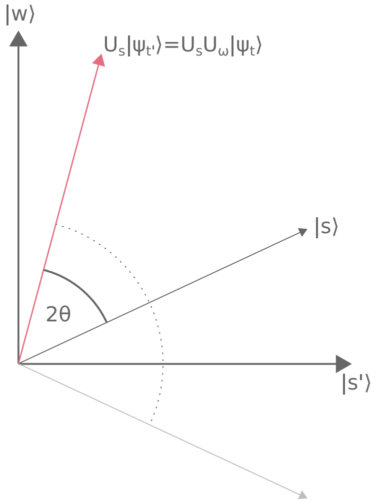
Grover's Algorithm

Hans Hohenfeld – June, 19th 2023



QC for Software Engineers
Universität Bremen & DFKI RIC
Robert-Hooke-Straße 1

28359 Bremen

Universität Bremen
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Searching and Finding. . .

No.	Name	Phone No.
1	Alice	0123/45678
2	Anna	9876/654321
3	Arthur	999/654456
⋮	⋮	

No.	Name	Phone No.
1	Alice	0123/45678
2	Anna	9876/654321
3	Arthur	999/654456
⋮	⋮	

What is **Mike**'s phone number?

No.	Name	Phone No.
1	Alice	0123/45678
2	Anna	9876/654321
3	Arthur	999/654456
⋮	⋮	
725	**Mike**	**555/789789**
⋮	⋮	

What is **Mike**'s phone number?

Assumptions

Assumptions

- **Order**: e. g. **John** $\leq$ **Sabine**.

Assumptions

- **Order**: e. g. **John** $\leq$ **Sabine**.
- The list is **sorted**.

Assumptions

- **Order**: e. g. **John** $\leq$ **Sabine**.
- The list is **sorted**.
- We know the list's **length**.

Assumptions

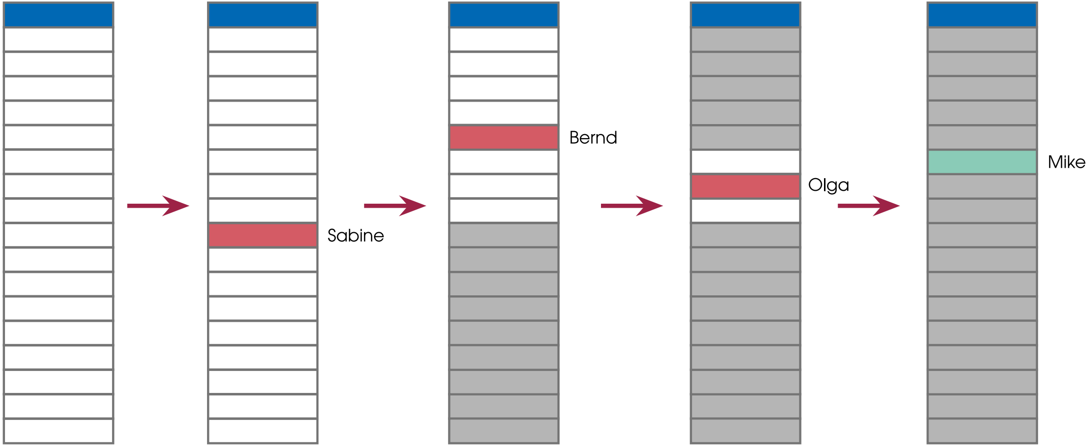
- **Order**: e. g. **John** $\leq$ **Sabine**.
- The list is **sorted**.
- We know the list's **length**.
- We can **read** every individual line.

Assumptions

- **Order**: e. g. **John** $\leq$ **Sabine**.
- The list is **sorted**.
- We know the list's **length**.
- We can **read** every individual line.
- Nothing else.

How do we find **Mike**'s phone number?

