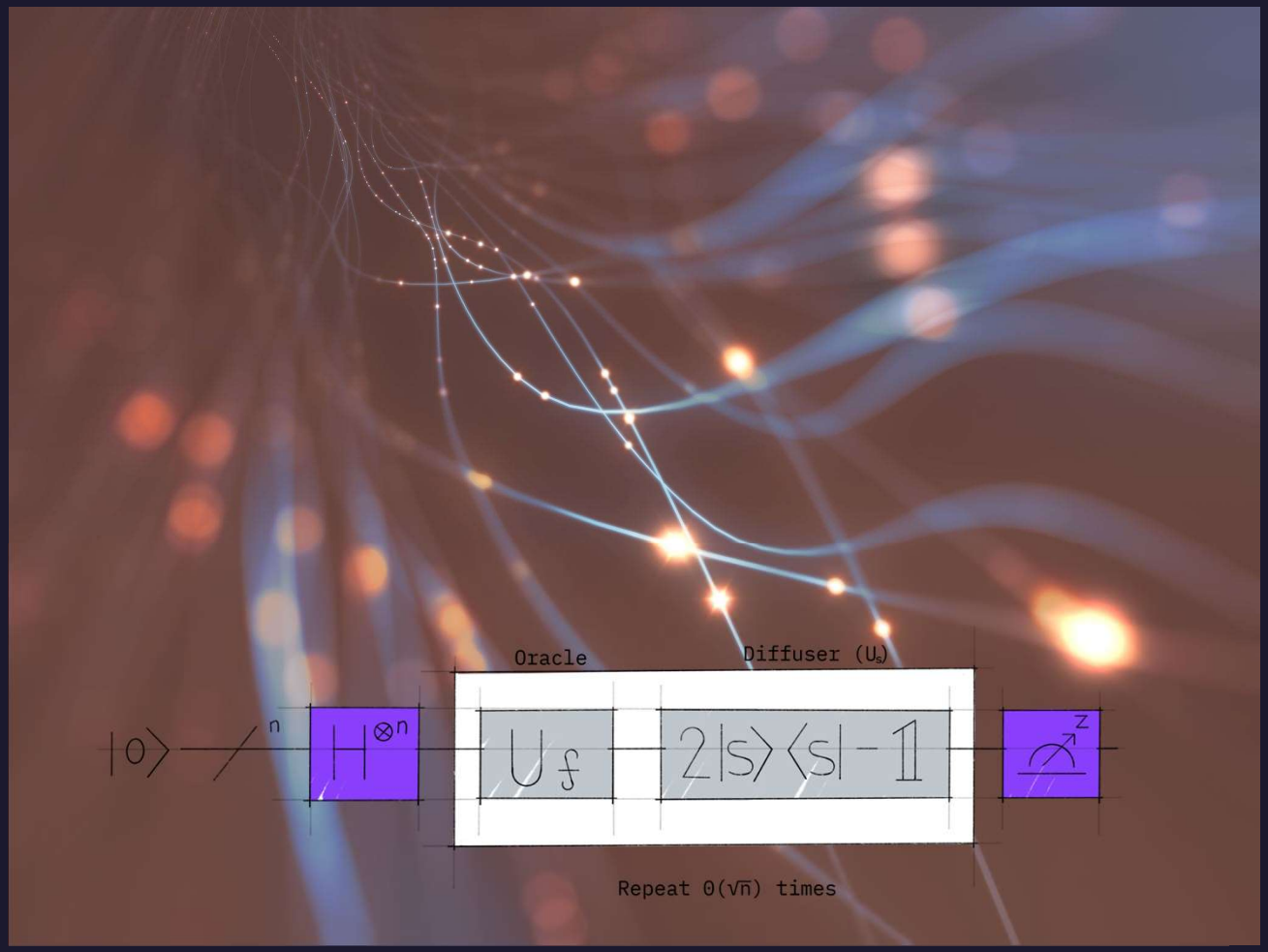


A real-life application of Grover's Algorithm

CMPT 409 – Team 1

Team 1

Brian Fu

Team Lead, Slides, Grover's Algorithm Analysis

William Horvath

Grover's Algorithm Implementation – IBM Qiskit

David Pham

Grover's Algorithm Analysis

Justin Mok

Raw data processing into python / IBM Qiskit friendly format

Ching Hang Lam

Classical Algorithm Implementation

Agenda

Premise

Explanation: Grover's Algorithm

Raw Data Processing

Classical Search Implementation

Quantum Search Implementation

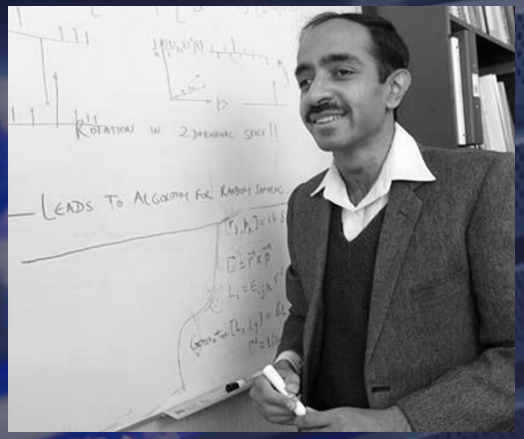
Conclusion

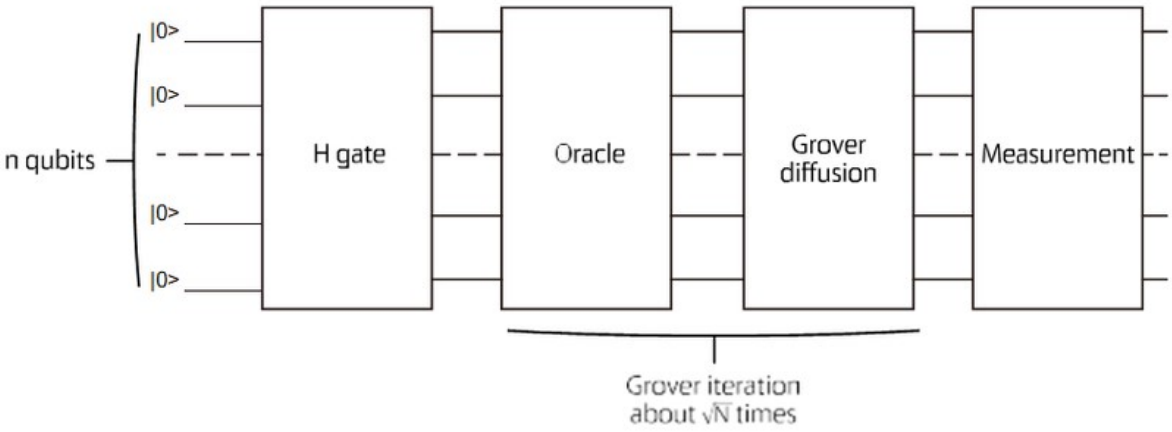
Premise

- Showcase a real-life example of quantum performance advantage
