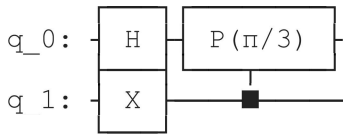
In [1]:

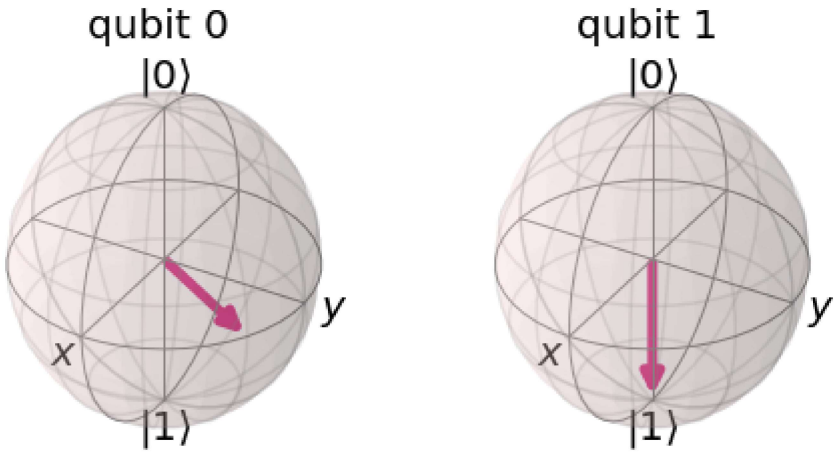
```
from qiskit import *
from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector

from qiskit.tools.monitor import job_monitor
from qiskit.providers.ibmq import least_busy

import matplotlib
import numpy

%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:

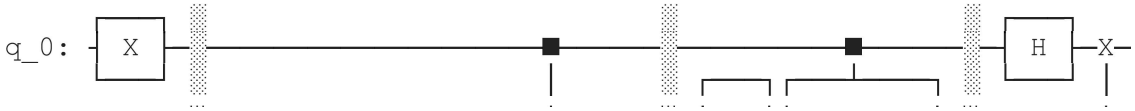
```
qc = QuantumCircuit(2);
qc.h(0);
qc.x(1);
qc.cp(pi/3, 1, 0);
display(qc.draw());

svsim = Aer.get_backend('statevector_simulator');
qobj = assemble(qc);
state = svsim.run(qobj).result().get_statevector()

display(plot_bloch_multivector(state))
```

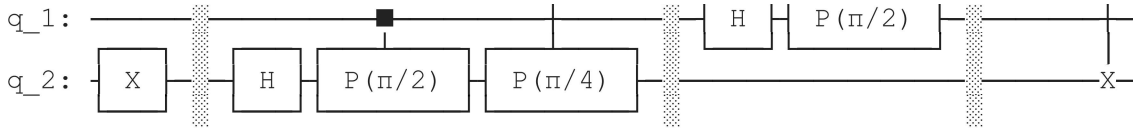


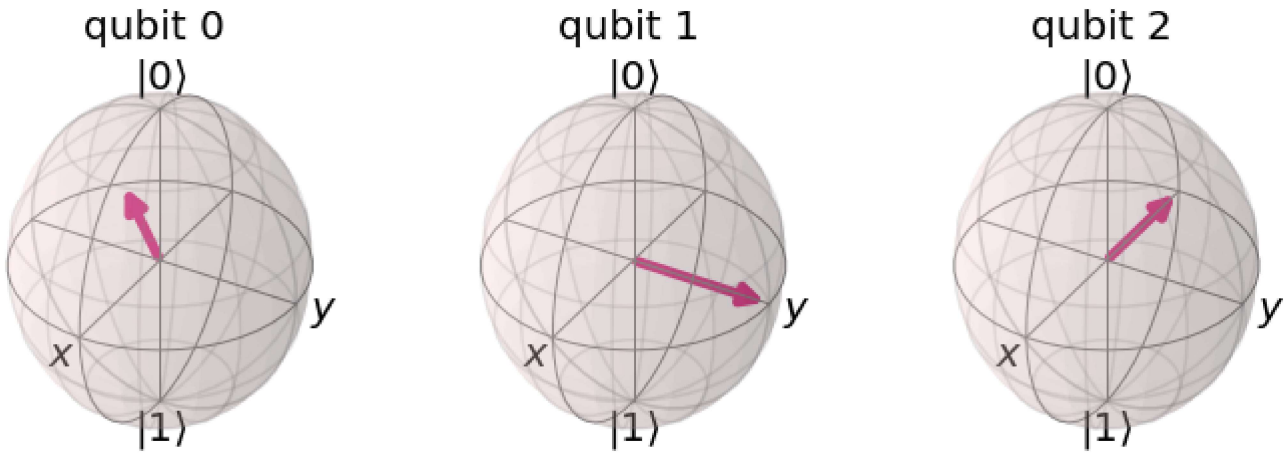


In [8]:

```
qc = QuantumCircuit(3)

qc.x([0, 2]);
qc.barrier()
qc.h(2)
qc.cp(pi/2, 1, 2)

qc.cp(pi/4, 0, 2) # CROT from qubit 2 to qubit 0
qc.barrier()

qc.h(1)
qc.cp(pi/2, 0, 1) # CROT from qubit 0 to qubit 1
qc.barrier()

qc.h(0)
qc.swap(0,2)

display(qc.draw())

svsim = Aer.get_backend('statevector_simulator')
qobj = assemble(qc)
final_state = svsim.run(qobj).result().get_statevector()

display(plot_bloch_multivector(final_state))
```



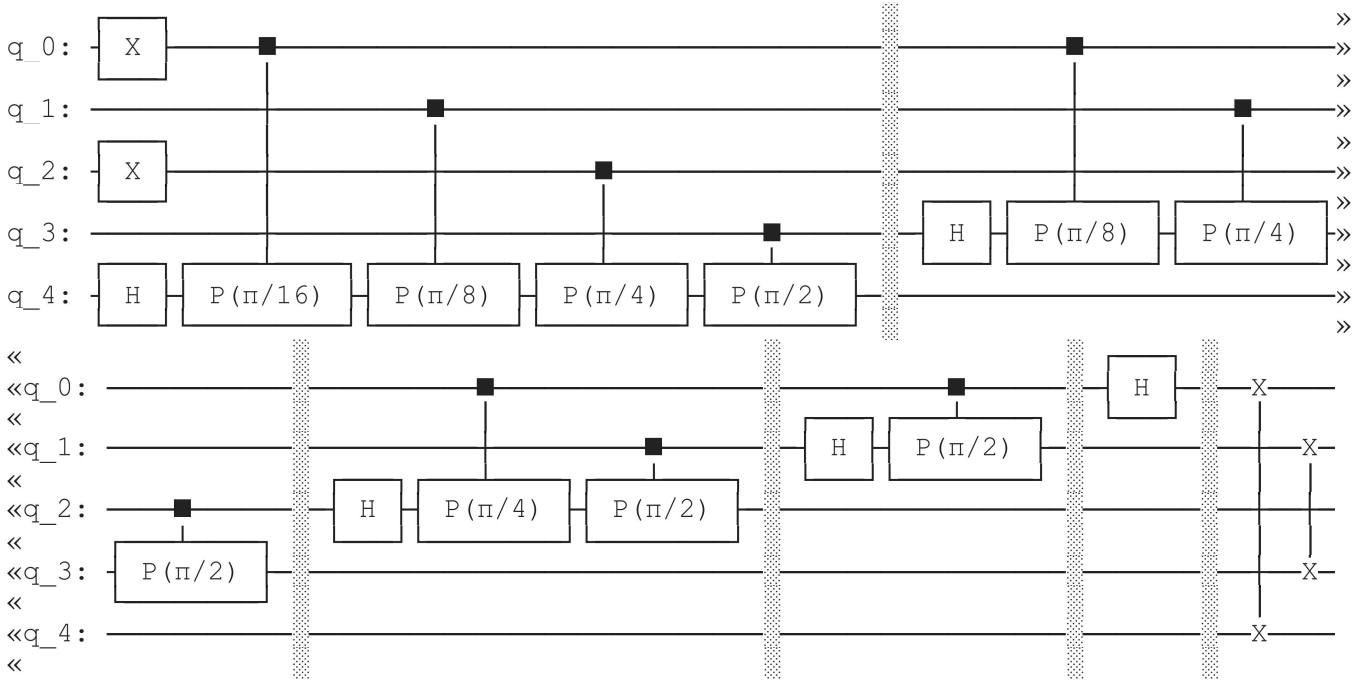


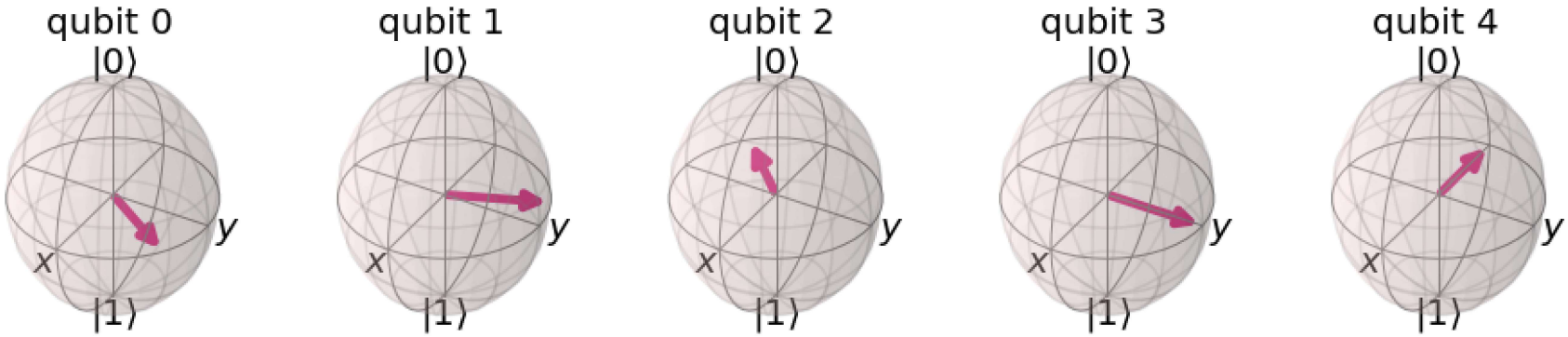


In [23]:

```
def qft_rotations(qc, n):
    if n == 0:
        return qc;

    n = n-1;
    qc.h(n);
    for qubit in arange(0, n):
        qc.cp(pi/(2**(n - qubit)), qubit, n);
    qc.barrier()
    qft_rotations(qc, n);

def swap_registers(qc, n):
    for qubit in arange(0, n//2):
        qc.swap(qubit, n-qubit-1);

n = 5;

qc = QuantumCircuit(n);
qc.x([0, 2])
qft_rotations(qc, n);
swap_registers(qc, n)
display(qc.draw())

svsim = Aer.get_backend('statevector_simulator')
```

```
qobj = assemble(qc)
final_state = svsim.run(qobj).result().get_statevector()

display(plot_bloch_multivector(final_state))
```





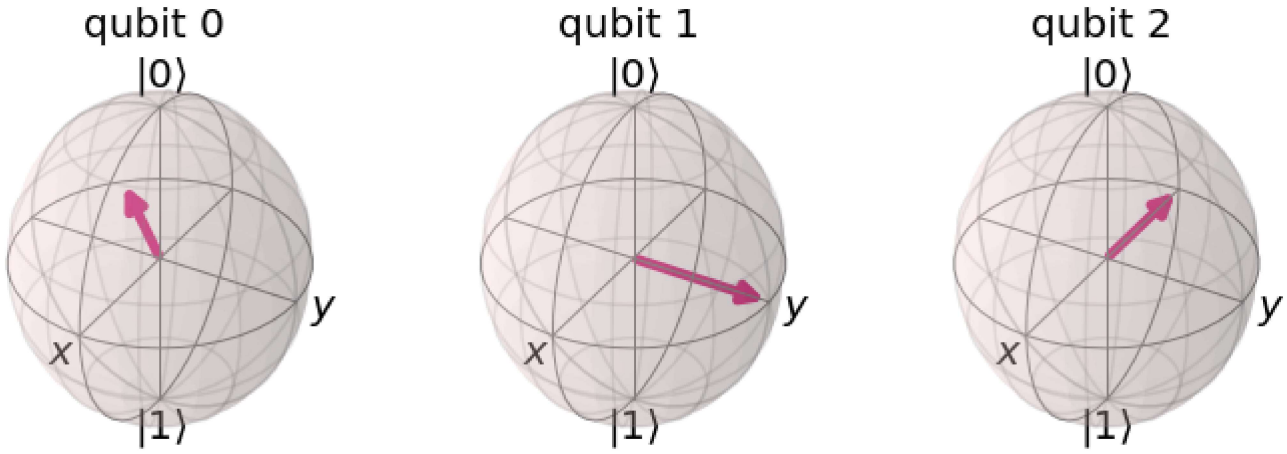
In [45]:

```
def qft(qc, n):
    qft_rotations(qc, n);
    swap_registers(qc, n);
```

```
    return qc;

def inverse_qft(qc, n):
    qft_cir = qft(QuantumCircuit(n), n);
    invqft_cir = qft_cir.inverse();
    qc.append(invqft_cir, qc.qubits[:n]);
    return qc.decompose()
```

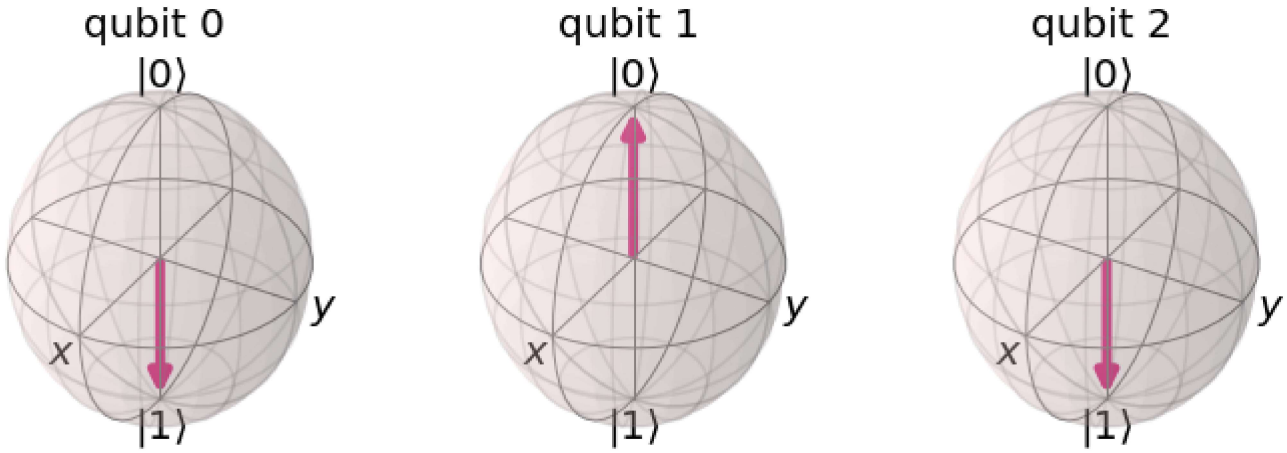
In [48]:

```
n = 3;
qc = QuantumCircuit(n)
qc.x([0, 2])

qc = qft(qc,n)

qobj = assemble(qc)
statevector = svsim.run(qobj).result().get_statevector()
display(plot_bloch_multivector(statevector))
```



In [49]:

```
qc2 = inverse_qft(qc, n);
qobj = assemble(qc2)
statevector = svsim.run(qobj).result().get_statevector()
display(plot_bloch_multivector(statevector))
```



In [43]:

In [ ]: