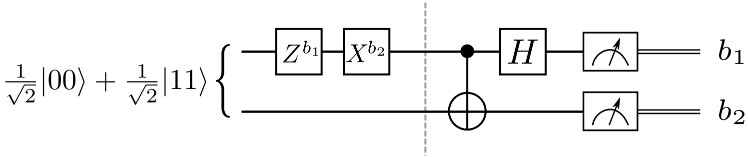
# CPEN 400Q / EECE 571Q Lecture 06
# The Oracle

Thursday 27 January 2022

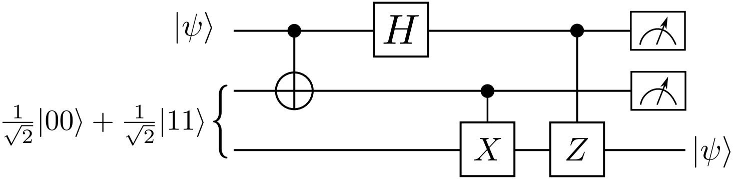
# Announcements

- Assignment 1 due tonight 23:59
  - Remember to format your code and add comments
  - Make a single PR to *your* fork of the repo, not the organization repo
- Assignment 2 will be available Monday at the latest

We implemented superdense coding:




and teleportation:



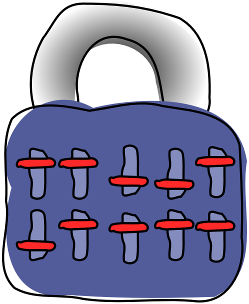
# Learning outcomes

- Define and compute the query complexity of a quantum algorithm
- Apply the phase kickback technique
- Implement Deutsch's algorithm in PennyLane
- Describe the strategy of amplitude amplification

# Oracles, queries, and Deutsch's algorithm

# Motivating problem

Suppose we would like to find the combination for a “binary” lock:

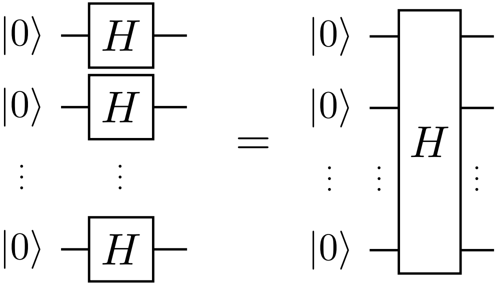


Classically, we would have to try every possible combination. If there are $n$ bits, that's $2^n$ attempts in the worst case. Can we do better with a quantum computer?

Image credit: Codebook node A.1

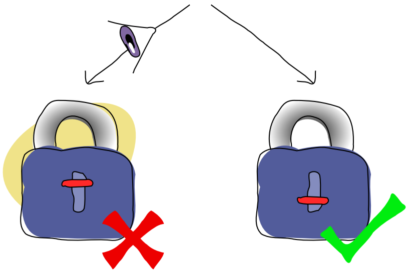
## Idea: use superposition

What if we take $n$ qubits and put them in a superposition with all possible combinations?




Often called the *Hadamard transform*. Let's check that this works...

Measurements are probabilistic - just because we put things into a uniform superposition of states, and our solution is “in” there, doesn’t mean we are any closer to solving our problem.

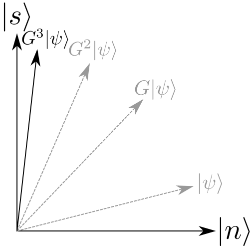


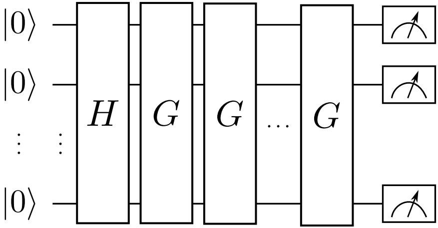
Quantum computers are **NOT** faster because they can “compute everything at the same time.”

Image credit: Codebook node A.1

*Can* we solve this problem better with a quantum computer?

Yes: **amplitude amplification**, and **Grover's algorithm**.



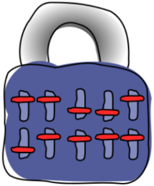




First, we will see some of the algorithmic primitives that are involved, and explore some smaller use cases where we can do better with quantum computing.

# Oracles

## Motivating problem

Suppose we would like to find combination for a "binary" lock:



Classically, we would have to try every possible combination. If there are $n$ bits, that's $2^n$ possible tries. Can we do better with a quantum computer?

Image credit: Codebook node A.1



What is a "try"?

We often express these tries as evaluations of a function that tells us whether we have found the correct answer.

