quantum_algo_exercises

1

1.a

QFT

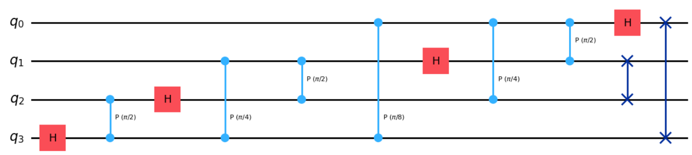


Figure 1: QFT

Inverse QFT

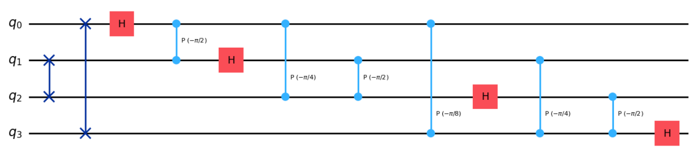


Figure 2: Inverse QFT

1.b

$$U_{QFT}\,|\phi\rangle = \left(\frac{1}{2\sqrt{2}} \sum_{j=0}^{7} \sum_{k=0}^{7} e^{-2\pi i jk/8} \; |k\rangle\langle j| \right)\left(\frac{1}{2}\sum_{l=0}^{7} \cos(2\pi l/8)\,|l\rangle\right)$$

$$= \frac{1}{4\sqrt{2}} \sum_{j=0}^{7}\left[\sum_{k=0}^{7} e^{-2\pi i jk/8} \cos\left(\frac{2\pi j}{8}\right)\right]|k\rangle$$

$$\frac{e^{-2\pi i jk/8} + e^{2\pi jk}}{2} = \cos\left(\frac{2\pi j}{8}\right)$$

$$\Rightarrow \sum_{k=0}^{7} e^{-2\pi i jk/8} \cos\left(\frac{2\pi j}{8}\right) = \sum_{k=0}^{7} e^{-2\pi i jk/8}\left(\frac{e^{2\pi j} + e^{-2\pi j}}{2}\right)$$

$$= \frac{1}{2}\left(\sum e^{-\frac{2\pi i j (k-1)}{8}} + \sum e^{-\frac{2\pi i j (k+1)}{8}}\right)$$

וכל: שווה 0 עבור $k \neq 1,7$! 1 עבור $k=1,7$

$\Leftarrow$

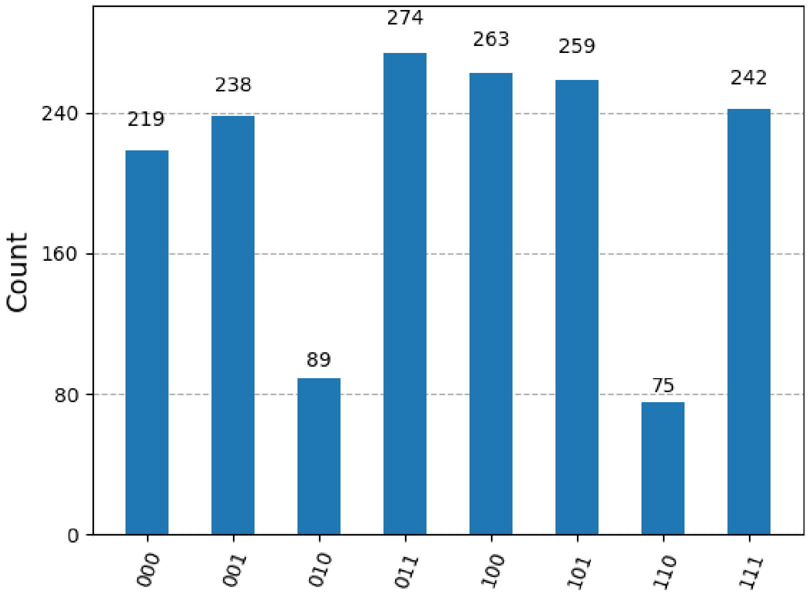
$$U_{QFT}\,|\phi\rangle = \frac{1}{\sqrt{2}}\left(|1\rangle + |7\rangle\right)$$

1.c

The result from b is encoded to the circuit with simple operations, then to reconstruct the original stage we used the inverse QFT and then $InverseQFT * QFT * \phi = \phi$. For more info see the src code: src/HW1.py

1.d

We can see that for j=2,6 the value is low, this is the excepted result from the original stage.



