УДК 512.6, 517.9, 519.6

EFFICIENT SIMULATION OF QUANTUM SEARCH ALGORITHMS ON A CLASSICAL COMPUTER. PT. 1: SHOR'S QUANTUM ALGORITHM FOR FACTORING

Ulyanov Sergey V.[1], Tyatyushkina Olga Yu.[2], Korenkov Vladimir V.[3]

[1]Doctor of Physical and Mathematical Sciences, professor;
Dubna State University;
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
Leading Researcher of LIT JINR;
Joint Institute for Nuclear Research;
6 Joliot-Curie Str., Dubna, Moscow region, 141980, Russia;
e-mail: ulyanovsv@mail.ru.

[2]PhD in Engineering Sciences, associate professor;
Dubna State University;
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
e-mail: tyatyushkina@mail.ru.

[3]Laboratory Director;
Joint Institute for Nnuclear Rresearch;
6 Joliot-Curie Str., Dubna, Moscow region, 141980, Russia;
Doctor of Technical Sciences, head of the Department;
Dubna State University;
19 Universitetskaya Str., Dubna, Moscow region, 141980, Russia;
e-mail: korenkov@jinr.ru.

The result of the proof of the effective implementation of the simulation on classical computers of quantum algorithms based on quantum algorithmic cells is demonstrated. The Shor factorization algorithm is considered as an example.

Keywords: quantum computing, Shor's quantum algorithm, quantum software engineering, classical accelerators of quantum computing.

For citation:
Ulyanov S., Tyatyushkina O., Korenkov V. Efficient simulation of quantum search algorithms on a classical computer. Pt 1: Shor's quantum algorithm for factoring. System Analysis in Science and Education, 2021;(1):58–80. Available from: http://sanse.ru/download/427.

ЭФФЕКТИВНОЕ МОДЕЛИРОВАНИЕ КВАНТОВЫХ ПОИСКОВЫХ АЛГОРИТМОВ НА КЛАССИЧЕСКОМ КОМЬЮТЕРЕ. Ч.1: КВАНТОВЫЙ АЛГОРИТМ ФАКТОРИЗАЦИИ ШОРА

Ульянов Сергей Викторович[1], Тятюшкина Ольга Юрьевна[2], Кореньков Владимир Васильевич[3]

[1]Доктор физико-математических наук, профессор;
Государственный университет «Дубна»;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
Ведущий научный сотрудник;
Объединенный институт ядерных исследований;
141980, Московская обл., г. Дубна, ул. Жолио-Кюри, д. 6;
e-mail: ulyanovsv@mail.ru.

[2]Кандидат технических наук, доцент;
Государственный университет «Дубна»;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: tyatyushkina@mail.ru.

[3]*Директор лаборатории;*
Объединенный институт ядерных исследований;
141980, Московская обл., г. Дубна, ул. Жолио-Кюри, д. 6;
Доктор технических наук, заведующий кафедрой;
Государственный университет «Дубна»;
141980, Московская обл., г. Дубна, ул. Университетская, 19;
e-mail: korenkov@jinr.ru.

Демонстрируется результат доказательства эффективной реализации моделирования на классических компьютерах квантовых алгоритмов на основе квантовых алгоритмических ячеек. В качестве примера рассматривается алгоритм Шора факторизации.

Ключевые слова: квантовые вычисления, квантовый алгоритм Шора, квантовая программная инженерия, классические ускорители квантовых вычислений.

Для цитирования:
Ульянов С. В., Тятюшкина О. Ю., Кореньков В. В. Эффективное моделирование квантовых поисковых алгоритмов на классическом комьютере. Ч.1: Квантовый алгоритм факторизации Шора // Системный анализ в науке и образовании: сетевое научное издание. 2021. № 1. С. 58–80. На англ. языке. URL : http://sanse.ru/download/427.

Introduction

One of the most radical changes that is being pursued in the last 20 years, is a change in the way that calculations are implemented physically. Rather than trying to avoid the quantum mechanical effects that manufacturers of processors face, researchers in the field of quantum computing try to harness these effects. Computers that are based on these quantum mechanical laws are collectively referred to as quantum computers, as opposed to classical computers that rely on the classical laws of physics.

There are, roughly speaking, two difficult moments that have to be overcome in order to realize a functional quantum computer. First of all, one needs to be able to construct a physical system that can be manipulated according to these quantum mechanical laws. Recently, headway has been made in this direction by researchers from IBM, as they have provided the scientific community with a first functional quantum computer. The experimental realization of the quantum computer, though, will not be the topic of this article.

The other challenge that needs to be overcome, is harnessing the full potential of such a quantum computer, when it is constructed. To this end, powerful quantum algorithms, we can use to build programs that can be run on quantum computers, need to be developed. Researchers that take up the challenge of devising these quantum algorithms will find that it is eminent to adopt the logic of quantum mechanics as an alternative form of reasoning, in contrast to the classical one we use on a daily basis.

This article will focus on covering the basis of quantum mechanics, and giving the reader insight in the alternative form of reasoning described above. These observations will be used to introduce quantum circuits. To this end, exemplary results will be covered. The first one shows that any quantum algorithm can be implemented using a finite set of component. Thereafter, the exemplary algorithm provided by Shor to factorize integers is covered.

Shor's algorithm is the most important algorithmic result in quantum computing. The algorithm builds on ideas that already appear in Deutsch and Jozsa's algorithm and in Simon's algorithm, and like these algorithms the basic ingredient of the algorithm is the Fourier Transform (FT) stated as follows:

Input:	*An integer N*
Output:	*A non-trivial factor of N, if exists*

Remark. There is no proof that there is no polynomial classical factorization algorithm. The problem is even not known to be NP-complete. However, factorization is regarded as hard, because many works have tried to solve it efficiently and failed. In 1994, Shor published a polynomial (in log(N)) QA for solving this problem. In fact, Shor presented a QA not for factoring, but for a different problem:

Order modulo *N*:	
Input:	*An integer N, and Y coprime to N*
Output:	*The order of Y, i.e. the minimal positive integer r such that* $Y^r = 1 \bmod N$.

In order to factor a number N it is enough to be able to find the order of x in ZN.

Let us consider theoretical aspects of Shor' algorithm.

1. Reduction of factorization to order-finding

Let us describe Shor's algorithm for finding the prime factors of a composite number *N*. Think of a large number such as one with 300 digits in decimal notation, since such numbers are used in cryptography. 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 $\lceil \log_2 N \rceil$. 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 an algorithm that factors *N* in a number of steps which is polynomial in *n*.

More technically, the question is: is there a factorization algorithm in the complexity class P?

Reduction of factorization of *N* to the problem of finding the order of an integer *x* less than *N* is as follows. If *x* and *N* have common factors, then *gcd*(*x*,*N*) gives a factor of *N*, therefore it suffices to investigate the case when *x* is coprime to *N*. The order of *x* mod *N* is defined as the least positive integer *r* such that $x^r \equiv 1 \bmod N$. If *r* is even, we can define *y* by $x^{r/2} \equiv y \bmod N$.

Remark. The above notation means that *y* is the remainder of $x^{r/2}$ divided by *N* and, by definition, $0 \le y < N$. Note that *y* satisfies $y^2 \equiv 1 \bmod N$, or equivalently $(y - 1)(y + 1) \equiv 0 \bmod N$, 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* (that yield *N* by multiplication).

So, *gcd*($y - 1$, *N*) and *gcd*($y + 1$, *N*) yield non trivial factors of *N* (*gcd* stands for the greatest common divisor). If *N* has remaining factors, they can be calculated applying the algorithm recursively.

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

$$2^4 \equiv 16 \bmod 21, \quad 2^5 \equiv 11 \bmod 21, \quad 2^6 \equiv 11 \times 2 \equiv 1 \bmod 21,$$

show that the order of 2 mod 21 is $r = 6$.

Therefore, $y \equiv 2^3 \equiv 8 \bmod 21$. $y - 1$ yields the factor 7 and $y + 1$ yields the factor 3 of 21.

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 coprime to *N*. In the latter case, if *x* satisfies the conditions (1) its order *r* is even, and (2) $0 < y - 1 < y + 1 < N$, 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 2 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*. In the worst case (*N* has 2 factors), the probability is greater than or equal to ½.

At first sight, it seems that we have just described an efficient algorithm to find a factor of *N*. That is not true, since it is not known an efficient classical algorithm to calculate the order of an integer *x* mod *N*. On the other hand, there is (after Shor's work) an efficient QA.

Let us describe it.

Theorem*: If there is a polynomial (in* log *N) algorithm to solve order modulo N, then there is a polynomial algorithm to solve factorization*.

Proof: We show a (classical probabilistic) reduction from FACTORING to ORDER. That is, we assume we have a black box algorithm for finding the order of a given integer *x* (coprime to *N*), in Z_N.

Lemma 1. *A solution to the equation* $x^2 \equiv 1 (\mathrm{mod}\ N)$ *with* $x \neq -1, +1$ *gives a factor of N.*
This is true since this equality implies $(x + 1)(x-1) = mN$ for some integer m, and both $1 < x-1, x+1 < N-1$. This means that either $\gcd(x-1, N) \neq 1, N$ or $\gcd(x-1, N) \neq 1, N$, as N divides $(x + 1)(x -1)$. This proves the lemma.

Now pick a random (nonzero) element of Z_N. If $gcd(x, N) > 1$, then we have found a factor of *N*. If not, find the order *r* of *x* in Z_N. Assume we are lucky, and *r* is even and also $y := x^{r/2} \neq -1, +1$. Then *y* is a nontrivial solution to the equation $y^2 \equiv 1 (\mathrm{mod}\ N)$, which gives us a factorization of *N* by the previous lemma. What is the probability that we are lucky?

Lemma 2. *Let* $N = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \ldots \cdot p_m^{\alpha_m}$, *where* p_i *are distinct primes, N is odd and not prime (i.e.* $m > 1$, $p_i \neq 2$*). Choose* $x \in Z_N$ *randomly. Then*

$$Prob\ (ORD_{Z_N}(x)\ \text{is even, and}\ x^{r/2} \neq -1, +1) \geq 1 - 2^{-m} \geq 1/2$$

We will not prove this lemma (proof can be found in Nielsen-Chuang (2000)). This shows that a randomly chosen *x* will give a factor of *x* using the procedure described above with probability ≥ ½. This can, as usual, be amplified by repeating this process several times. We have thus proved theorem.