- Apply Grover's algorithm to perform unstructured database search on a public data set
- Using: League of Legends – Diamond Ranked Games (10 min) Kaggle dataset

https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min

Grover's Algorithm



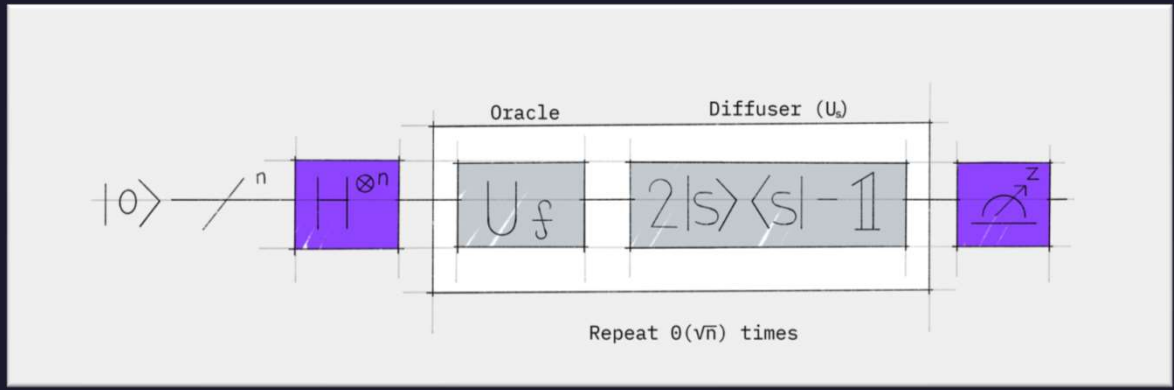


https://www.researchgate.net/publication/356001317_Quantum_Cryptography_A_voyage_from_RSA_to_QKD

Motivation

- Given some unstructured search problem; Find the answer faster than brute-force search!
  - Faster than O(N)
- For $M = 2^N$ (N bits needed), runs in O(sqrt(M)) time
- Quadratic speedup
- Using: Amplitude Amplification ("Phase/Sign inversion")