Let

- $\mathbf{x}$ be an $n$-bit string that represents an input to the lock
- $\mathbf{s}$ be the solution to the problem (i.e., the correct combination)

We can represent trying a lock combination as a function:

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} = \mathbf{s} \\ 0 & \text{otherwise.} \end{cases}$$

We don't necessarily care *how* this function gets evaluated, only that it gives us an answer (more specifically, a yes/no answer).

$$f(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} = \mathbf{s} \\ 0 & \text{otherwise.} \end{cases}$$

We consider this function as a black box, or an **oracle**.

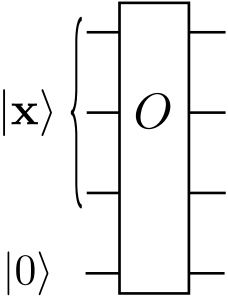
Every time we try a lock combination, we are **querying the oracle**. The amount of queries we make is the **query complexity**.

$O(2^n)$

To solve this problem using quantum computing, we need some circuit that plays the role of the oracle.

Perspective 1: encode the result in the state of an additional qubit.

$$O|\mathbf{x}\rangle|y\rangle = |\mathbf{x}\rangle|\underbrace{y \oplus f(\mathbf{x})}\rangle$$




$$O|000\rangle|0\rangle = |000\rangle|0\rangle$$
$$O|001\rangle|0\rangle = |001\rangle|0\rangle$$
$$O|010\rangle|0\rangle = |010\rangle|0\rangle$$
$$O|011\rangle|0\rangle = |011\rangle|0\rangle$$
$$O|100\rangle|0\rangle = |100\rangle|0\rangle$$
$$O|101\rangle|0\rangle = |101\rangle|0\rangle$$
$$O|110\rangle|0\rangle = |110\rangle|1\rangle$$
$$O|111\rangle|0\rangle = |111\rangle|0\rangle$$

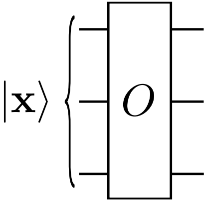
# Quantum oracles

Perspective 2: encode the result in the phase of a qubit.

$f(x)=0 \Rightarrow 1$

$(-1)^{f(x)}$ : $f(x)=1 \Rightarrow -1$

$$O|\mathbf{x}\rangle = \underbrace{(-1)^{f(\mathbf{x})}}|\mathbf{x}\rangle$$




$$
\begin{aligned}
O|000\rangle &= |000\rangle \\
O|001\rangle &= |001\rangle \\
O|010\rangle &= |010\rangle \\
O|011\rangle &= |011\rangle \\
O|100\rangle &= |100\rangle \\
O|101\rangle &= |101\rangle \\
O|110\rangle &= -|110\rangle \\
O|111\rangle &= |111\rangle
\end{aligned}
$$

# Deutsch's algorithm

**Motivation**: You are given access to an oracle and are promised that it implements one of the following 4 functions:

| Name | Action | Name | Action |
|---|---|---|---|
| $f_1$ | $f_1(0) = 0$<br>$f_1(1) = 0$ | $f_2$ | $f_2(0) = 1$<br>$f_2(1) = 1$ |
| $f_3$ | $f_3(0) = 0$<br>$f_3(1) = 1$ | $f_4$ | $f_4(0) = 1$<br>$f_4(1) = 0$ |

"constant"

"balanced"

Functions $f_1$ and $f_2$ are *constant* (same output no matter what the input), and $f_3$ and $f_4$ are balanced.

How many queries do you need to make to the oracle to determine if the function is constant or balanced? (i.e., either one of $f_1/f_2$, or one of $f_3/f_4$).

$f(0) \to 0$

| Name | Action | Name | Action |
|---|---|---|---|
| $f_1$ | $f_1(0) = 0$<br>$f_1(1) = 0$ | $f_2$ | $f_2(0) = 1$<br>$f_2(1) = 1$ |
| $f_3$ | $f_3(0) = 0$<br>$f_3(1) = 1$ | $f_4$ | $f_4(0) = 1$<br>$f_4(1) = 0$ |

**Classical solution:** 2

We always need to query both inputs 0 and 1 to find out the nature of the function.

