In: Proceedings, 35th Annual Symposium on Foundations of Computer Science,
Santa Fe, NM, November 20–22, 1994, IEEE Computer Society Press, pp. 124–134.

Algorithms for Quantum Computation: Discrete Logarithms and Factoring

Peter W. Shor
AT&T Bell Labs
Room 2D-149
600 Mountain Ave.
Murray Hill, NJ 07974, USA

Abstract

A computer is generally considered to be a universal computational device; i.e., it is believed able to simulate any physical computational device with a increase in computation time of at most a polynomial factor. It is not clear whether this is still true when quantum mechanics is taken into consideration. Several researchers, starting with David Deutsch, have developed models for quantum mechanical computers and have investigated their computational properties. This paper gives Las Vegas algorithms for finding discrete logarithms and factoring integers on a quantum computer that take a number of steps which is polynomial in the input size, e.g., the number of digits of the integer to be factored. These two problems are generally considered hard on a classical computer and have been used as the basis of several proposed cryptosystems. (We thus give the first examples of quantum cryptanalysis.)

1 Introduction

Since the discovery of quantum mechanics, people have found the behavior of the laws of probability in quantum mechanics counterintuitive. Because of this behavior, quantum mechanical phenomena behave quite differently than the phenomena of classical physics that we are used to. Feynman seems to have been the first to ask what effect this has on computation [13, 14]. He gave arguments as to why this behavior might make it intrinsically computationally expensive to simulate quantum mechanics on a classical (or von Neumann) computer. He also suggested 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. Other early work in the field of quantum mechanics and computing was done by Benioff [1, 2]. Although he did not ask whether quantum mechanics conferred extra power to computation, he did show that a Turing machine could be simulated by the reversible unitary evolution of a quantum process, which is a necessary prerequisite for quantum computation. Deutsch [9, 10] was the first to give an explicit model of quantum computation. He defined both quantum Turing machines and quantum circuits and investigated some of their properties.

The next part of this paper discusses how quantum computation relates to classical complexity classes. We will thus first give a brief intuitive discussion of complexity classes for those readers who do not have this background. There are generally two resources which limit the ability of computers to solve large problems: time and space (i.e., memory). The field of analysis of algorithms considers the asymptotic demands that algorithms make for these resources as a function of the problem size. Theoretical computer scientists generally classify algorithms as efficient when the number of steps of the algorithms grows as a polynomial in the size of the input. The class of problems which can be solved by efficient algorithms is known as P. This classification has several nice properties. For one thing, it does a reasonable job of reflecting the performance of algorithms in practice (although an algorithm whose running time is the tenth power of the input size, say, is not truly efficient). For another, this classification is nice theoretically, as different reasonable machine models produce the same class P. We will see this behavior reappear in quantum computation, where different models for quantum machines will vary in running times by no more than polynomial factors.

There are also other computational complexity classes discussed in this paper. One of these is PSPACE, which are those problems which can be solved with an amount of memory polynomial in the input size. Another important complexity class is NP, which intuitively is the class of exponential search problems. These are problems which may require the search of an exponential size space to find the solution, but for which the solution, once found, may be verified in polynomial time (possibly with a polynomial amount of additional supporting evidence). We will also discuss two other traditional complexity classes. One is BPP, which are problems which can be solved with high probability in polynomial time, given access to a random number generator. The other is $\mathrm{P}^{\#\mathrm{P}}$, which are those problems which could be solved in polynomial time if sums of exponentially many terms could be computed efficiently (where these sums must satisfy the requirement that each term is computable in polynomial time). These classes are related as follows:

$$\mathrm{P} \subseteq \mathrm{BPP}, \mathrm{NP} \subseteq \mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{PSPACE}.$$

The relationship of BPP and NP is not known.

The question of whether using quantum mechanics in a computer allows one to obtain more computational power has not yet been satisfactorily answered. This question was addressed in [11, 6, 7], but it was not shown how to solve any problem in quantum polynomial time that was not known to be solvable in BPP. Recent work on this problem was stimulated by Bernstein and Vazirani's paper [5] which laid the foundations of the quantum computation theory of computational complexity. One of the results contained in this paper was an oracle problem (a problem involving a "black box" subroutine, i.e., a function that the computer is allowed to perform, but for which no code is accessible.) which can be done in polynomial time on a quantum Turing machine and requires super-polynomial time on a classical computer. This was the first indication, other than the fact that nobody knew how to simulate a quantum computer on a classical computer without an exponential slowdown, that quantum computation might obtain a greater than polynomial speedup over classical computation augmented with a random number generator. This result was improved by Simon [29], who gave a much simpler construction of an oracle problem which takes polynomial time on a quantum computer and requires exponential time on a classical computer. Indeed, by viewing Simon's oracle as a subroutine, this result becomes a promise problem which takes polynomial time on a quantum computer and looks as if it would be very difficult on a classical computer (a promise problem is one where the input is guaranteed to satisfy some condition). The algorithm for the "easy case" of discrete log given in this paper is directly analogous to Simon's algorithm with the group Z_2^k replaced by the group Z_{p-1}; I was only able to discover this algorithm after seeing Simon's paper.

In another result in Bernstein and Vazirani's paper, a particular class of quantum Turing machine was rigorously defined and a universal quantum Turing machine was given which could simulate any other quantum Turing machine of this class. Unfortunately, it was not clear whether these quantum Turing machines could simulate other classes of quantum Turing machines, so this result was not entirely satisfactory. Yao [33] has remedied the situation by showing that quantum Turing machines can simulate, and be simulated by, uniform families of polynomial size quantum circuits, with at most polynomial slowdown. He has further defined quantum Turing machines with k heads and showed that these machines can be simulated with slowdown of a factor of 2^k. This seems to show that the class of problems which can be solved in polynomial time on one of these machines, possibly with a bounded probability $\epsilon < 1/3$ of error, is reasonably robust. This class is called BQP in analogy to the classical complexity class BPP, which are those problems which can be solved with a bounded probability of error on a probabilistic Turing machine. This class BQP could be considered the class of problems that are efficiently solvable on a quantum Turing machine.

