Chapter 4
Shor's Algorithm for Integer Factorization

The general number field sieve is the most efficient classical algorithm known for factorizing integers larger than 10^{100}. The second fastest method for integer factorization is the quadratic sieve, which is most recommended for factorizing numbers smaller than 10^{100}. Other important classical algorithms are the Lenstra elliptic curve method [14] and the Pollard's rho method [20]. The time complexity of each of those algorithms is exponential in the number of bits of the input. The time complexity of the general field sieve algorithm, for instance, is of the form

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\ln N)^{1/3}(\ln\ln N)^{2/3}\right), \tag{4.1}$$

where N is the number to be factorized. In fact, the difficulty of factorizing large integers on classical computers is at the heart of many cryptography protocols widely used nowadays.[1] Perhaps that is the reason why Shor's algorithm is still one of the most celebrated breakthroughs in quantum algorithm design. The original presentations of Shor's algorithm are references [21] and [22]. Shor's algorithm finds the prime factors of a composite number N in polynomial time on the number of input bits. A key ingredient of Shor's algorithm is the Quantum Fourier Transform, which has an exponential speed-up over the classical Fast Fourier Transform [10].

[1]There are several good references on the subject of integer factorization and classical cryptography. The reader may refer to Hoffstein, Pipher and Silverman [9], or to Crandall and Pomerance [5], or to Wagstaff [24], for instance.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2019

F. de Lima Marquezino et al., *A Primer on Quantum Computing*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-3-030-19066-8_4

4.1 A Reduction of Integer Factorization to Order Finding

Think of a large number such as one with 300 digits in decimal notation. Though N is large, the number of qubits necessary to store it is small. In general, $\log_2 N$ is not an integer, so let us define

$$n = \lceil \log_2 N \rceil. \tag{4.2}$$

A quantum computer with n qubits can store N or any other positive integer less than N. With a little thought, we see that the number of prime factors of N is at most n. If both the number of qubits and the number of factors are less than or equal to n, then it is natural to ask if there is a quantum algorithm that factors N in a number of steps which is polynomial in n. In fact, this question has a positive answer, and in order to describe it, we first need to understand how the problem of factorizing an integer N can be efficiently reduced to the problem of finding the *order* of a number.

We should start by choosing a random integer x less than N. We can efficiently[2] find the greatest common divisor—or GCD, for short—between x and N. If x and N have common factors, then GCD(x, N) gives a factor of N, and in this lucky case the problem is solved! Therefore, it suffices to investigate the case when x is coprime to N, which is much harder.

The order of x modulo N is defined as the least positive integer r such that

$$x^r \equiv 1 \mod N. \tag{4.3}$$

The notation $a \equiv b \mod N$ means that both a and b yields the same remainder when divided by N. If r is even, we can define y by

$$y \equiv x^{r/2} \mod N, \tag{4.4}$$

with $0 \leq y < N$. Since r depends on x, if r is not even, we have to start the process all over again by randomly choosing another x. Notice that y satisfies $y^2 \equiv 1 \mod N$, or equivalently $(y-1)(y+1) \equiv 0 \mod N$, which means that N divides $(y-1)(y+1)$. If $1 < y < N-1$, the factors $y-1$ and $y+1$ satisfy $0 < y-1 < y+1 < N$, therefore N cannot divide $y-1$ nor $y+1$ separately. The only alternative is that both $y-1$ and $y+1$ have factors of N, and they yield N by multiplication. Hence, GCD$(y-1, N)$ and GCD$(y+1, N)$ yield non trivial factors of N. If N still has remaining factors, they can be calculated by applying the algorithm recursively.

Consider $N = 21$ as an example. The sequence of equivalences

[2] By using Euclid's algorithm, for instance. The ancient greek mathematician Euclid described this algorithm in his *Elements*, c. 300 BC.

$$\begin{aligned} 2^4 &\equiv 16 \mod 21 \\ 2^5 &\equiv 11 \mod 21 \\ 2^6 &\equiv 1 \mod 21 \end{aligned} \tag{4.5}$$

show that the order of two modulo 21 is $r = 6$. Therefore, $y \equiv 2^3 \equiv 8 \mod 21$. The prime factors of 21 are given by $y - 1$, which yields the factor seven, and by $y + 1$, which yields the factor three. For a more detailed study of number theory, the reader may look for Gathen and Gerhard's book [8].

In summary, if we pick up at random a positive integer x less than N and calculate GCD(x, N), either we have a factor of N or we learn that x is co-prime to N. In the latter case, if x satisfies the conditions (1) that its order r is even, and (2) that $0 < y - 1 < y + 1 < N$, then GCD$(y - 1, N)$ and GCD$(y + 1, N)$ yield factors of N. If one of the conditions is not true, we start over until finding a proper candidate x. The method would not be useful if these assumptions were too restrictive, but fortunately that is not the case. The method systematically fails if N is a power of some odd prime, but an alternative efficient classical algorithm for this case is known. If N is even, we can keep dividing by two until the result turns out to be odd. It remains to apply the method for odd composite integers that are not a power of some prime number. It is cumbersome to prove that the probability of finding x coprime to N satisfying the conditions (1) and (2) is high—in fact, this probability is $1 - 1/2^{k-1}$, where k is the number of prime factors of N.[3] The worst case is when N has only two prime factors, then the probability is greater than or equal to $1/2$.

At first sight, it seems that we have just described an efficient algorithm to find a factor of N. That is not yet true, since it is not known an efficient *classical* algorithm to calculate the order of an integer x modulo N. On the other hand, there is an efficient *quantum* algorithm, which is precisely Shor's algorithm.[4] Let us describe it.

4.2 Quantum Algorithm to Calculate the Order

Consider the circuit of Fig. 4.1. It calculates the order r of the positive integer x less than N, coprime to N. The subroutine V_x is given by the unitary linear operator

$$V_x \, |j\rangle \, |k\rangle = |j\rangle \left| k \oplus (x^j \mod N) \right\rangle, \tag{4.6}$$

[3] The proof can be found in Appendix B of reference [7].

[4] The problem of finding order can be generalized as the hidden subgroup problem [1, 11]. Several quantum algorithms can be described under the same framework of the hidden subgroup problem [2, 17].

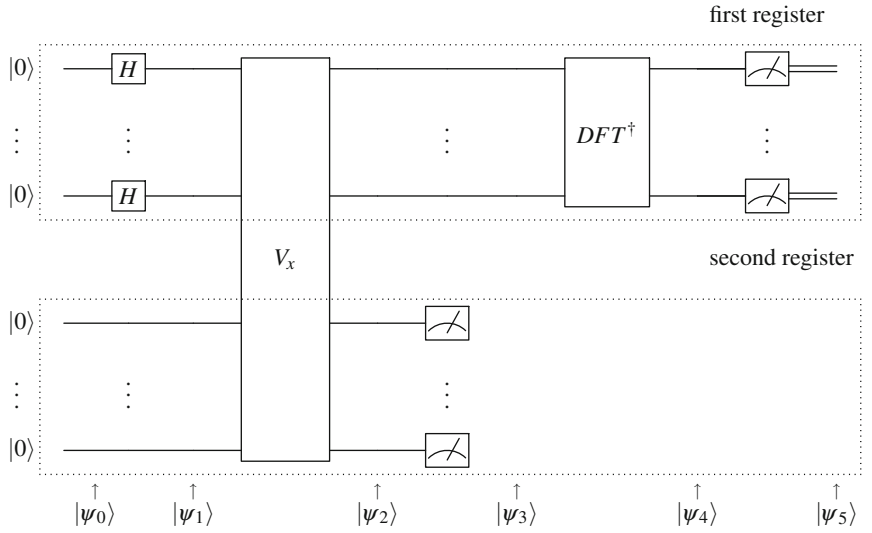

Fig. 4.1 Quantum circuit for finding the order of the positive integer x modulo N

where $|j\rangle$ and $|k\rangle$ are the states of the first and second registers, respectively, and $\oplus$ is the bitwise binary sum.[5] The subroutine DFT is the Discrete Fourier Transform operator, which will be described ahead.

The first register has t qubits, where t is generally chosen such that $N^2 \leq 2^t < 2N^2$, for reasons that will become clear later on. As an exception, if the order r is a power of 2, then it is enough to take $t = \log_2 N = n$. In this section, we consider this very special case and leave the general case for Sect. 4.4. We will keep the variable t in order to generalize the discussion later on. The second register has n qubits.

It is interesting to notice that, on a very important experiment with nuclear magnetic resonance, Vandersypen et al. bypassed part of Shor's algorithm and managed to factorize $N = 15$ on a quantum computer with only seven qubits [23]. A more recent implementation of Shor's algorithm performed by Monz et al. has the advantage of being scalable [18]. A key ingredient in that result was Kitaev's version of the Quantum Fourier Transform [12].

The states of the quantum computer are indicated by the vectors $|\psi_0\rangle$ to $|\psi_5\rangle$ in Fig. 4.1. The initial state is

$$|\psi_0\rangle = \underbrace{|00\ldots0\rangle}_{t \text{ times}} \underbrace{|00\ldots0\rangle}_{n \text{ times}}. \tag{4.7}$$

[5] At this point, k and x^j modulo N must be converted into base-2 notation in order to execute the binary sum.

Recall that the application of the Hadamard operator

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4.8}$$

on each qubit of the first register yields

$$|\psi_1\rangle = \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle\, |0\rangle\,. \tag{4.9}$$

The first register is then in a superposition of all states of the computational basis with equal amplitudes given by $1/\sqrt{2^t}$. Now, notice that when we apply V_x to $|\psi_1\rangle$, we yield the state

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} V_x\, |j\rangle\, |0\rangle \\ &= \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle\, \left|x^j\right\rangle. \end{aligned} \tag{4.10}$$

The state $|\psi_2\rangle$ is remarkable. Because V_x is linear, it acts on all basis states $|j\rangle\, |0\rangle$ for 2^t values of j, so this generates all powers of x simultaneously. This feature is the *quantum parallelism*, the same property observed in Grover's algorithm. Some of these powers are equal to one, which correspond to the states $|0\rangle\, |1\rangle\,, |r\rangle\, |1\rangle\,, |2r\rangle\, |1\rangle\,, \ldots, \left|\left(\frac{2^t}{r} - 1\right) r\right\rangle |1\rangle$. This explains the choice for V_x given by Eq. (4.6). Classically, one would calculate successively x^j, for j starting from two until reaching $j = r$. Quantumly, on the other hand, one can calculate all powers of x with just one application of V_x. At the quantum level, the values of j that yield $x^j \equiv 1 \mod N$ are "known". However, this quantum information is not fully available at the classical level. A classical information of a quantum state is obtained by practical measurements and, at this point, it does not help if we measure the first register, since all states in the superposition given by Eq. (4.10) have equal amplitudes. The first part of the strategy to find r is to observe that the first register of the states $|0\rangle\, |1\rangle\,, |r\rangle\, |1\rangle\,, |2r\rangle\, |1\rangle\,, \ldots, \left|2^t - r\right\rangle |1\rangle$ is periodic. Hence, the information we want is a period!

In order to simplify the calculation, let us measure the second register. Before doing this, we will rewrite $|\psi_2\rangle$ collecting equal terms in the second register. Since x^j is a periodic function with period r, substitute j by $ar + b$ in Eq. (4.10), where $0 \leq a \leq (2^t/r) - 1$ and $0 \leq b \leq r - 1$. Recall that we are supposing that $t = n$ and that r is a power of 2, therefore r divides 2^t. Thus, Eq. (4.10) is converted to

$$|\psi_2\rangle = \frac{1}{\sqrt{2^t}} \sum_{b=0}^{r-1} \left(\sum_{a=0}^{\frac{2^t}{r}-1} |ar+b\rangle \right) \left|x^b\right\rangle. \tag{4.11}$$

In the second register, we have replaced x^{ar+b} by x^b, since $x^r \equiv 1 \mod N$. Now, the second register is measured.[6] Any output $x^0, x^1, \ldots, x^{r-1}$ can be obtained with equal probability. Suppose that the result is x^{b_0}. The state of the quantum computer is now

$$|\psi_3\rangle = \sqrt{\frac{r}{2^t}} \sum_{a=0}^{\frac{2^t}{r}-1} |ar+b_0\rangle \left|x^{b_0}\right\rangle. \tag{4.12}$$

Notice that after the measurement, the constant is renormalized to $\sqrt{r/2^t}$, since there are $2^t/r$ terms[7] in the sum in Eq. (4.12). Figure 4.2 shows the probability of obtaining the states of the computational basis upon measuring the first register. The probabilities form a periodic function with period r. Their values are zero except for the states $|b_0\rangle, |r+b_0\rangle, |2r+b_0\rangle, \ldots, \left|2^t - r + b_0\right\rangle$.

How can one find out the period of a function efficiently? The answer is in the Fourier transform. The Fourier transform of a periodic function with period r is a new periodic function with period proportional to $1/r$. This makes a difference

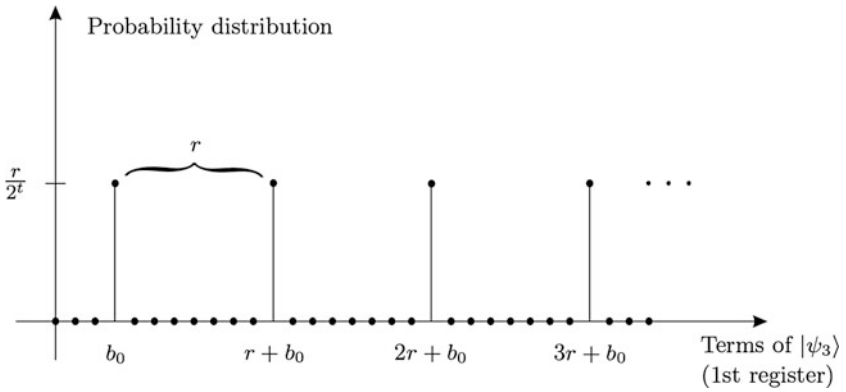

Fig. 4.2 Probability distribution of $|\psi_3\rangle$ measured in the computational basis (for the case $b_0 = 3$ and $r = 8$). The horizontal axis has 2^t points. The number of peaks is $2^t/r$ and the period is r

[6] In Nielsen and Chuang's book, there is a proof that measurements can always be moved from an intermediate stage of a quantum circuit to the end of the circuit without changing the results [19], so the intermediate measurement in Shor's algorithm is not really necessary—although it simplifies the remaining calculations.

[7] Or equivalently, there are r terms inside the parenthesis in Eq. (4.11).

for finding r. The Fourier transform is the second and last part of the strategy. The whole method relies on an efficient quantum algorithm for calculating the Fourier transform, which is not available classically. In Sect. 4.5, we show that the Fourier transform is calculated efficiently in a quantum computer.

4.3 The Quantum Discrete Fourier Transform

The classical algorithm of Fast Fourier Transform was developed by Cooley and Tukey [4] and an excellent exposition of it can be found in Donald Knuth's famous *The Art of Computer Programming* [13]. The quantum algorithm for Discrete Fourier Transform can be derived from the classical algorithm, as described in Ref. [16].

The Fourier transform of the function $F : \{0, \ldots, N-1\} \to \mathbb{C}$ is a new function $\tilde{F} : \{0, \ldots, N-1\} \to \mathbb{C}$ defined as

$$\tilde{F}(k) = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N} F(j). \tag{4.13}$$

We can apply the Fourier transform either on a function or on the states of the computational basis. The Fourier transform applied to the state $|k\rangle$ of the computational basis $\{|0\rangle, \ldots, |N-1\rangle\}$ is

$$\begin{aligned} \mathrm{DFT}\,|k\rangle &= |\psi_k\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N}\,|j\rangle\,, \end{aligned} \tag{4.14}$$

where the set $\{|\psi_k\rangle : k = 0, \ldots, N-1\}$ forms a new orthonormal basis. The Fourier transform is a unitary linear operator. Hence, if we know how it acts on the states of the computational basis, we also know how it acts on an arbitrary state

$$|\psi\rangle = \sum_{a=0}^{N-1} F(a)\,|a\rangle\,. \tag{4.15}$$

The Fourier transform of $|\psi\rangle$ can be performed indistinctly using either Eq. (4.13) or (4.14). We will use the latter.

To prove that $\{|\psi_k\rangle : k = 0, \ldots, N-1\}$ is an orthonormal basis, i.e.,

$$\langle \psi_{k'} | \psi_k \rangle = \delta_{k'k}, \tag{4.16}$$

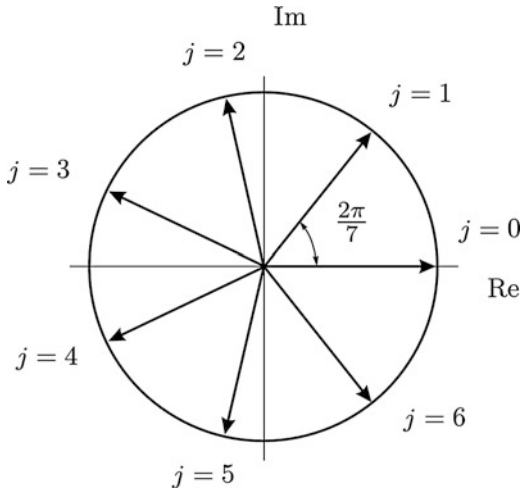

Fig. 4.3 Vectors $e^{2\pi ij/7}$, for $j = 0, \ldots, 6$, in the complex plane. Their sum is zero by symmetry arguments. This is an example of Eq. (4.17) for $N = 7, k = 1$

we can use the identity

$$\frac{1}{N}\sum_{j=0}^{N-1} e^{2\pi ijk/N} = \begin{cases} 1, & \text{if } k \text{ is a multiple of } N \\ 0, & \text{otherwise,} \end{cases} \tag{4.17}$$

which is useful in the context of Fourier transforms. It is not difficult to prove that Eq. (4.17) is true. If k is a multiple of N, then $e^{2\pi ijk/N} = 1$ and the first case of the identity follows. If k is not a multiple of N, Eq. (4.17) is true even if N is not a power of two. In Fig. 4.3, we have each term $e^{2\pi ijk/N}$, for the case where $k = 1$ and $N = 7$, as vectors in the complex plane. Note that the sum of vectors must be zero by a symmetry argument: the distribution of vectors is isotropic. Usually it is said that the interference is destructive in this case.

Using this identity, we can define the inverse Fourier transform, which is similar to Eq. (4.14), just with a minus sign on the exponent. Note that $\text{DFT}^{-1} = \text{DFT}^{\dagger}$, since DFT is a unitary operator.

We will present the details of a quantum circuit to perform the Fourier transform in Sect. 4.5. Now we will continue the calculation process of the circuit of Fig. 4.1. We are ready to find out $|\psi_4\rangle$, the next state of the quantum computer. Applying the inverse Fourier transform on the first register, using Eq. (4.14) and the linearity of $\text{DFT}^{\dagger}$, we obtain

$$\begin{aligned} |\psi_4\rangle &= \text{DFT}^{\dagger}\,|\psi_3\rangle \\ &= \sqrt{\frac{r}{2^t}}\sum_{a=0}^{\frac{2^t}{r}-1}\left(\frac{1}{\sqrt{2^t}}\sum_{j=0}^{2^t-1} e^{-2\pi ij(ar+b_0)/2^t}\,|j\rangle\right)\left|x^{b_0}\right\rangle. \end{aligned} \tag{4.18}$$

Inverting the summation order, we have

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \left(\sum_{j=0}^{2^t-1} \left[\frac{1}{2^t/r} \sum_{a=0}^{\frac{2^t}{r}-1} e^{\frac{-2\pi i j a}{2^t/r}} \right] e^{-2\pi i j b_0/2^t} |j\rangle \right) \left| x^{b_0} \right\rangle. \tag{4.19}$$

Using Eq. (4.17), we see that the expression in square brackets is zero except when $j = k2^t/r$, with $k = 0, \ldots, r-1$. When j takes such values, the expression in the square brackets is equal to one. Thus, we have

$$|\psi_4\rangle = \frac{1}{\sqrt{r}} \left(\sum_{k=0}^{r-1} e^{-2\pi i \frac{k}{r} b_0} \left| \frac{k2^t}{r} \right\rangle \right) \left| x^{b_0} \right\rangle. \tag{4.20}$$

In order to find r, the expression for $|\psi_4\rangle$ in Eq. (4.20), has two advantages over the expression for $|\psi_3\rangle$, in Eq. (4.12). The first advantage is that r is in the denominator of the ket label, and the second advantage is that the random parameter b_0 moved from the ket label to the exponent occupying now a harmless place. In Fig. 4.4, we have the probability distribution of $|\psi_4\rangle$ measured in the computational basis. Measuring the first register, we get the value $k_0 2^t/r$, where k_0 can be any number between 0 and $r-1$ with equal probability—see the peaks in Fig. 4.4. If we obtain $k_0 = 0$, we have no clue at all about r, and the algorithm must be run again. If $k_0 \neq 0$, we divide $k_0 2^t/r$ by 2^t, obtaining k_0/r. Neither k_0 nor r are known. If k_0 is coprime to r, we simply select the denominator.