How many queries do you need to make to the oracle to determine if the function is constant or balanced? (i.e., either one of $f_1/f_2$, or one of $f_3/f_4$).

| Name | Action | Name | Action |
|---|---|---|---|
| $f_1$ | $f_1(0) = 0$<br>$f_1(1) = 0$ | $f_2$ | $f_2(0) = 1$<br>$f_2(1) = 1$ |
| $f_3$ | $f_3(0) = 0$<br>$f_3(1) = 1$ | $f_4$ | $f_4(0) = 1$<br>$f_4(1) = 0$ |

**Quantum solution:** 1
How???

# Phase kickback

$$|0\rangle \;—\!—\!—\!— \;|0\rangle$$
$$|0\rangle \;—\boxed{X}—\boxed{H}—\; |-\rangle$$

The secret lies in something called *phase kickback*.

What happens when we apply a CNOT to the following state?

$$|0\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = |0\rangle|-\rangle$$
$$= \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|01\rangle$$

We get

$$\mathrm{CNOT}\left(\frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|01\rangle\right) = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|01\rangle = |0\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

$$\mathrm{CNOT}\,|0\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = |0\rangle \left( \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

The control qubit is in $|0\rangle$, so it doesn't have any effect on the target qubit.

What happens when we apply a CNOT to this state instead?

$$|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) = \frac{1}{\sqrt{2}}|10\rangle - \frac{1}{\sqrt{2}}|11\rangle$$

We get

$$\begin{aligned}\mathrm{CNOT}\left(\frac{1}{\sqrt{2}}|10\rangle - \frac{1}{\sqrt{2}}|11\rangle\right) &= \frac{1}{\sqrt{2}}|11\rangle - \frac{1}{\sqrt{2}}|10\rangle \\ &= |1\rangle\left(\frac{|1\rangle-|0\rangle}{\sqrt{2}}\right) \\ &= |1\rangle\left(-\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right)\end{aligned}$$

It looks like we've changed the phase of the second qubit.

# Phase kickback

$$= |1\rangle\left(-\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right) = |1\rangle\left(-\frac{|0\rangle}{\sqrt{2}}+\frac{|1\rangle}{\sqrt{2}}\right) = \frac{-|10\rangle}{\sqrt{2}} + \frac{|11\rangle}{\sqrt{2}}$$

But this is a *global* phase, and the math doesn't care which qubit it's attached to. We could equally well write

$$\mathrm{CNOT}\left(|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right) = -\frac{|10\rangle}{\sqrt{2}} + \frac{|11\rangle}{\sqrt{2}} = \underbrace{(-|1\rangle)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)}$$

Now it's as if the *target* qubit has done something to the *control* qubit!

We say that the phase has been "kicked back" from the second qubit to the first.

# Phase kickback

$CNOT\left(|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right)=|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$

$CNOT\left(|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right)=(-|1\rangle)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$

We can write a general version of this effect:

$$CNOT\left(\underset{b=0,\ b=1}{|b\rangle}\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right)=(-1)^b|b\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$$

But what does this have to do with Deutsch's algorithm and figuring out if a function is constant or balanced?

$$O|x\rangle=(-1)^{f(x)}|x\rangle$$

$$CNOT|b\rangle|stuff\rangle=(-1)^b|b\rangle|stuff\rangle$$

This is where our oracle comes in. Suppose we have a black box, $U_f$, that implements any of these four functions, $f$:

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

Setting $|y\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ will allow us to 'extract' the value of $f(0) \oplus f(1)$ with just a single query.

Let's work through the math.

$$U_f|x\rangle = (-1)^{f(x)}|x\rangle \qquad U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

$$U_f|x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \frac{1}{\sqrt{2}}U_f|x\rangle|0\rangle - \frac{1}{\sqrt{2}}U_f|x\rangle|1\rangle$$

$$= \frac{1}{\sqrt{2}}|x\rangle|0 \oplus f(x)\rangle - \frac{1}{\sqrt{2}}|x\rangle|1 \oplus f(x)\rangle$$

If $f(x) = 0$, we get

$$U_f|x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \frac{1}{\sqrt{2}}|x\rangle|0\rangle - \frac{1}{\sqrt{2}}|x\rangle|1\rangle = |x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

If $f(x) = 1$, we get

$$U_f|x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \frac{1}{\sqrt{2}}|x\rangle|1\rangle - \frac{1}{\sqrt{2}}|x\rangle|0\rangle$$

$$= -\left(\frac{1}{\sqrt{2}}|x\rangle|0\rangle - \frac{1}{\sqrt{2}}|x\rangle|1\rangle\right)$$

$$= (-|x\rangle)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

So just like the case of the CNOT where we wrote the general version

$b=0,\ b=1$

$$CNOT\left(|b\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right)=(-1)^b|b\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right),$$

we can write

$$U_f\left(|x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right)=(-1)^{f(x)}|x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$$

Essentially, before the CNOT was just playing the role of $U_f$ for the specific function $f(0)=0, f(1)=1$.

