Introduction to Quantum Computing: From Fundamentals to Quantum Algorithms

Ahyan Hassan
Mentor: Divyaansh Kumar

Summer, 2025

Abstract

This report provides a comprehensive introduction to quantum computing, covering fundamental concepts like qubits, superposition, entanglement, and quantum gates. It explores key quantum algorithms including Deutsch-Jozsa, Bernstein-Vazirani, Simon's, Shor's, and Grover's algorithms, with detailed mathematical analysis and quantum circuit implementations. The report also discusses practical implementations on quantum hardware and future directions for the field.

Contents

Chapter 1

Introduction to Quantum Computing

1.1 The Quantum Revolution

Quantum computing represents a paradigm shift from classical computing by leveraging quantum mechanical phenomena. While classical computers use bits (0/1), quantum computers use **qubits** that can exist in superpositions of states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{where} \quad |\alpha|^2 + |\beta|^2 = 1$$

This enables parallel computation on exponentially many states simultaneously.

1.2 Key Quantum Phenomena

1.2.1 Superposition

Qubits can exist in linear combinations of basis states. A qubit's state is represented as a unit vector in Hilbert space:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$$

which can visualized on the Bloch sphere.

1.2.2 Entanglement

When qubits become correlated such that their state cannot be described independently:

$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

Entanglement enables quantum parallelism and is essential for quantum speedups.

1.2.3 Bloch Sphere Representation

The Bloch sphere is a geometrical representation of a single qubit's pure state. Any pure state of a qubit can be visualized as a point on the surface of a unit sphere.

A general qubit state is:

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi)$ are spherical coordinates.

- θ controls how much the qubit is in state $|0\rangle$ vs $|1\rangle$. - ϕ determines the phase between the components.

This representation helps understand quantum gates like Hadamard (H), Pauli-X/Y/Z, which act as rotations on the sphere.

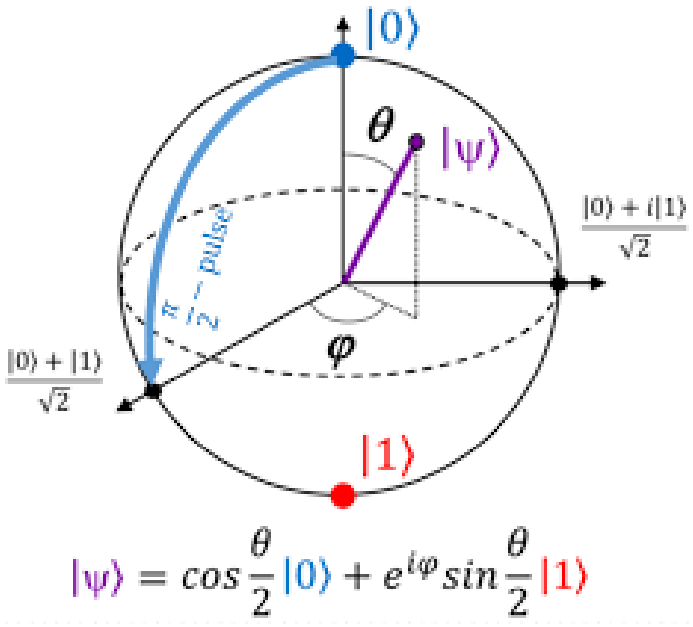


Figure 1.1: Bloch Sphere representation of a single qubit

1.3 Quantum vs. Classical Computing

Feature	**Classical**	**Quantum**
Basic unit	Bit (0/1)	Qubit ($\alpha\|0\rangle + \beta\|1\rangle$)
Operations	Boolean logic	Unitary transformations
Parallelism	Limited	Exponential (superposition)
Information	Copyable	No-cloning theorem

Table 1.1: Comparison of computing paradigms

Chapter 2

Building Blocks of Quantum Computation

2.1 Single-Qubit Gates

2.1.1 Hadamard Gate

Creates superposition: $H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $H|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

For any general n state qubit:

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}(-1)^{x\cdot y}|y\rangle.$$

2.1.2 Pauli Gates

$$X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \quad \text{(quantum NOT)}$$

$$Y = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix}$$

$$Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$

2.2 Multi-Qubit Gates

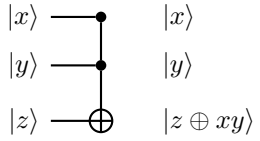
2.2.1 CNOT Gate

Entangles qubits: $CNOT|10\rangle = |11\rangle$

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

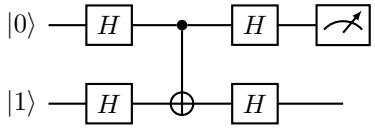
2.2.2 Toffoli Gate (CCNOT)

Three-qubit gate: $CCNOT|110\rangle = |111\rangle$



2.3 Quantum Circuit Model

Quantum algorithms are implemented as sequences of quantum gates:



Circuits must satisfy unitarity: $U^\dagger U = I$, preserving the norm.

