Analysis of Shor's Algorithm Cryptography and Security of Digital Devices, A.Y. 2019–2020, Semester: 2

Giacomo Lancellotti - Matteo Lodi

June 2020

Abstract

We present an overview of Shor's factoring algorithm describing the overall functioning, pointing out the Quantum subroutine of the algorithm and the implication on modern cryptography. We finally show an implementation using the IBM Qiskit framework.

Contents

1 Introduction

1.1 Classical Computation

What we have always thought as “classic” computer theory, meaning the architecture behind modern calculators, is based on an abstract model of a universal machine called Turing Machine. Alan Turing in the ‘30s, as well as John Von Neumann in the ‘40s, studied and stated the logic principles that constitute today’s (ideally speaking at least) architecture: input/output tapes, registers, instructions, as well as the actual definition of “state” of computation, thanks to the presence of a head that can move along those tapes. It’s been proven that given any modern computer algorithm, this machine is capable of simulating its logic behavior.

Modern classic computation is largely based on this machine and, without going down the rabbit hole of uncomputability and/or finite memory limitations, it deals specifically with what are called deterministic algorithms: the same input will always yield the same output. This property, which as we will see is quite restrictive from a computational point of view, is the reason why this paper’s topic is one example of the superiority of probabilistic algorithms: a chance factor influences the outcome, which may result in a different value even with the same given input. On the other hand, this is a trade-off since we give up the opportunity of a certain result (by introducing a probabilistic error) for an efficiency improvement.

1.2 Basics of Quantum Computing

The idea of taking advantage of a different kind of paradigm for calculation purposes arose in the ‘80s with the arrival of Quantum mechanics: a few very important physics phenomena can be exploited in order to have a exponential speed-up in certain tasks. In this paper we will analyze particularly how this advantage can be exploited against the so-called Factorization Problem and how Shor’s Algorithm can efficiently solve it, meaning in polynomial time.

In order to understand how and especially why this works, we need to explain the phenomena which we will base our results on, starting with the concept of superposition. The fundamental information element in classic computing is called ‘bit’ and its state, if put in sequence with other bits’ states, represents the pieces of information that we want to express. In a bit the state can either be 1 or 0. Quantum bits (or Qubits) work similarly: a Quantum System is a group of Qubits whose states represents information. What a Qubit and a classic bit differs in is the so called **Superposition** the latter can be in. A Qubit’s state isn’t simply a 0 or a 1 (or however we want to call them) but in a position in between, meaning any linear combination of the orthonormal basis is a valid state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, which is why the amplitudes can’t assume an arbitrary value but follow a normalization rule $\sqrt{|\alpha|^2 + |\beta|^2} = 1$ (that is the reason why they are also called Unitary Vectors). Any state whose amplitudes do not respect this rule is not a representation of a Qubit's (or system of Qubits') state. This means that the state is described by a state vector whose elements may assume not only Boolean values like a classic computation system but also what we call Superpositions. Mathematically speaking every Quantum System is associated with a Hilbert complex space, whereas from a physics perspective this superposition means that we may not know at all time instants whether the Qubit is in a state 0 or 1. The state vector for a simple Qubit may be represented by the sum of the products of the states that belong to an orthonormal basis (we chose 0 and 1 as an example) with the corresponding amplitudes.

$$|q_0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

The amplitudes can be considered, although not precisely, as the probabilities of the corresponding state to be "chosen" by the Qubit as the state to "fall into". How and why this happens is completely out of this paper's topic and very complicated, what we need to know is that the superposition and, especially, the Entanglement phenomena are essentially the reason why we can have such an efficient computing paradigm: from a "pure" state representation (0 and 1) we can encode pieces of information in between those pure values. This is the reason why Quantum Computation scales exponentially w.r.t. the Qubits number.

The Bra-Ket annotation is nothing more than another representation of a product between a column and row vector, as such it is just used as a shortening method.

$$|a\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \quad \langle a| = \begin{bmatrix} a_0 a_1 \end{bmatrix}$$

We can build Multi-Qubit systems as we have already anticipated. The system's state is represented by the joint state of the Qubits that form the system. (Technically speaking it is a tensor/Kronecker product. The vector state's elements are the combination of all the amplitudes that constitute the single Qubit's states, thus its dimension is equal to the product of those Qubits' state vectors dimensions).

$$|ab\rangle = |a\rangle \otimes |b\rangle = \begin{bmatrix} a_0 \times \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \\ a_1 \times \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{bmatrix}$$

Using this piece of information we can introduce what can be considered another key phenomenon that characterizes Quantum Computing: the phenomenon of the **Entanglement** and the corresponding entangled state. We said that if we want to represent the state of a system of Qubits we can simply calculate the joint state of those Qubits, this means that the resulting state is achievable via tensor product of single Qubits. Not all viable states are achievable via tensor product though. By using one or more ‘Gates’ (more on that later) we can achieve what we call entanglement of two (or more) Qubits. An entangled pair cannot be factorized as the product of two independent Qubits’ states. This is extremely useful because, as we will see better later, it means that through the measurement (meaning forcing the Qubit to “choose” a state) of one Qubit of the entangled pair we instantly know the state of the other Qubit, whatever is the distance between the two. (This phenomenon is subject to statistical analysis: the process is non-deterministic thus every piece of information that we want to subject to the system has to be statistically interpreted via multiple measurements). An entangled pair is also called EPR pair, and the most important pair is called Bell’s state.

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Measurement is an important element in Quantum Mechanics. When we state that a Qubit’s state can be between two basis’ states we underline the fact that we simply don’t know whether at any given moment the Qubit’s value is either one or the other. Technically speaking it can be in both with the corresponding amplitudes (meaning with the respective probabilities). The higher the amplitude, the higher the probability that the Qubit collapses in that state, but as long as we do not observe the Qubit it resides in both states. The measurement is what an external agent can perform in order to ‘force’ the Qubit into one of those states. The superposition holds as long as no one is observing: once we measure it the Qubit ‘answers’ with one of those basis states, with probability the square of the amplitude of that state.