This doesn't look like much on its own - we want to get a *combination* of $f(0)$ and $f(1)$. How can we do this?

$U_f |0\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$
or
$|1\rangle$

By setting $|x\rangle$ to be a superposition!

$$U_f \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$
$$= U_f \frac{|0\rangle}{\sqrt{2}}\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + U_f \frac{|1\rangle}{\sqrt{2}}\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$
$$= (-1)^{f(0)} \frac{|0\rangle}{\sqrt{2}}\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) + (-1)^{f(1)} \frac{|1\rangle}{\sqrt{2}}\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

Let's pull out a phase factor of $(-1)^{f(0)}$, since global phase doesn't matter anyways.

$$(-1)^{f(0)}\frac{|0\rangle}{\sqrt{2}}\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)+(-1)^{f(1)}\frac{|1\rangle}{\sqrt{2}}\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$$

$$=\cancel{(-1)^{f(0)}}\left[\frac{|0\rangle}{\sqrt{2}}\overbrace{\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)}+\underbrace{(-1)^{f(1)}(-1)^{-f(0)}}\frac{|1\rangle}{\sqrt{2}}\overbrace{\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)}\right]$$

$$=\left(\frac{|0\rangle+\underbrace{(-1)^{f(1)-f(0)}}_{+1 \text{ or } -1}|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$$

Now let's look at how this state is different when $f$ is a constant vs. a balanced function.

$$(-1)^{f(1)}(-1)^{-f(0)}=(-1)^{f(1)-f(0)}$$

$$f(0)=f(1)\Rightarrow \overbrace{(-1)^{f(1)-f(0)}}^{\text{constant}}=+1$$

$$f(0)\neq f(1)\Rightarrow \underbrace{(-1)^{f(1)-f(0)}}_{\text{balanced}}=-1$$

# Deutsch's algorithm

If the function is constant, $\overbrace{f(0) \oplus f(1)}^{f(1)-f(0)} = 0$ and the state is

$$U_f \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

$$U_f |+\rangle|-\rangle = |+\rangle|-\rangle \xrightarrow{H \text{ on qubit } 1} |0\rangle|-\rangle$$

But if the function is balanced, $f(0) \oplus f(1) = 1$ and the state is

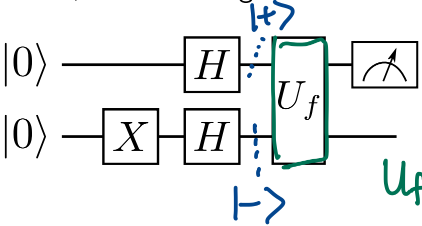
$$U_f \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

$$U_f |+\rangle|-\rangle = |-\rangle|-\rangle \longrightarrow |1\rangle|-\rangle$$

We can measure the first qubit in the *Hadamard basis* to determine exactly the value of $f(0) \oplus f(1)$!

# Deutsch's algorithm

As a circuit, Deutsch's algorithm looks like this:

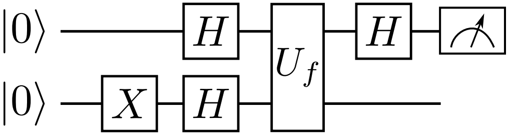



$|+\rangle$ if $f$ is constant
$|-\rangle$ if $f$ is balanced

$U_f\,|+\rangle|-\rangle \to |+\rangle|-\rangle$ const.

$U_f\,|+\rangle|-\rangle \to |-\rangle|-\rangle$ const.

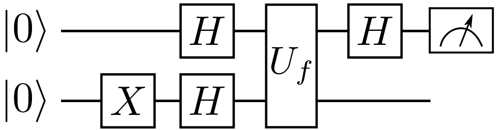
Or equivalently,




$|0\rangle$ if $f$ is constant
$|1\rangle$ if $f$ is balanced

We call $U_f$ just once, but obtain information about the ~~value of~~ relationship between ~~both~~ $f(0)$ and $f(1)$! Let's implement it.

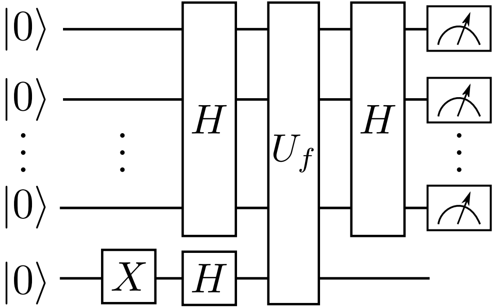
# Generalization: Deutsch-Jozsa algorithm




$|0\rangle$ if $f$ is constant

$|1\rangle$ if $f$ is balanced

We will reimp. Deutsch then start here next time.

