Practical 11

Name: Heja Bibani
Student Number: 16301173

In [1]:
```
import math
import sys
```

Section 1

Suppose the two integers a and N satisfy that a < N and they are co-prime. Use Python code to solve the following tasks.

1.1

Implement a Python function that takes these two integers as parameters and prints out all the values of $f_{a,N}(x) = a^x mod N$, where x ranges from 0 to r, the period of $f_{a,N}(x)$. Specifically, the output should consist of $r+1$ lines, with each line containing the value of x and the corresponding value of $f_{a,N}(x)$.

In [2]:
```
def func(a, N):
    if a >= N:
        sys.exit("Error: 'a' must be less than 'N'!")
    if math.gcd(a, N) != 1:
        sys.exit("Error: 'a' and 'N' must be co-prime!")
    # 'r' is the period to return
    # Let's start with 2 since a<N
    r = 2
    predecessor = a
    remainder = predecessor * a % N
    while remainder != 1:
        predecessor = remainder
        remainder = predecessor * a % N
        r += 1
    # Initialization
    predecessor = 1
    remainder = 1
    # Repeat until reaching the value of x
    for i in range(r+1):
        if(i == 0):
            print( str(0) + ": "  + str(1 % N))
        else:
            remainder = predecessor * a % N
            predecessor = remainder
            print( str(i) + ": "  + str(remainder))
```

1.2

1.2 Use the above function to output the result for $a = 21$ and $N = 391$.

In [3]:
```
func(21, 391)
```

```
0: 1
1: 21
2: 50
3: 268
4: 154
5: 106
6: 271
7: 217
8: 256
9: 293
10: 288
11: 183
12: 324
13: 157
14: 169
15: 30
16: 239
17: 327
18: 220
19: 319
20: 52
21: 310
22: 254
23: 251
24: 188
25: 38
26: 16
27: 336
28: 18
29: 378
30: 118
31: 132
32: 35
33: 344
34: 186
35: 387
36: 307
37: 191
38: 101
39: 166
40: 358
41: 89
42: 305
43: 149
44: 1
```

In [4]:
```
def period(a, N):
    if a >= N:
        sys.exit("Error: 'a' must be less than 'N'!")
    if math.gcd(a, N) != 1:
        sys.exit("Error: 'a' and 'N' must be co-prime!")

    # 'r' is the period to return
    # Let's start with 2 since a<N
    r = 2
    predecessor = a
    remainder = predecessor * a % N
    while remainder != 1:
        predecessor = remainder
        remainder = predecessor * a % N
        r += 1

    return r
```

In [5]:
```
print(period(21,391))
```

```
44
```

We see that r = 44 and is even. Next we test if $21^{22} + 1$ is a multiple of 391. We can test this by checking if $(21^{22} + 1)$ mod 391 = 0.

In [6]:
```
# Given three integers: a, x, and N,
# Calcualte the Modular Exponentiation: a ** x % N
def modular_power(a, x, N):
    # Initialization
    predecessor = 1
    remainder = 1

    # Repeat until reaching the value of x
    for i in range(x):
        remainder = predecessor * a % N
        predecessor = remainder

    return remainder
```

In [7]:
```
print(modular_power(21,22,391))
```

```
254
```

We simply add 1 and it equals 255. It is not a multiple of 391. And r is even.

1.4

Calculate gcd($21^{\frac{r}{2}}$+1,391) and gcd($21^{\frac{r}{2}}$−1,391). Verify that the product of these two resulting numbers equals 391.

In [8]:
```
gcd1 = math.gcd(255,391)
gcd1
```

Out[8]:
```
17
```

In [9]:
```
gcd2 = math.gcd(253,391)
gcd2
```

Out[9]:
```
23
```

In [10]:
```
value_prod = gcd1*gcd2
value_prod == 391
```

Out[10]:
```
True
```

We see that the value equals 391 as expected.

Section 2

Consider the 8×8 Vandermonde Matrix as defined in the lecture slides.

2.1

Use manual calculation to give all entries in its second row.

$\omega_8^0 = e^{\frac{i2\pi*0}{8}} = 1$

$\omega_8^1 = e^{\frac{i2\pi*1}{8}} = \frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}$

$\omega_8^2 = e^{\frac{i2\pi*2}{8}} = i$

$\omega_8^3 = e^{\frac{i2\pi*3}{8}} = -\frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}$

$\omega_8^4 = e^{\frac{i2\pi*4}{8}} = -1$

$\omega_8^5 = e^{\frac{i2\pi*5}{8}} = -\frac{1}{\sqrt{2}} - \frac{i}{\sqrt{2}}$

$\omega_8^6 = e^{\frac{i2\pi*6}{8}} = -i$

$\omega_8^7 = e^{\frac{i2\pi*7}{8}} = \frac{1}{\sqrt{2}} - \frac{i}{\sqrt{2}}$

2.2

Use manual calculation to give all entries in its third row.

$\omega_8^0 = e^{\frac{i2\pi*0}{8}} = 1$

$\omega_8^2 = e^{\frac{i2\pi*2}{8}} = i$

$\omega_8^4 = e^{\frac{i2\pi*4}{8}} = -1$

$\omega_8^6 = e^{\frac{i2\pi*6}{8}} = -i$

$\omega_8^8 = e^{\frac{i2\pi*8}{8}} = 1$

$\omega_8^{10} = e^{\frac{i2\pi*10}{8}} = i$

$\omega_8^{12} = e^{\frac{i2\pi*12}{8}} = -1$

$\omega_8^{14} = e^{\frac{i2\pi*14}{8}} = -i$

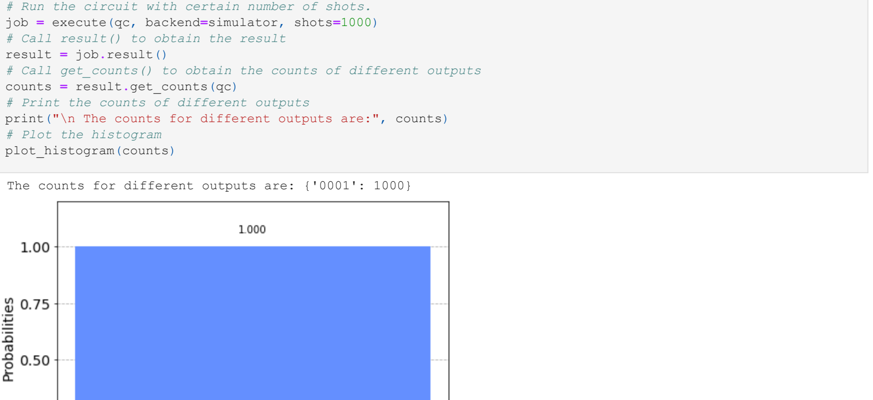
Section 3

Suppose we have a signal S, which takes the form of $cos(\theta) + isin(\theta)$, with θ rotating continuously from θ to -2π in a constant speed. Let's assume that, in a certain time interval, S rotates from 0 to -2π exactly once. Thus, S has the frequency of 1 in this time interval. To detect this frequency, we sample 16 values from this signal during that time interval. The samples happen on θ=0,$-\frac{\pi}{8}, \ldots, -\frac{15\pi}{8}$, with a step of $-\frac{\pi}{8}$. You should roughly follow the notebook accompanying Lecture 11 to implement a circuit with QFT to detect the frequency of the signal S by processing these sample values.

In [2]:
```
from qiskit import *
from qiskit.providers.aer import QasmSimulator
from qiskit.visualization import plot_histogram
from qiskit.circuit.library import QFT
import numpy as np
%matplotlib inline

S = []
for freq in range(1):
    for theta in [0, -np.pi/8, -np.pi*2/8, -np.pi*3/8, -np.pi*4/8, -np.pi*5/8, -np.pi*6/8, -np.pi*7/8, -np.pi*
        value = (np.cos(theta) + np.sin(theta)*1j) / 4
        S.append(value)
for sample in S:
    print( format(sample, ".3f") )

qc = QuantumCircuit(4, 4)


# Use the vector state in S1 to initialize the four qubits
qc.initialize(S, [0, 1, 2, 3])
qc = qc.compose(QFT(4))
qc.measure([0, 1, 2, 3], [0, 1, 2, 3])
qc.draw()
```

```
0.250+0.000j
0.231-0.096j
0.177-0.177j
0.096-0.231j
0.000-0.250j
-0.096-0.231j
-0.177-0.177j
-0.231-0.096j
-0.250-0.000j
-0.231+0.096j
-0.177+0.177j
-0.096+0.231j
-0.000+0.250j
0.096+0.231j
0.177+0.177j
0.231+0.096j
```

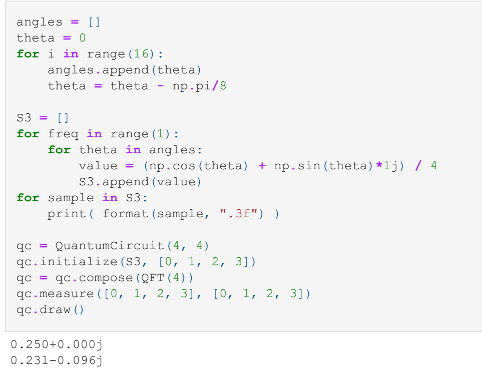
Out[2]:



In [81]:
```
simulator = QasmSimulator()
# Run the circuit with certain number of shots.
job = execute(qc, backend=simulator, shots=1000)
# Call result() to obtain the result
result = job.result()
# Call get_counts() to obtain the counts of different outputs
counts = result.get_counts(qc)
# Print the counts of different outputs
print("\n The counts for different outputs are:", counts)
# Plot the histogram
plot_histogram(counts)
```

```
 The counts for different outputs are: {'0001': 1000}
```

Out[81]:



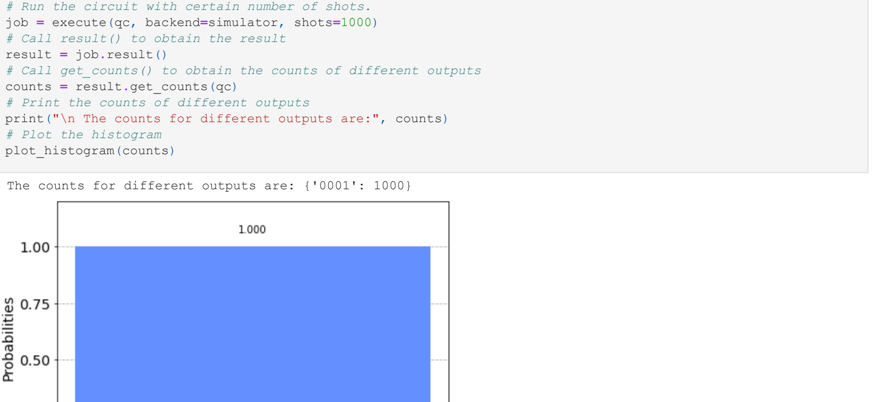
Thus the value is 0001 which is the value of 1 in decimal revealing the frequency of S.

The alternative method would be to do it this way, which would make the code more compact. As we can see the decimal value is the same.

In [96]:
```
angles = []
theta = 0
for i in range(16):
    angles.append(theta)
    theta = theta - np.pi/8

S3 = []
for freq in range(1):
    for theta in angles:
        value = (np.cos(theta) + np.sin(theta)*1j) / 4
        S3.append(value)
for sample in S3:
    print( format(sample, ".3f") )

qc = QuantumCircuit(4, 4)
qc.initialize(S3, [0, 1, 2, 3])
qc = qc.compose(QFT(4))
qc.measure([0, 1, 2, 3], [0, 1, 2, 3])
qc.draw()
```

```
0.250+0.000j
0.231-0.096j
0.177-0.177j
0.096-0.231j
0.000-0.250j
-0.096-0.231j
-0.177-0.177j
-0.231-0.096j
-0.250-0.000j
-0.231+0.096j
-0.177+0.177j
-0.096+0.231j
-0.000+0.250j
0.096+0.231j
0.177+0.177j
0.231+0.096j
```

Out[96]:



In [97]:
```
simulator = QasmSimulator()
# Run the circuit with certain number of shots.
job = execute(qc, backend=simulator, shots=1000)
# Call result() to obtain the result
result = job.result()
# Call get_counts() to obtain the counts of different outputs
counts = result.get_counts(qc)
# Print the counts of different outputs
print("\n The counts for different outputs are:", counts)
# Plot the histogram
plot_histogram(counts)
```

```
 The counts for different outputs are: {'0001': 1000}
```

Out[97]:

Section 4

Follow the notebook accompanying Lecture 11 to implement a circuit with Shor class to factor the integer 21.

4.1

Try $a = 4$ and show it's not successful. Explain this by manually following Shor's algorithm.

In [3]:
```
from qiskit import *
from qiskit.providers.aer import QasmSimulator
from qiskit.utils import QuantumInstance
from qiskit.algorithms import Shor

simulator = QasmSimulator()
q_instance = QuantumInstance(simulator, shots=1000)
my_shor = Shor(q_instance)

# Run Shor's algorithm with N=21, a=4
result = my_shor.factor(21, 4)
print(result)
```

```
{'factors': [], 'successful_counts': 0, 'total_counts': 38}
```

We can see in the above that successful counts = 0. If we manually do this. We have to:

1. Find out the period
2. Check if $a^{\frac{r}{2}} + 1$ is a mutiple of N or r is odd.

In [87]:
```
period(4,21)
```

Out[87]:
```
3
```

We note above that the period is odd. Therefore it must not work and we must pick another a. In the next question 4.2 we will use a = 2 as the value and see that this will work.

4.2

Try $a = 2$ and show it is successful.

In [95]:
```
from qiskit import *
from qiskit.providers.aer import QasmSimulator
from qiskit.utils import QuantumInstance
from qiskit.algorithms import Shor

simulator = QasmSimulator()
q_instance = QuantumInstance(simulator, shots=1000)
my_shor = Shor(q_instance)

# Run Shor's algorithm with N=21, a=2
result = my_shor.factor(21, 2)
print(result)
```

```
{'factors': [[3, 7]], 'successful_counts': 24, 'total_counts': 53}
```

We can see in the above that successful counts = 0. If we manually do this. We have to:

1. Find out the period
2. Check if $a^{\frac{r}{2}} + 1$ is a mutiple of N or r is odd.
3. calculate the gcd($a^{\frac{r}{2}} + 1$, 21) and gcd($a^{\frac{r}{2}} - 1$, 21)

In [88]:
```
period(2,21)
```

Out[88]:
```
6
```

In [33]:
```
a_r_2add1 = pow(2,3)+1
a_r_2add1
```

Out[33]:
```
9
```

R is even and 9 is not a multiple of 21.

In [89]:
```
gcd_1 = math.gcd(9,21)
gcd_1
```

Out[89]:
```
3
```

In [90]:
```
gcd_2 = math.gcd(7,21)
gcd_2
```

Out[90]:
```
7
```

In [91]:
```
gcd_1*gcd_2
```

Out[91]:
```
21
```

This equals the value of N as expected and now we have got the value of p and q which is the same as the shor's algorithm.