PES UNIVERSITY

Karnataka, Bangalore-560080

Course: Quantum Entanglement and Quantum Computing

Course code: UE22EC343AB3

Project on:

An overview of "Oracle Quantum Algorithms"

Project by: Purab P Bhat | PES1UG22EC220

Nithin M | PES1UG22EC184

Under guidance of :

Dr Kaustav Bhoumick

ECE Dept

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINNERING

1.Abstract:

This report is an overview and analysis of various Oracle based quantum algorithms. We will be looking into Deutsch, Deutsch-Jozsa, Bernstein-Vazirani, Simon's and Grover's quantum mechanical algorithms. This paper contains simulation and analysis of the various algorithms.

2.Introduction:

Quantum computing is a field that uses quantum mechanics and algorithms to solve complex problems fast and efficiently. They take advantages of various effects like superposition and entanglement. Oracle quantum techniques also known as "Black Box". It is a Boolean function or device that outputs 0 or 1 based on inputs.

3.Mathematics:

in reference to [1]

3.1: Quantum bit

A quantum bit, or qubit, is the basic unit of information in quantum computing. It's the quantum equivalent of the binary bit used in classical computers. It always exists in a superposition of two states (|0> and |1>). This infers that both, logic 0 and logic 1 can exist simultaneously, which enables parallel computation [4]. In quantum computing a classical bit is mapped to a qubit as represented as

$$0 \rightarrow |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } 1 \rightarrow |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (1)$$

Dirac notation-

- ket $|\Psi\rangle$ (represents column vector)
- bra $\langle\Psi|$(complext conj of column vector)

3.2: Quantum operators

$\langle\ |\ \rangle$-Dot product notation

$|\ \rangle\langle\ |$-Outer product notation

The operators are generally represented as matrices. A quantum operator represents physical observables like energy, position, momentum and kinetic energy or transformations on a qubit caused by quantum gates. To represent a physical observable, it is sufficient for the quantum operator to be Hermitian. A Hermitian operator, H, is an adjoint operator satisfying the relation

$$H\dagger = H, \text{ where } H\dagger = (H*) \quad (2)$$

T is the complex conjugate transpose operator.

3.3: Normal and unitary operators

Any operator that satisfies A'A=AA' is a normal operator. Spectral decomposition is one such important consequence, represented as

$$A = \sum i \lambda j \, |vj\rangle \langle vj| \qquad (3)$$

In addition, if A'A=AA'=I, it is said to be unitary. Quantum computing operators are unitary and hence conserve probabilities.

3.4 Quantum state evolution

Qubits can be manipulated using gates like Hadamard, C-NOT, Pauli's gate.

- Hadamard gate is defined as

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad (4)$$

- Controlled-NOT (CNOT) gate is defined as

$$CNOT \equiv CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad (5)$$

The state of an isolated physical system is described by a unit vector $|\Psi\rangle$. The time evolution of the state can be described by a unitary operator, U, i.e.,

$$|\Psi\rangle \, t_1 = U \, |\Psi\rangle \, t_0 \qquad (6)$$

with $t_0 < t_1$

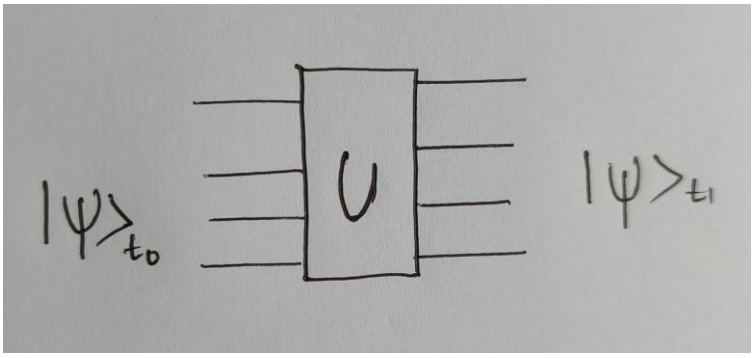

Fig 1: graphical representation of quantum circuit, system chances from t_0 to t_1

Quantum circuit shown in figure 2 illustrates the result for a balanced or unbalanced function which is explain in table 1.