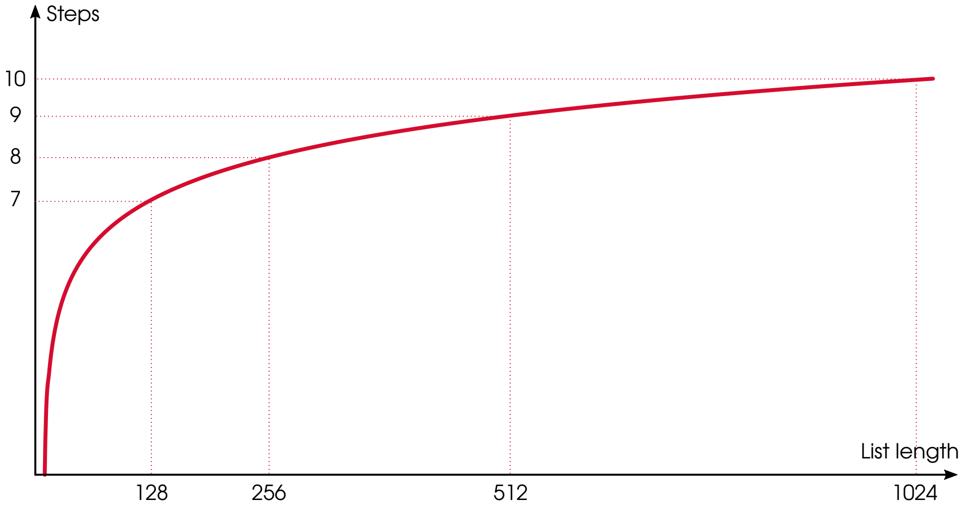
Binary Search



Time Complexity

Informally **time complexity** is a measure for how the computational effort in terms of time grows, relative to the problem size.

Time complexity of Binary Search



Time Complexity

More formally:

Time Complexity

More formally:

The computational effort to find an element in a sorted list of length n does not grow faster than $\log(n)$ (up to constant factors).

Or even more formally...

Time Complexity

Binary Search: Time Complexity

Let $T_{\mathrm{BS}} : \mathbb{N} \to \mathbb{N}$ be the **time step** function of Binary Search. It holds:

Time Complexity

Binary Search: Time Complexity

Let $T_{\text{BS}} : \mathbb{N} \to \mathbb{N}$ be the **time step** function of Binary Search. It holds:

$$\limsup_{n\to\infty} \frac{T_{\text{BS}}(n)}{\log(n)} < \infty.$$

Time Complexity

Binary Search: Time Complexity

Let $T_{\text{BS}} : \mathbb{N} \to \mathbb{N}$ be the **time step** function of Binary Search. It holds:

$$\limsup_{n \to \infty} \frac{T_{\text{BS}}(n)}{\log(n)} < \infty.$$

We write:

$$T_{\text{BS}} \in O(\log(n))$$

Time Complexity

Binary Search: Time Complexity

Let $T_{\text{BS}} : \mathbb{N} \to \mathbb{N}$ be the **time step** function of Binary Search. It holds:

$$\limsup_{n \to \infty} \frac{T_{\text{BS}}(n)}{\log(n)} < \infty.$$

We write:

$$T_{\text{BS}} \in O(\log(n))$$

We say: Binary Search has (time) complexity $O(\log(n))$.

Time Complexity

Big-O	Name	Problem

Time Complexity

Big-O	Name	Problem
$O(1)$	Constant	Median of a sorted list

Time Complexity

Big-O	Name	Problem
$O(1)$	Constant	Median of a sorted list
$O(\log(n))$	Logarithmic	Search in a sorted list

Time Complexity

Big-O	Name	Problem
$O(1)$	Constant	Median of a sorted list
$O(\log(n))$	Logarithmic	Search in a sorted list
$O(n)$	Linear	Minimum, Maximum (unsorted)

Time Complexity

Big-O	**Name**	**Problem**
$O(1)$	Constant	Median of a sorted list
$O(\log(n))$	Logarithmic	Search in a sorted list
$O(n)$	Linear	Minimum, Maximum (unsorted)
$O(n \log n)$	Super-Linear	Sorting

Time Complexity

Big-O	Name	Problem
$O(1)$	Constant	Median of a sorted list
$O(\log(n))$	Logarithmic	Search in a sorted list
$O(n)$	Linear	Minimum, Maximum (unsorted)
$O(n \log n)$	Super-Linear	Sorting
$O(n^c)$	Polynomial	Arithmetic, Linear Programming, ...

Time Complexity

Big-O	Name	Problem
$O(1)$	Constant	Median of a sorted list
$O(\log(n))$	Logarithmic	Search in a sorted list
$O(n)$	Linear	Minimum, Maximum (unsorted)
$O(n \log n)$	Super-Linear	Sorting
$O(n^c)$	Polynomial	Arithmetic, Linear Programming, ...
$O(c^n)$	Exponential	Chess, Go, ...

We change the problem a bit. . .

No.	Name	Phone No.
1	Alice	0123/45678
2	Anna	9876/654321
3	Arthur	999/654456
⋮	⋮	

No.	Name	Phone No.
1	Alice	0123/45678
2	Anna	9876/654321
3	Arthur	999/654456
⋮	⋮	

Who has the phone number **1234/999888**?

