https://colab.research.google.com/drive/1y3qloQ3lJ_HzkRfxX8MLQAelDZsy7l3S?usp=sharingpura%20qc%20run%20kiya%20experiment%20wisesave%20karke%20rakho%20gmail%20parexecution%20ke%20time%20pe%20bas%20copy%20karlo%20code%20isse

 # 1.1

```
Pip install qiskit
Pip install qiskit_aer

from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

qc = QuantumCircuit(1, 1)

# Apply multiple Hadamard gates
qc.h(0)
qc.h(0)
qc.h(0)
qc.h(0)

# Measure the qubit
qc.measure(0, 0)

# Use Aer simulator
simulator = AerSimulator()

# Compile and run
compiled_circuit = transpile(qc, simulator)
job = simulator.run(compiled_circuit, shots=100)
result = job.result()

# Get counts
counts = result.get_counts()
print(qc.draw())
print("Measurement counts:", counts)

# Plot histogram
plot_histogram(counts)
```

#1.2

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

qc = QuantumCircuit(1, 1)

# Apply multiple Hadamard gates
qc.x(0)
qc.h(0)
qc.h(0)
qc.h(0)

# Measure the qubit
```

```
qc.measure(0, 0)

# Use Aer simulator
simulator = AerSimulator()

# Compile and run
compiled_circuit = transpile(qc, simulator)
job = simulator.run(compiled_circuit, shots=100)
result = job.result()

# Get counts
counts = result.get_counts()
print(qc.draw())
print("Measurement counts:", counts)

# Plot histogram
plot_histogram(counts)
```

#1.3

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

qc = QuantumCircuit(1, 1)

# Apply multiple Hadamard gates
qc.x(0)
qc.z(0)
qc.h(0)
qc.y(0)
qc.s(0)
# Measure the qubit
qc.measure(0, 0)

# Use Aer simulator
simulator = AerSimulator()

# Compile and run
compiled_circuit = transpile(qc, simulator)
job = simulator.run(compiled_circuit, shots=100)
result = job.result()

# Get counts
counts = result.get_counts()
print(qc.draw())
print("Measurement counts:", counts)

# Plot histogram
plot_histogram(counts)
```

#2.1

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
```

```
# Create a Quantum Circuit with 2 qubits and 2 classical bits
qc = QuantumCircuit(2, 2)
# Apply gates
qc.h(0)
qc.cx(0,1)
qc.z(0)
qc.x(1)
qc.h(0)
# Measure both qubits
qc.measure([0, 1], [0, 1])
# Simulate
simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
job = simulator.run(compiled_circuit, shots=500)
print(qc.draw())
result = job.result()
```

#2.2

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
# Create a Quantum Circuit with 2 qubits and 2 classical bits
qc = QuantumCircuit(2, 2)
# Apply gates
qc.h(0)
qc.x(0)
qc.x(0)
qc.y(0)
qc.h(0)
qc.z(0)
qc.h(0)
# Measure both qubits
qc.measure([0, 1], [0, 1])
# Simulate
simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
job = simulator.run(compiled_circuit, shots=500)
print(qc.draw())
result = job.result()
```

#2.3

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram


# Create circuit
qc = QuantumCircuit(1, 1)

# Apply gates
qc.h(0)
qc.x(0)
qc.x(0)
qc.h(0)
```

```
qc.z(0)
qc.h(0)
qc.y(0)
qc.h(0)

# Measure
qc.measure(0, 0)

# Simulate
simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
job = simulator.run(compiled_circuit, shots=100)
result = job.result()

# Get results
counts = result.get_counts()
print(qc.draw())
print("Measurement counts:", counts)
```

#3.0

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram


def create_bell_state(bell_type):
    qc = QuantumCircuit(2, 2)
    qc.h(0)
    qc.cx(0, 1)

    # Apply transformations for different Bell states
    if bell_type == "Phi-":
        qc.z(0)
    elif bell_type == "Psi+":
        qc.x(1)
    elif bell_type == "Psi-":
        qc.x(1)
        qc.z(0)

    # Measure both qubits
    qc.measure([0, 1], [0, 1])
    return qc

# Setup simulator
simulator = AerSimulator()
bell_states = ["Phi+", "Phi-", "Psi+", "Psi-"]
results = {}

# Run each Bell state
for state in bell_states:
    qc = create_bell_state(state)
    compiled = transpile(qc, simulator)
    job = simulator.run(compiled, shots=1000)
    result = job.result()
    counts = result.get_counts()
    results[state] = counts
```

```
    print(f"\nBell State |{state}> Measurement Counts: {counts}")
    print(qc.draw())

# Plot combined histogram
plot_histogram(results)
```

