Quantum Computing Seminar 8

YongHyun "Aeren" An

Samsung Research

January 20, 2025

Integer Factorization with Quantum Computer

Problem (Integer factorization)

Input: N-bit integer A
Output: Factorization of A

Integer Factorization with Quantum Computer

Problem (Integer factorization)

Input: N-bit integer A
Output: Factorization of A

As we've discussed before, the best known classical algorithm has super-polynomial time complexity.

Integer Factorization with Quantum Computer

Problem (Integer factorization)

Input: N-bit integer A
Output: Factorization of A

As we've discussed before, the best known classical algorithm has super-polynomial time complexity.
We'll look at **Shor's algorithm**, which solves integer factorization problem using $O(N^2 \cdot \log(N))$ gates.

Order Finding Problem

Order Finding Problem

Definition (Euler's totient function)

Euler's totient function $\phi : \mathbb{Z}_{>0} \to \mathbb{Z}_{>0}$ is defined as
$\phi(n) =$ (Number of integers $1 \leq a < n$ which is coprime to n)

Order Finding Problem

Definition (Euler's totient function)

Euler's totient function $\phi : \mathbb{Z}_{>0} \to \mathbb{Z}_{>0}$ is defined as
$\phi(n) =$ (Number of integers $1 \leq a < n$ which is coprime to n)

Theorem

Let n be a positive integer with factorization $n = p_1^{e_1} \cdots p_k^{e_k}$. Then
$\phi(n) = (p_1^{e_1} - p_1^{e_1 - 1}) \cdots (p_k^{e_k} - p_k^{e_k - 1})$.

Order Finding Problem

Theorem

For all positive integers a and n with $\gcd(a, n) = 1$, $a^{\phi(n)} = 1 \mod n$

Order Finding Problem

Theorem

For all positive integers a and n with $\gcd(a, n) = 1$, $a^{\phi(n)} = 1 \mod n$

This theorem implies that there exists a positive integer k where $a^m = 1 \mod n$ if and only if k divides m, and such k must be a divisor of $\phi(n)$.

Order Finding Problem

Theorem

For all positive integers a and n with $\gcd(a, n) = 1$, $a^{\phi(n)} = 1 \mod n$

This theorem implies that there exists a positive integer k where $a^m = 1 \mod n$ if and only if k divides m, and such k must be a divisor of $\phi(n)$.

Definition (Multiplicative order)

Given a positive integer n and a positive integer a coprime to n, the **(multiplicative) order** of a modulo n is the minimum positive integer k with $a^k = 1 \mod n$.

Order Finding Problem

Problem (Order finding problem)

Input: Positive coprime integers a and n
Output: Order of a modulo n

Shor's Algorithm

Shor's Algorithm

Shor's algorithm consists of two parts.

1. Classical reduction of the integer factorization problem to the order finding problem
2. Quantum algorithm to solve order finding problem

Shor's Algorithm (Part 1)

Our goal is to factor a positive integer n given the order-finding oracle FindOrder_n.

Shor's Algorithm (Part 1)

Our goal is to factor a positive integer n given the order-finding oracle $\mathrm{FindOrder}_n$.

Algorithm	Factorize
Input	Positive integer n
Output	Factorization of n

Shor's Algorithm (Part 1)

Our goal is to factor a positive integer n given the order-finding oracle FindOrder_n.

Algorithm	Factorize
Input	Positive integer n
Output	Factorization of n

1. If n is prime, we're done.

Shor's Algorithm (Part 1)

Our goal is to factor a positive integer n given the order-finding oracle FindOrder_n.

Algorithm	Factorize
Input	Positive integer n
Output	Factorization of n

1. If n is prime, we're done.
2. Otherwise, let $d = \text{FindANontrivialFactor(n)}$. We output the combination of $\text{Factorize}(d)$ and Factorize(n/d).

Shor's Algorithm (Part 1)

Algorithm	FindANonTrivialFactor
Input	Positive composite integer n
Output	Non-trivial factor of n

Shor's Algorithm (Part 1)

Algorithm	FindANonTrivialFactor
Input	Positive composite integer n
Output	Non-trivial factor of n

1. If n is even, return 2.

Shor's Algorithm (Part 1)

Algorithm	FindANonTrivialFactor
Input	Positive composite integer n
Output	Non-trivial factor of n

1. If n is even, return 2.
2. For $e = 2, \cdots, \log_2(n)$ in this order, if $\text{floor}(n^{1/e})^e = n$, return $\text{floor}(n^{1/e})$.

Shor's Algorithm (Part 1)

Algorithm	FindANonTrivialFactor
Input	Positive composite integer n
Output	Non-trivial factor of n

1. If n is even, return 2.
2. For $e = 2, \cdots, \log_2(n)$ in this order, if $\text{floor}(n^{1/e})^e = n$, return $\text{floor}(n^{1/e})$.
3. Repeat the following indefinitely.
   3.1 Pick a random a with $2 \leq a < n$.

Shor's Algorithm (Part 1)

Algorithm	FindANonTrivialFactor
Input	Positive composite integer n
Output	Non-trivial factor of n

1. If n is even, return 2.
2. For $e = 2, \cdots, \log_2(n)$ in this order, if $\text{floor}(n^{1/e})^e = n$, return $\text{floor}(n^{1/e})$.
3. Repeat the following indefinitely.
   3.1 Pick a random a with $2 \leq a < n$.
   3.2 If $\gcd(a, n) > 1$, return $\gcd(a, n)$.

Shor's Algorithm (Part 1)

Algorithm	FindANonTrivialFactor
Input	Positive composite integer n
Output	Non-trivial factor of n

1. If n is even, return 2.
2. For $e = 2, \cdots, \log_2(n)$ in this order, if $\text{floor}(n^{1/e})^e = n$, return $\text{floor}(n^{1/e})$.
3. Repeat the following indefinitely.
   3.1 Pick a random a with $2 \leq a < n$.
   3.2 If $\gcd(a, n) > 1$, return $\gcd(a, n)$.
   3.3 If $\text{FindOrder}_n(a)$ is odd, go to step 3.

Shor's Algorithm (Part 1)

Algorithm	FindANonTrivialFactor
Input	Positive composite integer n
Output	Non-trivial factor of n

1. If n is even, return 2.
2. For $e = 2, \cdots, \log_2(n)$ in this order, if $\text{floor}(n^{1/e})^e = n$, return $\text{floor}(n^{1/e})$.
3. Repeat the following indefinitely.
   3.1 Pick a random a with $2 \leq a < n$.
   3.2 If $\gcd(a, n) > 1$, return $\gcd(a, n)$.
   3.3 If $\text{FindOrder}_n(a)$ is odd, go to step 3.
   3.4 If $\gcd(a^{\text{FindOrder}_n(a)/2} - 1, n) > 1$, return $\gcd(a^{\text{FindOrder}_n(a)/2} - 1, n)$.

Shor's Algorithm (Part 1)

Algorithm	FindANonTrivialFactor
Input	Positive composite integer n
Output	Non-trivial factor of n

1. If n is even, return 2.
2. For $e = 2, \cdots, \log_2(n)$ in this order, if $\mathrm{floor}(n^{1/e})^e = n$, return $\mathrm{floor}(n^{1/e})$.
3. Repeat the following indefinitely.
    3.1 Pick a random a with $2 \leq a < n$.
    3.2 If $\gcd(a, n) > 1$, return $\gcd(a, n)$.
    3.3 If $\mathrm{FindOrder}_n(a)$ is odd, go to step 3.
    3.4 If $\gcd(a^{\mathrm{FindOrder}_n(a)/2} - 1, n) > 1$, return $\gcd(a^{\mathrm{FindOrder}_n(a)/2} - 1, n)$.
    3.5 Otherwise, go to step 3.

Shor's Algorithm (Part 2)

Shor's Algorithm (Part 2)

Reminder

For a non-negative integer n and a complex number r,

$$1 + r + \cdots + r^{n-1} = \begin{cases} n & \text{if } r = 1 \\ \frac{1-r^n}{1-r} & \text{if } r \neq 1 \end{cases}$$

Shor's Algorithm (Part 2)

Definition (Primitive root of unity)

For a positive integer n,

$$\omega_n := e^{\frac{2\pi}{n}\mathrm{i}}$$

Shor's Algorithm (Part 2)

Definition (Primitive root of unity)

For a positive integer n,

$$\omega_n := e^{\frac{2\pi}{n}\mathrm{i}}$$

Theorem (Root-of-unity filter)

For a positive integer n and an integer k,

$$\omega_n^{0\cdot k} + \omega_n^{1\cdot k} + \cdots + \omega_n^{(n-1)\cdot k} = \begin{cases} n & \text{if } k = 0 \mod n \\ 0 & \text{if } k \neq 0 \mod n \end{cases}$$

Shor's Algorithm (Part 2)

Definition (Primitive root of unity)

For a positive integer n,

$$\omega_n := e^{\frac{2\pi}{n}\mathrm{i}}$$

Theorem (Root-of-unity filter)

For a positive integer n and an integer k,

$$\omega_n^{0\cdot k} + \omega_n^{1\cdot k} + \cdots + \omega_n^{(n-1)\cdot k} = \begin{cases} n & \text{if } k = 0 \mod n \\ 0 & \text{if } k \neq 0 \mod n \end{cases}$$

Proof

Shor's Algorithm (Part 2)

Definition (Primitive root of unity)

For a positive integer n,

$$\omega_n := e^{\frac{2\pi}{n}\mathrm{i}}$$

Theorem (Root-of-unity filter)

For a positive integer n and an integer k,

$$\omega_n^{0\cdot k} + \omega_n^{1\cdot k} + \cdots + \omega_n^{(n-1)\cdot k} = \begin{cases} n & \text{if } k = 0 \mod n \\ 0 & \text{if } k \neq 0 \mod n \end{cases}$$

Proof

- If $k = 0 \mod n$, $\omega_n^k = 1$, so $\omega_n^{0\cdot k} + \omega_n^{1\cdot k} + \cdots + \omega_n^{(n-1)\cdot k} = 1 + \cdots + 1 = n$.

Shor's Algorithm (Part 2)

Definition (Primitive root of unity)

For a positive integer n,

$$\omega_n := e^{\frac{2\pi}{n}\mathrm{i}}$$

Theorem (Root-of-unity filter)

For a positive integer n and an integer k,

$$\omega_n^{0\cdot k} + \omega_n^{1\cdot k} + \cdots + \omega_n^{(n-1)\cdot k} = \begin{cases} n & \text{if } k = 0 \mod n \\ 0 & \text{if } k \neq 0 \mod n \end{cases}$$

Proof

- If $k = 0 \mod n$, $\omega_n^k = 1$, so $\omega_n^{0\cdot k} + \omega_n^{1\cdot k} + \cdots + \omega_n^{(n-1)\cdot k} = 1 + \cdots + 1 = n$.
- If $k \neq 0 \mod n$, $\omega_n^k \neq 1$, so $\omega_n^{0\cdot k} + \omega_n^{1\cdot k} + \cdots + \omega_n^{(n-1)\cdot k} = \frac{1-\omega_n^{n\cdot k}}{1-\omega_n^k} = 0$.

Shor's Algorithm (Part 2)

Definition (Root-of-unity basis)

For a positive integer n, let $|b_k\rangle = \frac{1}{\sqrt{n}}(\omega_n^{0\cdot k}, \cdots, \omega_n^{(n-1)\cdot k})$ for integers $0 \leq k < n$. Then the set $\{ \, | \, b_0\rangle, \cdots, \, | \, b_{n-1}\rangle \, \}$ is an orthonormal basis of $\mathbb{C}^n$, called the **root-of-unity basis**.

Shor's Algorithm (Part 2)

Definition (Root-of-unity basis)

For a positive integer n, let $|b_k\rangle = \frac{1}{\sqrt{n}}(\omega_n^{0\cdot k}, \cdots, \omega_n^{(n-1)\cdot k})$ for integers $0 \le k < n$. Then the set $\{\ |\ b_0\rangle, \cdots, |\ b_{n-1}\rangle\ \}$ is an orthonormal basis of $\mathbb{C}^n$, called the **root-of-unity basis**.

Proof

Shor's Algorithm (Part 2)

Definition (Root-of-unity basis)

For a positive integer n, let $|b_k\rangle = \frac{1}{\sqrt{n}}(\omega_n^{0\cdot k}, \cdots, \omega_n^{(n-1)\cdot k})$ for integers $0 \leq k < n$. Then the set $\{\ |\ b_0\rangle, \cdots, \ |\ b_{n-1}\rangle\ \}$ is an orthonormal basis of $\mathbb{C}^n$, called the **root-of-unity basis**.

Proof

$$\begin{aligned}\langle b_i | b_j \rangle &= \frac{1}{n}\left(\overline{\omega_n^{0\cdot i}} \cdot \omega_n^{0\cdot j} + \cdots + \overline{\omega_n^{(n-1)\cdot i}} \cdot \omega_n^{(n-1)\cdot j}\right) \\ &= \frac{1}{n}\left(\omega_n^{0\cdot(j-i)} + \cdots + \omega_n^{(n-1)\cdot(j-i)}\right) \\ &= \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}\end{aligned}$$

Shor's Algorithm (Part 2)

Definition (Discrete Fourier transform)

For a positive integer n, **discrete Fourier transform** DFT_n is the n by n matrix transforming the standard basis vector $|i\rangle$ to the root-of-unity basis vector $|b_i\rangle$.

$$DFT_n := \begin{bmatrix} | & & | \\ |b_0\rangle & \cdots & |b_{n-1}\rangle \\ | & & | \end{bmatrix} = \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \omega_n^{i \cdot j} |i\rangle \langle j|$$

Shor's Algorithm (Part 2)

Definition (Discrete Fourier transform)

For a positive integer n, **discrete Fourier transform** DFT_n is the n by n matrix transforming the standard basis vector $|i\rangle$ to the root-of-unity basis vector $|b_i\rangle$.

$$DFT_n := \begin{bmatrix} | & & | \\ |b_0\rangle & \cdots & |b_{n-1}\rangle \\ | & & | \end{bmatrix} = \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \omega_n^{i \cdot j} |i\rangle \langle j|$$

Since $\{ b_0, \cdots, b_{n-1} \}$ is an orthonormal basis, DFT_n is a unitary matrix.

$$DFT_n^H = \begin{bmatrix} - & \langle b_0| & - \\ & \vdots & \\ - & \langle b_{n-1}| & - \end{bmatrix} = \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \omega_n^{-i \cdot j} |i\rangle \langle j|$$

Shor's Algorithm (Part 2)

Definition (Quantum Fourier transform)

For a non-negative integer m, **quantum Fourier transform** QFT_{2^m} is the unitary quantum gate on m-qubits, whose action corresponds to DFT_{2^m}.

Shor's Algorithm (Part 2)

Definition (Quantum Fourier transform)

For a non-negative integer m, **quantum Fourier transform** QFT_{2^m} is the unitary quantum gate on m-qubits, whose action corresponds to DFT_{2^m}.

Q) How would you perform the root-of-unity basis measurement on a given quantum state?

