Teleportation

```
!pip install qiskit
!pip install qiskit_aer
from qiskit import QuantumCircuit
from qiskit.quantum_info import
Statevector,partial_trace
from qiskit_aer import Aer
from math import sqrt
from qiskit.visualization import plot_histogram
qc = QuantumCircuit(3,2)
# state that want to teleport
state = [1/sqrt(2),1/sqrt(2)]
qc.initialize(state,0)
qc.h(1)
qc.cx(1,2)
qc.draw()
state = Statevector(qc)
state.draw('latex') # MSB is ist
qc.cx(0,1)
qc.h(0)
```

```
qc.measure(0,0)
qc.measure(1,1)
backend = Aer.get_backend('aer_simulator')
result = backend.run(qc,shots=1).result()
measurment_result =
list(result.get_counts().keys())[0][::-1] # reversing for
readability
measurment_result
if measurment_result == '11':
 qc.x(2)
 qc.z(2)
elif measurment_result == '10':
 qc.z(2)
elif measurment_result == '01':
 qc.x(2)
backend = Aer.get_backend('statevector_simulator')
result = backend.run(qc).result()
state = result.get_statevector()
state.draw('latex') # msb ist , so teleported qubit ist.
```

Superdense Coding

```
from qiskit import QuantumCircuit
from qiskit.visualization import *
from qiskit_aer import Aer
qc = QuantumCircuit(2,2)
qc.h(0)
qc.cx(0, 1)
qc.draw()
bit = '01'
if bit=='00':
  pass
elif bit=='01':
  qc.x(0)
elif bit=='10':
  qc.z(0)
elif bit=='11':
  qc.x(0)
  qc.z(0)
# Bob decodes
qc.cx(0,1)
qc.h(0)
```

```
qc.measure(0,0)
qc.measure(1,1)
sim = Aer.get_backend('aer_simulator')
result = sim.run(qc,shots=1).result()
counts = result.get_counts()
print(counts)
```

Deutsch-Jozsa Algorithm

```
! pip install qiskit_aer
def constant_oracle(n):
    """
    when f(x) = 0 or 1 (constant) , then for input xy ,
    output wil be xy (same)
    """
    oracle = QuantumCircuit(n + 1)
    # oracle.x(n)  # for f(x)=1
    return oracle


def balanced_oracle(n):
    """
    when f(x) is balanced , then outputs will half
opposite,
    so just applying operatios y xor f(x) , for all x
    """
    oracle = QuantumCircuit(n + 1)
    for i in range(n):
        oracle.cx(i, n)
    return oracle
```

```
from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt

def deutsch_jozsa(f_oracle, n):
    # n+1 bits(input) and n for classical bit(output)
    qc = QuantumCircuit(n + 1, n)

    # Step 1: Setting up y
    qc.x(n)
    qc.h(n)

    # Step 2: Hadamard transform to input x
    for i in range(n):
        qc.h(i)
    qc.barrier()
    # Step 3: Append the oraclef f(x)
    qc = qc.compose(f_oracle, qubits=range(n + 1))
    qc.barrier()
```

```
    # Step 4: Apply Hadamard again to input qubits
    for i in range(n):
        qc.h(i)
    # Step 5: Measure input qubits
    for i in range(n):
        qc.measure(i, i)
    return qc
# Choose number of input bits
n = 3
# Preparing Uf
oracle = constant_oracle(n)
dj_circuit = deutsch_jozsa(oracle, n)
dj_circuit.draw()
sim = Aer.get_backend('aer_simulator')
result = sim.run(dj_circuit).result()
f = result.get_counts()
f
```

Shor's Algo

Shor's Algorithm

```
import numpy as np
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
import math
def qft_dagger(n):
 circuit = QuantumCircuit(n)
 for j in range(n // 2):
 circuit.swap(j, n - j - 1)
 for j in range(n):
 for k in range(j):
 circuit.cp(-np.pi / float(2 ** (j - k)), k, j)
 circuit.h(j)
 return circuit
def modular_exponentiation(n_count, a, N):
 circuit = QuantumCircuit(n_count + 4, n_count)
 for x in range(n_count):
 for q in range(4):
```

```
 power = (2 ** x) % N
 if (a ** power % N) & (1 << q):
 circuit.cx(x, n_count + q)
 return circuit
def run_shor(N=15, a=7):
 print(f"\n Shor's Algorithm: Factoring N={N} with
a={a}\n")
 if N % 2 == 0:
 print("N is even, factor: 2")
 return
 if int(N ** (1 / float(2))) ** 2 == N:
 print(f"N is a square, factor: {int(N ** (1 / float(2)))}")
n_count = 4
 circuit = QuantumCircuit(n_count + 4, n_count)
 circuit.h(range(n_count))
 oracle = modular_exponentiation(n_count, a, N)
 circuit.append(oracle, range(n_count + 4))
 qft = qft_dagger(n_count)
 circuit.append(qft, range(n_count))
 circuit.measure(range(n_count), range(n_count))
```

```
simulator = Aer.get_backend('qasm_simulator')
result = simulator.run(transpile(circuit, simulator),
shots=1000).result()
counts = result.get_counts()
print("Measurement Results:", counts)
factors = set()
for measured in counts:
measured_int = int(measured, 2)
if measured_int == 0:
continue
fraction = measured_int / (2 ** n_count)
denominators = []
x = fraction
for _ in range(10):
a = int(1 / x) if x != 0 else 0
denominators.append(a)
x = 1 / x - a if x != 0 else 0
if x < 1e-6:
break
r = 1
```

```
for d in denominators:
r = 1 / (d + 1 / r) if r != 0 else d
r = int(round(1 / r))
if r % 2 == 0 and r > 0:
factor1 = math.gcd(a ** (r // 2) - 1, N)
factor2 = math.gcd(a ** (r // 2) + 1, N)
if 1 < factor1 < N:
factors.add(factor1)
if 1 < factor2 < N:
factors.add(factor2)
if factors:
print(f"Factors of {N}: {sorted(list(factors))}")
else:
print("No non-trivial factors found. Try a different a.")
if _name_ == "_main_":
run_shor()
```

Grover's Algo

```
! pip install qiskit_aer
from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.circuit.library import MCXGate
from qiskit.visualization import plot_histogram ,
plot_bloch_multivector
import numpy as np
def oracle(n,x):
  qc = QuantumCircuit(n)
  s = []
  for ind , i in enumerate(x):  # x is from least to
mostsignificant
    if i == '0':
      s.append(ind)
      qc.x(ind)
  # multicontrolled controlled z
  qc.h(n-1)
  qc.mcx(list(range(n-1)),n-1)
  qc.h(n-1)
```

```
  if s:
    for i in s:
        qc.x(i)
  return qc

def diffusion(n):
  qc = QuantumCircuit(n,n)
  qc.h(range(n))
  qc.x(range(n))
  # multicontrolled z gate
  qc.h(n-1)
  qc.mcx(list(range(n-1)),n-1)
  qc.h(n-1)
  qc.x(range(n))
  qc.h(range(n))
  return qc
def grovers_algo(n,x,iterations=1,m=1):
  if iterations == 'auto':
    iterations = int(np.floor((np.pi/4) * np.sqrt(2**n/m)
))

  qc = QuantumCircuit(n,n)
```

```
  qc.h(range(n))
  # two operations of Grover's operation
  for _ in range(iterations):
    print("iteration")
    qc.barrier(label='iter start')
    # ist  Uf (sign flip) assuming 2 bit input and x' = 11
    qc = qc.compose(oracle(n,x))
    qc.barrier()

    # 2nd W (Diffusion Operation)
    qc = qc.compose(diffusion(n))
    qc.barrier(label='iter end')
  qc.measure(range(n),range(n))
  return qc
qc = grovers_algo(5,x='10111',iterations='auto')
qc.draw()
sim = Aer.get_backend('aer_simulator')
result = sim.run(qc).result()
plot_histogram(result.get_counts()) # x is most
significant to least significant
```

BB84

```
from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
from copy import deepcopy
from numpy import random
import numpy as np
def basis(index):
  basis=[]
  for i in index:
    if i == 0:
      basis.append('Z')
    else:
      basis.append('X')
  return basis
def prep_qubit(bits,basises,n):
  qc = QuantumCircuit(n,n)
  for ind , (bit , basis) in enumerate(zip(bits,basises)):
    if basis == 'Z':
      if bit == 1:
        qc.x(ind)
```

```
    elif basis == 'X':
      if bit == 0:
        qc.h(ind)
      elif bit == 1:
        qc.h(ind)
        qc.z(ind)
  return qc
def measure_qubit(o_basis , qc , eve=False):
  for ind , basis in enumerate(o_basis):
    if basis == 'Z':
      qc.measure(ind,ind)
    elif basis == 'X':
      qc.h(ind)
      qc.measure(ind,ind)
  sim = Aer.get_backend('aer_simulator')
  result = sim.run(qc,shots=1).result()
  results = result.get_counts().keys()
  return list(results)[0][::-1]  # because qiskt writes msb
first
def bb84_sender(n):
  alice_bits = random.randint(2,size=n)
```

```
  alice_basis_ind = random.randint(2,size=n)
  alice_basis =  basis(alice_basis_ind)
  ciper_qubits = prep_qubit(alice_bits,alice_basis,n)
  return ciper_qubits , alice_basis , alice_bits
ciper_qubits ,alice_basis , alice_bits= bb84_sender(12)
def bb84_eve(n):
  eve_basis_ind = random.randint(2,size=n)
  eve_basis = basis(eve_basis_ind)
  data =
measure_qubit(eve_basis,ciper_qubits.copy(),eve=True
)
  return data , eve_basis
eve_bits , eve_basis = bb84_eve(12)
# making eve_bits string to array of int
eve_recieved_bits = []
for d in data:
  eve_recieved_bits.append(int(d))
eve_recieved_bits = np.array(eve_recieved_bits)
# ciper_qubits =
prep_qubit(eve_recieved_bits,eve_basis,12)  #uncomm
ent to include eve intercept
def bb84_reciever(n):
```

```
  bob_basis_ind = random.randint(2,size=n)
  bob_basis = basis(bob_basis_ind)
  data = measure_qubit(bob_basis,ciper_qubits)
  return data , bob_basis
data , bob_basis = bb84_reciever(12)
# making data string to array of int
bob_recieved_bits = []
for d in data:
  bob_recieved_bits.append(int(d))
bob_recieved_bits = np.array(bob_recieved_bits)
alice_bits
bob_recieved_bits
alice_basis = np.array(alice_basis)
bob_basis = np.array(bob_basis)
no = []
for ind ,(al,bo) in
enumerate(zip(alice_basis,bob_basis)):
  if al == bo:
    no.append(ind)
for i in no:
  print(alice_bits[i] , bob_recieved_bits[i])
```