#5.1

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt
# Create a quantum circuit with 3 qubits and 3 classical bits
qc = QuantumCircuit(3, 3)
# Apply gates
qc.x(0)
qc.z(0)
qc.x(0)
qc.cx(0, 1)
qc.swap(1, 2)
# Measure all qubits
qc.measure([0, 1, 2], [0, 1, 2])
# Simulate the circuit
simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
job = simulator.run(compiled_circuit, shots=100)
result = job.result()
# Get measurement counts
counts = result.get_counts()
print(qc.draw())
```

#5.2

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

# Create a 3-qubit, 3-classical-bit circuit
qc = QuantumCircuit(3, 3)

# Quantum operations
qc.y(0)
qc.h(0)
qc.z(0)
qc.h(0)
qc.cx(0, 1)
qc.swap(1, 2)

# Measurement
qc.measure([0, 1, 2], [0, 1, 2])

# Simulation
simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
job = simulator.run(compiled_circuit, shots=100)
result = job.result()
```

```
counts = result.get_counts()

# Outputs
print(qc.draw())
```

#5.3

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

# Create a 3-qubit, 3-classical-bit circuit
qc = QuantumCircuit(3, 3)

# Apply gates
qc.h(0)
qc.h(0)
qc.z(0)
qc.x(0)
qc.cx(0, 1)
qc.swap(1, 2)

# Measure all qubits
qc.measure([0, 1, 2], [0, 1, 2])

# Simulate
simulator = AerSimulator()
compiled_circuit = transpile(qc, simulator)
job = simulator.run(compiled_circuit, shots=100)
result = job.result()
counts = result.get_counts()

# Output circuit and measurement counts
print(qc.draw())
```

#6.0

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

def deutsch_jozsa(oracle, n):
    qc = QuantumCircuit(n + 1, n)

    qc.x(n)
    qc.h(n)

    qc.h(range(n))

    qc.append(oracle, range(n + 1))
```

```
    qc.h(range(n))

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

    backend = AerSimulator()
    qc_transpiled = transpile(qc, backend)
    job = backend.run(qc_transpiled, shots=200)
    result = job.result()
    counts = result.get_counts()

    return qc, counts

def constant_oracle(n, value=0):
    oracle = QuantumCircuit(n + 1)
    if value == 1:
        oracle.x(n)
    return oracle.to_gate(label="Const")

def balanced_oracle(n):
    oracle = QuantumCircuit(n + 1)
    for i in range(n):
        oracle.cx(i, n)
    return oracle.to_gate(label="Bal")

if __name__ == "__main__":
    n = 3

    const_oracle = constant_oracle(n, value=0)
    qc_const, counts_const = deutsch_jozsa(const_oracle, n)

    bal_oracle = balanced_oracle(n)
    qc_bal, counts_bal = deutsch_jozsa(bal_oracle, n)

    print("Constant Function Result:", counts_const)
    print("Balanced Function Result:", counts_bal)

    print("Circuit for Constant Oracle:")
    print(qc_const.draw())
    print("Circuit for Balanced Oracle:")
    print(qc_bal.draw())
```

#7.0

```
# Install Qiskit if not installed (uncomment if needed)
# !pip install qiskit qiskit-aer --upgrade -q

from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
```

```
# --- Parameters ---
n = 3  # Number of qubits
marked_state = '101'  # State we want to search for

# --- 1. Initialize Circuit ---
qc = QuantumCircuit(n, n)
qc.h(range(n))  # Put all qubits in superposition

# --- 2. Oracle: Flip phase of the marked state ---
for i, bit in enumerate(marked_state):
    if bit == '0':
        qc.x(i)

qc.h(n - 1)
qc.mcx(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)

# --- 3. Diffusion Operator (Grover's amplification) ---
qc.h(range(n))
qc.x(range(n))

qc.h(n - 1)
qc.mcx(list(range(n - 1)), n - 1)
qc.h(n - 1)

qc.x(range(n))
qc.h(range(n))

# --- 4. Measurement ---
qc.measure(range(n), range(n))

# --- 5. Run on Simulator ---
simulator = AerSimulator()
compiled = transpile(qc, simulator)
job = simulator.run(compiled, shots=1000)
result = job.result()
counts = result.get_counts()

# --- 6. Output ---
print(f"Grover's algorithm result for marked state |{marked_state}>:")
print(counts)

plot_histogram(counts)
print(qc.draw())
```

#8.0

```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.circuit.library import QFT
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt
```

```
# Create a 3-qubit circuit
qc = QuantumCircuit(3)

# Initialize qubits
qc.x(0)
qc.x(2)
qc.barrier()

# Apply Quantum Fourier Transform
qc.append(QFT(num_qubits=3), range(3))

# Measure all qubits
qc.measure_all()

# Display circuit diagram
print("--- QFT Circuit Diagram ---")
print(qc)

# Run on AerSimulator
backend = AerSimulator()
transpiled_qc = transpile(qc, backend)
job = backend.run(transpiled_qc, shots=1024)
result = job.result()
counts = result.get_counts()

# Display measurement results
print("\n--- QFT Measurement Results ---")
print(counts)

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