Michał Kałmucki 151944

Grover's Algorithm Implementation

Helper functions and constants

```
import math
import numpy as np
from math import sqrt, pi, floor
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, transpile
from qiskit.quantum_info import Operator
from qiskit.circuit.library import XGate
from qiskit.visualization import plot_histogram, plot_distribution
from qiskit_aer import Aer
from time import process_time
import matplotlib.pyplot as plt

# ------------------------------
# Parameters
# ------------------------------
student_id = 151944
shots = 1024
backend_sim = Aer.get_backend('qasm_simulator')

def create_oracle(n, a):
    oracle = np.identity(2**n)
    oracle[a, a] = -1
    Uf = Operator(oracle)
    return Uf

def grover_diffusion_circuit(q, n):
    # Multi-controlled X gate
    mccx = XGate().control(n-1)
    diffusion = QuantumCircuit(n)
    # Apply diffusion operator
    for i in range(n):
        diffusion.h(i)
        diffusion.x(i)
    diffusion.h(n-1)
    diffusion.append(mccx, range(n))
    diffusion.h(n-1)
    for i in range(n):
        diffusion.x(i)
        diffusion.h(i)
    return diffusion.to_gate(label="W")

def run_grover(n, a, r):
    q = QuantumRegister(n)
    c = ClassicalRegister(n)
    Circuit = QuantumCircuit(q, c)
    for i in range(n):
        Circuit.h(q[i])
    Circuit.barrier()

    Uf = create_oracle(n, a)
    CircuitUf = QuantumCircuit(q, name='Uf')
    CircuitUf.append(Uf, q)
    uf = CircuitUf.to_gate()

    W = grover_diffusion_circuit(q, n)

    prob_a_list = []

    for ii in range(r):
        Circuit.append(uf, q)
        Circuit.barrier()
        Circuit.append(W, q)
        Circuit.barrier()

        Circuit_tmp = Circuit.copy()
        Circuit_tmp.measure(q, c)
        job = backend_sim.run(transpile(Circuit_tmp, backend_sim), shots=shots)
        counts = job.result().get_counts()
        prob_a = counts.get(format(a, f'0{n}b'), 0)/shots
        prob_a_list.append(prob_a)
        print(f"Iteration {ii+1}: P(|{format(a, f'0{n}b')}>) = {prob_a:.4f}")

    Circuit.measure(q, c)
    job_final = backend_sim.run(transpile(Circuit, backend_sim), shots=shots)
    sim_result_final = job_final.result()
    counts_final = sim_result_final.get_counts()
    print("Final measurement counts:", counts_final)

    return prob_a_list, counts_final
```

Task 1: Determining Optimal Number of Iterations r

```
optimal_r_dict = {}
for n in range(2,7):
    N = 2**n
    r_opt = floor((pi/4)*sqrt(N))
    optimal_r_dict[n] = r_opt
    print(f"n={n}, optimal r={r_opt}")
```

Results

```
n=2, optimal r=1
n=3, optimal r=2
```

```
n=4, optimal r=3
n=5, optimal r=4
n=6, optimal r=6
```

Task 2: Implementing Grover's Algorithm for n Qubits

The Grover algorithm was implemented as a Python function `run_grover(n, a, r)` which allows:

- Selecting any number of qubits (n) (2 ≤ n ≤ 6)
- Selecting a target state (a)
- Running the algorithm for a specified number of iterations (r)
- Returning the probability of measuring the target state after each iteration and the final measurement counts

The helper functions `create_oracle` and `grover_diffusion_circuit` are used to construct the phase oracle and the diffusion operator.

Task 3: Grover Simulation for n = 6

For (n = 6) and target state (a = \text{student_id} \mod 2^n = 151944 \mod 64 = 56), Grover's algorithm was simulated for the optimal number of iterations (r = 6).