Example. Reduction to Period-Finding. Shor's algorithm finds a factor by finding the period of some sequence. We first show how efficient period-finding suffices for efficient factoring. Suppose we want to find factors of the composite number $N > 1$. Randomly choose some integer $x \in \{2, \ldots, N-1\}$. Consider the sequence:

$$1 = x^0 \bmod N, x^1 \bmod N, x^2 \bmod N, \ldots$$

This sequence will cycle after a while: there is a least $0 < r \leq N$ such that $x^r = 1 \bmod N$. This *r* is called the *period* of the sequence. It can be shown that with probability ≥ ¼, *r* is even and $x^{r/2} + 1$ and $x^{r/2} - 1$ are not multiples of *N*. In that case:

x^r		$1 \bmod N$	$\Leftrightarrow$
$(x^{r/2})^2$		$1 \bmod N$	$\Leftrightarrow$
$(x^{r/2}+1)(x^{r/2}-1)$		$0 \bmod N$	$\Leftrightarrow$
$(x^{r/2}+1)(x^{r/2}-1)$		*kN*	*for some k.*

Not that $k > 0$ because both $x^{r/2} + 1 > 0$ and $x^{r/2} - 1 > 0$ ($x > 1$). Hence $x^{r/2} + 1$ or $x^{r/2} - 1$ will share a factor with *N*. Because $x^{r/2} + 1$ and $x^{r/2} - 1$ are not multiples of *N* this factor will be <*N*, and in fact *both* these numbers will share a non-trivial factor with *N*. Accordingly, if we have *r* then we can efficiently (in $\tilde{O}(\log N)$ steps) compute the greatest common divisors $gcd(x^{r/2} + 1, N)$ and $gcd(x^{r/2} - 1, N)$, and both of these two numbers will be non-trivial factors of *N*. If we are unlucky, we might have chosen an *x* that does not give a factor (which we can detect efficiently), but trying a few different random *x* gives a high probability of finding a factor.

Thus, the problem of factoring reduces to finding *r*.

We will show below how the QFT enables us to do this.

2. Shor's algorithm for finding the order

Given N choose a random (with the uniform distribution) m ($1 < m \le N$). We assume $gcd(m, N) = 1$, otherwise we would already know a divisor of N. We want to find the order of m, i.e. the least integer r such that

$$m^r \equiv 1(\mathrm{mod}\ N).$$

Fix some q of the form $q = 2^s$ with $N^2 \le q < 2N^2$. The algorithm will use the Hilbert space

$$H = C^q \otimes \boxed{C^{N_1}} \otimes C^k$$

where C^q and C^{N_1} are two quantum registers which hold integers represented in binary. Here N_1 is an integer of the form $N_1 = 2^l$ for some l such that $N \le N_1$. There is also the work space C^k to make arithmetical operations.

We will not indicate it explicitly. If $a = a_0 2^0 + 2^1 a_1 + 2^2 a_2 + ... + 2^s a_s$ is the binary representation $(a_i = 0,1)$ of an integer a then we write $|a\rangle = |a_0\rangle \otimes ... \otimes |a_s\rangle$ where $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is the basis in the two-dimensional complex space C^2. We have the data (N, m, q).

As abovementioned, the algorithm for finding the order r of m consists of 5 steps:

1.	Preparation of quantum state.
2.	Modular exponentiation.
3.	Quantum Fourier transform.
4.	Measurement.
5.	Computation of the order at the classical computer.

Description of the algorithm

Step 1: Preparation of quantum state. Put the first register in the uniform superposition of states representing numbers $a(\mathrm{mod}\ q)$. The quantum computer will be in the state $|\psi_1\rangle = \frac{1}{\sqrt{q}} |a> \otimes |0>.$

Step 2: Modular exponentiation. Compute $m^a(\mathrm{mod}\ N)$ in the second register. This leaves the quantum computer in the state

$$|\psi_2\rangle = \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a> \otimes |m^a \left(\mathrm{mod}\ N\right)>.$$

Step 3: *Quantum Fourier transform* (QFT). Perform the QFT on the first register, mapping $|a>$ to $\frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} e^{2\pi i a c/q} |c>.$ The quantum computer will be in the state

$$|\psi_3\rangle = \frac{1}{q} \sum_{c=0}^{q-1} \sum_{c=0}^{q-1} e^{2\pi i a c/q} |\tilde{n}\rangle \otimes |m^a \left(\mathrm{mod}\ N\right)\rangle.$$

Step 4: *Measurement.* Make the measurement on both registers $|\tilde{n}\rangle$ and $|m^a \left(\mathrm{mod}\ N\right)\rangle$.

Remark To find the period r we will need only the value of $|\tilde{n}\rangle$ in the first register but for clarity of computations we made the measurement on the both registers. The probability $P(c, m^k(\mathrm{mod}\ N))$ that the quantum computing ends in a particular state

$$\left|c; m^k \left(\mathrm{mod}\, N\right)\right\rangle = \left|\tilde{n}\right\rangle \otimes \left|m^k \left(\mathrm{mod}\left(N\right)\right)\right\rangle$$

according to quantum mechanics law is

$$P(c, m^k (\mathrm{mod}\, N)) = \left| < m^k (\mathrm{mod}\, N); c \left| \psi_3 \right> \right|^2$$

where we can assume $0 \le k < r$.

We will use the following Theorem, which shows that the probability $P(c, m^k (\mathrm{mod}\, N))$ is large if the residue of $rc (\mathrm{mod}\, q)$ is small. Here r is the order of m in the group $(Z / NZ)^*$ of residues of modulo N.

Theorem: If there is an integer d such that $-\frac{r}{2} \le rc - dq \le \frac{r}{2}$ and N is sufficiently large then $P(c, m^k (\mathrm{mod}\, N)) \ge \frac{1}{3r^2}$.

Step 5: *Computation of the order at the classical computer.* We know N, c and q and we want to find the order r. Because $q > N^2$, there is at most one fraction d/r with $r < N$ that satisfies the inequality. We can obtain the fraction d/r in lowest terms by rounding c/q to the nearest fraction having a denominator smaller than N. To this end we can use the continued fraction expansion of c/q and Theorem.

We will introduce the following theorem which summarizes main results of the quantum algorithm for finding the order.

Theorem: If the integer N is sufficiently large then by repeating the first four steps of the algorithm for finding the order $O(\log \log N)$ times one can obtain the value of the order r with the probability $\gamma > 0$ where the constant γ does not depend on N.

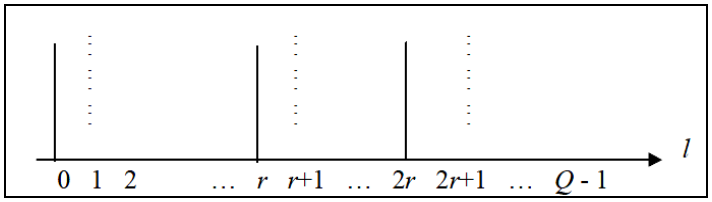
3. The quantum algorithm for order finding: Circuit work analysis

Again, we will work with two registers. The first one will hold a number between 1 to Q. (Q will be fixed later: it is much larger than N, but still polynomial in N). The second register will carry numbers between 1 to N. Hence the two registers will consist of $O(\log(N))$ qubits.

Let us now understand how this algorithm works. In the second step of the algorithm, all the numbers between 0 and $Q - 1$ are present in the superposition, with equal weights. In the third step of the algorithm, they are separated to sets, each has periodicity r. This is done as follows: there are r possible values written on the second register: $a \in \left\{Y^0, Y^1, ...Y^{r-1}\right\}$. The third state can thus be written as:

$$\frac{1}{\sqrt{Q}}\left(\left(\sum_{l=0 \left| Y^l = Y \right.}^{Q-1} \left|l\right\rangle \otimes \left|Y\right\rangle \right) + \left(\sum_{l=0 \left| Y^l = Y^2 \right.}^{Q-1} \left|l\right\rangle \otimes \left|Y^2\right\rangle \right) + ... + \left(\sum_{l=0 \left| Y^l = Y^r \right.}^{Q-1} \left|l\right\rangle \otimes \left|Y^r = 1\right\rangle \right) \right)$$

Note that the values l that give $Y^l = a$ have periodicity r: If the smallest such l is l_0, then $l = l_0 + r$, $l_0 + 2r$, … will also give $Y^l = a$. Hence each term in the brackets has periodicity r. Each set of l's, with periodicity r, is attached to a different state of the second register. Before the computation of Y^l, all l's appeared equally in the superposition. Writing down the Y^l on the second register can be thought of as giving a different "color" to each periodic set in [0, Q – 1]. Visually, this can be viewed as follows:



The measurement of the second register picks randomly one of these sets, and the state collapses to a superposition of *l's* with periodicity *r*, with an arbitrary shift l_0. Now, how to obtain the periodicity? The first idea that comes to mind is to measure the first register twice, in order to get two samples from the same periodic set, and somehow deduce *r* from these samples.

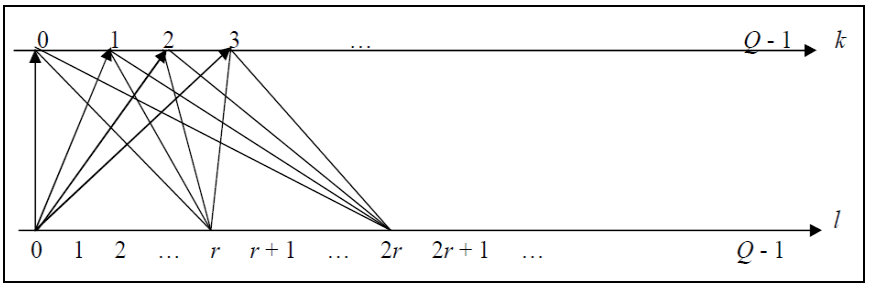
However, the probability that the measurement of the second register yields the same shift in two runs of the algorithm, i.e. that the same periodic set is chosen twice, is exponentially small.

How to gain information about the periodicity in the state without simply sampling it?