Since BQP $\subseteq \mathrm{P}^{\#\mathrm{P}} \subseteq$ PSPACE [5], any non-relativized proof that BQP is strictly larger than BPP would imply the structural complexity result BPP $\subsetneq$ PSPACE which is not yet proven. In view of this difficulty, several approaches come to mind; one is showing that BQP $\subseteq$ BPP would lead to a collapse of classical complexity classes which are believed to be different. A second approach is to prove results relative to an oracle. In Bennett *et al.* [4] it is shown that relative to a random oracle, it is not the case that NP $\subseteq$ BQP. This proof in fact suggests that a quantum computer cannot invert one-way functions, but only proves this for one-way oracles, i.e. "black box" functions given as a subroutine which the quantum computer is not allowed to look inside. Such oracle results have been misleading in the past, most notably in the case of IP $=$ PSPACE [15, 28]. A third approach, which we take, is to solve in BQP some well-studied problem for which no polynomial time algorithm is known. This shows that the extra power conferred by quantum interference is at least hard to achieve using classical computation. Both Bernstein and Vazirani [5] and Simon [29] also gave polynomial time algorithms for problems which were not known to be in BPP, but these problems were invented especially for this purpose, although Simon's problem does not appear contrived and could conceivably be useful.

Discrete logarithms and integer factoring are two number-theory problems which have been studied extensively but for which no polynomial-time algorithms are known [16, 20, 21, 26]. In fact, these problems are so widely believed to be hard that cryptosystems based on their hardness have been proposed, and the RSA public key cryptosystem [27], based on the hardness of factoring, is in use. We show that these problems can be solved in BQP.

Currently, nobody knows how to build a quantum computer, although it seems as though it could be possible within the laws of quantum mechanics. Some suggestions have been made as to possible designs for such computers [30, 22, 23, 12], but there will be substantial difficulty in building any of these [18, 32]. Even if it is possible to build small quantum computers, scaling up to machines large enough to do interesting computations could present fundamental difficulties. It is hoped that this paper will stimulate research on whether it is feasible to actually construct a quantum computer.

Even if no 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 algorithm for factoring.

2 Quantum computation

In this section we will give a brief introduction to quantum computation, emphasizing the properties that we will use. For a more complete overview I refer the reader to Simon's paper in this proceedings [29] or to earlier papers on quantum computational complexity theory [5, 33].

In quantum physics, an experiment behaves as if it proceeds down all possible paths simultaneously. Each of these paths has a complex *probability amplitude* determined by the physics of the experiment. The probability of any particular outcome of the experiment is proportional to the square of the absolute value of the sum of the amplitudes of all the paths leading to that outcome. In order to sum over a set of paths, the outcomes have to be identical in all respects, i.e., the universe must be in the same state. A quantum computer behaves in much the same way. The computation proceeds down all possible paths at once, and each path has associated with it a complex amplitude. To determine the probability of any final state of the machine, we add the amplitudes of all the paths which reach that final state, and then square the absolute value of this sum.

An equivalent way of looking at this process is to imagine that the machine is in some *superposition of states* at every step of the computation. We will represent this superposition of states as

$$\sum_i a_i \left|S_i\right\rangle, \tag{2.1}$$

where the amplitudes a_i are complex numbers such that $\sum_i \left|a_i\right|^2 = 1$ and each $\left|S_i\right\rangle$ is a *basis state* of the machine; in a quantum Turing machine, a basis state is defined by what is written on the tape and by the position and state of the head. In a quantum circuit a basis state is defined by the values of the signals on all the wires at some level of the circuit. If the machine is examined at a particular step, the probability of seeing basis state $\left|S_i\right\rangle$ is $\left|a_i\right|^2$; however, by the Heisenberg uncertainty principle, looking at the machine during the computation will disturb the rest of the computation.

The laws of quantum mechanics only permit unitary transformations of the state. 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 the probabilities of obtaining all possible outcomes will add up to one. Further, the definitions of quantum Turing machine and quantum circuit only allow local unitary transformations, that is, unitary transformations on a fixed number of bits.

Perhaps an example will be informative at this point. Suppose our machine is in the superposition of states

$$\tfrac{i}{\sqrt{2}}\left|000\right\rangle + \tfrac{1}{2}\left|100\right\rangle - \tfrac{1}{2}\left|110\right\rangle \tag{2.2}$$

and we apply the unitary transformation

$$\begin{array}{c|cccc|} & 00 & 01 & 10 & 11 \\ 00 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 01 & \frac{1}{2} & \frac{i}{2} & -\frac{1}{2} & -\frac{i}{2} \\ 10 & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 11 & \frac{1}{2} & -\frac{i}{2} & -\frac{1}{2} & \frac{i}{2} \end{array} \tag{2.3}$$

to the last two bits of our state. That is, we multiply the last two bits of the components of the vector (2.2) by the matrix (2.3). The machine will then go to the superposition of states

$$\tfrac{i}{2\sqrt{2}}\left(\left|000\right\rangle + \left|001\right\rangle + \left|010\right\rangle + \left|011\right\rangle\right) + \tfrac{1}{2}\left|101\right\rangle + \tfrac{1}{2}\left|111\right\rangle. \tag{2.4}$$

Notice that the result would have been different had we started with the superposition of states

$$\tfrac{i}{\sqrt{2}}\left|000\right\rangle + \tfrac{1}{2}\left|100\right\rangle + \tfrac{1}{2}\left|110\right\rangle, \tag{2.5}$$

which has the same probabilities of being in any particular configuration if it is observed.

We now give certain properties of quantum computation that will be useful. These facts are not immediately apparent from the definition of quantum Turing machine or quantum circuit, and they are very useful for constructing algorithms for quantum machines.

Fact 1: A deterministic computation is performable on a quantum computer if and only if it is reversible.

From results on reversible computation [3, 19, 31], we can compute any polynomial time function $f(a)$ as long as we keep the input, a, on the machine. To erase a and replace it with $f(a)$ we need in addition that f is one-to-one and that a is computable in polynomial time from $f(a)$; i.e., that both f and f^{-1} are polynomial-time computable.

Fact 2: Any polynomial size unitary matrix can be approximated using a polynomial number of elementary unitary transformations [10, 5, 33] and thus can be approximated in polynomial time on a quantum computer. Further, this approximation is good enough so as to introduce at most a bounded probability of error into the results of the computation.

3 Building unitary transformations

Since quantum computation deals with unitary transformations, it is helpful to be able to build certain useful unitary transformations. In this section we give some techniques for constructing unitary transformations on quantum machines, which will result in our showing how to construct one particular unitary transformation in polynomial time. These transformations will generally be given as matrices, with both rows and columns indexed by states. These states will correspond to representations of integers on the computer; in particular, the rows and columns will be indexed beginning with 0 unless otherwise specified.

A tool we will use repeatedly in this paper is the following unitary transformation, the summation of which gives a Fourier transform. Consider a number a with $0 \le a < q$ for some q where the number of bits of q is polynomial. We will perform the transformation that takes the state $|a\rangle$ to the state

$$\frac{1}{q^{1/2}} \sum_{b=0}^{q-1} |b\rangle \exp(2\pi i a b/q). \tag{3.6}$$

That is, we apply the unitary matrix whose (a, b)'th entry is $\frac{1}{q^{1/2}} \exp(2\pi i a b/q)$. This transformation is at the heart of our algorithms, and we will call this matrix A_q. Since we will use A_q for q of exponential size, we must show how this transformation can be done in polynomial time. In fact, we will only be able to do this for *smooth* numbers q, that is, ones with small prime factors. In this paper, we will deal with *smooth* numbers q which contain no prime power factor that is larger than $(\log q)^c$ for some fixed c. It is also possible to do this transformation in polynomial time for all smooth numbers q; Coppersmith shows how to do this for $q = 2^k$ using what is essentially the fast Fourier transform, and that this substantially reduces the number of operations required to factor [8].

If we know a factorization $q = q_1 q_2 q_3 \cdots q_k$ where $\gcd(q_i, q_j) = 1$ and where k and all of the q_i are of polynomial size we will show how to build the transformation A_q in polynomial time by composing the A_{q_i}. For this, we first need a lemma on quantum computation.

Lemma 3.1 *Suppose the matrix B is a block-diagonal $mn \times mn$ unitary matrix composed of n identical unitary $m \times m$ matrices B' along the diagonal and 0's everywhere else. Suppose further that the state transformation B' can be done in time $T(B')$ on a quantum Turing machine. Then the matrix B can be done in $T(B') + (\log mn)^c$ time on a quantum Turing machine, where c is a constant.*

We will call this matrix B the direct sum of n copies of B' and use the notation $B = \bigoplus_n B'$. This matrix B is the tensor product of B' and I_n, where I_n is the $n \times n$ identity matrix.

Proof: Suppose that we have a number a on our tape. We can reversibly compute α_1 and α_2 from a where $a = m\alpha_1 + \alpha_2$. This computation erases a from our tape and replaces it with α_1 and α_2. Now α_1 tells in which block the row a is contained, and α_2 tells which row of the matrix within that block is the row a. We can then apply B' to α_2 to obtain β_2 (erasing α_2 in the process). Now, combining α_1 and β_2 to obtain $b = m\alpha_1 + \beta_2$ gives the result of B applied to a (with the right amplitude). The computation of B' takes $T(B')$ time, and the rest of the computation is polynomial in $\log m + \log n$. ∎

We now show how to obtain A_q for smooth q. We will decompose A_q into a product of a polynomial number of unitary transformations, all of which are performable in polynomial time; this enables us to construct A_q in polynomial time. Suppose that we have $q = q_1 q_2$ with $\gcd(q_1, q_2) = 1$. What we will do is represent $A_q = CD$, where by rearranging the rows and columns of D we obtain $\bigoplus_{q_1} A_{q_2}$ and rearranging the rows and columns of C we obtain $\bigoplus_{q_2} A_{q_1}$. As long as these rearrangements of the rows and columns of C and D are performable in polynomial time (i.e., given row r, we can find in polynomial time the row r' to which it is taken) and the inverse operations are also performable in polynomial time, then by using the lemma above and recursion we can obtain a polynomial-time way to perform A_q on a quantum computer.

We now need to define C and D and check that $A_q = CD$. To define C and D we need some preliminary definitions. Recall that $q = q_1 q_2$ with q_1 and q_2 relatively prime. Let $\omega = \exp(2\pi i/q)$. Let u be the number $(\bmod\ q)$ such that $u \equiv 0 \pmod{q_1}$ and $u \equiv -1 \pmod{q_2}$. Such a number exists by the Chinese remainder theorem, and can be computed in polynomial time. We will decompose row and column indices a, b and c as follows: $a = \alpha_1 q_2 + \alpha_2$,

$b = \beta_1 q_1 + \beta_2$, and $c = \gamma_1 q_1 + \gamma_2$. Note the asymmetry in the definitions of a, b and c.

We can now define C and D:

$$C(a,b) = \begin{cases} 0 & \text{if } \alpha_2 \neq \beta_1 \\ \frac{1}{q_1^{1/2}}\, \omega^{\alpha_1\beta_2 q_2 + \beta_1\beta_2(u+1)} & \text{otherwise,} \end{cases} \tag{3.7}$$

and

$$D(b,c) = \begin{cases} 0 & \text{if } \beta_2 \neq \gamma_2 \\ \frac{1}{q_2^{1/2}}\, \omega^{\beta_1\gamma_1 q_1 - \beta_1\beta_2 u} & \text{otherwise.} \end{cases} \tag{3.8}$$

It is easy to see that $CD(a,c) = C(a,b)D(b,c)$ where $b = \alpha_2 q_1 + \gamma_2$ since we need $\alpha_2 = \beta_1$ and $\beta_2 = \gamma_2$ to ensure non-zero entries in $C(a,b)$ and $D(b,c)$. Now,

$$\begin{aligned} CD(a,c) &= \frac{1}{q_1^{1/2} q_2^{1/2}}\, \omega^{\alpha_1\beta_2 q_2 + \beta_1\beta_2(u+1) + \beta_1\gamma_1 q_1 - \beta_1\beta_2 u} \\ &= \frac{1}{q^{1/2}}\, \omega^{\alpha_1\gamma_2 q_2 + \alpha_2\gamma_1 q_1 + \alpha_2\gamma_2} \\ &= \frac{1}{q^{1/2}}\, \omega^{(\alpha_1 q_2 + \alpha_2)(\gamma_1 q_1 + \gamma_2)} \\ &= \frac{1}{q^{1/2}}\, \omega^{ac} \end{aligned} \tag{3.9}$$

so $CD(a,c) = A_q(a,c)$.