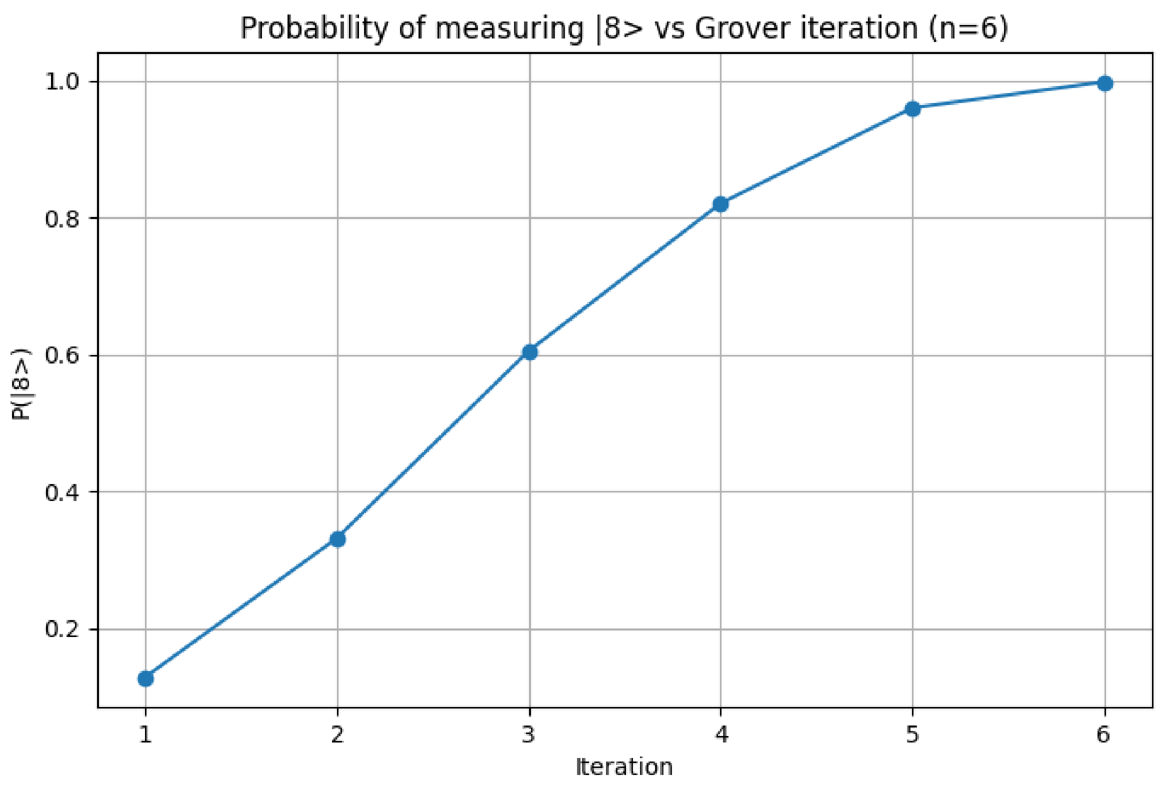
Probability of Target State per Iteration

```
Iteration 1: P(|001000>) = 0.1279
Iteration 2: P(|001000>) = 0.3311
Iteration 3: P(|001000>) = 0.6045
Iteration 4: P(|001000>) = 0.8203
Iteration 5: P(|001000>) = 0.9600
Iteration 6: P(|001000>) = 0.9980
```

Final Measurement Counts

```
Final measurement counts: {'001000': 1019, '101101': 1, '001111': 1, '011011': 1,
'101011': 1, '011101': 1}
```

Graph



Task 4: Extended Iterations to Observe Probability Oscillation

The algorithm was repeated for (s = 1, \dots, \pi/4 \cdot 2^n) to study the overshooting behavior of the target probability. Probabilities gradually increase, reach a maximum, and then start decreasing.

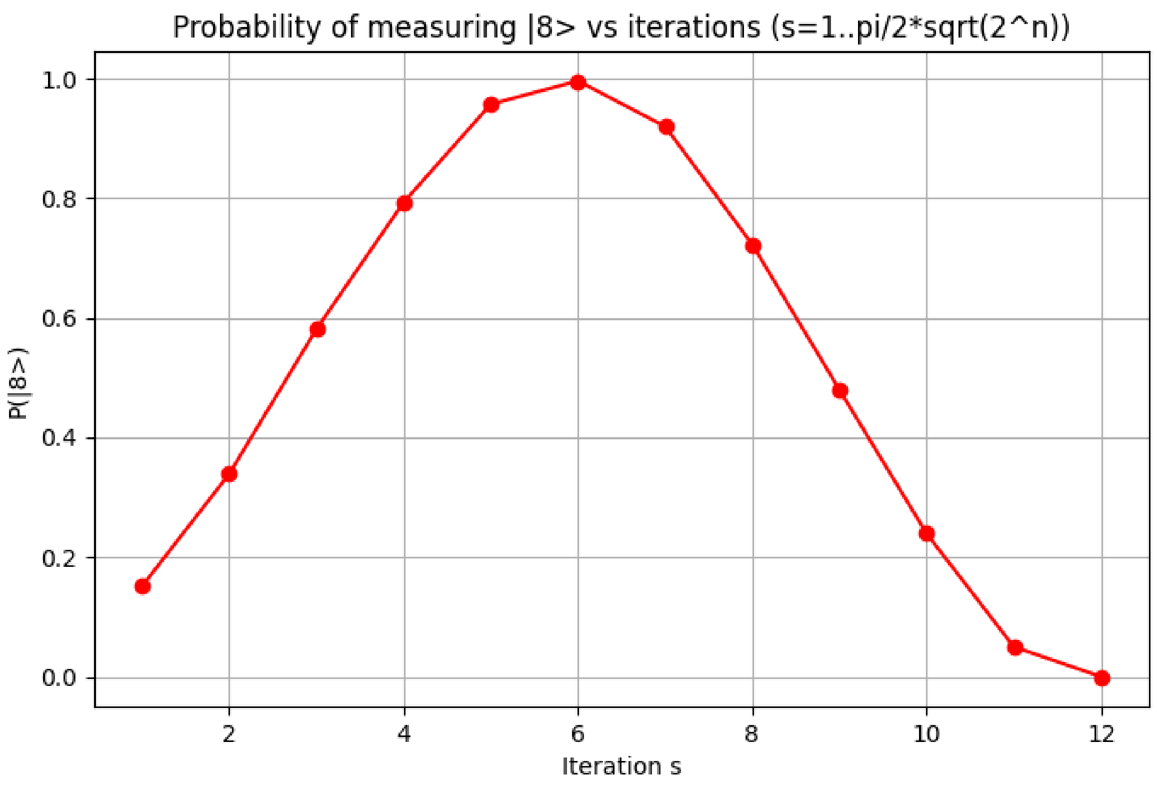
Probability of Target State per Iteration

```
Iteration 1: P(|001000>) = 0.1514
Iteration 2: P(|001000>) = 0.3389
Iteration 3: P(|001000>) = 0.5811
Iteration 4: P(|001000>) = 0.7930
Iteration 5: P(|001000>) = 0.9570
Iteration 6: P(|001000>) = 0.9961
Iteration 7: P(|001000>) = 0.9199
Iteration 8: P(|001000>) = 0.7227
Iteration 9: P(|001000>) = 0.4795
Iteration 10: P(|001000>) = 0.2393
Iteration 11: P(|001000>) = 0.0498
Iteration 12: P(|001000>) = 0.0000
```

Final Measurement Counts

```
Final measurement counts: {'101111': 9, '000110': 17, '110001': 20, '010111': 19,
'001111': 14, '100110': 19, '111010': 21, '001110': 14, '111011': 19, '100101':
14, '001101': 19, '111100': 16, '100100': 16, '101010': 21, '110110': 16,
'000001': 13, '010000': 27, '111110': 14, '100010': 8, '001011': 11, '001010': 19,
'100001': 13, '111111': 18, '011000': 13, '011001': 16, '011110': 14, '101110':
20, '000101': 15, '110010': 25, '011011': 18, '111101': 18, '001100': 13,
'100011': 15, '000010': 21, '101011': 18, '110101': 14, '101001': 17, '100000':
11, '000000': 19, '110111': 16, '110100': 13, '000011': 15, '101100': 16,
'010011': 15, '010110': 13, '010001': 18, '101000': 16, '011111': 17, '011010':
19, '101101': 22, '110011': 19, '000100': 12, '010010': 15, '011100': 16,
'100111': 17, '011101': 14, '111000': 16, '110000': 20, '000111': 17, '010101':
17, '010100': 14, '001001': 9, '111001': 14}
```

Graph



Task 5: Simulations for n = 2 ... 6

n = 2

```
Iteration 1: P(|00>) = 1.0000
Final measurement counts: {'00': 1024}
```

n = 3

```
Iteration 1: P(|000>) = 0.7744
Iteration 2: P(|000>) = 0.9590
Final measurement counts: {'000': 958, '100': 7, '010': 10, '110': 9, '111': 11,
'001': 6, '101': 16, '011': 7}
```

n = 4

```
Iteration 1: P(|1000>) = 0.5098
Iteration 2: P(|1000>) = 0.9189
Iteration 3: P(|1000>) = 0.9688
Final measurement counts: {'1000': 984, '1100': 5, '0010': 2, '0101': 4, '1101':
5, '0001': 5, '0110': 4, '0111': 4, '0100': 3, '1111': 2, '1110': 1, '0011': 1,
'1001': 1, '1011': 3}
```

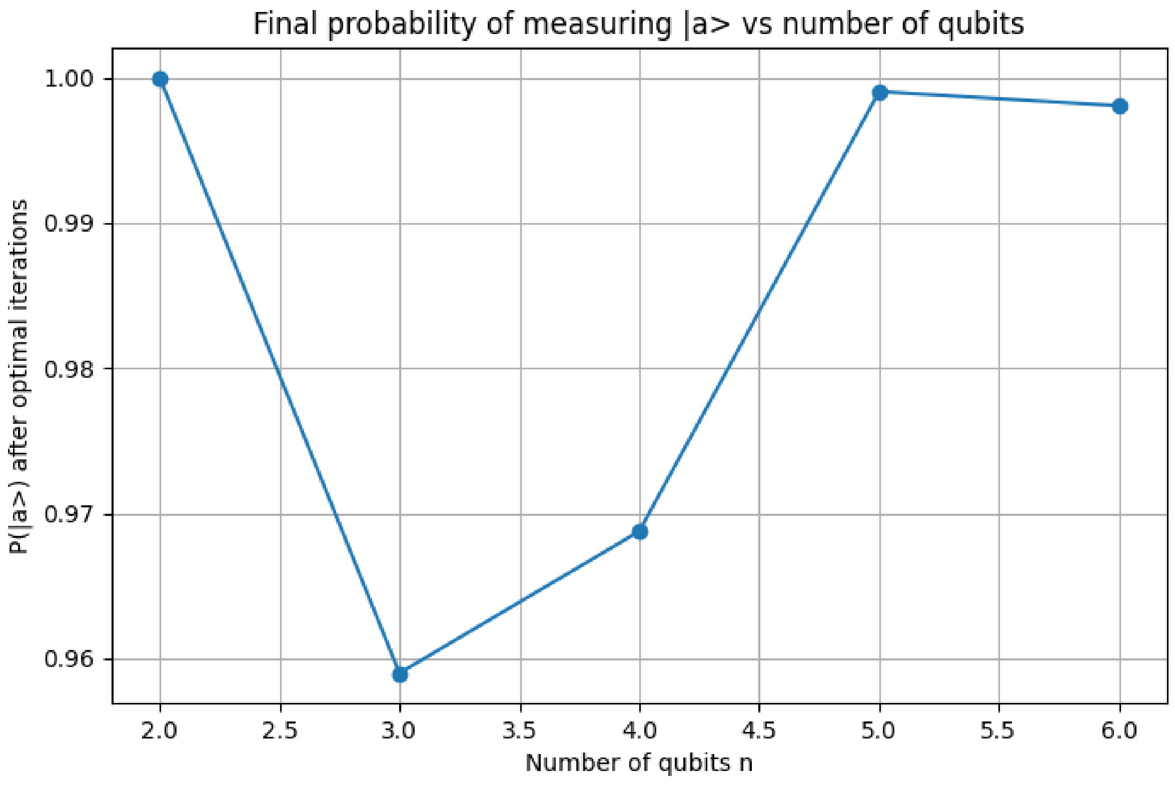
n = 5

```
Iteration 1: P(|01000>) = 0.2402
Iteration 2: P(|01000>) = 0.5859
Iteration 3: P(|01000>) = 0.9004
Iteration 4: P(|01000>) = 0.9990
Final measurement counts: {'01000': 1022, '10010': 1, '10111': 1}
```

n = 6

```
Iteration 1: P(|001000>) = 0.1543
Iteration 2: P(|001000>) = 0.3584
Iteration 3: P(|001000>) = 0.6201
Iteration 4: P(|001000>) = 0.8359
Iteration 5: P(|001000>) = 0.9590
Iteration 6: P(|001000>) = 0.9980
Final measurement counts: {'001000': 1019, '100100': 1, '111100': 1, '010010': 1,
'110010': 1, '010100': 1}
```

Graph


Final probability of measuring |a> vs number of qubits
P(|a>) after optimal iterations
Number of qubits n
1.00
0.99
0.98
0.97
0.96
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0