This is done by the FT. To understand the operation of the FT, we use a diagram again:

Step	Shor's algorithm
1	$\left\|\vec{0}\right\rangle \otimes \left\|\vec{0}\right\rangle$
2	Apply FT over Z_Q on the first register $\frac{1}{\sqrt{Q}}\sum_{l=0}^{Q-1}\left\|l\right\rangle \otimes \left\|\vec{0}\right\rangle$
3	Call subroutine which computes $\left\|l\right\rangle\left\|d\right\rangle \mapsto \left\|l\right\rangle\left\|d \oplus Y^l \bmod N\right\rangle$ $\frac{1}{\sqrt{Q}}\sum_{l=0}^{Q-1}\left\|l\right\rangle \otimes \left\|Y^l \bmod N\right\rangle$
4	Measure second register $\frac{1}{\sqrt{A}}\sum_{l=0\left\|Y^l=Y^{l_0}\right.}^{Q-1}\left\|l\right\rangle \otimes \left\|Y^{l_0}\right\rangle = \frac{1}{\sqrt{A}}\sum_{j=0}^{A-1}\left\|jr+l_0\right\rangle \otimes \left\|Y^{l_0}\right\rangle$
5	Apply FT over Z_Q on the first register $\frac{1}{\sqrt{Q}}\sum_{k=0}^{Q-1}\left(\frac{1}{\sqrt{A}}\sum_{j=0}^{A-1}e^{2\pi i(jr+l_0)k/Q}\right)\left\|k\right\rangle \otimes \left\|Y^{l_0}\right\rangle$
6	Measure first register.
7	Let k_1 be the outcome. Approximate the fraction $\frac{k_1}{Q}$ by a fraction with denominator smaller than *N*, using the (classical) method of continued fractions. If the denominator *d* doesn't satisfy $Y^d = 1 \bmod N$, throw it away. Else call the denominator r_1.
8	Repeat all the previous steps poly(log(N)) times to get r_1, r_2, ..
9	Output the minimal *r*.

Each edge in the diagram indicates that there is some probability amplitude to transform from the bottom basis state to the upper one.



Now we measure the first register, to obtain *k*. To find the probability to measure each *k*, we need to sum up the weights coming from all the *j's* in the periodic set.

$$\text{Prob}(k) = \left| \frac{1}{\sqrt{QA}} \sum_{j=0}^{A-1} e^{2\pi i k(jr+l_0)/Q} \right|^2 = \left| \frac{1}{\sqrt{QA}} \sum_{j=0}^{A-1} (e^{2\pi i k r/Q})^j \right|^2$$

Hence, in order to compute the probability to measure each *k*, we need to evaluate a geometrical series. Alternatively, the geometric series is a sum over unit vectors in the complex plane.

Exact periodicity. Let us assume for a second *exact periodicity*, i.e. that *r* divides *Q* exactly. Then *A* = *Q*/*r*. In this case, the above geometrical series is equal to zero, unless $e^{2\pi i k r / Q} = 1$. Thus, we measure with probability 1 only *k's* such that $kr = 0 \bmod Q$. This is where destructive interference comes to play: only "good" *k's*, which satisfy $kr = 0 \bmod Q$, remain, and all the others cancel out. Why are such *k's* "good" ? We can write *kr* = *mQ*, for some integer *m*, or *k*/*Q* = *m*/*r*. We know *k* since we have measured it. Therefore, we can reduce the fraction *k*/*Q*. If *m* and *r* are coprime the denominator will be exactly *r* which we are looking for. The probability for all "good" *k's* is the same, so *m* is chosen randomly between 0 to *r* – 1. By the prime number theorem, there are approximately $r/\log(r)$ primes smaller than *r*. Repeating the experiment a large enough number of times we will with very high probability eventually get *m* prime, i.e. coprime to *r*.

Example. r divides q (easy case). Assume we have picked a random *x* and we want to find the corresponding period *r*. We can always efficiently pick some smooth *q* such that $N^2 < q \le 2N^2$ (for instance take *q* a power of 2). The QFT for Z_q can be implemented using $O((\log q)^2) = O((\log N)^2)$ elementary gates. We will first assume that the unknown *r* divides *q*, in which case everything works out smoothly. It is known that in $\tilde{O}((\log N)^2)$ steps we can compute the transformation $|a\rangle|0\rangle \to |a\rangle|x^a \bmod N\rangle$ using the Schonhage-Strassen algorithm for fast multiplication. We now find *r* as follows. Start with $|0\rangle|0\rangle$, two registers of $\lceil \log q \rceil$ and $\lceil \log N \rceil$ zeroes, respectively. Apply the QFT to the first register to build $QFT := \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|0\rangle.$

1 register	$\lvert O_1 \rangle$	$\to$	$\lceil \log q \rceil$
2 register	$\lvert O_2 \rangle$	$\to$	$\lceil \log N \rceil$

Then compute $x^a \bmod N$ in quantum parallel: $\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle|x^a \bmod N\rangle$. Observing the second register gives some $x^s \bmod N$, with *s* < *r*. Note that because *r* divides *q*, the *a* of the form $a = jr + s (0 \le j < q/r)$ are exactly the *a* for which $x^a \bmod N$ equals the observed value $x^s \bmod N$. Thus, the first register collapses to a superposition of $|s\rangle, |r+s\rangle, |2r+s\rangle, ..., |q-r+s\rangle$ and the second register collapses to the classical state $x^s \bmod N$. We can now ignore the second register, and gave in the first: $\sqrt{\frac{r}{q}} \sum_{j=0}^{q/r-1} |jr+s\rangle.$ Apply the QFT again gives

$$\sqrt{\frac{r}{q}} \sum_{j=0}^{q/r-1} \sum_{b=0}^{q-1} e^{2\pi i \frac{(jr+s)b}{q}} |b\rangle = \sqrt{\frac{r}{q}} \sum_{b=0}^{q-1} e^{2\pi i \frac{sb}{q}} \left(\sum_{j=0}^{q/r-1} e^{2\pi i \frac{jrb}{q}} \right) |b\rangle .$$

Using that $\sum_{j=0}^{n-1} a^j = (1-a^n)/(1-a)$ for $a \ne 1$, we compute:

$$\sum_{j=0}^{q/r-1} e^{2\pi i\frac{jrb}{q}} = \sum_{j=0}^{q/r-1}\left(e^{2\pi i\frac{rb}{q}}\right)^j = \begin{cases} q/r & \text{if } e^{2\pi i\frac{rb}{q}} = 1 \\ \dfrac{1-\left(e^{2\pi i\frac{rb}{q}}\right)^{q/r}}{1-e^{2\pi i\frac{rb}{q}}} = \dfrac{1-e^{2\pi ib}}{1-e^{2\pi i\frac{rb}{q}}} = 0 & \text{if } e^{2\pi i\frac{rb}{q}} \neq 1 \end{cases}$$

Note that $e^{2\pi irb/q} = 1$ iff rb/q is an integer iff b is a multiple of q/r. Accordingly, we are left with a superposition where only the multiples of q/r have non-zero amplitude. Observing this final superposition gives some random multiple $b = cq/r$, with c a random number $0 \le c < r$. Thus, we get a b such that $\frac{b}{q} = \frac{c}{r}$, where b and q are known and c and $r$are unknown. There are $\phi(r) \in \Omega(r/\log\log r)$ numbers smaller than r which are coprime to r, so c will be coprime to r with probability $\Omega(1/\log\log r)$. Accordingly, an expected number of $O(\log\log N)$ repetitions of the procedure of this section suffices to obtain a $b = cq/r$ with c coprime to r. Once we have such a b, we can obtain r as the denominator by writing b/q in lowest terms.

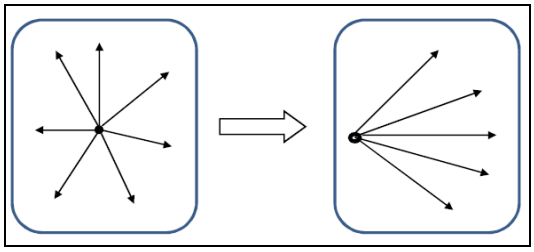
Example. r does not divide q (hard case). In case r does not divide q (which is actually quite likely), it can be shown that applying exactly the same algorithm will still yield with high probability a b such that $\left|\frac{b}{q} - \frac{c}{r}\right| \le \frac{1}{2q}$, with b, q known and c, r unknown. Two distinct fractions, each with denominator $\le N$, must be at least $1/N^2 > 1/q$ apart.

Remark. Consider two fractions $z = x/y$ and $z' = x'/y'$ with $y, y' \le N$. If $z \ne z'$ then $|xy' - x'y| \ge 1$, and hence $|z - z'| = |(xy' - x'y)/yy'| \ge 1/N^2$.

Therefore c/r is the only fraction with denominator $\le N$ at distance $\le 1/2q$ from b/q. Applying continued-fraction expansion to b/q efficiently gives us the fraction with denominator $\le N$ that is closest to b/q. This fraction must be c/r. Again, with good probability c and r will be coprime, in which case writing c/r in lowest terms gives r. The whole algorithm finds a factor of N in expected time $\tilde{O}((\log N)^2)$.

Let us consider this case more in detail.

Example. Imperfect periodicity. In the general case, r does not divide Q, and this means that the picture is less clear. «Bad» *k's* do not completely cancel out. We distinguish between two types of *k's*, for which the geometrical series of vectors in the complex plain looks as follows:



In the left case, all the vectors point in different directions, and they tend to cancel each other. This will cause destructive interference, which will cause the amplitude of such *k's* to be small. In the right case, all vectors point almost to the same direction. In this case there will be constructive interference of all the vectors. This happens when $e^{2\pi ikr/Q}$ is close to one, or when $kr \bmod Q$ is close to zero.

This means that with high probability, we will measure only *k's* which satisfy an *approximate* criterion $kr \approx 0 \bmod Q$. In particular, consider *k's* which satisfy: $-r/2 \le kr \bmod Q \le r/2$. There are exactly r values of k satisfying this requirement, because k runs from 0 to $Q-1$, therefore kr runs from 0 to $(Q-1)r$, and this set of integers contains exactly r multiples of Q.

