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

16 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 raiz cuadrada, no tienen soluciones exactas o son costosas en tiempo para obtenerlas.Para obtener una aproximación útil, se debe conocer la presición de los resultados y cuantos recursos se necesita para lograrlo, estos conceptos cuantificados son conocidos como *presición, 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 erroneas 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
$O(logN)$	Logarítmica	Búsqueda binaria
$O(N)$	Lineal	Búsqueda secuencial
$O(NlogN)$	Lineal-Logarítmica	Algoritmo de ordenamiento *quicksort*
$O(N^2)$	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 resulto en computación clásica. Esto puede ser de gran ayuda al avance en el ambito 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 $\mathrm{O}(b^K)$, para cualquier constante k. Aunque, existen algoritmos que son más rápidos que $\mathrm{O}(a^b)$ para cualquier a 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:

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

Para una computadora ordinaria, la CGCN es el mejor algoritmo conocido para números grandes.
Al inicio de la decada de 1960, Rolf Landauer se cuestionaba acerca de si las leyes físicas determinaban limites a los procesos de computo. 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 tunel, 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 decada 1980, esto de la mente de Paul Beinoff, quien hacia operar computadoras clásicas con operaciones cuánticas. Entre 1981 y 1982, Richard Feymann propusó el uso de fenónemos cuánticos para realizar cálculos computacio-

nales exponiendo la idea que, dada la naturaleza de la mecánica cuántica algunos procesos se realizarian 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.*

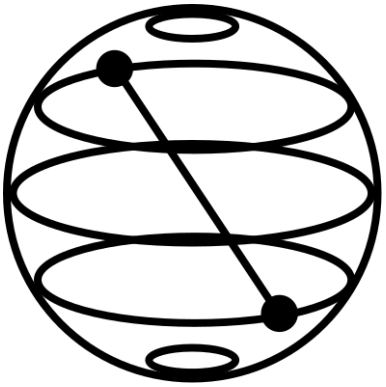
Tesis 2 (de Turing)**.** *La clase de las funciones que pueden ser calculadas mediante un método definido coincide con la clase de las funciones calculables mediante una Máquina de Turing.*

En base a estos enunciados, surgió la idea de que un computador cuántico podría ejecutar diferentes algoritmos cuánticos. A lo largo de la decada 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 Benett descubre el tele-transporte cuántico, que abre una nueva vía de investigación dirigida hasta las comunicaciones.
Entre 1994 y 1995 Peter Shor definió el algortimo 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. El algoritmo de Shor tiene una complejidad algorítmica [9]

$$O(logN)^3. \tag{2}$$

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.

(a)



(b)

```
In [7]: from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
        from qiskit.tools.visualization import circuit_drawer
        import numpy as np

        qr = QuantumRegister(2)
        cr = ClassicalRegister(2)
        qp = QuantumCircuit(qr,cr)

        qp.rx( np.pi/2,qr[0])
        qp.cx(qr[0],qr[1])

        qp.measure(qr,cr)

        circuit_drawer(qp)
```

Out[7]:

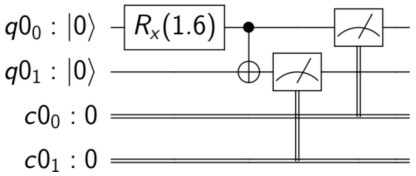


Figura 1: (a) Logo de Qiskitl, (b) Ejemplo de un circuito cuántico implementado dentro de Jupyter Notebook de Python.

En el año 2017, IBM lanzo su primer version de la libreria 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 esta 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 algortimo de Shor usando una computadora cuántica.

4. Marco teórico

4.1. Estados cuánticos y el término Qubit

En física cuántica, el estado cuántico es cualquier estado posible en el que puede estar un sistema mecánico cuántico. Un estado cuántico completamente especificado puede describirse mediante un vector de estado, una función de onda o un conjunto completo de números cuánticos para un sistema específico. Un estado cuántico parcialmente conocido, como un conjunto con algunos números cuánticos fijos, puede ser descrito por un operador de densidad.

