1 Quantum Computers

This chapter serves as an introduction to the subject of Quantum Computation. It contains no new results, though certain derivations and proofs are given in an original manner.

Before describing what a QC is, let us first look at what a classical computer is in terms we will later use to describe a QC.

A classical computer is built up of a collection of two state systems (bits). These states are usually labelled 0 and 1, so information can be represented using several of these states.

Numbers can be represented quite easily by using these bits. For instance the number 11 can be represented by a collection of four of these bits, $1011_b$ (see Appendix A). It has been shown that all the manipulations of this information can be broken down into one two-bit operator called the NAND instruction. The results of this operation are given in Table 1-1.

<table>
<thead>
<tr>
<th>N Chandur</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1-1: Truth table for the NAND instruction. The horizontal and vertical axis labels denote the input bits.

The state $S$ of a two-bit computer can be described as a unit length vector $S = (s_1, s_2, s_3, s_4)$, representing the state of the computer in the “Hilbert” space with basis vectors $\{(00_b), (01_b), (10_b), (11_b)\}$. So $S$ denotes the state $s_1(00_b) + s_2(01_b) + s_3(10_b) + s_4(11_b)$. Of course, on a deterministic classical computer, only one of the components can be different from 0.

We can now write a NAND instruction, which takes the two bits and stores the result in the first bit, as a matrix acting on this state-vector. On the basis states it acts as follows

\begin{align}
(00_b) &\rightarrow (10_b), & (1.1a) \\
(01_b) &\rightarrow (11_b), & (1.1b) \\
(10_b) &\rightarrow (10_b), & (1.1c) \\
(11_b) &\rightarrow (01_b). & (1.1d)
\end{align}
Therefore we can write

\[
\text{NAND} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.
\] (1.2)

The NAND operation is universal, i.e., any operation on a classical computer can be decomposed into NAND instructions.

Clearly this operation does not have an inverse. So a classical computer that uses the NAND instruction as a building block for higher order instruction performs an irreversible operation. Hence, from a thermodynamic point of view, such a computer will have to exchange energy with its surroundings [Lan61].

It is possible to find universal operations which do have an inverse. This makes reversible classical computing possible, at the cost of more storage space.\(^1\)

### 1.0.1 Qubits and Bytes

A QC is also built up of a collection of two level systems (now called qubits). These states are usually labelled \(|0\rangle\) and \(|1\rangle\), so, like before, information can be represented using several of these states.

For example, one could use a spin one-half system and make the identification \(|0\rangle \equiv |\uparrow\rangle\) and \(|1\rangle \equiv |\downarrow\rangle\) (for details on the representation of numbers on a QC see Appendix A).

Due to the nature of quantum mechanics, we can only act on the qubits with unitary operations, since all the operations are based on quantum mechanical time evolutions, i.e., solutions of the time-dependent Schrödinger equation. Hence we cannot implement the NAND instruction directly. There is another two-qubit instruction, however, called the Control-NOT (CNOT) instruction, which can be implemented unitarily. It acts on the basis states as follows

\[
\begin{align*}
|00_b\rangle & \rightarrow |00_b\rangle, \\
|01_b\rangle & \rightarrow |11_b\rangle, \\
|10_b\rangle & \rightarrow |10_b\rangle, \\
|11_b\rangle & \rightarrow |01_b\rangle,
\end{align*}
\] (1.3a-d)

and therefore the CNOT instruction can be written as

\[
\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.
\] (1.4)

\(^1\)When we want to calculate a function that does not have an inverse, in a reversible manner, we have to keep the input value in memory as well.
It is clear that this operation has an inverse and is also unitary. In general, an operation on a QC can be any unitary transformation. So clearly the CNOT instruction by itself cannot be universal for the QC in the same sense a NAND instruction is universal for the classical computer. It has been shown, [DiV95, SW95, Lio95, DBE95] however, that the CNOT instruction combined with arbitrary one-qubit phase rotations are universal, i.e., any unitary operation can be expressed in terms of them. In general, the decomposition of an arbitrary unitary operation on $n$ qubits into two-qubit operations might take an exponential number of them [BBC+95].

### 1.0.2 Larger Scale Operations

Like with classical computers, the most elementary instructions are not the most sensible operations to express algorithms in. It is expedient to construct operations like addition, multiplication, etc., and use these as the basic building blocks. In order to get a taste for how to build more complicated operations, we first introduce a graphical notation for the one-, two-, and three-qubit operations, see Fig. 1-1. The Toffoli gate can be seen as a Control-Control-Not, where a qubit is flipped on the condition that two other qubits are 1. The graphical notation was introduced by Feynman [Fey85].

![Figure 1-1: Schematic notation together with the truth tables for the NOT, the CNOT, and the Toffoli gates. The operations should be read from left to right. The ⨯ denotes a (conditional) write operation and the ◦ symbolises a read operation.](image)

These logical operations can be combined to obtain arithmetical ones, see Fig. 1-2, the
easiest one being the addition of two qubits [Fey85]. Like with classical computers, we have to take care of overflows and define a carry, defined as the product $a \times b$ of the two qubits. From the basic building blocks like the ones given in given in Fig. 1-2 one can construct more complex operations, like the addition of two registers$^2$. Examples of how such instructions can be constructed are given in Ref. [VBE96]. The general upshot is that the construction goes along similar lines as on classical computers, except that in general more memory space will be required, because the operations are intrinsically reversible.

### 1.0.3 Superposition and Entanglement

Two important notions that distinguish quantum mechanics from classical mechanics are superposition and entanglement. For a QC this means that it does not have to be in a definite state, i.e., it can be in a superposition.

Say we have a function $f(x)$. We can implement this function as a unitary operation $U_f$ on a quantum register. So we could have

$$U_f \left( |x\rangle \otimes |0\rangle \right) = |x\rangle \otimes |f(x)\rangle. \tag{1.5}$$

See Appendix A.2 for more details on the meaning of $|x\rangle$. So far, this is not different from a classical computer. But we can also have this function act on a superposition of states

$$U_f \frac{1}{\sqrt{N}} \sum_{x=0}^{N} |x\rangle \otimes |0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N} |x\rangle \otimes |f(x)\rangle, \tag{1.6}$$

thus we have calculated $f(x)$ for $N$ values of $x$ at once. In principle this “quantum parallelism” could provide an exponential speedup, although making use of this feature is not a trivial matter.

In the final state (1.6) we not only have a clear case of superposition, but also of entanglement, i.e., the final state cannot be written as a direct product of the states of the input register $|x\rangle$ and the output register $|f(x)\rangle$.

---

$^2$By register we mean a collection of (qu-)bits representing a larger number.
1.1 Algorithms

For a QC to be of any practical use, one has to build algorithms that specifically use the superposition of states. Probably the most widely known example of such an algorithm is Shor’s algorithm to factor large integers, see [Sho94, Sho97, Sho99]. The important core of this algorithm is based on a quantum fast Fourier transform (FFT) that provides the exponential speedup. We will describe this algorithm in detail in Section 1.1.3. But as a test of the feasibility of quantum computation, this algorithm is far too complex for present-day hardware. Less involved algorithms, like the Grover algorithm [Gro96] (See Section 1.1.2) to search an unsorted database, are better suited to fit this purpose. But we will start with arguably the simplest, the Deutsch-Josza algorithm [DJ92].

1.1.1 Deutsch-Josza

Consider the task of telling whether a coin is fair, i.e., to check if it has both a head and a tail and not two heads or two tails. What a classical observer would have to do is look at both sides of the coin and check if they are different. So, classically, two operations have to be done: Look at side 1 and Look at side 2. For a QC this is not necessarily the case. A QC can make use of a superposition of the operations Look-at-side-1 and Look-at-side-2 to perform both operations at the same time.