Note, that for such *k's* all the complex vectors lie in the upper half of the complex plane, so they are instructively interfering. Now the probability to measure such a *k* is bounded below, by choosing the largest exponent possible:

Prob(k)	=	$\left\|\frac{1}{\sqrt{QA}}\sum_{j=0}^{A-1}(e^{2\pi ikr/Q})^j\right\|^2 \geq \left\|\frac{1}{\sqrt{QA}}\sum_{j=0}^{A-1}(e^{i\pi r/Q})^j\right\|^2$				
	=	$\frac{1}{QA}\left\|\frac{1-e^{\pi irA/Q}}{1-e^{i\pi r/Q}}\right\|^2$	=	$\frac{1}{QA}\left\|\frac{\sin\left(\frac{\pi rA}{2Q}\right)}{\sin\left(\frac{\pi r}{2Q}\right)}\right\|^2$	≈	$\frac{4}{\pi^2 r}$

4. Quantum algorithm to calculate the order

Let us any practical examples of Shor's algorithm application.

Example. Quantum circuit for finding the order of the positive integer x mod *N*. Consider the circuit of Fig. 1. It calculates the order *r* of the positive integer x less than *N*, coprime to *N*.

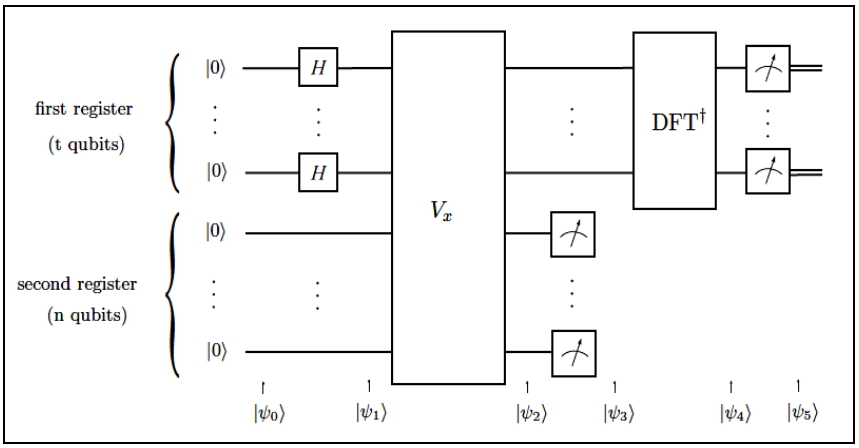


Fig. 1. Quantum circuit for finding the order of the positive integer x mod N

V_x is the unitary linear operator

$$V_x\left(|j\rangle|k\rangle\right) = |j\rangle\left|k + x^j\right\rangle, \tag{1}$$

where $|j\rangle$ and $|k\rangle$ are the states of the first and second registers, respectively. The arithmetical operations are performed mod *N*, so $0 \leq k + x^j < N$. 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. As an exception, if the order *r* is a power of 2, then it is enough to take $t = n$. In this section we consider this very special case and leave the general case for next section. We will keep the variable *t* in order to generalize the discussion later on.

The states of the quantum computer are indicated by $|\psi_0\rangle$ to $|\psi_5\rangle$ in Fig. 1. The initial state is

$$|\psi_0\rangle = |\underbrace{0\ldots0}_{t}\rangle|\underbrace{0\ldots0}_{n}\rangle.$$

The application of the Hadamard operator on each qubit of the first register yields

$$|\psi_1\rangle = \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|0\rangle . \quad (2)$$

The first register is then in a superposition of all states of the computational basis with equal amplitudes given by $\frac{1}{\sqrt{2^t}}$. Now we call the reader's attention to what happens when we apply V_x to $|\psi_1\rangle$ as:

$$|\psi_2\rangle = V_x|\psi_1\rangle = \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} V_x\left(|j\rangle|0\rangle\right) = \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle|x^j\rangle . \quad (3)$$

The state $|\psi_2\rangle$ is a remarkable one. Because V_x is linear, it acts on all $|j\rangle|0\rangle$ for 2^t values of j, so this generates all powers of x simultaneously. This feature is called quantum parallelism. Some of these powers are 1, 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 (3) for V_x.

Classically, one would calculate successively x^j, for j starting from 2 until reaching $j = r$.

Quantumly, one can calculate all powers of x with just one application of V_x.

At the quantum level, the values of j that yield $x_j \equiv 1 \bmod N$ are "known". But this quantum information is not fully available at the classical level. 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 the states in the superposition (1.10) have equal amplitudes. The first part of the strategy to find r is to observe that the first register of the states $|0\rangle|1\rangle, |r\rangle|1\rangle, |2r\rangle|1\rangle, \ldots, |2^t - r\rangle|1\rangle$ is periodic. So 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 $ar + b$ for j in Eq. (10), where $0 \le a \le (2^t/r) - 1$ and $0 \le b \le r - 1$. Recall that we are supposing that $t = n$ and r is a power of 2, therefore r divides 2^t. Eq. (3) is conveted to

$$|\psi_2\rangle = \frac{1}{\sqrt{2^t}} \sum_{b=0}^{r-1} \left(\sum_{a=0}^{\frac{2^t}{r}-1} |ar+b\rangle \right) |x^b\rangle \quad (4)$$

In the second register, we have substituted x^b for x^{ar+b}, since $x^r \equiv 1 \bmod N$. Now the second register is measured. 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

$$|\psi_3\rangle = \sqrt{\frac{r}{2^t}} \left(\sum_{a=0}^{\frac{2^t}{r}-1} |ar+b_0\rangle \right) |x^{b_0}\rangle . \quad (5)$$

Remark. Note that after the measurement, the constant is renormalized to $\sqrt{r/2^t}$, since there are $2^t / t$ terms in the sum (5). Figure 2 shows the probability of obtaining the states of the computational basis upon measuring the first register.

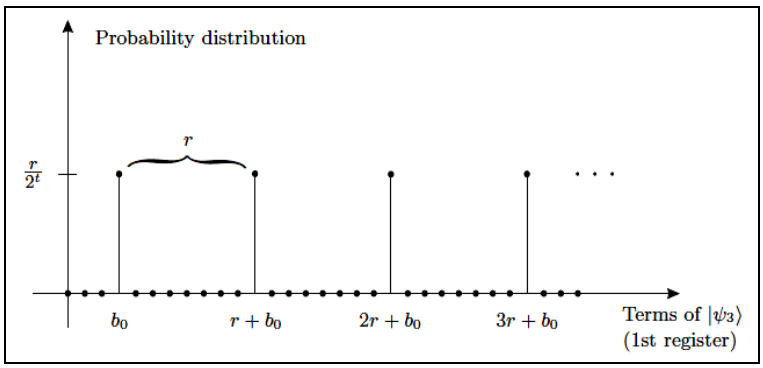


Fig. 2. Probability distribution of $|\psi_3\rangle$ *measured in the computational basis (for the case* $b_0 = 3$ *and* $r = 8$*)*

The horizontal axis has 2^t points. The number of peaks is $2^t/r$ and the period is r. The probabilities form a periodic function with period r. Their values are zero except for the states $|b_0\rangle, |r+b_0\rangle, |2r+b_0\rangle, \ldots, |2^t - r + b_0\rangle$.

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

The FT is the second and last part of the strategy. The whole method relies on an efficient QA for calculating the FT, which is not available classically. In next section, we show that the FT is calculated efficiently in a quantum computer.

Example. The quantum discrete FT (DFT). The FT 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

$$\tilde{F}(k) = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N} F(j) \tag{6}$$

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

$$DFT(|k\rangle) = |\psi_k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N} |j\rangle \tag{7}$$

where the set $\{|\psi_k\rangle : k = 0,\ldots,N-1\}$ forms a new orthonormal basis, i.e., $\langle\psi_{k'}|\psi_k\rangle = \delta_{k'k}$. The FT is a unitary linear operator. So, if we know how it acts on the states of the computational basis, we also know how it acts on a generic state: $|\psi\rangle = \sum_{a=0}^{N-1} F(a)|a\rangle$.

The FT of $|\psi\rangle$ can be performed indistinctly using either (6) or (7).

Now we will continue the calculation process of the circuit of Fig. 1. We are ready to find out the next state of the quantum computer — $|\psi_4\rangle$. Applying the inverse FT on the first register, using Eq. (7) and the linearity of DFT†, we obtain

$$|\psi_4\rangle = DFT^{\dagger}(|\psi_3\rangle) = \sqrt{\frac{r}{2^t}} \sum_{a=0}^{\frac{2^t}{r}-1} \left(\frac{1}{\sqrt{2^t}} \sum_{j=0}^{\frac{2^t}{r}-1} \left(e^{-2\pi i j (ar+b_0)/2^t} \right) |j\rangle \right) |x^{b_0}\rangle .$$

Inverting the summation order, we have

$$\left|\psi_4\right\rangle = \frac{1}{\sqrt{r}}\left(\sum_{j=0}^{2^t-1}\left[\frac{r}{2^t}\sum_{a=0}^{\frac{2^t}{r}-1}\left(e^{-\frac{2\pi ija}{2^t}r}\right)\right]e^{-\frac{2\pi ijb_0}{2^t}}\left|j\right\rangle\right)\left|x^{b_0}\right\rangle \quad (8)$$

Using (7), we see that the expression in the square brackets is not zero if and only if $j = k2^t/r$, with $k = 0,..., r - 1$. When j takes such values, the expression in the square brackets is equal to 1. So, we have

$$\left|\psi_4\right\rangle = \frac{1}{\sqrt{r}}\left(\sum_{k=0}^{r-1}e^{-2\pi i\frac{k}{r}b_0}\left|\frac{k2^t}{r}\right\rangle\right)\left|x^{b_0}\right\rangle \quad (9)$$

In order to find r, the expression for $\left|\psi_4\right\rangle$ has two advantages over the expression for $\left|\psi_3\right\rangle$ (Eq. (6)): r is in the denominator of the ket label and the random parameter b_0 moved from the ket label to the exponent occupying now a harmless place.

Figure 2 shows the probability distribution of $\left|\psi_4\right\rangle$ 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 (the peaks in Fig. 2). 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 in the first round, the algorithm halts, otherwise we apply it recursively. The recursive process does not last, 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, coprime to 15 is {1, 2, 4, 7, 8, 11, 13, 14}. The numbers in the set {4, 11, 14} have order 2 and the numbers in the set {2, 7, 8, 13} have order 4. Therefore, in any case r is a power of 2 and the factors of $N = 15$ can be found in a 8-bit quantum computer $\left(t + n = 2\lceil \log_2 15\rceil = 8\right)$. A 7-qubit quantum computer is used, bypassing part of the algorithm.

We have considered a special case when the order r is a power of 2 and $t = n$ (t is the number of qubits in the first register — Fig. 1 — and $n = \lceil \log_2 N \rceil$).

Now we consider the factorization of $N = 21$, that is the next nontrivial composite number.

Example. Generalization by means. 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 9. This is the simplest example allowed by the constraints, but enough to display all the 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 *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 $\left|\psi_0\right\rangle = \left|0\right\rangle\left|0\right\rangle$, 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

$$\left|\psi_1\right\rangle = \frac{1}{\sqrt{512}}\sum_{j=0}^{511}\left|j\right\rangle\left|0\right\rangle.$$

The next step is the application of V_x, which yields

$$|\psi_2\rangle = \frac{1}{\sqrt{512}}\sum_{j=0}^{511}|j\rangle|2^j \bmod N\rangle =$$
$$= \frac{1}{\sqrt{512}}\begin{pmatrix}|0\rangle|1\rangle+|1\rangle|2\rangle+|2\rangle|4\rangle+|3\rangle|8\rangle+|4\rangle|16\rangle+|5\rangle|11\rangle+\\ |6\rangle|1\rangle+|7\rangle|2\rangle+|8\rangle|4\rangle+|9\rangle|8\rangle+|10\rangle|16\rangle+|11\rangle|11\rangle+\\ |12\rangle|1\rangle+\ldots\end{pmatrix}$$

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:

$$|\psi_2\rangle =$$
$$= \frac{1}{\sqrt{512}}\begin{pmatrix}\left(|0\rangle+|6\rangle+|12\rangle+\ldots+|504\rangle+|510\rangle\right)|1\rangle+\left(|1\rangle+|7\rangle+|13\rangle+\ldots+|505\rangle+|511\rangle\right)|2\rangle+\\ \left(|2\rangle+|8\rangle+|14\rangle+\ldots+|506\rangle\right)|4\rangle+\left(|3\rangle+|9\rangle+|15\rangle+\ldots+|507\rangle\right)|8\rangle+\\ \left(|4\rangle+|10\rangle+|16\rangle+\ldots+|508\rangle\right)|16\rangle+\left(|5\rangle+|11\rangle+|17\rangle+\ldots+|509\rangle\right)|11\rangle\end{pmatrix}. \quad (10)$$

This feature was made explicit in Eq. (4). Because the order is not a power of 2, here there is a small difference: the first two lines of Eq. (10) have 86 terms, while the remaining ones have 85.

Now one measures the second register1, yielding one of the following numbers equiprobably: {1, 2, 4, 8, 16, 11}. Suppose that the result of the measurement is 2, then

$$|\psi_3\rangle = \frac{1}{\sqrt{86}}\left(|1\rangle+|7\rangle+|13\rangle+\ldots+|505\rangle+|511\rangle\right)|2\rangle \quad (11)$$

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 (11).

The period of the states of the first register is the solution to the problem and the FT can reveal the value of the period. So, the next step is the application of the inverse Fourier transform on the first register of $|\psi_3\rangle$:

$$\begin{aligned}|\psi_4\rangle &= DFT^{\dagger}\left(|\psi_3\rangle\right) = DFT^{\dagger}\left(\frac{1}{\sqrt{86}}\sum_{a=0}^{85}|6a+1\rangle\right)|2\rangle \\ &= \frac{1}{\sqrt{512}}\sum_{j=0}^{511}\left(\left[\frac{1}{\sqrt{86}}\sum_{a=0}^{85}e^{-2\pi i\frac{6ja}{512}}\right]e^{-2\pi i\frac{j}{512}}|j\rangle\right)|2\rangle\end{aligned} \quad (12)$$

where we have used Eq. (8) and have rearranged the sums. The last equation is similar to Eq. (10), but with an important difference. We were assuming that r divides 2^t. This is not true in the present example (6 does not divide 512), therefore we cannot use the identity to simplify the term in brackets in Eq. (9). This term never vanishes, but its main contribution is still around j = 0, 85, 171, 256, 341, 427, which are obtained rounding 512 $k_0/6$ for k_0 from 0 to 5.

To see this, let us plot the probability of getting the result j (in the interval 0 to 511) by measuring the first register of the state $|\psi_4\rangle$. From (12), we have that the probability is

$$\mathrm{Prob}(j) = \frac{1}{512\times 86}\left|\sum e^{-2\pi i\frac{6ja}{512}}\right|^2 \quad (13)$$

The plot of Prob (j) is shown in Fig. 3.

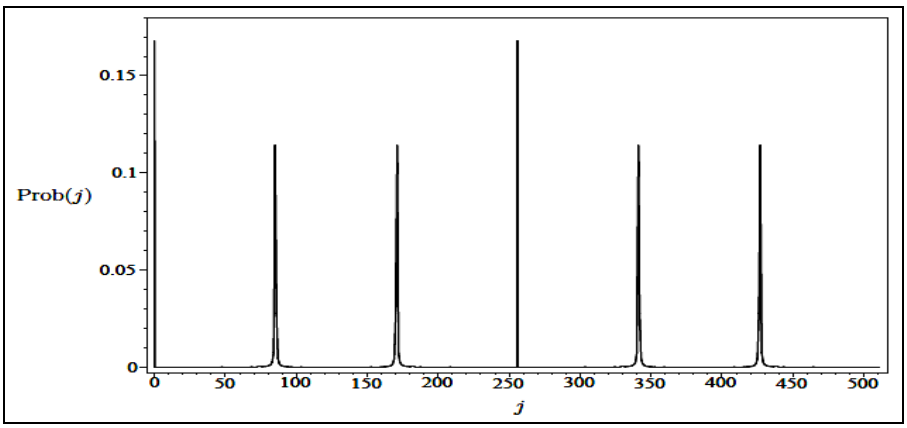


Fig. 3. Plot of Prob(j) against j

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 (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.

Let us analyze the possible measurement results. If we get j = 0 (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. The probability of getting j = 0 is low: from Eq. (13) we have that Prob(0) = 86/512 ≈ 0.167. Now suppose we get j = 85 (or any value in the second peak). We divide by 512 yielding 85/512, which is a rational approximation of $k_0/6$, for k_0 = 1.

Compare to the plot of Fig. 2, where the peaks are not spread and have the same height.

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

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

usually represented as $[a_0, a_1, ..., a_p]$, where a_0 is a non-negative integer and a_1, ..., a_p are positive integers. The q-th convergent ($0 \leq q \leq p$) is defined as the rational number $[a_0, a_1, ..., a_q]$. 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 numerator 1. The result is

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

So, the convergent of 85/512 are 1/6, 42/253, and 85/512. We must select the convergent that have a denominator smaller than N = 21 (since $r < N$). This method yields 1/6, and then r = 6. We check that $2^6 \equiv 1$ mod 21, and the quantum part of the algorithm ends with the correct answer. The order r = 6 is an even number, therefore $gcd(2^{(6/2)} \pm 1, 21)$ 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 $k_0/6$, 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 a factor of r (r_1 = 3), since $2^3 \equiv 8$ 6≡ 1 mod 21. We run the quantum part of the algorithm again to find the order of 8. We eventually obtain r_2 = 2, which yields $r = r_1 r_2 = 3 \times 2 = 6$. 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 all the peaks is approximately the same: ≈ 0.167. The first and fourth peaks have a nature different from the others — they are not spread. To calculate their contribution to the total probability, we take the basis equal to 1. The area under the second, third, fifth, and last peaks are calculated by adding up Prob(*j*), for *j* running around the center of each peak.

So, in approximately 17% cases, the algorithm fails (1st peak). In approximately 33% cases, the algorithm returns *r* in the first round (2nd and 6th peaks). In approximately 50% cases, the algorithm returns *r* in the second round or more (3rd, 4th, and 5th peaks).

Now we calculate the probability of finding *r* in the second round. For the 3rd and 5th peaks, the remaining factor is $r_2 = 2$. The graph equivalent to Fig. 3 in this case has 2 peaks, then the algorithm returns r_2 in 50% cases. For the 4th peak, the remaining factor is $r = 3$ and the algorithm returns r_2 in 66.6% cases. This amounts to $\frac{2\times 50\% + 66.6\%}{3}$ of 50%, which is equal to around 22%. In summary, the success probability for $x = 2$ is around 55%.

Remark. We have shown that Shor's algorithm is an efficient probabilistic algorithm, assuming that the FT could be implemented efficiently. The complete circuit for the QFT is given in Fig. 4.

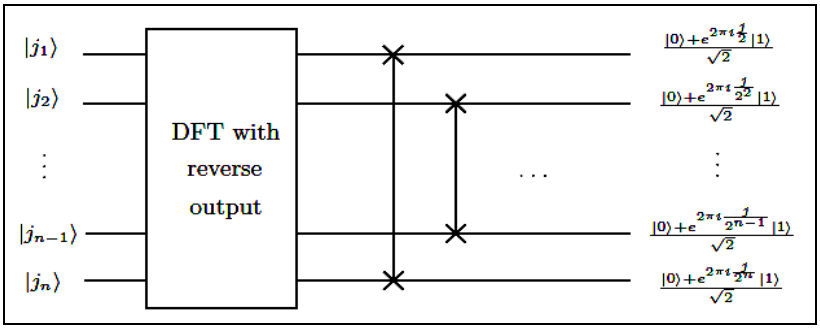


Fig. 4. The complete circuit for the quantum Fourier Transform

Now we can calculate the complexity of the quantum Fourier circuit. Counting the number of elementary gates, 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 Fourier Transform. V_x is the largest gate of Fig. 1. Actually, Shor stated in his 1997 paper that V_x is the «bottleneck of the quantum factoring algorithm» due to the time and space consumed to perform the modular exponentiation. The bottleneck is not so strict though 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. V_x is a problem in recursive calls of the algorithm when *x* changes. For each *x*, a new circuit must be built, what is troublesome at the present stage of hardware development.

Example: *Quantum Shor's Algorithm* (*Quantum factorization promise*). Figure 5 shows the factorization problem.

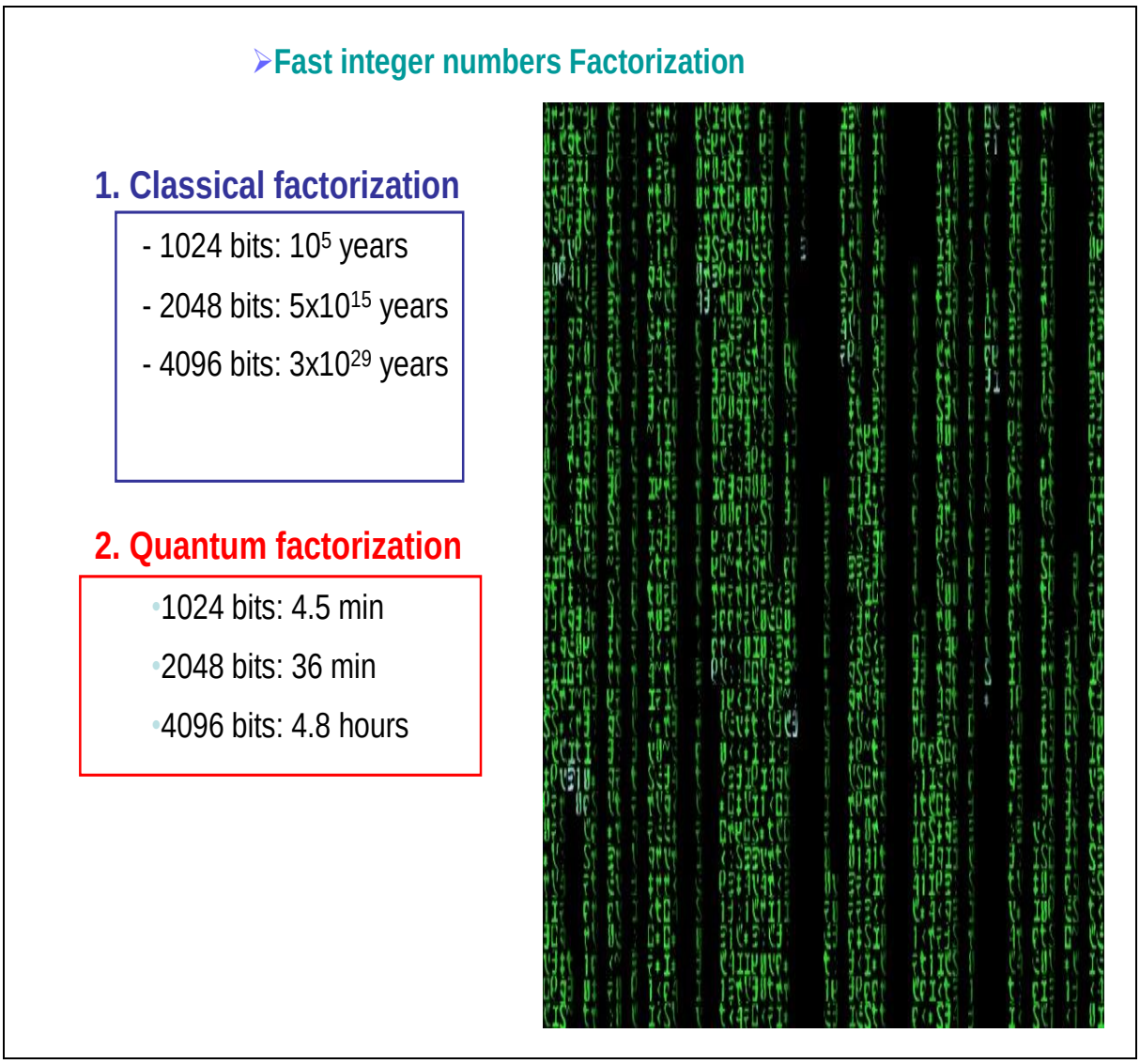


Fig. 5. Fast factorization problem and its solutions

Figure 6 shows the quantum Shor algorithm and its describing circuit (see Fig. 1).

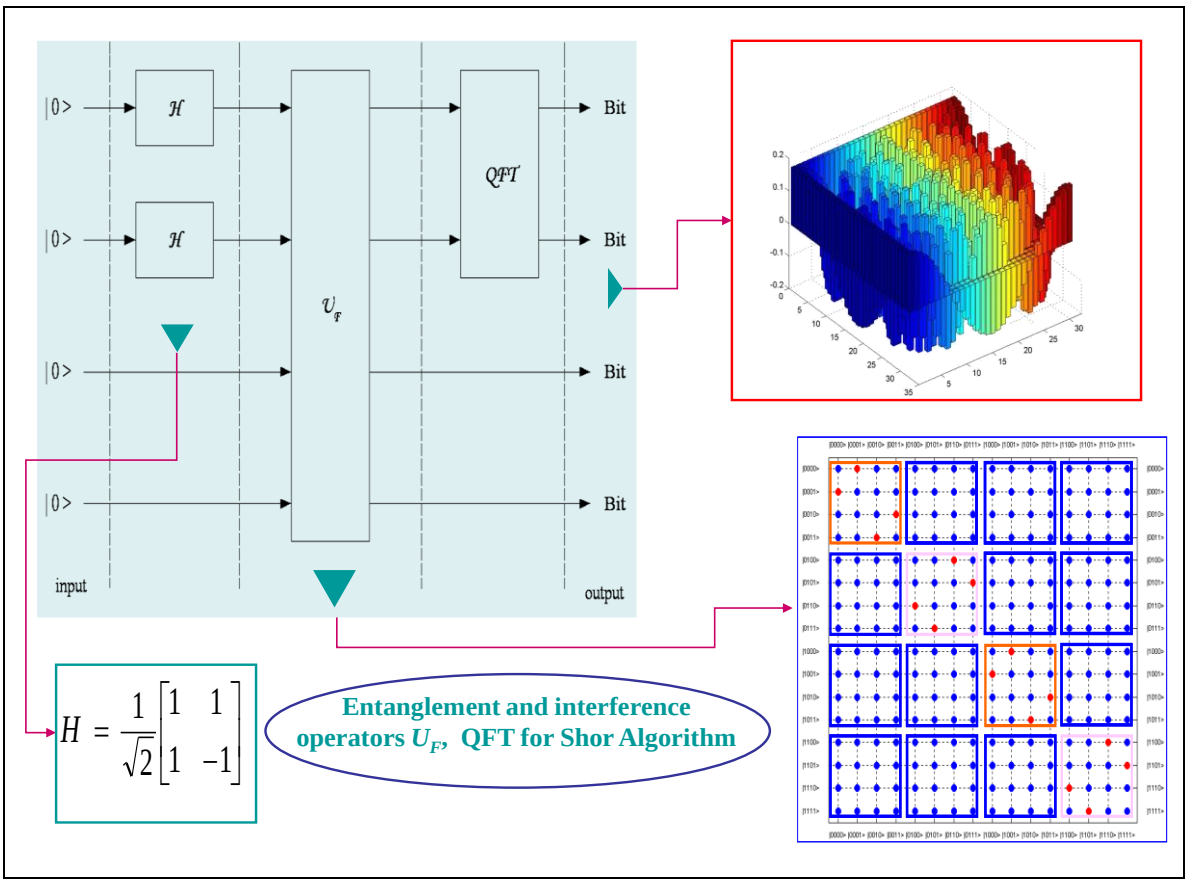


Fig. 6. Quantum Shor's algorithm circuit and main quantum operators

Factorization time using matrix and vector approach are here reported (see, Fig. 7 and Fig. 8).

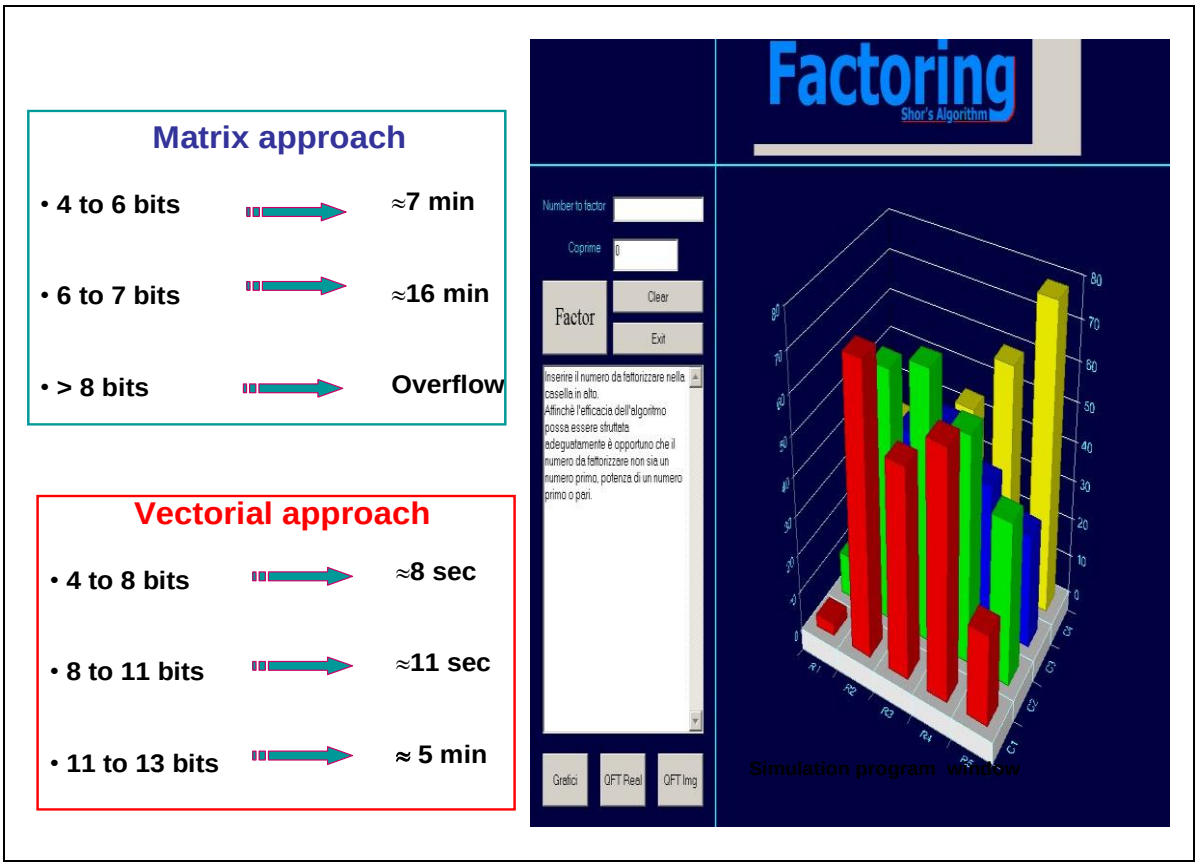


Fig. 7. SW simulation of Shor's quantum factorization algorithm

Quantum Algorithm	**Superposition Operator**	*Common Part*
Deutsch - Jozsa	$^{n+1}H = {}^{n}H \otimes {}^{1}H$	^{n}H
Grover	$^{n+1}H = {}^{n}H \otimes {}^{1}H$	^{n}H
Shor	$^{n}H \otimes {}^{n}I = {}^{n}H \otimes {}^{n}I$	^{n}H

$$\Downarrow$$

Quantum Algorithm	**Interference Operator**	*Common Part*
Deutsch - Jozsa	$^{n}H \otimes I = {}^{\otimes n}H \otimes I$	$H \otimes I$
Grover	$D_n \otimes I = -H\left[PhaseInv_0\right]H \otimes I = -H\left[\underbrace{\begin{Bmatrix} -1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{Bmatrix}}_{n \times n}\right] \boxed{H \otimes I}$	$H \otimes I$
Shor	$QFT_n \otimes {}^{n}I = n \cdot H \otimes {}^{n}I = nH \otimes \left[\underbrace{I \otimes \cdots \otimes I}_{n}\right] = \underbrace{H \otimes I \otimes \cdots \boxed{H \otimes I}}_{n}$	$H \otimes I$