Implementing quantum Fourier transform

Shor's Algorithm (Part 2)

Implementing quantum Fourier transform

We recursively build QFT_{2^m}, starting from $QFT_{2^0} = I_1$. For an arbitrary integer $0 \leq x < 2^m$,

$$
\begin{aligned}
QFT_{2^m} |\text{binary}(x)\rangle &= \frac{1}{\sqrt{2^m}} \sum_{i=0}^{2^m-1} \omega_{2^m}^{x \cdot i} |\text{binary}(i)\rangle \\
&= \frac{1}{\sqrt{2^m}} \sum_{i=0}^{2^{m-1}-1} \left(\omega_{2^m}^{2 \cdot x \cdot i} |\text{binary}(2 \cdot i)\rangle + \omega_{2^m}^{2 \cdot x \cdot i + x} |\text{binary}(2 \cdot i + 1)\rangle \right) \\
&= \frac{1}{\sqrt{2^m}} \sum_{i=0}^{2^{m-1}-1} \omega_{2^{m-1}}^{x \cdot i} |\text{binary}(i)\rangle \otimes (|0\rangle + \omega_{2^m}^{x} |1\rangle) \\
&= QFT_{2^{m-1}} |\text{binary}(x - 2^{\lfloor \log_2(x) \rfloor})\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle + \omega_{2^m}^{x} |1\rangle)
\end{aligned}
$$

Implementing quantum Fourier transform

$$
\begin{aligned}
& QFT_{2^m} \left|\text{binary}(x)\right\rangle \\
& = QFT_{2^{m-1}} \left|\text{binary}(x - 2^{\lfloor \log_2(x) \rfloor})\right\rangle \otimes \frac{1}{\sqrt{2}} \left(\left|0\right\rangle + \omega_{2^m}^{x} \left|1\right\rangle\right) \\
& = \text{BITSWAPS}_{m-1\to 0} \left(\frac{1}{\sqrt{2}} \left(\left|0\right\rangle + \omega_{2^m}^{x} \left|1\right\rangle\right) \otimes QFT_{2^{m-1}} \left|\text{binary}(x - 2^{\lfloor \log_2(x) \rfloor})\right\rangle \right) \\
& = \text{BITSWAPS}_{m-1\to 0} \left(\left(I_2 \otimes QFT_{2^{m-1}}\right) \cdot \left(\frac{1}{\sqrt{2}} \left(\left|0\right\rangle + \omega_{2^m}^{x} \left|1\right\rangle\right) \otimes \left|\text{binary}(x - 2^{\lfloor \log_2(x) \rfloor})\right\rangle \right) \right)
\end{aligned}
$$

Implementing quantum Fourier transform

Reminder: action of H and CP

The action of the Hadamard gate H and the controlled phase gate CP is given by the following matrices

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, CP_\theta = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\theta \mathrm{i}} \end{bmatrix}$$

Shor's Algorithm (Part 2)

Implementing quantum Fourier transform

Reminder: action of H and CP

The action of the Hadamard gate H and the controlled phase gate CP is given by the following matrices

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, CP_\theta = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\theta \mathrm{i}} \end{bmatrix}$$

Therefore, for $x_0, \cdots, x_{m-1} \in \{0, 1\}$, $\alpha, \beta \in \mathbb{C}$ with $|\alpha|^2 + |\beta|^2 = 1$, and $k \in \{0, \cdots, m-2\}$,

- $H_{m-1} |\overline{x_{m-1} \cdots x_0}\rangle = \frac{1}{\sqrt{2}} (|0\rangle + (-1)^{x_{m-1}} |1\rangle) \otimes |\overline{x_{m-2} \cdots x_0}\rangle$, and
- $CP_{\theta,k,m-1} ((\alpha |0\rangle + \beta |1\rangle) \otimes |\overline{x_{m-2} \cdots x_0}\rangle) = \left(\alpha |0\rangle + \beta e^{x_k \theta \mathrm{i}} |1\rangle\right) \otimes |\overline{x_{m-2} \cdots x_0}\rangle$

Shor's Algorithm (Part 2)

Implementing quantum Fourier transform

Let $x = 2^0 x_0 + \cdots + 2^{m-1} x_{m-1}$ for binary integers $x_0, \cdots, x_{m-1}$. Then

$$
\begin{aligned}
&\frac{1}{\sqrt{2}} \left(|0\rangle + \omega_{2^m}^{x} |1\rangle \right) \otimes |\overline{x_{m-2} \cdots x_0}\rangle \\
&= \frac{1}{\sqrt{2}} \left(|0\rangle + \omega_{2^m}^{2^{m-1} x_{m-1}} \cdots \omega_{2^m}^{2^0 x_0} |1\rangle \right) \otimes |\overline{x_{m-2} \cdots x_0}\rangle \\
&= \frac{1}{\sqrt{2}} \left(|0\rangle + \omega_{2^1}^{x_{m-1}} \cdots \omega_{2^m}^{x_0} |1\rangle \right) \otimes |\overline{x_{m-2} \cdots x_0}\rangle \\
&= CP_{\frac{\pi}{2^{m-1}},0,m-1} \left(\frac{1}{\sqrt{2}} \left(|0\rangle + \omega_{2^1}^{x_{m-1}} \cdots \omega_{2^{m-1}}^{x_1} |1\rangle \right) \otimes |\overline{x_{m-2} \cdots x_0}\rangle \right) \\
&\vdots \\
&= CP_{\frac{\pi}{2^{m-1}},0,m-1} \cdots CP_{\frac{\pi}{2},m-2,m-1} \left(\frac{1}{\sqrt{2}} \left(|0\rangle + \omega_{2^1}^{x_{m-1}} |1\rangle \right) \otimes |\overline{x_{m-2} \cdots x_0}\rangle \right)
\end{aligned}
$$

