



**UANL**

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



**FCFM**

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

**UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN**  
**FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS**

## **Aplicaciones de la Mecánica Cuántica**

**Proyecto final:**

**Algoritmo de Shor**

Carlos Luna

Nombre:

Giovanni Gamaliel López Padilla

Ivan Arturo Pla Guzman

Matricula:

1837522

1837515

15 de octubre de 2020

# 1. Resumen

**Palabras clave:** Algoritmo de Shor, Transformada de Fourier Cuántica, Python, Qiskit, complejidad algorítmica

## 2. Introducción

En el área de la matemáticas, la factorización es una técnica que consiste en la descomposición en factores de una expresión algebraica en forma de un producto. El teorema fundamental de la aritmética cubre la factorización de números enteros, este teorema pertenece a la teoría de números. El teorema de factorización afirma lo siguiente:

**Teorema 1.** *Cada entero positivo tiene una única descomposición en números primos.*

El teorema fue demostrado por primera vez por Euclides, aunque la primer demostración completa apareció en las *Disquisitiones Arithmeticae* de Carl Friedrich Gauss.

El análisis numérico es el estudio de algoritmos, el cual consiste en un conjunto de instrucciones o reglas ordenadas y finitas que permiten realizar una actividad mediante pasos sucesivos con el proposito de resolver problemas matemáticos dentro de la matemática continua, esto da una aproximación numérica. Hay problemas sencillos de calcular como una raíz cuadrada, no tienen soluciones exactas o son costosas en tiempo para obtenerlas. Para obtener una aproximación útil, se debe conocer la precisión de los resultados y cuantos recursos se necesita para lograrlo, estos conceptos cuantificados son conocidos como *precisión, convergencia, estabilidad y complejidad*.

La complejidad algorítmica, informalmente, es una medida que permite a los programadores conocer la cantidad de recursos que necesita un algoritmo para resolver un problema en función de su tamaño. El objetivo es comparar la eficiencia de algoritmos a la hora de resolver un problema conocido [4]. Para realizar una clasificación en la complejidad de un algoritmo se usa normalmente la notación asintótica, la cual

es que por el número de operaciones básicas ejecutadas por un algirtmo se obtiene una función. Cada una se ve denotada por  $T(N)$ , donde  $N$  es el número de elementos numéricos dentro del algoritmo. Para valores pequeños de  $N$ , las constantes que acompañan a los términos de la función  $T$  pueden influir de manera significativa al coste tota y con ello obtener conclusiones erróneas respecto a la eficiencia del algoritmo. En cambio este tipo de análisis se realiza con números grandes de  $N$ . Deendiendo del análisis que se realice, podemos encontrar diferentes tipos de notaciones. La notación más usada es la llamada O grande y se denota por  $O$ .

**Definición 1.** Se dice que un algoritmo  $F$  tiene una complejidad  $O(G(N))$  si existen dos constantes  $C$  y  $N_0$  para las que se cumpla  $|F(N)| < C \cdot G(N)$  para todo  $N > N_0$

En otras palabras, que el algoritmo  $F$  tiene una complejidad  $O(G)$  si el número de operaciones necesarias es constante para un número grande de  $N$ . Un ejemplo de esta clasificación puede observarse en la siguiente tabla: algorit

| Notación      | Nombre             | Ejemplo de algoritmo                       |
|---------------|--------------------|--|
| $O(1)$        | Constante          | Acceso a un elemento de un vector          |
| $O(\log N)$   | Logarítmica        | Búsqueda binaria                           |
| $O(N)$        | Lineal             | Búsqueda secuencial                        |
| $O(N \log N)$ | Lineal-Logarítmica | Algoritmo de ordenamiento <i>quicksort</i> |
| $O(N^2)$      | Cuadrática         | Algoritmo de ordenamiento simple           |
| $O(N^3)$      | Cúbica             | Multiplicación de matrices                 |
| $O(2^N)$      | Exponencial        | Partición de conjuntos                     |

Tabla 1: Ejemplos de algoritmos numéricos con su clasificación  $O$  de complejidad