Fig. 8. Relations between the quantum operators in quantum algorithms

We can observe U_F block that is a diagonal matrix of $2^{2n} \times 2^{2n}$ dimension. Finally output of entanglement is processed by interference block composed of Quantum Fourier Transform (*QFT*) and identity matrix *I*. The output of entire algorithm is therefore the vector obtained after application of operator $QFT_n \otimes {}^{n}I$.

5. The Quantum Fourier Transform and Quantum Fourier Sampling

One of the most basic building blocks for quantum algorithms is the quantum Fourier transform (QFT) algorithm. The Fourier transform, a critical step in many classical calculations and computations, is an operation that transforms one representation of a signal of interest into a different representational form. The classical Fourier transform turns a signal represented as a function of time into its corresponding signal represented as a function of frequency. For example, this could mean transforming a mathematical description of a musical chord in terms of air pressure as a function of time into the amplitudes of the set of musical tones (or notes) that combine to form the chord. This transformation is reversible via the inverse Fourier transform, so involves no information loss - a key requirement for any operation on a quantum computer. Concretely, the input is an *N*-dimensional vector with complex entries (a_1, a_2, …, a_N), and the output is an *N*-dimensional vector with complex entries (b_1, b_2, …, b_N) which is obtained by multiplying the input vector with the $N \times N$ Fourier transform matrix. Given the utility of the Fourier transform, many clever algorithms have been developed to implement it on classical computers. The best, the fast Fourier transform (FFT), takes $O(N\log N)$ time, which is only slightly longer than it takes to read the input data [$O(N)$]. While the classical FFT is quite efficient, quantum Fourier transform (QFT) is exponentially faster, requiring only $O(\log^2 N) = O(n^2)$ time (where $N = 2^n$) in its original formulation, later improved to $O(n\log n)$.

Before describing the QFT, it is important to understand how the input and output are represented as quantum states. The input $(a_1, a_2, \ldots, a_N)$ is represented as the quantum state $\sum_i a_i |i\rangle$, and the output $(b_1, b_2, \ldots, b_N)$ is represented as the quantum state $\sum_i b_i |i\rangle$. Thus, the input and output are represented as states of just *n* qubits, where $n = \log N$. This is shown in Fig. 9.

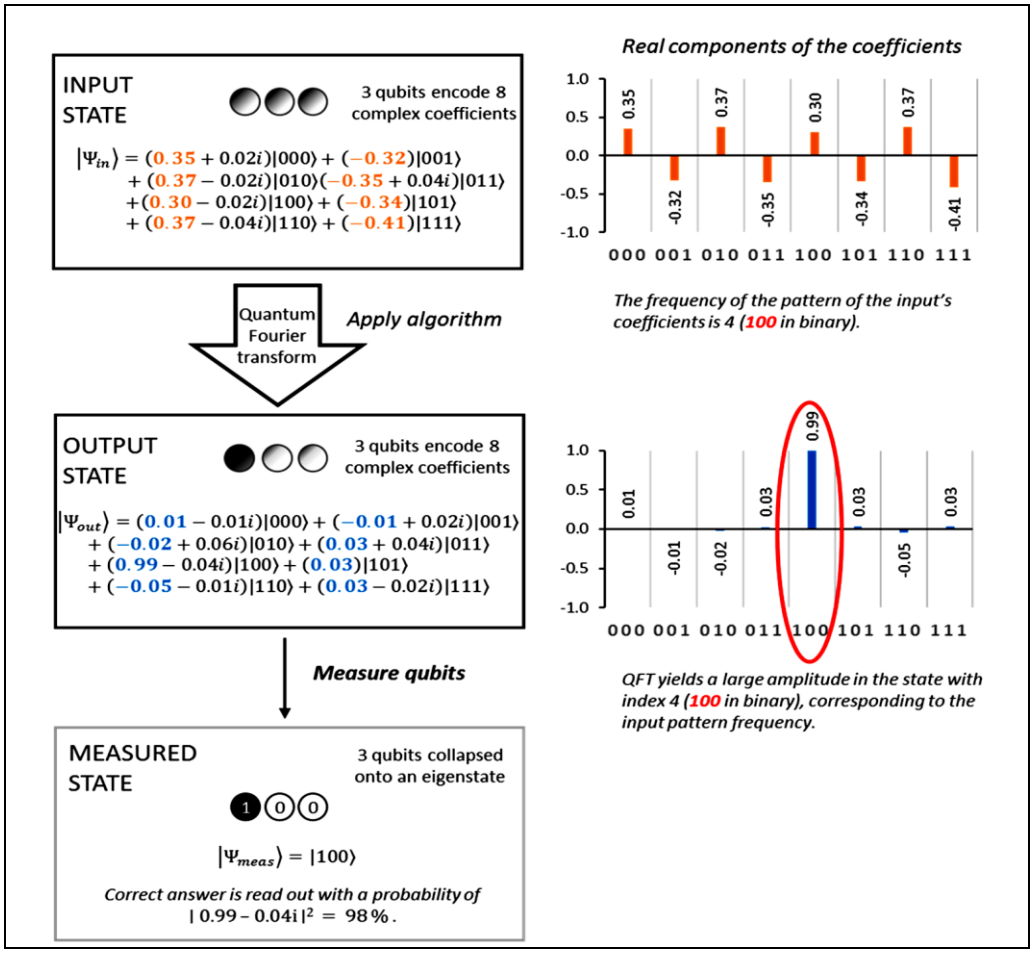


Fig. 9. An illustrative example of the quantum Fourier transform (QFT) applied to a three-qubit system

The three qubits must be initially prepared such that the eight ($2^3 = 8$) complex coefficients encode the system state corresponding to the sequence of values to be transformed. Since the number of coefficients, *N*, is 2^n, where *n* is the number of qubits, only log(*N*) bits are needed: 3 qubits can represent the 8 complex values shown. The QFT effectively finds patterns in the input sequence and identifies their frequency of repetition. In this example, all the input states have similar probability, with the real components of coefficients alternating sign four times. The coefficients of the output state (shown on the right) capture this pattern: the coefficient of the i state, a_i, is large if there are i cycles in the input sequence. Thus, in this example all the outputs are all small except for one state, 100, corresponding to the input pattern frequency. Thus, measuring this output is likely to provide the index of this strong pattern in the input sequence.

Exponential speedup is possible only if the input data has already been encoded into a compact quantum state, or can be encoded into this state in *O*(log *N*) steps. The quantum circuit that carries out this transformation has total number of gates that scales as *O*(*n* log *n*). Another caveat is that one of course cannot access the amplitudes b_i through measurement. Indeed, if the output of the QFT is measured, it yields the index *i* with probability $|b_i|^2$. Thus, measuring this algorithm's output only yields the index of a probable output, which is called quantum Fourier sampling (QFS). QFS is an important primitive in quantum algorithms, and entails applying the QFT and measuring the output state, resulting in the sampling of an index *i* from a certain probability distribution.

It turns out that sampling the output of the Fourier transform is useful in some cases for finding structure in a sequence of numbers, as illustrated in Fig. 9. Notice that the coefficients of the input data are periodic, with four periods in this sequence. This periodicity causes the amplitude of state $|100\rangle$ to be much larger than all the others, so with high probability, measuring the final system state will return 100 (binary for 4), revealing the input sequence repeated 4 times, or had a repeat distance of 2. This example illustrates the power and pitfalls of quantum computing. If the initial input superposition already exists, the Fourier transform can be performed on the superposition coefficients exponentially faster than would be possible classically. However, at the end of this operation, one samples only one of the *N* states, rather than obtaining the entire set of output coefficients.

Furthermore, it is not clear in general how to create the input superposition without taking $O(N)$ time - although this becomes less of a problem if QFT is performed on a preloaded input quantum state as one step in a longer algorithm. The QFT, which cleverly leverages the characteristics of quantum computation, is useful in constructing a host of quantum algorithms.

There exist many different definitions of the Fourier transform, though, as every branch of science in which the Fourier transform is used tends to use its own definition. In that respect, quantum computing is no different, as the definition of the Fourier transform used in quantum computing is subtly different from the definitions found in other areas. The version of the Fourier transform used in quantum computing is referred to as the quantum Fourier transform. We will spend the next part of this section on introducing this transform and after that, we will elaborate on how this transform can be implemented in a quantum circuit. It completes the description of the implementation of the quantum Fourier transform as a quantum circuit. The entire process is summarized and depicted in Fig. 10.

