for a candidate r one should consider not only r' but also its small multiples $2r'$, $3r'$, …, to see if these are the actual order of x. Although the first technique will only reduce the expected number of trials required to find r by a constant factor, the second technique will reduce the expected number of trials for the hardest n from $O(\log\log n)$ to $O(1)$ if the first $(\log n)^{1+\epsilon}$ multiples of r' are considered [Odylzko 1995]. A third technique, if two candidates for r, say r_1 and r_2, have been found, is to test the least common multiple of r_1 and r_2 as a candidate r. This third technique is also able to reduce the expected number of trials to a constant [Knill 1995], and will work in some cases where the first two techniques fail.

Note that in this algorithm for determining the order of an element, we did not use many of the properties of multiplication $\pmod n$. In fact, if we have a permutation f mapping the set $\{0, 1, 2, \ldots, n-1\}$ into itself such that its kth iterate, $f^{(k)}(a)$, is computable in time polynomial in $\log n$ and $\log k$, the same algorithm will be able to find the order of an element a under f, i.e., the minimum r such that $f^{(r)}(a) = a$.

6. Discrete logarithms. For every prime p, the multiplicative group $\pmod p$ is cyclic, that is, there are generators g such that 1, g, g^2, …, g^{p-2} comprise all the non-zero residues $\pmod p$ [Hardy and Wright 1979, Theorem 111, Knuth 1981]. Suppose we are given a prime p and such a generator g. The *discrete logarithm* of a number x with respect to p and g is the integer r with $0 \leq r < p-1$ such that $g^r \equiv x \pmod{p}$. The fastest algorithm known for finding discrete logarithms modulo arbitrary primes p is Gordon's [1993] adaptation of the number field sieve, which runs in time $\exp(O(\log p)^{1/3}(\log\log p)^{2/3}))$. We show how to find discrete logarithms on a quantum computer using two modular exponentiations and two quantum Fourier transforms.

This algorithm will use three quantum registers. We first find q a power of 2 such that q is close to p, i.e., with $p < q < 2p$. Next, we put the first two registers in our quantum computer in the uniform superposition of all $|a\rangle$ and $|b\rangle \pmod{p-1}$. One way to do this in quantum polynomial time is to put the register in a uniform superposition of all the numbers from 0 to $q-1$, use quantum computation to test whether the number is less than p, and restart the algorithm if the results of this test are unfavorable. We next compute $g^a x^{-b} \pmod{p}$ in the third register. This leaves our machine in the state

$$\frac{1}{p-1}\sum_{a=0}^{p-2}\sum_{b=0}^{p-2}|a, b, g^a x^{-b} \pmod{p}\rangle. \tag{6.1}$$

As before, we use the Fourier transform A_q to send $|a\rangle \to |c\rangle$ and $|b\rangle \to |d\rangle$ with probability amplitude $\frac{1}{q}\exp(2\pi i(ac+bd)/q)$. This is, we take the state $|a, b\rangle$ to the state

$$\frac{1}{q}\sum_{c=0}^{q-1}\sum_{d=0}^{q-1}\exp\left(\tfrac{2\pi i}{q}(ac+bd)\right)|c, d\rangle. \tag{6.2}$$

This leaves our quantum computer in the state

$$\frac{1}{(p-1)q}\sum_{a,b=0}^{p-2}\;\sum_{c,d=0}^{q-1}\exp\left(\tfrac{2\pi i}{q}(ac+bd)\right)|c, d, g^a x^{-b} \pmod{p}\rangle. \tag{6.3}$$

Finally, we observe the state of the quantum computer.

