In [1]:
```
#initialization
import matplotlib.pyplot as plt
import numpy as np
import math

# importing Qiskit
from qiskit import transpile, assemble
from qiskit_aer import AerSimulator
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister

# import basic plot tools
from qiskit.visualization import plot_histogram
```

In [2]:
```
def qft_dagger(qc, n):
    """n-qubit QFTdagger the first n qubits in circ"""
    # Don't forget the Swaps!
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-math.pi/float(2**(j-m)), m, j)
        qc.h(j)
```

In [6]:
```
#Aer.get_backend('aer_simulator')
aer_sim = AerSimulator()
```

In [7]:
```
# Create and set up circuit
qpe2 = QuantumCircuit(4, 3)

# Apply H-Gates to counting qubits:
for qubit in range(3):
    qpe2.h(qubit)

# Prepare our eigenstate |psi>:
qpe2.x(3)

# Do the controlled-U operations:
angle = 2*math.pi/3
```

```
repetitions = 1
for counting_qubit in range(3):
    for i in range(repetitions):
        qpe2.cp(angle, counting_qubit, 3);
    repetitions *= 2

# Do the inverse QFT:
qft_dagger(qpe2, 3)

# Measure of course!
for n in range(3):
    qpe2.measure(n,n)

qpe2.draw()
```

Out[7]:

```
     ┌───┐                                                                     »
q_0: ┤ H ├─■───────────────────────────────────────────────────────────────────»
     ├───┤ │                                                                   »
q_1: ┤ H ├─┼──────────■──────────■─────────────────────────────────────────────»
     ├───┤ │          │          │                                             »
q_2: ┤ H ├─┼──────────┼──────────┼──────────■──────────■──────────■────────────»
     ├───┤ │P(2π/3)   │P(2π/3)   │P(2π/3)   │P(2π/3)   │P(2π/3)   │P(2π/3)     »
q_3: ┤ X ├─■──────────■──────────■──────────■──────────■──────────■────────────»
     └───┘                                                                     »
c: 3/══════════════════════════════════════════════════════════════════════════»
                                                                               »
«                     ┌───┐                                         ┌─┐
«q_0: ──────────────X─┤ H ├─■──────────────■────────────────────────┤M├───────────
«                   │ └───┘ │P(-π/2) ┌───┐ │                        └╥┘      ┌─┐
«q_1: ──────────────┼───────■────────┤ H ├─┼─────────■───────────────╫───────┤M├───
«                   │                └───┘ │P(-π/4)  │P(-π/2)        ║ ┌───┐ └╥┘┌─┐
«q_2: ─■────────────X──────────────────────■─────────■───────────────╫─┤ H ├──╫─┤M├
«      │P(2π/3)                                                      ║ └───┘  ║ └╥┘
«q_3: ─■─────────────────────────────────────────────────────────────╫────────╫──╫─
«                                                                    ║        ║  ║
«c: 3/═══════════════════════════════════════════════════════════════╩════════╩══╩═
«                                                                    0        1  2
```