If k_0 and r have a common factor, the denominator of the reduced fraction k_0/r is a factor of r but not r itself. Suppose that the denominator is r_1. Let $r = r_1 r_2$. Now the goal is to find r_2, which is the order of x^{r_1}. We run again the quantum part of the algorithm to find the order of x^{r_1}. If we find r_2 this time, the algorithm

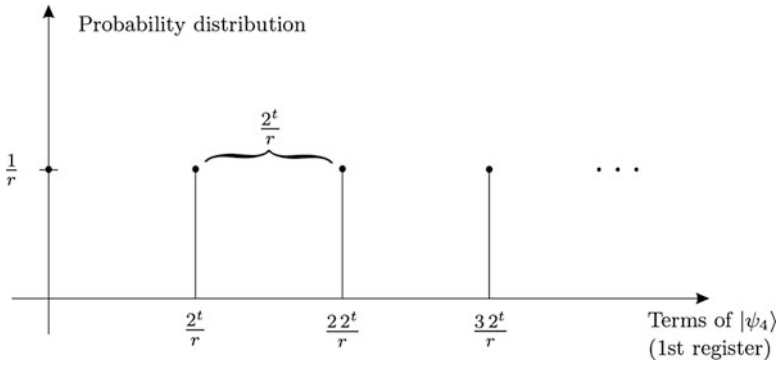

Fig. 4.4 Probability distribution of $|\psi_4\rangle$ measured in the computational basis. The horizontal axis has 2^t points, only the non-null terms are shown. The number of peaks is r and the period is $2^t/r$

halts, otherwise we keep applying it recursively. The recursive process is efficient, because the number of iterations is less than or equal to $\log_2 r$.

Take $N = 15$ as an example, which is the least nontrivial composite number. The set of numbers less than 15 that are coprime to 15 is $\{1, 2, 4, 7, 8, 11, 13, 14\}$. The numbers in the set $\{4, 11, 14\}$ have order two and the numbers in the set $\{2, 7, 8, 13\}$ have order four. Therefore, in any case r is a power of two and the factors of $N = 15$ can be found in a 8-bit quantum computer—because $t + n = 2\lceil \log_2 15 \rceil = 8$.

4.4 Generalization by Means of an Example

In the previous sections, we have considered a special case when the order r is a power of two and $t = n$ (t is the number of qubits in the first register—see Fig. 4.1—and $n = \lceil \log_2 N \rceil$). In this section, we consider the factorization of $N = 21$, that is the next nontrivial composite number. We must choose t such that 2^t is between N^2 and $2N^2$, which is always possible. For $N = 21$, the smallest value of t is nine. This is the simplest example allowed by the constraints, but enough to display all properties of Shor's algorithm.

The first step is to pick up x at random such that $1 < x < N$, and to test whether x is coprime to N. If not, we easily find a factor of N by calculating $\mathrm{GCD}(x, N)$. If yes, the quantum part of the algorithm starts. Suppose that $x = 2$ has been chosen. The goal is to find out that the order of x is $r = 6$. The quantum computer is initialized in the state

$$|\psi_0\rangle = |0\rangle\,|0\rangle\,, \tag{4.21}$$

where the first register has $t = 9$ qubits and the second has $n = 5$ qubits. Next step is the application of $H^{\otimes 9}$ on the first register yielding

$$|\psi_1\rangle = \frac{1}{\sqrt{512}} \sum_{j=0}^{511} |j\rangle\,|0\rangle\,, \tag{4.22}$$

as in Eq. (4.9). The next step is the application of V_x as defined in Eq. (4.6), which yields

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{\sqrt{512}} \sum_{j=0}^{511} |j\rangle\,\left|2^j \bmod N\right\rangle \\ &= \frac{1}{\sqrt{512}}\Big(|0\rangle\,|1\rangle + |1\rangle\,|2\rangle + |2\rangle\,|4\rangle + |3\rangle\,|8\rangle + |4\rangle\,|16\rangle + |5\rangle\,|11\rangle + \\ &\qquad |6\rangle\,|1\rangle + |7\rangle\,|2\rangle + |8\rangle\,|4\rangle + |9\rangle\,|8\rangle + |10\rangle\,|16\rangle + |11\rangle\,|11\rangle + \\ &\qquad |12\rangle\,|1\rangle + \ldots \Big). \end{aligned} \tag{4.23}$$

Notice that the above expression has the following pattern: the states of the second register of each "column" are the same. Therefore we can rearrange the terms in order to collect the second register:

$$
\begin{aligned}
|\psi_2\rangle = \frac{1}{\sqrt{512}}\Big[&\big(|0\rangle + |6\rangle + |12\rangle + \ldots + |504\rangle + |510\rangle \big)\,|1\rangle + \\
&\big(|1\rangle + |7\rangle + |13\rangle + \ldots + |505\rangle + |511\rangle \big)\,|2\rangle + \\
&\big(|2\rangle + |8\rangle + |14\rangle + \ldots + |506\rangle \big)\,|4\rangle + \\
&\big(|3\rangle + |9\rangle + |15\rangle + \ldots + |507\rangle \big)\,|8\rangle + \\
&\big(|4\rangle + |10\rangle + |16\rangle + \ldots + |508\rangle \big)\,|16\rangle + \\
&\big(|5\rangle + |11\rangle + |17\rangle + \ldots + |509\rangle \big)\,|11\rangle \Big].
\end{aligned}
\tag{4.24}
$$

This feature was made explicit in Eq. (4.11). Since the order is not a power of two, there is a small difference here: the first two lines of Eq. (4.24) have 86 terms, while the remaining ones have 85.

Now one measures the second register, yielding one of the numbers from set $\{1, 2, 4, 8, 16, 11\}$ equiprobably. Suppose that the result of the measurement is two, then

$$
|\psi_3\rangle = \frac{1}{\sqrt{86}} \left(|1\rangle + |7\rangle + |13\rangle + \ldots + |505\rangle + |511\rangle \right) |2\rangle \,. \tag{4.25}
$$

Notice that the state $|\psi_3\rangle$ was renormalized in order to have unit norm. It does not matter what is the result of the measurement; what matters is the periodic pattern of Eq. (4.25). The period of the states of the first register is the solution to the problem and the Fourier transform can reveal the value of the period. Thus, the next step is the application of the inverse Fourier transform on the first register of $|\psi_3\rangle$, yielding

$$
\begin{aligned}
|\psi_4\rangle &= \mathrm{DFT}^\dagger \, |\psi_3\rangle \\
&= \mathrm{DFT}^\dagger \left(\frac{1}{\sqrt{86}} \sum_{a=0}^{85} |6a+1\rangle \right) |2\rangle \\
&= \frac{1}{\sqrt{512}} \sum_{j=0}^{511} \left(\left[\frac{1}{\sqrt{86}} \sum_{a=0}^{85} e^{-2\pi i \frac{6ja}{512}} \right] e^{-2\pi i \frac{j}{512}} \, |j\rangle \right) |2\rangle \,,
\end{aligned}
\tag{4.26}
$$

where we have used Eq. (4.14) and have rearranged the sums. The last equation is similar to Eq. (4.19), although with an important difference. In Sect. 4.2, we were assuming that r divides 2^t. This is not true in the present example, since 6 does not divide 512, and thus we cannot use the identity of Eq. (4.17) to simplify the term in brackets in Eq. (4.26). This term never vanishes, but its main contribution is

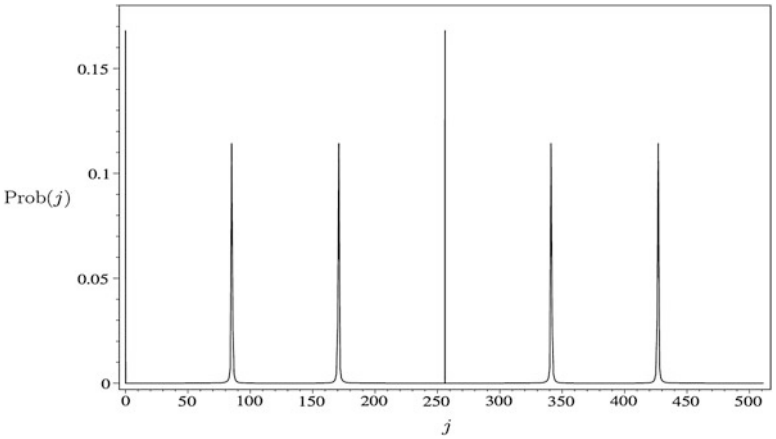

Fig. 4.5 Plot of Prob(j) against j. Compare to the plot of Fig. 4.4, where peaks are not spread and have the same height

still around $j = 0,\ 85,\ 171,\ 256,\ 341,\ 427$, which are obtained rounding $512k_0/6$ for k_0 from 0 to 5—compare to the discussion that follows Eq. (4.20). In order to observe this, let us plot the probability of getting the result j, in the interval 0 to 511, by measuring the first register of state $|\psi_4\rangle$. From Eq. (4.26), we have that the probability is

$$\text{Prob}(j) = \frac{1}{512 \times 86} \left| \sum_{a=0}^{85} e^{-2\pi i \frac{6ja}{512}} \right|^2 . \tag{4.27}$$

The plot of Prob(j) is shown in Fig. 4.5. We see the peaks around $j = 0,\ 85,\ 171,\ 256,\ 341,\ 427$, indicating a high probability of getting one of these values, or some value very close to them. In between, the probability is almost zero. The sharpness of the peaks depends on t—the number of qubits in the first register. The lower limit $2^t \geq N^2$ ensures a high probability in measuring a value of j carrying the desired information. Lomonaco performed a careful analysis of the expression in Eq. (4.27) [15], and Einarson performed a meticulous study of the peak form of Fig. 4.5 [6].

Let us analyze the possible measurement results. If we get $j = 0$, the first peak, the algorithm has failed in this round. It must be run again. We keep $x = 2$ and rerun the quantum part of the algorithm. From Eq. (4.27), we have that $\text{Prob}(0) = 86/512 \approx 0.167$, and thus the probability of getting $j = 0$ is low.

Now suppose we get $j = 85$ or any value in the second peak. We divide the result by 512 yielding $85/512$, which is a rational approximation of $k_0/6$, for $k_0 = 1$. How can we obtain r from $85/512$? The method of continued fraction approximation allows one to extract the desired information. A general continued fraction expansion of a rational number j_1/j_2 has the form

$$\frac{j_1}{j_2} = a_0 + \frac{1}{a_1 + \frac{1}{\ldots + \frac{1}{a_p}}}, \tag{4.28}$$

usually represented as $[a_0, a_1, \ldots, a_p]$, where a_0 is a non-negative integer and $a_1, \ldots, a_p$ are positive integers. The q-th convergent, $0 \leq q \leq p$, is defined as the rational number $[a_0, a_1, \ldots, a_q]$. It is an approximation to j_1/j_2 and has a denominator smaller than j_2.

This method is easily applied by inversion of the fraction followed by integer division with rational remainder. Inverting $85/512$ yields $512/85$, which is equal to $6 + 2/85$. We repeat the process with $2/85$ until we get the numerator equal to one. The result is

$$\frac{85}{512} = \frac{1}{6 + \frac{1}{42 + \frac{1}{2}}}. \tag{4.29}$$

Therefore, the convergents of $85/512$ are $1/6$, $42/253$, and $85/512$. We must select the convergents that have a denominator smaller than $N = 21$, since $r < N$. The inequality $r \leq \varphi(N)$ follows from the Euler's theorem, which states that $x^{\varphi(N)} \equiv 1 \mod N$, where x is a positive integer coprime to N and φ is the Euler's totient function, which gives the number of positive integers less than N, coprime to N. The inequality $\varphi(N) < N$ follows from the definition of φ.

Returning to our example, we find then that the method of continued fraction approximation yields $1/6$, and then $r = 6$. We check that $2^6 \equiv 1 \mod 21$, and then the quantum part of the algorithm halts with the correct answer. The order $r = 6$ is an even number, therefore $\mathrm{GCD}(2^{(6/2)} \pm 1, 21)$ gives two non trivial factors of 21. A straightforward calculation shows that any measured result in the second peak, say, $81 \leq j \leq 89$, yields the convergent $1/6$.

Consider now the third peak, which corresponds to $k_0/6$, for $k_0 = 2$. We apply again the method of continued fraction approximation, which yields $1/3$ for any j in the third peak, say, $167 \leq j \leq 175$. In this case, we have obtained $r_1 = 3$, which is only a factor of r, since $2^3 \equiv 8 \not\equiv 1 \mod 21$. We run the quantum part of the algorithm again to find the order of eight. We eventually obtain $r_2 = 2$, which yields $r = r_1 r_2 = 6$. The fourth and fifth peaks yield also factors of r. The last peak is similar to the second, yielding r directly.

The general account of the succeeding probability is as follows. The area under each peak is approximately 0.167. The first and fourth peaks are very different from the others—they are not spread. To calculate their contribution to the total probability, we take the basis equal to one. The area under the second, third, fifth, and last peaks are calculated by adding up $\mathrm{Prob}(j)$, for j running around the center of each peak. Hence, in approximately 17% cases, corresponding to first peak, the algorithm fails. In approximately 33% cases, corresponding to second and sixth peaks, the algorithm returns r in the first round. In approximately 50% cases, corresponding to all other peaks, the algorithm returns r in the second round or more. Now we should calculate the probability of finding r in the second round. For

the third and fifth peaks, the remaining factor is $r_2 = 2$. The graph equivalent to Fig. 4.5 in this case has two peaks, then the algorithm returns r_2 in 50% cases. For the fourth peak, the remaining factor is $r = 3$ and the algorithm returns r_2 in 66.6% cases. This amounts to $(2 \times 50\% + 66.6\%)/3$ of 50%, which is equal to around 22%. In summary, the success probability for $x = 2$ is around 55%.

Exercise 4.1 Calculate each step, classical and quantum, for factorizing the number $N = 35$ using Shor's algorithm. Take $x = 2$ as the required random integer. If that "random" integer does not work in the algorithm, take $x = 3, 4, 5$, and so on.

Exercise 4.2 Give a circuit for V_x explicitly in terms of universal quantum gates. Consider that the circuit would be used for factorizing the number $N = 15$, and take $x = 2$.

Exercise 4.3 Consider that you want to perform the Fourier transform of an arbitrary basis vector of four qubits, say, $|j_1 j_2 j_3 j_4\rangle$, on the computational basis over four qubits.

(a) Show the full quantum circuit for it in terms of universal gates.
(b) Show the matrix representation corresponding to the circuit.
(c) Calculate each intermediate step of the algorithm for the arbitrary input.