Putting together the last couple of topics is what can be considered the spooky part of Quantum Mechanics. As we have already hinted, measuring an EPR pair forces a collapse of the measured Qubit to one of the base states. On the other hand we can already guess what happens in the system:

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad \xrightarrow{\text{measure}} \quad |11\rangle$$

Once the state of that Qubit has been revealed, the other Qubit’s state can be one and only one, based on how the pair was built. In the Bell’s state example, if we measure a 1 in one of the Qubit, the other one will be in the 1 state as well (but it is true for whatever base we decide to use). This is the reason why at the beginning of the studies of Quantum Mechanics physicists were skeptical about the whole theory truthfulness: this seems an example of instant information passage. The Qubits forming the pair can be physically very distant from each other, yet we know how one of those behaves with regard to the other. Einstein himself couldn’t believe this to be true as it breaks what was considered a fundamental notion of physics: information cannot travel faster than light itself. But as it turns out we are not conveying anything here. Once we measure one Qubit, we know where the other Qubit’s state would collapse to, but the other receiver does not. Technically speaking, nothing travels between the Qubits: we simply acknowledge something from a measurement. This is the power of Quantum Mechanics.

Another couple of important properties are worth mentioning now. The first one has a sort of negative impact on a possible computation and is called **No-Cloning Theorem**. It states that it is not possible to learn a superposition from a single copy, meaning we cannot have another Qubit be an exact replica of a Qubit in a superposition. It is easy to produce as many Qubits in a 0 state as we want, as an example, but from a single superposition we cannot create another which behaves in the same way. It is possible though, given enough copies. We do have a solution that we achieve using the second important theorem we mentioned, called **Quantum Teleportation** theorem. Using a shared EPR pair we can make a copy of a Qubit's superposition and transfer it on another Qubit, but the resulting operation will erase that superposition from the original Qubit (otherwise we would violate the No-Cloning Theorem). An EPR pair and two-bits worth of information is all we need to pack the info contained in a superposition in order to transfer it.

1.3 Quantum Gates

Classic computation is based on logic operators such as NOT, AND or OR; these "truth tables" are what we refer to when interpreting the Boolean value of a classic bit. Voltages are what we use to represent these values in our every-day modern machines, but in a Quantum computer all we can do is modify the rotation of a Qubit. We remind the reader that a Qubit's position is a vector in the Hilbert space of C^2 basically, since we have an orthonormal base of two elements; this means that the transformations we need in order to encode and interpret the logic operations are rotations of this vector. The only Quantum Mechanics law that we need to follow states that any Qubit state can only be modified by linear transformations that preserve lengths: in other words **Unitary Transformations**. These transformations also preserve inner products as well as angles. Now, via 2x2 rotation matrix, we can apply the transformations we need to the Qubit's state. A first example is the NOT gate, which flips the position according to the base we use.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad H = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Another very important gate is the Hadamard gate, which can be considered a sort of reflection in the Hilbert space. Its properties will be clear later, as we will see it will be a key piece for the circuit we are going to build. Using multiple Qubits we can create the so-called CNOT gate: a control Qubit's value determines whether the second Qubit will be NOTed or not. The Bell's State is achieved starting from two Qubits, applying an Hadamard gate to one of them and use it to control the other in a CNOT gate; the final position is the famous entangled pair.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

1.4 Quantum Circuit

Now, as the reader may have guessed already, up until now we have introduced all kinds of elements that are strictly connected with the old computing paradigms. What we are trying to achieve is to transform an already known circuit, meaning a classic Boolean one, into a Quantum equivalent. It's been proven that the gates we have just presented, specifically the Hadamard gate together with the CNOT gate, have the capabilities of representing any classical circuit we want: any function F computable by a classic circuit is efficiently convertible to a Quantum circuit. From this point of view, Quantum Computing is at least as powerful as a classic circuit: we can input classic bits of information into a Quantum Circuit and by rotating (thus introducing the superpositions), computing, and rotating back again, we will have a correct classic answer (no superpositions as outputs). The key difference here is that even though everything is based on a non-deterministic process, the result will be certain. **"Rotate, Compute, Rotate"** can be considered the model which Quantum Computing is built upon.

This different nature of computation brings, as we've seen, many advantages when performing certain types of tasks, Feynman was the first to observe this inherent speed-up. When we think of a function whose outcome has to be computed we imagine some kind of registers used as arguments to work on. Those values will be subject to the logic operators that build a circuit, and will influence some other kind of output registers' value. What a Quantum Circuit rely on is the so-called **Quantum Parallelism**. The Quantum superpositions do not simply act as Boolean registers to work on, but each component of the superposition, which again is relative to a base state, may be thought as a single argument of the function. This is the reason why Feynman correctly guessed that an exponential advantage could have been possible: since the number of states on a n Qubit machine is 2^n, we can perform an operation only once whenever a classic computer would need exponentially more times to do so. Again, the drawback is that the outcome is probabilistic, randomly chosen among all possible components, but a workaround is possible (i.e. the second rotation of the Q.C. model).

2 Quantum Fourier Transform

Another key piece of the puzzle that dictates this Quantum advantage is the implementation of the Fourier Transformation. In classical computation it is used in many different areas, from signal processing to information theory. The

Quantum Fourier Transform is the Quantum implementation of the Discrete Fourier Transform over the amplitudes of a state. Specifically, the QFT is an implementation of the Discrete Fourier Transform (DFT), which operates on N equally-distanced points on the [0,2pi) interval and yields a function whose domain is composed by integers between 0 and N-1. A DFT of a r-periodic function is itself a function concentrated on the multiples of N/r (if r divides N then the multiples of N/r will have non-zero value, otherwise other domain values will have a non-zero image).

The Discrete Fourier Transform acts on a vector $(x_0, ..., x_{N-1})$ and maps it to the vector $(y_0, ..., y_{N-1})$ in the following way

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk}$$

where $\omega_N^{jk} = e^{2\pi i \frac{jk}{N}}$

2.1 Definition of QFT

The QFT is a DFT variant where N is a power of 2, meaning that we operate on the Quantum states: $\sum_{i=0}^{N-1} x_i |i\rangle$ is mapped to the state $\sum_{i=0}^{N-1} y_i |i\rangle$ according to the formula

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_N^{jk}$$

It is important to notice that only the amplitudes of the state are affected by the mapping.