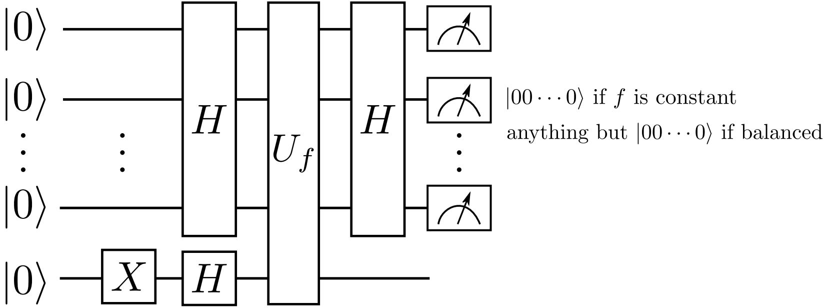



$|00\cdots0\rangle$ if $f$ is constant

anything but $|00\cdots0\rangle$ if balanced

(Challenge: try implementing it yourself to check if this works!)

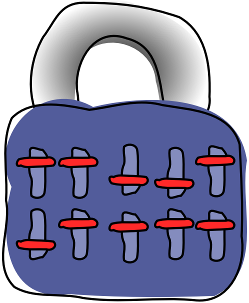
$2^{n-1} + 1$ classical queries in worst case; still only 1 quantum query.



# Grover's algorithm

# Motivation

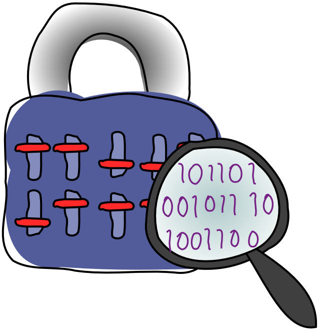
Let's break that lock!



We input the combination to the lock as an $n$-bit (binary) string.
The correct combination is labelled $\mathbf{s}$.

Image credit: Codebook node A.1

How many times do we have to query the oracle to find the solution?




Classically: in the worst case we have to query the oracle $2^n$ times!

The idea behind Grover's search algorithm is to start with a uniform superposition and then *amplify* the amplitude of the state corresponding to the solution.

In other words we want to go from the uniform superposition

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}\rangle$$

to something that looks more like this:

$$|\psi'\rangle = (\text{big number})|\mathbf{s}\rangle + (\text{small number}) \sum_{\mathbf{x} \neq \mathbf{s}} |\mathbf{x}\rangle$$

Q: Why do we want a state of this form?

$$|\psi'\rangle = (\text{big number})|\mathbf{s}\rangle + (\text{small number}) \sum_{\mathbf{x} \neq \mathbf{s}} |\mathbf{x}\rangle$$

Q: Why do we want a state of this form?

$$|\psi'\rangle = (\text{big number})|\mathbf{s}\rangle + (\text{small number}) \sum_{\mathbf{x} \neq \mathbf{s}} |\mathbf{x}\rangle$$

A: When we make a measurement, we will very likely get the solution to our problem!

How do we do this?

# Grover's algorithm: amplitude visualization

We start with the uniform superposition (i.e., perform a Hadamard transform on the qubits).

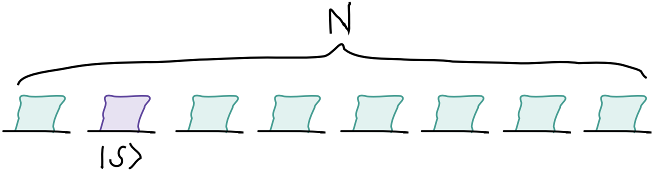



Image credit: Codebook node G.1

# Grover's algorithm: amplitude visualization

If we apply the oracle, we flip the sign of the amplitude of the solution state:

$$|\mathbf{x}\rangle \to (-1)^{f(\mathbf{x})}|\mathbf{x}\rangle$$

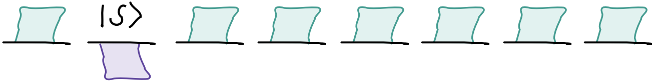



Image credit: Codebook node G.1

Now what? Can't just apply the oracle again... need to do something different.