We will now sketch how to rearrange the rows and columns of C to get the matrix $\bigoplus_{q_2} A_{q_1}$. The matrix C can be put in block-diagonal form where the blocks are indexed by $\alpha_2 = \beta_1$ (since all entries with $\alpha_2 \neq \beta_1$ are 0). Let $u + 1 \equiv tq_2 \pmod q$. Within a given block $\alpha_2 = \beta_1$, the entries look like

$$\begin{aligned} \sqrt{q_1}\; C(a,b) &= \omega^{\alpha_1\beta_2 q_2 + \beta_1\beta_2(u+1)} \\ &= \exp(2\pi i(\alpha_1\beta_2 + \beta_1\beta_2 t)q_2/q) \\ &= \exp(2\pi i(\alpha_1 + \alpha_2 t)\beta_2/q_1). \end{aligned} \tag{3.10}$$

Thus, if we rearrange the rows within this block so that they are indexed by $\alpha' \equiv \alpha_1 + \alpha_2 t \pmod{q_1}$, we obtain the transformation $\alpha' \to \beta_2$ with amplitude $\frac{1}{q_1^{1/2}} \exp(2\pi i\alpha'\beta_2/q_1)$; that is, the transformation given by the unitary matrix with the (α', β_2) entry equal to $\frac{1}{q_1^{1/2}} \exp(2\pi i\alpha'\beta_2/q_1)$, which is A_{q_1}. The matrix D can similarly be rearranged to obtain the matrix $\bigoplus_{q_1} A_{q_2}$.

We also need to show how to find a smooth q that lies between n and $2n$ in polynomial time. There are actually smooth q much closer to n than this, but this is all we need. It is not known how to find smooth numbers very close to n in polynomial time.

Lemma 3.2 *Given n, there is a polynomial-time algorithm to find a number q with $n \le q < 2n$ such that no prime power larger than $c \log q$ divides q, for some constant c independent of n.*

Proof: To find such a q, multiply the primes $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdots p_k$ until the product is larger than n. Now, if this product is larger than $2n$, divide it by the largest prime that keeps the number larger than n. This produces the desired q. There is always a prime between m and $2m$ [17, Theorem 418], so $n \le q < 2n$. The prime number theorem [17, Theorem 6] and some calculation show that the largest prime dividing q is of size $O(\log n)$. ∎

Note that if we are using Coppersmith's transformation A_{2^k} using the 2^kth roots of unity, we set $q = 2^k$ where $k = \lfloor \log_2 n \rfloor + 1$.

4 Discrete log: the easy case

The discrete log problem is: given a prime p, a generator g of the multiplicative group $\pmod p$ and an $x \pmod p$, find an r such that $g^r \equiv x \pmod p$. We will start by giving a polynomial-time algorithm for discrete log on a quantum computer in the case that $p-1$ is smooth. This algorithm is analogous to the algorithm in Simon's paper [29], with the group Z_2^k replaced by Z_{p-1}. The smooth case is not in itself an interesting accomplishment, since there are already polynomial time algorithms for classical computers in this case [25]; however, explaining this case is easier than explaining either the general case of discrete log or the factoring algorithm, and as the three algorithms are similar, this example will illuminate how the more complicated algorithms work.

We will start our algorithm with x, g and p on the tape (i.e., in the quantum memory of our machine). We are trying to compute r such that $g^r \equiv x \pmod p$. Since we will never delete them, x, g, and p are constants, and we will specify a state of our machine by the other contents of the tape.

The algorithm starts out by "choosing" numbers a and $b \pmod{p-1}$ uniformly, so the state of the machine after this step is

$$\frac{1}{p-1} \sum_{a=0}^{p-2} \sum_{b=0}^{p-2} |a, b\rangle\,. \tag{4.11}$$

The algorithm next computes $g^a x^{-b} \pmod p$ reversibly, so we must keep the values a and b on the tape. The state of the machine is now

$$\frac{1}{p-1} \sum_{a=0}^{p-2} \sum_{b=0}^{p-2} \left| a, b, g^a x^{-b} \pmod p \right\rangle. \tag{4.12}$$

What we do now is use the transformation A_{p-1} to map $a \to c$ with amplitude $\frac{1}{(p-1)^{1/2}} \exp(2\pi i ac/(p-1))$ and $b \to d$ with amplitude $\frac{1}{(p-1)^{1/2}} \exp(2\pi i bd/(p-1))$. As was discussed in the previous section, this is a unitary transformation, and since $p-1$ is smooth it can be accomplished in polynomial time on a quantum machine. This leaves the machine in state

$$\frac{1}{(p-1)^2}\sum_{a,b,c,d=0}^{p-2}\exp\left(\tfrac{2\pi i}{p-1}(ac+bd)\right)\,\left|c,d,g^a x^{-b}\ (\mathrm{mod}\ p)\right\rangle. \tag{4.13}$$

We now compute the probability that the computation ends with the machine in state $|c,d,y\rangle$ with $y \equiv g^k\ (\mathrm{mod}\ p)$. This probability is the absolute value of the square of the sum over all ways the machine could produce this state, or

$$\left|\frac{1}{(p-1)^2}\sum_{\substack{a,b\\ a-rb\equiv k}}\exp\left(\tfrac{2\pi i}{p-1}(ac+bd)\right)\right|^2, \tag{4.14}$$

where the sum is over all a, b satisfying $a - rb \equiv k\ (\mathrm{mod}\ p-1)$. This condition arises from the fact that computational paths can only interfere when they give the same $y \equiv g^{a-rb} \equiv g^k\ (\mathrm{mod}\ p)$. We now substitute the equation $a \equiv k + rb\ (\mathrm{mod}\ p-1)$ in the above exponential. The above sum then reduces to

$$\left|\frac{1}{(p-1)^2}\sum_{b=0}^{p-2}\exp\left(\tfrac{2\pi i}{p-1}(kc+b(d+rc))\right)\right|^2. \tag{4.15}$$

However, if $d + rc \not\equiv 0\ (\mathrm{mod}\ p-1)$ the above sum is over a set of $(p-1)^{\mathrm{st}}$ roots of unity evenly spaced around the unit circle, and thus the probability is 0. If $d \equiv -rc$ the above sum is over the same root of unity $p-1$ times, giving $(p-1)e^{2\pi ikc/(p-1)}$, so the probability is $1/(p-1)^2$. We can check that these probabilities add up to one by counting that there are $(p-1)^2$ states $|c,-rc,y\rangle$ since there are $p-1$ choices of $c\ (\mathrm{mod}\ p-1)$ and $p-1$ choices of $y \not\equiv 0\ (\mathrm{mod}\ p)$.

Our computation thus produces a random $c\ (\mathrm{mod}\ p-1)$ and the corresponding $d \equiv -rc\ (\mathrm{mod}\ p-1)$. If c and $p-1$ are relatively prime, we can find r by division. Because we are choosing among all possible c's with equal probability, the chance that c and $p-1$ are relatively prime is $\phi(p-1)/(p-1)$, where ϕ is the Euler ϕ-function. It is easy to check that $\phi(p-1)/(p-1) > 1/\log(p)$. (Actually, from [17, Theorem 328], $\liminf \phi(p-1)/(p-1) \approx e^{-\gamma}/\log\log p$.) Thus we only need a number of experiments that is polynomial in $\log p$ to obtain r with high probability. In fact, we can find a set of c's such that at least one is relatively prime to every prime divisor of $p-1$ by repeating the experiment only an expected constant number of times. This would also give us enough information to obtain r.

5 A note on precision

The number of bits of precision needed in the amplitude of quantum mechanical computers could be a barrier to practicality. The generally accepted theoretical dividing line between feasible and infeasible is that polynomial precision (i.e., a number of bits logarithmic in the problem size) is feasible and that more is infeasible. This is because on a quantum computer the phase angle would need to be obtained through some physical device, and constructing such devices with better than polynomial precision seems unquestionably impractical. In fact, even polynomial precision may prove to be impractical; however, using this as the theoretical dividing line results in nice theoretical properties.

We thus need to show that the computations in the previous section need to use only polynomial precision in the amplitudes. The very act of writing down the expression $\exp(2\pi iac/(p-1))$ seems to imply that we need exponential precision, as this phase angle is exponentially precise. Fortunately, this is not the case. Consider the same matrix A_{p-1} with every term $\exp(2\pi iac/(p-1))$ replaced by $\exp(2\pi iac/(p-1) \pm \pi i/20)$. Each positive case, i.e., one resulting in $d \equiv -rc$, will still occur with nearly as large probability as before; instead of adding $p-1$ amplitudes which have exactly the same phase angle, we add $p-1$ amplitudes which have nearly the same phase angle, and thus the size of the sum will only be reduced by a constant factor. The algorithm will thus give a (c, d) with $d \equiv -rc$ with constant probability (instead of probability 1).

Recall that we obtain the matrix A_{p-1} by multiplying at most $\log p$ matrices A_{q_i}. Further, each entry in A_{p-1} is the product of at most $\log p$ terms. Suppose that each phase angle were off by at most $\epsilon/\log p$ in the A_{q_i}'s. Then in the product, each phase angle would be off by at most ϵ, which is enough to perform the computation with constant probability of success. A similar argument shows that the magnitude of the amplitudes in the A_{q_i} can be off by a polynomial fraction. Similar arguments hold for the general case of discrete log and for factoring to show that we need only polynomial precision for the amplitudes in these cases as well.

We still need to show how to construct A_{q_i} from constant size unitary matrices having limited precision. The arguments are much the same as above, but we will not give them in this paper because, in fact, Bennett *et al.* [4] have shown that it is sufficient to use polynomial precision for any computation on a quantum Turing machine to obtain the answer with high probability.

Since precision could easily be the limiting factor for practicality of quantum computation, it might be advisable to investigate how much precision is actually needed for quantum algorithms. Although Bernstein and Vazirani [4] show that the number of bits of precision needed is never more than the logarithm of the number of computational steps a quantum computer takes, in some algorithms it could conceivably require less. Interesting open questions are whether it is possible to do discrete logarithms or factoring with less than polynomial precision and whether some tradeoff between precision and time is possible.

6 Factoring

The algorithm for factoring is similar to the one for the general case of discrete log, only somewhat simpler. I present this algorithm before the general case of discrete log so as to give the three algorithms in this paper in order of increasing complexity. Readers interested in discrete log may skip to the next section.

Instead of giving a quantum computer algorithm to factor n, we will give a quantum computer algorithm for finding the order of an element x in the multiplicative group $(\bmod n)$; that is, the least integer r such that $x^r \equiv 1 \pmod n$. There is a randomized reduction from factoring to the order of an element [24].

To factor an odd number n, given a method for computing the order of an element, we choose a random x, find the order r_x of x, and compute $\gcd(x^{r_x/2} - 1, n)$. This fails to give a non-trivial divisor of n only if r_x is odd or if $x^{r_x/2} \equiv -1 \pmod n$. Using this criterion, it can be shown that the algorithm finds a factor of n with probability at least $1-1/2^{k-1}$, where k is the number of distinct odd prime factors of n. This scheme will thus work as long as n is not a prime power; however, factoring prime powers can be done efficiently with classical methods.

Given x and n, to find r such that $x^r \equiv 1 \pmod n$, we do the following. First, we find a smooth q with $2n^2 \le q < 4n^2$. Next, we put our machine in the uniform superposition of states representing numbers $a \pmod q$. This leaves our machine in state

$$\frac{1}{q^{1/2}} \sum_{a=0}^{q-1} |a\rangle\,. \tag{6.16}$$

As in the algorithm for discrete log, we will not write n, x, or q in the state of our machine, because we never change these values.

Next, we compute $x^a \pmod n$. Since we keep x and a on the tape, this can be done reversibly. This leaves our machine in the state

$$\frac{1}{q^{1/2}} \sum_{a=0}^{q-1} \left|a, x^a \pmod n\right\rangle . \tag{6.17}$$

We then perform our Fourier transform A_q mapping $a \to c$ with amplitude $\frac{1}{q^{1/2}}\exp(2\pi i a c/q)$. This leaves our machine in state

$$\frac{1}{q} \sum_{a=0}^{q-1} \exp(2\pi i a c/q) \left|c, x^a \pmod n\right\rangle . \tag{6.18}$$

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

$$\left| \frac{1}{q} \sum_{a:\, x^a \equiv x^k} \exp(2\pi i a c/q) \right|^2 . \tag{6.19}$$

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 equivalently over all a satisfying $a \equiv k \pmod r$. Writing $a = br + k$, we find that the above probability is

$$\left| \frac{1}{q} \sum_{b=0}^{\lfloor (q-k-1)/r \rfloor} \exp(2\pi i (br+k)c/q) \right|^2 . \tag{6.20}$$