Also, it is convenient to express a generic state $|j\rangle$ as the tensor product $|j_0 j_1 ... j_{n-1}\rangle$ $(N = 2^n)$ of the bits of the binary representation $j_0 * 2^{n-1} + j_1 * 2^{n-2} + \ldots + j_{n-1} * 2^0$. Using the Bra-Ket notation, we obtain:

$$|x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{xy} |y\rangle$$

Intuitively, the QFT is a map between the computational Z basis and the Fourier basis denoted with the tilde.

$$\text{QFT}|x\rangle = |\widetilde{x}\rangle$$

The H-gate performs the QFT operation on a single Qubit, thus transforming the state $|0\rangle$ and $|1\rangle$ to the state $|+\rangle$ and $|-\rangle$.

If we apply the H operator to a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$,we obtain the new state

$$\frac{1}{\sqrt{2}}(\alpha + \beta)|0\rangle + \frac{1}{\sqrt{2}}(\alpha - \beta)|1\rangle \equiv \tilde{\alpha}|0\rangle + \tilde{\beta}|1\rangle$$

This is the QFT application with N=2.

As we will see shortly, the same reasoning can be applied to a multi Qubit state, but with the addition of a "controller" gate. Hadamard and "controlling" gates are all we need to implement the DFT in a Quantum machine: the QFT.

For $N = 2^n$ the QFT acts on the state $|x\rangle = |x_1 \dots x_n\rangle$ where x_1 is the most significant bit. The mathematical expression over $N = 2^n$ is:

$$\mathrm{QFT}_N|x\rangle = \tfrac{1}{\sqrt{N}} \textstyle\sum_{y=0}^{N-1} \omega_N^{xy}|y\rangle$$
$$= \tfrac{1}{\sqrt{N}} \textstyle\sum_{y=0}^{N-1} e^{2\pi i xy/2^n}|y\rangle$$
$$= \tfrac{1}{\sqrt{N}} \textstyle\sum_{y=0}^{N-1} e^{2\pi i \left(\sum_{k=1}^{n} y_k/2^k\right)x}$$
$$= \tfrac{1}{\sqrt{N}} \textstyle\sum_{y=0}^{N-1} \prod_{k=1}^{n} e^{2\pi i x y_k/2^k}|y_1 \dots y_n\rangle$$
$$= \tfrac{1}{\sqrt{N}} \textstyle\bigotimes_{k=1}^{n} \left(|0\rangle + e^{2\pi i x/2^k}|1\rangle\right)$$
$$= \tfrac{1}{\sqrt{N}} \left(|0\rangle + e^{\frac{2\pi i}{2}x}|1\rangle\right)\otimes\left(|0\rangle + e^{\frac{2\pi i}{2^2}x}|1\rangle\right)\otimes\dots\otimes\left(|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}|1\rangle\right)\otimes\left(|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle\right)-$$

2.2 QFT Circuit

As we have briefly already seen, the circuit that implements the QFT is made of two gates: the first is a single Qubit Hadamard gate.

$$H|x_k\rangle = |0\rangle + \exp\left(\frac{2\pi i}{2}x_k\right)|1\rangle$$

The second is a two Qubit controlled rotation $CROT_k$ in the form

$$CROT_k = \begin{bmatrix} I & 0 \\ 0 & UROT_k \end{bmatrix}$$

where $UROT_k$ is

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

The result of a controlled rotation where the first Qubit is the controller and the second is the target is

$$CROT_k|0x_j\rangle = |0x_j\rangle$$

$$CROT_k|1x_j\rangle = \exp\left(\frac{2\pi i}{2^k}x_j\right)|1x_j\rangle$$

Now we can build a n-Qubit QFT!

The circuit works as follow. Starting with an n-Qubit state $|x_1x_2 \dots x_n\rangle$ we initially apply an H gate to the first Qubit changing the state into:

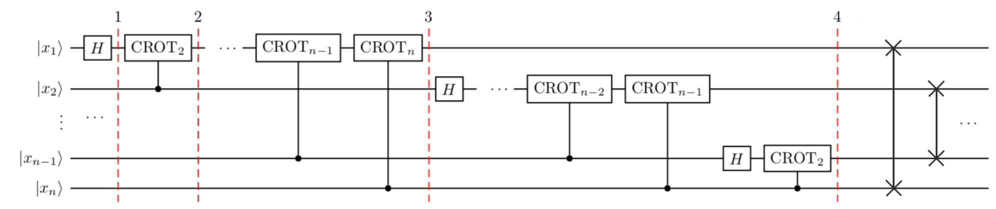


Figure 1: QFT Circuit

$$H_1|x_1x_2\dots x_n\rangle = \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2}x_1\right)|1\rangle\right] \otimes |x_2x_3\dots x_n\rangle$$

Then a $CROT_2$ is applied on Qubit 1 with Qubit 2 as controller, the state now becomes:

$$\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1\right)|1\rangle\right] \otimes |x_2x_3\dots x_n\rangle$$

Following the application of all the $CROT_n$ gates on Qubit 1 controlled by the corresponding Qubit, the new state is:

$$\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^n}x_n + \frac{2\pi i}{2^{n-1}}x_{n-1} + \dots + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1\right)|1\rangle\right] \otimes |x_2x_3\dots x_n\rangle$$

To simplify the expression now we can rewrite $x = 2^{n-1}x_1 + 2^{n-2}x_2 + \dots + 2^1x_{n-1} + 2^0x_n$ to get a more compact form

$$\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^n}x\right)|1\rangle\right] \otimes |x_2x_3\dots x_n\rangle$$

After the application of this sequence of gates over the whole initial state, we derive the final outcome that is the application of the QFT:

$$\frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^n}x\right)|1\rangle\right] \otimes \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^{n-1}}x\right)|1\rangle\right] \otimes \dots$$
$$\otimes \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^2}x\right)|1\rangle\right] \otimes \frac{1}{\sqrt{2}}\left[|0\rangle + \exp\left(\frac{2\pi i}{2^1}x\right)|1\rangle\right]$$

Finally a sequence of SWAP is applied to reverse the order of the Qubits. Note that only the last Qubit is influenced by all the others in the form a phase shift: moving away from the first, each Qubit depends less and less on the input states.This foresight is very useful in real physical application because rotations between near Qubits are easier to achieve with respect to the farther one. As it turns out we can ignore rotations below a certain threshold and still get decent results, this is known as the **approximate QFT**. This is also important in physical implementations, as reducing the number of operations can greatly reduce incoherence and potential gate errors.