4.5 Fourier Transform in Terms of the Universal Gates

In the previous section, we have shown that Shor's algorithm is an efficient probabilistic algorithm, assuming that the Fourier transform could be implemented efficiently. In this section, we decompose the Fourier transform in terms of a set of universal gates composed by CNOT and one-qubit gates. This decomposition allows one to measure the efficiency of the quantum discrete Fourier transform and shows how to implement it in an actual quantum computer.

Recall that the Fourier transform of the states of the computational basis is given by

$$\mathrm{DFT}\,|j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N}\,|k\rangle\,. \tag{4.30}$$

After observing that the right hand side of Eq. (4.30) has N terms and the computational basis has N states, we may conclude that the time complexity to calculate classically the Fourier transform of the computational basis using Eq. (4.30) is $O(N^2)$, or equivalently $O(2^{2n})$, which means a double exponential growth if we consider the number of input bits. A very important result in computer science was the development of the classical algorithm known as Fast Fourier Transform—usually abbreviated to FFT—which reduced that complexity to $O(n2^n)$. In this context, we show the improvement by recognizing that the right-hand side of Eq. (4.30) is a very special kind of expansion, which can be fully factored. For example, the Fourier transform of the basis $\{|0\rangle, |1\rangle, |2\rangle, |3\rangle\}$ can be written as

$$\begin{aligned}
\mathrm{DFT}\,|0\rangle &= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \\
\mathrm{DFT}\,|1\rangle &= \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle + i\,|1\rangle}{\sqrt{2}}\right) \\
\mathrm{DFT}\,|2\rangle &= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
\mathrm{DFT}\,|3\rangle &= \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle - i\,|1\rangle}{\sqrt{2}}\right).
\end{aligned} \tag{4.31}$$

Note that, in the example of Eq. (4.31), we are using the basis of a two-qubit system in order to factor the right-hand side. Let us now factor the general expression. The first step is to write Eq. (4.30) in the form

$$\mathrm{DFT}\,|j\rangle = \frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} e^{2\pi i j \sum_{l=1}^{n} \frac{k_l}{2^l}} |k_1\rangle \otimes \ldots \otimes |k_n\rangle\,, \tag{4.32}$$

where the ket $|k\rangle$ was converted to the binary base and we have used the expansion $k = \sum_{l=1}^{n} k_l 2^{n-l}$ in the exponent. Using the fact that the exponential of a sum is a product of exponentials, Eq. (4.32) turns into a (non-commutative) product of the following kets:

$$\mathrm{DFT}\,|j\rangle = \frac{1}{\sqrt{2^n}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \prod_{l=1}^{n} \left(e^{2\pi i j \frac{k_l}{2^l}} |k_l\rangle\right). \tag{4.33}$$

Now we factor Eq. (4.33) by interchanging the sums and the product,

$$\mathrm{DFT}\,|j\rangle = \frac{1}{\sqrt{2^n}} \prod_{l=1}^{n} \sum_{k_l=0}^{1} \left(e^{2\pi i j \frac{k_l}{2^l}} |k_l\rangle\right). \tag{4.34}$$

We can easily check that the last equation is correct by performing the calculations backwards. Simply expand the product in Eq. (4.34) and then put all sums at the beginning of the resulting expression to obtain Eq. (4.33). By expanding the sum of Eq. (4.34) and then the product, we finally get

$$\begin{aligned}
\mathrm{DFT}\,|j\rangle &= \frac{1}{\sqrt{2^n}} \prod_{l=1}^{n} \left(|0\rangle + e^{2\pi i j/2^l}\,|1\rangle\right) \\
&= \left(\frac{|0\rangle + e^{2\pi i \frac{j}{2}}\,|1\rangle}{\sqrt{2}}\right) \otimes \left(\frac{|0\rangle + e^{2\pi i \frac{j}{2^2}}\,|1\rangle}{\sqrt{2}}\right) \otimes \ldots \otimes \left(\frac{|0\rangle + e^{2\pi i \frac{j}{2^n}}\,|1\rangle}{\sqrt{2}}\right).
\end{aligned} \tag{4.35}$$

The complexity to calculate Eq. (4.35) for one $|j\rangle$ is $O(n)$, since there are n terms in the product and each term can be calculated in constant time. The complexity in the classical calculation of the fast Fourier transform of the whole computational basis is still exponential, $O(n2^n)$, since the calculation is performed on each of the 2^n basis elements, one at a time. On the other hand, the quantum computer uses quantum parallelism, and the Fourier transform of the state

$$|\psi\rangle = \sum_{a=0}^{2^n-1} F(a)\,|a\rangle\,, \tag{4.36}$$

which has an exponential number of terms, is calculated with one application of the quantum Fourier transform. The Fourier transform of the 2^n basis elements is performed simultaneously, so the complexity of the quantum Fourier transform is measured by the size of its circuit. We now show that it requires $O(n^2)$ gates.

Consider the circuit of Fig. 4.6, which depicts part of the quantum algorithm for Fourier transform. It is easy to check that the value of the qubits $|j_m\rangle$, $m \neq l$, does not change. We should still check what happens with qubit $|j_l\rangle$.

The unitary matrices R_k are defined as

$$R_k = \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(2\pi i \frac{1}{2^k}\right) \end{bmatrix}. \tag{4.37}$$

Each R_k gate in Fig. 4.6 is controlled by qubit $|j_{k+l-1}\rangle$. Therefore, if $j_{k+l-1} = 0$, then R_k must be replaced by the identity matrix, which means that no action will be performed, and if $j_{k+l-1} = 1$, then R_k must be applied to qubit $|j_l\rangle$. This means that, for calculation purposes, gates R_k controlled by qubit $|j_{k+l-1}\rangle$ can be replaced by "one-qubit gates" defined as

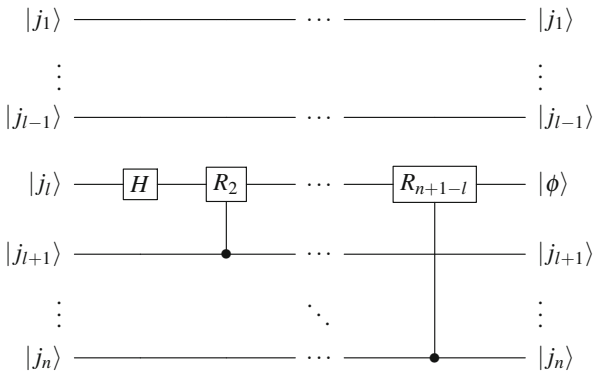

Fig. 4.6 Part of the quantum Fourier transform circuit that acts on qubit $|j_l\rangle$. The value of all qubits does not change, except $|j_l\rangle$ that changes to $|\phi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \exp\left(\frac{2\pi i j}{2^{n+1-l}}\right)|1\rangle$

$$CR_k = \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(2\pi i \frac{j_{k+l-1}}{2^k}\right) \end{bmatrix}. \tag{4.38}$$

In fact, CR_k is a two-qubit gate that is temporarily being represented as an one-qubit gate.