La mayor parte de los algoritmos de factorización elementales son de proposito general, es decir, permiten descomponer cualquier número introducido, la diferencia entre algoritmos es el tiempo que se toman para encontrar la factorización del número dado. El problema de factorizar enteros de tiempo polinómico no ha sido resuelto

en computación clásica. Esto puede ser de gran ayuda al avance en el ámbito de la criptografía, ya que muchos sistemas criptográficos dependen de la imposibilidad de ser resueltos en un tiempo corto. La complejidad de este problema se encuentra en el núcleo de varios sistemas criptográficos importantes. Un algoritmo veloz para la factorización de enteros significaría que el algoritmo de clave pública RSA es inseguro. Si un número grande, de  $b$  bits es el producto de dos primos de aproximadamente el mismo tamaño, no existe algoritmo conocido capaz de factorizarlo en tiempo polinómico. Esto significa que ningún algoritmo conocido puede factorizarlo en tiempo  $O(b^k)$ , para cualquier constante  $k$ . Aunque, existen algoritmos que son más rápidos que  $O(b^k)$  para cualquier  $k$  mayor que 1. En otras palabras, los mejores algoritmos son súper-polinomiales, pero sub-exponenciales. En particular, el mejor tiempo asintótico de ejecución lo contiene el algoritmo de *criba general del cuerpo de números* [1] (CGCN), que para un número  $n$  es:

$$O\left(\exp\left((c + \epsilon_n) (\ln p)^{\frac{1}{3}} (\ln(\ln p))^{\frac{2}{3}}\right)\right). \quad (1)$$

Para una computadora ordinaria, la CGCN es el mejor algoritmo conocido para números grandes.

Al inicio de la década de 1960, Rolf Landauer se cuestionaba acerca de si las leyes físicas determinaban límites a los procesos de cómputo. En concreto se interesó por el origen del calor disipado por los ordenadores. Uno de los problemas actuales de los ordenadores de alta velocidad es el calor que se produce durante su funcionamiento. Cada vez se fabrican más microchips más pequeños en el cual se administra una mayor cantidad de transistores, sin embargo, existe un límite en el cual podemos ir disminuyendo el tamaño de cada transistor debido a que los canales donde circulan los electrones pueden escaparse por el fenómeno del efecto túnel, es por ello que la computación digital clásica no debe estar muy lejos del límite.

Las ideas esenciales de la computación cuántica surgieron en los primeros años de la década 1980, esto de la mente de Paul Beinoff, quien hacía operar computadoras clásicas con operaciones cuánticas. Entre 1981 y 1982, Richard Feynman propuso

el uso de fenómenos cuánticos para realizar cálculos computacionales exponiendo la idea que, dada la naturaleza de la mecánica cuántica algunos procesos se realizarían en un menor periodo de tiempo a comparación de una computadora clásica. En 1985, David Deutsch describió el primer computador cuántico universal, este ordenador era capaz de simular cualquier otro computador cuántico por medio de la tesis de Church-Turing ampliado, el cual es un enunciado formalmente indemostrable, no obstante, tiene una aceptación universal. Las tesis de Church y Turing son las siguientes:

**Tesis 1** (de Church). *La clase de las funciones que pueden ser calculadas mediante un algoritmo coincide con la clase de las funciones recursivas.*

**Tesis 2** (de Turing). *La clase de las funciones que pueden ser calculadas mediante un método definido coincide con la clase de las funciones calculables mediante una Máquina de Turing.*

En base a estos enunciados, surgió la idea de que un computador cuántico podría ejecutar diferentes algoritmos cuánticos. A lo largo de la década de 1990, la teoría acerca de la computación cuántica empezó a tener sus primeras aplicaciones en forma de algoritmos cuánticos, el primero en realizar estos aportes Dan Simon demostrando la ventaja de usar un computador cuántico frente a uno tradicional al comparar el modelo de probabilidad clásica con el modelo cuántico. Sus ideas sirvieron como base para el desarrollo de algoritmos de auténtico interés práctico. En el año 1993, Charles Bennett descubre el tele-transporte cuántico, que abre una nueva vía de investigación dirigida hasta las comunicaciones.

Entre 1994 y 1995 Peter Shor definió el algoritmo que permite calcular los factores primos de números a un tiempo menor a cualquier computadora clásica. En el año 2000, IBM diseña un computador cuántico de 5 qubits capaz de ejecutar un algoritmo de búsqueda de orden que forma parte del algoritmo de Shor. En 2001, nuevamente, IBM en conjunto con la Universidad de Standford, consiguen ejecutar por primera vez el algoritmo de Shor en el primer computador cuántico de 7-qubits, en el experimento se calcularon los factores primos de 15, dando como resultado 3 y 5 utilizando para ello 1018 moléculas, cada una de ellas con 7 átomos.