3 Quantum Phase Estimation

Quantum Phase Estimation is one of the most useful procedure in Quantum Computing therefore it is used in many Quantum algorithm. The main focus of the QPE is, given an unitary operator U, estimating θ in $U|\psi\rangle = e^{\boldsymbol{2\pi i\theta}}|\psi\rangle$ where $|\psi\rangle$ is an eigenvector of the matrix U and $e^{\boldsymbol{2\pi i\theta}}$ its eigenvalues.

Intuitively the procedure uses the phase kickback to encode the phase of U in the Fourier basis to the counting register.

3.1 QPE Circuit

The circuit is made of two Quantum register the first is initialized with $|0\rangle^{\otimes n}$ and it is used to encode the whole phase while the second contains the state $|\psi\rangle$

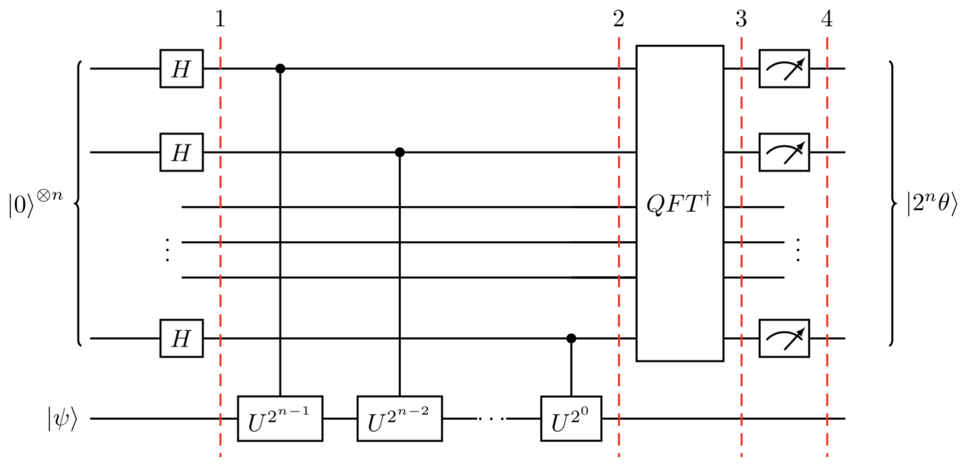


Figure 2: QPE Circuit

We can represent the initial circuit state as:

$$\psi_0 = |0\rangle^{\otimes n}|\psi\rangle$$

Now we put the the first register in superposition applying a H-gate on every Qubit, the result global state is:

$$\psi_1 = \frac{1}{2^{\frac{n}{2}}}\left(|0\rangle + |1\rangle\right)^{\otimes n}|\psi\rangle$$

Now we have to apply a controlled rotation through the unitary operator U on the second register, it is done only if the corresponding control Qubit is in the state $|1\rangle$. The sequence of U operator can be seen as

$$U^{2^j}|\psi\rangle = U^{2^j-1}U|\psi\rangle = U^{2^j-1}e^{2\pi i\theta}|\psi\rangle = \cdots = e^{2\pi i 2^j\theta}|\psi\rangle$$

This means that the resulting state is in the following form:

$$\psi_2 = \frac{1}{2^{\frac{n}{2}}} \left(|0\rangle + e^{\boldsymbol{2\pi i}\theta 2^{n-1}}|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{\boldsymbol{2\pi i}\theta 2^{1}}|1\rangle\right) \otimes \left(|0\rangle + e^{\boldsymbol{2\pi i}\theta 2^{0}}|1\rangle\right) \otimes |\psi\rangle$$

$$= \frac{1}{2^{\frac{n}{2}}} \sum_{k=0}^{2^n-1} e^{\boldsymbol{2\pi i}\theta k}|k\rangle \otimes |\psi\rangle$$

Where k is the integer representation of a n-bit binary number.

Note the similarity with the QFT, indeed it is exactly the result of the application of the QFT only that now we encode an arbitrary phase θ in the exponent. This is the reason of the application of the inverse QFT, in that we want to retrieve the phase encoded in the exponent to use it in computational basis.

$$|\psi_3\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{k=0}^{2^n-1} e^{\boldsymbol{2\pi i}\theta k}|k\rangle \otimes |\psi\rangle \xrightarrow{\mathcal{QFT}_n^{-1}} \frac{1}{2^n} \sum_{x=0}^{2^n-1} \sum_{k=0}^{2^n-1} e^{-\frac{2\pi i k}{2^n}(x-2^n\theta)}|x\rangle \otimes |\psi\rangle$$

The final measurement allows us to read the value of

$$|\psi_4\rangle = |2^n\theta\rangle \otimes |\psi\rangle$$

and finally obtain the phase.

The QPE doesn't seem very helpful for now since we have to know the initial value of the phase θ to built the circuit, but in the next section we point out how to create a circuit in which θ is a secret and how to take advantages out of it.

4 Description of the algorithm

In this section we describe one of the most important Quantum algorithms that is also the main topic of this paper: the Shor's factoring algorithm. The problem facing us is how to efficiently factor a number. The best result achievable so far on a classical machine is the General Number Sieve Field algorithm which has a sub-exponential complexity in the order of $O(e^{1,9(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}})$. We present now a brief description of the factorization problem, as well as how Shor's algorithm works in solving it. Lastly, in the Quantum Period finding section we describe why it works.

4.1 Factorization

Given an positive odd and composite integer number N, the problem of factorization is to find the prime factors of N. Determining if a number N is prime or composite is not a computationally hard problem since many probabilistic algorithm such as Miller-Rabin primality test can do it with a relative small error probability; on the contrary factorizing a number is not so easy. Thanks to some modular arithmetic properties now we see how to obtain factors from the period. Integers less than N and coprime with it form the multiplicative group of integers modulo N. We have to chose a random value a out of this group and, since the group's size $\varphi(N)$ is finite, the base a must have a finite order r. The order r is the smallest positive integer such that:

$$a^r = 1 \bmod N$$

Therefore N must divides $a^r - 1$. Assuming that we know r, that it is even and $a^{\frac{r}{2}} \neq -1 \bmod N$, we suppose the prime factorization of N is equal to $N = p_1^{n_1} \cdots p_m^{n_m}$. If a is chosen randomly between 1 and N such that a is coprime with N and r is the order, then:

$$P(r \textit{ is even AND } a^{\frac{r}{2}} \neq -1 \bmod N) \geq 1 - \frac{1}{2^m}$$

Now we claim that a proper factor of N is given by $p = gcd(a^{\frac{r}{2}} - 1, N)$ OR $gcd(a^{\frac{r}{2}} + 1, N)$.

The only challenge now is how to find the order r.

4.2 Algorithm steps

The algorithm is composed of five main procedures of which only the sub-routine of the period finding requires the use of a Quantum machine, all the others can be performed efficiently on a traditional hardware. The steps can be summarized as follow:

1. Choose a random number a between 1 and $N - 1$
2. With Euclidean algorithm calculate $gcd(a, N)$. If $gcd(a, N) > 1$ we find a non trivial factor of N otherwise go to step 3
3. Use Quantum period finding sub-routine to discover r the order of a mod N
4. If r is odd or is even and $a^{\frac{r}{2}} = -1 \bmod N$ return to step 1
5. With Euclidean algorithm calculate $gcd(a^{\frac{r}{2}} - 1, N)$ and $gcd(a^{\frac{r}{2}} + 1, N)$ if one of the two resulting integer is a non trivial factor of N the algorithm ends successfully otherwise go to step 1

4.3 Quantum Period finding

The last point that remains to be clarified is how the Quantum period finding works. We reformulate our period finding problem into a phase estimation problem to find the period of a function defined as:

$$f(a) = x^a \bmod N$$

In order to do that we have to define an operator U whose eigenvalues contain the period of the function f. Considering that QPE is about finding the eigenvalue of a unitary operator we can use it to compute the period of f. The operator U is defined as:

$$U|y\rangle = |xy \mod N\rangle$$

Its eigenvalue is $e^{\boldsymbol{2\pi i\theta}}$ and the eigenvector $U|\psi\rangle = e^{\boldsymbol{2\pi i\theta}}|\psi\rangle$ where θ is equal to $\frac{s}{r}$ with r period of $x^a \bmod N$.

The eigenvector can be seen as:

$$|\psi\rangle = \frac{1}{r^{\frac{1}{2}}} \sum_{k=0}^{r-1} e^{-2\pi i \frac{sk}{r}} |x^k \bmod N\rangle$$

Fortunately we do not need to know the period r directly, in fact we just compute the superposition of all the inputs and the QPE will output the phases of all eigenvectors in parallel. The final measurement will allow us to find the period.

Now we see how the Shor circuit is built. Two register are needed: the second should have enough bits to represent N ($n = \log_2 N$) while the first has t Qubits. In order to have enough precision for the phase estimation, we can use $t \geq 2n$.

We can represent the initial circuit state as:

$$\psi_0 = |0\rangle^{\otimes t}|1\rangle^{\otimes n}$$

The state $|1\rangle$ on the second register can be achieved using a simple X-gate on each Qubit.

Now we put the the first register in superposition applying a H-gate on every Qubit, the resulting global state is :

$$\psi_1 = \frac{1}{2^{\frac{t}{2}}} \left(|0\rangle + |1\rangle\right)^{\otimes t} |1\rangle^{\otimes n} \;\; = \frac{1}{2^{\frac{t}{2}}} \sum_{j=0}^{2^t-1} |j\rangle \otimes |1\rangle^{\otimes n}$$

Now we have to apply the series of controlled rotation on the second register. This operation brings the circuit in the state:

$$\psi_2 = \frac{1}{(r2^t)^{\frac{1}{2}}} \sum_{s=0}^{r-1} \sum_{j=0}^{2^t-1} e^{\boldsymbol{2\pi i} j \frac{s}{r}} |j\rangle \otimes |\psi\rangle$$

That is an approximation of:

$$\psi_2 = \frac{1}{2^{\frac{t}{2}}} \sum_{j=0}^{2^t-1} |j\rangle |x^j \bmod N\rangle$$

Applying the inverse QFT to the first register the state becomes:

$$\psi_3 = \frac{1}{r^{\frac{1}{2}}} \sum_{s=0}^{r-1} |\frac{s}{r}\rangle|\psi\rangle$$

The final measurement gives us the result $\frac{s}{r}$.

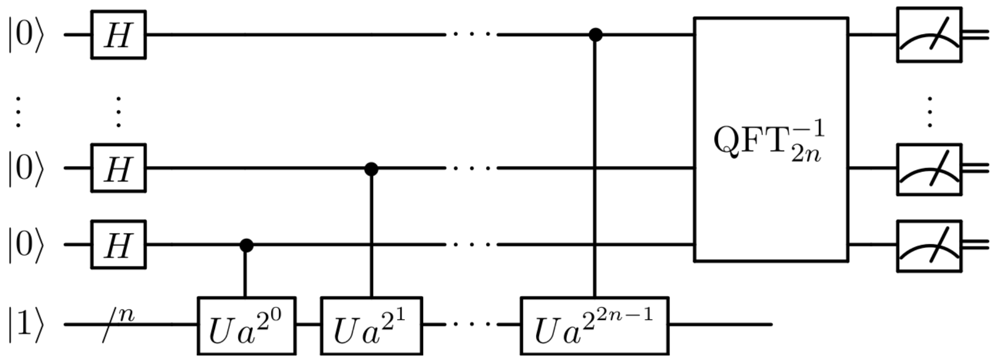


Figure 3: QPF Circuit

At this point we only have to derive the order from the previous expression. We use the mathematical technique of continuous fractions which allows us to approximate any real number with a sequence of rational numbers in the form:

$$[a_0, a_1, a_2, \cdots, a_p] = a_0 + \cfrac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\cdots + \frac{1}{a_p}}}}$$

For a real number c we define the expansion in continuous fractions in this way. Starting from c we built recursively the sequences $[a_j]_{j\geq 0}$ and $[r_j]_{j\geq 0}$,taking:

$$a_0 = \lfloor c \rfloor \quad r_0 = c - a_0$$

$$a_j = \lfloor \frac{1}{r_{r-1}} \rfloor \quad r_j = \frac{1}{r_{j-1}} - \lfloor \frac{1}{r_{r-1}} \rfloor$$

So the number c can be expressed as:

$$c = a_0 + \cfrac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\cdots + \frac{1}{a_j} + r_j}}}$$

The number $[a_0, a_1, a_2, \cdots, a_j]$ is said to be $j-$ *convergent* of c. If $r_j = 0$ the expansion ends with a_j and we have the equality $c = [a_0, a_1, a_2, \cdots, a_j]$

We are interested in the case when the number c is rational such as $\frac{s}{r}$ because only in this specific case the expansion in continuous fractions is finite. From this statement we can derive the numerator and the denominator of a rational number having its expansion in continuous fractions. The rational number $\frac{p_j}{q_j} = [a_0, a_1, a_2, \cdots, a_j]$ where:

$$p_0 = a_0 \quad q_0 = 1$$

$$p_1 = 1 + a_0 a_1 \quad q_1 = a_1$$

$$p_j = a_j p_{j-1} + p_{j-2} \quad q_j = a_j q_{j-1} + q_{j-2}$$

At this point we are able say that $\frac{s}{r}$ is a *convergent* of the estimated phase and r is the order we are looking for if $x^r = 1 \bmod N$ otherwise the algorithm fails.

Now we have seen how to calculate the order of $a \bmod N$ using the Quantum period finding and some mathematical tricks, in the next section we will examine why this new approach is advantageous compared to classical non-Quantum methods.

5 Analysis & Complexity

As we have already described, part of today's communication methods rely heavily on the fact that factorization is an inherently difficult problem. The best classic algorithm, the General Number Field Sieve, achieves a sub-exponential complexity. The complexity of Shor's Algorithm is polynomial, and is equal to $O((\log n)^2(\log\log n)(\log\log\log n))$; specifically it factors a number N in time complexity $O((\log N)3)$ and $O(\log N)$ space complexity.

As we have discussed already, this is achieved at the cost of determinism: the result is obtained through a repetition of the algorithm itself, since the measurement of the last rotation of the process is essentially a random pick among the amplitudes of the estimation, and comparing it with the input to check whether it is a trivial result or not.

5.1 Post-Quantum cryptography

Now, as the reader may have already guessed, the reason why Shor's algorithm stood up in the mid 90s as being quite a revolution has to do with the fact that its capabilities question today's communication security. Today's encryption methods are, in one way or another, solvable much more efficiently using this algorithm. The RSA algorithm is an immediate victim, since its security is based on the fact that factorization is a "hard" problem, but also ElGamal and elliptic curves based algorithms are greatly affected by this incredible speed up.

To make up for this problems, researchers have started developing a new field of cryptography called **Post-Quantum** cryptography: its focus is to study and classify Quantum-resistant algorithms for encryption. Currently we are still at quite an early stage in this field as Quantum machines can only handle a very limited amount of Qubits at the same time, as such the "threat" isn't immediate.

Nonetheless, we can safely assume that from a security point of view of communication a new era has come, as a consequence any new algorithm involved in this field has to take into consideration this new computation paradigm.

6 Qiskit Implementation

To give an idea of how to write a Quantum algorithm we have implemented Shor's algorithm and executed some tests.

For the implementation the IBM Qiskit framework has been used: it allows users to run Quantum circuits locally on a classical machine or remotely on a real Quantum device.

From the online documentation :

"Qiskit is an open-source framework for working with Quantum computers at the level of circuits, pulses, and algorithms. A central goal of Qiskit is to build a software stack that makes it easy for anyone to use Quantum computers. However, Qiskit also aims to facilitate research on the most important open issues facing Quantum computation today. You can use Qiskit to easily design experiments and run them on simulators and real Quantum computers."