This type of problem is called

Unstructured Search.

For reasons that become more clear later, we divide the problem between two instances.

The Seeker

- Has access to the list . . .

The Seeker

- Has access to the list ...
- Can read arbitrary entries in constant time ...

The Seeker

- Has access to the list . . .
- Can read arbitrary entries in constant time . . .
- Decides which entry to read next . . .

The Oracle

- Knows the answer ...

The Oracle

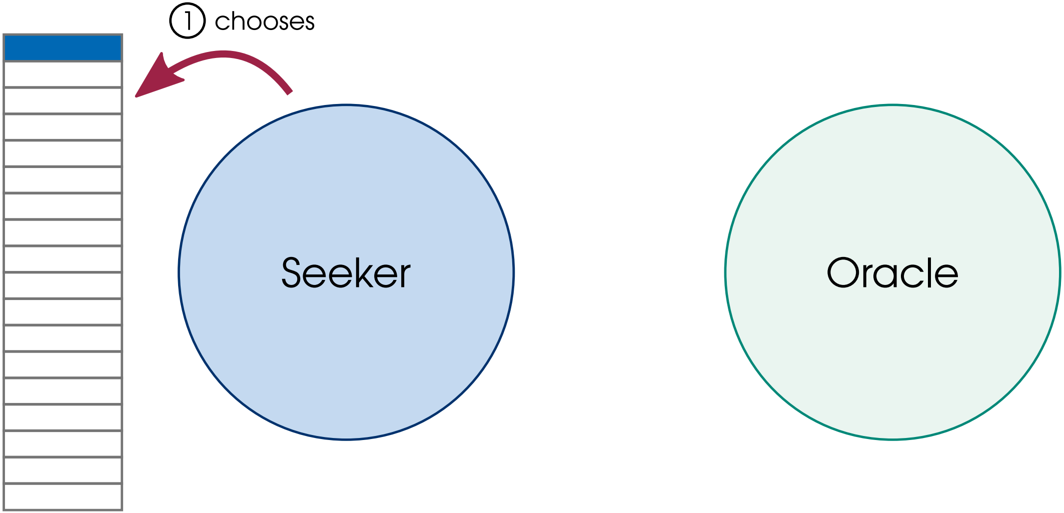
- Knows the answer ...
- Gets entries suggested by the seeker and answers with **yes/no** ...

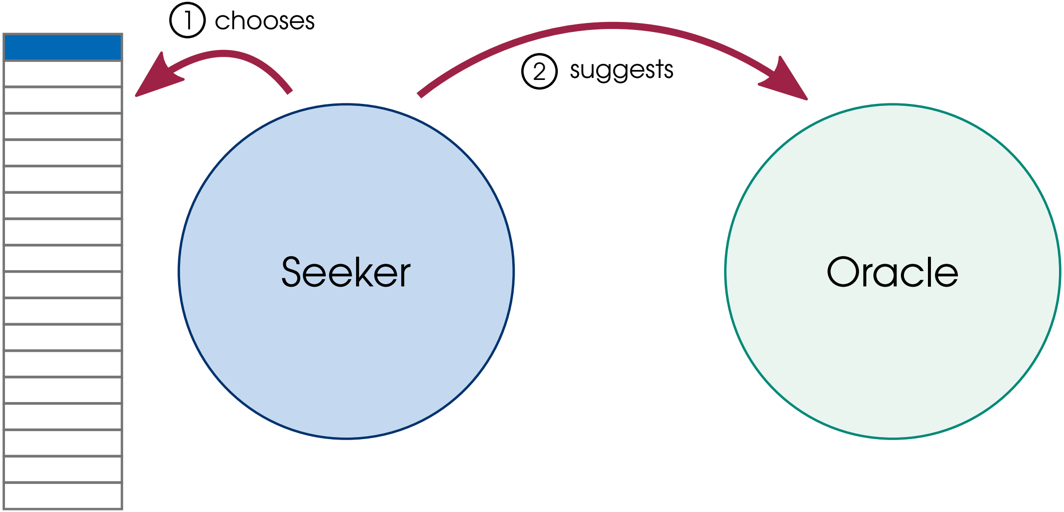
The Oracle

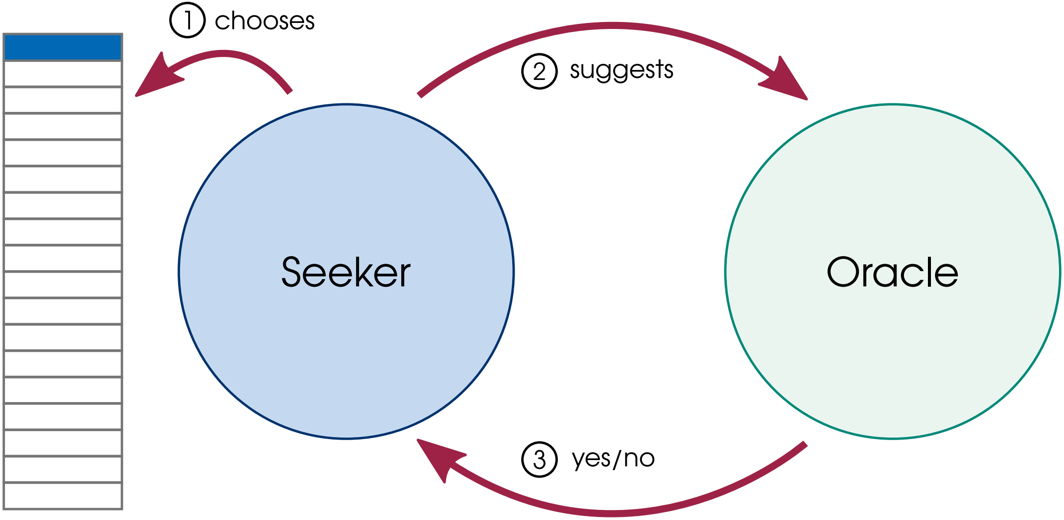
- Knows the answer ...
- Gets entries suggested by the seeker and answers with **yes/no** ...
- Takes constant time for an answer ...

The Oracle

- Knows the answer ...
- Gets entries suggested by the seeker and answers with **yes/no** ...
- Takes constant time for an answer ...
- Is never wrong ...


1 chooses
Seeker
Oracle


1 chooses
2 suggests
Seeker
Oracle


1 chooses
2 suggests
3 yes/no
Seeker
Oracle

How often does the Seeker need to query the Oracle?

Linear Search

- The only reasonable strategy is to query the Oracle for one entry after the other, starting from the first.

Linear Search

- The only reasonable strategy is to query the Oracle for one entry after the other, starting from the first.
- For a list of length n this requires n queries in the worst case.

Linear Search

- The only reasonable strategy is to query the Oracle for one entry after the other, starting from the first.
- For a list of length n this requires n queries in the worst case.
- On average, the search is successful after $\frac{n}{2}$ queries.

Linear Search

- The only reasonable strategy is to query the Oracle for one entry after the other, starting from the first.
- For a list of length n this requires n queries in the worst case.
- On average, the search is successful after $\frac{n}{2}$ queries.
- **Important:** The Oracle does not change anything with regards to complexity.

Linear Search

Lineare Suche has (query/time) complexity of $O(n)$.

Questions?

Searching and Finding . . .

Searching and Finding . . . on Quantum Computers

Grover's Algorithm

Grover's algorithm is a quantum algorithm for unstructured search problems.

Grover's Algorithm

Grover's algorithm is a quantum algorithm for unstructured search problems.

- Formulated 1996 by Lov K. Grover [4].

Grover's Algorithm

Grover's algorithm is a quantum algorithm for unstructured search problems.

- Formulated 1996 by Lov K. Grover [4].
- The second quantum algorithm of practical relevance after Shor's factoring algorithm in 1994 [6].

Grover's Algorithm

Grover's algorithm is a quantum algorithm for unstructured search problems.

- Formulated 1996 by Lov K. Grover [4].
- The second quantum algorithm of practical relevance after Shor's factoring algorithm in 1994 [6].
- Implemented for 3 qubits on a programmable quantum computer in 2017 [3].

Problem Statement

Given n qubits with $N = 2^n$ basis states

Problem Statement

Given n qubits with $N = 2^n$ basis states

$$
\begin{gathered}
|0\ldots0\rangle = |0\rangle \\
|0\ldots1\rangle = |1\rangle \\
\ldots \\
|1\ldots1\rangle = |N-1\rangle
\end{gathered}
$$

Problem Statement

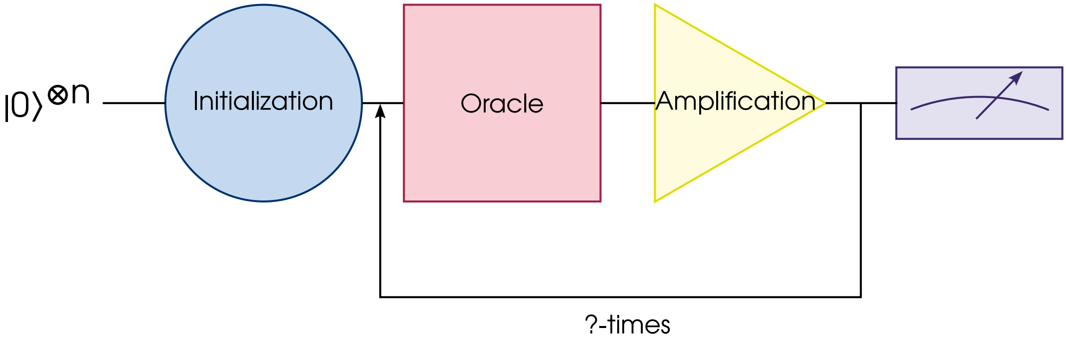
One of the basis states is the one we are searching for. We will call this state:

Problem Statement

One of the basis states is the one we are searching for. We will call this state:

$$|\omega\rangle$$

Let's first look at a schematic of the algorithm...



$|0\rangle^{\otimes n}$
Initialization
Oracle
Amplification
?-times

Oracle and **Amplitude Amplification** are executed several times

Oracle and **Amplitude Amplification** are executed several times

How often will be the result of the following discussion.

First the details of each component. . .

Initialization

At the start, we know nothing about the state we are searching.

Initialization

At the start, we know nothing about the state we are searching. This indifference is expressed by initializing the system into a superposition over all basis states.

Initialization

We achieve this, by applying a **Hadamard** gate to each of the *n* qubits.

Initialization

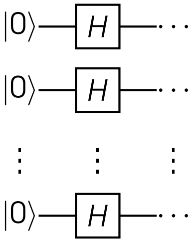
We achieve this, by applying a **Hadamard** gate to each of the *n* qubits.

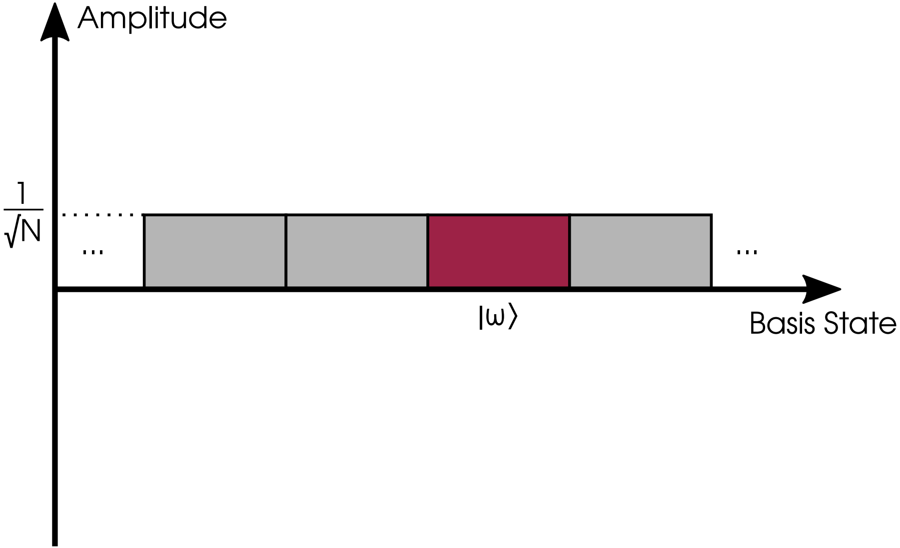
$$|s\rangle = H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

Initialization

We achieve this, by applying a **Hadamard** gate to each of the n qubits.

$$|s\rangle = H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$




Amplitude
$\frac{1}{\sqrt{N}}$
...
...
$|\omega\rangle$
Basis State

After initializing the system, we have an equal probability of $\frac{1}{N}$ to measure either of the N basis states.

Oracel

The oracle U_ω marks the state $|\omega\rangle$ we are searching for. After applying the oracle, we want for each basis state:

Oracel

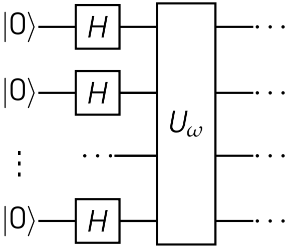
The oracle U_ω marks the state $|\omega\rangle$ we are searching for. After applying the oracle, we want for each basis state:

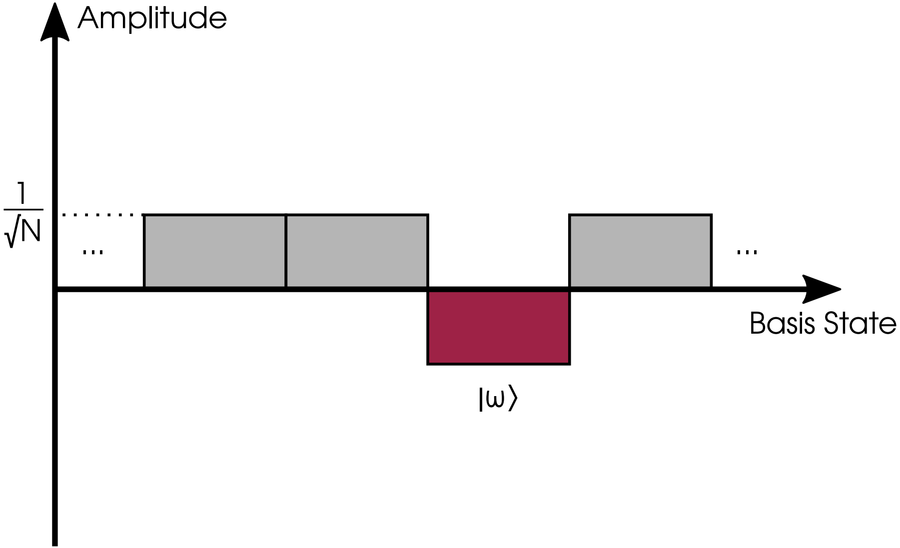
$$U_\omega |x\rangle = \begin{cases} -|x\rangle & \text{if } |x\rangle = |\omega\rangle \\ |x\rangle & \text{else} \end{cases}$$

Oracel

The oracle U_ω marks the state $|\omega\rangle$ we are searching for. After applying the oracle, we want for each basis state:

$$U_\omega |x\rangle = \begin{cases} -|x\rangle & \text{if } |x\rangle = |\omega\rangle \\ |x\rangle & \text{else} \end{cases}$$




Amplitude
$\frac{1}{\sqrt{N}}$
...
...
Basis State
$|\omega\rangle$

Probabilities of measuring either basis state remain unchanged by this.
The oracle only inverted the phase of the state we are searching for!

Oracle

How to construct such an oracle?

Oracle

How to construct such an oracle?

- U_ω is a diagonal matrix.

Oracle

How to construct such an oracle?

- U_ω is a diagonal matrix.
- All but one entry on its diagonal are 1.

Oracle

How to construct such an oracle?

- U_ω is a diagonal matrix.
- All but one entry on its diagonal are 1.
- In the line with index corresponding to the searched basis state, the entry is -1.

Oracle

Let $|N-1\rangle$ be the searched basis state. The corresponding oracle is then:

Oracle

Let $|N-1\rangle$ be the searched basis state. The corresponding oracle is then:

$$U_\omega = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -1 \end{pmatrix}$$

Amplitude Amplification

The last step of the algorithm is the amplitude amplification U_s. It does three things:

Amplitude Amplification

The last step of the algorithm is the amplitude amplification U_s. It does three things:

- It inverts the phase of the searched basis state again.

Amplitude Amplification

The last step of the algorithm is the amplitude amplification U_S. It does three things:

- It inverts the phase of the searched basis state again.
- It increases (amplifies) the amplitude of the searched basis stated.

Amplitude Amplification

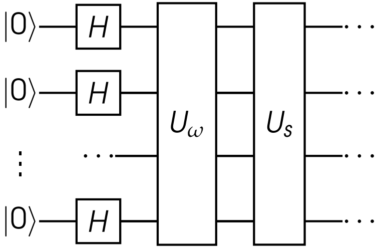
The last step of the algorithm is the amplitude amplification U_S. It does three things:

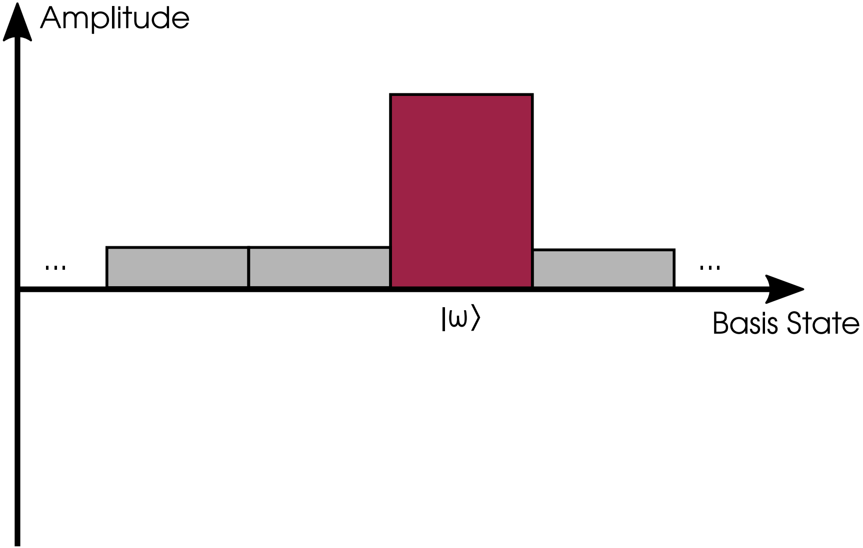
- It inverts the phase of the searched basis state again.
- It increases (amplifies) the amplitude of the searched basis stated.
- It decreases all other amplitudes proportionally.

Amplitude Amplification

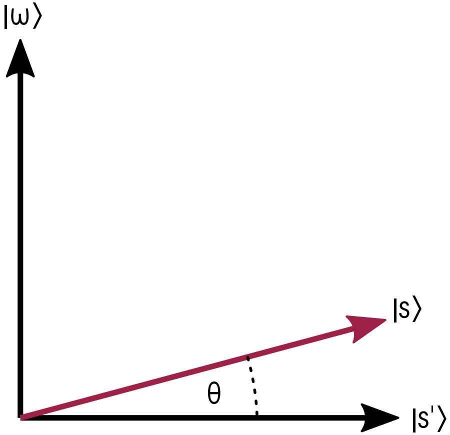
The last step of the algorithm is the amplitude amplification U_s. It does three things:

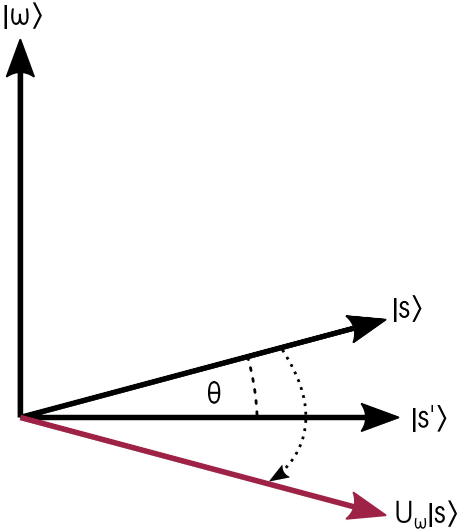
- It inverts the phase of the searched basis state again.
- It increases (amplifies) the amplitude of the searched basis stated.
- It decreases all other amplitudes proportionally.

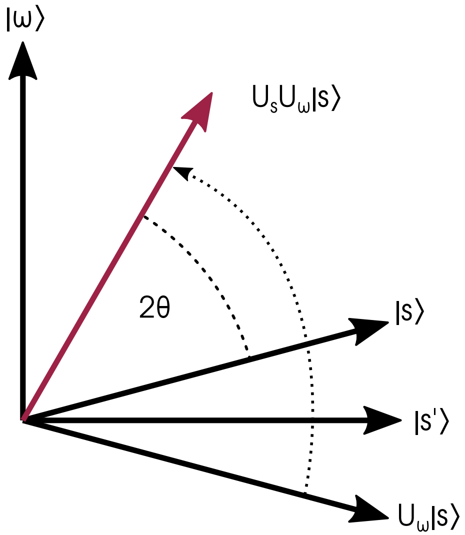



Amplitude
...
...
|ω⟩
Basis State

A geometric view on the algorithm…


|ω⟩
|s⟩
θ
|s'⟩


|ω⟩
|s⟩
θ
|s'⟩
U_ω|s⟩



$|\omega\rangle$
$U_sU_\omega|s\rangle$
2θ
$|s\rangle$
$|s'\rangle$
$U_\omega|s\rangle$

Amplitude Amplification

The amplitude amplification step mirrors $U_\omega |s\rangle$ on the vector $|s\rangle$.

Amplitude Amplification

The amplitude amplification step mirrors $U_\omega |s\rangle$ on the vector $|s\rangle$.The operator to achieve this is given by:

Amplitude Amplification

The amplitude amplification step mirrors $U_\omega \left|s\right\rangle$ on the vector $\left|s\right\rangle$.The operator to achieve this is given by:

$$U_s = 2\left|s\right\rangle\left\langle s\right| - I.$$

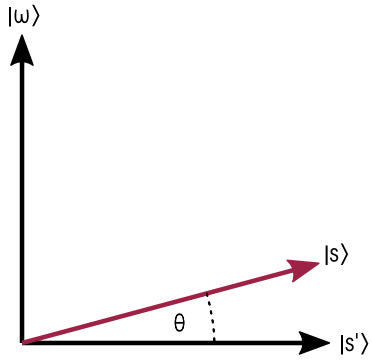
Grover's Algorithm: Complexity

How often do we have to apply the oracle and amplitude amplification?

Looking at θ gives an intuition.

Grover's Algorithm: Complexity

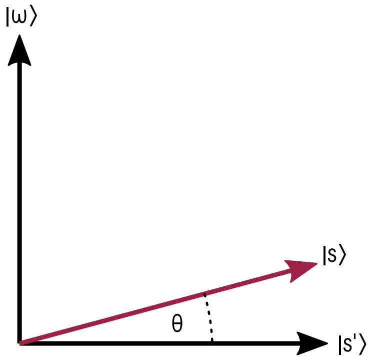
We have:



Grover's Algorithm: Complexity

We have:

$$|s\rangle = \sin\theta\,|\omega\rangle + \cos\theta\,|s'\rangle$$



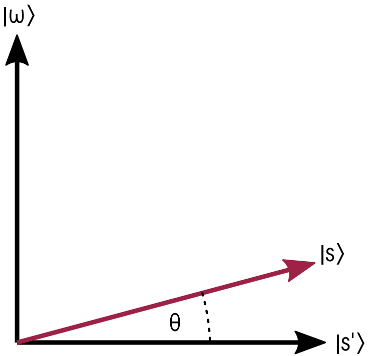
Grover's Algorithm: Complexity

We have:

$$|s\rangle = \sin\theta\,|\omega\rangle + \cos\theta\,|s'\rangle$$

with

$$\theta = \arcsin\langle s|\omega\rangle = \frac{1}{\sqrt{N}}$$



Grover's Algorithm: Complexity

The amplitude of the searched basis state increases linearly in

Grover's Algorithm: Complexity

The amplitude of the searched basis state increases linearly in

$$\frac{1}{\sqrt{N}}.$$

Grover's Algorithm: Complexity

The amplitude of the searched basis state increases linearly in

$$\frac{1}{\sqrt{N}}.$$

Hence, after approximately $\sqrt{N}$ steps, the probability of measuring the correct state is very close to 100%.

Grover's Algorithm: Complexity

We observe:

Grover's Algorithm: Complexity

We observe:

- Unstructured search, classical (Linear search): $O(n)$.

Grover's Algorithm: Complexity

We observe:

- Unstructured search, classical (Linear search): $O(n)$.
- Unstructured search, quantum (Grover): $O(\sqrt{n})$.

Grover's Algorithm: Complexity

We observe:

- Unstructured search, classical (Linear search): $O(n)$.
- Unstructured search, quantum (Grover): $O(\sqrt{n})$.

A quantum computer can solve this problem **quadratically faster**.

Questions?

Thank you.

Images

- Slide 29: Based on [3].
- Slides 34, 37, 42, 44-46 and title slide: Based on [1].

References

[1] Amira Abbas u. a. *Learn Quantum Computation Using Qiskit*. 2020. URL: http://community.qiskit.org/textbook.

[2] Charles H. Bennett u. a. „Strengths and Weaknesses of Quantum Computing". In: *SIAM Journal on Computing* 26.5 (1997), S. 1510–1523. DOI: 10.1137/S0097539796300933.

[3] Caroline Figgatt u. a. „Complete 3-qubit Grover search on a programmable quantum computer". In: *Nature communications* 8.1 (2017), S. 1–9. DOI: 10.1038/s41467-017-01904-7.

Universität Bremen Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

References (cont.)

[4] Lov K. Grover. „A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, S. 212–219. DOI: 10.1145/237814.237866.

[5] Michael A. Nielsen und Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge Univ Press, 9. Dez. 2010.

[6] P.W. Shor. „Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, S. 124–134. DOI: 10.1109/SFCS.1994.365700.