Shor’s Algorithm

By Riya Raina

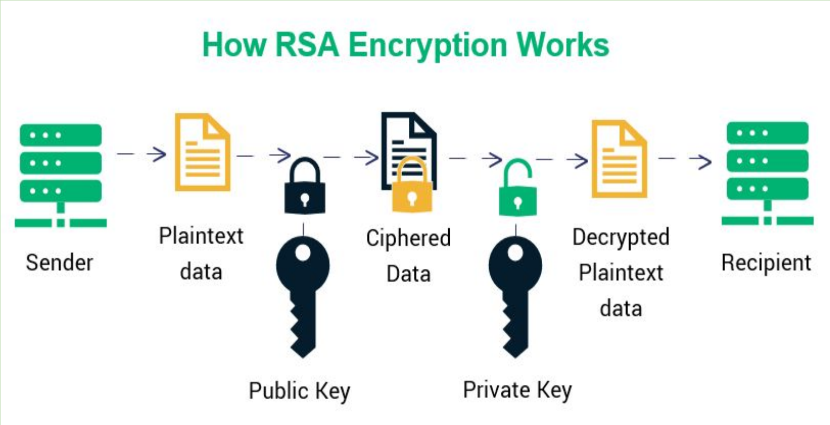
The Problem

Find the prime factorization of a large integer in an efficient manner

Why is it important to solve?

Some applications of prime factorization include:

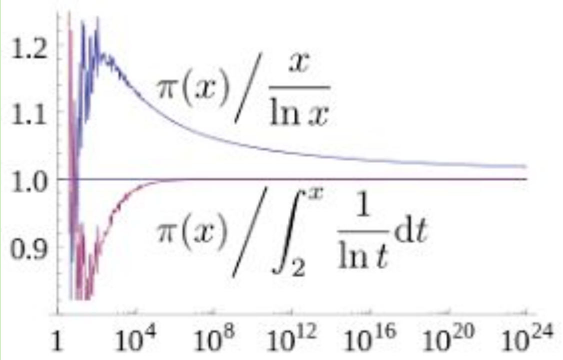
- Cryptography and Cybersecurity
  - RSA Algorithm used to encrypt messages
  - Public and private keys are large integers, multiplied to encrypt data
  - Relies on large numbers being hard to factor
  - Potential Cybersecurity Threat



Why is it important to solve?

Some applications of prime factorization include:

- Prime Number Theorem
  - Estimates frequency of prime numbers
- Discovery of New Largest Prime Numbers
  - Current Largest Known: 2^82,589,933-1



Classical Solution - Method I

- Loop through numbers 0 - √num,
- Add current number to factors array if perfectly divisible

```python
# Method 1: time complexity - O(n^(1/2)log(n))
import math


def prime_factors(num):
    factors = []
    while num % 2 == 0:
        factors.append(2)
        num = num / 2

    for i in range(3, int(math.sqrt(num)) + 1, 2):

        while num % i == 0:
            factors.append(i)
            num = num / i

    if num > 2:
        factors.append(int(num))

    print(factors)


# example to test
n = 9023724723
prime_factors(n)
```

```
[3, 13, 113, 983, 2083]
```

Classical Solution - Method II

- Variable i = 2
- Continuous loop while num > 1
- If num is divisible by i, add i to factors and divide num by i
- Else, increment i by 1

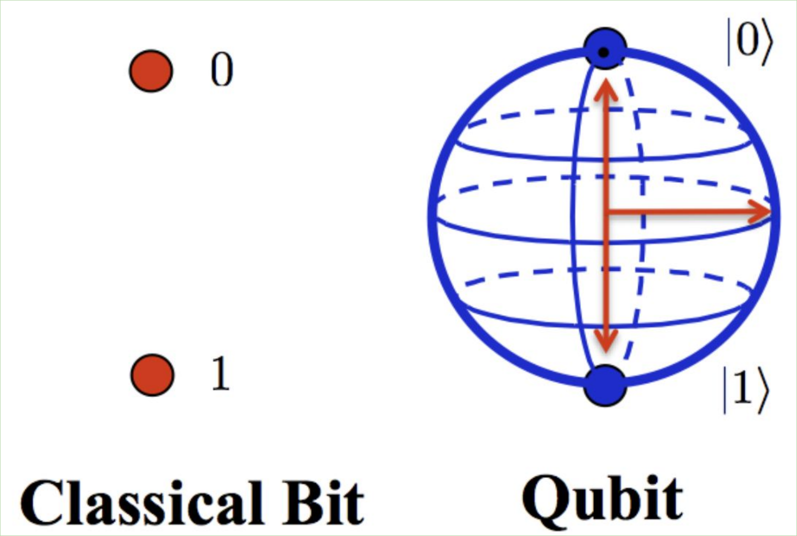
```python
# Method 2: time complexity - O(n)

def prime_factors(num):
    factors = []
    i = 2
    while num > 1:
        if num % i == 0:
            factors.append(i)
            num = num / i
        else:
            i = i + 1
    print(factors)


# example to test
n = 9023724723
prime_factors(n)
```

```
[3, 13, 113, 983, 2083]
```