The probability of observing a state $|c, d, y\rangle$ with $y \equiv g^k \pmod{p}$ is

$$\left|\frac{1}{(p-1)q}\sum_{\substack{a,b\\ a-rb\equiv k}}\exp\left(\tfrac{2\pi i}{q}(ac+bd)\right)\right|^2 \tag{6.4}$$

where the sum is over all (a, b) such that $a - rb \equiv k \pmod{p-1}$. Note that we now have two moduli to deal with, $p-1$ and q. While this makes keeping track of things more confusing, it does not pose serious problems. We now use the relation

$$a = br + k - (p-1)\left\lfloor \tfrac{br+k}{p-1}\right\rfloor \tag{6.5}$$

and substitute (6.5) in the expression (6.4) to obtain the amplitude on $|c, d, g^k \pmod{p}\rangle$, which is

$$\frac{1}{(p-1)q}\sum_{b=0}^{p-2}\exp\left(\tfrac{2\pi i}{q}\left(brc + kc + bd - c(p-1)\left\lfloor\tfrac{br+k}{p-1}\right\rfloor\right)\right). \tag{6.6}$$

The square of the absolute value of this amplitude is the probability of observing the state $|c, d, g^k \pmod p\rangle$. We will now analyze the expression (6.6). First, a factor of $\exp(2\pi ikc/q)$ can be taken out of all the terms and ignored, because it does not change the probability. Next, we split the exponent into two parts and factor out b to obtain

$$\frac{1}{(p-1)q}\sum_{b=0}^{p-2}\exp\left(\tfrac{2\pi i}{q}bT\right)\exp\left(\tfrac{2\pi i}{q}V\right), \tag{6.7}$$

where

$$T = rc + d - \tfrac{r}{p-1}\{c(p-1)\}_q, \tag{6.8}$$

and

$$V = \left(\tfrac{br}{p-1} - \left\lfloor \tfrac{br+k}{p-1}\right\rfloor\right)\{c(p-1)\}_q. \tag{6.9}$$

Here by $\{z\}_q$ we mean the residue of $z \pmod q$ with $-q/2 < \{z\}_q \le q/2$, as in (5.7).

We now classify possible outputs (observed states) of the quantum computer as "good" or "bad." We will show that if we get enough good outputs, then we will likely be able to deduce r, and that furthermore, the chance of getting a good output is constant. The idea is that if

$$\left|\{T\}_q\right| = \left|rc + d - \tfrac{r}{p-1}\{c(p-1)\}_q - jq\right| \le \frac{1}{2}, \tag{6.10}$$

where j is the closest integer to T/q, then as b varies between 0 and $p-2$, the phase of the first exponential term in (6.7) only varies over at most half of the unit circle. Further, if

$$|\{c(p-1)\}_q| \le q/12, \tag{6.11}$$

then $|V|$ is always at most $q/12$, so the phase of the second exponential term in (6.7) never is farther than $\exp(\pi i/6)$ from 1. If conditions (6.10) and (6.11) both hold, we will say that an output is good. We will show that if both conditions hold, then the contribution to the probability from the corresponding term is significant. Furthermore, both conditions will hold with constant probability, and a reasonable sample of c's for which condition (6.10) holds will allow us to deduce r.

We now give a lower bound on the probability of each good output, i.e., an output that satisfies conditions (6.10) and (6.11). We know that as b ranges from 0 to $p-2$, the phase of $\exp(2\pi ibT/q)$ ranges from 0 to $2\pi iW$ where

$$W = \frac{p-2}{q}\left(rc + d - \frac{r}{p-1}\{c(p-1)\}_q - jq\right). \tag{6.12}$$

It follows from (6.10) that $|W| \le (p-2)/(2q) \le 1/2$. Thus, the component of the amplitude of the first exponential in the summand of (6.7) in the direction

$$\exp(\pi iW) \tag{6.13}$$

is at least $\cos(2\pi(W/2 - Wb/(p-2)))$, where $2\pi(W/2 - Wb/(p-2))$ is between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$. By condition (6.11), the phase can vary by at most $\pi i/6$ due to the second exponential $\exp(2\pi iV/q)$. Applying this variation in the manner that minimizes the component in the direction (6.13), we get that the component in this direction is at least

$$\cos(2\pi\,|W/2 - Wb/(p-2)| + \tfrac{\pi}{6}). \tag{6.14}$$

Thus we get that the absolute value of the amplitude (6.7) is at least

$$\frac{1}{(p-1)q}\sum_{b=0}^{p-2}\cos\left(2\pi\,|W/2 - Wb/(p-2)| + \tfrac{\pi}{6}\right). \tag{6.15}$$