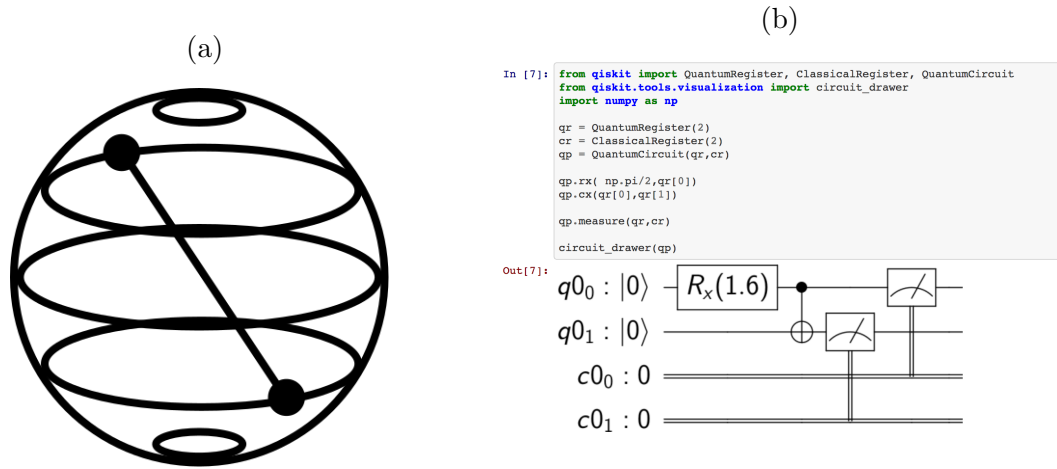


Figura 1: (a) Logo de Qiskit, (b) Ejemplo de un circuito cuántico implementado dentro de Jupyter Notebook de Python.

En el año 2017, IBM lanzó su primera versión de la librería Qiskit, la cual, es un framework de libre acceso para computación cuántica. El framework te provee de herramientas para la creación y manipulación de programas cuánticos y correrlos en los prototipos de dispositivos cuánticos llamados *IBM Q Experience*. Este framework está desarrollado en el lenguaje *Python*.

### 3. Objetivo

- Desarrollar el algoritmo de Shor para la factorización de un número dado usando la librería Qiskit en el lenguaje python.
- Calcular la diferencia de tiempo de procesamiento para un número  $N$  entre el algoritmo de criba general de cuerpo de números usando una computadora clásica, el algoritmo de Shor usando una computadora clásica y el algoritmo de Shor usando una computadora cuántica.

## 4. Marco teórico

### 4.1. Estados cuánticos y el término Qubit

### 4.2. Puertas lógicas clásicas y cuánticas

### 4.3. Operador de Hadamart

#### 4.3.1. Transformada de Fourier clásica y cuántica

### 4.4. Algoritmo de Shor

### 4.5. Implementación con Qiskit

## 5. Resultados

## 6. Discusión

## 7. Conclusiones

## 8. Código

## Referencias

- [1] Agrios. *Introducción a La Introducción a La Teoría de Números.*, volume 3. 2003.
- [2] G. P. Berman, G. D. Doolen, G. V. López, and V. I. Tsifrinovich. Nonresonant effects in the implementation of the quantum Shor algorithm. *Physical Review A - Atomic, Molecular, and Optical Physics*, 61(4):7, 2000.
- [3] Vicente Moret Bonillo. Principios Fundamentales De Computación Cuántica. *universidad de A Coruña*, pages 1–181, 2013.
- [4] Luanne S. Cohen and Tanya Wendling. Técnicas de diseño. *Técnicas de diseño*, pages 15–18, 1998.

- [5] Edward Gerjuoy. Shor's factoring algorithm and modern cryptography. An illustration of the capabilities inherent in quantum computers. *American Journal of Physics*, 73(6):521–540, 2005.
- [6] F. Ghisi and S. V. Ulyanov. The information role of entanglement and interference operators in Shor quantum algorithm gate dynamics. *Journal of Modern Optics*, 47(12):2079–2090, 2000.
- [7] Daniel Koch, Saahil Patel, Laura Wessing, and Paul M. Alsing. Fundamentals In Quantum Algorithms: A Tutorial Series Using Qiskit Continued. 2020.
- [8] Samuel J. Lomonaco and Louis H. Kauffman. A continuous variable Shor algorithm. pages 97–108, 2005.
- [9] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.
- [10] D E Ingenieros D E Telecomunicación. Números primos especiales y sus aplicaciones criptográficas Números primos especiales y sus aplicaciones criptográficas TRIBUNAL CALIFICADOR. 2003.
- [11] Lieven M.K. Vandersypen, Matthias Breyta, Gregory Steffen, Costantino S. Yannoni, Mark H. Sherwood, and Isaac L. Chuang. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(6866):883–887, 2001.
- [12] Anocha Yimsiriwattana and Samuel J. Lomonaco Jr. Distributed quantum computing: a distributed Shor algorithm. *Quantum Information and Computation II*, 5436:360, 2004.
- [13] S. S. Zhou, T. Loke, J. A. Izaac, and J. B. Wang. Quantum Fourier transform in computational basis. *Quantum Information Processing*, 16(3):1–19, 2017.