Exercises for Introduction to Quantum Computing

Name: Pugazharasu Anancia Devaneyan (s6puanan)

Matriculation number: 3300280

In [ ]:
```
import numpy as np
import qiskit as qi
from qiskit import IBMQ, Aer
from qiskit.providers.ibmq import least_busy
from qiskit import QuantumCircuit, transpile

from qiskit.visualization import plot_histogram
```

1. Quantum Fourier Transform

The quantum Fourier transform operation acting on a state is a unitary operation, thus for

$$\text{qFT}^{\dagger}|\phi\rangle = |100\rangle$$

we simply act with a qFT on both sides of the equation to get,

$$|\phi\rangle = \text{qFT}|100\rangle$$

Thus, to obtain the state, we simply need to evaluate the RHS. This can be done as the qFT in the computational basis is expressed as,

$$\text{qFT}|j\rangle = \frac{1}{\sqrt{8}}\sum_{k=0}^{7} e^{2\pi i\frac{jk}{8}}|k\rangle$$

Therefore, we have

$$|\phi\rangle = \frac{1}{\sqrt{(8)}}(|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle)$$

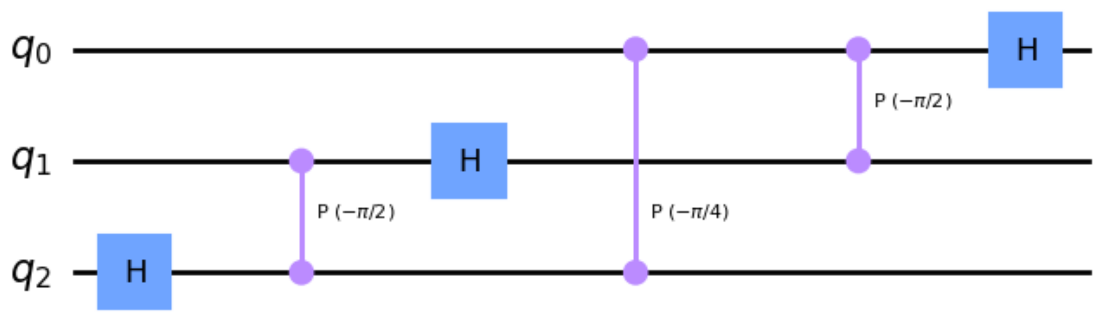
The probability of measuring the output 100 from the state $|\phi\rangle$ is given by computing the inner product of $|\phi\rangle$ with $|100\rangle$,

$$\langle 100|\phi\rangle = \left(\frac{1}{\sqrt{8}}\right)^2 = \frac{1}{8}$$

The inverse qFT circuit can be implemented by simply reversing the order of unitaries and inverting the angles of the phase gates applied i.e. $R_h \to R_h^{-1}$, thus for the 3-qubit case (excluding swaps) we have:

In [ ]:
```
inv_qFT = QuantumCircuit(3)
inv_qFT.h(2)
inv_qFT.cp(-np.pi/2, 2, 1)
inv_qFT.h(1)
inv_qFT.cp(-np.pi/4, 2, 0)
inv_qFT.cp(-np.pi/2, 1, 0)
inv_qFT.h(0)
inv_qFT.draw('mpl')
```

Out[ ]:



2. Deutsch–Jozsa algorithm

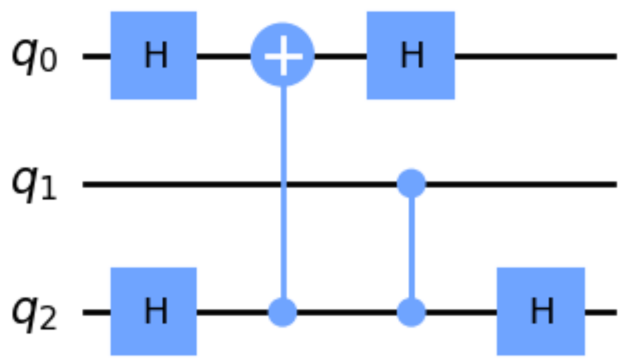
We wish to implement an Oracle U_f with the action

$$U_f|q_0q_1q_2\rangle = |q_0q_1(q_2 \oplus f(q_0,q_1))\rangle$$

This is given to us via the quantum circuit,

In [ ]:
```
oracle = QuantumCircuit(3)
oracle.h(0)
oracle.h(2)
oracle.cx(2,0)
oracle.h(0)
oracle.cz(2,1)
oracle.h(2)
oracle.draw('mpl')
```

Out[ ]:



We can check if this does indeed implement the Oracle we desire by means of applying it to an arbitary state,

In [ ]:
```
state = qi.quantum_info.Statevector.from_label('001')
state = state.evolve(oracle)
state.draw(output='latex')
```

Out[ ]:

$$|101\rangle$$

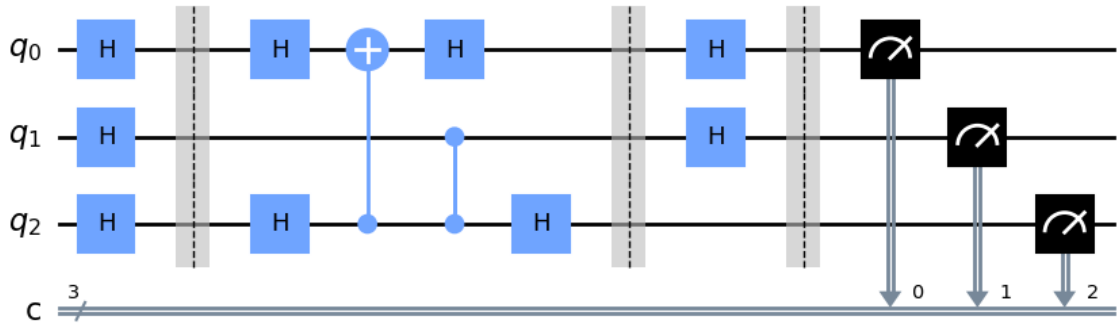
Now that we have shown that the quantum circuit implements an Oracle, we can check if the Oracle is balanced or constant by means of implementing a truth table for it,

| $q_0q_1q_2$ | U_f || - | - || 000 | 000 || 001 | 100 || 010 | 011 || 011 | 010 || 100 | 101 || 101 | 100 | | 110 | 110 || 111 | 111 |

We can see that the Oracle is balanced! We shall now construct the Deutsch Josza algorithm using Qiskit,

In [ ]:
```
deutsch_algo = QuantumCircuit(3,3)
deutsch_algo.h(0)
deutsch_algo.h(1)
deutsch_algo.h(2)
deutsch_algo.barrier()
deutsch_algo = deutsch_algo.compose(oracle)
deutsch_algo.barrier()
deutsch_algo.h(0)
deutsch_algo.h(1)
deutsch_algo.barrier()
deutsch_algo.measure(0, 0)
deutsch_algo.measure(1, 1)
deutsch_algo.measure(2, 2)
deutsch_algo.draw('mpl')
```

Out[ ]:



We can now check using the circuit if the function is balanced or not by means of running it on a QASM simulator,

In [ ]:

```
backend = qi.Aer.get_backend('qasm_simulator')
job = qi.execute(deutsch_algo, backend, shots = 1024)
results = job.result()
counts = results.get_counts(deutsch_algo)
print(counts)
```

```
{'000': 495, '100': 529}
```

We can see that the function is balanced as shown above, thus the circuit implements the Deutsch Josza algorithm.