Replacing this sum with an integral, we get that the absolute value of this amplitude is at least

$$\frac{2}{q}\int_0^{1/2}\cos(\tfrac{\pi}{6} + 2\pi|W|u)du \;+\; O\Big(\tfrac{W}{pq}\Big). \tag{6.16}$$

From condition (6.10), $|W| \le \frac{1}{2}$, so the error term is $O(\frac{1}{pq})$. As W varies between $-\frac{1}{2}$ and $\frac{1}{2}$, the integral (6.16) is minimized when $|W| = \frac{1}{2}$. Thus, the probability of arriving at a state $|c, d, y\rangle$ that satisfies both conditions (6.10) and (6.11) is at least

$$\left(\frac{1}{q}\frac{2}{\pi}\int_{\pi/6}^{2\pi/3}\cos u\; du\right)^2, \tag{6.17}$$

or at least $.054/q^2 > 1/(20q^2)$.

We will now count the number of pairs (c, d) satisfying conditions (6.10) and (6.11). The number of pairs (c, d) such that (6.10) holds is exactly the number of possible c's, since for every c there is exactly one d such that (6.10) holds. Unless $\gcd(p-1, q)$ is large, the number of c's for which (6.11) holds is approximately $q/6$, and even if it is large, this number is at least $q/12$. Thus, there are at least $q/12$ pairs (c, d) satisfying both conditions. Multiplying by $p-1$, which is the number of possible y's, gives approximately $pq/12$ good states $|c, d, y\rangle$. Combining this calculation with the lower bound $1/(20q^2)$ on the probability of observing each good state gives us that the probability of observing some good state is at least $p/(240q)$, or at least $1/480$ (since $q < 2p$). Note that each good c has a probability of at least $(p-1)/(20q^2) \ge 1/(40q)$ of being observed, since there $p-1$ values of y and one value of d with which c can make a good state $|c, d, y\rangle$.

We now want to recover r from a pair c, d such that

$$-\frac{1}{2q} \le \frac{d}{q} + r\left(\frac{c(p-1) - \{c(p-1)\}_q}{(p-1)q}\right) \le \frac{1}{2q} \qquad (\mathrm{mod}\ 1), \tag{6.18}$$

where this equation was obtained from condition (6.10) by dividing by q. The first thing to notice is that the multiplier on r is a fraction with denominator $p-1$, since q evenly divides $c(p-1) - \{c(p-1)\}_q$. Thus, we need only round d/q off to the nearest multiple of $1/(p-1)$ and divide (mod $p-1$) by the integer

$$c' = \frac{c(p-1) - \{c(p-1)\}_q}{q} \tag{6.19}$$

to find a candidate r. To show that the quantum calculation need only be repeated a polynomial number of times to find the correct r requires only a few more details.

The problem is that we cannot divide by a number c' which is not relatively prime to $p-1$.

For the discrete log algorithm, we do not know that all possible values of c' are generated with reasonable likelihood; we only know this about one-twelfth of them. This additional difficulty makes the next step harder than the corresponding step in the algorithm for factoring. If we knew the remainder of r modulo all prime powers dividing $p-1$, we could use the Chinese remainder theorem to recover r in polynomial time. We will only be able to prove that we can find this remainder for primes larger than 18, but with a little extra work we will still be able to recover r.