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

$$\left| \frac{1}{q} \sum_{b=0}^{\lfloor (q-k-1)/r \rfloor} \exp(2\pi i b \{rc\}_q/q) \right|^2 . \tag{6.21}$$

We will now show that if $\{rc\}_q$ is small enough, all the amplitudes in this sum will be in nearly the same direction, giving a large probability. If $\{rc\}_q$ is small with respect to q, we can use the change of variables $t = b/q$ and approximate this sum with the integral

$$\left| \int_0^{\frac{1}{q}\lfloor (q-k-1)/r \rfloor} \exp(2\pi i \{rc\}_q t) dt \right|^2 . \tag{6.22}$$

If $|\{rc\}_q| \le r/2$, this quantity can be shown to be asymptotically bounded below by $4/(\pi^2 r^2)$, and thus at least $1/3r^2$. The exact probabilities as given by Equation (6.21) for an example case are plotted in Figure 1.

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

$$\frac{-r}{2} \le \{rc\}_q \le \frac{r}{2}, \tag{6.23}$$

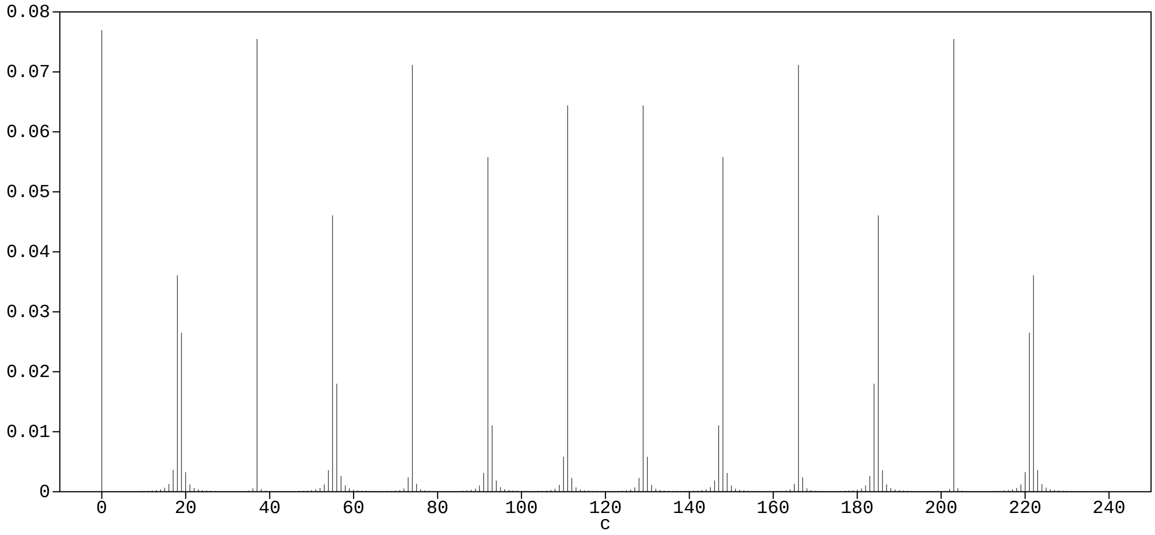

Figure 1: The probability of observing values of c between 0 and 239, given $q = 240$ and $r = 13$, using the quantum algorithm described for factoring. With high probability the observed value of c is near an integer multiple of 240/13.

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

$$\frac{-r}{2} \leq rc - dq \leq \frac{r}{2}. \tag{6.24}$$

Dividing by rq and rearranging the terms gives

$$\left| \frac{c}{q} - \frac{d}{r} \right| \leq \frac{1}{2q}. \tag{6.25}$$

We know c and q. Because $q \geq 2n^2$, 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 [17, Chapter X].

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 $\left|c, x^k \pmod{n}\right\rangle$ which enable us to compute r in this way. There are $\phi(r)$ possible values for d relatively prime to r, where ϕ is Euler's ϕ function. Each of these fractions d/r is close to one fraction c/q with $|c/q - d/r| \leq 1/2q$. There are also r possible values for x^k, since r is the order of x. Thus, there are $r\phi(r)$ states $\left|c, x^k \pmod{n}\right\rangle$ 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 $\phi(r)/r > k/\log\log r$ for some fixed k [17, Theorem 328], this shows that we find r at least a $k/\log\log r$ fraction of the time, so by repeating this experiment only $O(\log\log r)$ times, we are assured of a high probability of success.

Note that in the algorithm for order, 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 $f^{(r)}(a) = a$.

7 Discrete log: the general case

For the general case, we first find a smooth number q such that q is close to p, i.e., with $p \leq q \leq 2p$ (see Lemma 3.2).

Next, we do the same thing as in the easy case, that is, we choose a and b uniformly $\pmod{p-1}$, and then compute $g^a x^{-b} \pmod{p}$. This leaves our machine in the state

$$\frac{1}{p-1} \sum_{a=0}^{p-2} \sum_{b=0}^{p-2} \left|a, b, g^a x^{-b} \pmod{p}\right\rangle. \tag{7.26}$$

As before, we use the Fourier transform A_q to send $a \rightarrow c$ and $b \rightarrow d \pmod{q}$, with amplitude $\frac{1}{q}\exp(2\pi i(ac + bd)/q)$, giving us the state

$$\frac{1}{(p-1)q} \sum_{a,b=0}^{p-2} \sum_{c,d=0}^{q-1} \exp\left(\tfrac{2\pi i}{q}(ac+bd)\right) \left|c, d, g^a x^{-b} \pmod{p}\right\rangle. \tag{7.27}$$

Note that we now have two moduli to deal with, $p-1$ and q. While this makes keeping track of things more confusing, we will still be able to obtain r using a algorithm similar to the easy case. The probability of observing a state $|c, d, y\rangle$ with $y \equiv g^k \pmod p$ is, almost as before,

$$\left| \frac{1}{(p-1)q} \sum_{\substack{a,b \\ a-rb\equiv k}} \exp\left(\tfrac{2\pi i}{q}(ac+bd)\right) \right|^2 \qquad (7.28)$$

where the sum is over all (a, b) such that $a - rb \equiv k \pmod{p-1}$. We now use the relation

$$a = br + k - (p-1)\left\lfloor \tfrac{br+k}{p-1} \right\rfloor \qquad (7.29)$$

and substitute in the above expression to obtain the amplitude

$$\tfrac{1}{(p-1)q} \sum_{b=0}^{p-2} \exp\left(\tfrac{2\pi i}{q}\left(brc + kc + bd - c(p-1)\left\lfloor \tfrac{br+k}{p-1} \right\rfloor\right)\right). \qquad (7.30)$$

The absolute value of the square of this amplitude is the probability of observing the state $\left|c, d, g^k \pmod p\right\rangle$. We will now analyze this expression. 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

$$\frac{1}{(p-1)q} \sum_{b=0}^{p-2} \exp\left(\tfrac{2\pi i}{q} U\right) \exp\left(\tfrac{2\pi i}{q} V\right), \qquad (7.31)$$

where

$$\begin{aligned} U &= bT, \\ T &= rc + d - \tfrac{r}{p-1}\{c(p-1)\}_q, \end{aligned} \qquad (7.32)$$

and

$$V = \left(\tfrac{br}{p-1} - \left\lfloor \tfrac{br+k}{p-1} \right\rfloor\right) \{c(p-1)\}_q. \qquad (7.33)$$

Here by $\{z\}_q$ we mean the residue of $z \pmod q$ with $-q/2 < \{z\}_q \le q/2$. We will show that if we get enough "good" outputs, then we still can deduce r, and that furthermore, the chance of getting a "good" output is constant. The idea is that if

$$|\{T\}_q| = \left| rc + d - \tfrac{r}{p-1}\{c(p-1)\}_q - jq \right| \le \frac{1}{2}, \qquad (7.34)$$

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 Eq. (7.31) only varies over at most half of the unit circle. Further, if

$$|\{c(p-1)\}_q| \le q/20, \qquad (7.35)$$

then $|V|$ is always at most $q/20$, so the phase of the second exponential term in Eq. (7.31) never is farther than $\exp(\pi i/10)$ from 1. By combining these two observations, 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 (7.34) 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 (7.34) and (7.35). We know that as b ranges from 0 to $p-2$, the phase of $\exp(2\pi iU/q)$ ranges from 0 to $2\pi iW$ where

$$W = \frac{p-2}{q}\left(rc + d - \tfrac{r}{p-1}\{c(p-1)\}_q - jq\right) \qquad (7.36)$$

and j is as in Eq. (7.34). Thus, the component of the amplitude of the first exponential in Eq. (7.31) in the direction

$$\exp(\pi iW) \qquad (7.37)$$

is at least $\cos(2\pi\,|W/2 - Wb/(p-2)|)$. Now, by Condition (7.35), the phase can vary by at most $\pi i/10$ due to the second exponential $\exp(2\pi iV/q)$. Applying this variation in the manner that minimizes the component in the direction (7.37), we get that the component in this direction is at least $\cos(2\pi\,|W/2 - Wb/(p-2)| + \pi/10)$. Since $p < q$, and from Condition (7.34), $|W| \le 1/2$, putting everything together, the probability of arriving at a state $|c, d, y\rangle$ that satisfies both Condition (7.34) and (7.35) is at least

$$\left(\frac{1}{q}\frac{2}{\pi}\int_{\pi/10}^{7\pi/20} \cos t\; dt\right)^2, \qquad (7.38)$$

or at least $.137/q^2$.

We will now count the number of pairs (c, d) satisfying Conditions (7.34) and (7.35). The number of pairs (c, d) such that (7.34) holds is exactly the number of possible c's, since for every c there is exactly one d such that (7.34) holds (round off the fraction to the nearest integer to obtain this d). The number of c's for which (7.35) holds is approximately $q/10$. Thus, there are $q/10$ pairs (c, d) satisfying both conditions. Multiplying by $p-1$, which is the number of possible y's, gives approximately $pq/10$ states $|c, d, y\rangle$. Combining this calculation with the lower bound on the probability of each good state gives us that the probability of obtaining any good state is at least $p/80q$, or at least $1/160$ (since $q < 2p$).

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

$$-\frac{1}{2q} \le \frac{d}{q} + \frac{r}{q}\left(c - \frac{\{c(p-1)\}_q}{p-1}\right) \le \frac{1}{2q} \pmod 1, \qquad (7.39)$$

where this equation was obtained from Condition (7.34) 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 $(\mathrm{mod}\ p-1)$ by

$$c' = \frac{c(p-1) - \{c(p-1)\}_q}{q} \tag{7.40}$$

to find a candidate r. To show that this experiment need only be repeated a polynomial number of times to find the correct r requires only a few more details. The problem is again that we cannot divide by a number which is not relatively prime to $p-1$.

For the general case of 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-tenth of them. This additional difficulty makes the next step harder than the corresponding step in the two previous algorithms. 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 find this remainder for primes larger than 20, but with a little extra work we will still be able to recover r.

What we have is that each good (c, d) pair is generated with probability at least $.137p/q > 1/16q$, and that at least a tenth of the possible c's are in a good (c, d) pair. From Eq. (7.40), it follows that these c's are mapped from c/q to $c'/(p-1)$ by rounding to the nearest integer 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 the algorithm, we can just ignore the prime powers under 20; if we know r modulo all prime powers over 20, we can try all possible residues for primes under 20 with only a (large) constant factor increase in running time. Because at least one tenth of the c's were in a good (c, d) pair, at least one tenth 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 $10/p_i^{\alpha_i}$. If we have t good c''s, the probability of having a prime power over 20 that divides all of them is therefore at most

$$\sum_{\substack{p_i^{\alpha_i} > 20 \\ p_i^{\alpha_i} | p-1}} \left(\frac{10}{p_i^{\alpha_i}}\right)^t, \tag{7.41}$$

where the sum is over all prime powers greater than 20 that divide $p-1$. This sum (over all integers > 20) converges for $t = 2$, and goes down by at least a factor of 2 for each further increase of t by 1; thus for some large constant t it is less than $1/2$.

Recall that each good c' is obtained with probability at least $1/16q$ from any experiment. Since there are $q/10$ good c''s, after $160t$ 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. For each prime p_i less than 20, we thus have at most 20 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 possibilites for residues modulo powers of primes less than 20: 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.

This algorithm does not use very many properties of Z_p, so we can use the same algorithm to find discrete logarithms over other fields such as Z_{p^α}. What we need is that we know the order of the generator, and that we can multiply and take inverses of elements in polynomial time.