Let us state this in more mathematical terms and generalise it a little. We introduce a two-valued function \( f \) for the output of the coin. Let the input be an \( L \)-bit number \( x \). Now we have

\[
\begin{align*}
  f(x) &= \{0, 1\},
\end{align*}
\]

where \( x \) is an integer number between 0 (inclusive) and \( 2^L \). A toss of the coin is now simulated by generating a random number \( x \) and calculating \( f(x) \), e.g., 0 is heads and 1 is tails. For a fair coin, exactly half the number of inputs gives 0 and the other half gives 1. If this is the case we call the function balanced. If, however, the function produces 0 (or 1) whatever the input we call it constant. As an example we look at all the possible functions with one bit input values, see Table 1-2. In this case, there are only constant or balanced functions, and with a classical algorithm we would have to “look” twice to determine to which class our function belongs. For two bit input values, the

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f_1(x) )</th>
<th>( f_2(x) )</th>
<th>( f_3(x) )</th>
<th>( f_4(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(_b)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1(_b)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1-2: All the possible functions for one bit inputs values. The subscript \( b \) denotes a binary representation, see Appendix A.2.
number of functions that fall in the classes constant or balanced increases, see Table 1-3. Classically, if we have $N = 2^L$ possible different input values, we have to “look” maximally $N/2 + 1$ times, to determine whether the function belongs to the balanced or the constant class. So the maximum time needed to perform a classical algorithm scales exponentially in the length $L$ of the input $x$. The average time, however, is much smaller. Given $N$, the number of constant functions is 2 and the number of balanced functions is given by

$$N_B = \binom{N}{N/2}. \quad (1.8)$$

So how long does it take on the average to see whether a function is balanced? A general balanced function starts with (as a function of $x$) $p > 0$ ones (zeros), then a zero (one), and then $N/2 - p$ ones (zeros) distributed over $N - p - 1$ positions, e.g.,

$$(1.9)$$

$$\underbrace{1, 1, \cdots, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, \cdots, 1}_{N/2 - p \text{ ones and } N/2 - 1 \text{ zeros}}.$$

It takes $p + 1$ “measurements” to determine that such a function is balanced. So now we can easily derive that the average number of trials to determine whether a function is constant or balanced is

$$N_c = \frac{1}{2 + \binom{N}{N/2}} \left[ 2(N/2 + 1) + 2 \sum_{p=1}^{N/2} (p + 1) \binom{N - p - 1}{N/2 - p} \right]$$

$$= 3 - \frac{4}{N + 2} + \frac{N - 2}{N + 2} \frac{N}{2 + \binom{N}{N/2}}, \quad (1.10)$$

and $N_c \to 3$ as $N \to \infty$. So a classical algorithm that samples the functions at random will on the average need approximately three trials.

The claim is now that on a QC we only have to “look” once. How do we go about achieving this? Classically, we have to calculate $f(x)$ for every possible input value,
1.1 Algorithms

With a QC, this is no different, except that now we have the possibility to do it somehow in parallel, using a superposition of all the possible input values. Let us write the input \( x \) as

\[
|X\rangle = \frac{1}{\sqrt{2^L}} \sum_{x=0}^{2^L-1} |x\rangle.
\]

(1.11)

Now we want to apply \( f \) to this superposition and read out a result which tells us that \( f \) is either constant or balanced. Intuitively we would make the guess to map the result of \( f(x) \) to a phase rotation of some sort, such that if \( f(x) = 0 \) we get a rotation in one direction and with \( f(x) = 1 \) in the opposite, giving an overall phase rotation if \( f \) is constant and no phase rotation if \( f \) is balanced. Let us look at this in more mathematical terms. We make a quantum register with \( L + 1 \) bits and label the states a \( |x, y\rangle \), where \( x \) is an \( L \) qubit number and \( y \) is a single qubit. We initialise the QC with \( |x\rangle|y\rangle = 1/\sqrt{2}|0\rangle\left(|0\rangle - |1\rangle\right) \). A way to generate the full superposition (1.11) is to perform a Walsh-Hadamard (WH) transformation on all the qubits separately. A WH transformation acts on a single qubit as

\[
W|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle),
\]

(1.12a)

\[
W|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).
\]

(1.12b)

We define a WH transformation on an \( L \)-qubit register as follows

\[
W_T = \prod_{p=0}^{L-1} W_p,
\]

(1.13)

where \( p \) labels the qubit of the register the transformation \( W_p \) acts on. For a register in a general state \( |x\rangle \) we have

\[
W_p|x\rangle = W_p|(x_0 \ldots x_p \ldots x_{L-1})_b\rangle
= \frac{1}{\sqrt{2}} \left[ (x_0 \ldots 0 \ldots x_{L-1})_b\rangle + (-1)^{x_p} |(x_0 \ldots 1 \ldots x_{L-1})_b\rangle \right]
= \frac{1}{\sqrt{2}} \sum_{z_p=0}^{1} (-1)^{z_p x_p} |(x_0 \ldots z_p \ldots x_{L-1})_b\rangle,
\]

(1.14)
therefore

\[ W_T |x\rangle = \prod_{p=0}^{L-1} W_p |x\rangle \]

\[ = \frac{1}{\sqrt{2^L}} \sum_{z_0, z_1, \ldots, z_{L-1} = 0}^{1} (-1)^{x_0 z_0 + x_1 z_1 + \cdots + x_{L-1} z_{L-1}} |z_0 z_1 \ldots z_{L-1}\rangle_b \]

\[ = \frac{1}{\sqrt{2^L}} \sum_{z=0}^{2^L-1} (-1)^{(x,z)} |z\rangle, \quad (1.15) \]

where \((x, z)\) denotes a bitwise inner product between \(x\) and \(z\). For \(x = 0\) we have a special case, which is often used to generate a uniform superposition of a set of numbers,

\[ W_T |0\rangle = \frac{1}{\sqrt{2^L}} \sum_{z=0}^{2^L-1} |z\rangle. \quad (1.16) \]

Indeed, we use this operation on the \(x\)-register of our QC,

\[ W_T \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) = \frac{1}{\sqrt{2^{L+1}}} \sum_{z=0}^{2^L-1} |z\rangle (|0\rangle - |1\rangle). \quad (1.17) \]

Now we represent our \(f\) as a unitary operation, such that

\[ U_f |x, y\rangle = |x, y \oplus f(x)\rangle, \quad (1.18) \]

where \(\oplus\) denotes modular addition, i.e., \(a \oplus b = (a + b) \mod 2\). It is clear that

\[ U_f U_f |x, y\rangle = U_f |x, y \oplus f(x)\rangle = |x, y \oplus f(x) \oplus f(x)\rangle = |x, y\rangle. \quad (1.19) \]

If we apply this operation to our quantum register, we have

\[ U_f \frac{1}{\sqrt{2^{L+1}}} \sum_{x=0}^{2^L-1} |x\rangle (|0\rangle - |1\rangle) = \frac{1}{\sqrt{2^{L+1}}} \sum_{x=0}^{2^L-1} |x\rangle (|f(x)\rangle - |1 \oplus f(x)\rangle) \]

\[ = \frac{1}{\sqrt{2^{L+1}}} \sum_{x=0}^{2^L-1} (-1)^{f(x)} |x\rangle (|0\rangle - |1\rangle), \quad (1.20) \]

and we apply \(W_T\), which gives

\[ W_T \frac{1}{2^{(L+1)/2}} \sum_{x=0}^{2^L-1} (-1)^{f(x)} |x\rangle (|0\rangle - |1\rangle) \]

\[ = \frac{1}{2^{L+1/2}} \sum_{x,z=0}^{2^L-1} (-1)^{f(x)+(x,z)} |z\rangle (|0\rangle - |1\rangle). \quad (1.21) \]
We have now arrived at our final state and measure the value of the register. If we find
our system to be in the state $|0\rangle$ then our function was constant; if not, our function
was balanced. How do we see this? Well, let us calculate the probability for the input
register to be in the state $|0\rangle$, we have

$$
\left| \langle 0 | \frac{1}{2^L} \sum_{x,z=0}^{2^L-1} (-1)^{f(x)+(x,z)} | z \rangle \right|^2 = \left| \frac{1}{2^L} \sum_{x,z=0}^{2^L-1} (-1)^{f(x)+(x,z)} \langle 0 | z \rangle \right|^2
$$

$$
= \left| \frac{1}{2^L} \sum_{x=0}^{2^L-1} (-1)^{f(x)} \langle 0 | z \rangle \right|^2.
$$

