Name: Okoye Adunife Kizito
Student ID: 100611918
CSCI 4140U

CSCI 4140 Assignment One

GENERALIZING THE ALGORITHM

Modify both programs to have a parameter, n, which is the number of qubits.
Note, the number of classical bits will also depend on n

```
n = 4
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = 2*math.pi/3 #alter for exact or approx.
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_dagger(qpe3, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

ADDING NOISE

So far we have been dealing with perfect qubits, but this isn't the case in the real world. Existing quantum computers have noise, which impact gates and measurements.
Our first question is how seriously does noise effect our results?
In order to do this we need a noise model, which I provide for you.
This noise model is in the following procedure which is similar to one in the Qiskit textbook, modified for the purposes of our experiment.
This procedure is:

In [79]:
```
def get_noise(p_meas,p_gate):
    error_meas = pauli_error([('X',p_meas), ('I', 1 - p_meas)])
    error_gate1 = depolarizing_error(p_gate, 1)
    error_gate2 = error_gate1.tensor(error_gate1)
    noise_model = NoiseModel()
    noise_model.add_all_qubit_quantum_error(error_meas, "measure")
    noise_model.add_all_qubit_quantum_error(error_gate1, ["u1", "u2", "u3"])
    noise_model.add_all_qubit_quantum_error(error_gate2, ["cx", "cu1"])
    return noise_model
```

The first parameter to this procedure is the noise level for measurement. This will be left at 0.01 for our experiments. The second parameter is the gate noise level, which will be varied, but will start at 0.01.

This noise model is used in the following way:

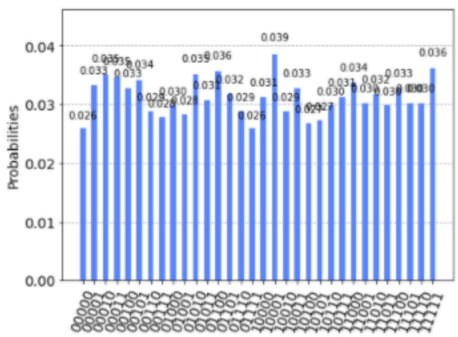
```
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

For the exact version of the algorithm when using “qft_dagger()”

In [85]:
```python
# Where n = 6
n = 6
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = math.pi/4
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_dagger(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

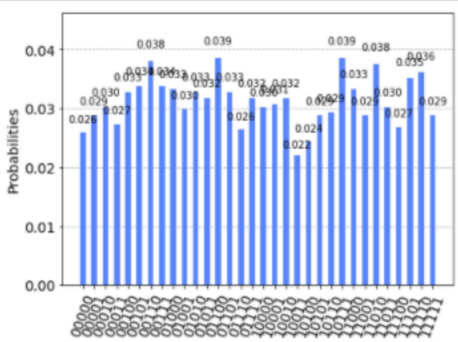
In [82]:
```python
# Where n = 6, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[82]:



In [83]:
```python
# Where n = 6, noise level = 0.05
noise_model = get_noise(0.05,0.05)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

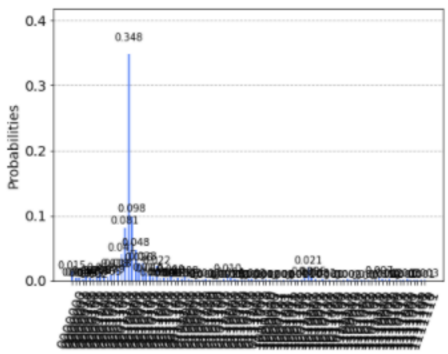
Out[83]:



In [87]:
```
# Where n = 8
n = 8
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = math.pi/4
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_dagger(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

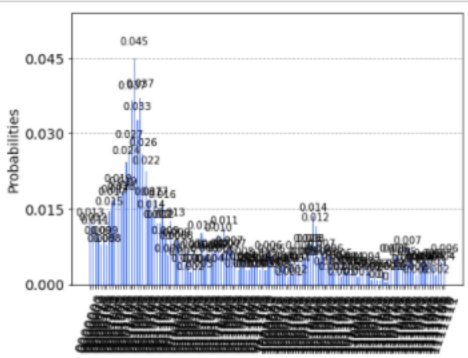
In [88]:
```
# Where n = 8, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[88]:



In [89]:
```
# Where n = 8, noise level = 0.05
noise_model = get_noise(0.05,0.05)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

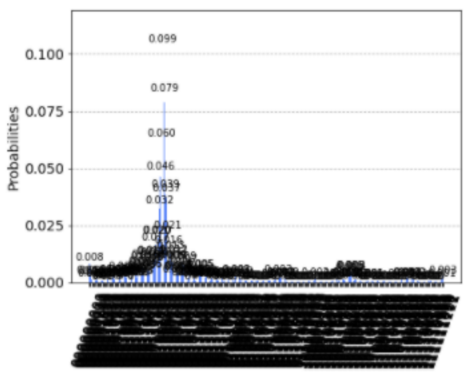
Out[89]:



In [90]:
```python
# Where n = 10
n = 10
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = math.pi/4
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_dagger(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

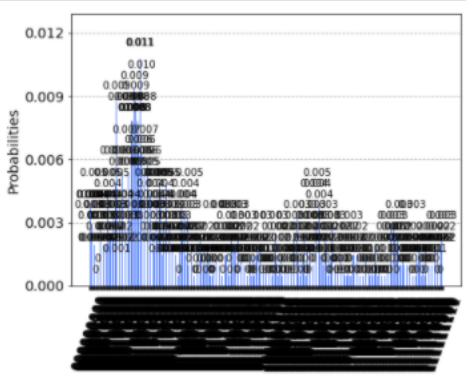
In [91]:
```python
# Where n = 8, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[91]:



In [92]:
```python
# Where n = 8, noise level = 0.05
noise_model = get_noise(0.05,0.05)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[92]:

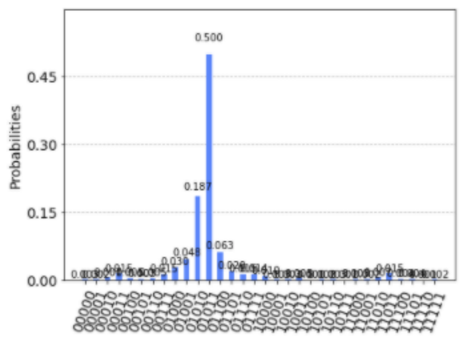


For the approximate version of the algorithm when using “qft_dagger()”

In [93]:
```
#Where n = 6
n = 6
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_dagger(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

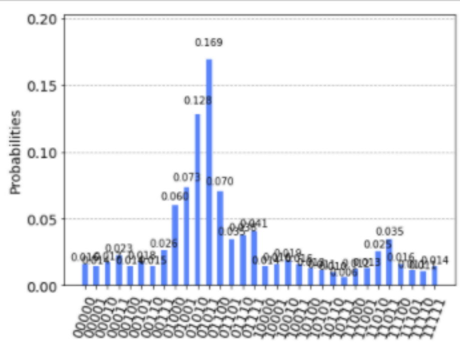
In [94]:
```
# Where n = 6, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[94]:



In [95]:
```
# Where n = 6, noise level = 0.05
noise_model = get_noise(0.05,0.05)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

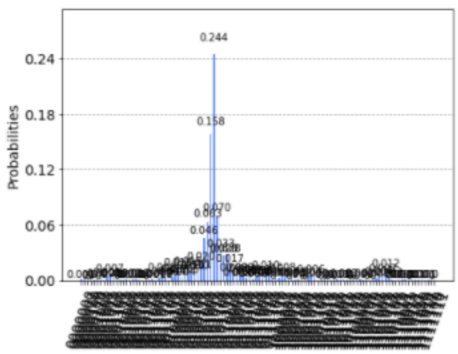
Out[95]:



In [96]:
```
#Where n = 8
n = 8
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_dagger(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

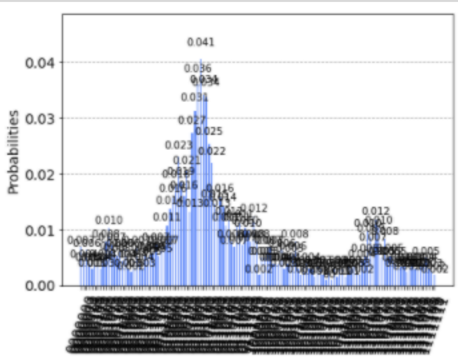
In [97]:
```
# Where n = 8, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[97]:



In [98]:
```
# Where n = 8, noise level = 0.05
noise_model = get_noise(0.05,0.05)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

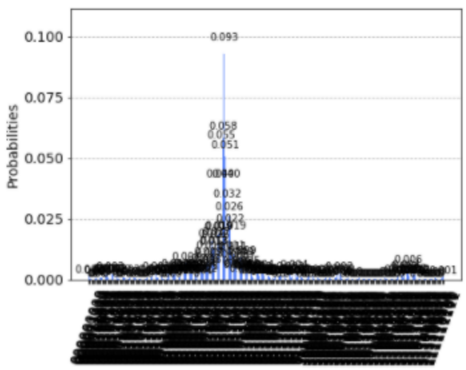
Out[98]:



In [99]:
```
#Where n = 10
n = 10
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_dagger(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

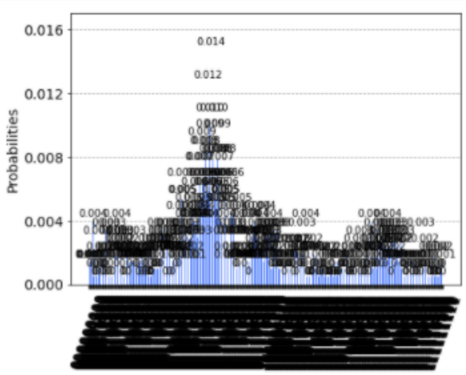
In [100]:
```
# Where n = 10, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[100]:



In [101]:
```
# Where n = 10, noise level = 0.05
noise_model = get_noise(0.05,0.05)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[101]:



For the exact version of the algorithm when using “qft_limited()”

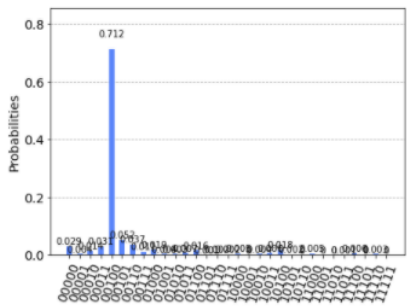
In [107]:

```
# Where n = 6
n = 6
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = math.pi/4
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_limited(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

In [108]:

```
# Where n = 6, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[108]:



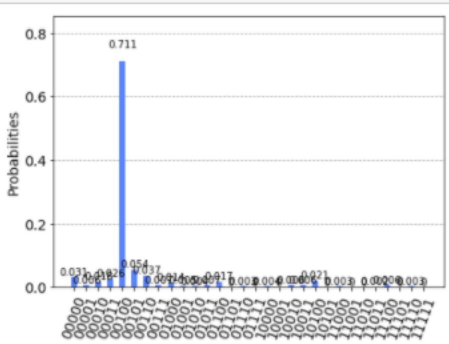
In [110]:

```
# Where n = 8
n = 6
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = math.pi/4
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_limited(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

In [111]:

```
# Where n = 8, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[111]:



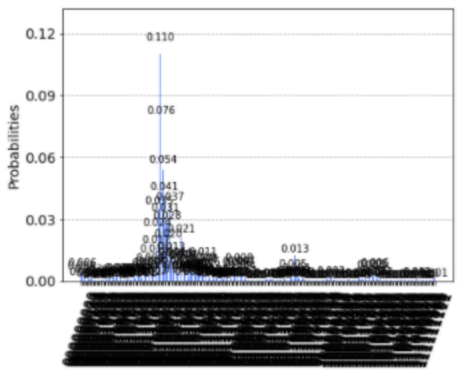
In [113]:

```python
# Where n = 10
n = 10
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = math.pi/4
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_limited(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

In [114]:

```python
# Where n = 10, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[114]:

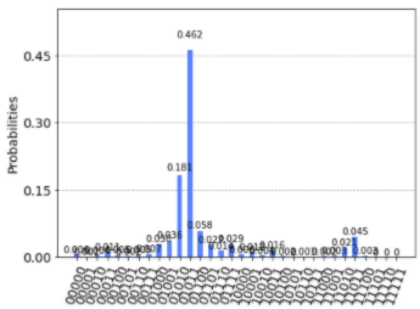


For the approximate version of the algorithm when using “qft_limited()*

In [116]:
```
# Where n = 6
n = 6
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_limited(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

In [117]:
```
# Where n = 6, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

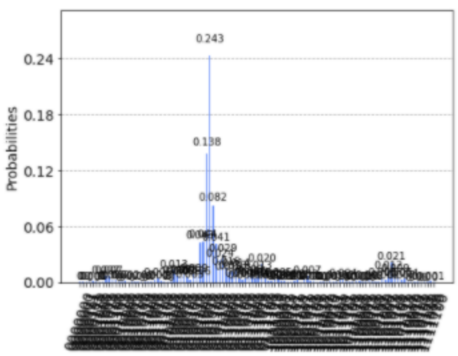
Out[117]:



In [119]:
```python
# Where n = 8
n = 8
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_limited(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

In [120]:
```python
# Where n = 8, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

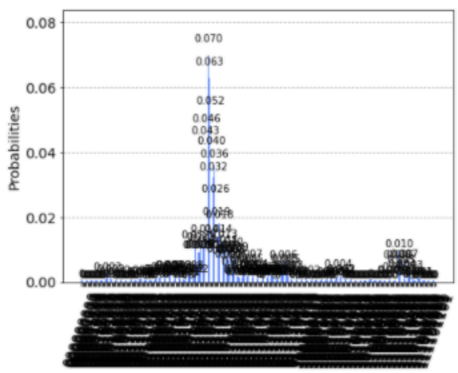
Out[120]:



In [122]:
```python
# Where n = 10
n = 10
qbit = n
cqbit = n-1
qpen = QuantumCircuit(qbit,cqbit)

for qubit in range(cqbit):
    qpen.h(qubit)
qpen.x(cqbit)

angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(cqbit):
    for  i in range(repetitions):
        qpen.cu1(angle, counting_qubit, cqbit);
    repetitions *= 2
qft_limited(qpen, cqbit)
for n in range(cqbit):
    qpen.measure(n,n)
```

In [123]:
```python
# Where n = 10, noise level = 0.01
noise_model = get_noise(0.01,0.01)


backend = Aer.get_backend('qasm_simulator')
shots = 2048
results = execute(qpen, backend=backend, shots=shots, noise_model=noise_model).result()
answer = results.get_counts()
plot_histogram(answer)
```

Out[123]:



In conclusion, when using the approximate version I noticed there seemed to be and extreme outlier and that seem to prevail throughout the experiment.