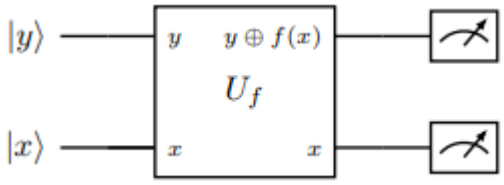

Fig 2: Quantum circuit for balanced/unbalanced func.

$\|x\rangle \|y\rangle$	$f(0) = 0 \wedge f(1) = 0$	$f(0) = 1 \wedge f(1) = 1$
$\|0\rangle \|0\rangle$	$\|0\rangle \|0 \oplus f(0)\rangle = \|0\rangle \|0\rangle$	$\|0\rangle \|0 \oplus f(0)\rangle = \|0\rangle \|1\rangle$
$\|0\rangle \|1\rangle$	$\|0\rangle \|1 \oplus f(0)\rangle = \|0\rangle \|1\rangle$	$\|0\rangle \|1 \oplus f(0)\rangle = \|0\rangle \|0\rangle$
$\|1\rangle \|0\rangle$	$\|1\rangle \|0 \oplus f(1)\rangle = \|1\rangle \|0\rangle$	$\|1\rangle \|0 \oplus f(1)\rangle = \|1\rangle \|1\rangle$
$\|1\rangle \|1\rangle$	$\|1\rangle \|1 \oplus f(1)\rangle = \|1\rangle \|1\rangle$	$\|1\rangle \|1 \oplus f(1)\rangle = \|1\rangle \|0\rangle$

Table 1: Results for unbalanced function f(x) [1]

$\|x\rangle \|y\rangle$	$f(0) = 0 \wedge f(1) = 1$	$f(0) = 1 \wedge f(1) = 0$
$\|0\rangle \|0\rangle$	$\|0\rangle \|0 \oplus f(0)\rangle = \|0\rangle \|1\rangle$	$\|0\rangle \|0 \oplus f(0)\rangle = \|0\rangle \|0\rangle$
$\|0\rangle \|1\rangle$	$\|0\rangle \|1 \oplus f(1)\rangle = \|0\rangle \|0\rangle$	$\|0\rangle \|1 \oplus f(0)\rangle = \|0\rangle \|0\rangle$
$\|1\rangle \|0\rangle$	$\|1\rangle \|0 \oplus f(1)\rangle = \|1\rangle \|0\rangle$	$\|1\rangle \|0 \oplus f(1)\rangle = \|1\rangle \|1\rangle$
$\|1\rangle \|1\rangle$	$\|1\rangle \|1 \oplus f(0)\rangle = \|1\rangle \|1\rangle$	$\|1\rangle \|1 \oplus f(1)\rangle = \|1\rangle \|1\rangle$

Table 2: Results for balanced function f(x) [1]

4.Algorithms:

4.1: Deutsch Algorithm

[5]This algorithm demonstrates whether 1-bit Boolean function f(x) where x∈ {0,1} is balanced or constant.

If output is the same for all inputs f (0) =f (1) then it is constant. However, if output is different for the inputs f (0) ≠f (1) then it is balanced.

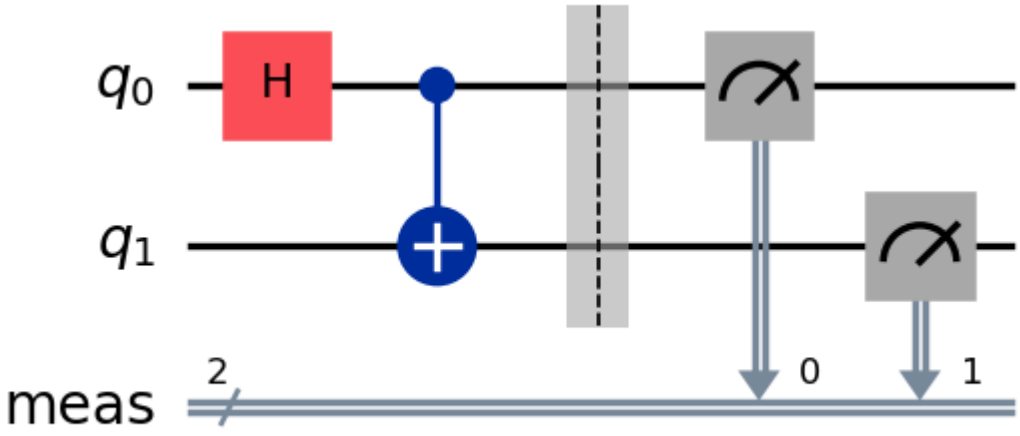

Fig 3: graphical representation of the quantum circuit for Deutsch algorithm.