Implementing quantum Fourier transform

$$
\begin{aligned}
&\frac{1}{\sqrt{2}}\left(|0\rangle + \omega_{2^m}^{x}|1\rangle\right) \otimes |\overline{x_{m-2}\cdots x_0}\rangle \\
&= CP_{\frac{\pi}{2^{m-1}},0,m-1}\cdots CP_{\frac{\pi}{2},m-2,m-1}\left(\frac{1}{\sqrt{2}}\left(|0\rangle + \omega_{2^1}^{x_{m-1}}|1\rangle\right) \otimes |\overline{x_{m-2}\cdots x_0}\rangle\right) \\
&= CP_{\frac{\pi}{2^{m-1}},0,m-1}\cdots CP_{\frac{\pi}{2},m-2,m-1}H_{m-1}|\mathrm{binary}(x)\rangle
\end{aligned}
$$

Implementing quantum Fourier transform

$$
\begin{aligned}
&\frac{1}{\sqrt{2}}\left(|0\rangle + \omega_{2^m}^{x}|1\rangle\right) \otimes |\overline{x_{m-2}\cdots x_0}\rangle \\
&= CP_{\frac{\pi}{2^{m-1}},0,m-1}\cdots CP_{\frac{\pi}{2},m-2,m-1}\left(\frac{1}{\sqrt{2}}\left(|0\rangle + \omega_{2^1}^{x_{m-1}}|1\rangle\right) \otimes |\overline{x_{m-2}\cdots x_0}\rangle\right) \\
&= CP_{\frac{\pi}{2^{m-1}},0,m-1}\cdots CP_{\frac{\pi}{2},m-2,m-1}H_{m-1}|\text{binary}(x)\rangle
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
&QFT_{2^m}|\text{binary}(x)\rangle \\
&= \text{BITSWAPS}_{m-1\to 0}\cdot\left(I_2 \otimes QFT_{2^{m-1}}\right)\cdot CP_{\frac{\pi}{2^{m-1}},0,m-1}\cdots CP_{\frac{\pi}{2},m-2,m-1}\cdot H_{m-1}|\text{binary}(x)\rangle
\end{aligned}
$$

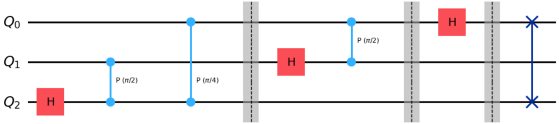
Shor's Algorithm (Part 2)

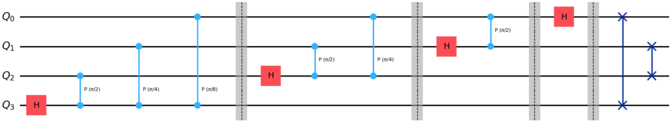
Implementing quantum Fourier transform

```
from qiskit import QuantumRegister, QuantumCircuit
from numpy import pi

def QFT(m):
    assert m >= 0
    q = QuantumRegister(m, "Q")
    C = QuantumCircuit(q)
    for i in reversed(range(m)):
        C.h(i)
        for j in reversed(range(i)):
            C.cp(pi / 2**(i-j), j, i)
        C.barrier()
    i, j = 0, m - 1
    while i < j:
        C.swap(i, j)
        i, j = i + 1, j - 1
    return C

display(QFT(3).draw(output = "mpl"))
display(QFT(4).draw(output = "mpl"))
```





Shor's Algorithm (Part 2)

Q) Asymptotically, how many gates does the circuit for QFT_{2^m} have?

Shor's Algorithm (Part 2)

Q) Asymptotically, how many gates does the circuit for QFT_{2^m} have? $O(m^2)$

Shor's Algorithm (Part 2)

Q) Asymptotically, how many gates does the circuit for QFT_{2^m} have? $O(m^2)$

Note fast Fourier transform, the best classical algorithm currently known for computing DFT, takes $O(m \cdot 2^m)$ time, assuming all the underlying field operation takes $O(1)$ time.

Shor's Algorithm (Part 2)

Problem (Phase estimation problem)

Input: A unitary quantum circuit for an operation U on n qubits, along with one of its unit eigenvector $|\psi\rangle$

Output: An approximation to the number $\theta \in [0, 1)$ satisfying $U|\psi\rangle = e^{2\pi\theta \mathrm{i}}|\psi\rangle$

Shor's Algorithm (Part 2)

Special Case

- We first solve with the assumption that $\theta = y/2^m$ for some integer $0 \le y < 2^m$.