Procedure Overview

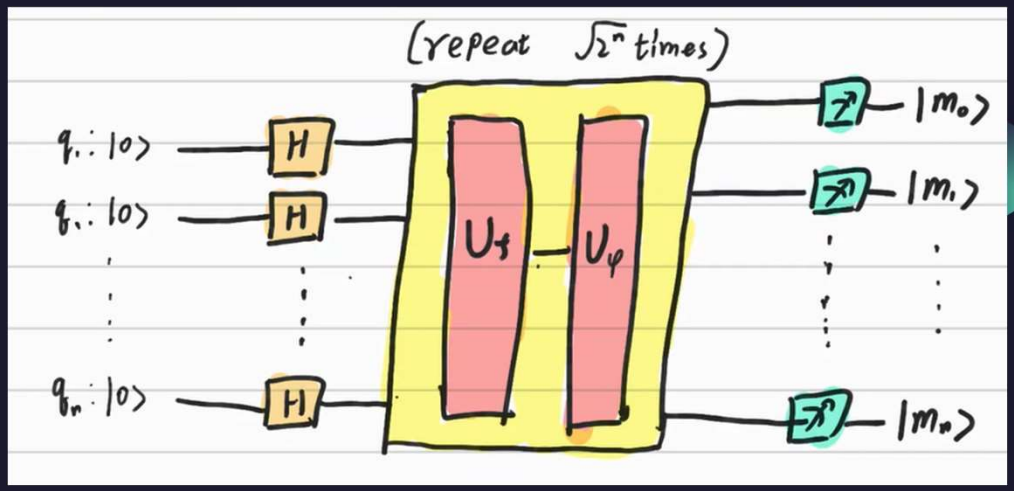


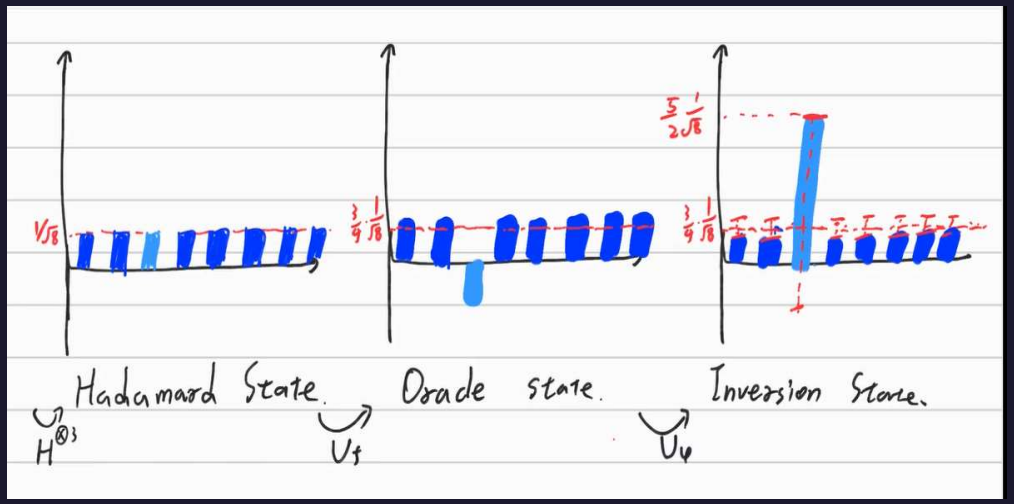
- First: Place qubits in superposition; Perform initial Hadamard!
- Core: Grover's Iterate
  - Oracle + Grover Diffusion operator ("Reflection")
- Oracle: Inverse sign of the target state
  - This lowers the mean (avg) amplitude of all states
- Diffuser: Inverse the target state again, about the new (lower!) mean amplitude
  - This increases the magnitude of the target state (amplifies its amplitude!), whilst decreasing the magnitude of other states

https://qiskit.org/textbook/ch-algorithms/grover.html

Core: Grover's Iterate

- Consists of 2 parts: U_f, the Grover oracle, and $U\psi$, the Grover diffusion ("Inversion") operator
- Grover Iterate will be repeatedly performed O(sqrt(M)) times for $M = 2^N$ values, s.t. N qubits needed.
- Each iteration will amplify the amplitude of the target state, and minimize the amplitudes of the non-target states (by decreasing the mean each iteration)
  - Target state is found using the oracle
- Note: Must Hadamard again to collapse superposition before final measurement



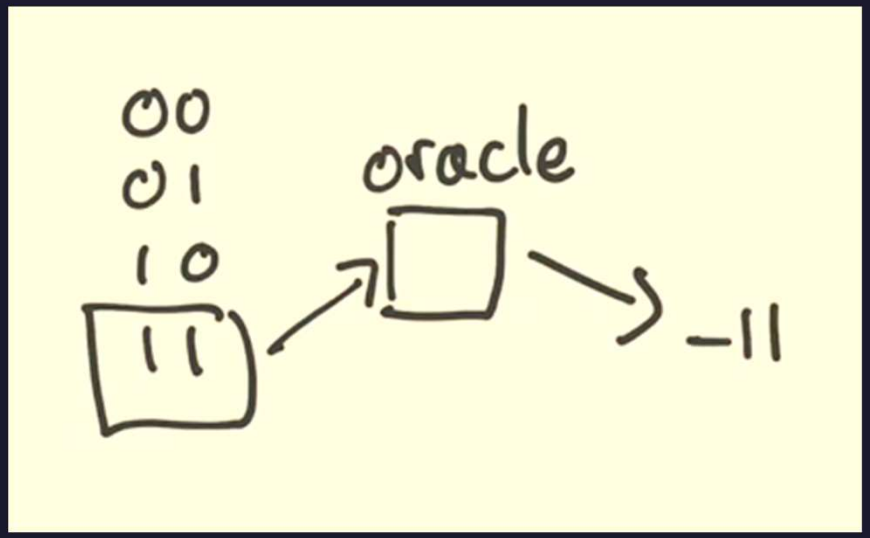


https://hgmin1159.github.io/quantum/quantum4_eng/

https://www.youtube.com/watch?v=0RPFWZj7Jm0
https://en.wikipedia.org/wiki/Quantum_logic_gate

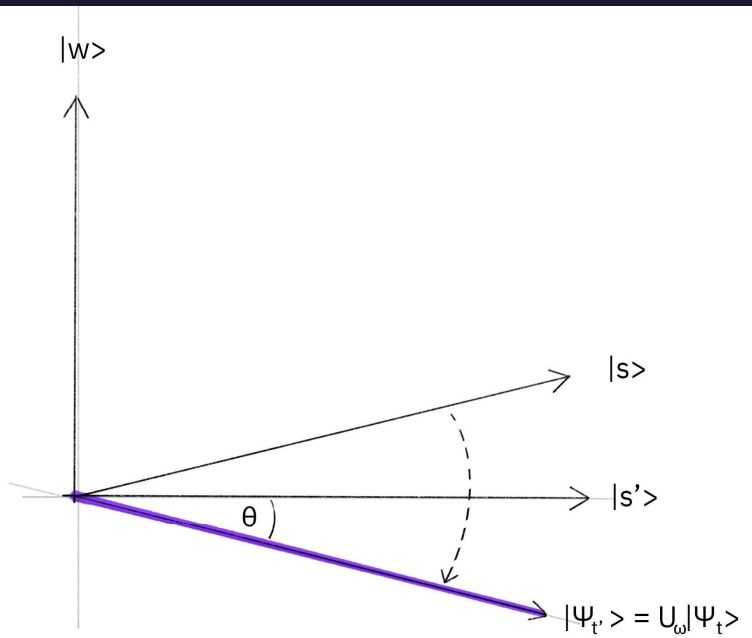
Constructing the Oracle

- Oracle must: Flip the sign of the target state!*
  - ONLY the target state! Requires unique input for some output
- Should perform an identity operation on all other states
- Example: 2 qubit search. Find ket(11). Oracle may be the unitary operator CZ (Controlled Z gate), with this operator matrix:



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

*More on this later...