Una mezcla de estados cuánticos es nuevamente un estado cuántico. Los estados cuánticos que no se pueden escribir como una mezcla de otros estados se denominan estados cuánticos puros, mientras que todos los demás estados se denominan estados cuánticos mixtos. Un estado cuántico puro se puede representar

mediante un rayo en un espacio de Hilbert sobre los números complejos, mientras que los estados mixtos se representan mediante matrices de densidad, que son operadores semidefinitos positivos que actúan en el espacio de Hilbert.

Un bit cuántico, qbit o qubit es la unidad básica de información cuántica, es la versión cuántica del clásico bit binario.

NOTA: AGREGAR NOTACION DE 00010 to 4

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

4.4.1. Periodo de la función a^xmod N

Se tiene la función

$$f(x) = a^x mod N \tag{3}$$

donde a y N son enteros positivos tal que $a < N$ y que no tienen ningún factor en común. El periodo (o el orden r), es el valor más pequeño diferente de cero tal que:

$$a^r mod N = 1 \tag{4}$$

Usando como ejemplos $a = 3$ y $N = 35$, se tiene el siguiente periodo:

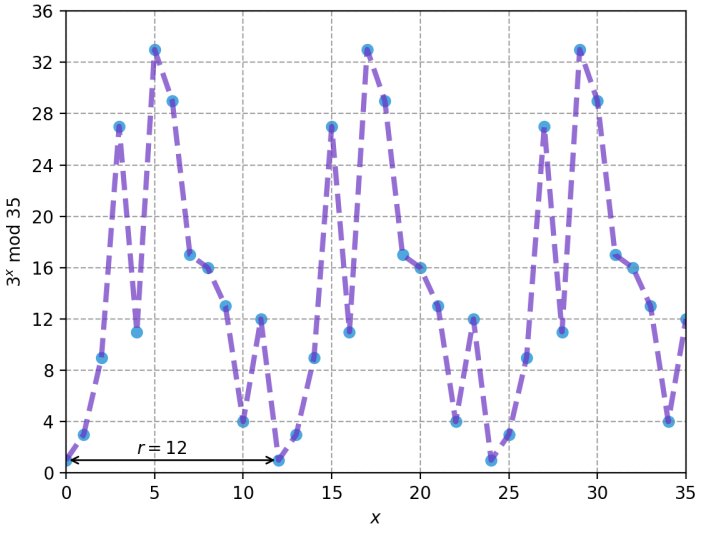


Figura 2: Periodo de la función 3 para visualizar la condición 4 usando el código 3

Para realizar esta operación dentro del Algoritmo de Shor se propusó un operador unitario tal que:

$$U|y\rangle \equiv |aymodN\rangle \tag{5}$$

Para visualizar el funcionamiento de este operador se usará el ejemplo de $a = 3$ y $N = 35$, trabajando con los eigenestados de U empezando con el estado $|1\rangle$ podemos visualizar que la transformación sucesiva del operador U estariasmos multiplicando el estado por $amodN$, por lo tanto, después de r transformaciones regresaremos al estado $|1\rangle$.

$$\begin{aligned} U|1\rangle &= |3\rangle \\ U^2|1\rangle &= |9\rangle \\ U^3|1\rangle &= |12\rangle \\ &\vdots \\ U^{r-1}|1\rangle &= |12\rangle \\ U^r|1\rangle &= |1\rangle \end{aligned}$$

Utilizando una superposición de estos estados, obtendremos lo siguiente:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \left| a^k mod N \right\rangle . \tag{6}$$

Realizando esa serie para $a = 3$ y $N = 35$, encontramos que el estado descrito en **??** es invariante bajo la transformación de U.

$$\begin{aligned} |u_0\rangle &= \frac{1}{\sqrt{12}} \left(|1\rangle + |3\rangle + |9\rangle + \cdots + |4\rangle + |12\rangle \right) \\ U|u_0\rangle &= \frac{1}{\sqrt{12}} \left(U|1\rangle + U|3\rangle + U|9\rangle + \cdots + U|4\rangle + U|12\rangle \right) \\ &= \frac{1}{\sqrt{12}} \left(|3\rangle + |9\rangle + |27\rangle + \cdots + |12\rangle + |1\rangle \right) \\ &= |u_0\rangle . \end{aligned}$$

Lo cual obtenemos que el eigenvalor es 1. Un eigenestado más general es aquel el cual contenga una fase diferente para cada base de estados. Especificamente, veremos el caso donde cada fase del k-esimo es proporcional a k, de tal manera que:

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i k}{r}} \left| a^k mod N \right\rangle \tag{7}$$

$$U|u_1\rangle = e^{\frac{2\pi i}{r}} |u_1\rangle . \tag{8}$$

Teniendo así un único eigenestado para cada valor entero en s donde $0 \leq s \leq r-1$. Esto es muy conveniente, ya que realizando la suma de los eigenestados, las diferencias de fases serán canceladas, obteniendo así el estado $|1\rangle$.

$$\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |u_s\rangle = |1\rangle .$$

Siendo así que el estado $|1\rangle$ es una superposición de estos eigenestados, lo cual indica que podemos realizar QPE al opeeador U usando el estado $|1\rangle$, para así obtener la medición de la fase:

$$\phi = \frac{s}{r}$$

Realizando del algoritmo de fracciones continuas sobre ϕ podemos encontrar el valor de ra. El circuito cuántico que realiza esta tarea es el siguiente:

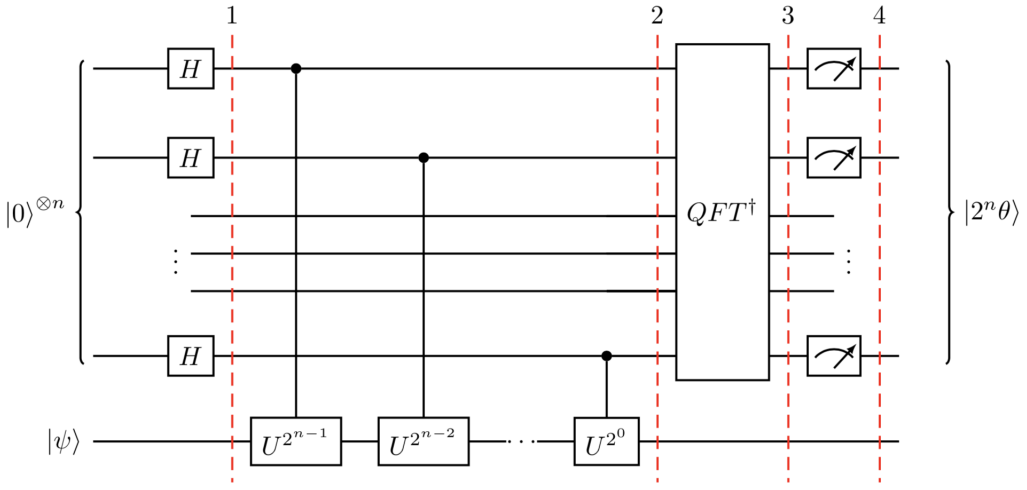


Figura 3: Circuito cuántico que realiza el algortimo QPE sobre una serie de qubits.

4.5. Implementación con Qiskit

Para mostrar de forma detallada se trabajara con el problema cuando $a = 7$ y $N = 15$. Con esto, realizamos el operador U aplicado al estado $|y\rangle$ de la siguiente manera:

$$U|y\rangle = |aymod15\rangle \tag{9}$$

Para crear las operaciones acumulativas U^x, se creará una repetición del circuito x veces, por lo que la función *c_amod15* regresará la compuerta U al valor de a y la añadira al circuito cuántico con la función *modular_exponentiation*, repetida x veces como lo siguiente:

[]:
```
def c_amod15(a, x):
    if a not in [2,7,8,11,13]:
        raise ValueError("'a' debe ser 2,7,8,11 o 13")
    U = QuantumCircuit(4)
    for iteration in range(x):
        if a in [2,13]:
            U.swap(0,1)
            U.swap(1,2)
            U.swap(2,3)
        if a in [7,8]:
            U.swap(2,3)
            U.swap(1,2)
            U.swap(0,1)
        if a == 11:
```

```
                U.swap(1,3)
                U.swap(0,2)
            if a in [7,11,13]:
                for q in range(4):
                    U.x(q)
        U = U.to_gate()
        U.name = "%i^%i mod 15" % (a, x)
        c_U = U.control()
        return c_U
    def modular_exponentiation(given_circuit, n, m, a):
        for x in range(n):
            exponent = 2**x
            given_circuit.append(a_x_mod15(a, exponent),
                         [x] + list(range(n, n+m)))
```

Al algortimo también se le implemento la transformación inversa de Fourier cuántica, se le nombre como *qft_dagger*, esta función es la siguiente:

[]:
```
    def qft_dagger(n):
        qc = QuantumCircuit(n)
        for qubit in range(n//2):
            qc.swap(qubit, n-qubit-1)
        for j in range(n):
            for m in range(j):
                qc.cu1(-np.pi/float(2**(j-m)), m, j)
            qc.h(j)
        qc.name = "iqft"
        return qc
```

Ya con estas funciones preestablecidas, el algoritmo de Shor es construido de una manera amigable a la vista, este es el siguiente:

[]:
```
    def shor_program(n, m, a):
        # set up quantum circuit
        shor = QuantumCircuit(n+m, n)
        # initialize the qubits
        initialize_qubits(shor, n, m)
        shor.barrier()
        # apply modular exponentiation
        modular_exponentiation(shor, n, m, a)
        shor.barrier()
        # apply inverse QFT
        apply_iqft(shor, range(n))
        # measure the first n qubits
        shor.measure(range(n), range(n))
        return shor
    n = 4; m = 4; a = 7
    mycircuit = shor_program(n, m, a)
    mycircuit.draw(output='text')
```

Al termino de estas lineas de código tendremos la serie de posibles factores primos del número 15, por lo que con ayuda de un algoritmo clásico se podra

comprobar si los cálculos antes realizados son correctos, este algoritmo es el siguiente:

[]:
```
from math import gcd

for measured_value in counts:
    measured_value_decimal = int(measured_value[::-1], 2)
    print(f"Medición {measured_value_decimal}")

    if measured_value_decimal % 2 != 0:
        print("Fallo. El número no es par")
        continue
    x = int((a ** (measured_value_decimal/2)) % 15)
    if (x + 1) % 15 == 0:
        print("Fallo. x + 1 = 0 (mod N) donde x = a^(r/2) (mod␣
↪N)")
        continue
    guesses = gcd(x + 1, 15), gcd(x - 1, 15)
    print(guesses)
```

Dando como resultado lo siguiente:

```
Medición 0
(1, 15)
Medición 8
(1, 15)
Medición 4
(5, 3)
Medición 12
(5, 3)
```

En donde se puede observar que no necesariamente el algoritmo cuántico encontrará la factorización que nosotros esperariamos, si no que necesita ayuda de una computadora clásica para comprobar sus resultados. Este circuito esta hecho Especificamente para factorizar el número 15, el circuito cuántico que factoriza cualquier número N es el siguiente:

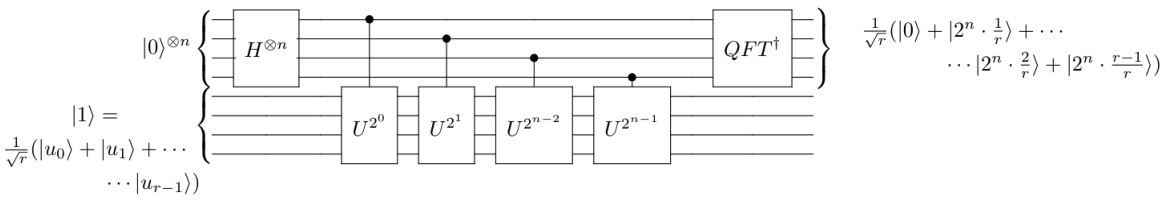


Figura 4: Algoritmo de Shor para la factorización de un número visto desde la estructura que propone Qiskit.

5. Resultados

6. Discusión

7. Conclusiones

8. Código

1. Shor_quantum.py
   Este algoritmo realiza la factorización de un número N en dos números primos, este código es capaz de correr en una computadora clásica y en el propotito de IBM Q Experience

2. Functions.py
   Esta serie de funciones son usadas en el programa *Shor_quantum.py*.

3. r_period.py
   Este código realiza el calculo del periodo r dados a y N, da como resultado la figura 2.

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. *univerasidad 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.