In reference to [fig 3] To check this, 2 qubits are used, input qubit initialized to |0⟩ and target qubit initialized to |1⟩. Hadamard gate is applied to q_0(superposition). CNOT gate on q_0 (control bit) and q_1 (target bit). Both are measure for basis. Due

to entanglement the measurement outcomes of q_0 and q_1 are correlated. Hence $q_0 = q_1$.

4.2 Deutsch-Jozsa Algorithm

[1]Extension of Deutsch algorithm which is used to determine if the given function is constant(same output for all inputs) or balanced (0 for first half or 1 for second half).

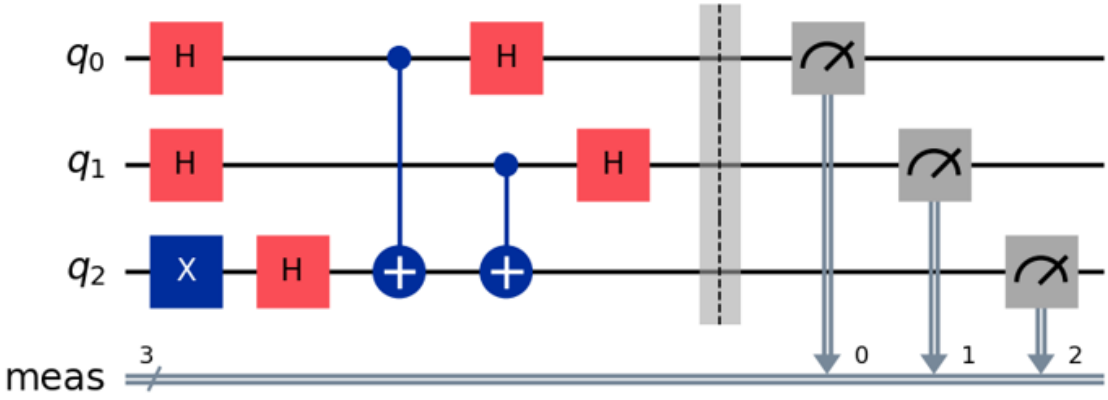

Fig 4: graphical representation of the quantum circuit for Deutsch-Jozsa algorithm

In reference to fig 4, the qubits q_0 and q_1 are initialised to |0> and q_2 is initialised to |1> using X gate.

$$|\Psi_0\rangle = |0\rangle\,|0\rangle\,|1\rangle$$

We take n+1 qubits with first n qubits initialized as input |0⟩ and last qubit initialized as target |1⟩. Apply Hadamard gate to all the qubits to create a superposition (first 2 in uniform superposition and then the auxiliary bit using function evaluation (3) and (4).

$$|\Psi\rangle_{t_1} = H^{\otimes_n}|0\rangle^{\otimes_n} = |+\rangle^{\otimes_n} = \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle)^{\otimes_n} = \frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}|x\rangle\ .$$

.Oracle flips phase of target qubit based on f(x) (CNOT flips state of q_2). Once more Hadamard gate is applied for interference.

If z=0 then f(x) is a constant whereas if z≠0 then f(x) is balanced.

4.3 Bernstein-Vazirani Algorithm

[1] extension of Deutsch-Jozsa algorithm and implemented to find a hidden binary string in the function f(x).