(1.22)

It is easy to see that for a balanced function this probability is 0 and for a constant func-
tion it is 1. Hence we see that with a quantum algorithm (QA), we only need to “calcu-
late” $f$ once. Of course there are some remarks one should make about how the number
of operations scales with $L$. Measuring the whole register takes $O(L) = O(\log N)$ ob-
servations. But in principle, we could say the same about reading of the result of a
classical computer. The same holds for the $W_T$ operation, which can be build up from
$L$ one-qubit operations. Addition on a classical computer, for instance, is generally
counted as an $O(1)$ operation, while in principle it is an $O(\log(N))$ operation. It is
just that most modern computers can do addition for numbers below some maximum
number in a constant amount of time.

### 1.1.2 Searching an Unordered List

Finding an object according to some criteria is a basic problem which turns up as an
intermediate or final step in many algorithms. For every problem we need to do different
preprocessing on our data, e.g., calculating the length of a path, the assessment of a
chess position, the remainder of a division, etc. We will not go into this preprocessing
because it is far from generic. We will assume that there is a function $f(n)$, where
$n$ labels the objects, that returns $-1$ if the object in question is the desired one, and
1 otherwise. Suppose now we have $N$ objects with no special order, for which $f(n)$ returns $-1$ for a single object only.

On a classical computer we cannot do better than a simple linear search, starting
with the first object. Assuming a homogeneous distribution, the probability for finding
the desired object in the first try is $1/N$, for finding it in two trials it is $(N-1)/N \times
1/(N-1) = 1/N$, etc. The only special case is the penultimate one, because if we
have not found the object after $N-1$ trials, we know for certain it is the last one.
Thus the average numbers of trials is given by

$$
\sum_{\text{trials}=1}^{N-2} \frac{\text{trials}}{N} + 2 \frac{N-1}{N} = \frac{N^2 + N - 2}{2N},
$$

(1.23)
which goes like $O(N)$ for large $N$, as is to be expected. The total execution time of a program based on this algorithm scales like $O(N)$ times the average time it takes to calculate $f(n)$.

How can we do better on a QC? This question is answered by Grover [Gro96]. Armed with the tricks we have described so far, the first thing we have to do is define a unitary operation $F$, such that

$$F|n\rangle = f(n)|n\rangle,$$

and apply this to a uniform superposition of states, such that

$$\frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} |n\rangle \rightarrow F \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} |n\rangle = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f(n)|n\rangle,$$

assuming of course that the preprocessing, i.e., calculating $f(n)$, is done on the QC. Otherwise we would have to calculate $f(n)$ for $N$ different values of $n$ in a sequential manner, which would make our algorithm use $O(N)$ steps again.

Simply measuring the state of the system at this stage is evidently of no use, since the probability for finding the right state is again given by $1/N$. We need to increase the probability for finding the state with amplitude $-1$ and on a QC we can only do this through a unitary operation.

Let us denote this unitary operation by $D$. Considering the symmetry of the problem, $D$ should not depend on the manner in which we labelled our objects. The basis state $|n\rangle$ is transformed into

$$D|n\rangle = \sum_{m=0, m \neq n}^{N-1} D_{nm}|m\rangle + D_{nn}|n\rangle.$$

But $n$ is just a label, so the previous equation should not depend on it. Furthermore, within the sum, we could have changed the labelling of the other states also. Combining these ideas we see that we should be able to write $D$ as

$$D = \lambda \sum_{n=0}^{N-1} |n\rangle\langle n| + \Gamma \sum_{n,m=0}^{N-1} |n\rangle\langle m|.$$

We want $D$ to be unitary, so we have

$$D^\dagger D = |\lambda|^2 \sum_{n=0}^{N-1} |n\rangle\langle n| + (\lambda \Gamma^* + \lambda^* \Gamma) \sum_{n,m=0}^{N-1} |n\rangle\langle m| + |\Gamma|^2 \sum_{n,m,k,l=0}^{N-1} |n\rangle\langle m|\langle k|\langle l|$$

$$= \sum_{n=0}^{N-1} |n\rangle\langle n|,$$

(1.27)
which leads to the equations

\[ |\lambda|^2 = 1, \]
\[ \lambda \Gamma^* + \lambda^* \Gamma + N|\Gamma|^2 = 0. \]  

Since we are only interested in the final probability, i.e., the absolute square of the amplitude, we can take out an overall phase factor, and choose \( \lambda = -1 \). The first equation is now automatically satisfied and with \( \Gamma = \gamma \exp ir \) the second equation reduces to

\[ \gamma = \frac{2}{N} \cos r, \]

where \(-\pi/2 < r \leq \pi/2\), since \( \gamma \geq 0 \). Now we return to our goal: we want to increase as much as possible the probability for finding the state for which \( f(n) = -1 \).

Let us first generalise this requirement a little. Suppose the state of our QC is such that the amplitude of the state \(|\text{needle}\rangle\) we want to find is \(-a\) and that of the other states is \(b\) where \(0 \leq a \leq 1\) and we have \((N - 1)b^2 + a^2 = 1\). Hence the state of the QC is given by

\[ F|\Phi_0\rangle = b \sum_{n=0, n \neq \text{needle}}^{N-1} |n\rangle - a|\text{needle}\rangle, \]

where \(|\Phi_0\rangle\) is the initial state of the QC, i.e., the uniform superposition with \(a = b = 1/\sqrt{N}\). Now we apply \(D\) and look at the resulting amplitude \(a'\) of the state \(|\text{needle}\rangle\) only

\[ a' = -1(-a) + \frac{2}{N} \left( (N - 1) \sqrt{\frac{1 - a^2}{N - 1}} - a \right) e^{ir} \cos r, \]

assuming that \(b = \sqrt{(1 - a^2)/(N - 1)}\) is the positive root. The probability for finding the state \(|\text{needle}\rangle\) upon measurement of the QC is given by

\[ |a'|^2 = a^2 + \frac{4}{N^2} \left( \sqrt{(1 - a^2)(N - 1)} - a \right) \times \left[ \sqrt{(1 - a^2)(N - 1)} + (N - 1)a \right] \cos^2 r. \]

We want to increase the probability, so \(|a'|^2 > a^2\). This gives the following restriction

\[ \sqrt{(1 - a^2)(N - 1)} - a > 0. \]

This restriction is independent of our choice for \(r\), i.e., \(D\). If this inequality is not satisfied, there is no unbiased unitary operation that increases the amplitude of the
state $|\text{needle}\rangle$. Thus once the amplitude has grown beyond $\sqrt{(N-1)/N}$, it can no longer be increased.