Shor's Algorithm (Part 2)

Special Case

- We first solve with the assumption that $\theta = y/2^m$ for some integer $0 \leq y < 2^m$.
- The idea is to construct a state which has exactly one non-zero coordinate in the root-of-unity basis for each value of y.

Shor's Algorithm (Part 2)

Special Case

- We first solve with the assumption that $\theta = y/2^m$ for some integer $0 \le y < 2^m$.
- The idea is to construct a state which has exactly one non-zero coordinate in the root-of-unity basis for each value of y.

1. Initial state is $|\psi\rangle \otimes |\bar{0}^m\rangle$. Here, the lower m indices will act as the control qubits.

Shor's Algorithm (Part 2)

Special Case

- We first solve with the assumption that $\theta = y/2^m$ for some integer $0 \le y < 2^m$.
- The idea is to construct a state which has exactly one non-zero coordinate in the root-of-unity basis for each value of y.

1. Initial state is $|\psi\rangle \otimes |\overline{0}^m\rangle$. Here, the lower m indices will act as the control qubits.
2. We first uniformize the coefficients of the control qubits:

$$\left(I_n \otimes H^{\otimes m}\right)\left(|\psi\rangle \otimes |\overline{0}^m\rangle\right) = \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} |\psi\rangle \otimes |\text{binary}(x)\rangle$$

Shor's Algorithm (Part 2)

Special Case

- We first solve with the assumption that $\theta = y/2^m$ for some integer $0 \leq y < 2^m$.
- The idea is to construct a state which has exactly one non-zero coordinate in the root-of-unity basis for each value of y.

1. Initial state is $|\psi\rangle \otimes |\bar{0}^m\rangle$. Here, the lower m indices will act as the control qubits.
2. We first uniformize the coefficients of the control qubits:

$$\left(I_n \otimes H^{\otimes m}\right)\left(|\psi\rangle \otimes |\bar{0}^m\rangle\right) = \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} |\psi\rangle \otimes |\text{binary}(x)\rangle$$

3. By applying U^k on $|\psi\rangle$ with control qubit k for each $0 \leq k < m$, we obtain the following state

$$\frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} U^x |\psi\rangle \otimes |\text{binary}(x)\rangle = |\psi\rangle \otimes \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} e^{2\pi\theta x \mathrm{i}} |\text{binary}(x)\rangle$$

Special Case

4. Now substitute $\theta = y/2^m$.

$$|\psi\rangle \otimes \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} e^{2\pi\theta x \mathrm{i}} \, |\mathrm{binary}(x)\rangle = |\psi\rangle \otimes \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} \omega_{2^m}^{yx} \, |\mathrm{binary}(x)\rangle$$
$$= |\psi\rangle \otimes |b_y\rangle$$

Shor's Algorithm (Part 2)

Special Case

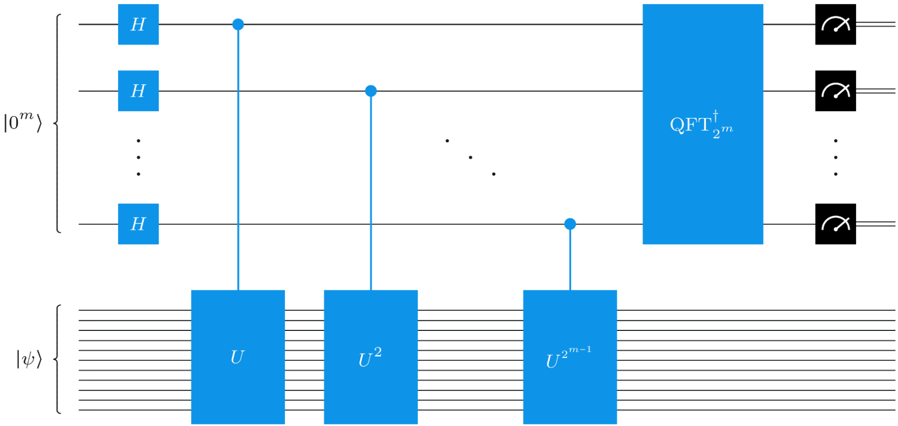
4. Now substitute $\theta = y/2^m$.

$$|\psi\rangle \otimes \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} e^{2\pi\theta x \mathrm{i}} \left|\mathrm{binary}(x)\right\rangle = |\psi\rangle \otimes \frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} \omega_{2^m}^{yx} \left|\mathrm{binary}(x)\right\rangle$$
$$= |\psi\rangle \otimes |b_y\rangle$$

5. We have obtained the desired state. For $s \in \{0, 1\}^m$, the probability that the root-of-unity basis measurement yields the bitstring s is

$$\left| |b_s\rangle \langle b_s | b_y\rangle \right|^2 = \begin{cases} 1 & \text{if } s = y \\ 0 & \text{if } s \neq y \end{cases}$$

Special Case



Shor's Algorithm (Part 2)

General Case

- For arbitrary θ, we run the exact circuit used for the special case.