6.1 Code

```
from qiskit import *
from qiskit.visualization import plot_histogram
import math


#super position in the first reg
def super_pos(qc,n_qubit):
    for qubit in range(n_qubit):
         qc.h(qubit)
    return qc

#initialize the second reg
def one_state(qc,n_qubit,n_first):
    for qubit in range(n_qubit):
        qc.x(n_first+qubit)
    return qc


#phase estimation procedure
def phase_ext(qc,n_first,n_second,a):
    mul=1
    for one_qubit  in range(n_first):
        for i in range(mul):
            for second_qubit in range(n_second):
                qc.cu1(a,n_first-one_qubit-1,n_first+second_qubit)
        mul *=2
    return qc

#inverse QFT
def qft_inv(qc,n_first,a):
    # swap
    for qubit in range(n_first// 2):  # floor division
        qc.swap(qubit, n_first - qubit - 1)
    # rephase
    for i in range(n_first):
        for j in range(i):
            qc.cu1( 2*a / float(2 ** (i - j)), j, i)
        qc.h(i)
    return qc

#measure first reg
```

```
def measure (qc,n_qubit):
    bit=0
    for qubit in range(n_qubit):
        qc.measure(qubit,bit)
        bit +=1
    return qc

#continuos fractions of the phase of the result measurment
def cont_fraction(n,d):
    if d == 0: return []
    q = n//d
    r = n - q*d
    return [q] + cont_fraction(d,r)


#find the convergent of the continuos fractons
def find_converget(list,a,n):
    i=0
    r=0
    list_p=[]
    list_q=[]
    p=list[i]
    list_p.append(p)
    q=1
    list_q.append(q)
    #print("passo0",list_p, list_q)
    if (a**q)%n == 1 :
        r=q
        return r
    p=1+(list[i]*list[i+1])
    list_p.append(p)
    q=list[1]
    list_q.append(q)
    #print("passo1",list_p, list_q)
    if(a**q)%n == 1 :
        r=q
        return r
    else:
        j=2
        for i in range(len(list)-2):
            p=list[j]*list_p[j-1]+list_p[j-2]
            list_p.append(p)
            q=list[j]*list_q[j-1]+list_q[j-2]
            list_q.append(q)
            #print("passoj:", j, list_p, list_q)
            j +=1
            if (a**q)%n == 1 :
                r=q
                return r
    print("failure_convergent")
    return r

#control the order
def control_order (r,a,n):
    #r is odd or
    if  r%2 != 0 :
        return "failure_order_odd"
    #r is even
    if (a**(r//2)) % n == -1 :
        return "failure_order_mod_-1"
    return "order_ok_"

# factors cpmputation
def find_factor(order,a,n):
    fact_1 = math.gcd(int((a**(order//2)))-1,n)
    fact_2 = math.gcd(int((a**(order//2)))+1,n)
    if n % fact_1 == 0:
        #print("fact1",fact_1)
        fact_2  = int(n//fact_1)
        return (fact_1,fact_2)
    elif n % fact_2 == 0:
        #print("fact2", fact_2)
        fact_1 = int(n // fact_2)
        return (fact_1,fact_2)
    else:
        return "failure_factor"


#n is the number to factorize , a is the base , e number of simulations
def shor_alg(n,a,e):

    # number of bit to represent n
    bit = bin(n)
    m = len(bit[2:])

    # configure the circuit
    n_first = 2 * m
    n_second = m
    n_classical = n_first

    for i in range (e):
```

```
s_circuit = QuantumCircuit(n_first + n_second, n_classical)

super_pos(s_circuit, n_first)
one_state(s_circuit, n_second, n_first)
s_circuit.barrier()

phase_ext(s_circuit, n_first, n_second, a)
s_circuit.barrier()

qft_inv(s_circuit, n_first, a)
s_circuit.barrier()

measure(s_circuit, n_first)

# print(s_circuit)

# start simulation
backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(s_circuit, backend=backend, shots=shots).result()
answer = results.get_counts()

#print(answer)

plot_histogram(answer).show()

# read the data
# return the count of the most pr state
max_cont = max(answer.values())
total_shot = shots

phase_continuous = cont_fraction(max_cont, total_shot)

order = find_converget(phase_continuous, a, n)

order_status = control_order(order, a, n)
print(order_status ,    order)

#print("fattorizzare:", n, "base:", a, "ordine:", order)

# print("verifica che a^r mod N=", (a ** order) % n)

factors = find_factor(order, a, n)

print("fattore_di_n_:", factors)
print("\n")
```

6.2 Simulation results

To simulate the circuit, the "QUASM simulator" has been used; it is designed to simulate a real Quantum device. It runs the same circuit an arbitrary number of times using the variable **shots**, collecting each time the measurement outcome and returning a dictionary containing the final values. For our purpose **shots** has been set to the value 2048 in order to have a semblance of "real simulation value"and a reasonable computation time on our machine. The measurement shows the probability of any estimated phases: in the implementation we take the most likely.

In this simulation we try to factorize 21 with the base 2 and run the algorithm for 10 times (figure 4).

For readability reasons of the phase histogram, in this run the first Quantum register has been truncated to m Qubit, the same number of Qubit used to encode in binary the number to factorize (figure 5). Due to this approximation we find multiples of the order.

Notice how increasing the number of Qubit used to estimated the phase gives the real order of $2^6 \bmod 21 = 1$ (figure 6).

7 Conclusions

As we seen so far the Shor algorithm is a milestone in Quantum computing, it is one of the first applications of this new type of computation on a real world problem: factorization. Nowadays the number of Qubits on a real Quantum device and their incoherence does not allow us to have some useful application, but Noisy Intermediate-Scale Quantum (NISQ) technology will be available in the near future. Quantum computers with 50-100 Qubits may be able to perform tasks which surpass the capabilities of today's classical digital computers, but noise in Quantum gates is limiting the size of Quantum circuits that can be executed and expecting a reliable result from. NISQ devices will be useful tools for exploring many-body Quantum physics and may have other useful applications, but a 100 Qubit Quantum computer will not change the world right away: we should regard it as a significant step toward the more powerful Quantum technologies of the future. Quantum technologists should continue to strive for more accurate Quantum gates and, eventually, fully fault-tolerant Quantum Computing.

References

[1] Ryan O'Donnell. *Quantum Computation and Quantum Information.* Carnegie Mellon School of Computer Science `https://www.cs.cmu.edu/~odonnell/quantum18/`

[2] Ryan O'Donnell. *Lecture 1: Introduction to the Quantum Circuit Model.* Carnegie Mellon School of Computer Science `https://www.cs.cmu.edu/~odonnell/quantum15/lecture01.pdf`

[3] Richard P. Feynman. *Quantum Mechanical Computers.* `http://www.quantum-dynamic.eu/doc/feynman85_qmc_optics_letters.pdf`

[4] John Preskill. *Quantum Computing in the NISQ era and beyond.* Carnegie Mellon School of Computer Science `https://arxiv.org/pdf/1801.00862.pdf`

[5] Qiskit Documentation. `https://qiskit.org/documentation/`

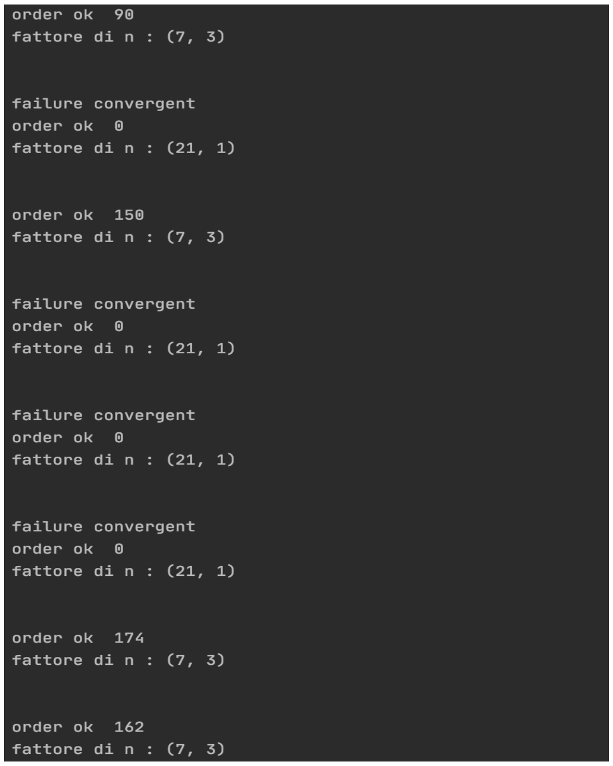
[6] Jonathan Hui. *QC - Period finding in Shor's Algorithm.* `https://medium.com/@jonathan_hui/qc-period-finding-in-shors-algorithm-7eb0c22e8202`