Recall that each good (c, d) pair is generated with probability at least $1/(20q^2)$, and that at least a twelfth of the possible c's are in a good (c, d) pair. From (6.19), it follows that these c's are mapped from c/q to $c'/(p-1)$ by rounding to the nearest integral multiple of $1/(p-1)$. Further, the good c's are exactly those in which c/q is close to $c'/(p-1)$. Thus, each good c corresponds with exactly one c'. We would like to show that for any prime power $p_i^{\alpha_i}$ dividing $p-1$, a random good c' is unlikely to contain p_i. If we are willing to accept a large constant for our algorithm, we can just ignore the prime powers under 18; if we know r modulo all prime powers over 18, we can try all possible residues for primes under 18 with only a (large) constant factor increase in running time. Because at least one twelfth of the c's were in a good (c, d) pair, at least one twelfth of the c''s are good. Thus, for a prime power $p_i^{\alpha_i}$, a random good c' is divisible by $p_i^{\alpha_i}$ with probability at most $12/p_i^{\alpha_i}$. If we have t good c''s, the probability of having a prime power over 18 that divides all of them is therefore at most

$$\sum_{18 < p_i^{\alpha_i} \,\big|\, (p-1)} \left(\frac{12}{p_i^{\alpha_i}}\right)^t , \tag{6.20}$$

where $a|b$ means that a evenly divides b, so the sum is over all prime powers greater than 18 that divide $p-1$. This sum (over all integers > 18) converges for $t = 2$, and goes down by at least a factor of $2/3$ for each further increase of t by 1; thus for some constant t it is less than $1/2$.

Recall that each good c' is obtained with probability at least $1/(40q)$ from any experiment. Since there are $q/12$ good c''s, after $480t$ experiments, we are likely to obtain a sample of t good c''s chosen equally likely from all good c''s. Thus, we will be able to find a set of c''s such that all prime powers $p_i^{\alpha_i} > 20$ dividing $p-1$ are relatively prime to at least one of these c''s. To obtain a polynomial time algorithm, all one need do is try all possible sets of c''s of size t; in practice, one would use an algorithm to find sets of c''s with large common factors. This set gives the residue of r for all primes larger than 18. For each prime p_i less than 18, we have at most 18 possibilities for the residue modulo $p_i^{\alpha_i}$, where α_i is the exponent on prime p_i in the prime factorization of $p-1$. We can thus try all possibilities for residues modulo powers of primes less than 18: for each possibility we can calculate the corresponding r using the Chinese remainder theorem and then check to see whether it is the desired discrete logarithm.

If one were to actually program this algorithm there are many ways in which the efficiency could be increased over the efficiency shown in this paper. For example, the estimate for the number of good c''s is likely too low, especially since weaker conditions than (6.10) and (6.11) should suffice. This means that the number of times the experiment need be run could be reduced. It also seems improbable that the distribution of bad values of c' would have any relationship to primes under 18; if this is true, we need not treat small prime powers separately.

This algorithm does not use very many properties of $\mathbf{Z}_p$, so we can use the same algorithm to find discrete logarithms over other fields such as Z_{p^α}, as long as the field has a cyclic multiplicative group. All we need is that we know the order of the generator, and that we can multiply and take inverses of elements in polynomial time. The order of the generator could in fact be computed using the quantum order-finding algorithm given in §5 of this paper. Boneh and Lipton [1995] have generalized the algorithm so as to be able to find discrete logarithms when the group is abelian but not cyclic.

7. Comments and open problems. It is currently believed that the most difficult aspect of building an actual quantum computer will be dealing with the problems of imprecision and decoherence. It was shown by Bernstein and Vazirani [1993] that the quantum gates need only have precision $O(1/t)$ in order to have a reasonable probability of completing t steps of quantum computation; that is, there is a c such that if the amplitudes in the unitary matrices representing the quantum gates are all perturbed by at most c/t, the quantum computer will still have a reasonable chance of producing the desired output. Similarly, the decoherence needs to be only polynomially small in t in order to have a reasonable probability of completing t steps of computation successfully. This holds not only for the simple model of decoherence where each bit has a fixed probability of decohering at each time step, but also for more complicated models of decoherence which are derived from fundamental quantum mechanical considerations [Unruh 1995, Palma et al. 1996, Chuang et al. 1995]. However, building quantum computers with high enough precision and low enough decoherence to accurately perform long computations may present formidable difficulties to experimental physicists. In classical computers, error probabilities can be reduced not only though hardware but also through software, by the use of redundancy and error-correcting codes. The most obvious method of using redundancy in quantum computers is ruled out by the theorem that quantum bits cannot be cloned [Peres 1993, §9-4], but this argument does not rule out more complicated ways of reducing inaccuracy or decoherence using software. In fact, some progress in the direction of reducing inaccuracy [Berthiaume et al. 1994] and decoherence [Shor 1995] has already been made. The result of Bennett et al. [1996] that quantum bits can be faithfully transmitted over a noisy quantum channel gives further hope that quantum computations can similarly be faithfully carried out using noisy quantum bits and noisy quantum gates.

Discrete logarithms and factoring are not in themselves widely useful problems. They have only become useful because they have been found to be crucial for public-key cryptography, and this application is in turn possible only because they have been presumed to be difficult. This is also true of the generalizations of Boneh and Lipton [1995] of these algorithms. If the only uses of quantum computation remain discrete logarithms and factoring, it will likely become a special-purpose technique whose only *raison d'être* is to thwart public key cryptosystems. However, there may be other hard problems which could be solved asymptotically faster with quantum computers. In particular, of interesting problems not known to be NP-complete, the problem of finding a short vector in a lattice [Adleman 1994, Adleman and McCurley 1994] seems as if it might potentially be amenable to solution by a quantum computer.

In the history of computer science, however, most important problems have turned out to be either polynomial-time or NP-complete. Thus quantum computers will likely not become widely useful unless they can solve NP-complete problems. Solving NP-complete problems efficiently is a Holy Grail of theoretical computer science which very few people expect to be possible on a classical computer. Finding polynomial-time algorithms for solving these problems on a quantum computer would be a momentous discovery. There are some weak indications that quantum computers are not powerful enough to solve NP-complete problems [Bennett et al. 1997], but I do not believe that this potentiality should be ruled out as yet.

Acknowledgements. I would like to thank Jeff Lagarias for finding and fixing a critical error in the first version of the discrete log algorithm. I would also like to thank him, David Applegate, Charlie Bennett, Gilles Brassard, Andrew Odlyzko, Dan Simon, Bob Solovay, Umesh Vazirani, and correspondents too numerous to list, for productive discussions, for corrections to and improvements of early drafts of this paper, and for pointers to the literature.

REFERENCES

L. M. ADLEMAN (1994), *Algorithmic number theory—The complexity contribution,* in Proc. 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, pp. 88–113.

L. M. ADLEMAN AND K. S. MCCURLEY (1994), *Open problems in number-theoretic complexity* II, in Algorithmic Number Theory, Proc. 1994 Algorithmic Number Theory Symposium, Ithaca, NY, May 6–9, Lecture Notes in Computer Science 877, L. M. Adleman and M.-D. Huang, eds., Springer-Verlag, Berlin, pp. 291–322.

A. BARENCO, C. H. BENNETT, R. CLEVE, D. P. DIVINCENZO, N. MARGOLUS, P. SHOR, T. SLEATOR, J. A. SMOLIN, AND H. WEINFURTER (1995a), *Elementary gates for quantum computation,* Phys. Rev. A, 52, pp. 3457–3467.

A. BARENCO, D. DEUTSCH, A. EKERT, AND R. JOZSA (1995b), *Conditional quantum dynamics and logic gates,* Phys. Rev. Lett., 74, pp. 4083–4086.

D. BECKMAN, A. N. CHARI, S. DEVABHAKTUNI, AND J. PRESKILL (1996), *Efficient networks for quantum factoring,* Phys. Rev. A, 54, pp. 1034–1063.

P. BENIOFF (1980), *The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines,* J. Statist. Phys., 22, pp. 563–591.

——— (1982a), *Quantum mechanical Hamiltonian models of Turing machines,* J. Statist. Phys., 29, pp. 515–546.

——— (1982b), *Quantum mechanical Hamiltonian models of Turing machines that dissipate no energy,* Phys. Rev. Lett., 48, pp. 1581–1585.

C. H. BENNETT (1973), *Logical reversibility of computation,* IBM J. Res. Develop., 17, pp. 525–532.

——— (1989), *Time/space trade-offs for reversible computation,* SIAM J. Comput., 18, pp. 766–776.

C. H. BENNETT, E. BERNSTEIN, G. BRASSARD, AND U. VAZIRANI (1997), *Strengths and weaknesses of quantum computing,* SIAM J. Comput., to appear (this issue).

C. H. BENNETT, G. BRASSARD, S. POPESCU, B. SCHUMACHER, J. A. SMOLIN, AND W. K. WOOTERS (1996), *Purification of noisy entanglement and faithful teleportation via noisy channels,* Phys. Rev. Lett., 76, pp. 722–725.

E. BERNSTEIN AND U. VAZIRANI (1993), *Quantum complexity theory,* in Proc. 25th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, pp. 11–20.

A. BERTHIAUME AND G. BRASSARD (1992), *The quantum challenge to structural complexity theory,* in Proc. Seventh Annual Structure in Complexity Theory Conference, IEEE Computer Society Press, Los Alamitos, CA, pp. 132–137.

——— (1994), *Oracle quantum computing,* J. Mod. Optics, 41, pp. 2521–2535.

A. BERTHIAUME, D. DEUTSCH, AND R. JOZSA (1994), *The stabilisation of quantum computations,* in Proc. Workshop on Physics of Computation: PhysComp '94, IEEE Computer Society Press, Los Alamitos, CA, pp. 60–62.

M. BIAFORE (1994), *Can quantum computers have simple Hamiltonians,* in Proc. Workshop on Physics of Computation: PhysComp '94, IEEE Computer Society Press, Los Alamitos, CA, pp. 63–68.

D. BONEH AND R. J. LIPTON (1995), *Quantum cryptanalysis of hidden linear functions,* Advances in Cryptology—CRYPTO '95, Proc. 15th Annual International Cryptology Conference, Santa

Barbara, CA, Aug. 27–31, D. Coppersmith, ed. Springer-Verlag, Berlin, pp. 424–437.

J. F. CANNY AND J. REIF (1987), *New lower bound techniques for robot motion planning problems,* in Proc. 28th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, pp. 49–60.

J. CHOI, J. SELLEN, AND C.-K. YAP (1995), *Precision-sensitive Euclidean shortest path in 3-space,* in Proc. 11th Annual Symposium on Computational Geometry, Association for Computing Machinery, New York, pp. 350–359.

I. L. CHUANG, R. LAFLAMME, P. W. SHOR, AND W. H. ZUREK (1995), *Quantum computers, factoring and decoherence,* Science, 270, pp. 1633–1635.

I. L. CHUANG AND Y. YAMAMOTO (1995), *A simple quantum computer,* Phys. Rev. A, 52, pp. 3489–3496.

A. CHURCH (1936), *An unsolvable problem of elementary number theory,* Amer. J. Math., 58, pp. 345–363.

J. I. CIRAC AND P. ZOLLER (1995), *Quantum computations with cold trapped ions,* Phys. Rev. Lett., 74, pp. 4091–4094.

R. CLEVE (1994), *A note on computing Fourier transforms by quantum programs,* preprint.

D. COPPERSMITH (1994), *An approximate Fourier transform useful in quantum factoring,* IBM Research Report RC 19642.

D. DEUTSCH (1985), *Quantum theory, the Church–Turing principle and the universal quantum computer,* Proc. Roy. Soc. London Ser. A, 400, pp. 96–117.

——— (1989), *Quantum computational networks,* Proc. Roy. Soc. London Ser. A, 425, pp. 73–90.

D. DEUTSCH, A. BARENCO, AND A. EKERT (1995), *Universality of quantum computation,* Proc. Roy. Soc. London Ser. A, 449, pp. 669-677.

D. DEUTSCH AND R. JOZSA (1992), *Rapid solution of problems by quantum computation,* Proc. Roy. Soc. London Ser. A, 439, pp. 553–558.

D. P. DIVINCENZO (1995), *Two-bit gates are universal for quantum computation,* Phys. Rev. A, 51, pp. 1015–1022.

A. EKERT AND R. JOZSA (1996), *Shor's quantum algorithm for factorising numbers,* Rev. Mod. Phys., 68, pp. 733–753.

R. FEYNMAN (1982), *Simulating physics with computers,* Internat. J. Theoret. Phys., 21, pp. 467–488.

——— (1986), *Quantum mechanical computers,* Found. Phys., 16, pp. 507–531; originally in Optics News (February 1985), pp. 11–20.

E. FREDKIN AND T. TOFFOLI (1982), *Conservative logic,* Internat. J. Theoret. Phys., 21, pp. 219–253.

D. M. GORDON (1993), *Discrete logarithms in* GF(p) *using the number field sieve,* SIAM J. Discrete Math., 6, pp. 124–139.

R. B. GRIFFITHS AND C.-S. NIU (1996), *Semiclassical Fourier tranform for quantum computation,* Phys. Rev. Lett., 76, pp. 3228–3231.

G. H. HARDY AND E. M. WRIGHT (1979), *An Introduction to the Theory of Numbers,* Fifth ed., Oxford University Press, New York.

J. HARTMANIS AND J. SIMON (1974), *On the power of multiplication in random access machines,* in Proc. 15th Annual Symposium on Switching and Automata Theory, IEEE Computer Society, Long Beach, CA, pp. 13–23.

A. KARATSUBA AND YU. OFMAN (1962), *Multiplication of multidigit numbers on automata,* Dokl. Akad. Nauk SSSR, 145, pp. 293–294 (in Russian); Sov. Phys. Dokl., 7 (1963), pp. 595–596 (English translation).

E. KNILL (1995), personal communication.

D. E. KNUTH (1981), *The Art of Computer Programming, Vol.* 2: *Seminumerical Algorithms,* Second ed., Addison-Wesley, Reading, MA.

R. LANDAUER (1995), *Is quantum mechanics useful*? Philos. Trans. Roy. Soc. London Ser. A, 353, pp. 367-376.

——— (1996), *Is quantum mechanically coherent computation useful*? in Proceedings of the Drexel-4 Symposium on Quantum Nonintegrability—Quantum Classical Correspondence, D. H. Feng and B-L. Hu, eds., International Press, Cambridge, MA, to appear.

Y. LECERF (1963), *Machines de Turing réversibles. Récursive insolubilité en* $n \in N$ *de l'équation* $u = \theta^n u$, *où* θ *est un isomorphisme de codes,* C. R. Acad. Française Sci., 257, pp. 2597–2600.

A. K. LENSTRA AND H. W. LENSTRA, JR., eds. (1993), *The Development of the Number Field Sieve,* Lecture Notes in Mathematics 1554, Springer-Verlag, Berlin.

A. K. LENSTRA, H. W. LENSTRA, JR., M. S. MANASSE, AND J. M. POLLARD (1990), *The number field sieve,* in Proc. 22nd Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, pp. 564–572; expanded version in Lenstra and Lenstra [1993], pp. 11–42.

R. Y. LEVINE AND A. T. SHERMAN (1990), *A note on Bennett's time-space tradeoff for reversible* *computation,* SIAM J. Comput., 19, pp. 673–677.

S. LLOYD (1993), *A potentially realizable quantum computer,* Science, 261, pp. 1569–1571.

——— (1994), *Envisioning a quantum supercomputer,* Science, 263, p. 695.

——— (1995), *Almost any quantum logic gate is universal,* Phys. Rev. Lett., 75, pp. 346–349.

N. MARGOLUS (1986), *Quantum computation,* Ann. New York Acad. Sci., 480, pp. 487–497.

——— (1990), *Parallel quantum computation,* in Complexity, Entropy and the Physics of Information, Santa Fe Institute Studies in the Sciences of Complexity Vol. VIII, W. H. Zurek, ed., Addison-Wesley, Reading, MA, pp. 273–287.

G. L. MILLER (1976), *Riemann's hypothesis and tests for primality,* J. Comput. System Sci., 13, pp. 300–317.

A. M. ODLYZKO (1995), personal communication.

G. M. PALMA, K.-A. SUOMINEN, AND A. K. EKERT (1996), *Quantum computers and dissipation,* Proc. Roy. Soc. London Ser. A, 452, pp. 567–584.