In order to simplify the calculations, notice that

$$\begin{aligned} H\,|j_l\rangle &= \frac{|0\rangle + e^{2\pi i \frac{j_l}{2}}\,|1\rangle}{\sqrt{2}} \\ &= CR_1\,|+\rangle\,, \end{aligned} \tag{4.39}$$

where $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Thus, instead of using the equation

$$|\psi\rangle = CR_{n+1-l} \ldots CR_2 H\,|j_l\rangle\,, \tag{4.40}$$

which can be read directly from Fig. 4.6, we will use

$$|\psi\rangle = CR_{n+1-l} \ldots CR_2 CR_1\,|+\rangle\,. \tag{4.41}$$

We define the operator

$$PR_{n+1-l} = \prod_{k=n+1-l}^{1} CR_k, \tag{4.42}$$

where the product is in reversed order—remember this is a product of unitary operators, and thus non-commutative. From Eqs. (4.38) and (4.42), we get

$$\begin{aligned} PR_{n+1-l} &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 0 & \exp 2\pi i \left(\frac{j_n}{2^{n+1-l}} + \ldots + \frac{j_l}{2}\right) \end{bmatrix} \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(2\pi i \frac{j}{2^{n+1-l}}\right) \end{bmatrix}, \end{aligned} \tag{4.43}$$

where we have used that $j = \sum_{m=1}^{n} j_m 2^{n-m}$ and the fact that the first $l-1$ terms of this expansion do not contribute—they are integer multiples of $2\pi i$ in Eq. (4.43). We finally get

$$\begin{aligned} |\psi\rangle &= PR_{n+1-l}\,|+\rangle \\ &= \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{j}{2^{n+1-l}}}\,|1\rangle\right). \end{aligned} \tag{4.44}$$

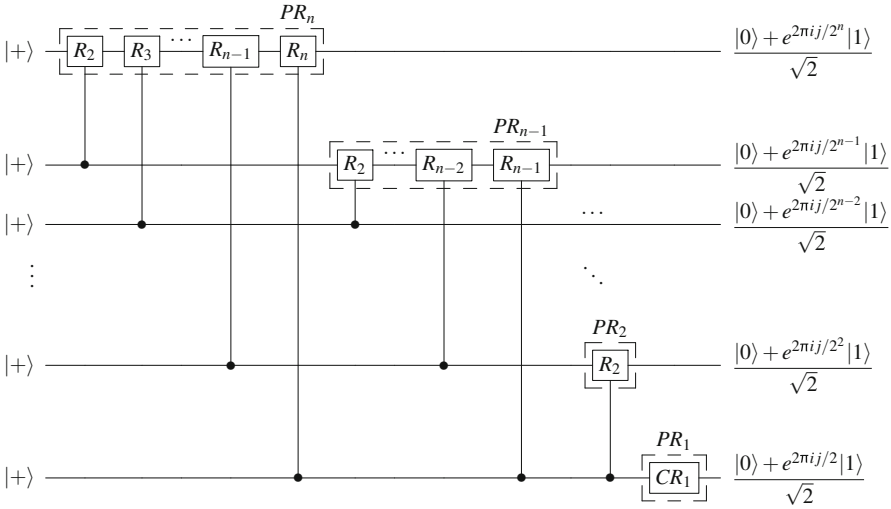

Fig. 4.7 Intermediate circuit for the quantum Fourier Transform. The input is taken as $|+\rangle$ for calculation purposes as explained in Eq. (4.39). The output is in reverse order with respect to Eq. (4.35)

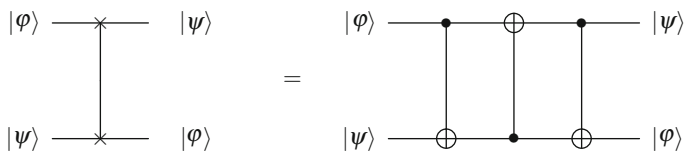

Fig. 4.8 Decomposition of swap gate in terms of universal gates

Notice that PR_{n+1-l} cannot be implemented directly acting only in the l-th qubit, because it requires the values of j_{l+1} to j_n.

The next step is the circuit depicted in Fig. 4.7. We have merged the R_k gates using Eq. (4.42). The gates PR_k, with k from n to 1, are placed in sequence in Fig. 4.7, so that the output of the first qubit is the last term of Eq. (4.35), corresponding to the action of PR_n on $|\psi_1\rangle$ controlled by the other qubits, which do not change. The same process is repeated by PR_{n-1} acting on $|\psi_2\rangle$, yielding the term before the last in Eq. (4.35), and so on, until reproducing all the terms of the Fourier transform. Now it remains to reverse the order of the states of the qubits.

In order to reverse the states of two generic qubits, we use the circuit of Fig. 4.8. Let us show why this circuit works as desired. Take as input a vector of the computation basis, say, $|\varphi\rangle\,|\psi\rangle = |0\rangle\,|1\rangle$. The first CNOT gate of Fig. 4.8 does not change this state. Then, the second CNOT gate—which is upside down—changes it to $|1\rangle\,|1\rangle$. Finally, the last CNOT gate changes the state to $|1\rangle\,|0\rangle$. The output is $|\psi\rangle\,|\varphi\rangle$, which means that the circuit worked for basis state $|0\rangle\,|1\rangle$ as input. If we repeat the same process with $|0\rangle\,|0\rangle$, $|1\rangle\,|0\rangle$, and $|1\rangle\,|1\rangle$ as inputs, we conclude that

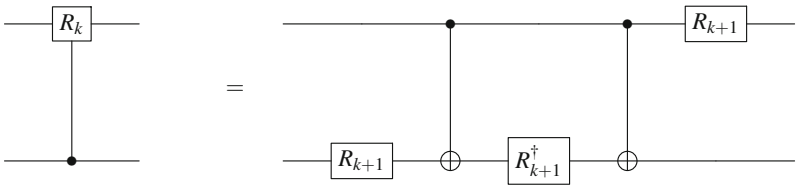

Fig. 4.9 Decomposition of the controlled R_k gates in terms of the universal gates

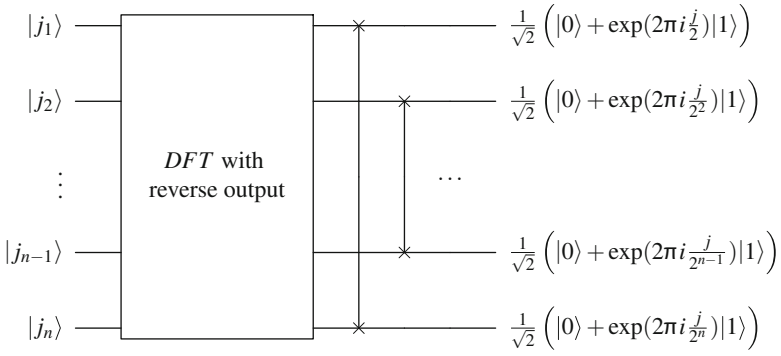

Fig. 4.10 The complete circuit for the quantum Fourier Transform

the circuit inverts all states of the computational basis, therefore it inverts a generic state of the form $|\varphi\rangle\,|\psi\rangle$.

The decomposition is still not complete. It remains to write the controlled R_k gates in terms of CNOT and one-qubit gates. This decomposition is given in Fig. 4.9. The verification of this decomposition is straightforward. One simply follows what happens to each vector of the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ in both circuits.

The complete circuit for the quantum Fourier transform is given in Fig. 4.10. Now we can calculate the complexity of the quantum Fourier circuit. By counting the number of elementary gates in Figs. 4.6, 4.7, 4.8 and 4.9, we get the leading term $5n^2/2$, which implies that the complexity is $O(n^2)$.