Eq. (1.33) determines $D$, because the optimal value (maximising $|a'|^2$) for $r$ is 0, hence $D$ can be written as

$$D = -\sum_{n=0}^{N-1} |n\rangle\langle n| + \frac{2}{N} \sum_{n,m=0}^{N-1} |n\rangle\langle m|.$$  \hspace{1cm} (1.35)

If we look closely at this transformation, we see that for every basis state it does

$$\text{amplitude} \rightarrow -\text{amplitude} + 2 \times (\text{average amplitude}),$$  \hspace{1cm} (1.36)

hence it reflects the amplitude with respect to the average. The effect of this transform-

![Diagram showing the effect of the unitary operations $F$ and $D$ of Grover's algorithm on a list of 4 items, represented by a uniform superposition of 4 basis states. The desired object is located in position 2.](image)

**Figure 1-3:** The effect of the unitary operations $F$ and $D$ of Grover’s algorithm on a list of 4 items, represented by a uniform superposition of 4 basis states. The desired object is located in position 2.

formation is shown in Fig. 1-3. We can write the effect of several consecutive $D \cdot F$ transformations as a recursion relation for the amplitudes $a$ and $b$. Let us denote the amplitudes after $t$ applications of $D \cdot F$ by $a_t$ and $b_t$ respectively. This gives

$$a_0 = \frac{1}{\sqrt{N}},$$  \hspace{1cm} (1.37a)

$$b_0 = \frac{1}{\sqrt{N}},$$  \hspace{1cm} (1.37b)

$$a_{t+1} = a_t + \frac{2}{N} ((N-1)b_t - a_t),$$  \hspace{1cm} (1.37c)

$$b_{t+1} = -b_t + \frac{2}{N} ((N-1)b_t - a_t).$$  \hspace{1cm} (1.37d)

We can solve this relation with the following matrix equation

$$\begin{pmatrix} a_t \\ b_t \end{pmatrix} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 2(N-1)/N \\ -2/N & 1 + 2(N-1)/N \end{pmatrix} \left( \begin{array}{c} 1 \\ 1 \end{array} \right),$$  \hspace{1cm} (1.38)
where
\[
\sin z = \frac{1}{\sqrt{N}}, \quad \cos z = \sqrt{1 - \frac{1}{N}}, \quad \tan z = \frac{1}{\sqrt{N} - 1}.
\] (1.39)

Finally, this gives the amplitude of the state \(|\text{needle}\rangle\) after \(t\) consecutive applications of \(D \cdot F\)
\[
a_t = \sin((2t + 1)z).
\] (1.40)

For large \(N\) we have \(z \approx 1/\sqrt{N}\). Thus if we want to make the probability for finding the state \(|\text{needle}\rangle\) of order 1, we need to apply \(D \cdot F\) exactly \(t\) times with
\[
t = \left\lfloor \frac{\pi}{4 \arcsin \left(\frac{1}{\sqrt{N}}\right)} - \frac{1}{2} \right\rfloor = \left\lfloor \frac{\pi \sqrt{N}}{4} - \frac{1}{2} \right\rfloor.
\] (1.41)

We can conclude that on a QC finding a needle in a haystack takes \(O(\sqrt{N})\) steps, which is only polynomially faster than on a classical computer.

There remains one thing we have to show: \(D\) as defined in (1.35) is not a local unitary transformation, i.e., it acts on all the qubits at the same time. Therefore we have to find a way to decompose \(D\) into local unitary transformations, i.e., one or two qubit operations.

For simplicity, let us assume \(N = 2^L\) is a power of two. In matrix notation, we have
\[
D = -\mathbb{1} + \frac{2}{N} \begin{pmatrix}
1 & 1 & \cdots & 1 \\
1 & 1 & \cdots & 1 \\
\vdots & \vdots & \ddots & \vdots \\
1 & 1 & \cdots & 1
\end{pmatrix}
= -\mathbb{1} + \frac{2}{N} \begin{pmatrix}
1 \\
1 \\
\vdots \\
1
\end{pmatrix} \cdot \begin{pmatrix}
1 & 1 & \cdots & 1
\end{pmatrix}.
\] (1.42)

The last part of this expression is the matrix product of the uniform superposition and its hermitian conjugate. We know that this uniform superposition can be written as \(W_T|0\rangle\), hence we can write
\[
D = -\mathbb{1} + 2 (W_T|0\rangle \langle 0|) \cdot \mathbb{1} = W_T \cdot R \cdot W_T,
\] (1.43)

where we used \(W_T = W_T^\dagger\). \(R\) is diagonal and merely a conditional phase shift, i.e., all the states \(|n\rangle\) are transformed to \(-|n\rangle\), except for \(n = 0\). \(W_T\) takes \(L\) local\(^3\) unitary

\(^3\)By local we mean a one or two qubit operation.
Quantum Computers

operations (see Eq. (1.13)). Grover [Gro96] calls $R$ a local operation, but even though $R$ is a rather trivial matrix, it is not local. This is due to the fact that the conditional phase flip depends on all the qubits because only the amplitude of the state with all qubits in the zero position is not flipped. Thus an OR-like operation between all the qubits is needed to decide if the amplitude gets multiplied by a factor $-1$. For this total OR operation we need $\mathcal{O}(\log^2 N)$ operations.

So if we now combine all the steps so far, we need

$$\mathcal{O}(\sqrt{N}) \times \mathcal{O}(\log^2 N),$$

(1.44)

steps to perform the whole Grover algorithm. Of course we still have to multiply this with the worst case (not the average, like in the classical algorithm) time it takes to calculate $f(n)$, needed to define $F$.

### 1.1.3 Number Factoring

Although the idea of quantum computation is much older than Shor’s algorithm for number factoring [Sho94], it is this contribution that got many more people interested in the field. This is because many of the public-key encryption schemes (see Intermezzo 1-1), which are used in most secure internet transactions today, are based on the assumption that factoring a product of two large primes is assumed to be a hard computational problem. The opposite, i.e., creating the product by multiplication, is an efficient process. This makes it possible to construct “one-way functions”, necessary for public-key encryption schemes. The first such scheme was developed by Rivest, Shamir, and Adleman [RSA78] and is now known under the almost household name: RSA (see Intermezzo 1-2).

The section is loosely based on Ref. [EJ96]. The key to “cracking” the encryption algorithm is to factor a number. Say the number we want to factor is $N$. Let us now look at the following integer equation [Rie85]

$$x^2 \mod N = 1,$$

(1.45)

which always has two trivial classes of solutions, because it means that $x^2 - 1$ has to be a multiple of $N$. So we have

$$x \equiv \pm 1 \text{ (modulo } N).$$

(1.46)

If $N$ is an odd prime, these are the only two solutions. If $x$ is composite, however, say a product of two primes ($k = 2$), then there are two more non-trivial solutions. Take for example, $N = 3 \times 7 = 21$; then Eq. (1.45) has two more solutions, namely $x \equiv \pm 8 \text{ (modulo } 21)$. 

Before explaining what public or asymmetric key encryption is, let us first look at the more familiar symmetric key encryption. Say two parties, Alice (A) and Bob (B), want to exchange information without a third party eavesdropping. They build a case with a lock to which they both have a copy of the same key. All the information they send to each other, they send in this locked briefcase. In principle this method will work very well, but there are a few drawbacks. The most important drawback, especially when it comes to communication over the Internet, is that without Alice and Bob physically meeting, a key has to be transferred between them before the first secure transaction. If an eavesdropper were there at the right time, he could copy the key, and from then on monitor all the communications between Alice and Bob, without their knowledge. Another problem is that this method does not scale very well: If Alice or Bob would want to communicate with other people too, they would make a case and corresponding set of distinct keys per communication channel. Since the number of possible communication channels is $N(N-1)/2$, where $N$ is the number of parties, this is very unpractical.

A very elegant way out of this is the public-key encryption system. A real life analogy of this system would be a box with a lock that fits two different keys, one of which can only turn the lock clockwise and another only counterclockwise (hence antisymmetric). With this protocol everybody has two keys: a public one and secret one. Public keys are exactly what their name says: public, i.e., everybody has access to them, even third parties. Let us denote the keys by $k^\text{pub}_A$, $k^\text{sec}_A$, $k^\text{pub}_B$, and $k^\text{sec}_B$. Now we assume that Alice has access to Bob's public key and there exists an encoding function $E(k^\text{pub}_B, M)$ ($E$ for encode) which generates an encrypted message out of the original message $M$. This is the message that Alice sends to Bob via an open non-secure channel. Of course one might say that since the message is based on $k^\text{pub}_B$ and $M$ and the full information of $M$ is still contained in $E(k^\text{pub}_B, M)$, there should exist a function $B$ (for break) with the property $B(k^\text{pub}_B, E(k^\text{pub}_B, M)) = M$. So everybody with the sufficient knowledge of $E$ and $k^\text{pub}_B$ could in principle construct such a function. Indeed this is always true, but the trick is to make sure that this $B$ would take ages, i.e., thousands of years, to calculate. But how will Bob be able to read the message then?