If one were to actually program this algorithm (which must wait until a quantum computer is built) there are many ways in which the efficiency could be increased over the efficiency shown in this paper.

Acknowledgements

I would like to thank Jeff Lagarias for finding and fixing a critical bug in the first version of the discrete log algorithm. I would also like to thank him, Charles Bennett, Gilles Brassard, Andrew Odlyzko, Dan Simon, Umesh Vazirani, as well as other 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

1. P. Benioff, "Quantum mechanical Hamiltonian models of Turing machines," *J. Stat. Phys.* Vol. 29, pp. 515–546 (1982).
2. P. Benioff, "Quantum mechanical Hamiltonian models of Turing machines that dissipate no energy," *Phys. Rev. Lett.* Vol. 48, pp. 1581–1585 (1982).
3. C. H. Bennett, "Logical reversibility of computation," *IBM J. Res. Develop.* Vol. 17, pp. 525–532 (1973).
4. C. H. Bennett, E. Bernstein, G. Brassard and U. Vazirani, "Strengths and weaknesses of quantum computing," manuscript (1994). Currently available through the World-Wide Web at URL http://vesta.physics.ucla.edu:7777/.

5. E. Bernstein and U. Vazirani, "Quantum complexity theory," in *Proc. 25th ACM Symp. on Theory of Computating,* pp. 11–20 (1993).
6. A. Berthiaume and G. Brassard, "The quantum challenge to structural complexity theory," in *Proc. 7th Conf. on Structure in Complexity Theory,* IEEE Computer Society Press, pp. 132–137 (1992).
7. A. Berthiaume and G. Brassard, "Oracle quantum computing," in *Proc. Workshop on Physics and Computation,* pp. 195–199, IEEE Computer Society Press (1992).
8. D. Coppersmith, "An approximate Fourier transform useful in quantum factoring," *IBM Research Report RC 19642* (1994).
9. D. Deutsch, "Quantum theory, the Church–Turing principle and the universal quantum computer," *Proc. Roy. Soc. Lond. Ser. A,* Vol. 400, pp. 96–117 (1985).
10. D. Deutsch, "Quantum computational networks," *Proc. Roy. Soc. Lond. Ser. A,* Vol. 425, pp. 73–90 (1989).
11. D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proc. Roy. Soc. Lond. Ser. A* Vol. 439, pp. 553–558 (1992).
12. D. P. DiVincenzo, "Two-bit gates are universal for quantum computation," *Phys. Rev. A,* Vol. 51, pp. 1015–1022 (1995).
13. R. Feynman, "Simulating physics with computers," *International Journal of Theoretical Physics,* Vol. 21, No. 6/7, pp. 467–488 (1982).
14. R. Feynman, "Quantum mechanical computers," *Foundations of Physics,* Vol. 16, pp. 507–531 (1986). (Originally appeared in *Optics News*, February 1985.)
15. L. Fortnow and M. Sipser, "Are there interactive protocols for co-NP languages?" *Inform. Proc. Lett.* Vol. 28, pp. 249–251 (1988).
16. D. M. Gordon, "Discrete logarithms in GF(p) using the number field sieve," *SIAM J. Discrete Math.* Vol. 6, pp. 124–139 (1993).
17. G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers, Fifth Edition,* Oxford University Press, New York (1979).
18. R. Landauer, "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, to appear.
19. Y. Lecerf, "Machines de Turing réversibles. Récursive insolubilité en $n \in N$ de l'équation $u = \theta^n u$, où θ est un isomorphisme de codes," *Comptes Rendues de l'Académie Française des Sciences,* Vol. 257, pp. 2597-2600 (1963).
20. A. K. Lenstra and H. W. Lenstra, Jr., eds., *The Development of the Number Field Sieve*, Lecture Notes in Mathematics No. 1554, Springer-Verlag (1993); this book contains reprints of the articles that were critical in the development of the fastest known factoring algorithm.
21. H. W. Lenstra, Jr. and C. Pomerance, "A rigorous time bound for factoring integers," *J. Amer. Math. Soc.* Vol. 5, pp. 483–516 (1992).
22. S. Lloyd, "A potentially realizable quantum computer," *Science,* Vol. 261, pp. 1569–1571 (1993).
23. S. Lloyd, "Envisioning a quantum supercomputer," *Science,* Vol. 263, p. 695 (1994).
24. G. L. Miller, "Riemann's hypothesis and tests for primality," *J. Comp. Sys. Sci.* Vol. 13, pp. 300–317 (1976).
25. S. Pohlig and M. Hellman, "An improved algorithm for computing discrete logarithms over GF(p) and its cryptographic significance," *IEEE Trans. Information Theory,* Vol. 24, pp. 106–110 (1978).
26. C. Pomerance, "Fast, rigorous factorization and discrete logarithm algorithms," in *Discrete Algorithms and Complexity (Proc. Japan-US Joint Seminar),* pp. 119-143, Academic Press (1986).
27. R. L. Rivest, A. Shamir, and L. Adleman "A method of obtaining digital signatures and public-key cryptosystems," *Communications ACM,* Vol. 21, No. 2, pp. 120–126 (1978).
28. A. Shamir, "IP = PSPACE," in *Proc. 31th Ann. Symp. on Foundations of Computer Science,* pp. 11–15, IEEE Computer Society Press (1990).
29. D. Simon, "On the power of quantum computation," in *Proc. 35th Ann. Symp. on Foundations of Computer Science,* pp. 116–123, IEEE Computer Society Press (1994).
30. W. G. Teich, K. Obermayer, and G. Mahler, "Structural basis of multistationary quantum systems II: Effective few-particle dynamics," *Phys. Rev. B*, Vol. 37, pp. 8111–8121 (1988).
31. T. Toffoli, "Reversible computing," in *Automata, Languages and Programming, Seventh Colloq.,* Lecture Notes in Computer Science No. 84 (J. W. De Bakker and J. van Leeuwen, eds.) pp. 632-644, Springer-Verlag (1980).
32. W. G. Unruh, "Maintaining coherence in quantum computers," *Phys. Rev. A,* Vol. 51, pp. 992–997 (1995).
33. A. Yao, "Quantum circuit complexity," in *Proc. 34th Ann. Symp. on Foundations of Computer Science,* pp. 352–361, IEEE Computer Society Press (1993).