Quantum Solution

Background Info about Quantum Computing

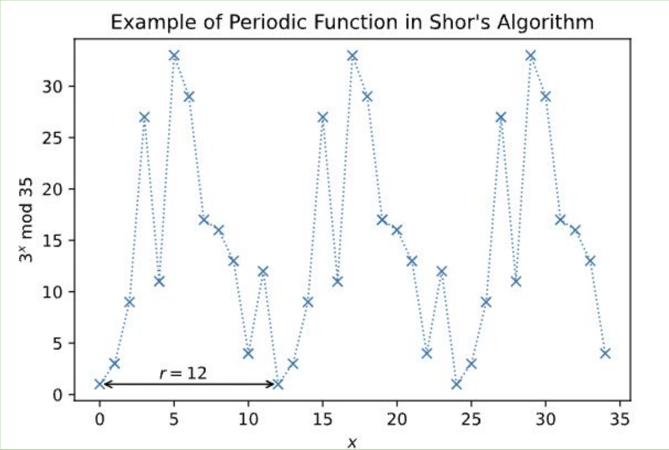
- Classical Bit - Unit of info with 2 distinct states, 1 and 0
- Quantum Qubit - Unit of info that can be in superposition of both 0 and 1 states simultaneously
- Qubits can be represented as 3D vector states that are rotated by gates and operations



Quantum Solution

- A Factoring problem can be turned into the problem of finding the period of a function
- The period (r) is the smallest non-zero integer that satisfies the following equation:

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



Quantum Solution

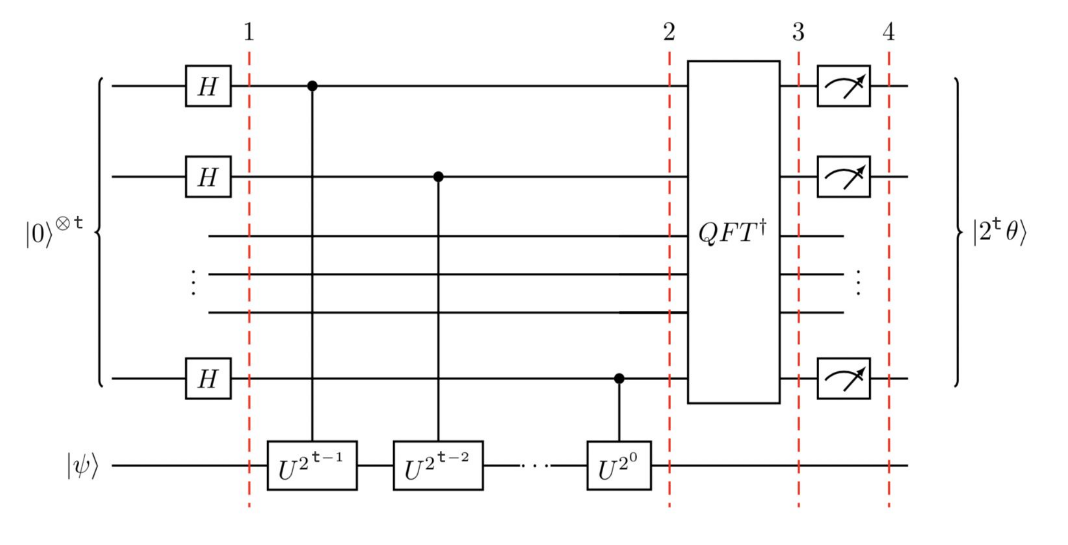
Example of Shor's Algorithm with N = 15

- Generate a random number less than N, such as a = 2
- Calculate gcd $gcd(a, N) = gcd(2, 15) = 1$
- Find period of $a^r \bmod N$ using quantum phase estimation

Quantum Solution

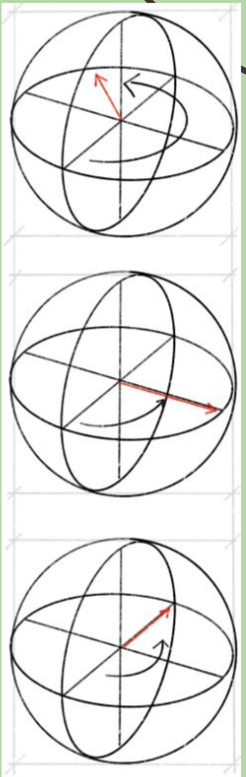
A look into Quantum Phase Estimation

- Top register: t counting qubits that control unitary operations
- Bottom register: qubits in state of $|\psi\rangle$ which operations are applied to