A. PERES (1993), *Quantum Theory: Concepts and Methods,* Kluwer Academic Publishers, Dordrecht, The Netherlands.

C. POMERANCE (1987), *Fast, rigorous factorization and discrete logarithm algorithms,* in Discrete Algorithms and Complexity, Proc. Japan–US Joint Seminar, June 4–6, 1986, Kyoto, D. S. Johnson, T. Nishizeki, A. Nozaki, and H. S. Wilf, eds., Academic Press, pp. 119–143.

E. POST (1936), *Finite combinatory processes. Formulation* I, J. Symbolic Logic, 1, pp. 103–105.

R. L. RIVEST, A. SHAMIR, AND L. ADLEMAN (1978), *A method of obtaining digital signatures and public-key cryptosystems,* Comm. Assoc. Comput. Mach., 21, pp. 120–126.

L. A. RUBEL (1989), *Digital simulation of analog computation and Church's thesis,* J. Symbolic Logic, 54, pp. 1011–1017.

A. SCHÖNHAGE (1982), *Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients,* in Computer Algebra EUROCAM '82, Lecture Notes in Computer Science 144, J. Calmet, ed., Springer-Verlag, Berlin, pp. 3–15.

A. SCHÖNHAGE, A. F. W. GROTEFELD, AND E. VETTER (1994), *Fast Algorithms: A Multitape Turing Machine Implementation,* B. I. Wissenschaftsverlag, Mannheim, Germany.

A. SCHÖNHAGE AND V. STRASSEN (1971), *Schnelle Multiplikation grosser Zahlen,* Computing, 7, pp. 281–292.

P. W. SHOR (1994), *Algorithms for quantum computation: Discrete logarithms and factoring,* in Proc. 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, pp. 124–134.

——— (1995), *Scheme for reducing decoherence in quantum computer memory,* Phys. Rev. A, 52, pp. 2493–2496.

D. SIMON (1994), *On the power of quantum computation,* in Proc. 35th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, pp. 116–123.

T. SLEATOR AND H. WEINFURTER (1995), *Realizable universal quantum logic gates,* Phys. Rev. Lett., 74, pp. 4087–4090.

R. SOLOVAY (1995), personal communication.

K. STEIGLITZ (1988), *Two non-standard paradigms for computation: Analog machines and cellular automata,* in Performance Limits in Communication Theory and Practice, Proc. NATO Advanced Study Institute, Il Ciocco, Castelvecchio Pascoli, Tuscany, Italy, July 7–19, 1986, J. K. Skwirzynski, ed., Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 173–192.

W. G. TEICH, K. OBERMAYER, AND G. MAHLER (1988), *Structural basis of multistationary quantum systems* II: *Effective few-particle dynamics,* Phys. Rev. B, 37, pp. 8111–8121.

T. TOFFOLI (1980), *Reversible computing,* in Automata, Languages and Programming, Seventh Colloquium, Lecture Notes in Computer Science 84, J. W. de Bakker and J. van Leeuwen, eds., Springer-Verlag, Berlin, pp. 632–644.

A. M. TURING (1936), *On computable numbers, with an application to the Entscheidungsproblem,* Proc. London Math. Soc. (2), 42, pp. 230–265; Corrections in Proc. London Math. Soc. (2), 43 (1937), pp. 544–546.

W. G. UNRUH (1995), *Maintaining coherence in quantum computers,* Phys. Rev. A, 51, pp. 992–997.

P. VAN EMDE BOAS (1990), *Machine models and simulations,* in Handbook of Theoretical Computer Science Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, pp. 1–66.

A. VERGIS, K. STEIGLITZ, AND B. DICKINSON (1986), *The complexity of analog computation,* Math. Comput. Simulation, 28, pp. 91–113.

A. YAO (1993), *Quantum circuit complexity,* in Proc. 34th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, pp. 352–361.