Chapter 3

Quantum Algorithms

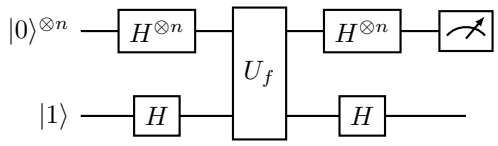
Quantum algorithms harness quantum phenomena such as superposition and entanglement to solve problems faster than classical algorithms. In this chapter, we explore foundational quantum algorithms that provide speedups in specific computational tasks like searching, factorization, and function identification.

3.1 Deutsch-Jozsa Algorithm

3.1.1 Problem Formulation

Determine if $f : \{0,1\}^n \to \{0,1\}$ is **constant** (all outputs equal) or **balanced** (50% 0s and 1s) using minimal queries.

3.1.2 Quantum Circuit



3.1.3 Mathematical Analysis

Initial State:

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

After Hadamard Gates:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \otimes \underbrace{\frac{|0\rangle - |1\rangle}{\sqrt{2}}}_{|-\rangle}$$

After Oracle U_f:

$$U_f|x\rangle|j\rangle = |x\rangle|j \oplus f(x)\rangle$$

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x} (-1)^{f(x)}|x\rangle \otimes |-\rangle$$

After Final Hadamards:

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{y=0}^{2^n-1} \left(\sum_{x=0}^{2^n-1} (-1)^{x\cdot y + f(x)} \right) |y\rangle \otimes |1\rangle$$

Measurement Probability:

$$P(y=0) = \left| \frac{1}{2^n} \sum_{x} (-1)^{f(x)} \right|^2 = \begin{cases} 1 & \text{if } f \text{ constant} \\ 0 & \text{if } f \text{ balanced} \end{cases}$$

3.1.4 Qiskit Implementation of Deutsch-Jozsa Algorithm

Below is a sample Qiskit implementation of the Deutsch-Jozsa algorithm for a balanced function with 3 input qubits.

Listing 3.1: Deutsch-Jozsa algorithm in Qiskit (balanced function)

```
from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

n=3  # number of input qubits
qc=QuantumCircuit(n+1,n)

# Initialize ancilla in |1>
qc.x(n)

# Apply Hadamard to all qubits
for i in range(n+1):
    qc.h(i)

# Oracle for a balanced function f(x) = x0 XOR x1 (just an example)
# Apply X gate to control qubits where needed (depends on function)
qc.cx(0,n)
qc.cx(1,n)

# Apply Hadamard to input qubits again
for i in range(n):
    qc.h(i)

# Measure input qubits
for i in range(n):
    qc.measure(i,i)

# Execute
backend=Aer.get_backend('qasm_simulator')
result=backend.run(qc, shots=1024).result()
counts=result.get_counts()

print(counts)
plot_histogram(counts)
plt.show()
```

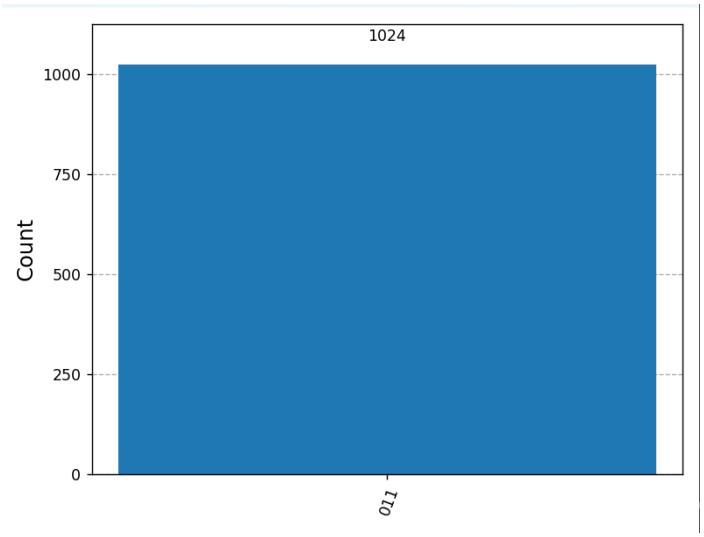


Figure 3.1: Graph of code. It gives 011 as output, which means function is balanced

3.1.5 Entanglement Analysis

States remain separable throughout:

$$|\psi_1\rangle = \left(\frac{1}{\sqrt{2^n}}\sum_x |x\rangle\right) \otimes |-\rangle$$

$$|\psi_2\rangle = \left(\frac{1}{\sqrt{2^n}}\sum_x (-1)^{f(x)}|x\rangle\right) \otimes |-\rangle$$

No entanglement is created or destroyed by U_f for this oracle.

3.1.6 Efficiency

Algorithm	**Classical Queries**	**Quantum Queries**
Deutsch-Jozsa	$\Omega(2^{n-1}) + 1$	1

Table 3.1: Query complexity comparison (n-bit functions)

3.2 Bernstein-Vazirani Algorithm

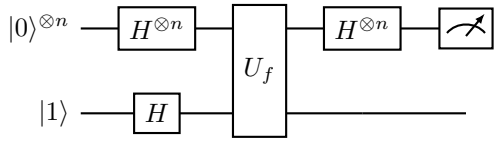
3.2.1 Problem Formulation

Given a hidden string $s \in \{0,1\}^n$, define a function $f(x) = s \cdot x \mod 2$, where $\cdot$ is the bitwise dot product (XOR-sum of matching bits). The goal is to determine s using as few queries as possible.

3.2.2 Classical vs Quantum Complexity

- **Classical:** Requires n queries — input e_i to learn each s_i.
- **Quantum:** Only **1 query** using quantum parallelism.

3.2.3 Quantum Circuit



3.2.4 Mathematical Analysis

Initial State:

$$|0\rangle^{\otimes n}|1\rangle$$

After First Hadamards:

$$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle \otimes |-\rangle$$

After Oracle U_f: (flips phase using $f(x) = s \cdot x$)

$$\frac{1}{\sqrt{2^n}} \sum_x (-1)^{s \cdot x} |x\rangle \otimes |-\rangle$$

After Final Hadamards:
After encoding the hidden string s into the phase of each computational basis state as:

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{s \cdot x} |x\rangle \otimes |-\rangle,$$

we now apply Hadamard gates again to the first n qubits.
Recall the identity for Hadamard transform on n qubits:

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} (-1)^{x \cdot y} |y\rangle.$$

Applying this to our state:

$$H^{\otimes n} \left(\sum_x (-1)^{s \cdot x} |x\rangle \right) = \sum_y \left[\frac{1}{2^n} \sum_x (-1)^{s \cdot x + x \cdot y} \right] |y\rangle.$$

This inner sum is a **Fourier-like sum** that behaves like a Kronecker delta:

$$\sum_x (-1)^{x \cdot (s \oplus y)} = \begin{cases} 2^n & \text{if } y = s, \\ 0 & \text{otherwise.} \end{cases}$$

So, the final state becomes:

$$|s\rangle \otimes |-\rangle.$$

Hence, upon measurement, we obtain the hidden string s with 100% probability.

3.2.5 Qiskit Implementation of Bernstein-Vazirani Algorithm

Below is a Qiskit implementation of the Bernstein-Vazirani algorithm for a 3-bit hidden string $s = 101$.

Listing 3.2: Bernstein-Vazirani algorithm in Qiskit (s = '101')

```
from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

# Secret string
s='101'
n=len(s)

# Create circuit with n input + 1 ancilla qubit
qc=QuantumCircuit(n+1,n)

# Put ancilla in |1>
qc.x(n)

# Hadamard to all qubits
for i in range(n+1):
    qc.h(i)

# Oracle: apply CX from each s_i=1
for i, bit in enumerate(s):
    if bit=='1':
        qc.cx(i, n)

# Final Hadamard to input qubits
for i in range(n):
    qc.h(i)

# Measure input qubits
for i in range(n):
    qc.measure(i,i)

# Simulate
backend=Aer.get_backend('qasm_simulator')
result=backend.run(qc, shots=1024).result()
counts=result.get_counts()

print("Result:", counts)
plot_histogram(counts)
plt.show()
```

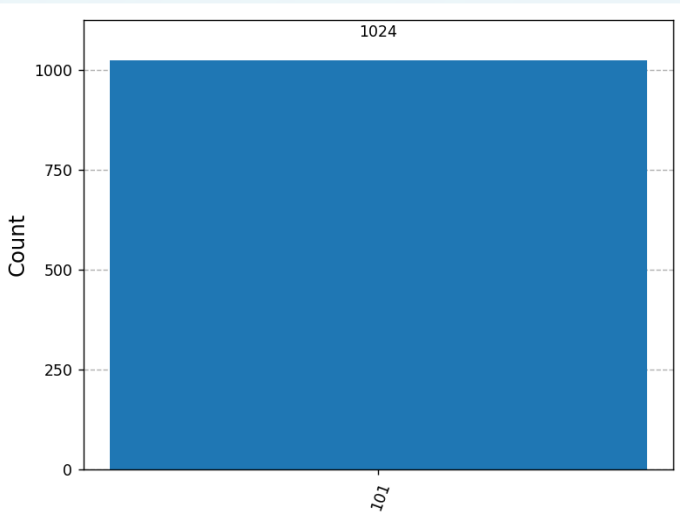


Figure 3.2: Graph of code. It implies the hidden string is s='101'

3.2.6 Efficiency

Algorithm	**Classical Queries**	**Quantum Queries**
Bernstein-Vazirani	n	1

Table 3.2: Query complexity for finding hidden string s

3.3 Simon's Algorithm

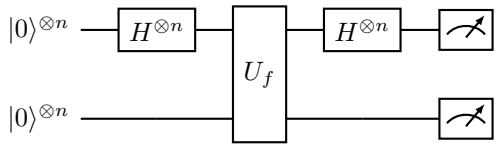
3.3.1 Problem Formulation

Find hidden string $s \neq 0$ where $f : \{0,1\}^n \to \{0,1\}^n$ satisfies:

$$f(x) = f(y) \iff y = x \oplus s$$

for all $x, y \in \{0,1\}^n$.

3.3.2 Quantum Circuit



3.3.3 Mathematical Analysis

Initial State:

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

After First Hadamards:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|0\rangle^{\otimes n}$$

After Oracle U_f:

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x} |x\rangle|f(x)\rangle$$

After Measuring Second Register (outcome z): Collapses to two preimages:

$$|\psi_3\rangle = \frac{1}{\sqrt{2}} \left(|x'\rangle + |x' \oplus s\rangle\right) \otimes |z\rangle$$

After Final Hadamards:

$$H^{\otimes n}|x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y} (-1)^{x \cdot y} |y\rangle$$

$$|\psi_4\rangle = \frac{1}{\sqrt{2^{n-1}}} \sum_{\substack{y \\ y \cdot s = 0}} (-1)^{x' \cdot y} |y\rangle$$

Measurement: Outputs random y satisfying $y \cdot s = 0$ with uniform probability.

3.3.4 Classical Post-Processing

After k iterations, solve linear system over $\mathbb{Z}_2$:

$$\begin{cases} y^{(1)} \cdot s = 0 \\ \vdots \\ y^{(k)} \cdot s = 0 \end{cases}$$

Success Probability for $k = n - 1$:

$$p(n-1) = \prod_{i=0}^{n-2} \left(1 - \frac{2^i}{2^n}\right) > \frac{1}{2} + \frac{1}{2^n}$$

3.3.5 Qiskit Implementation of Simon's Algorithm

Below is a Qiskit implementation of Simon's algorithm for a 3-bit secret string $s = 101$.

Listing 3.3: Simon's algorithm in Qiskit (s = '101')

```
from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt
from collections import Counter

s = '101'
n = len(s)

def create_oracle():
    oracle = QuantumCircuit(2*n, name="Oracle")

    # Hard-coded oracle for s='101'
    oracle.cx(0, 3)
    oracle.cx(2, 3)
    oracle.cx(1, 4)
    oracle.cx(0, 5)
    oracle.cx(2, 5)

    return oracle

def simons_algorithm(s, shots=1024):
    qc = QuantumCircuit(2*n, n)

    # Hadamard on input qubits
    for i in range(n):
        qc.h(i)

    # Apply Oracle
    oracle = create_oracle()
    qc.compose(oracle, inplace=True)

    # Hadamard again
    for i in range(n):
        qc.h(i)

    # Measure input qubits
    for i in range(n):
        qc.measure(i, i)

    return qc

qc = simons_algorithm(s)
qc.draw('mpl')

backend = Aer.get_backend('qasm_simulator')
result = backend.run(qc, shots=1024).result()
counts = result.get_counts()
filtered_counts = {k: v for k, v in counts.items() if k != '000'}

print("Measurement Results:", filtered_counts)
plot_histogram(filtered_counts)
plt.show()
```

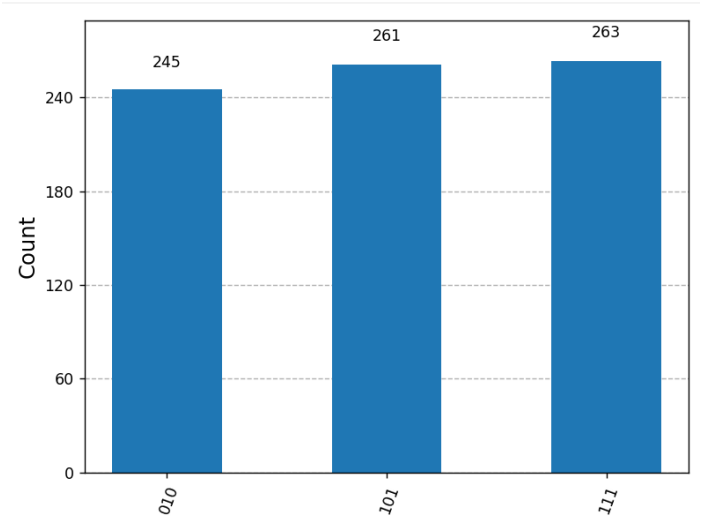


Figure 3.3: Graph of code.

Classical Post-Processing from Qiskit Output

From the histogram in Figure 1, the three most common outputs are:

$$y^{(1)} = 010, \quad y^{(2)} = 101, \quad y^{(3)} = 111$$

Each corresponds to an equation:

$$\begin{aligned} 010 &\Rightarrow s_1 = 0 \\ 101 &\Rightarrow s_0 \oplus s_2 = 0 \Rightarrow s_0 = s_2 \\ 111 &\Rightarrow s_0 \oplus s_1 \oplus s_2 = 0 \end{aligned}$$

Substituting:

$$\begin{aligned} s_1 &= 0 \\ s_0 &= s_2 \\ s_0 + 0 + s_2 &= 0 \Rightarrow s_0 = s_2 \end{aligned}$$

Thus, any string of form $s = s_0\ 0\ s_0$ satisfies all three. Choosing $s_0 = 1$, we get:

$$\boxed{s = 101}$$

This matches the hidden string used in the oracle.

3.3.6 Entanglement Analysis

The state after measurement:

$$|\psi_3\rangle = \frac{|x'\rangle + |x' \oplus s\rangle}{\sqrt{2}} \otimes |z\rangle$$

is maximally entangled when HammingWeight$(s) = n$:

$$\text{GHZ State: } \frac{|0\rangle^{\otimes n} + |1\rangle^{\otimes n}}{\sqrt{2}}$$

3.3.7 Efficiency

Algorithm	Classical Queries	Quantum Queries
Simon's	$\Omega(\sqrt{2^n})$	$\mathcal{O}(n)$

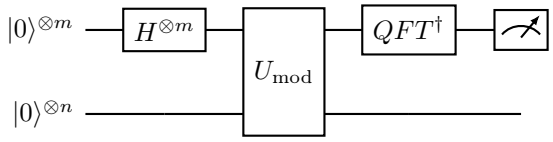
Table 3.3: Query complexity comparison (n-bit hidden string)

3.4 Shor's Factoring Algorithm

3.4.1 Key Insight

Factoring reduces to finding the period of $f(x) = a^x \mod N$.

3.4.2 Quantum Circuit



where $m = 2\lceil \log_2 N \rceil$, $n = \lceil \log_2 N \rceil$.

3.4.3 Quantum Fourier Transform (QFT)

The Quantum Fourier Transform (QFT) is the quantum analogue of the classical discrete Fourier transform (DFT). It operates on the amplitudes of quantum states and is essential for algorithms like Shor's, where it helps extract hidden periodicity in a superposition.

Given a quantum state over $N = 2^n$ basis states, the QFT acts on a basis state $|x\rangle$ as follows:

$$\text{QFT}|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i x y/N} |y\rangle$$

This transformation changes the basis of the quantum state to one where periodic structures become visible through measurement. QFT can be implemented efficiently using Hadamard gates and controlled phase rotations, with a complexity of $O(n^2)$ for n qubits.

In Shor's algorithm, QFT is applied to the control register after modular exponentiation and measurement of the output register. The result is a sharp probability distribution that reveals the period of the function.

3.4.4 Mathematical Analysis of Shor's Algorithm

Shor's algorithm reduces the factoring problem to the problem of finding the period of the function:

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

for a randomly chosen integer a such that $1 < a < N$ and $\gcd(a, N) = 1$.

Quantum Period Finding

The quantum subroutine is used to find the period r of $f(x)$. To do so, we prepare two registers:

- The first register (control) with $m = 2n$ qubits initialized to $|0\rangle^{\otimes m}$.
- The second register (target) with n qubits initialized to $|0\rangle^{\otimes n}$.

Apply Hadamard gates to the control register:

$$\frac{1}{\sqrt{2^m}} \sum_{x=0}^{2^m-1} |x\rangle \otimes |0\rangle$$

Then apply the modular exponentiation unitary U_f:

$$|x\rangle \otimes |0\rangle \xrightarrow{U_f} |x\rangle \otimes |f(x)\rangle = |x\rangle \otimes |a^x \mod N\rangle$$

Measure the second register. The output collapses to $|f(x_0)\rangle$ for some x_0, leaving the control register in a periodic superposition:

$$\frac{1}{\sqrt{K}} \sum_{k=0}^{K-1} |x_0 + kr\rangle$$

Applying the Quantum Fourier Transform

Now apply the QFT to the control register:

$$\text{QFT}\left(\frac{1}{\sqrt{K}}\sum_{k=0}^{K-1}|x_0+kr\rangle\right)$$

This results in a superposition with constructive interference at integer multiples of $\frac{2^m}{r}$:

$$\sum_{j=0}^{2^m-1} c_j|j\rangle \quad \text{with peak amplitudes at } j \approx \frac{k\cdot 2^m}{r}$$

Measure the register and obtain a value j. Then estimate:

$$\frac{j}{2^m} \approx \frac{k}{r}$$

Using continued fractions, extract the denominator r (the period).

3.4.5 Classical Post-Processing

Once r is found:

- Check if r is even.
- Compute $\gcd(a^{r/2}\pm 1, N)$.

If these values are non-trivial divisors of N, the algorithm succeeds. If not, choose another a and repeat.

Success probability: The success probability is high for random choices of a, and Shor's algorithm runs in polynomial time.

Note: If r is odd, then $r/2$ is not defined for integer modular arithmetic. In such cases, the algorithm cannot proceed with factor extraction, and a new random a must be selected.

Worked Example: Factoring $N = 15$

To illustrate Shor's algorithm, consider factoring $N = 15$ using $a = 2$.

We compute:

$$f(x) = 2^x \mod 15$$

for increasing x:

x	$2^x \mod 15$
0	1
1	2
2	4
3	8
4	1 $\Rightarrow$ Period $r = 4$

Since $r = 4$ is even, we compute:

$$2^{r/2} = 2^2 = 4$$

Then:

$$\gcd(4-1, 15) = \gcd(3, 15) = 3$$

$$gcd(4+1, 15) = \gcd(5, 15) = 5$$

Thus, we successfully factor 15:

$$\boxed{15 = 3\times 5}$$

This illustrates how Shor's quantum period-finding combined with classical GCD computation yields the non-trivial factors of a composite number.

3.4.6 Qiskit Implementation of Shor's Algorithm

Below is a basic Qiskit implementation of Shor's algorithm to factor $N = 15$. This uses Qiskit's built-in high-level 'Shor' class and simulates the quantum circuit using the 'Aer' backend.

Listing 3.4: Qiskit simulation of Shor's algorithm for N = 15

```
from qiskit.algorithms import Shor
from qiskit_aer import Aer
from qiskit.utils import QuantumInstance

# Set up simulator backend
backend=Aer.get_backend('aer_simulator')
qi=QuantumInstance(backend, shots=1024)

# Run Shors algorithm
shor=Shor(quantum_instance=qi)
result=shor.factor(N=15)

# Display result
print("Factors of 15 are:", result.factors)
```

Output of code will be: **Factors of 15 are: [[3, 5]]**

This code demonstrates the application of Shor's algorithm in a simulated quantum environment. It highlights how classical post-processing combines with quantum subroutines to recover the non-trivial factors of a composite number.

3.4.7 Entanglement Analysis

Entanglement plays a critical role in Shor's algorithm, especially during the modular exponentiation step. When the oracle U_f maps $|x\rangle|0\rangle \to |x\rangle|f(x)\rangle$, the control and target registers become entangled. This entanglement encodes the hidden periodicity of $f(x)$ across the amplitudes of the control register. After measuring the target register, the control register collapses into a periodic superposition:

$$\frac{1}{\sqrt{K}} \sum_{k=0}^{K-1} |x_0 + kr\rangle$$

This superposition remains entangled until the QFT is applied, which transforms it into a state with high probability peaks revealing multiples of $\frac{1}{r}$. The presence of entanglement enables the quantum interference required for this step to succeed.

3.4.8 Efficiency

Shor's algorithm runs in polynomial time on a quantum computer, contrasting with the best known classical factoring algorithms which run in sub-exponential time. The time complexity is:

$$\mathcal{O}((\log N)^3)$$

This arises because:

- Modular exponentiation is computed in $\mathcal{O}((\log N)^2)$ time.
- Quantum Fourier Transform is performed in $\mathcal{O}((\log N)^2)$ gates.
- The overall circuit is repeated polynomially many times to ensure success.

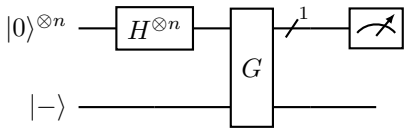
Thus, Shor's algorithm provides an exponential speedup over classical factoring methods such as the general number field sieve.

3.5 Grover's Search Algorithm

3.5.1 Problem Formulation

Given a function $f : \{0,1\}^n \to \{0,1\}$ such that $f(x^*) = 1$ for a unique unknown x^* and $f(x) = 0$ otherwise, the goal is to find x^* with high probability using fewer queries than a classical brute-force approach.

3.5.2 Circuit



where G is the Grover iterate.

3.5.3 Mathematical Analysis

Let $|\psi_0\rangle$ be the initial state:

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

Assume only one marked solution x^*.

We define:

$$|\alpha\rangle = \frac{1}{\sqrt{N-1}} \sum_{x \neq x^*} |x\rangle, \quad |\beta\rangle = |x^*\rangle$$

So the state can be written in 2D space:

$$|\psi_0\rangle = \sqrt{\frac{N-1}{N}}|\alpha\rangle + \frac{1}{\sqrt{N}}|\beta\rangle$$

Let θ be the angle between $|\psi_0\rangle$ and $|\alpha\rangle$:

$$\sin\theta = \frac{1}{\sqrt{N}}, \quad \theta \approx \frac{1}{\sqrt{N}} \text{ for large } N$$

Each Grover iteration (oracle + diffusion) is a rotation by angle 2θ toward $|\beta\rangle$.

After r iterations:

$$|\psi_r\rangle = \sin((2r+1)\theta)|\beta\rangle + \cos((2r+1)\theta)|\alpha\rangle$$

To maximize probability of measuring x^*, set:

$$(2r+1)\theta \approx \frac{\pi}{2} \Rightarrow r \approx \left\lfloor \frac{\pi}{4\theta} \right\rfloor \approx \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$$

Diffusion Operator

The diffusion operator reflects amplitudes about the average. If current state is:

$$|\psi\rangle = \sum_x \alpha_x |x\rangle$$

and $\mu = \frac{1}{N}\sum_x \alpha_x$, then diffusion transforms:

$$\alpha_x \to 2\mu - \alpha_x$$

This increases amplitude of marked state after oracle flips its sign.

Circuit implementation:

- Apply $H^{\otimes n}$, then $X^{\otimes n}$
- Apply multi-controlled-Z gate on $|0\rangle^{\otimes n}$
- Apply $X^{\otimes n}$, then $H^{\otimes n}$ again

This is equivalent to:

$$D = 2|\psi_0\rangle\langle\psi_0| - I$$

a reflection about the initial uniform superposition.

Success Probability

After r Grover iterations:

$$P_{\text{success}} = \sin^2((2r+1)\theta)$$

This becomes very close to 1 after $\approx \frac{\pi}{4}\sqrt{N}$ steps.

3.5.4 Qiskit Implementation of Grover's Algorithm ($N = 8$, target $= |101\rangle$)

Below is a Qiskit implementation of Grover's algorithm for $n = 3$ qubits, where the target marked state is $|101\rangle$.

Listing 3.5: Grover's algorithm in Qiskit for $|101\rangle$

```
from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

n=3  # number of input qubits
marked_state='101'  # target state

# Grover Circuit
qc=QuantumCircuit(n,n)

# Step 1: Superposition
qc.h(range(n))

# Step 2: Oracle for |101
for i, bit in enumerate(marked_state):
    if bit=='0':
        qc.x(i)
qc.h(n-1)
qc.mct(list(range(n-1)),n-1)  # multi-controlled-Z
qc.h(n-1)
for i, bit in enumerate(marked_state):
    if bit=='0':
        qc.x(i)

# Step 3: Diffusion
qc.h(range(n))
qc.x(range(n))
qc.h(n-1)
qc.mct(list(range(n-1)),n-1)
qc.h(n-1)
qc.x(range(n))
qc.h(range(n))

# Step 4: Measurement
qc.measure(range(n),range(n))

# Simulate
backend=Aer.get_backend('qasm_simulator')
result=backend.run(qc, shots=1024).result()
counts=result.get_counts()
print("Measurement Result:", counts)
plot_histogram(counts)
plt.show()
```

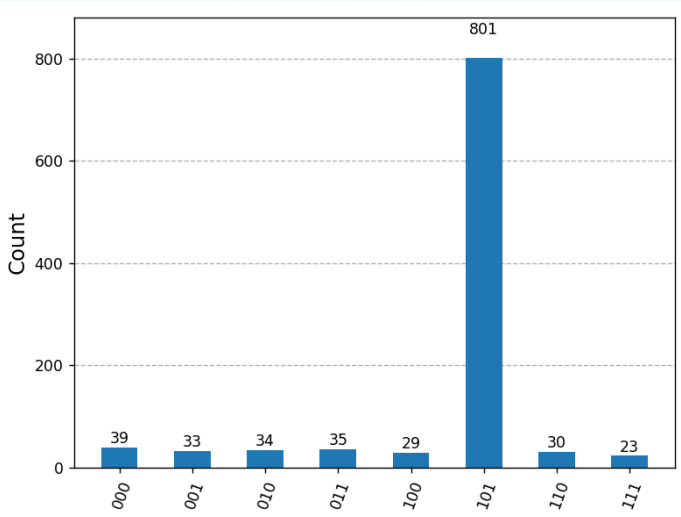


Figure 3.4: Graph of code. We found required no., ie. '101'

3.5.5 How it worked : Grover's Algorithm for $N = 8$

Consider a search problem with $N = 8$ possible items ($n = 3$ qubits). The marked item is $x^* = 5$, which corresponds to the basis state $|101\rangle$.

Step 1: Initial Superposition

All 8 basis states are initialized with equal amplitude:

$$|\psi_0\rangle = \frac{1}{\sqrt{8}} \sum_{x=0}^{7} |x\rangle$$

Step 2: Oracle Application

The oracle flips the phase of the marked state $|101\rangle$:

$$O_f|\psi_0\rangle = \frac{1}{\sqrt{8}} \left(\sum_{x \neq 5} |x\rangle - |101\rangle \right)$$

Step 3: Diffusion Operator

The diffusion operator reflects amplitudes about the average. Let μ be the average of all amplitudes before diffusion:

- Before Oracle:

$$\mu = \frac{1}{\sqrt{8}} = 0.3536 \quad \text{(all states)}$$

- After Oracle:

$$\alpha_{101} = -0.3536, \quad \text{others} \ = 0.3536$$

- After Diffusion:

$$\alpha_x \to 2\mu - \alpha_x$$

Applying this:

$$\alpha_{101} = 2(0.2895) - (-0.3536) \approx 0.9966$$

$$\alpha_{\text{others}} = 2(0.2895) - 0.3536 \approx 0.2254$$

Marked state's amplitude rises significantly.

Step 4: Second Iteration

Repeat oracle and diffusion:

- Flip phase of $|101\rangle$ - Recompute average amplitude and reflect all again

After 2 iterations, amplitude of $|101\rangle$ becomes approximately 0.980, while others are close to zero. Thus, measuring the system yields $x^* = 5$ with high probability.

Conclusion: Grover's algorithm increases the amplitude of the marked state quadratically faster than classical random sampling.

State	Initial Amplitude	After 1st Iteration	After 2nd Iteration
$\|000\rangle$	0.3536	0.2254	0.0281
$\|001\rangle$	0.3536	0.2254	0.0281
$\|010\rangle$	0.3536	0.2254	0.0281
$\|011\rangle$	0.3536	0.2254	0.0281
$\|100\rangle$	0.3536	0.2254	0.0281
$\|101\rangle$	0.3536	0.9966	0.9801
$\|110\rangle$	0.3536	0.2254	0.0281
$\|111\rangle$	0.3536	0.2254	0.0281

Table 3.4: Amplitude evolution for Grover's algorithm with $N = 8$, target state $|101\rangle$

3.5.6 Entanglement Analysis

Grover's algorithm relies on entanglement during intermediate steps, particularly due to the multi-qubit operations such as the oracle and the diffusion operator.

1. Oracle Stage: The oracle flips the phase of the marked state using a multi-controlled Z gate. This operation introduces entanglement among the qubits, especially when implemented with ancilla-based decompositions.

2. Diffusion Operator: The diffusion step (inversion about the mean) involves multiple Hadamard, Pauli-X, and multi-controlled gates. These also entangle the qubits, especially since the inversion acts globally on the entire quantum state.

Although the initial and final states may appear product-like (superpositions), the process of amplitude amplification passes through entangled states. Entanglement helps redistribute and concentrate amplitude toward the solution state through quantum interference.

Final Observation: - Grover's algorithm doesn't maintain maximal entanglement like GHZ states (as in Simon's). - However, entanglement is essential during the iterative steps to ensure coherent interference and probability amplification toward the correct result.

3.5.7 Efficiency

Algorithm	Classical Queries	Quantum Queries
Grover's	$O(N)$	$O(\sqrt{N})$

Table 3.5: Query complexity of Grover's search

3.6 Comparison of Quantum Algorithms

Algorithm	Problem Type	Classical Complexity	Quantum Complexity	Speedup
Deutsch-Jozsa	Function type (balanced/-constant)	$O(2^n)$	$O(1)$	Exponential
Bernstein-Vazirani	Hidden string detection	$O(n)$	$O(1)$	Linear
Simon's	Hidden XOR mask	$O(\sqrt{2^n})$	$O(n)$	Exponential
Shor's	Integer factorization	Sub-exponential	$O((\log N)^3)$	Exponential
Grover's	Unstructured search	$O(N)$	$O(\sqrt{N})$	Quadratic

Table 3.6: Comparison of quantum algorithms on complexity and speedup

Chapter 4

Implementation and Applications

4.1 Quantum Hardware

- **Superconducting qubits** (IBM, Google): Microwave circuits
- **Trapped ions** (IonQ): Atomic qubits manipulated by lasers
- **Topological qubits** (Microsoft): Protected quantum states

4.2 IBM Quantum Experience

Implementation of Deutsch-Jozsa algorithm:

Results show probability distribution confirming constant/balanced function.

4.3 Applications

- Cryptanalysis (RSA, ECC)
- Quantum simulation (materials, chemistry)
- Optimization problems
- Machine learning acceleration

Conclusion

Quantum computing represents a fundamental shift in how we process and understand information. By leveraging the principles of superposition, entanglement, and quantum interference, it offers computational advantages that challenge classical boundaries.

In this report, we explored:

- The foundational concepts of qubits, gates, circuits, and the Bloch sphere.
- Quantum algorithms like:
  - **Deutsch-Jozsa**: Deterministically identifies balanced vs. constant functions with a single query.
  - **Bernstein-Vazirani**: Recovers hidden bit strings exponentially faster than classical methods.
  - **Simon's Algorithm**: Demonstrates exponential speedup for hidden XOR structures.
  - **Shor's Algorithm**: Factors large integers in polynomial time, threatening modern cryptography.
  - **Grover's Algorithm**: Offers quadratic speedup for unstructured search problems.
- Detailed Qiskit implementations and quantum circuit designs.
- Analysis of entanglement, efficiency, and real-world relevance of each algorithm.

While today's quantum hardware is still in the Noisy Intermediate-Scale Quantum (NISQ) era, the potential of these algorithms is immense. As error correction improves and qubit counts scale, the theoretical speedups presented here will increasingly translate to practical advantage.

Quantum computing is no longer just theoretical physics — it's a rising technology poised to impact cryptography, optimization, machine learning, and beyond. With continued progress in both algorithms and hardware, the quantum future looks not just possible, but inevitable.

Bibliography

[1] Nielsen, M. A., & Chuang, I. L. (2010). *Quantum Computation and Quantum Information*. Cambridge University Press. (10th Anniversary Edition)

[2] Sakurai, J. J., & Napolitano, J. (2017). *Modern Quantum Mechanics*. Cambridge University Press. (2nd Edition)

[3] Portugal, R. (2023). *Basic Quantum Algorithms*. Springer Nature.

Thank You

For your time, interest, and attention.

It has been a pleasure sharing this journey through the fascinating world of quantum computing.