Shor's Algorithm (Part 2)

General Case

- For arbitrary θ, we run the exact circuit used for the special case.
- As θ changes, the probability will change continuously, and we expect to observe integer y where $y/2^m$ is closest to θ.

Shor's Algorithm (Part 2)

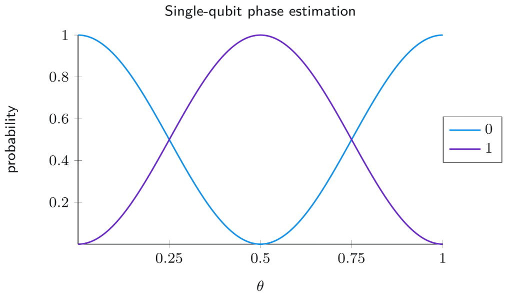
General Case

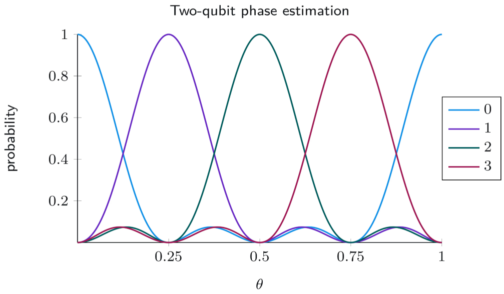
- For arbitrary θ, we run the exact circuit used for the special case.
- As θ changes, the probability will change continuously, and we expect to observe integer y where $y/2^m$ is closest to θ.
- We repeat the experiment few times, and report $y/2^m$ as the estimation where y is the most frequently observed value.

Shor's Algorithm (Part 2)

General Case

- For arbitrary θ, we run the exact circuit used for the special case.
- As θ changes, the probability will change continuously, and we expect to observe integer y where $y/2^m$ is closest to θ.
- We repeat the experiment few times, and report $y/2^m$ as the estimation where y is the most frequently observed value.





Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- We're given coprime integers a and n, and our goal is to find the order r of a modulo n.

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- We're given coprime integers a and n, and our goal is to find the order r of a modulo n.
- In the circuit used in the phase estimation problem, we set the gate U as the following for all integer $0 \leq x < 2^N$ where $N = \lceil \log_2(n) \rceil$.

$$U|x\rangle = \begin{cases} |a \cdot x \mod n\rangle & \text{if } x < n \\ |x\rangle & \text{if } x \geq n \end{cases}$$

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- We're given coprime integers a and n, and our goal is to find the order r of a modulo n.
- In the circuit used in the phase estimation problem, we set the gate U as the following for all integer $0 \leq x < 2^N$ where $N = \lceil \log_2 (n) \rceil$.

$$U |x\rangle = \begin{cases} |a \cdot x \mod n\rangle & \text{if } x < n \\ |x\rangle & \text{if } x \geq n \end{cases}$$

- Q) How many gates do we need to implement U?

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- We're given coprime integers a and n, and our goal is to find the order r of a modulo n.
- In the circuit used in the phase estimation problem, we set the gate U as the following for all integer $0 \leq x < 2^N$ where $N = \lceil \log_2(n) \rceil$.

$$U|x\rangle = \begin{cases} |a \cdot x \mod n\rangle & \text{if } x < n \\ |x\rangle & \text{if } x \geq n \end{cases}$$

- Q) How many gates do we need to implement U? $O(N \cdot \log(N))$

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- We're given coprime integers a and n, and our goal is to find the order r of a modulo n.
- In the circuit used in the phase estimation problem, we set the gate U as the following for all integer $0 \leq x < 2^N$ where $N = \lceil \log_2(n) \rceil$.

$$U|x\rangle = \begin{cases} |a \cdot x \mod n\rangle & \text{if } x < n \\ |x\rangle & \text{if } x \geq n \end{cases}$$

- Q) How many gates do we need to implement U? $O(N \cdot \log(N))$
- Q) For a non-negative integer k, how many gates do we need to implement U^k?

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- We're given coprime integers a and n, and our goal is to find the order r of a modulo n.
- In the circuit used in the phase estimation problem, we set the gate U as the following for all integer $0 \leq x < 2^N$ where $N = \lceil \log_2(n) \rceil$.

$$U|x\rangle = \begin{cases} |a \cdot x \mod n\rangle & \text{if } x < n \\ |x\rangle & \text{if } x \geq n \end{cases}$$

- Q) How many gates do we need to implement U? $O(N \cdot \log(N))$
- Q) For a non-negative integer k, how many gates do we need to implement U^k? $O(N \cdot \log(N))$

Solving order-finding problem with phase estimation problem

- Due to the structure of U, we get that the following vector is a unit eigenvector of U with eigenvalue $\omega_r = e^{2\pi \mathrm{i}/r}$.

$$|\psi_1\rangle = \frac{\omega_r^{-0}\left|a^0 \mod n\right\rangle + \omega_r^{-1}\left|a^1 \mod n\right\rangle + \cdots + \omega_r^{-(r-1)}\left|a^{r-1} \mod n\right\rangle}{\sqrt{r}}$$

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- Due to the structure of U, we get that the following vector is a unit eigenvector of U with eigenvalue $\omega_r = e^{2\pi \mathrm{i}/r}$.

$$|\psi_1\rangle = \frac{\omega_r^{-0}\,|a^0 \mod n\rangle + \omega_r^{-1}\,|a^1 \mod n\rangle + \cdots + \omega_r^{-(r-1)}\,|a^{r-1} \mod n\rangle}{\sqrt{r}}$$

- We run phase estimation on U and $|\psi_1\rangle$ with accuracy $m = 2N$, and we obtain a number k such that $\left|\frac{1}{r} - \frac{k}{2^{2N}}\right| \leq \frac{1}{2^{2N+1}}$, and $r = \lfloor \frac{2^{2N}}{k} + \frac{1}{2} \rfloor$.