By now one should be asking about the decomposition of V_x in terms of the elementary gates. Operator V_x is the largest gate of Fig. 4.1 and the "bottleneck" of the quantum factoring algorithm, due to the time and space consumed to perform the modular exponentiation. However, this bottleneck is not so strict, since by using the well known classical method of repeated squaring and ordinary multiplication algorithms, the complexity to calculate modular exponentiation is $O(n^3)$. The quantum circuit can be obtained from the classical circuit by replacing the irreversible classical gates by the reversible quantum counterpart.

For simplicity, when we discussed operator V_x we kept x fixed. This approach is a problem in recursive calls of the algorithm when x changes, since for each x, a new circuit must be built. In order to be practical, the circuit for operator V_x should

be designed in such a way that the calculations are performed for an arbitrary x, which in turn should be provided through an ancillary register.

4.6 Computational Simulations

The *LIQUiD* programming language[8] has been developed by Microsoft, and allows advanced simulations of several quantum algorithms, including a builtin test of Shor's algorithm. The user just needs to tell the number to be factorized and whether the circuit should be optimized. For instance, if one wants to factorize $N = 21$ with the circuit just studied in this chapter, without optimization, the command to be executed should be

```
$ mono Liquid.exe "--Shor(21, false)"
```

on Linux or MacOS operating systems, or

```
> Liquid.exe --Shor(21, false)
```

on Microsoft Windows. In either case, the output would inform several details concerning time and memory consumption besides the result.

Sympy is a Python library for symbolic mathematics which has already some commands for simulating Shor's algorithm.[9] The simulation can be performed as easily as

```
import sympy.physics.quantum.shor as q
q.shor(21)
```

The result is a factor of N if the random number leads to a successful run, and an error message otherwise. The output does not give much details, although the source code is open and much more interesting as a learning resource.

The Quantum Computing Playground[10] contains a predefined open-source simulation of Shor's algorithm, and provides several visualizations. The source code for this simulation uses a language called QScript, and the website gives a tutorial teaching how to use it.

There is also an implementation of Shor's algorithm for IBM Quantum Experience in [3].

References

1. Boneh, D., Lipton, R.J.: Quantum cryptanalysis of hidden linear functions. In: Annual International Cryptology Conference, pp. 424–437. Springer, Heidelberg (1995)
2. Childs, A.M., van Dam, W.: Quantum algorithms for algebraic problems. Rev. Mod. Phys. **82**, 1–52 (2010). https://doi.org/10.1103/RevModPhys.82.1

[8] Available at http://stationq.github.io/Liquid/ as of October 2016.

[9] Available at http://docs.sympy.org/dev/modules/physics/quantum/shor.html as of October 2016.

[10] Available at http://www.quantumplayground.net as of October 2016.

3. Coles, P.J., Eidenbenz, S., Pakin, S., Adedoyin, A., Ambrosiano, J., Anisimov, P., Casper, W., Chennupati, G., Coffrin, C., Djidjev, H., Gunter, D., Karra, S., Lemons, N., Lin, S., Lokhov, A., Malyzhenkov, A., Mascarenas, D., Mniszewski, S., Nadiga, B., O'Malley, D., Oyen, D., Prasad, L., Roberts, R., Romero, P., Santhi, N., Sinitsyn, N., Swart, P., Vuffray, M., Wendelberger, J., Yoon, B., Zamora, R., Zhu, W.: Quantum Algorithm Implementations for Beginners. Technical Report LA-UR:2018-229, Los Alamos National Laboratory, Los Alamos (2018)
4. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**(90), 297 (1965). https://doi.org/10.2307/2003354
5. Crandall, R., Pomerance, C.: Prime Numbers: A Computational Perspective, 2nd edn. Springer, New York (2005). https://doi.org/10.1007/0-387-28979-8
6. Einarsson, G.: Probability Analysis of a Quantum Computer (2003). http://arxiv.org/abs/quant-ph/0303074
7. Ekert, A., Jozsa, R.: Quantum computation and Shor's factoring algorithm. Rev. Mod. Phys. **68**(3), 733–753 (1996). https://doi.org/10.1103/RevModPhys.68.733
8. Gathen, J.V.Z., Gerhard, J.: Modern Computer Algebra. Cambridge University Press, Cambridge (1999)
9. Hoffstein, J., Pipher, J., Silverman, J.: An Introduction to Mathematical Cryptography. Undergraduate Texts in Mathematics. Springer, New York (2008). https://doi.org/10.1007/978-0-387-77993-5
10. Jozsa, R.: Quantum algorithms and the Fourier transform. Proc. R. Soc. Lond. A Math. Phys. Eng. Sci. **454**(1969), 323–337 (1998). https://doi.org/10.1098/rspa.1998.0163
11. Jozsa, R.: Quantum factoring, discrete logarithms, and the hidden subgroup problem. Comput. Sci. Eng. **3**(2), 34–43 (2001). https://doi.org/10.1109/5992.909000
12. Kitaev, A.Y.: Quantum measurements and the abelian stabilizer problem (1995). ArXiv: quant-ph/9511026
13. Knuth, D.E.: The Art of Computer Programming, vol. 2, Seminumerical Algorithms, 3rd edn. Addison-Wesley, Upper Saddle River (1997)
14. Lenstra, H.W.: Factoring integers with elliptic curves. Ann. Math. **126**(3), 649–673 (1987)
15. Lomonaco, S.J.: Quantum computation: a grand mathematical challenge for the twenty-first century and millennium. In: Samuel, J., Lomonaco, J. (ed.) Proceedings of Symposia in Applied Mathematics, pp. 161–179. American Mathematical Society, Washington (2000). https://doi.org/10.1090/psapm/058/1922897
16. Marquezino, F., Portugal, R., Sasse, F.: Obtaining the quantum Fourier transform from the classical FFT with QR decomposition. J. Comput. Appl. Math. **235**(1), 74–81 (2010). https://doi.org/10.1016/j.cam.2010.05.012
17. Montanaro, A.: Quantum algorithms: an overview. Npj Quantum Inf. **2**(15023) (2016). https://doi.org/10.1038/npjqi.2015.23
18. Monz, T., Nigg, D., Martinez, E.A., Brandl, M.F., Schindler, P., Rines, R., Wang, S.X., Chuang, I.L., Blatt, R.: Realization of a scalable shor algorithm. Science **351**(6277), 1068–1070 (2016). https://doi.org/10.1126/science.aad9480
19. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2010). https://doi.org/10.1017/CBO9780511976667
20. Pollard, J.M.: A monte carlo method for factorization. BIT Numer. Math. **15**(3), 331–334 (1975). https://doi.org/10.1007/BF01933667
21. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE Computer Society, Washington (1994). https://doi.org/10.1109/SFCS.1994.365700
22. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. **41**(2), 303–332 (1999). https://doi.org/10.1137/S0036144598347011
23. Vandersypen, L.M.K., Steffen, M., Breyta, G., Yannoni, C.S., Sherwood, M.H., Chuang, I.L.: Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. Nature **414**(6866), 883–887 (2001). https://doi.org/10.1038/414883a
24. Wagstaff Jr., S.S.: The joy of factoring. In: Student Mathematical Library, vol. 68. American Mathematical Society, Washington (2013). https://doi.org/10.1090/stml/068