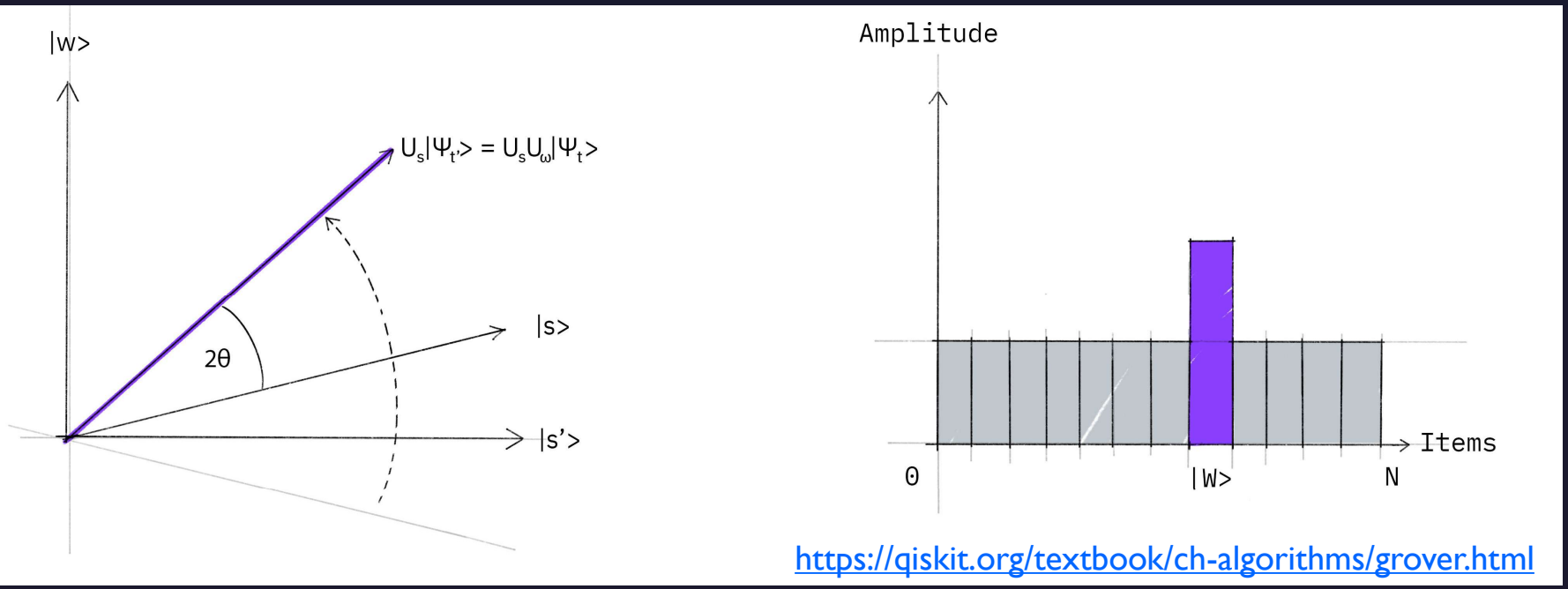
https://qiskit.org/textbook/ch-algorithms/grover.html

Oracle Operator

- Given target state ket(w) and some current superposition state ket(s), construct another state ket(s') s.t. it will be perpendicular to ket(w)
  - Remember - A quantum state IS a vector!
- Then, apply the oracle operator (U_w in this picture): Inverse your initial state ket(s) about this perpendicular state ket(s')
- In a probability amplitude plot: Represented as inversing the sign of the target state!

Diffusion Operator

- NEXT, apply the diffusion operator (U_s in this picture): Inverse the (already inversed!) current state ket(psi) about the original initial state ket(s)
- This is also represented by an inverse of the target state about the new mean amplitude (on the probability amplitude plot); The mean was previously lowered due to the first inversion of the target state!
  - This causes a net increase in magnitude of the target state, and a net decrease in magnitude of all other states!
- These 2 reflections correspond to a rotation on the Bloch sphere, moving the current state ket(psi) closer and closer to the desired state ket(w), for additional iterations of Grover's iterate



https://qiskit.org/textbook/ch-algorithms/grover.html

Raw Data Processing

File Handling

```
# Generate the modified csv file and the minimum id file
def generate_modified_csv():
    original_csv = pd.read_csv(original_csv_file_name)
    # minimum_id = original_csv['gameId'].min()
    # original_csv['gameId'] = original_csv['gameId'] - minimum_id
    # original_csv.to_csv(modified_csv_file_name, index=False)

    # minimum_id_file = open(minimum_id_file_name, 'w')
    # minimum_id_file.write(str(minimum_id))
    return original_csv

original_file = generate_modified_csv()
original_file

# modified_file = generate_modified_csv()
# modified_file
```

✓ 0.2s

	gameId	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	blueDeaths	blueAssists	blueEliteMonsters	blueDragons	...	redTowersDestroyed	redTotalGold
0	4519157822	0	28	2	1	9	6	11	0	0	...	0	16567
1	4523371949	0	12	1	0	5	5	5	0	0	...	1	17620
2	4521474530	0	15	0	0	7	11	4	1	1	...	0	17285
3	4524384067	0	43	1	0	4	5	5	1	0	...	0	16478
4	4436033771	0	75	4	0	6	6	6	0	0	...	0	17404
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Generate contents of original file

File Modification

```python
# Generate the modified csv file and the minimum id file
def generate_modified_csv():
    original_csv = pd.read_csv(original_csv_file_name)
    minimum_id = original_csv['gameId'].min()
    original_csv['gameId'] = original_csv['gameId'] - minimum_id
    original_csv.to_csv(modified_csv_file_name, index=False)

    minimum_id_file = open(minimum_id_file_name, 'w')
    minimum_id_file.write(str(minimum_id))
    return original_csv

# original_file = generate_modified_csv()
# original_file

modified_file = generate_modified_csv()
modified_file
```

✓ 0.8s

	gameId	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	blueDeaths	blueAssists	blueEliteMonsters	blueDragons	...
0	223799751	0	28	2	1	9	6	11	0	0	...
1	228013878	0	12	1	0	5	5	5	0	0	...
2	226116459	0	15	0	0	7	11	4	1	1	...
3	229025996	0	43	1	0	4	5	5	1	0	...
4	140675700	0	75	4	0	6	6	6	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...

Generate Modified CSV

Generate List

```
# Create a id list
def create_id_list(modified_csv):
    return modified_csv.index.to_list()
```

✓ 0.1s

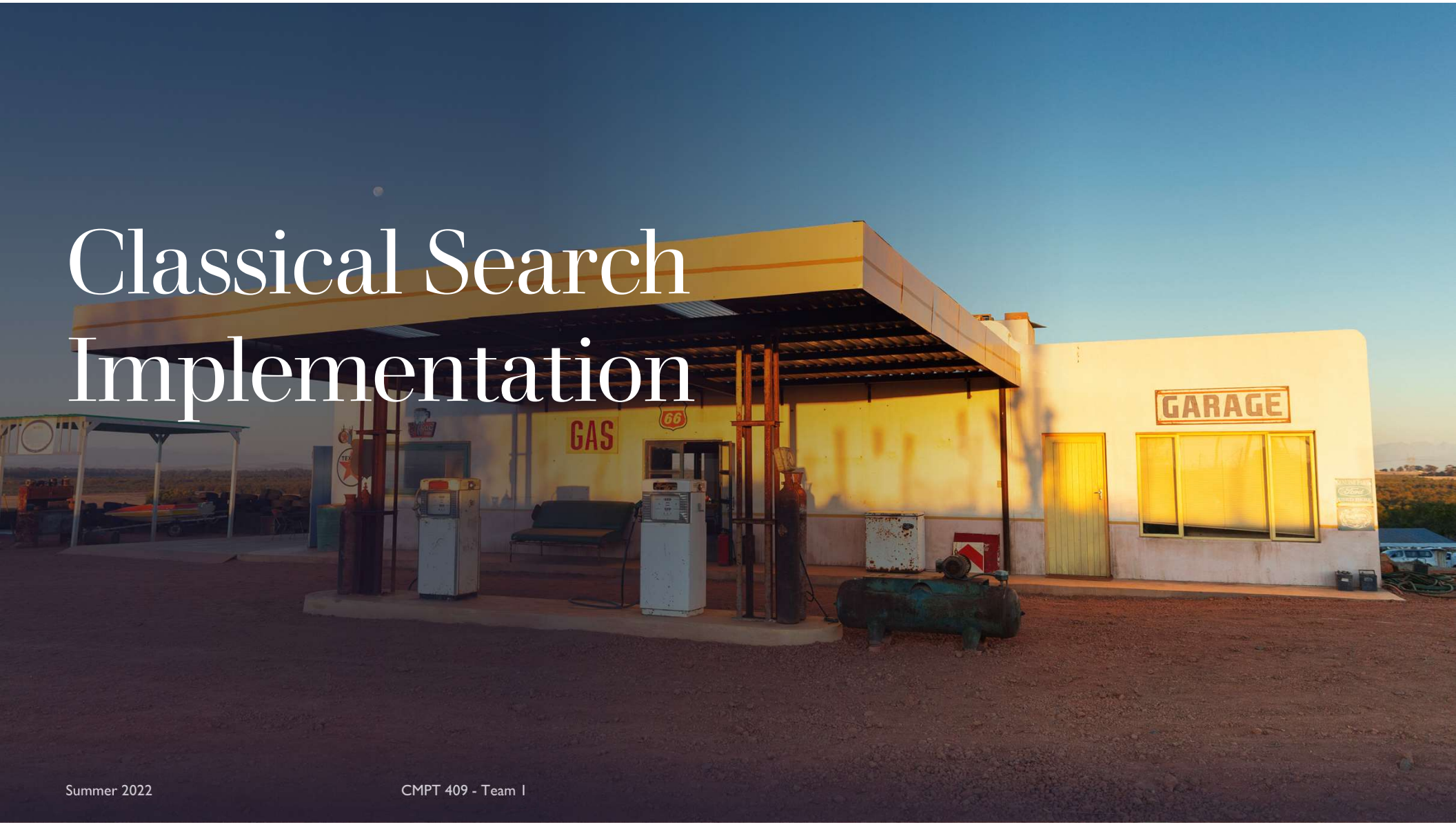
```
id_list = create_id_list(modified_csv)
id_list
```

✓ 0.1s

```
Output exceeds the size limit. Open the full output data in a text editor
[223799751,
 228013878,
 226116459,
 229025996,
 140675700,
 180007638,
 197652561,
 201401287,
 147689959,
 214075275,
 156804502,
 157680085,
 220236714,
```

List of GameIDs.