Well, if there exists a function $D$ (for decode), with the property $D(k^\text{sec}_B, E(k^\text{pub}_B, M)) = M$, which is very easy to calculate, Bob can decode the message easily. Everybody can have this $D$, but without the knowledge of Bob's secret key $k^\text{sec}_B$ it would be impossible to derive $M$. So, provided we find a suitable set of $E$ and $D$ and make sure that calculating $B$ takes a very long time, we have cured both problems describe above. Everybody can give everybody access to their public key (There is the “man-in-the-middle” problem, but this can be cured easily by key-certification authorities) and the number of necessary distinct keys scales with $N$.

This method also introduces a new feature, the possibility of digital signatures. Alice can also encrypt the message with her own secret key, i.e., send Bob the message $E(k^\text{sec}_A, M)$. Of course everybody would be able to read this message now, because everybody has access to Alice’s public key and can do $D(k^\text{pub}_A, E(k^\text{sec}_A, M)) = M$. A third party wanting to fake being Alice needs a function $F$ (for fake) with the property $D(k^\text{pub}_A, F(k^\text{pub}_A, M)) = M$, and this function $F$ is just as hard to make as $D$, described above.

Now Alice can combine both techniques to send an encrypted and authenticated message to Bob, using $E(k^\text{pub}_B, E(k^\text{sec}_A, M))$.

Intermezzo 1.1: Public-key encryption scheme.
In 1978 Rivest, Shamir, and Adleman [RSA78] developed an encryption method, now called RSA encryption. The core of the scheme is based on the assumption that multiplying two prime numbers is exponentially faster than doing the reverse, i.e., finding the prime factors of a large integer. Multiplication of two $L$ bit numbers takes $O(L \log(L) \log \log(L))$ operations [SS71] whilst the fastest known algorithm to factor a large integer takes $\exp O(L^{1/3})$ operations [LL93].

The RSA algorithm consists of a few computationally straightforward steps:

1. Find two large prime numbers, $p$ and $q$, and calculate their product $N = pq$.

2. Choose two numbers, $d$ and $e$, smaller than $N$ such that they have no factors in common with $\phi(N) = (p-1)(q-1)$ (see Eq. (1.52)) and are multiplicative inverses, i.e., $de \equiv 1 \pmod{\phi(N)}$ (see Appendix B.1).

From these numbers, the encryption functions we talked about in Intermezzo 1-1 are defined as follows:

$$E(\{N, e\}, M) \equiv M^e \pmod{N},$$

$$D(\{N, d\}, C) \equiv C^d \pmod{N}.$$

The secret and public keys used in the previous intermezzo are given by the pairs $k^\text{sec}_A = \{N, d\}$ and $k^\text{pub}_A = \{N, e\}$, respectively, and $M$ is the message (coprime to $N$) that Bob wants to send to Alice in the form of ciphertext $C$. Alice can distribute her public key without fear that her secret key $\{N, d\}$ is derived from the public one $\{N, e\}$ in polynomial time (although this has not been proven rigorously). We can check that the decryption function works by letting it act on the encryption function:

$$D(k^\text{pub}_A, E(k^\text{sec}_A, M)) = (M^e \pmod{N})^d \pmod{N} = M^{de} \pmod{N} = M^{1+x\phi(N)} \pmod{N} = ((M \pmod{N}) \times (M^{\phi(N)} \pmod{N})^x) \pmod{N} = M.$$

The last step follows from Euler's theorem (1.52) and from $M$ being coprime to $N$. These manipulations seem surprisingly simple, but there are no other known ways to derive $M$ from $C$ without the knowledge of $d$, except one that involves the factoring of $N$. The hardest part of the algorithm is finding large enough prime numbers ($p$ and $q$) that are not in the class of primes that give "easily" factorable numbers.

To make the method more concrete, let us illustrate it with an example. First Alice has to make a set of keys (she only has to do this only once). She generates two prime numbers, $p = 7919$ and $q = 9733$, randomly. This gives $N = pq = 77075627$ and $\phi(N) = (p-1)(q-1) = 77057976$. She can choose $e$ at random with no factors in common with $\phi(N)$, say $e = 44857$ and find $d$ by calculating $d = (x\phi(N) + 1)/e$ for increasing values of $x$ until $d$ is an integer. The first such value is $d = 19501129$. Now Alice can distribute her public key, $k^\text{pub}_A = \{77075627, 44857\}$, and keep her secret key, $k^\text{sec}_A = \{77075627, 19501129\}$, in a safe place. Suppose Bob wants to send the message $M = 'Hi'$ to Alice. Using ASCII coding ($H \equiv 72$, $i \equiv 105$), we have $M = 72 \cdot 256 + 105 = 18537$. Using Alice's public key he calculates $C = 18537^{44857} \pmod{77075627} = 48578037$ and passes this ciphertext to Alice. Alice now uses her secret key and calculates $48578037^{19501129} \pmod{77075627} = 18537 \equiv 'Hi'$. 

Intermezzo 1-2: Public-key encryption scheme based on the product of two large primes.
Let us look at the more general case, where \( N = n_1 n_2 \) and \( \gcd(n_1, n_2) = 1 \). In this case Eq.(1.45) can be split up in
\[
\begin{align*}
\text{Eq} (a) & \quad x^2 \mod n_1 = 1, \\
\text{Eq} (b) & \quad x^2 \mod n_2 = 1,
\end{align*}
\]
using the Chinese remainder theorem (see Appendix B.2), because the unique (for \( 0 \leq x^2 < N \)) solution to (1.47) is given by

\[
x^2 \equiv 1 \pmod{N}.
\]

The solutions to (1.47) can be split up into the solutions of the four different sets of equations,
\[
\begin{align*}
& x \equiv 1 \pmod{n_1}, \quad x \equiv -1 \pmod{n_1} \quad \{A, \}
& x \equiv 1 \pmod{n_2}, \quad x \equiv -1 \pmod{n_2} \quad \{B, \}
& x \equiv -1 \pmod{n_1}, \quad x \equiv 1 \pmod{n_2} \quad \{C, \}
& x \equiv 1 \pmod{n_1}, \quad x \equiv -1 \pmod{n_2} \quad \{D. \}
\end{align*}
\]

Now we can use the Chinese remainder theorem again to get the solutions to these sets of equations. \( A \) and \( B \) clearly lead to the trivial solutions, \( x \equiv \pm 1 \pmod{N} \). But \( C \) and \( D \) must lead to non-trivial solutions, i.e., \( x \equiv \pm a \pmod{N} \). From the base equation (1.45) we have
\[
(a - 1)(a + 1) \equiv 0 \pmod{N},
\]
therefore \( N \) divides \( (a - 1)(a + 1) \) but not \( a \pm 1 \) separately, because the solution of \( C \) and \( D \) can be chosen to be smaller than \( N \) (see Appendix B.2). So if \( |a| > 1 \), then from \( C \) and \( D \) we can see that \( a \pm 1 \) contains one and only one of the factors \( n_1, n_2 \). Thus finding the greatest common denominator of \( N \) and \( a \pm 1 \), i.e., calculating \( \gcd(N, a \pm 1) \), will give us a factor of \( N \).

We have replaced the task of finding factors of \( N \) with finding solutions to (1.45). But finding a solution to these equations directly, using Grover’s algorithm for instance, only gives a polynomial speedup (square root actually). Somehow we have to make use of the superposition to get some global property of a function, such as frequency.

Let us look at the equation
\[
f_{y,N}(a) \equiv y^a \mod N = 1,
\]
where \( y \) is a number coprime to \( N \). We know this equation has a solution (for \( a \), with \( y \) fixed), because Euler’s theorem (see [Sch90], Section 8.3) states that
\[
y^\phi(N) \mod N = 1,
\]
where \( y \) is a number coprime to \( N \) and \( \phi(N) \) is the number of integers smaller than \( N \) and coprime to \( N \). For a prime number \( \phi(p) = p - 1 \), for a product of two primes \( \phi(pq) = (p - 1)(q - 1) \), etc. Let us call the smallest solution to Eq. (1.51) \( r \). We then see that

\[
f_{y,N}(r + a) = y^{r+a} \mod N \\
= ((y^{r} \mod N)(y^{a} \mod N)) \mod N \\
= y^{a} \mod N \\
= f_{y,N}(a),
\]

so not only is \( r \) a solution to (1.51), it is also the period of the function \( f_{y,N}(a) \). Now, if this period \( r \) turns out to be even we have also found a solution to (1.45), namely \( x = y^{r/2} \). Since \( r \) is the smallest value for \( a \) that solves (1.51), \( r/2 \) cannot be a solution. Therefore \( r \) does not give the trivial solution to (1.45) with \( x = 1 \) (modulo \( N \)). Intuitively, we would guess that the probability of finding an even \( r \) and \( y^{r/2} \neq -1 \) (modulo \( N \)) is about one half. Actually, it has been proven that, for a general number \( N \) with prime decomposition

\[
N = p_1^{\alpha_1} p_2^{\alpha_2} p_3^{\alpha_3} \cdots p_k^{\alpha_k},
\]

given a random number \( y \) coprime to \( N \), we have

\[
P(r \text{ even and } y^{r/2} \neq -1 \text{ (modulo } N)) \geq 1 - \frac{1}{2^{k-1}}.
\]

(1.55)

For a complete proof we refer to Ref. [EJ96]. In the worst case—for a product of two primes—we have

\[
P(r \text{ even and } y^{r/2} \neq -1 \text{ (modulo } N)) \geq \frac{1}{2}.
\]

(1.56)

Thus the average number of trials necessary is smaller than or equal to two.

So everything boils down to finding the period of the function \( f_{y,N}(a) \). Since a period of a function is a *global* property, we might be able to use a QC to find it exponentially faster than a classical computer can. This was also the case with the Deutsch-Josza problem, which was also concerned with a global property. To find this period, we will need a quantum Fourier transform.

### 1.1.4 Quantum Fourier Transform

The fastest known algorithms to perform a Fourier transform on a classical computer are based on an implementation generally known as the Fast Fourier Transform (FFT). The number of operations necessary to perform a Fourier transform in this way scales like \( O(\log_2 q) \), where \( q \) is the length of the vector to be transformed. In our case \( q \) would have to be at least of the same order as \( N \), later on we will show that we
need a number which is at least \( N^2 \). Were we to use a classical computer to continue the algorithm we have described so far, it would be the FFT that would cause the algorithm to scale exponentially in the input size \( \log_2 N \) again.

It was shown by Shor [Sho94], however, that on a QC, a Fourier transform can be performed in \( \mathcal{O} \left( \log^2 N \right) \) steps. The algorithm is based on the FFT algorithm given by Knuth [Knu81].

We want a unitary operator, which can be written as follows:

\[
DFT_q = \frac{1}{\sqrt{q}} \sum_{a,c=0}^{q-1} e^{2\pi i ac/q} |c\rangle \langle a|.
\] (1.57)

Naively, we have to calculate this sum, containing \( q^2 \) terms, giving \( \mathcal{O}(q^2) \) operations. In what follows, it will be convenient to reverse the order of the output register \( |c\rangle \), hence we define

\[
|b\rangle = |b_{L-1}, b_{L-2}, \ldots, b_0\rangle = |c_0, c_1, \ldots, c_{L-1}\rangle.
\] (1.58)

Of course it would only take \( \lfloor L/2 \rfloor \) two-qubit operations to reverse the order of the bits in a \( L \) qubit register. We will continue to derive an algorithm that accomplishes

\[
DFT_{2^L} = \frac{1}{\sqrt{2^L}} \sum_{a,b=0}^{2^L-1} e^{2\pi i ac/2^L} |b\rangle \langle a|,
\] (1.59)

where \( c_i \equiv b_{L-1-i} \). Like with the FFT, we take \( q \) to be a power of two. Now we split the calculation into two parts

\[
DFT_{2^L} = \frac{1}{\sqrt{2^L}} \sum_{b=0}^{2^L-1} \sum_{a=0}^{2^L-1} \left( e^{2\pi i a b/2^L} |b\rangle \langle a| + e^{2\pi i (2b+1)/2^L} |b+2^L-1\rangle \langle a| \right)
= \frac{1}{\sqrt{2^L}} \sum_{b=0}^{2^L-1} \sum_{a=0}^{2^L-1} e^{2\pi i ac/2^L} \left( |0, b\rangle \langle a| + e^{\pi i a/2^L-1} |1, b\rangle \langle a| \right),
\] (1.60)

where we note that \( b \)'s with \( 0 \leq b < 2^L-1 \) correspond to the even \( c \)'s and \( b \)'s with \( 2^L-1 \leq b < 2^L \) to the odd \( c \)'s. We have also written the most significant bit of \( b \) explicitly, hence the notation change \( |b+2^L-1\rangle \rightarrow |1, b\rangle \).

So far we have split a sum of \( 2^L \) terms into two sums of \( 2^{L-1} \) terms, with no obvious gain. We see that \( a \) still runs from \( 0 \) to \( 2^L - 1 \), but now the exponents within the sum are periodic in \( a \) with period \( 2^{L-1} \), which means that we can rewrite the sum
We still have done nothing but rewrite the sum and use the periodicity property of the complex exponent. If we look closely at the beginning of the expression, it looks a lot like \( DFT_{2^{L-1}} \). Indeed, it might be possible to write

\[
DFT_{2^{L-1}} = (\mathbb{1}_{L-1} \otimes DFT_{2^{L-1}}) M,
\]

where \( \mathbb{1}_{L-1} \) denotes a unit operation on the most significant qubit and \( M \) is a transformation we are looking for.

Now we only have to get rid of the \( \exp(\pi ia/2^{L-1}) \) term to be left with \( DFT_{2^{L-1}} \). We can write this exponent as

\[
e^{\pi ia/2^{L-1}} = \prod_{l=0}^{L-2} e^{\pi ia2^l/2^{L-1}}.
\]

From (1.62) we see that we need to add an \( a \) dependent phase, if the \((L-1)\)-th qubit in the output is 1. In (1.63) we showed that this \( a \) dependence can be split up into an \( a_i \) dependence. We will therefore define the following two qubit operator

\[
B_{k,l} = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & e^{\pi i/2^{l-k}}
\end{pmatrix},
\]

acting on qubits \( k \) and \( l \). Inserting this result into (1.62), we obtain

\[
DFT_{2^{L-1}} = \frac{1}{\sqrt{2^{L-1}}} \sum_{a,b=0}^{2^{L-1}-1} e^{2\pi iac/2^{L-1}} \left( |0, b\rangle \langle a| + |a+2^{L-1}| \right)
\]
\[
+ |1, b\rangle \left[ e^{\pi ia/2^{L-1}} (|a| + e^{\pi i(a+2^{L-1})/2^{L-1}} |a+2^{L-1}|) \right]
\]
\[
= \frac{1}{\sqrt{2^{L-1}}} \sum_{a,b=0}^{2^{L-1}-1} e^{2\pi iac/2^{L-1}}
\]
\[
\times \left( |0, b\rangle \left( |0, a\rangle + e^{\pi ia/2^{L-1}} |1, b\rangle \right) \right) W_{L-1}.
\]

(1.65)
From the last expression we see that we have almost turned the expression into a recursive one, where $DFT_{2L}$ depends on $DFT_{2L-1}$. We can easily identify $DFT_{2L-1}$ and solve the recursion relation,

$$DFT_{2L} = (\mathbf{1}_{L-1} \otimes DFT_{2L-1}) \prod_{l=0}^{L-2} B_{l,L-1} W_{L-1}$$

$$= \prod_{k=0}^{L-1} \left( \prod_{l=0}^{k-1} B_{l,k} \right) W_k,$$

where the product is a product of matrices, so the order is important. We take

$$\prod_{k=l}^{u} M_k \doteq M_l \cdot M_{l+1} \cdot M_{l+2} \cdots M_{u-1} \cdot M_u.$$

(1.66)

In Fig. 1-4 a schematic example is given for $DFT_{16}$, i.e., a Fourier transform on four qubits.

We have managed to perform a Fourier transform of $q = 2^L$ “numbers” in $L$ one-qubit operations and $L(L - 1)/2$ two-qubit operations, i.e., $L(L + 1)/2$ operations in total. In terms of $q$ this would be $O(\log_2^2 q)$ operations.

Actually the algorithm is not much different from the classical FFT. So where is the exponential speedup? For a general Fourier transform there is none, as we will show. Only in special cases can we use the exponential speedup of the QC.

Assume our input state is the following:

$$|\text{input}\rangle = \sum_{a=0}^{2^L-1} f(a)|a\rangle,$$

(1.68)
where \( f(a) \) is an arbitrary function of \( a \). Therefore, up to a global phase factor, this state could be called a representation of the function (information) \( f(a) \).

If we perform \( DFT_{2L} \) on this state, we get an output state with the following property:

\[
|\text{output} \rangle = DFT_{2L} |\text{input} \rangle = \frac{1}{\sqrt{2^L}} \sum_{a,b=0}^{2^L-1} e^{2\pi i ac/2L} f(a) |b \rangle = \sum_{b=0}^{2^L-1} F(c) |b \rangle, \quad (1.69)
\]

with

\[
F(c) = \frac{1}{\sqrt{2^L}} \sum_{a=0}^{2^L-1} e^{2\pi i ac/2L} f(a), \quad (1.70)
\]
i.e., the Fourier transform of \( f(a) \). As shown above, this operation would take \( L(L + 1)/2 \) steps. We could say now that the output state represents the Fourier transform of \( f(a) \) and we have performed a Fourier transform “exponentially faster” than a classical FFT.

However, there is a catch. The rules of quantum mechanics tell us that a single measurement of the output register (in the basis used here), just gives us a single number.

Say we are interested in knowing \( |F(c)|^2 \) only, for every \( 0 \leq c < 2^L \). Then for an arbitrary probability distribution, we would still need to perform the measurement, and therefore the whole algorithm \( O(2^L) \) times.

If \( f(a) \) is periodic, however, and we are interested in one property only, e.g., the period, there is a way out, as we will show in the next subsection.

### 1.1.5 Period Finding

We will now continue with Shor’s algorithm and try to find the period \( r \) of the function \( f_{y,N}(a) \). We choose \( y \) coprime\(^4\) to \( N \) at random. Let us prepare our QC such that it represents the function \( f_{y,N}(a) \), like we described in the last part of the previous subsection. A first guess would be to devise a QA that does

\[
|0 \rangle \rightarrow \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a \rangle \rightarrow \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |f_{y,N}(a)\rangle. \quad (1.71)
\]

There is a problem, however. Since \( f_{y,N}(a) \) is periodic with period \( r \) and \( r < q \), this function has no inverse on this interval, hence it cannot be represented by a unitary transformation. For the whole calculation to be reversible, we have to keep track of \( a \). One can show that if the algorithm is carried out as follows:

\[
|0, 0 \rangle \rightarrow \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a, 0 \rangle \rightarrow \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a, f_{y,N}(a)\rangle, \quad (1.72)
\]

\(^4\)If \( y \) was not coprime, we could stop the algorithm, because we would have found a factor of \( N \).
it is invertable and can therefore be implemented using a unitary transformation. The details of how to carry out this step are given by Beckman et al. [BCDP96].

The first register has to have $\log_2 q = L$ qubits and the second register needs $\lceil \log_2 N \rceil$. It is on the final state of (1.72) where we want to apply the Fourier transform as follows:

$$
\frac{1}{q} \sum_{g,d,a=0}^{q-1} e^{2\pi i d g / q} (|g\rangle \langle d| \otimes 1) |a, f_{y,N}(a)\rangle = \frac{1}{q} \sum_{g,a=0}^{q-1} e^{2\pi i a g / q} |g, f_{y,N}(a)\rangle.
$$

By the $\otimes 1$ we mean to say that we leave the register containing $f$ untouched.

At this point we observe the state of the QC. We should make clear that we do not measure the probability distribution, which would take at least $O(2^L)$ repeated measurements. We measure the state of the machine once, giving a number $c$ for the first register and a number $\zeta$ for the second register. The probability for a single measurement to give the numbers $c$ and $\zeta$ is given by

$$
\left| \frac{1}{q} \sum_{g,a=0}^{q-1} e^{2\pi i a g / q} \langle c, \zeta | g, f_{y,N}(a) \rangle \right|^2 = \left| \frac{1}{q} \sum_{a=0}^{q-1} e^{2\pi i a c / q} \delta_{\zeta,f_{y,N}(a)} \right|^2.
$$

(1.74)

Now we define $k$ to be the lowest integer, such that $\zeta = f_{y,N}(k)$. Since $f$ is periodic in $a$ with period $r$, we have $0 \leq k < r$ and we define $m = \lfloor (q - k - 1)/r \rfloor + 1$. The delta function in (1.74) is only non-zero at values $a = b r + k$. This reduces the sum in (1.74) to

$$
\left| \frac{1}{q} \sum_{b=0}^{m-1} e^{2\pi i (br+k)c / q} \right|^2 = \left| \frac{1}{q} \sum_{b=0}^{m-1} e^{2\pi i b r c / q} \right|^2 = \left| \frac{1}{q} e^{2\pi i m c r / q} - 1 \right|^2
$$

$$
= \frac{\sin^2 \pi m c r / q}{q^2} \frac{\sin^2 \pi u r / q}{\pi u r / q}.
$$

(1.75)

In the expression above, we can replace $cr$ with $\{cr\}_q$, where $\{cr\}_q$ is defined such that it is congruent to $cr \mod q$, i.e., can be written as $cr - dq$, and has minimal absolute value. We can see that $-q/2 < \{cr\}_q \leq q/2$. In the unlikely case that $r$ is a divisor of $q$, the above expression reduces to a delta function with non-zero values only at points where $cr$ is a multiple of $q$, i.e., $\{cr\}_q = 0$. In general, however, this will not be the case, but the probability distribution will be strongly peaked around values for which $\{cr\}_q$ will be relatively small. Let us approximate the probability for values of $\{cr\}_q$ for which $\|\{cr\}_q\| \leq r/2$. The probability to find such a state is given by

$$
P = \frac{\sin^2 \pi m u r / q}{q^2 \sin^2 \pi u r / q},
$$

(1.76)
where \( u \) is a number between \(-1/2\) and \(1/2\). Because of the definition of \( k \) and \( m \), we have \(|mr - q| < r\). We define \( s = mr - q \) (one can check that \(|s| < r\)) and write

\[
\mathcal{P} = \frac{\sin^2 (\pi u (1 + s/q))}{q^2 \sin^2 \frac{\pi ur}{q}}. \tag{1.77}
\]