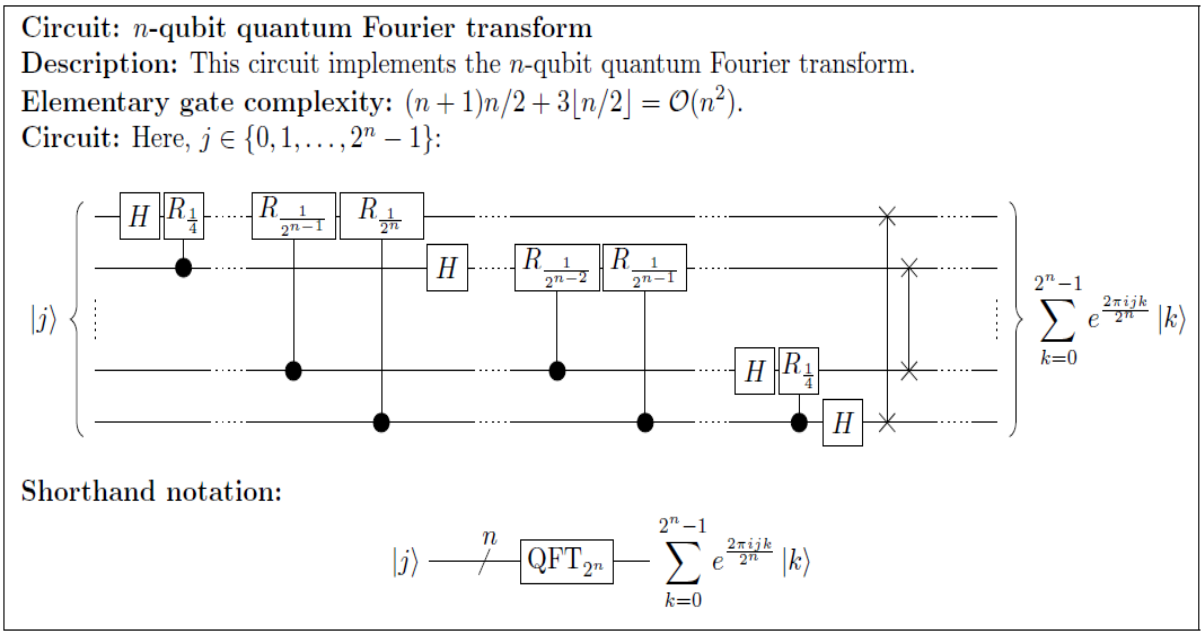
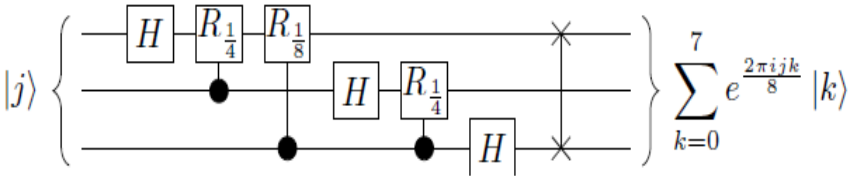
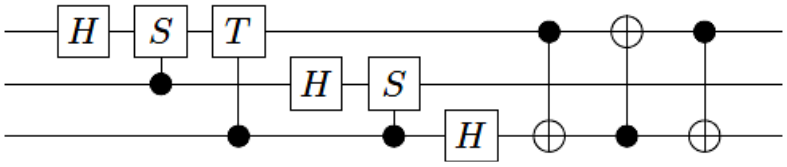


Fig. 10. N-qubit Fourier transform circuit

For extra clarification, let us write out the full quantum circuit implementing the three-qubit quantum Fourier transform QFT_2^3:



Recall that the $R_{1/4}$ and $R_{1/8}$ gates are equal to the *S* and *T* gates. Hence, we can rewrite the circuit as follows:



The elementary gate complexity is equal to 9, which is equal to the expected number when n = 3: $(n+1)n/2+3\lfloor n/2 \rfloor = 4\cdot 3/2+3\cdot 1=6+3=9$.

Here shown how to implement the mapping QFT_2^n exactly, using a number of elementary gates that grows quadratically in n. Note that the number of CNOT-gates and the number of Hadamard gates grows only linearly in n, though, and that only the number of controlled-R-gates grows quadratically in n. More precisely, observe that we use exactly $n - k + 1$ controlled application of $R_{\frac{1}{2^n}}$. In addition, observe that the controlled $R_{\frac{1}{2^n}}$-gates where k is big, are very close to the identity gate in operator norm, indicating that their action is almost negligible. Hence, we might just as well leave out these controlled-R-gates where k is big. This idea allows us to obtain a significant reduction in the elementary gate complexity. We can implement the quantum Fourier transform up to some error ε in operator norm, using only $O(n \log(n))$ elementary gates.

The quantum Fourier transform (QFT) can be derived by further decomposing the diagonal factors of the fast Fourier transform (FFT) matrix decomposition into products of matrices with Kronecker product structure. Such a structure can take advantage of an important quantum computer feature that enables the QFT algorithm to attain an exponential speedup on a quantum computer over the FFT algorithm on a classical computer. The connection between the matrix decomposition of the DFT matrix and a quantum circuit is made.

The following complete QFT quantum circuit is shown on Fig. 11.

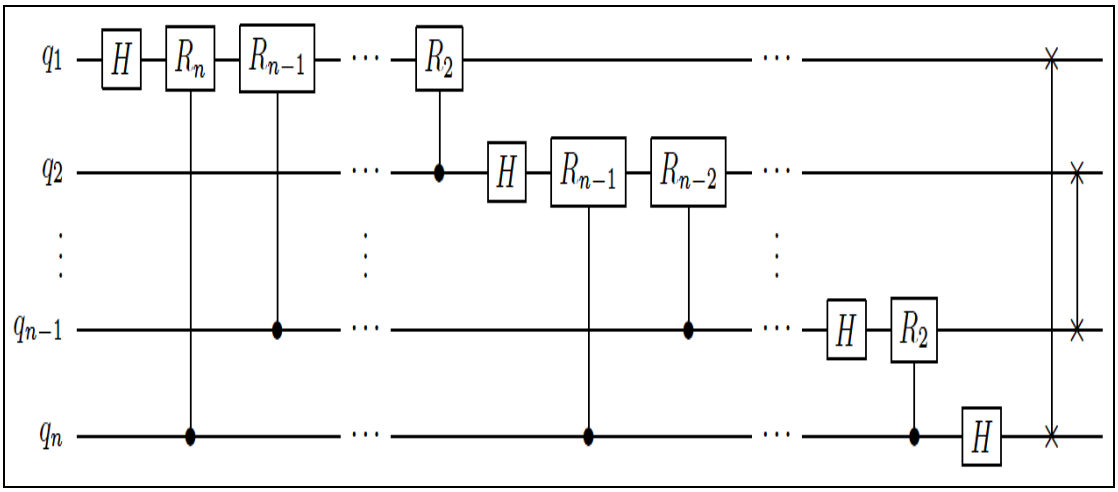


Fig. 11. Complete QFT quantum circuit. [adapted from D. Camps, Quantum Fourier transform revisited // arxiv.org [math. NA]. 2003.03011v1. 6 March. 2020.]

In this QFT circuit, the control and target qubits swapped for all the controlled-R gates. This particular choice gives the same QFT circuit [14].

Thus, the quantum Fourier transform algorithm can be derived as a decomposition of the discrete Fourier matrix. The derivation starts from the radix-2 decomposition of the DFT matrix that yields the FFT algorithm, and only makes use of Kronecker products to further decompose a diagonal matrix into a product of simpler unitary matrices, each of which can be written as the sum of two Kronecker products of 2×2 matrices. This alternative approach to the derivation of the quantum Fourier requires little knowledge of quantum computing.

Conclusions

- First of all, it was shown that any quantum circuit can be implemented using a finite set of components, up to arbitrary accuracy. Secondly, it was shown that the algorithm of Shor provides an efficient way of factorizing integers using a quantum computer. Together, they provide an introduction into the realm of quantum computing.
- If one were to do a follow-up study, then it would be interesting to investigate the lower bound on the probability that Shor's algorithm successfully factorizes the number given. This lower bound can probably be improved by investigating which step introduces the most inaccuracy into the bound.
- Furthermore, the classical simulation of Shor's algorithm could be improved significantly by having a look at the implementation of the inverse Fourier transformation. Simple measurements reveal that over 80% of the runtime of the code is spent calculating the inverse Fourier transform, which could be significantly sped up by implementing other algorithms to perform this operation.
- Ultimately, though, the field of quantum computing will only become a very exciting field of study when the quantum computer is at one's disposal. Until this is experimentally feasible, devising other quantum algorithms that definitively beat the implementations on classical computers is one of the most important tasks for researchers. More generally, investigating the full potential of the concept of the quantum computer is of vital importance to the field as a whole.

Reference

1. Gruska J. Quantum computing. Advanced Topics in Computer Science Series, McGraw-Hill Companies, London, 1999.
2. Nielsen M. A., Chuang I. L. Quantum computation and quantum information. Cambridge University Press, Cambridge, England, 2000.
3. Hirvensalo M. Quantum computing. Natural Computing Series, Springer-Verlag, Berlin, 2001.
4. Hardy Y., Steeb W.-H. Classical and quantum computing with C++ and Java Simulations. Birkhauser Verlag, Basel, 2001.
5. Hirota O. The foundation of quantum information science: Approach to quantum computer (in Japanese), Japan, 2002.
6. Pittenberg A. O. An introduction to quantum computing and algorithms.-Progress in Computer Sciences and Applied Logic. 1999. Vol. 19.
7. Brylinski F. K., Chen G. (Eds) Mathematics of quantum computation. Computational Mathematics Series, CRC Press Co., 2002.
8. Lo H.-K., Popescu S., Spiller T. (Eds) Introduction to quantum computing and information. World Scientific Publ. Co., 1998.
9. Berman G. P., Doolen G.D., Mainieri R., Tsifrinovich V.I. Introduction to quantum computers. World Scientific Publ. Co., 1999.
10. Rieffel E., Polak W. An introduction to quantum computing for non-physicists // ACM Computing Surveys. 2000. Vol. 32. N. 3. P. 300–335.
11. Hogg T., Mochon C., Polak W., Rieffel E. Tools for quantum algorithms // Intern. J. of Modern Physics. 1999. Vol. C10. N. 7. P. 1347–1361.
12. Uesaka Y. Mathematical principle of quantum computation (in Japanese). Corona Publ. Co. Ltd, 2000.
13. Marinescu D. C., Marinescu G.M. Approaching quantum computing. Pearson Prentice Hall, New Jersey, 2005.
14. Cornelissen A. J. Quantum Computation: Shor's algorithm. Bachelor Thesis // Electrical Engineering, Mathematics and Computer Science Applied Sciences, 2016.