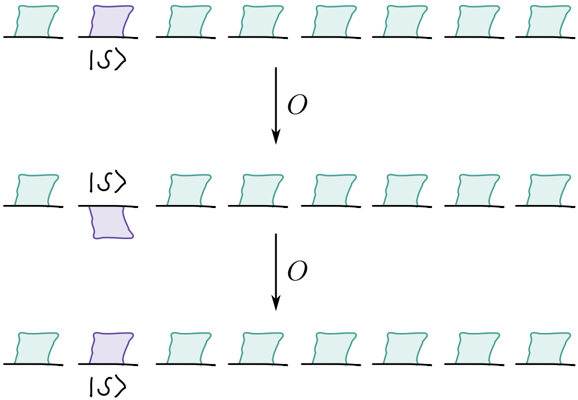
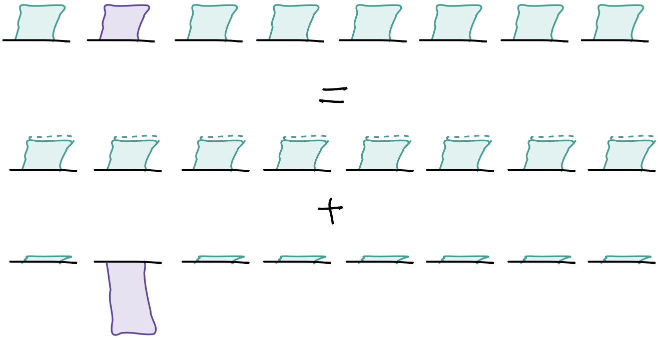



Image credit: Codebook node G.1

# Grover's algorithm: amplitude visualization

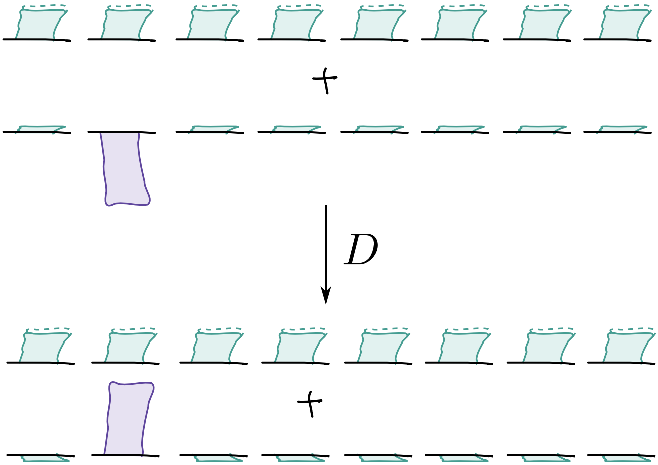
Let's write the amplitudes in a different way:



Why does this help?

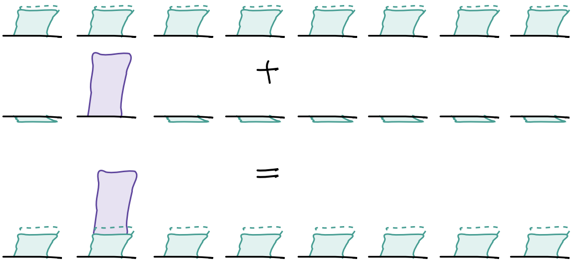
Image credit: Codebook node G.1

What if we had an operation that would flip everything in the second part of the linear combination?



# Grover's algorithm: amplitude visualization

Let's add these back together...




We have "stolen" some amplitude from the other states, and added it to the solution state!

Image credit: Codebook node G.1

# Grover's algorithm: amplitude visualization

Doing this sequence once is one "iteration":

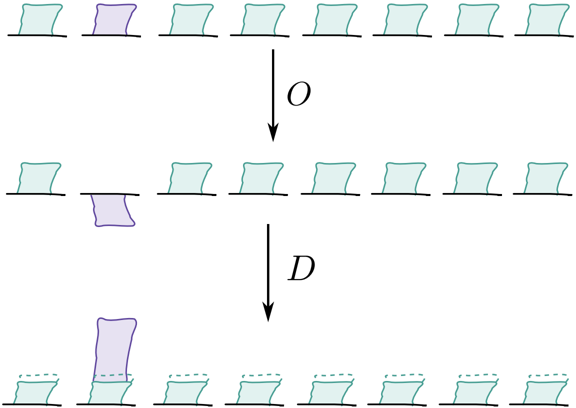
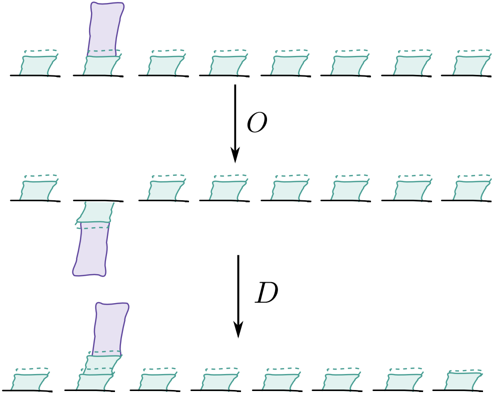


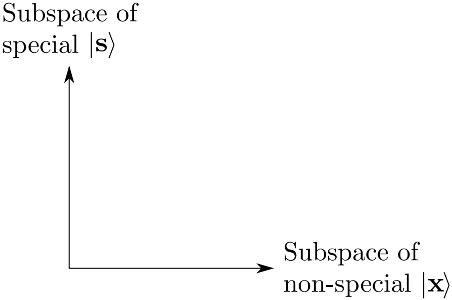

Image credit: Codebook node G.1

If we do it again, we can steal even more amplitude!




Grover's algorithm works by iterating this sequence multiple times until the probability of observing the solution state is maximized.

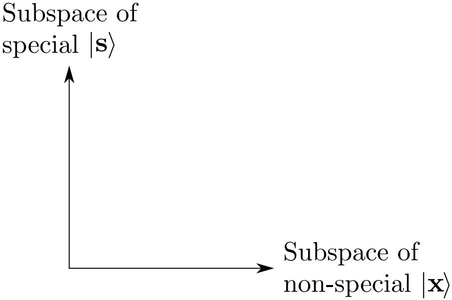
Image credit: Codebook node G.1




We can partition the subspace of all $2^n$ computational basis states into two parts:

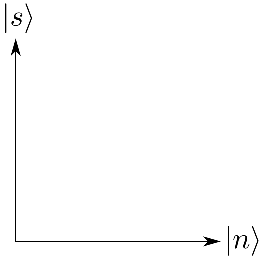
1. The special state $|\mathbf{s}\rangle$

# Grover's algorithm: geometric visualization




We can partition the subspace of all $2^n$ computational basis states into two parts:

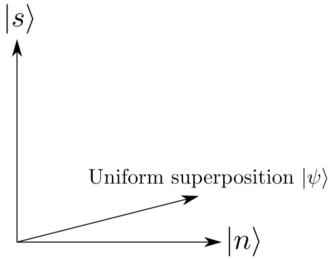
1. The special state $|\mathbf{s}\rangle$
2. All the other states




Let's write these out as superpositions:

$$
\begin{aligned}
|s\rangle &= |\mathbf{s}\rangle \\
|n\rangle &= \frac{1}{\sqrt{2^n - 1}} \sum_{\mathbf{x} \neq \mathbf{s}} |\mathbf{x}\rangle
\end{aligned}
$$

# Grover's algorithm: geometric visualization

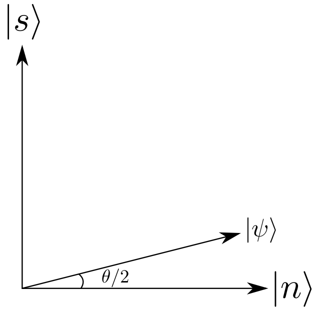



$$
\begin{aligned}
|s\rangle &= |\mathbf{s}\rangle \\
|n\rangle &= \frac{1}{\sqrt{2^n - 1}} \sum_{\mathbf{x} \neq \mathbf{s}} |\mathbf{x}\rangle
\end{aligned}
$$

We can write the uniform superposition in terms of these subspaces:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}}|s\rangle + \frac{\sqrt{2^n - 1}}{\sqrt{2^n}}|n\rangle$$

# Grover's algorithm: geometric visualization



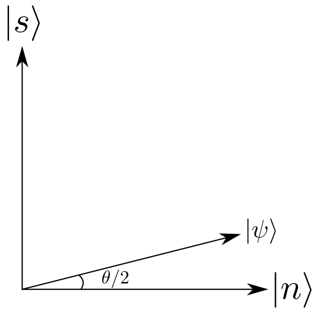

Instead of working with these complicated coeffients:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}}|s\rangle + \frac{\sqrt{2^n - 1}}{\sqrt{2^n}}|n\rangle,$$

let's rexpress them in terms of an angle $\theta$:

$$|\psi\rangle = \sin\left(\frac{\theta}{2}\right)|s\rangle + \cos\left(\frac{\theta}{2}\right)|n\rangle$$

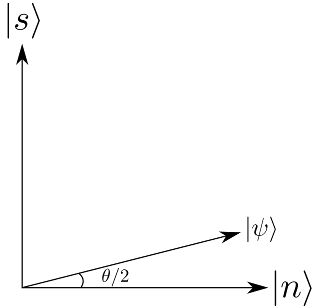
# Grover's algorithm: geometric visualization




Now we want to apply some operations to this state

$$|\psi\rangle = \sin\left(\frac{\theta}{2}\right)|s\rangle + \cos\left(\frac{\theta}{2}\right)|n\rangle$$

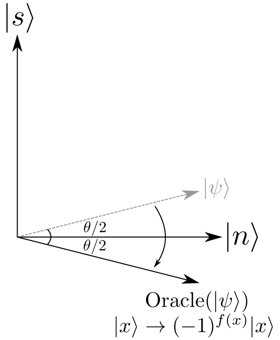
in order to increase the amplitude of $|s\rangle$ while decreasing the amplitude of $|n\rangle$.




We will do this in two steps:

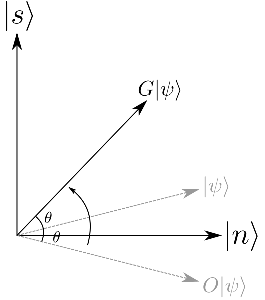
1. Apply the oracle $O$ to 'pick out' the solution
2. Apply a 'diffusion operator' $D$ to adjust the amplitudes.

We will call the product $DO = G$ the *Grover iteration*. Grover's algorithm consists of repeating this procedure many times.




The effect of the oracle, $O|\psi\rangle$ *flips* the amplitudes of the basis states that are special.

We can visualize this as a *reflection about the subspace* of non-special elements.



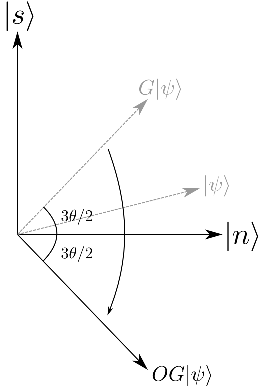

The diffusion operator is a bit less intuitive to interpret* - it performs a *reflection about the uniform superposition* state.

A full Grover iteration $G = DO$ sends

$$G\left(\sin\left(\frac{\theta}{2}\right)|s\rangle + \cos\left(\frac{\theta}{2}\right)|n\rangle\right) = \sin\left(\frac{3\theta}{2}\right)|s\rangle + \cos\left(\frac{3\theta}{2}\right)|n\rangle$$

*Mathematically it looks like $D = 2|+\rangle^{\otimes n}\langle +|^{\otimes n} - \mathbb{1}$.

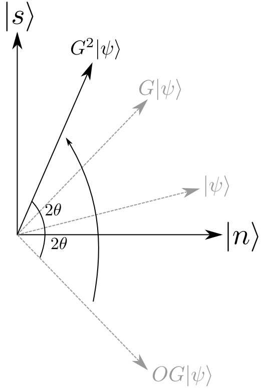
# Grover's algorithm: geometric visualization




Now we repeat this...

Apply the oracle and reflect about the non-special elements.

# Grover's algorithm: geometric visualization

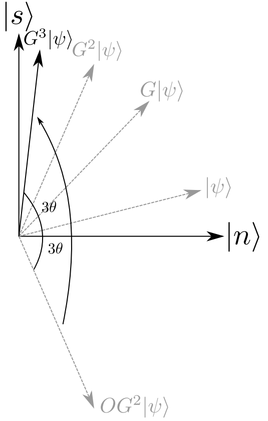



Apply the diffusion operator and reflect about the uniform superposition to boost the amplitude of the special state.

# Grover's algorithm: geometric visualization

After $k$ Grover iterations we will have the state

$$G^k|\psi\rangle = \sin\left(\frac{(2k+1)\theta}{2}\right)|s\rangle + \cos\left(\frac{(2k+1)\theta}{2}\right)|n\rangle$$




It *is* possible to over-rotate! We can differentiate to find the optimal $k$:

$$k \leq \left\lceil \frac{\pi}{4}\sqrt{2^n} \right\rceil$$

After $k$ operations we will be most likely to obtain the special state as our measurement outcome.

Recall that to solve the classical problem we needed to make $2^n$ oracle queries.

Using Grover's algorithm, we only need to make on the order of $\sqrt{2^n}$ queries!

Grover's algorithm provides a *polynomial speedup*, or *square-root* speedup over classical algorithms for searching unsorted spaces.

# Next time

Content:

- Deep dive into implementation of Grover's algorithm

Action items:

1. Finish up Assignment 1

Recommended reading:

- Codebook nodes A.1-A.6, G.1
- Nielsen & Chuang 1.4.2-1.4.4, 6.1