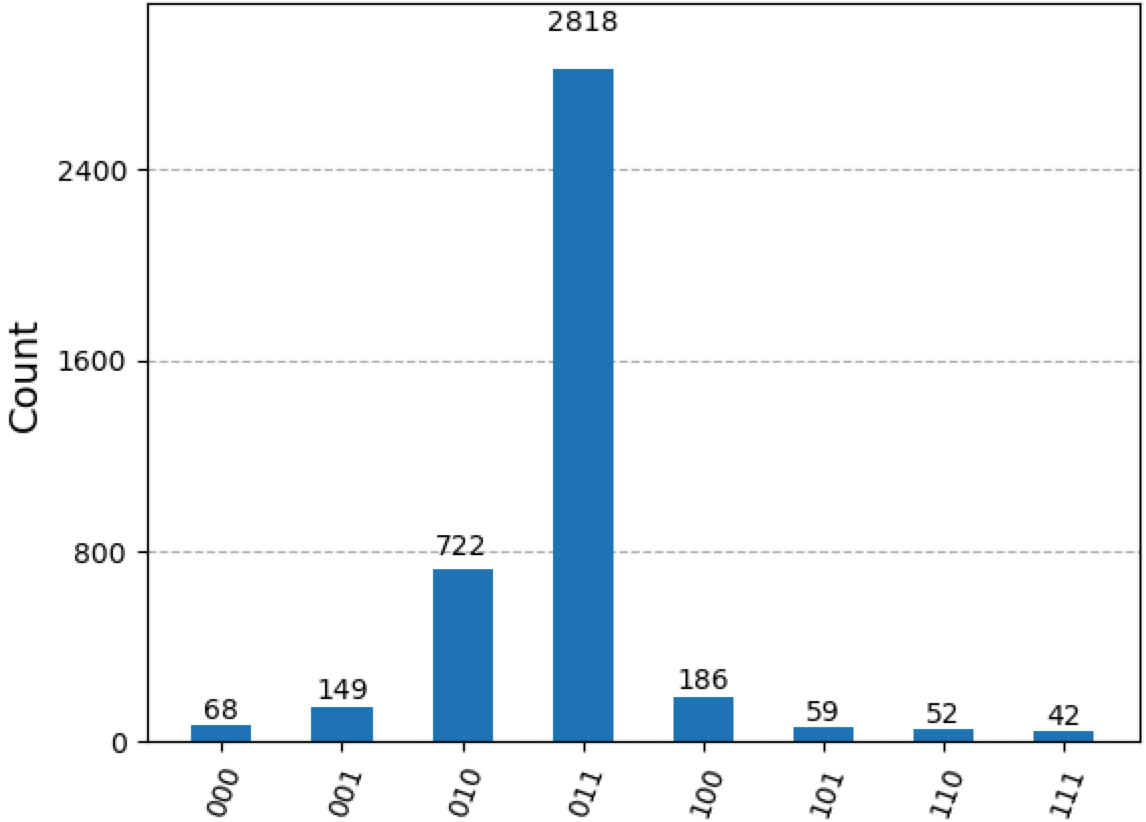
In [8]:

```
# Let's see the results!
# aer_sim = Aer.get_backend('aer_simulator')
shots = 4096
t_qpe2 = transpile(qpe2, aer_sim)
qobj = assemble(t_qpe2, shots=shots)
```

```
results = aer_sim.run(qobj).result()
answer = results.get_counts()

plot_histogram(answer)
```

```
/tmp/ipykernel_5723/1318575361.py:6: DeprecationWarning: Using a qobj for run() is deprecated as of qiskit-aer 0.14 and will be removed no sooner than 3 months from that release date. Transpiled circuits should now be passed directly using `backend.run(circuits, **run_options).
  results = aer_sim.run(qobj).result()
```

Out[8]:

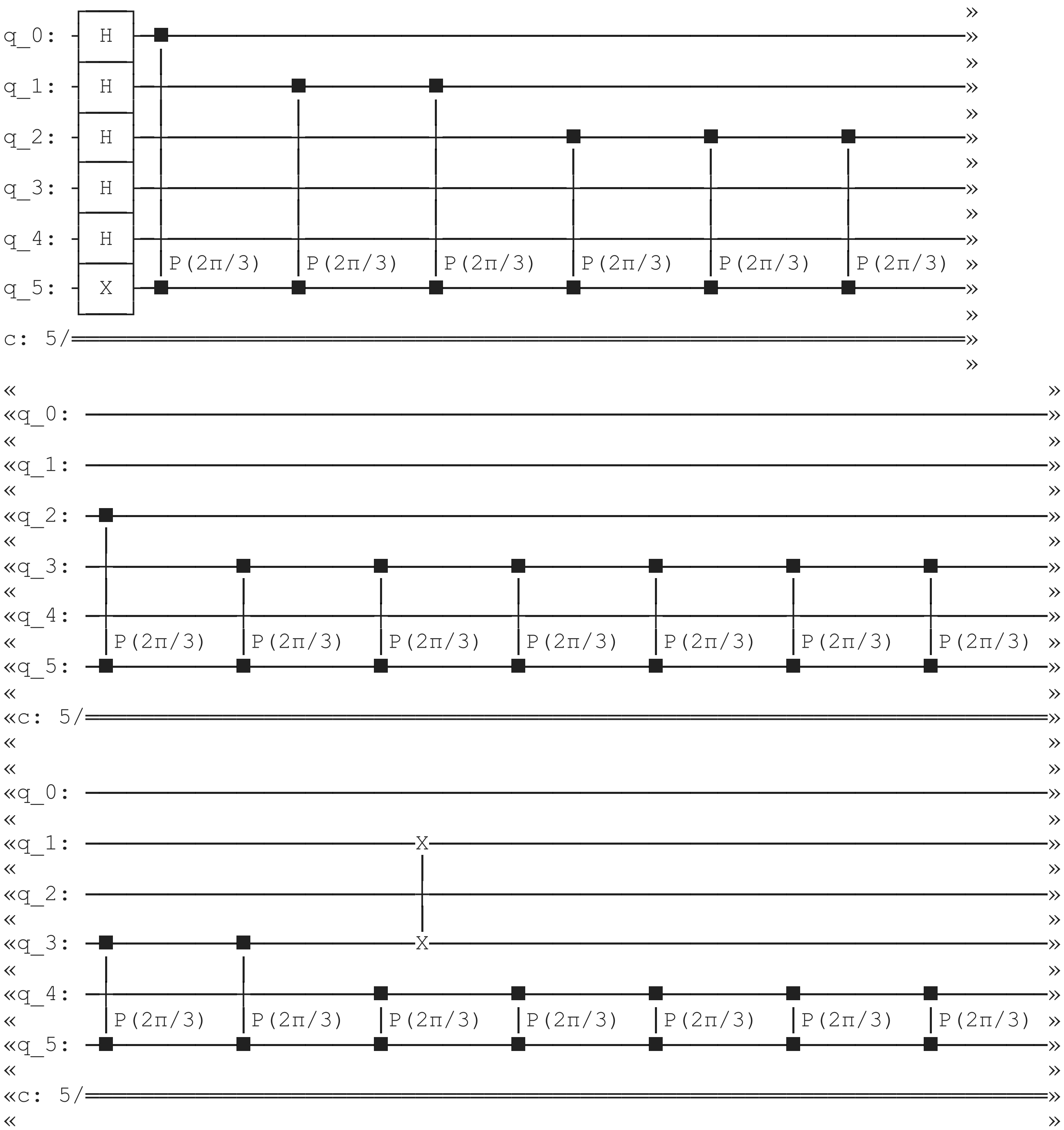


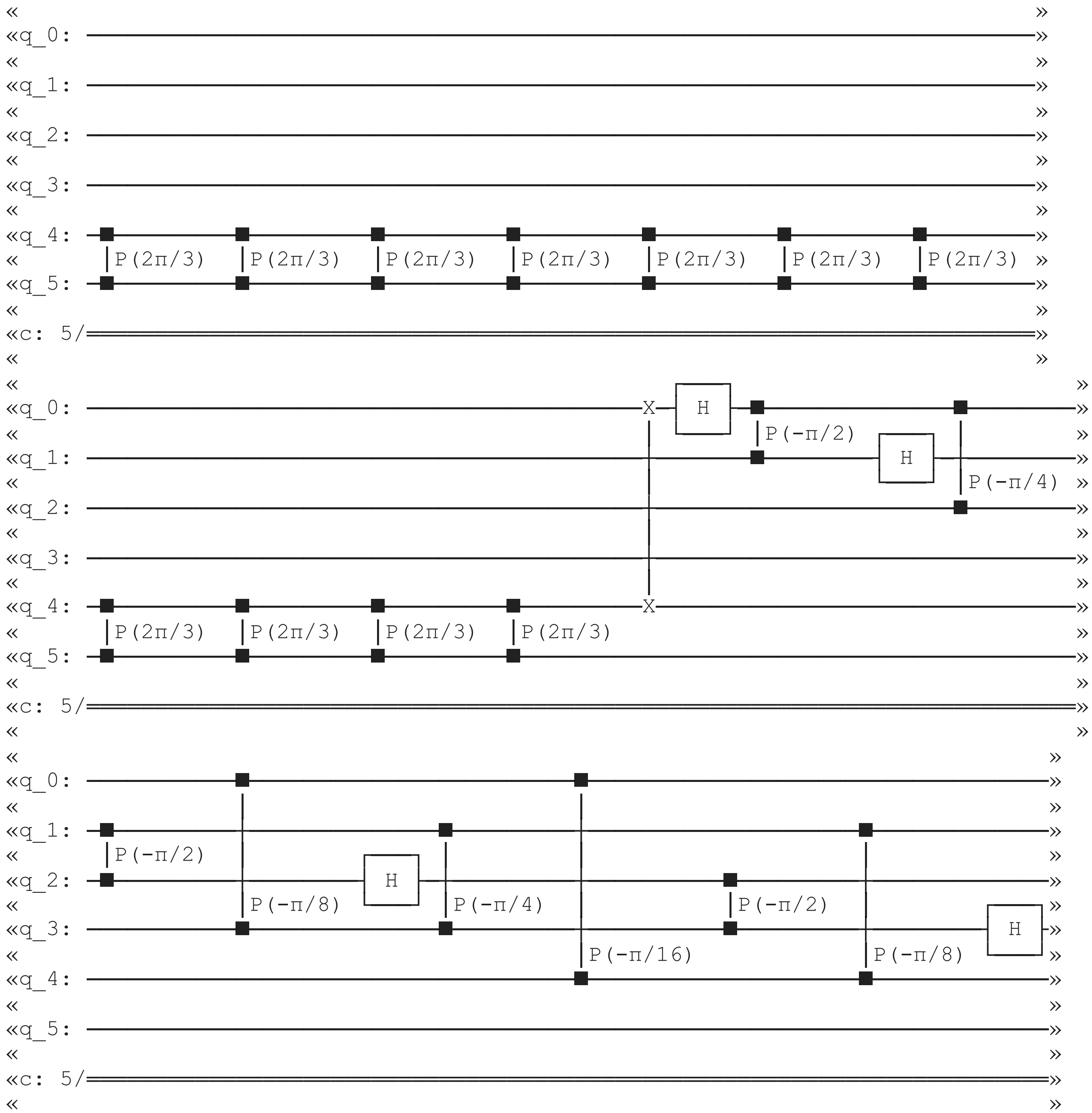
We are expecting the result $\theta = 0.3333\ldots$, and we see our most likely results are `010(bin) = 2(dec)` and `011(bin) = 3(dec)`. These two results would tell us that $\theta = 0.25$ (off by 25%) and $\theta = 0.375$ (off by 13%) respectively. The true value of θ lies between the values we can get from our counting bits, and this gives us uncertainty and imprecision.