We want to find a lower limit for this probability. For the numerator we have

\[
\sin^2 (\pi u (1 + s/q)) \geq (2u)^2 \sin^2 \left(\frac{\pi (1 + s/q)}{2}\right) \geq 4u^2 \left(1 - \frac{\pi^2 r^2}{4q^2}\right), \tag{1.78}\]

as long as \( r/q \leq 1 \), and for the denominator we have

\[
q^2 \sin^2 \frac{\pi ur}{q} \leq \pi^2 u^2 r^2. \tag{1.79}\]

Combining the last two inequalities, we find that the probability of finding a state labelled by \( c \), and \( \|\{cr\}_q\| \leq r/2 \) is

\[
\mathcal{P} = \frac{\sin^2 (\pi u (1 + s/q))}{q^2 \sin^2 \frac{\pi ur}{q}} \geq \frac{4}{\pi^2 r^2} - \frac{1}{q^2}. \tag{1.80}\]

Suppose now we have measured a number \( c \) that satisfies this requirement. Then there exists a number \( d \), such that

\[
|cr - dq| \leq \frac{r}{2}, \tag{1.81}\]

or, written differently

\[
\left|\frac{c}{q} - \frac{d}{r}\right| \leq \frac{1}{2q}. \tag{1.82}\]

In the case where \( r \) is a divisor of \( q \), i.e., \( \{cr\}_q \) is zero, we can easily find \( d \) and \( r \) by dividing \( c/q \) down to the lowest fraction (provided \( r \) is coprime to \( d \)). In general, however, the only thing we have is (1.82). We could just look for two numbers \( d \) and \( r \), such that this inequality holds, but searching naively would just take \( O(r) \equiv (N) \) steps again. There is a way, however, to find \( d/r \) in polynomial time, using continued fractions [HW79, Knu81]. We use a theorem that says that given a positive real number \( \xi \) and two positive integers \( d \) and \( r \)

\[
\left|\frac{\xi - d}{r}\right| \leq \frac{1}{2r^2}, \tag{1.83}\]

then the rational number \( d/r \) is a convergent (see Appendix B.3) of the continued fraction expansion of \( \xi \). In order to satisfy this equation, we need \( q \geq r^2 \). We do not know \( r \) yet, but we know that \( r < N \). So if \( q \geq N^2 \) then (1.83) is satisfied, provided that (1.82) is satisfied.
Once we have found the convergent using the technique described in Appendix B.3 for which \((1.82)\) holds, we have found \(d/r\) in lowest terms. If \(d\) also happens to be coprime to \(r\), then we have found that \(r\). \(d\) is a random number smaller than \(r\). The number of integers smaller than \(r\) that are coprime to \(r\) is given by Euler’s totient function \(\phi(r)\) \cite{Knu81, HW79, Sch90}. And from (1.74), on we calculated the probability for finding \(c\) and \(\zeta\). There are, however, \(r\) possible values for \(\zeta\) that lead to \(|\{cr\}_q| \leq r/2\) with the same probability. Moreover, we could have sufficed with measuring \(c\) alone, ignoring the second register. So the probability for finding a value for \(c\) from which we can deduce \(r\) is bounded from below by

\[
\mathcal{P}_L = r\phi(r) \left( \frac{4}{\pi^2 r^2} - \frac{1}{q^2} \right). \tag{1.84}
\]

For Euler’s totient function we have

\[
\frac{\phi(r)}{r} \geq \frac{e^\gamma - \delta(r)}{\log \log r}, \tag{1.85}
\]

where \(\gamma\) is Euler’s constant and \(\delta(r)\) is a monotone decreasing sequence of reals converging to zero. Therefore, the probability of success of a single run of this algorithm is of \(O\left(1/\log \log r\right)\). Finally, we can estimate the average number of operations to be

\[
(W_H \underbrace{O(\log N)}_{\text{W_H}} + O(\log^3 N) + O(\log^2 N)) \times (QFT \underbrace{O(\log \log N)}_{\text{QFT}}) = O(\log^3 N \log \log N),
\]

and \(O(\log N)\) memory space. An explicit example of the whole algorithm is given in Fig. 1-5.
Let us assume we want to factor $N = 15$. Now we have to choose a $q = 2^L$ with $N^2 \leq q$. Take for instance $L = 8$, $q = 256$. This means we need an 8 qubit register to store $q$ and a 4 qubit register to store $y^a \mod N$. So we need a 12 bit QC where we write the base functions in terms of $|s, t\rangle$, where $s$ is an 8 bit number with $0 \leq s < 256$ and $t$ is a 4 bit number with $0 \leq t < 16$. We initialize the QC with the state

$$|\Phi_0\rangle = |0, 0\rangle$$

and perform Walsh-Hadamard operations on the $s$ register, such that

$$|\Phi_1\rangle = \frac{1}{16} \sum_{a=0}^{255} |a, 0\rangle.$$ 

Now we have to choose a random $y$ with $1 < y < 15$ coprime to 15. Take $y = 8$, for instance. Then we perform the calculation of $8^a \mod 15$ and store it in the $t$ register. The state of the QC is now

$$|\Phi_2\rangle = \frac{1}{16} \sum_{a=0}^{255} |a, 8^a \mod 15\rangle.$$

One can verify that the period of $8^a \mod 15$ is 4, and that the first 4 values are 1, 8, 4, and 2. Thus we can rewrite $|\Phi_2\rangle$ as

$$|\Phi_2\rangle = \frac{1}{16} \sum_{b=0}^{63} (|4b, 1\rangle + |4b + 1, 8\rangle + |4b + 2, 4\rangle + |4b + 3, 2\rangle).$$

The next step is to perform a discrete quantum Fourier transform on the state $\Phi_2$. This gives a new state

$$|\Phi_3\rangle = \frac{1}{256} \sum_{g=0}^{255} \sum_{b=0}^{63} e^{2\pi igb/256} (|g, 1\rangle + e^{2\pi ig/256} |g, 8\rangle + e^{4\pi ig/256} |g, 4\rangle + e^{6\pi ig/256} |g, 2\rangle).$$

The sum over $b$ gives rise to a delta function; it is only non-zero if $g$ is a multiple of $q/r = 64$, in that case it equals $q/r = 64$. So the final state will be

$$|\Phi_3\rangle = \frac{1}{4} |0\rangle \otimes (|1\rangle + |8\rangle + |4\rangle + |2\rangle) + |64\rangle \otimes (|1\rangle - |8\rangle + |4\rangle - |2\rangle) + |128\rangle \otimes (|1\rangle + |8\rangle + |4\rangle + |2\rangle) + |192\rangle \otimes (|1\rangle - |8\rangle + |4\rangle - |2\rangle).$$

We now decide to measure the first register only. Although both registers are still a bit entangled, for the probability distribution for the first register, this is of no influence and there are four equally probable outcomes:

0) Bad luck, we have to start over again; it tells us nothing about $r$

64) Calculate $c/q = d/r$ and cancel to lowest fraction; this gives 1/4. Assume $\gcd(d, r) = 1$ and conclude $r = 4$. Calculate $\gcd(8^{r/2} - 1, 15) = \gcd(63, 15) = 3$. Found factor!

128) Calculate $c/q = d/r$ and cancel to lowest fraction; this gives 1/2. Assume $\gcd(d, r) = 1$ and conclude $r = 2$. Calculate $\gcd(8^{r/2} - 1, 15) = \gcd(7, 15) = 1$ and $\gcd(8^{r/2} + 1, 15) = \gcd(9, 15) = 3$. Found factor! But this really was a coincidence, since $r$ was not 2, because the assumption $\gcd(d, r) = 1$ was false.

192) Calculate $c/q = d/r$ and cancel to lowest fraction; this gives 3/4. Assume $\gcd(d, r) = 1$ and conclude $r = 4$. Calculate $\gcd(8^{r/2} - 1, 15) = \gcd(63, 15) = 3$. Found factor!

Thus two out of four outcomes gave the right answer in the predicted manner.

**Figure 1-5:** Factoring 15.