2

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)
```

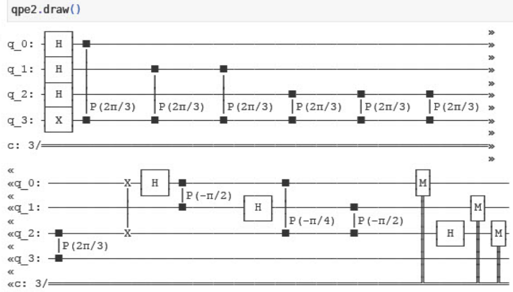
In [6]:
```
#Aer.get_backend('aer_simulator')
aer_sim = AerSimulator()
```

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]:

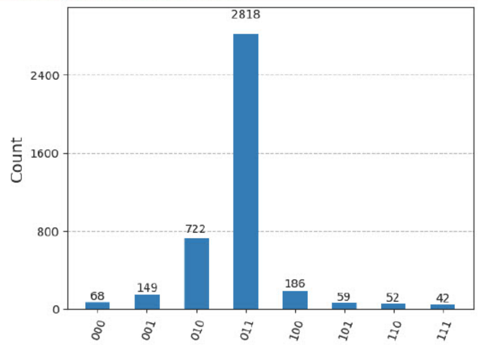


In [8]:
```
# Let's see the results!
# aer_sim = Aer.get_backend('aer_simulator')
shots = 4096
t_qpe2 = transpile(qpe2, aer_sim)
qobj = assemble(t_qpe2, shots=shots)
```

```
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 directl
y using `backend.run(circuits, **run_options).
  results = aer_sim.run(qobj).result()
```

Out[8]:



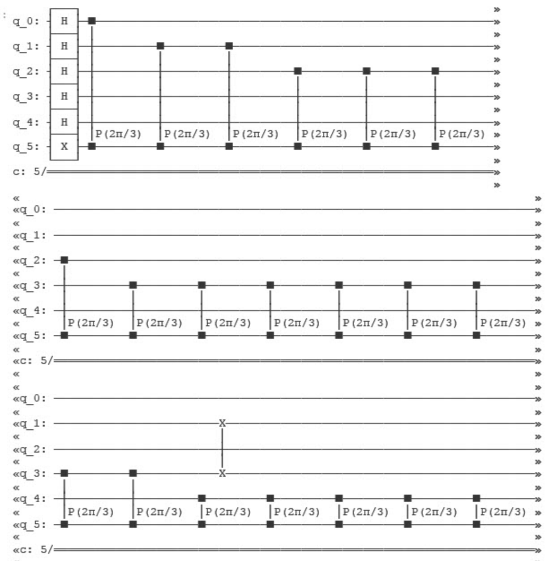
We are expecting the result $\theta = 0.3333\ldots$, and we see our most likely results are `010(bin) = 2(dec)` and `011(bin) = 3(dec)`. These two results would tell us that $\theta = 0.25$ (off by 25%) and $\theta = 0.375$ (off by 13%) respectively. The true value of θ lies between the values we can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

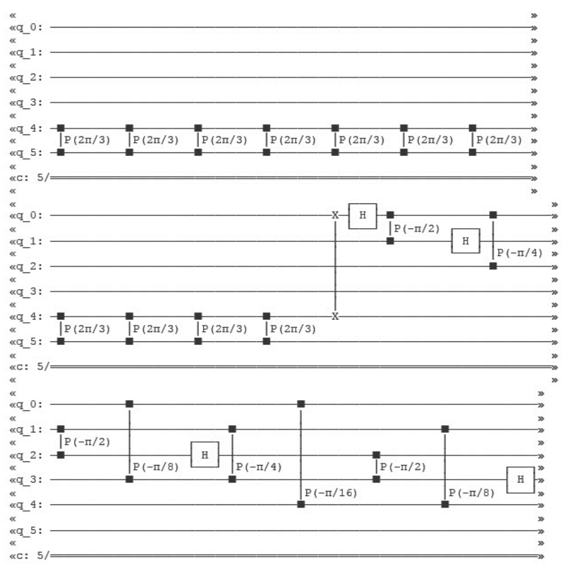
The second question is for t=5

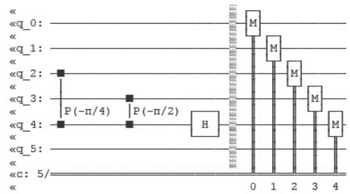
In [10]:

```
# 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/
H
X
P(2π/3)


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

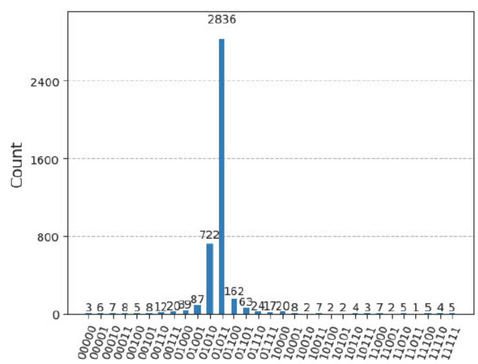


In [11]:

```python
# 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:

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

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

In [ ]:

3

תרגיל 3

נשתמש בלוגיקה דומה לזו של האלגוריתם
במאמר למציאה של מינימום.
בשונה, הפעם נגדיר הפונקציה:

$$f_i(j) = \begin{cases} 1 & \text{if } T[j] < T[i] \\ 0 & \text{otherwise} \end{cases}$$

האלגוריתם עצמו והניתוח של הסיבוכיות
יהיו כמו שמופיע במאמר.
נעבור על השלבים שמופיעים במאמר:

(1) נבחר אינדקס y אקראי בתחום $\{0..N-1\}$

(2) נחזור על הפעולות הבאות $O(\sqrt{N})$ פעמים:

(א) נאתחל מצב כמו כאן $|\psi\rangle = \sum_i \frac{1}{\sqrt{N}}|i\rangle$
סיבוכיות $O(\log N)$

(ב) נפעיל את האלגוריתם של גרובר על מנת לחפש אינדקסים X שקטנים מ y כלומר
שמקיימים $g_y(x) = 1$.
במידה ויש כזה x_0
ונסמן אותו כ- y עבור האיטרציה הבאה.

(3) בסוף, נחזיר את y כפלט של האלגוריתם.

הסיבוכיות זהה במאמר. כיון שכן יש
$O(\sqrt{N})$ איטרציות וכל הסיבוכיות של כל
איטרציה היא $O(\log N)$

4

4.a

שאלה 4

סעיף א:

$$\sum_a a_i \frac{\Delta_3(\Pi_\phi, \Pi_{a_i}, \Pi_\psi)}{\Delta_2(\Pi_\phi, \Pi_\psi)} =$$

$$\sum_a a_i \frac{Tr\left(|\phi\rangle\langle\phi| \, |a_i\rangle\langle a_i| \, |\psi\rangle\langle\psi| \right)}{Tr\left(|\phi\rangle\langle\phi| \, |\psi\rangle\langle\psi|\right)} =$$

$$\sum_a a_i \frac{Tr\left(\langle\phi a_i\rangle\langle a_i \psi\rangle\langle\psi|\phi\rangle\right)}{Tr\left(\langle\phi\psi\rangle\langle\psi\phi\rangle\right)} =$$

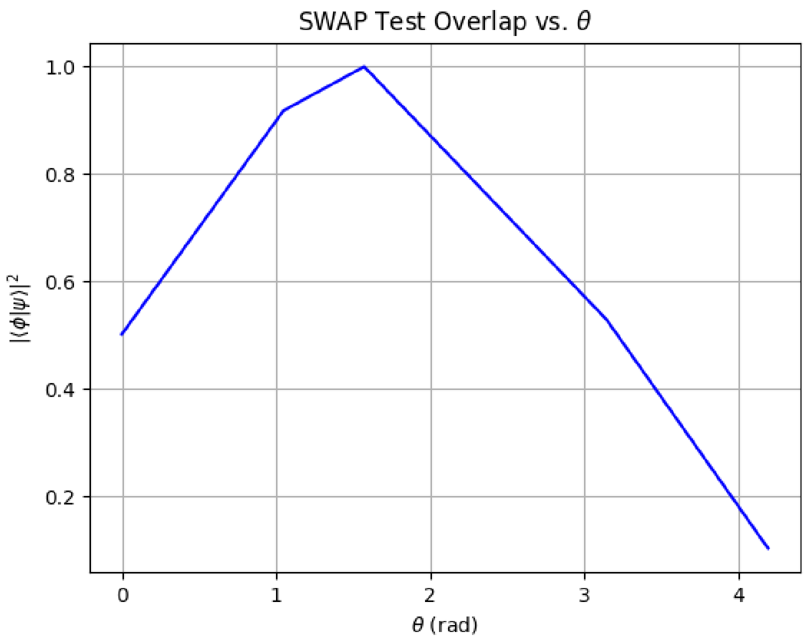
$$\sum_a a_i \frac{Tr\left(\langle\phi a_i\rangle\langle a_i \psi\rangle\right)}{Tr\left(\langle\phi\psi\rangle\right)} =$$

$$\sum_a \frac{\langle\phi|\left(\sum_a a_i\rangle\langle a_i\right)|\psi}{\langle\phi|\psi\rangle} =$$

$$\frac{\langle\phi|A|\psi\rangle}{\langle\phi|\psi\rangle}$$

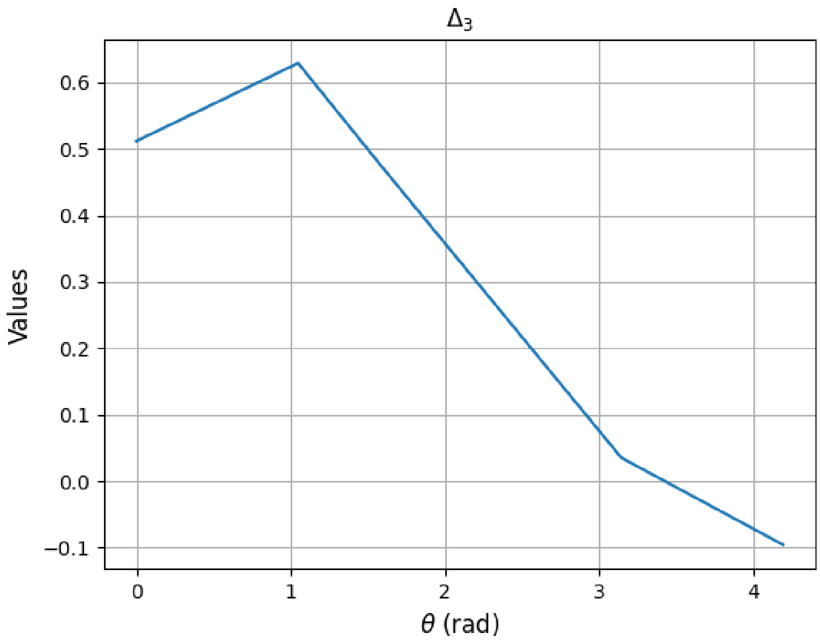
4.b

(i)



SWAP Test Overlap vs. θ
$|\langle\phi|\psi\rangle|^2$
1.0
0.8
0.6
0.4
0.2
0
1
2
3
4
θ (rad)

(ii)



Note that

$$\Delta_3(\Pi_\phi, |1\rangle\langle 1|\Pi_\psi)$$

is computed as: $\Delta_3(\Pi_\phi, |1\rangle\langle 1|\Pi_\psi) = \mathrm{Tr}(\Pi_\phi|1\rangle\langle 1|\Pi_\psi) = \mathrm{Tr}(\langle 1|\Pi_\psi\Pi_\phi|1\rangle) = \Pi_\psi\Pi_\phi(2,2) = \mathrm{Tr}(\Pi_\psi\Pi_\phi) - (\Pi_\psi\Pi_\phi)(1,1) = \Delta_2(\Pi_\psi\Pi_\phi) - \Delta_3(\Pi_\phi|0\rangle\langle 0|\Pi_\psi)$

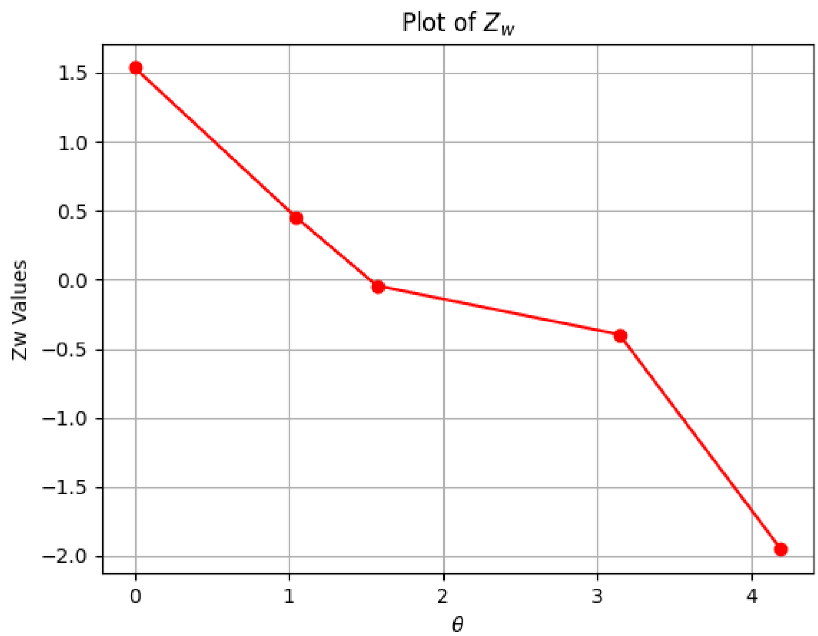
Since the right-hand side has already been calculated, we can now compute the left-hand side directly without any further measurements. ### Answer Recall that Z is defined as:

$$Z = 1 \cdot |0\rangle\langle 0| + (-1) \cdot |1\rangle\langle 1|$$

We will now use the identity proven in part 4(a), along with the previously computed values of Δ_2 and Δ_3, to compute Z_w:

$$Z_w = \frac{\Delta_3(\Pi_\phi|0\rangle\langle 0|\Pi_\psi)}{\Delta_2(\Pi_\phi\Pi_\psi)} - \frac{\Delta_3(\Pi_\phi|1\rangle\langle 1|\Pi_\psi)}{\Delta_2(\Pi_\phi\Pi_\psi)}$$

The result Z_w can be computed directly using the above equation.


Plot of Z_w
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Zw Values
0
1
2
3
4
θ