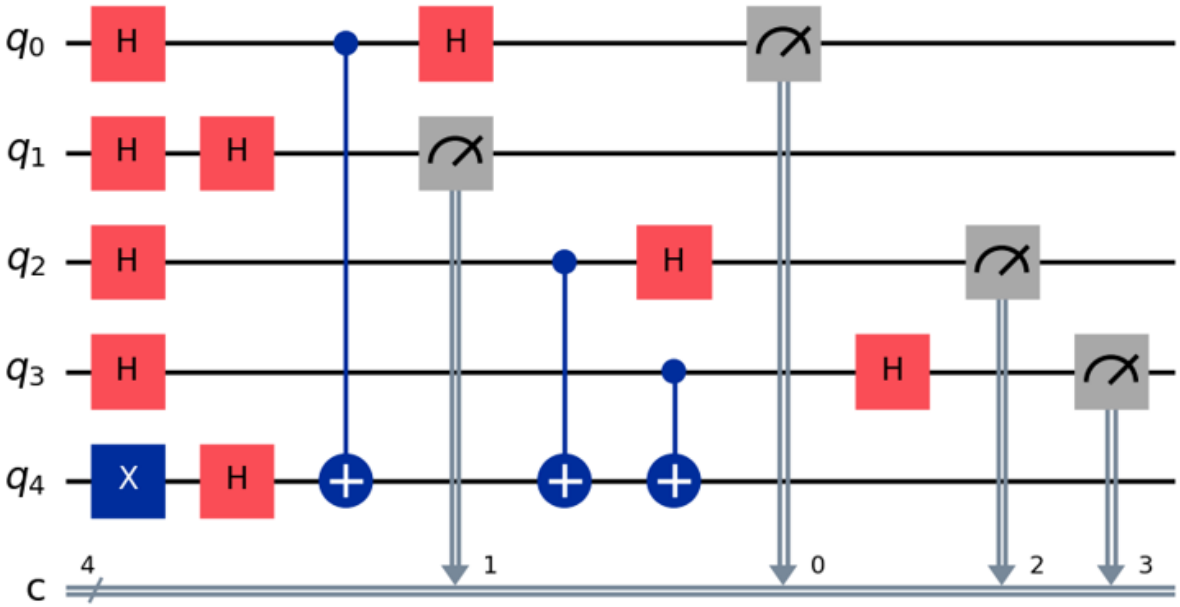

Fig 5: graphical representation of quantum of Berstein-Vazirani algorithm

The probability of finding s is

$$\left| \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^0 \right|^2 = \left| \frac{1}{2^n} 2^n \right|^2 = 1$$

5 qubits(n+1) are initialised to $|\Psi_0\rangle=|0\rangle$. Last qubit set to $|1\rangle$ using X gate. First Hadamard gate performs superposition on n+1 qubits (3),(4) and (6).final state looks like

$$|\Psi\rangle_{t_3} = \sum_{z \in \{0,1\}^n} \left(\frac{1}{2} \sum_{x \in \{0,1\}^n} (-1)^{f(x) \oplus (x \cdot z)} \right) |z\rangle$$

Last bit is set for phase kickback. Encoding is carried out on the oracle and second Hadamard gate is applied on n qubits. This corresponds to the output string. This algorithm gives an efficient query complexity between 01 and 02.

4.4 Simon's Algorithm.

Similar to Bernstein-Vazirani, the goal of the algorithm is to find s in a single half. So the same implementation in case of 4.2 .

Initialise n+1 qubits, set last to $|1\rangle$ and others to $|0\rangle$.

$$f : \{0,1\}^n \longrightarrow \{0,1\}^n$$

$$x = (x_{n-1} \ldots x_1 x_0) \longmapsto f(x) = f(x_n - 1) \ldots f(x_1) f(x_0)$$

After applying the Hadamard gate we get

$$: \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0 \ldots 00\rangle$$

The oracle produces

$$= \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |f(x)\rangle$$

The probability of measuring a |z⟩ such that s · z = 0 mod 2 is

$$\left| \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} \left((-1)^{x' \cdot z} + (-1)^{x'' \cdot z} \right) \right|^2 = \left| \frac{1}{\sqrt{2^{n+1}}} \pm 2 \right|^2 = \frac{4}{2^{n+1}} = \frac{1}{2^{n-1}}$$