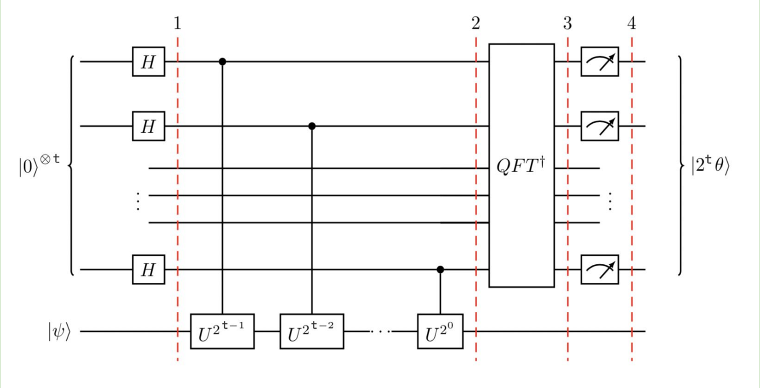
Quantum Solution

- Hadamard gates put the counting qubits into superposition
- Apply unitary operations on bottom qubits
  - Controlled phase rotations of a specific angle - the phase we wish to estimate
  - First qubit does 1 rotation
  - Second qubit does 2 rotations
  - Third qubit does 4 rotations and so on



Quantum Solution

- Apply an inverse Quantum Fourier Transformation (QFT) to the counting qubits and measure them
- The inverse QFT transforms the qubit state from the Fourier basis to the computational basis so it can be measured as a binary value
- Convert binary to decimal



Quantum Solution

- To estimate the phase we use the formula:
- Θ is the estimated phase
- M is the measured value
- n is the number of counting qubits

$$\theta_{estimated} = \frac{M}{2^n}$$

Now that we have calculated the phase, it can be used in the previous calculation

Quantum Solution

$$\theta_{estimated} = \frac{M}{2^n}$$

Back to example with N = 15 and a = 2

- Quantum Phase Estimation yields 10000000 in binary
- Convert to 128 in decimal
- 5 counting qubits used, so n = 5

Quantum Phase Estimation has calculated that the phase (r) = 4

Quantum Solution

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

$$(a^r - 1) \bmod N = 0$$

$$a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1)$$

- Rewrite original equation
- Substitute in values of a and r

$$gcd(a^{\frac{r}{2}} + 1, N) = gcd(5, 15) = 5$$
$$gcd(a^{\frac{r}{2}} - 1, N) = gcd(3, 15) = 5$$

```
guesses = [gcd(a**(r//2) - 1, N), gcd(a**(r//2) + 1, N)]
```

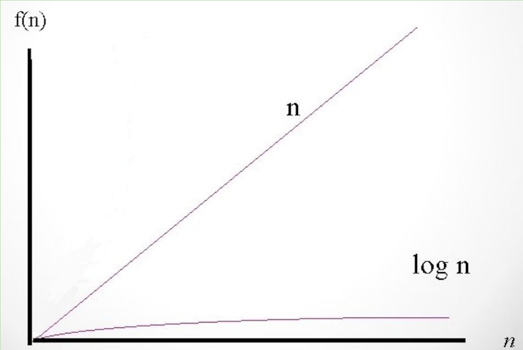
- For N = 15, the two decomposed prime numbers were 3 and 5

```
Guessed Factors: 3 and 5
*** Non-trivial factor found: 3 ***
*** Non-trivial factor found: 5 ***
Done!
```

Conclusion

Classical Solution

Time complexity:
O(n)



Quantum Solution

Time complexity:
O(logN)

When the classical solution takes 1 billion seconds, the quantum solution takes 9 seconds!

Sources

Shor’s Algorithm Diagrams:
https://qiskit.org/textbook/ch-algorithms/shor.html

Time Complexity Diagram:
https://slidetodoc.com/algorithm-analysis-with-big-oh-data-structures-and/

Quantum Phase Estimation Diagrams:
https://qiskit.org/textbook/ch-algorithms/quantum-phase-estimation.html

RSA Algorithm Diagram:
https://sectigostore.com/blog/ecdsa-vs-rsa-everything-you-need-to-know/

Prime Number Theory Diagram:
https://en.wikipedia.org/wiki/Prime_number_theorem

Questions?