[7] `https://github.com/Michaelvll/myQShor`

[8] Elisa Baumer, Jan-Grimo Sobez, Stefan Tessarini. *Shor's Algorithm.* `https://qudev.phys.ethz.ch/static/content/QSIT15/Shors%20Algorithm.pdf`

[9] Peter W. Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer.*
`https://arxiv.org/pdf/quant-ph/9508027.pdf`

[10] Yongshan Ding, Pranav Gokhale. *Novel Computing Architectures and Technologies.*
`http://people.cs.uchicago.edu/~ftchong/33001/lecture08.pdf`



```
order ok  90
fattore di n : (7, 3)


failure convergent
order ok  0
fattore di n : (21, 1)


order ok  150
fattore di n : (7, 3)


failure convergent
order ok  0
fattore di n : (21, 1)


failure convergent
order ok  0
fattore di n : (21, 1)


failure convergent
order ok  0
fattore di n : (21, 1)


order ok  174
fattore di n : (7, 3)


order ok  162
fattore di n : (7, 3)
```

Figure 4: Shor outputs with m Qubit in the first register

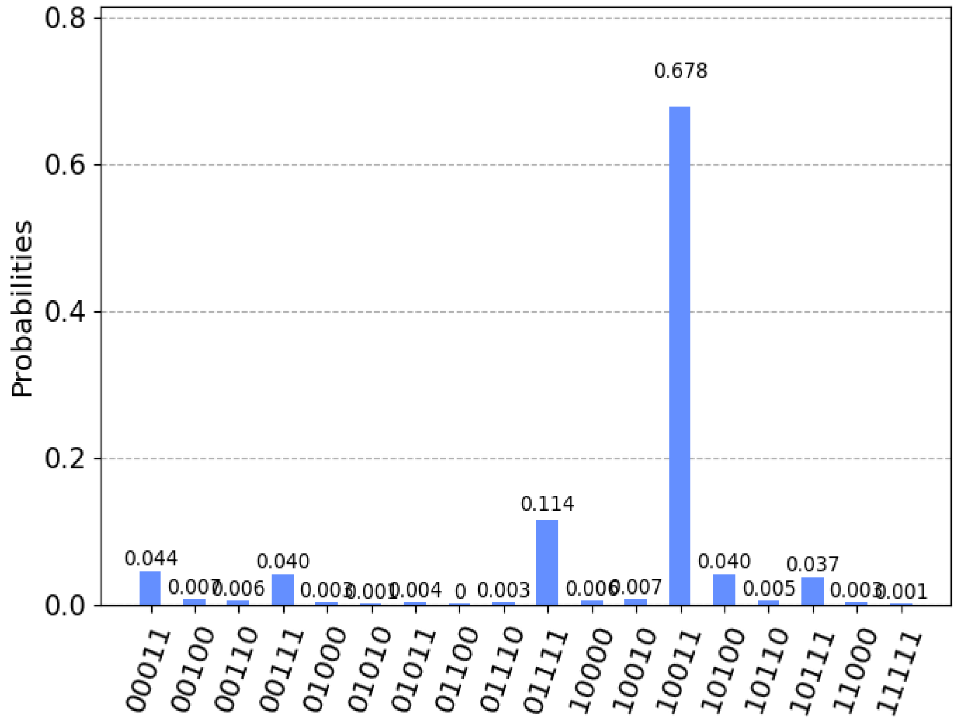
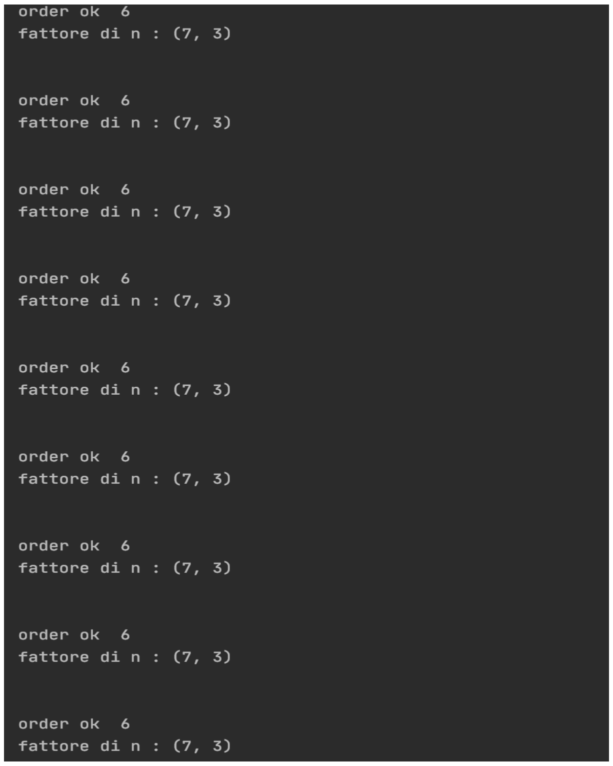


Figure 5: Phase probability in the first simulation



```
order ok  6
fattore di n : (7, 3)


order ok  6
fattore di n : (7, 3)


order ok  6
fattore di n : (7, 3)


order ok  6
fattore di n : (7, 3)


order ok  6
fattore di n : (7, 3)


order ok  6
fattore di n : (7, 3)


order ok  6
fattore di n : (7, 3)


order ok  6
fattore di n : (7, 3)


order ok  6
fattore di n : (7, 3)
```

Figure 6: Shor outputs with $2m$ Qubit in the first register