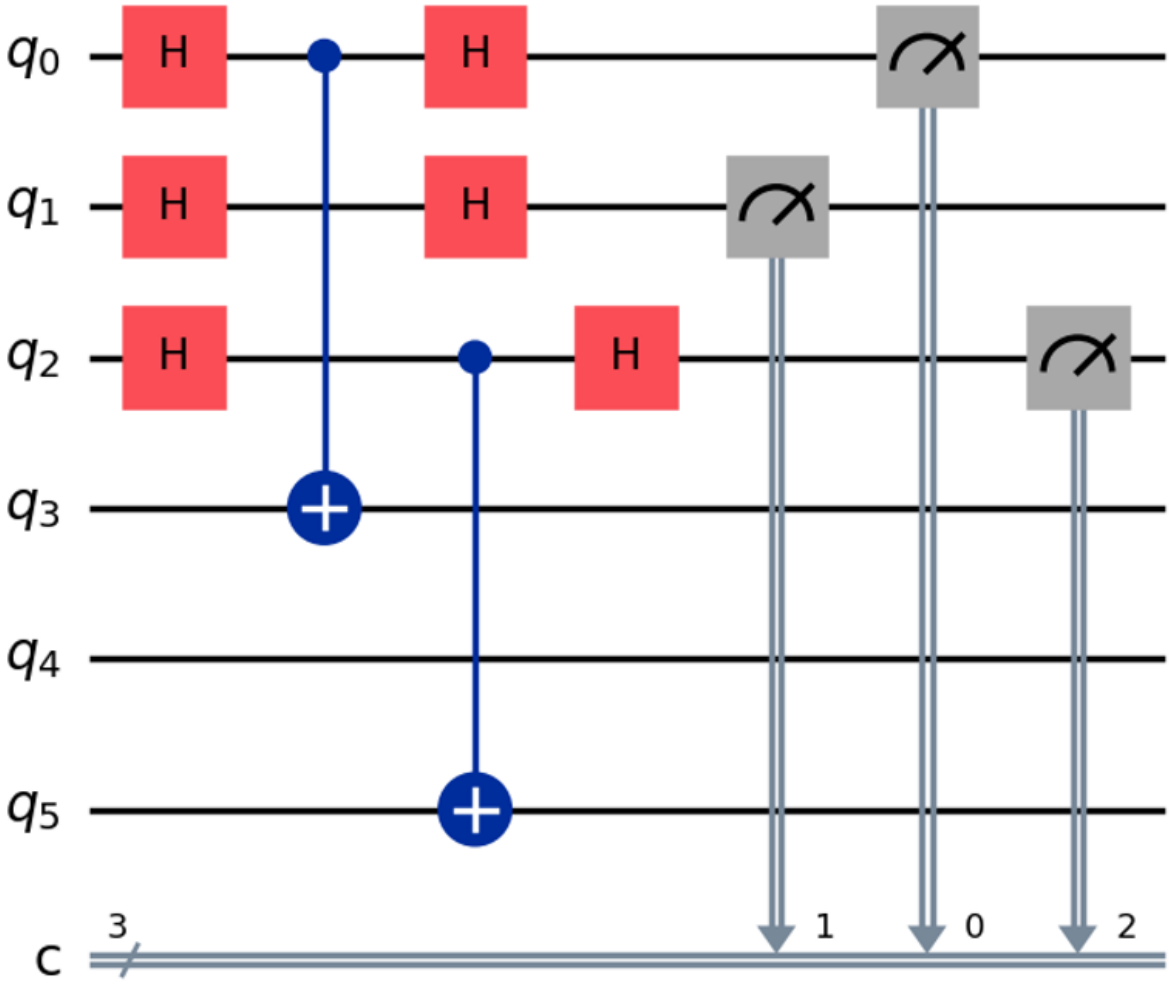

Fig 6: graphical representation of quantum of Simon's algorithm

4.5 Grover's Algorithm

[1]The last variation of the unknown function f is the following

$$f : \{0, 1\}\ n \longrightarrow \{0, 1\}$$

where only one input, s, outputs the value 1.

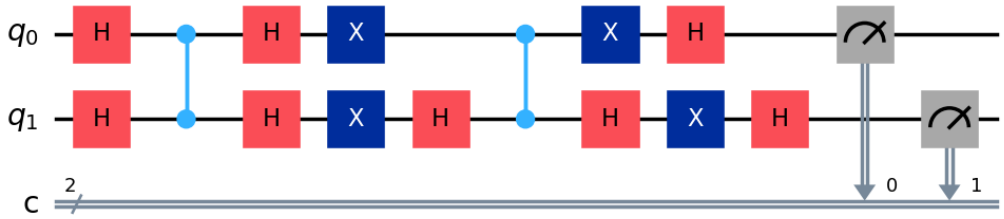

Fig 7: graphical representation of quantum of Grover's algorithm

Applying Hadamard gate to the input qubit yields

$$|\Psi\rangle_{t_1} = H^{\otimes n}|0\rangle^{\otimes n} = |+\rangle^{\otimes n} = \frac{1}{\sqrt{N}}(|0\rangle + |1\rangle)^{\otimes n} = \frac{1}{\sqrt{N}}\sum_{x\in\{0,1\}^n}|x\rangle$$

Hence the state $|\Psi\rangle_{t1} =$

$$= \frac{1}{\sqrt{N}}|s\rangle + \sqrt{\frac{N-1}{N}}|w\rangle$$

Applying the oracle for phase inversion we obtain

$$\begin{aligned}|\Psi\rangle_{t_2} &= U_f|\Psi\rangle_{t_1} \\ &= (-1)^{f(s)}\sin(\theta)|s\rangle + (-1)^{f(w)}\cos(\theta)|w\rangle \\ &= -\sin(\theta)|s\rangle + \cos(\theta)|w\rangle\ ,\end{aligned}$$

Query complexity is O(√ n)

5. Results

All simulations were carried out on Microsoft Visual Studio Code using qiskit, qiskit_Aersimulator, QuantumCircuit and matlab(matplotlib) libraries on python. Intel core i7-12650H CPU and Nvidea RTX 3050 GPU. Deutsch, Deutsch Jozsa, Bernstein Vazirani, Simon and Grover's Algorithms are simulated and output analysed.

6.References:

[1] R. Pereira da Silva, "Oracle Quantum Algorithms: An Overview," *SSRN Electronic Journal*, Jun. 2024. [Online]. Available: https://ssrn.com/abstract=4862127. [Accessed: Nov. 22, 2024].

[2] https://www.google.com/search?q=what+is+quantum+computing&rlz=1C1ONGR_en-GBIN1073IN1073&oq=what+is+quantum+computing&gs_lcrp=EgZjaHJvbWUyDAgAEEUYORixAxiABDIHCAEQABiABDIHCAIQABiABDIHCAMQABiABDIHCAQQABiABDIHCAUQABiABDIHCAYQABiABDIHCAcQABiABDIHCAgQABiABDIHCAkQABiABNIBCDg0MzlqMGo3qAIAsAIA&sourceid=chrome&ie=UTF-8

[3] I. G. Karafyllidis, "Quantum computer simulator based on the circuit model of quantum computation," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 52, no. 8, pp. 1590-1596, Aug. 2005, doi: 10.1109/TCSI.2005.851999.

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

[5] D. Deutsch, "Quantum theory, the Church-Turing principle, and the universal quantum computer," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985

[6] https://github.com/Qiskit/qiskit-aer

Course material for- “Quantum Entanglement and Quantum Computing”.

7.Appendix:

1)Deutsch Algorithm-

```
# Import libraries
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer  # Import Aer for simulation
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

# Deutsch Algorithm Circuit
qc = QuantumCircuit(2, 1)  # 2 qubits, 1 classical bit
qc.h(0)  # Apply Hadamard to input qubit
qc.x(1)  # Initialize target qubit in |1>
qc.h(1)  # Hadamard to target qubit

# Oracle for f(x) = x (balanced function)
qc.cx(0, 1)

qc.h(0)  # Apply Hadamard to input qubit
qc.measure(0, 0)  # Measure input qubit

# Simulate the circuit
simulator = Aer.get_backend('qasm_simulator')
compiled_circuit = transpile(qc, simulator)  # Transpile for the simulator
result = simulator.run(compiled_circuit, shots=1024).result()
counts = result.get_counts()
print("Deutsch Algorithm Results:", counts)

# Draw the circuit
qc.draw('mpl')
```

```
plt.show()

# Plot histogram
plot_histogram(counts)
plt.show()
```

2)Deutsch-Jozsa Algorith-

```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator

# Define the Deutsch-Jozsa oracle for a balanced function
def deutsch_jozsa_oracle(qc, n):
    """Implement a balanced oracle f(x) = x[0] XOR x[1]."""
    qc.cx(0, n)  # XOR between qubit 0 and output qubit
    qc.cx(1, n)  # XOR between qubit 1 and output qubit

# Number of input qubits
n = 2
# Total qubits (n input + 1 output)
total_qubits = n + 1

# Create the quantum circuit
qc = QuantumCircuit(total_qubits)

# Initialize the output qubit in the |1> state
qc.x(n)
qc.h(n)

# Apply Hadamard gates to the input qubits
qc.h(range(n))
```

```
# Apply the Deutsch-Jozsa oracle
deutsch_jozsa_oracle(qc, n)

# Apply Hadamard gates to the input qubits again
qc.h(range(n))

# Measure all qubits
qc.measure_all()

# Initialize the AerSimulator
simulator = AerSimulator()

# Run the circuit on the simulator
job = simulator.run(qc, shots=1024)
result = job.result()
counts = result.get_counts()

# Output the results
print("Counts:", counts)
qc.draw(output="mpl")
plt.show()
```

3)Bernstein-Vazirani Algorithm

```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
import matplotlib.pyplot as plt

n = 4  # Number of bits in the secret string
s = "1011"  # Secret string
```

```
# Create a quantum circuit with n input qubits and 1 output qubit
qc = QuantumCircuit(n + 1, n)

# Apply Hadamard gates to all input qubits
for qubit in range(n):
    qc.h(qubit)
# Initialize the last qubit in the |1> state
qc.x(n)
qc.h(n)

# Oracle for the secret string
for i, bit in enumerate(s):
    if bit == "1":
        qc.cx(i, n)

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

# Measure the input qubits
qc.measure(range(n), range(n))

# Simulate using AerSimulator
simulator = AerSimulator()
job = simulator.run(qc, shots=1024)
result = job.result()
counts = result.get_counts()

# Output the results
```

```
print("Bernstein-Vazirani Algorithm Results:", counts)
qc.draw('mpl')
plt.show()
```

4)Simon’s Algorithm-

```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
import matplotlib.pyplot as plt
n = 3  # Number of qubits
s = "101"  # Secret string

# Create a quantum circuit with 2n qubits: n input and n output
qc = QuantumCircuit(2 * n, n)

# Apply Hadamard gates to the input qubits
for qubit in range(n):
    qc.h(qubit)

# Oracle for f(x) = f(x XOR s)
for i, bit in enumerate(s):
    if bit == "1":
        qc.cx(i, n + i)

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

# Measure the input qubits
qc.measure(range(n), range(n))
```

```
# Simulate using AerSimulator
simulator = AerSimulator()
job = simulator.run(qc, shots=1024)
result = job.result()
counts = result.get_counts()

# Output the results
print("Simon's Algorithm Results:", counts)
qc.draw('mpl')
plt.show()
```

5)Grover’s Algorithm-

```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
import matplotlib.pyplot as plt

n = 2  # Number of qubits
qc = QuantumCircuit(n, n)

# Apply Hadamard gates to all qubits
qc.h(range(n))

# Oracle: Flip the amplitude of |11>
qc.cz(0, 1)

# Diffusion operator
qc.h(range(n))
qc.x(range(n))
qc.h(n - 1)
qc.cz(0, 1)
```

```
qc.h(n - 1)
qc.x(range(n))
qc.h(range(n))

# Measure
qc.measure(range(n), range(n))

# Simulate using AerSimulator
simulator = AerSimulator()
job = simulator.run(qc, shots=1024)
result = job.result()
counts = result.get_counts()

# Output the results
print("Grover's Algorithm Results:", counts)
qc.draw('mpl')
plt.show()
```