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- Due to the structure of U, we get that the following vector is a unit eigenvector of U with eigenvalue $\omega_r = e^{2\pi \mathrm{i}/r}$.

$$|\psi_1\rangle = \frac{\omega_r^{-0}\,|a^0 \mod n\rangle + \omega_r^{-1}\,|a^1 \mod n\rangle + \cdots + \omega_r^{-(r-1)}\,|a^{r-1} \mod n\rangle}{\sqrt{r}}$$

- We run phase estimation on U and $|\psi_1\rangle$ with accuracy $m = 2N$, and we obtain a number k such that $\left|\frac{1}{r} - \frac{k}{2^{2N}}\right| \leq \frac{1}{2^{2N+1}}$, and $r = \lfloor \frac{2^{2N}}{k} + \frac{1}{2} \rfloor$.
- There is one issue: we don't know r so we can't construct $|\psi_1\rangle$.

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- Note that the following vector is a unit eigenvector of U with eigenvalue ω_r^k for all integer $0 \leq k < r$.

$$|\psi_k\rangle = \frac{\omega_r^{-0 \cdot k} |a^0 \mod n\rangle + \omega_r^{-1 \cdot k} |a^1 \mod n\rangle + \cdots \omega_r^{-(r-1) \cdot k} |a^{r-1} \mod n\rangle}{\sqrt{r}}$$

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- Note that the following vector is a unit eigenvector of U with eigenvalue ω_r^k for all integer $0 \leq k < r$.

$$|\psi_k\rangle = \frac{\omega_r^{-0\cdot k}\,|a^0 \mod n\rangle + \omega_r^{-1\cdot k}\,|a^1 \mod n\rangle + \cdots \omega_r^{-(r-1)\cdot k}\,|a^{r-1} \mod n\rangle}{\sqrt{r}}$$

- The following equality is immediate from the root-of-unity filter

$$|\psi_0\rangle + |\psi_1\rangle + \cdots + |\psi_{r-1}\rangle = |1\rangle$$

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- Note that the following vector is a unit eigenvector of U with eigenvalue ω_r^k for all integer $0 \leq k < r$.

$$|\psi_k\rangle = \frac{\omega_r^{-0\cdot k} |a^0 \mod n\rangle + \omega_r^{-1\cdot k} |a^1 \mod n\rangle + \cdots \omega_r^{-(r-1)\cdot k} |a^{r-1} \mod n\rangle}{\sqrt{r}}$$

- The following equality is immediate from the root-of-unity filter

$$|\psi_0\rangle + |\psi_1\rangle + \cdots + |\psi_{r-1}\rangle = |1\rangle$$

- We run phase estimation with $|1\rangle$ instead of $|\psi_1\rangle$, and we'll obtain an approximation $\frac{t}{2^{2n}}$ of $\frac{k}{r}$ where $0 \leq k < r$ is selected uniformly at random.

Shor's Algorithm (Part 2)

Solving order-finding problem with phase estimation problem

- Note that the following vector is a unit eigenvector of U with eigenvalue ω_r^k for all integer $0 \le k < r$.

$$|\psi_k\rangle = \frac{\omega_r^{-0\cdot k}|a^0 \mod n\rangle + \omega_r^{-1\cdot k}|a^1 \mod n\rangle + \cdots \omega_r^{-(r-1)\cdot k}|a^{r-1} \mod n\rangle}{\sqrt{r}}$$

- The following equality is immediate from the root-of-unity filter

$$|\psi_0\rangle + |\psi_1\rangle + \cdots + |\psi_{r-1}\rangle = |1\rangle$$

- We run phase estimation with $|1\rangle$ instead of $|\psi_1\rangle$, and we'll obtain an approximation $\frac{t}{2^{2n}}$ of $\frac{k}{r}$ where $0 \le k < r$ is selected uniformly at random.
- It turns out that we can uniquely recover the pair $k/\gcd(k,r), r/\gcd(k,r)$ with this given constraint in polynomial time, using continued fraction algorithm.

Solving order-finding problem with phase estimation problem

- We run this procedure few times, and take the least common multiple of all denominator as the order r.

List of experiments (from wikipedia)

- In 2001, a group at IBM demonstrated Shor's algorithm by factoring 15 using an NMR implementation of a quantum computer with seven qubits.
- In 2012, factorization of 21 was achieved.
- In 2019, an attempt was made to factor the number 35 using Shor's algorithm on an IBM Q System One, but the algorithm failed because of accumulating errors.

The End