Classical Search Implementation



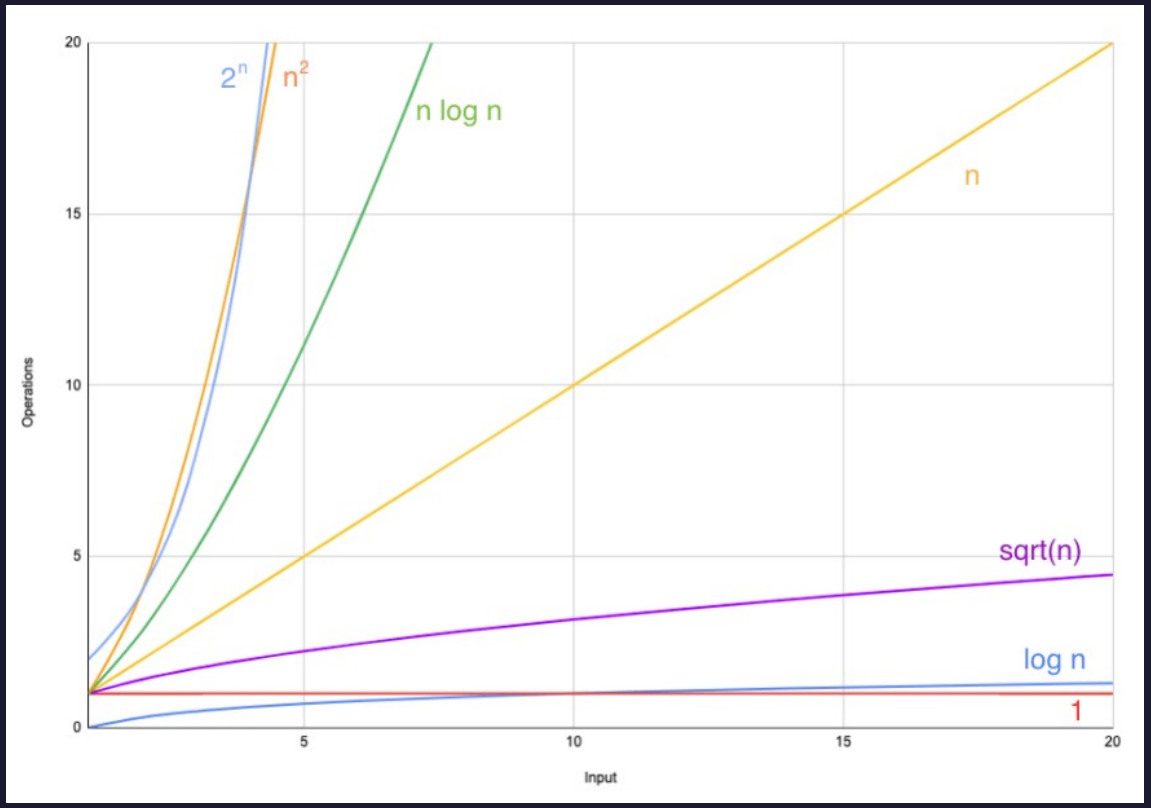
Summer 2022

CMPT 409 - Team 1

Objectives

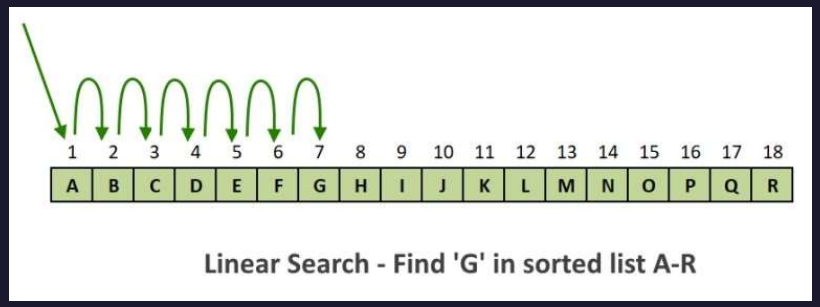
- Use Grover's Algorithm to search in the database
- Is it possible to get better performance using Grover's Algorithm compared to using classical algorithms?
- Which situations have better performance?
- Which situations have poor performance?

Big-O Complexity

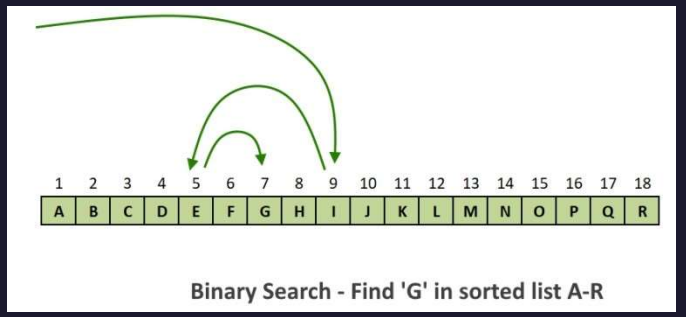


Classical Search

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)



Linear Search - Find 'G' in sorted list A-R



Binary Search - Find 'G' in sorted list A-R

Quantum Search

Algorithm	Best Time Complexity	Average Time Complexity
Grover's Algorithm	O(sqrt(n))	O(sqrt(n))

Structured Data

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity
Array	O(1)	O(n)	O(n)
Stack	O(1)	O(n)	O(n)
Queue	O(1)	O(n)	O(n)
Singly Linked List	O(1)	O(n)	O(n)
Doubly Linked List	O(1)	O(n)	O(n)
Hash Table	O(1)	O(1)	O(n)

Linear Search Implementation

```python
# Linear search in the list (index version)
# Returns
# target_index: the index of target, -1: target id does not exist
def linear_search_list_index(id_list, target_id):
    for i in range(len(id_list)):
        if id_list[i] == target_id:
            return i
    return -1
```

```python
# Get the target info using target index
def get_target_info_list_index(modified_csv, target_index):
    return modified_csv.iloc[target_index]
```

Brief Summary

- Works on Unstructured and Unsorted data
- Grover's algorithm has quadratic speed up over the Linear Search
  - With an unforeseen caveat...

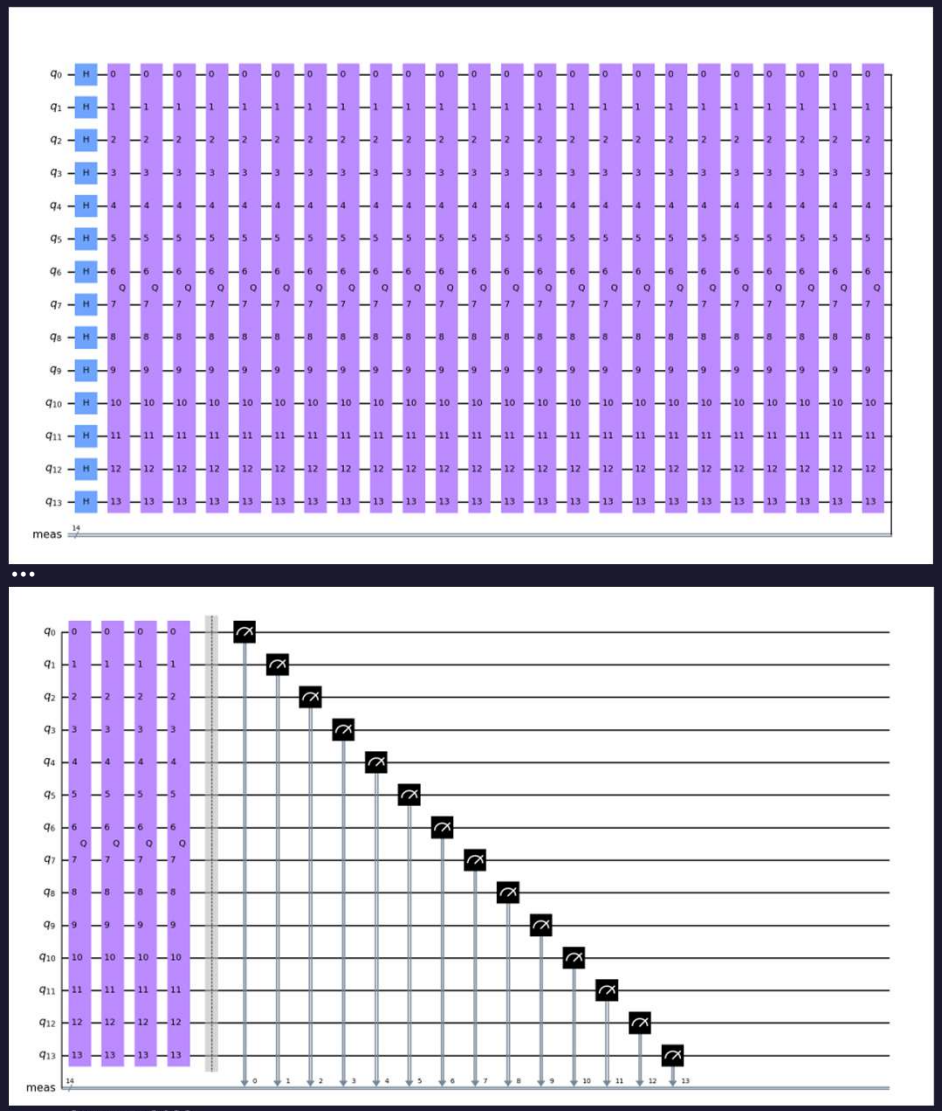
Quantum Search Implementation

Qiskit makes it easy

Except for one thing...

```
# number of qbits required (14 in our case)
n = math.ceil(math.log(size, 2))

# create an oracle from a state vector with the phases flipped for the value(s) we want to find

oracle = Statevector([1 if state < size and id_list[state] == target_id else 0 for state in range(0, 2**n)])

problem = AmplificationProblem(oracle, is_good_state=target_binary)

grover = Grover(iterations=iters)
grover_circuit = grover.construct_circuit(problem)
grover_circuit.measure_all()

# grover iterations within circuit (78 in our case)
# (pi*sqrt(N))/4
iters = int((math.pi*math.sqrt(size))/4)

# circuit executions (1024 is default)
shots = 1024

# calculate result
result = execute(grover_circuit, backend=Aer.get_backend('aer_simulator'), shots=shots).result()
all_probabilities = result.get_counts()
```

...we need to do a brute force search to create our oracle



Qiskit Circuit Diagram

- Rows: single qubits
  - Our database (N = 9879) requires 14 qubits to store each index*
- Purple columns: Grover's algorithm iteration
  - Our database requires 78 iterations to achieve the correct result
  - General case: π*sqrt(N)/4

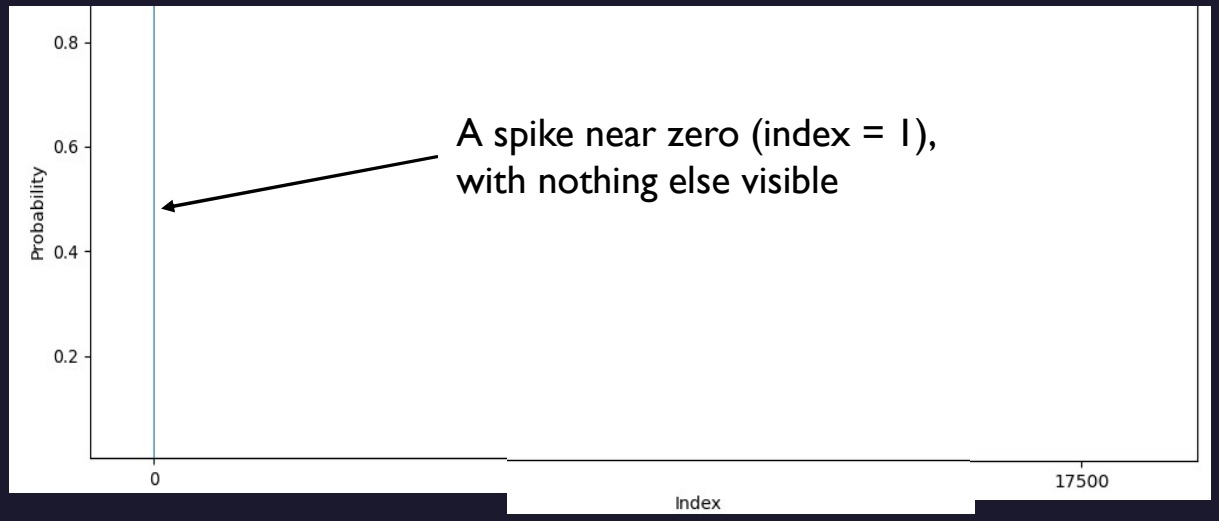
*To store each possible key value (<300million), we would need 28 qubits

Result

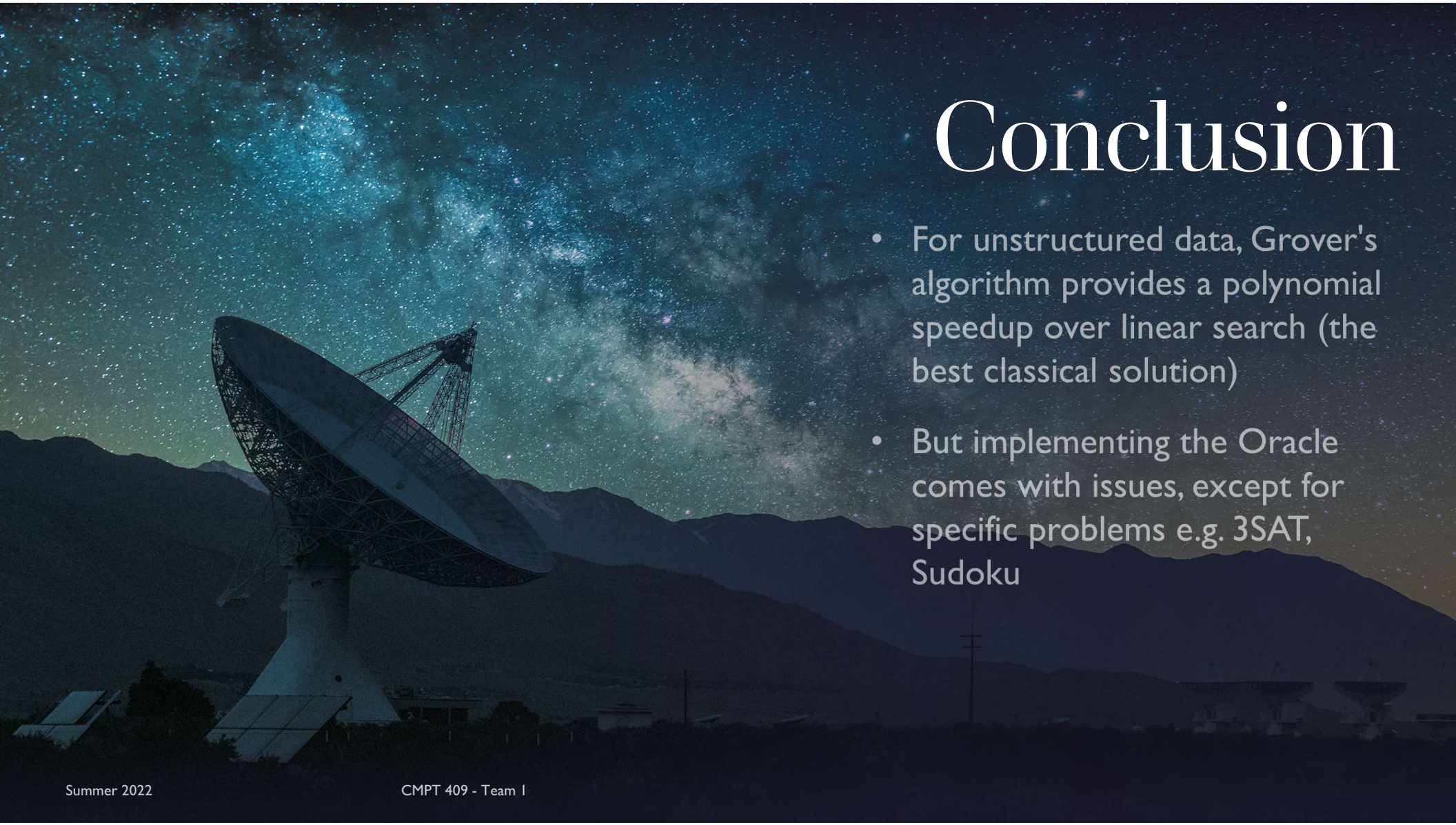
It achieves the same result for searching the key "228013878" as our reference classical algorithm.

The index found is 1 (second entry) in both cases.

```
Linear search result index: 1

Begin Grover's search...
Index probabilities:
1 : 0.8984375
Other possible indices have insignificant probabilities.
Grover's search result index: 1
Quantum index: 1
Correct index: 1
Success!
```



Execution time:
27 minutes on an i7-8700k



Conclusion

- For unstructured data, Grover's algorithm provides a polynomial speedup over linear search (the best classical solution)
- But implementing the Oracle comes with issues, except for specific problems e.g. 3SAT, Sudoku

Q & A

References

- https://qiskit.org/textbook/ch-algorithms/grover.html
- https://www.youtube.com/watch?v=0RPFWZj7Jm0
- https://www.youtube.com/watch?v=YEI5vYdcoQ4
- https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min
- https://en.wikipedia.org/wiki/Quantum_logic_gate
- https://www.quantiki.org/wiki/grovers-search-algorithm
- https://hgmin1159.github.io/quantum/quantum4_eng/
- https://www.researchgate.net/publication/356001317_Quantum_Cryptography_A_voyage_from_RSA_to_QKD
- https://www.geeksforgeeks.org/searching-algorithms/
- Dr. Steven Pearce's CMPT 409 Course Slides