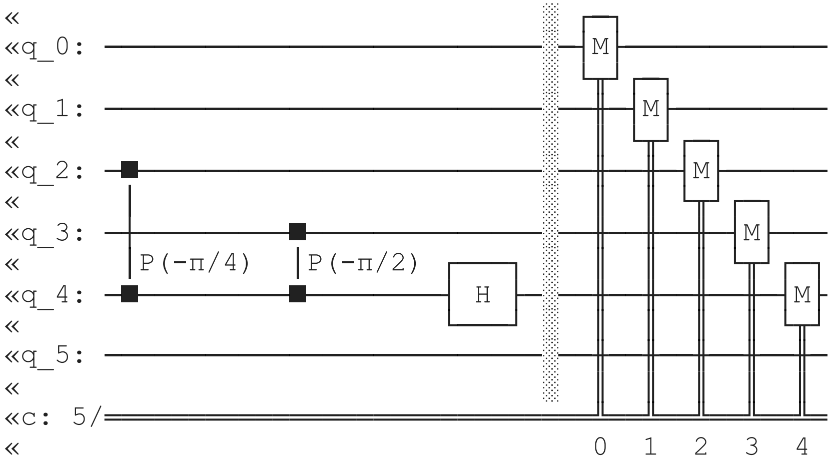
The second question is for t=5

In [10]:
```
# Create and set up circuit
qpe3 = QuantumCircuit(6, 5)

# Apply H-Gates to counting qubits:
for qubit in range(5):
    qpe3.h(qubit)

# Prepare our eigenstate |psi>:
qpe3.x(5)

# Do the controlled-U operations:
angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(5):
    for i in range(repetitions):
        qpe3.cp(angle, counting_qubit, 5);
    repetitions *= 2

# Do the inverse QFT:
qft_dagger(qpe3, 5)

# Measure of course!
qpe3.barrier()
for n in range(5):
    qpe3.measure(n,n)

qpe3.draw()
```

Out[10]:



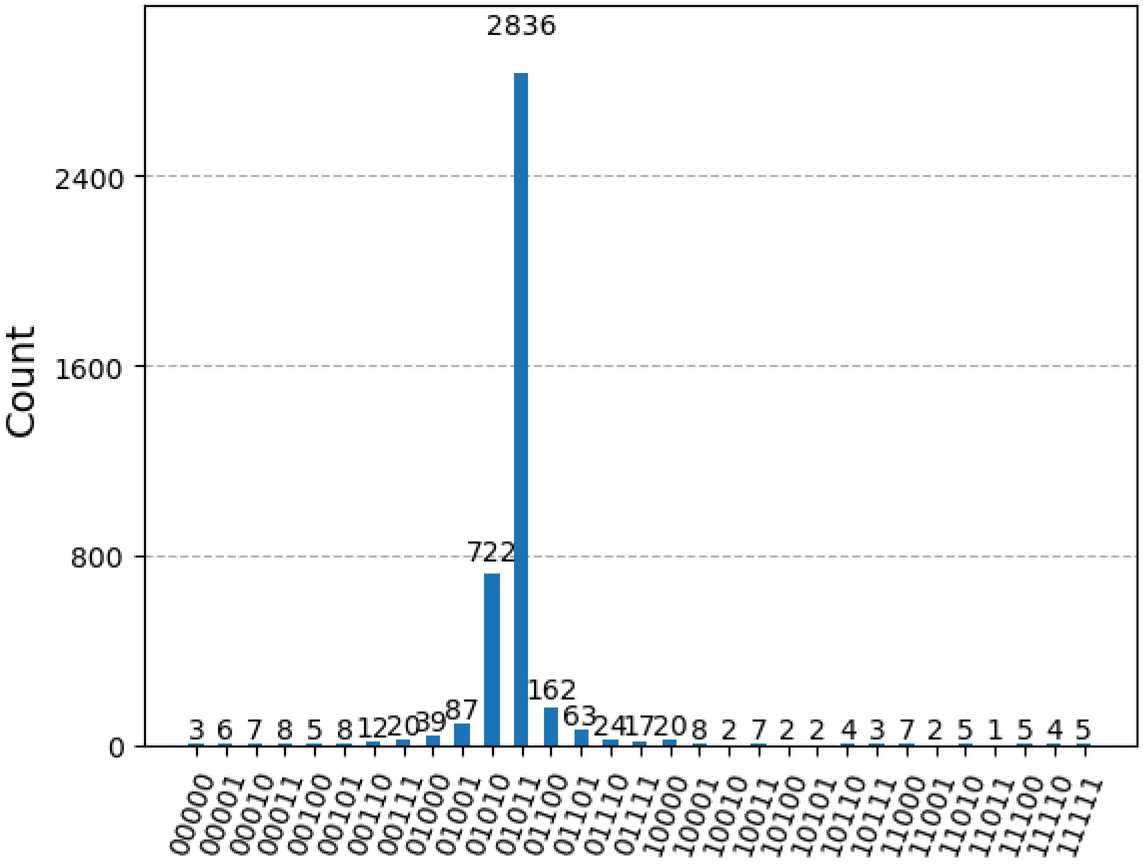

q_0:
q_1:
q_2:
q_3:
q_4:
q_5:
c: 5/
P(2π/3)
P(2π/3)
P(2π/3)
P(2π/3)
P(2π/3)
P(2π/3)
P(2π/3)
q_0:
q_1:
q_2:
q_3:
q_4:
q_5:
c: 5/
P(2π/3)
P(2π/3)
P(2π/3)
P(2π/3)
X
X
H
P(−π/2)
H
P(−π/4)
q_0:
q_1:
q_2:
q_3:
q_4:
q_5:
c: 5/
P(−π/2)
P(−π/8)
H
P(−π/4)
P(−π/16)
P(−π/2)
P(−π/8)
H



In [11]:
```
# Let's see the results!
# aer_sim = Aer.get_backend('aer_simulator')
shots = 4096
t_qpe3 = transpile(qpe3, aer_sim)
qobj = assemble(t_qpe3, shots=shots)
results = aer_sim.run(qobj).result()
answer = results.get_counts()

plot_histogram(answer)
```

```
/tmp/ipykernel_5723/652080245.py:6: DeprecationWarning: Using a qobj for run() is deprecated as of qiskit-aer 0.14 a
nd will be removed no sooner than 3 months from that release date. Transpiled circuits should now be passed directly
using `backend.run(circuits, **run_options).
  results = aer_sim.run(qobj).result()
```

Out[11]:



The two most likely measurements are now `01011` (decimal 11) and `01010` (decimal 10). Measuring these results would tell us θ is:

$$\theta = \frac{11}{2^5} = 0.344, \quad \text{or} \quad \theta = \frac{10}{2^5} = 0.313$$

These two results differ from $\frac{1}{3}$ by 3% and 6% respectively. A much better precision!

In [ ]: