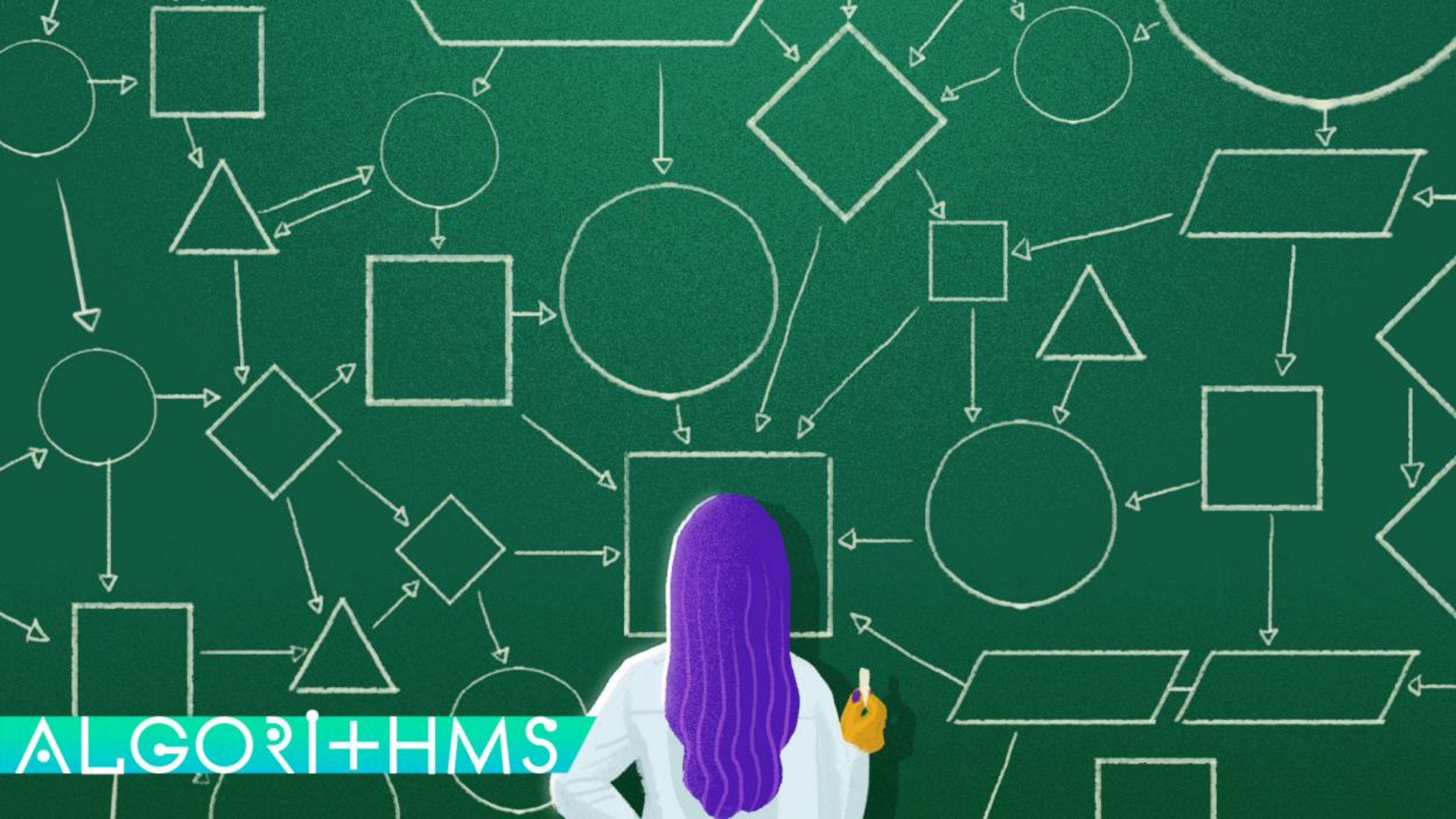
11:00 – 12:30: Quantum computer algorithms: what can we do with them and how can we realize them?

1. Quantum algorithms
    1. Hamiltonian simulation
    2. Deutsch-Jozsa's
    3. QFT
    4. Phase Estimation
    5. Shor's algorithm
    6. Grover's search

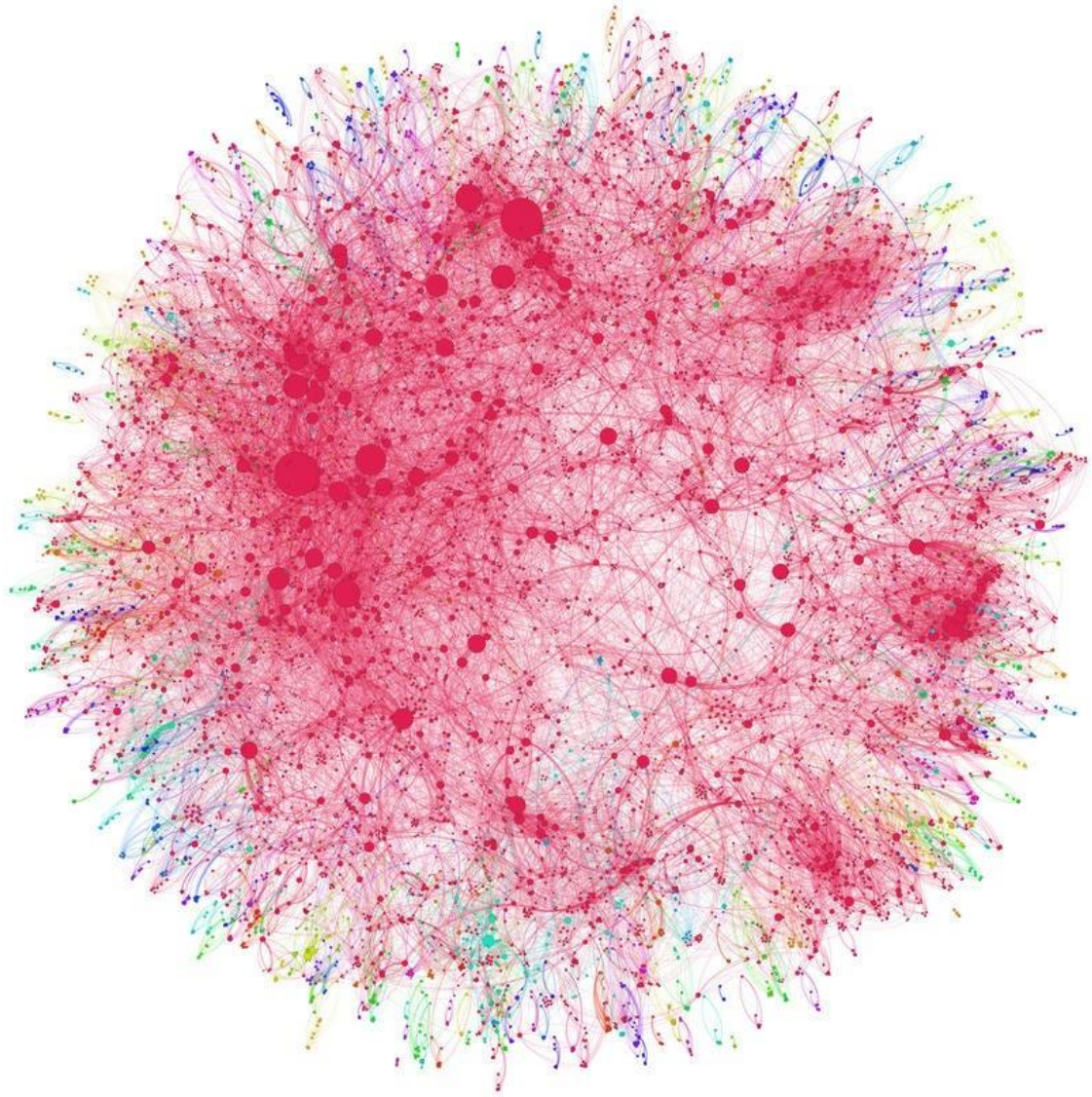
2. Quantum realizations
    1. Quantum error correction
    2. Magic states
    3. Gate Teleportation

Reference

IQC Introduction to Quantum Computing - Petros Wallden
The University of Edinburgh Open Course Materials
http://pwallden.gr/courseiqc.asp


ALGORITHMS

Complexity Theory



Notation & Definitions

- Computational Complexity: Classification of problems according to their difficulty. We usually measure the **amount of resources** (e.g . time, space, gates) used by an algorithm as a function of the **input size** n.

Notation & Definitions

- Computational Complexity: Classification of problems according to their difficulty. We usually measure the **amount of resources** (e.g . time, space, gates) used by an algorithm as a function of the **input size** n.
- Complexity class: Is a set of problems with related resource-based complexity

Notation & Definitions

- Computational Complexity: Classification of problems according to their difficulty. We usually measure the **amount of resources** (e.g . time, space, gates) used by an algorithm as a function of the **input size** n.
- Complexity class: Is a set of problems with related resource-based complexity
- Notation:

- $f(n)$ is in $O(g(n))$ if for some constant m there exists a positive constant such that $f(n) \leq cg(n)$ for all $n \geq m$
- $f(n)$ is in $\Omega(g(n))$ if for some constant m there exists a positive constant c such that $f(n) \geq cg(n)$ for all $n \geq m$
- $f(n)$ is in $\Theta(g(n))$ if for some constant m there exists positive constants $c_1 \leq c_2$ such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq m$

Some Useful Complexity Classes

Decision problems:

1. P: Solved by a deterministic Turing machine in polynomial time (classical deterministic computer solves efficiently)

Some Useful Complexity Classes

Decision problems:

1. P: Solved by a deterministic Turing machine in polynomial time (classical deterministic computer solves efficiently)
2. NP: *Verifiable* in polynomial time by deterministic Turing machine.
   Alternatively, the "yes" instances can be accepted in polynomial time by a non-deterministic Turing machine

Some Useful Complexity Classes

Decision problems:

1. P: Solved by a deterministic Turing machine in polynomial time (classical deterministic computer solves efficiently)
2. NP: *Verifiable* in polynomial time by deterministic Turing machine.
   Alternatively, the "yes" instances can be accepted in polynomial time by a non-deterministic Turing machine
3. PSPACE: Solved by Turing machine using polynomial amount of space (irrespective of the time needed)

Some Useful Complexity Classes

Decision problems:

1. P: Solved by a deterministic Turing machine in polynomial time (classical deterministic computer solves efficiently)
2. NP: *Verifiable* in polynomial time by deterministic Turing machine.
   Alternatively, the "yes" instances can be accepted in polynomial time by a non-deterministic Turing machine
3. PSPACE: Solved by Turing machine using polynomial amount of space (irrespective of the time needed)
4. BPP: Solved by probabilistic TM in poly time with bounded error (classical probabilistic computer solves efficiently)

Some Useful Complexity Classes

Decision problems:

1. P: Solved by a deterministic Turing machine in polynomial time (classical deterministic computer solves efficiently)
2. NP: *Verifiable* in polynomial time by deterministic Turing machine.
   Alternatively, the "yes" instances can be accepted in polynomial time by a non-deterministic Turing machine
3. PSPACE: Solved by Turing machine using polynomial amount of space (irrespective of the time needed)
4. BPP: Solved by probabilistic TM in poly time with bounded error (classical probabilistic computer solves efficiently)
5. BQP: Solved by probabilistic *quantum computer* in poly time with bounded error (quantum computer solves efficiently)

Complexity Classes Relations

Proven Relations:

- $\mathrm{BQP} \subseteq \mathrm{PSPACE}$. Problems solved by quantum computers can be solved (potentially inefficiently) by classical computers

Complexity Classes Relations

Proven Relations:

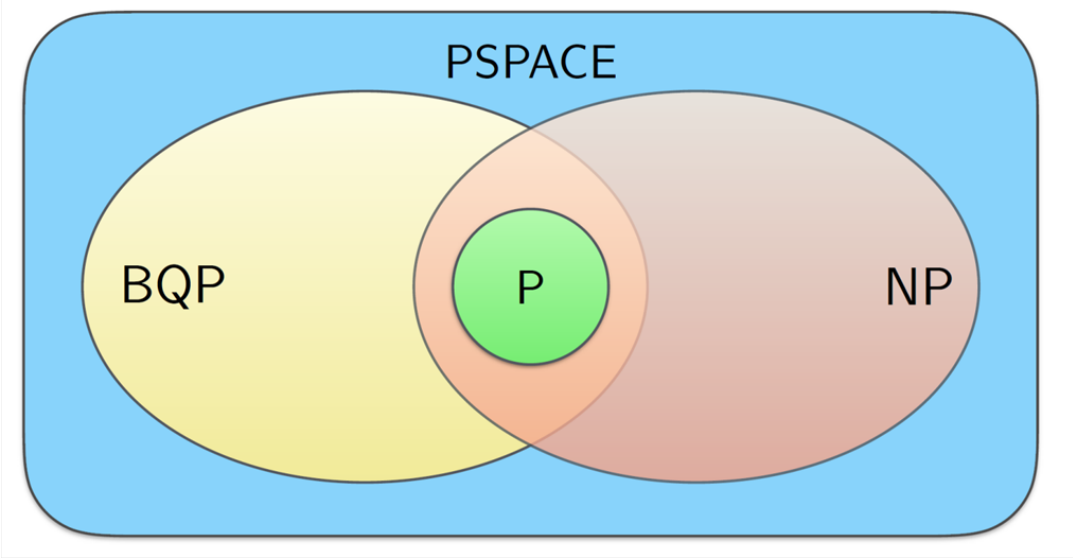
- $\mathrm{BQP} \subseteq \mathrm{PSPACE}$. Problems solved by quantum computers can be solved (potentially inefficiently) by classical computers
- $\mathrm{BPP} \subseteq \mathrm{BQP}$. Quantum computers are at least as efficient as probabilistic Turing machines

Complexity Classes Relations

Proven Relations:

- $\mathrm{BQP} \subseteq \mathrm{PSPACE}$. Problems solved by quantum computers can be solved (potentially inefficiently) by classical computers
- $\mathrm{BPP} \subseteq \mathrm{BQP}$. Quantum computers are at least as efficient as probabilistic Turing machines

Conjectured Relations: (based on other plausible assumptions)



Complexity Classes Relations

Proven Relations:

- $\mathrm{BQP} \subseteq \mathrm{PSPACE}$. Problems solved by quantum computers can be solved (potentially inefficiently) by classical computers
- $\mathrm{BPP} \subseteq \mathrm{BQP}$. Quantum computers are at least as efficient as probabilistic Turing machines

Conjectured Relations: (based on other plausible assumptions)

1. There are problems outside NP that quantum computers **can** solve
2. There are problems in NP that quantum computers **cannot** solve (therefore NP-complete problems should be outside BQP)

The Oracle Model

- We are given a classical gate corresponding to an unknown function f as a **black box** (oracle)

The Oracle Model

- We are given a classical gate corresponding to an unknown function f as a **black box** (oracle)

- Access: Query the oracle, i.e. insert x and obtain $f(x)$

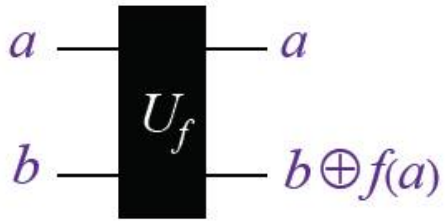
The Oracle Model

- We are given a classical gate corresponding to an unknown function f as a **black box** (oracle)

- Access: Query the oracle, i.e. insert x and obtain $f(x)$
- Goal: Determine properties of the function f with the fewest queries to the oracle

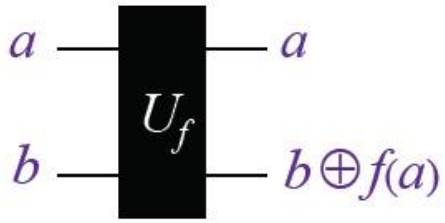
The Quantum Oracle Model

- We are given a quantum gate corresponding to an unknown classical function f as a **black box** (oracle) acting on two qubits in the following way:



The Quantum Oracle Model

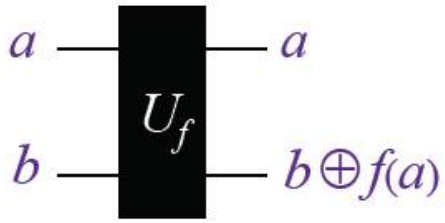
- We are given a quantum gate corresponding to an unknown classical function f as a **black box** (oracle) acting on two qubits in the following way:



- Access: Query the quantum oracle, i.e. insert $|a\rangle |b\rangle$ and obtain $|a\rangle |b \oplus f(a)\rangle$

The Quantum Oracle Model

- We are given a quantum gate corresponding to an unknown classical function f as a **black box** (oracle) acting on two qubits in the following way:



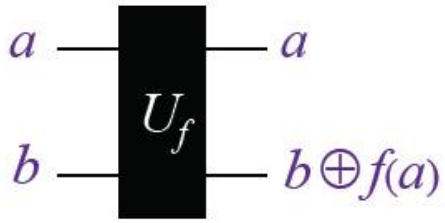
- Access: Query the quantum oracle, i.e. insert $|a\rangle |b\rangle$ and obtain $|a\rangle |b \oplus f(a)\rangle$

  By linearity, we can also query in superposition:

$$\sum_{a,b} C_{a,b} |a\rangle |b\rangle \rightarrow \sum_{a,b} C_{a,b} |a\rangle |b \oplus f(a)\rangle$$

The Quantum Oracle Model

- We are given a quantum gate corresponding to an unknown classical function f as a **black box** (oracle) acting on two qubits in the following way:



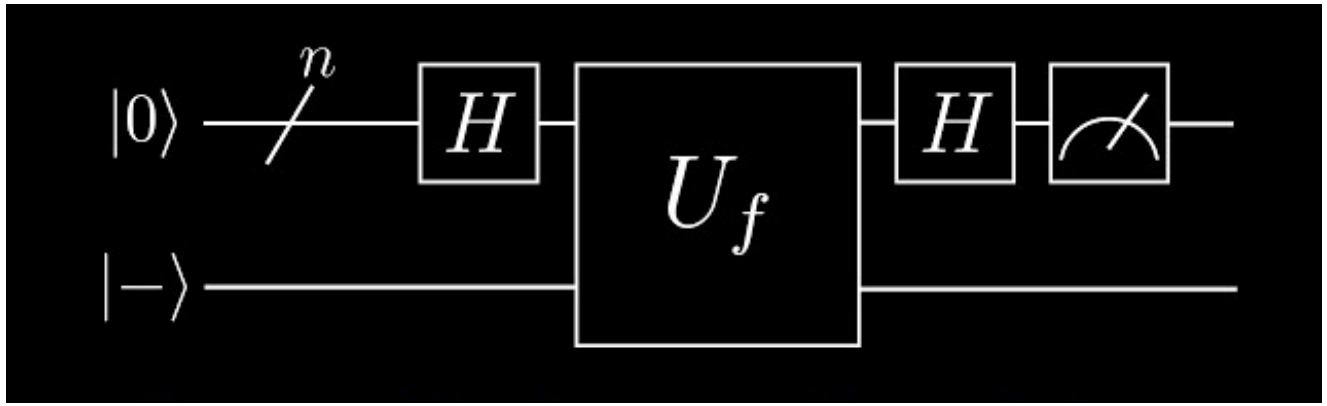
- Access: Query the quantum oracle, i.e. insert $|a\rangle |b\rangle$ and obtain $|a\rangle |b \oplus f(a)\rangle$

  By linearity, we can also query in superposition:

$$\sum_{a,b} C_{a,b} |a\rangle |b\rangle \rightarrow \sum_{a,b} C_{a,b} |a\rangle |b \oplus f(a)\rangle$$

- Goal: Determine properties of the classical function f with the fewest queries to the quantum oracle

Deutsch-Jozsa's quantum algorithm



The Deutsch - Jozsa Algorithm

- Inspiration for Shor's and Grover's algorithms
- Initial protocol by **Deutsch** 1985, improved by **Jozsa**. Current version, is result of further research (**Cleve**, **Ekert**, **Macchiavello** and **Mosca**)

The Deutsch - Jozsa Algorithm

- **Input:** A boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$

The Deutsch - Jozsa Algorithm

- **Input:** A boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$
- **Promise:** The function is either **constant** or **balanced**

The Deutsch - Jozsa Algorithm

- **Input:** A boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$
- **Promise:** The function is either **constant** or **balanced**

  Constant: $f(x) = c \ \forall \ x$ and $c = 0$ or 1

  Balanced: $|f^{-1}(0)| = |f^{-1}(1)|$ i.e. half inputs give 0 and half give 1

The Deutsch - Jozsa Algorithm

- **Input:** A boolean function $f : \{0,1\}^n \to \{0,1\}$
- **Promise:** The function is either **constant** or **balanced**

  Constant: $f(x) = c \ \forall \ x$ and $c = 0$ or 1

  Balanced: $|f^{-1}(0)| = |f^{-1}(1)|$ i.e. half inputs give 0 and half give 1
- **Output:** **Determine** whether the function is **constant** or is **balanced** with the **fewest queries**

The Deutsch - Jozsa Algorithm

- **Input:** A boolean function $f : \{0,1\}^n \to \{0,1\}$
- **Promise:** The function is either **constant** or **balanced**

  Constant: $f(x) = c \ \forall \ x$ and $c = 0$ or 1

  Balanced: $|f^{-1}(0)| = |f^{-1}(1)|$ i.e. half inputs give 0 and half give 1
- **Output:** **Determine** whether the function is **constant** or is **balanced** with the **fewest queries**
- **Performance:**
  1. Classical: To know with certainty we need at least $2^n/2 + 1$ queries
  2. Quantum: With a **single** oracle query

The Deutsch - Jozsa Algorithm

- Recall that U_f is defined as:

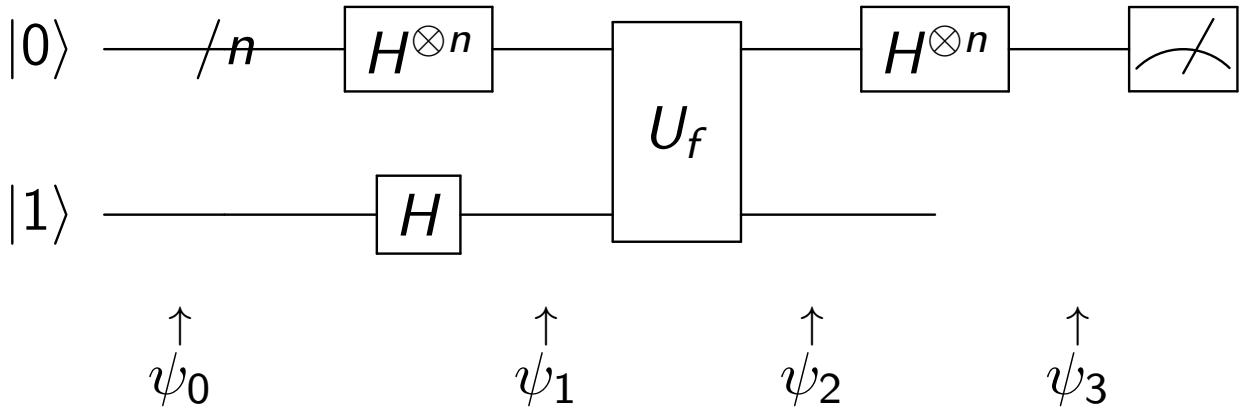
$$\sum_{x,y} C_{x,y} |x\rangle |y\rangle \rightarrow \sum_{x,y} C_{x,y} |x\rangle |y \oplus f(x)\rangle$$

The Deutsch - Jozsa Algorithm

- Recall that U_f is defined as:

$$\sum_{x,y} C_{x,y} |x\rangle |y\rangle \rightarrow \sum_{x,y} C_{x,y} |x\rangle |y \oplus f(x)\rangle$$

- The Quantum Circuit of the algorithm is given by:



The Deutsch - Jozsa Algorithm

Property:

$$H|x\rangle = \frac{1}{\sqrt{2}} \sum_{y \in \{0,1\}} (-1)^{xy} |y\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + (-1)^x |1\rangle \right)$$

The Deutsch - Jozsa Algorithm

Property:

$H|x\rangle = \frac{1}{\sqrt{2}} \sum_{y\in\{0,1\}} (-1)^{xy} |y\rangle = \frac{1}{\sqrt{2}} (|0\rangle + (-1)^x |1\rangle)$

The protocol's steps:

1. $|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle$. Note that the first register refers to string of n qubits, while the second register is a single qubit.

The Deutsch - Jozsa Algorithm

Property:

$H|x\rangle = \frac{1}{\sqrt{2}} \sum_{y\in\{0,1\}} (-1)^{xy} |y\rangle = \frac{1}{\sqrt{2}} (|0\rangle + (-1)^x |1\rangle)$

The protocol's steps:

1. $|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle$. Note that the first register refers to string of n qubits, while the second register is a single qubit.
2. Apply H to all qubits:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle (|0\rangle - |1\rangle)$$

The Deutsch - Jozsa Algorithm

Property:

$H|x\rangle = \frac{1}{\sqrt{2}} \sum_{y\in\{0,1\}} (-1)^{xy} |y\rangle = \frac{1}{\sqrt{2}} (|0\rangle + (-1)^x |1\rangle)$

The protocol's steps:

1. $|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle$. Note that the first register refers to string of n qubits, while the second register is a single qubit.
2. Apply H to all qubits:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle (|0\rangle - |1\rangle)$$

3. Apply the oracle U_f:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle (|f(x)\rangle - |1 \oplus f(x)\rangle)$$

The Deutsch - Jozsa Algorithm

Property:

$H|x\rangle = \frac{1}{\sqrt{2}} \sum_{y\in\{0,1\}} (-1)^{xy} |y\rangle = \frac{1}{\sqrt{2}} (|0\rangle + (-1)^x |1\rangle)$

The protocol's steps:

1. $|\psi_0\rangle = |0\rangle^{\otimes n} |1\rangle$. Note that the first register refers to string of n qubits, while the second register is a single qubit.
2. Apply H to all qubits:

$$|\psi_1\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle (|0\rangle - |1\rangle)$$

3. Apply the oracle U_f:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} |x\rangle (|f(x)\rangle - |1 \oplus f(x)\rangle)$$

it can be rewritten as:

$$|\psi_2\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle (|0\rangle - |1\rangle)$$

4. Apply $H^{\otimes n}$ to the first n qubits:

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{y=0}^{2^n-1} \left(\sum_{x=0}^{2^n-1} (-1)^{f(x)} (-1)^{x \cdot y} \right) |y\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)$$

where $x \cdot y = x_0 y_0 \oplus x_1 y_1 \oplus \cdots \oplus x_{n-1} y_{n-1}$ is the sum of the bitwise product.

4. Apply $H^{\otimes n}$ to the first n qubits:

$$|\psi_3\rangle = \frac{1}{2^n}\sum_{y=0}^{2^n-1}\left(\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{x\cdot y}\right)|y\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

where $x \cdot y = x_0y_0 \oplus x_1y_1 \oplus \cdots \oplus x_{n-1}y_{n-1}$ is the sum of the bitwise product.

5. We measure the first n qubits in the computational basis and we examine the probability of obtaining all zero's ($|0\rangle^{\otimes n}$):

$$p(0) = |\frac{1}{2^n}\sum_{0}^{2^n-1}(-1)^{f(x)}|^2 \tag{1}$$

4. Apply $H^{\otimes n}$ to the first n qubits:

$$|\psi_3\rangle = \frac{1}{2^n}\sum_{y=0}^{2^n-1}\left(\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{x\cdot y}\right)|y\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

where $x \cdot y = x_0y_0 \oplus x_1y_1 \oplus \cdots \oplus x_{n-1}y_{n-1}$ is the sum of the bitwise product.

5. We measure the first n qubits in the computational basis and we examine the probability of obtaining all zero's ($|0\rangle^{\otimes n}$):

$$p(0) = |\frac{1}{2^n}\sum_{0}^{2^n-1}(-1)^{f(x)}|^2 \tag{1}$$

If $f(x)$ is constant, all terms have the same sign and Eq. (1) gives $p(0) = 1$

4. Apply $H^{\otimes n}$ to the first n qubits:

$$|\psi_3\rangle = \frac{1}{2^n}\sum_{y=0}^{2^n-1}\left(\sum_{x=0}^{2^n-1}(-1)^{f(x)}(-1)^{x\cdot y}\right)|y\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

where $x \cdot y = x_0y_0 \oplus x_1y_1 \oplus \cdots \oplus x_{n-1}y_{n-1}$ is the sum of the bitwise product.

5. We measure the first n qubits in the computational basis and we examine the probability of obtaining all zero's ($|0\rangle^{\otimes n}$):

$$p(0) = |\frac{1}{2^n}\sum_{0}^{2^n-1}(-1)^{f(x)}|^2 \qquad (1)$$

If $f(x)$ is constant, all terms have the same sign and Eq. (1) gives $p(0) = 1$
If $f(x)$ is balanced, half terms are $+1$ and half terms are -1 resulting to Eq. (1) giving $p(0) = 0$

The Deutsch - Jozsa Algorithm

- The algorithm is **deterministic**. Belongs to EQP (Exact Quantum Polynomial time) which is the quantum version of P (rather than in BQP which is the quantum version of BPP).

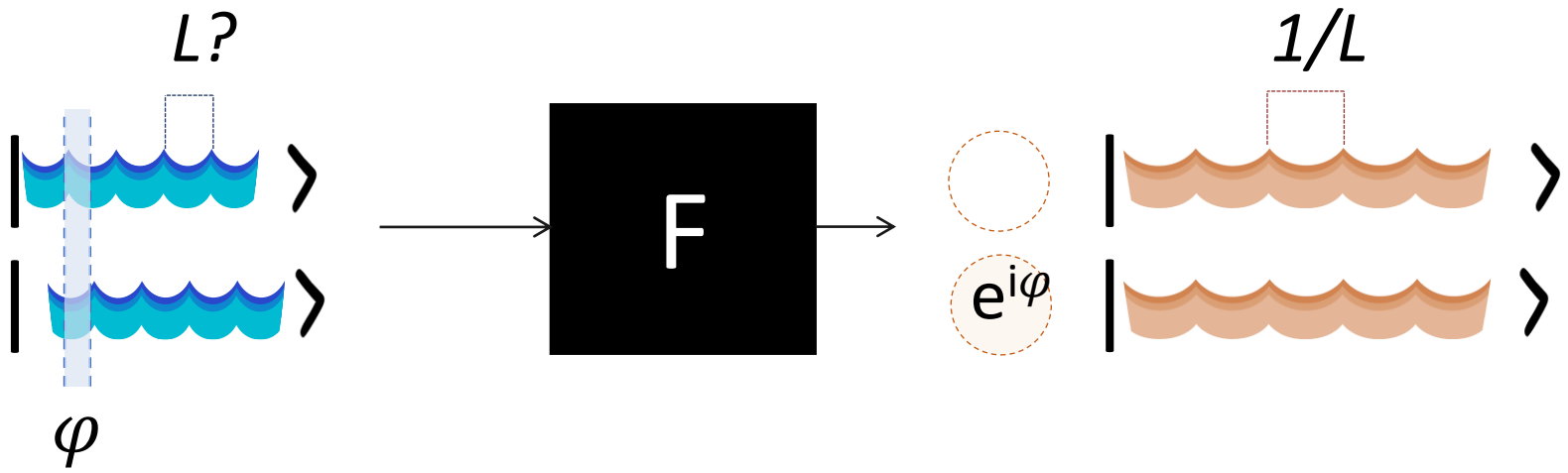
The Deutsch - Jozsa Algorithm

- The algorithm is **deterministic**. Belongs to EQP (Exact Quantum Polynomial time) which is the quantum version of P (rather than in BQP which is the quantum version of BPP).
- It constitutes the first **exponential quantum speed-up**. From exponential many oracle calls for P algorithms, we succeed with a single oracle query in EQP!

The Deutsch - Jozsa Algorithm

- The algorithm is **deterministic**. Belongs to EQP (Exact Quantum Polynomial time) which is the quantum version of P (rather than in BQP which is the quantum version of BPP).
- It constitutes the first **exponential quantum speed-up**. From exponential many oracle calls for P algorithms, we succeed with a single oracle query in EQP!
- It does not give a speed-up compared to BPP, since if we allow for (small) probability of failure, there exist efficient classical algorithms with constant oracle calls (example?)

Quantum Fourier Transform



Quantum Fourier Transform

- **Definition** (quantum): The **Quantum Fourier Transform** (QFT) is a unitary operation F that performs DFT to the **amplitudes** of a quantum state:

$$F \sum_{x=0}^{N-1} a_x |x\rangle = \sum_{y=0}^{N-1} b_y |y\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} a_x \exp(2\pi i xy/N) |y\rangle$$

where the amplitudes a_x, b_y are related as in DFT.

Quantum Fourier Transform

- **Definition** (quantum): The **Quantum Fourier Transform** (QFT) is a unitary operation F that performs DFT to the **amplitudes** of a quantum state:

$$F \sum_{x=0}^{N-1} a_x |x\rangle = \sum_{y=0}^{N-1} b_y |y\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} a_x \exp(2\pi i xy/N) |y\rangle$$

  where the amplitudes a_x, b_y are related as in DFT.
- To determine a quantum operation it suffices to know how it acts on computational basis state and then extend by linearity

Quantum Fourier Transform

- **Definition** (quantum): The **Quantum Fourier Transform** (QFT) is a unitary operation F that performs DFT to the **amplitudes** of a quantum state:

$$F \sum_{x=0}^{N-1} a_x |x\rangle = \sum_{y=0}^{N-1} b_y |y\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} a_x \exp(2\pi i xy/N) |y\rangle$$

  where the amplitudes a_x, b_y are related as in DFT.
- To determine a quantum operation it suffices to know how it acts on computational basis state and then extend by linearity
- The QFT acts as (note that $N = 2^n$):

$$\begin{aligned} F |x_1 x_2 \cdots x_n\rangle \quad = \quad & \frac{1}{\sqrt{N}} \left(|0\rangle + e^{2\pi i [0.x_n]} |1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i [0.x_{n-1} x_n]} |1\rangle\right) \otimes \\ & \otimes \cdots \otimes \left(|0\rangle + e^{2\pi i [0.x_1 x_2 \cdots x_n]} |1\rangle\right) \end{aligned} \tag{4}$$

Quantum Fourier Transform

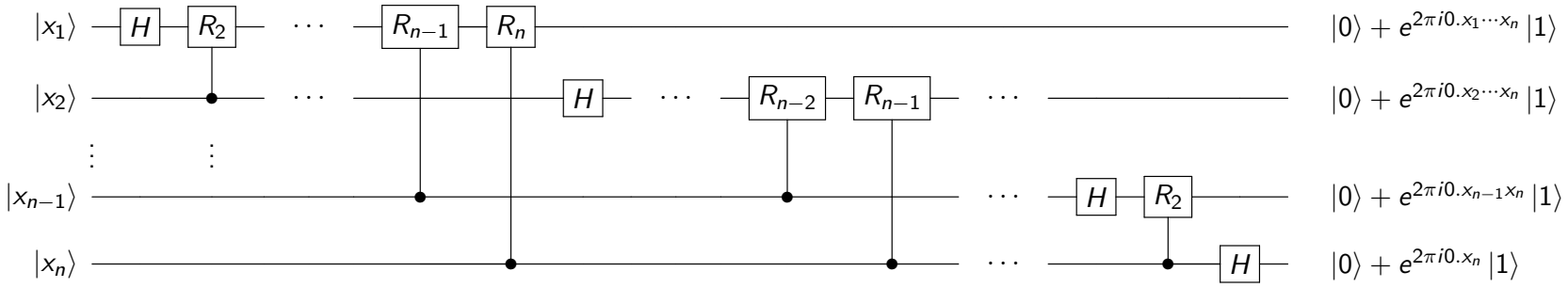
- We will use Eq. (4) as **definition** for QFT

  Is not hard to see that it is essentially the same with Eq. (3) of the DFT (see Appendix)

Quantum Fourier Transform

- We will use Eq. (4) as **definition** for QFT

  Is not hard to see that it is essentially the same with Eq. (3) of the DFT (see Appendix)
- The Quantum Circuit for F is:

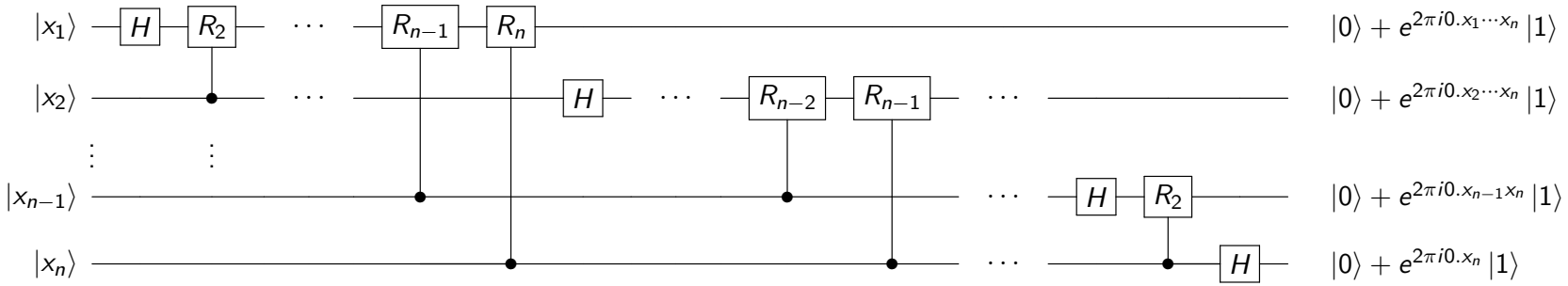


where the gate $R_k := \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$ and we omitted a swap of the qubits and a factor $\frac{1}{\sqrt{2}}$ at the end of the circuit

Quantum Fourier Transform

- We will use Eq. (4) as **definition** for QFT

  Is not hard to see that it is essentially the same with Eq. (3) of the DFT (see Appendix)
- The Quantum Circuit for F is:



  where the gate $R_k := \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$ and we omitted a swap of the qubits and a factor $\frac{1}{\sqrt{2}}$ at the end of the circuit
- It is simple to see that F is unitary since the circuit consists of unitary gates (see also Appendix)

Quantum Fourier Transform

Example: Three qubits

$$F\left|x_1x_2x_3\right\rangle = \left|\psi_1\psi_2\psi_3\right\rangle$$

$$\left|\psi_1\right\rangle = \frac{1}{\sqrt{2}}\left(\left|0\right\rangle + e^{2\pi i[0.x_3]}\left|1\right\rangle\right)$$
$$\left|\psi_2\right\rangle = \frac{1}{\sqrt{2}}\left(\left|0\right\rangle + e^{2\pi i[0.x_2x_3]}\left|1\right\rangle\right)$$
$$\left|\psi_3\right\rangle = \frac{1}{\sqrt{2}}\left(\left|0\right\rangle + e^{2\pi i[0.x_1x_2x_3]}\left|1\right\rangle\right)$$

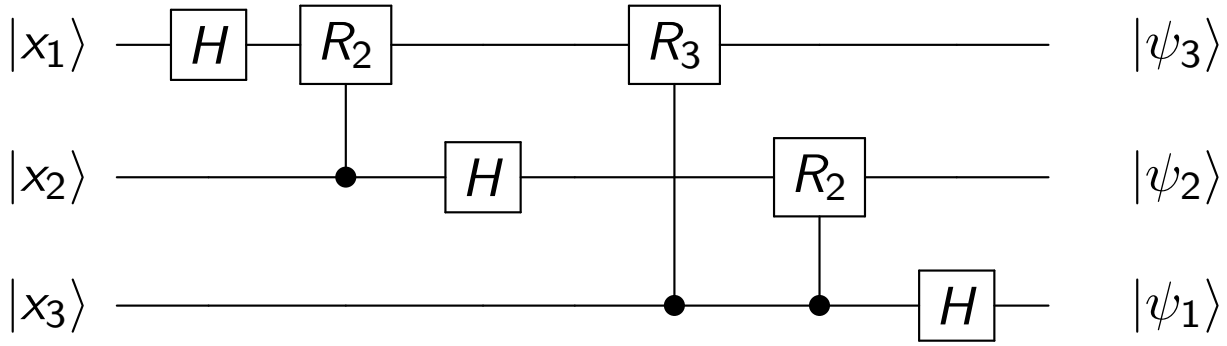
Quantum Fourier Transform

Example: Three qubits

$$F\left|x_1x_2x_3\right\rangle = \left|\psi_1\psi_2\psi_3\right\rangle$$

$$\left|\psi_1\right\rangle = \frac{1}{\sqrt{2}}\left(\left|0\right\rangle + e^{2\pi i[0.x_3]}\left|1\right\rangle\right)$$
$$\left|\psi_2\right\rangle = \frac{1}{\sqrt{2}}\left(\left|0\right\rangle + e^{2\pi i[0.x_2x_3]}\left|1\right\rangle\right)$$
$$\left|\psi_3\right\rangle = \frac{1}{\sqrt{2}}\left(\left|0\right\rangle + e^{2\pi i[0.x_1x_2x_3]}\left|1\right\rangle\right)$$

The corresponding circuit is:



Quantum Fourier Transform

- The number of gates in QFT (including the final swaps) is $\Theta(n^2)$

Quantum Fourier Transform

- The number of gates in QFT (including the final swaps) is $\Theta(n^2)$
- To implement the **classical Fast Fourier Transform** $\Theta(n2^n)$ gates are needed

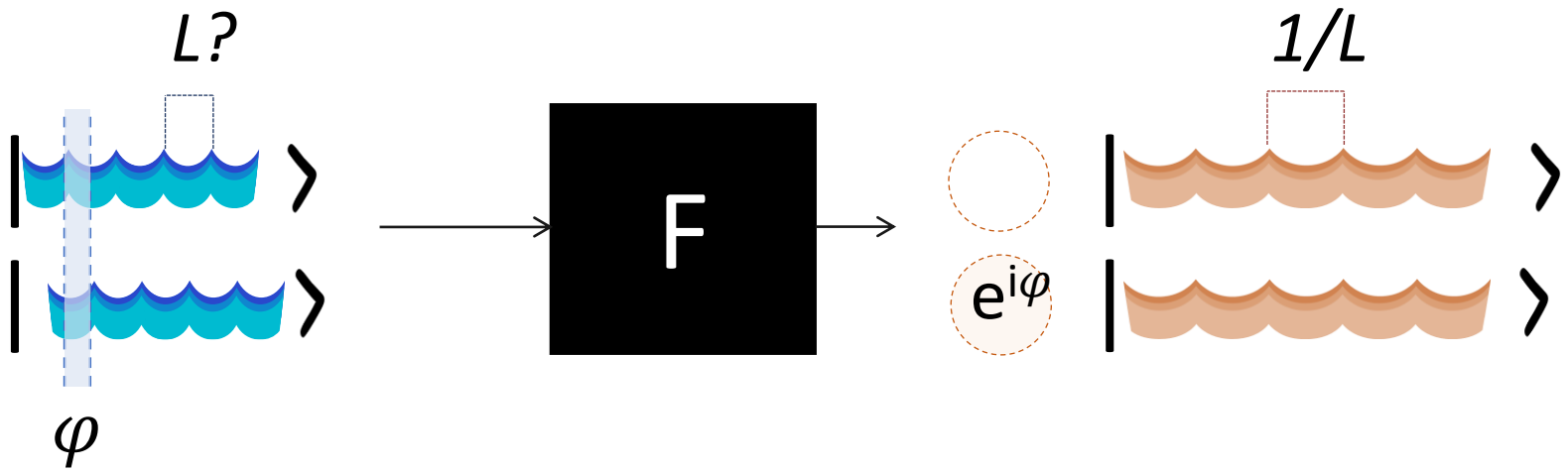
Quantum Fourier Transform

- The number of gates in QFT (including the final swaps) is $\Theta(n^2)$
- To implement the **classical Fast Fourier Transform** $\Theta(n2^n)$ gates are needed
- It appears we obtained an exponential speed-up for a task (DFT) that has many application

Quantum Fourier Transform

- The number of gates in QFT (including the final swaps) is $\Theta(n^2)$
- To implement the **classical Fast Fourier Transform** $\Theta(n2^n)$ gates are needed
- It appears we obtained an exponential speed-up for a task (DFT) that has many application
- However, we **cannot access**(read-out) the amplitudes of a quantum state, so we cannot extract the classical values of the DFT.
- To achieve real speed-up, we need to use QFT as part of larger algorithm (see later!)

Quantum Phase Estimation



Phase Estimation

Setting:

1. A **unitary operation** U on m-qubits, along with a black box (oracle) that can perform $\wedge U^j$ for any integer j
2. An **eigenstate** $|u\rangle$ of U with eigenvalue $e^{2\pi i \phi_u}$. i.e.

$$U|u\rangle = e^{2\pi i \phi_u}|u\rangle$$

3. Sufficient (n or if we want higher probability of success $t \in O(n)$) **ancilla qubits** initialised to $|0\rangle$

Phase Estimation

Setting:

1. A **unitary operation** U on m-qubits, along with a black box (oracle) that can perform $\wedge U^j$ for any integer j
2. An **eigenstate** $|u\rangle$ of U with eigenvalue $e^{2\pi i \phi_u}$. i.e.

$$U|u\rangle = e^{2\pi i \phi_u}|u\rangle$$

3. Sufficient (n or if we want higher probability of success $t \in O(n)$) **ancilla qubits** initialised to $|0\rangle$

Result: An n-bit approximation $\bar{\phi_u}$ of the phase ϕ_u

Phase Estimation

Setting:

1. A **unitary operation** U on m-qubits, along with a black box (oracle) that can perform $\wedge U^j$ for any integer j
2. An **eigenstate** $|u\rangle$ of U with eigenvalue $e^{2\pi i \phi_u}$. i.e.

$$U|u\rangle = e^{2\pi i \phi_u}|u\rangle$$

3. Sufficient (n or if we want higher probability of success $t \in O(n)$) **ancilla qubits** initialised to $|0\rangle$

Result: An n-bit approximation $\bar{\phi_u}$ of the phase ϕ_u

Cost: $O(t^2)$ operations and one call to the $\wedge U^j$ oracle for each j. Succeeds with probability $1 - \epsilon$

Phase Estimation

Setting:

1. A **unitary operation** U on m-qubits, along with a black box (oracle) that can perform $\wedge U^j$ for any integer j
2. An **eigenstate** $|u\rangle$ of U with eigenvalue $e^{2\pi i \phi_u}$. i.e.

$$U|u\rangle = e^{2\pi i \phi_u}|u\rangle$$

3. Sufficient (n or if we want higher probability of success $t \in O(n)$) **ancilla qubits** initialised to $|0\rangle$

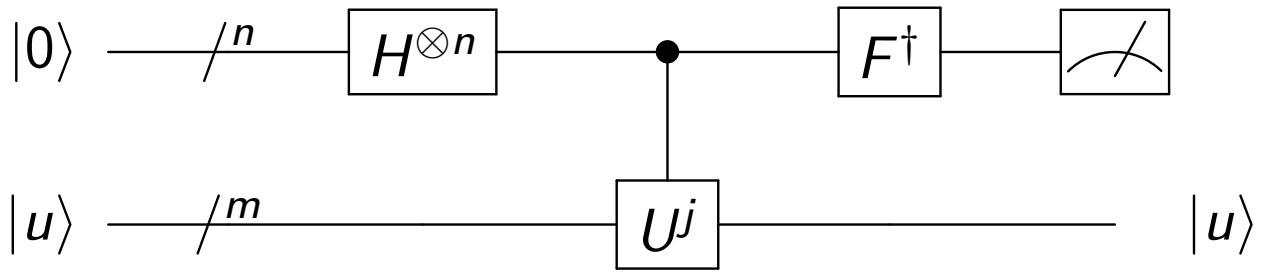
Result: An n-bit approximation $\bar{\phi_u}$ of the phase ϕ_u

Cost: $O(t^2)$ operations and one call to the $\wedge U^j$ oracle for each j. Succeeds with probability $1 - \epsilon$

Note: Since U is unitary, any eigenvalue $U|u\rangle = e_u|u\rangle$ implies that $1 = \langle u| U^\dagger U |u\rangle = |e_u|^2$, and thus e_u is a unit norm complex number, i.e. of the form $e^{2\pi i \phi_u}$ for some ϕ_u.

Phase Estimation

The circuit for Phase Estimation is given:

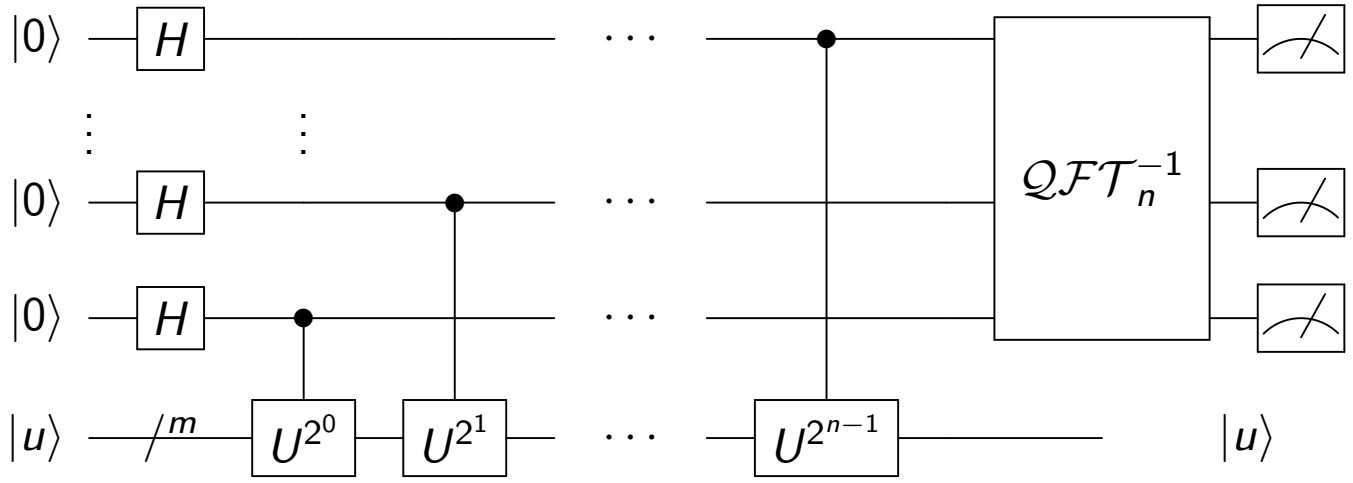


Remarks:

- We drop the subscript u in the phase ϕ
- We assume that $\phi = 0.\phi_1\phi_2\cdots\phi_n\cdots$. Any integer part is irrelevant, since ϕ appears in the expression $e^{2\pi i\phi}$
- We will obtain a n-bit value $\bar{\phi}$ which is an approximation to the phase ϕ
- If ϕ has less or equal to n-digits we obtain the exact result.
- We focus on this case here. One can show that for the general case, the n-bit value $\bar{\phi}$ we obtain is the best such approximation for the phase ϕ

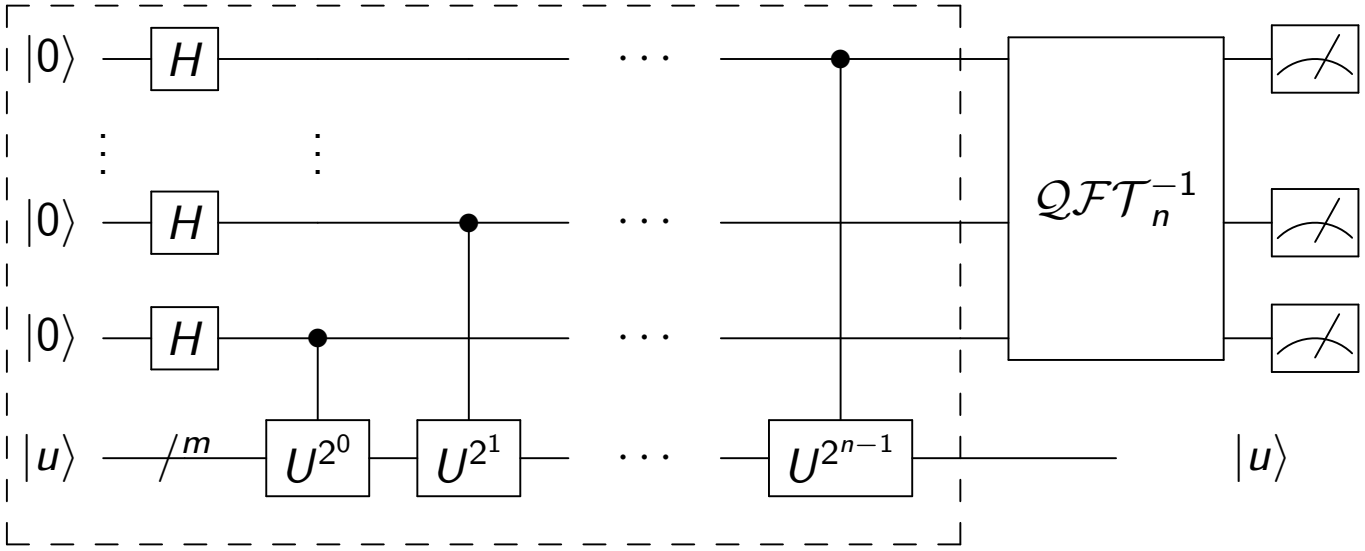
Phase Estimation

The circuit in greater detail is the following:



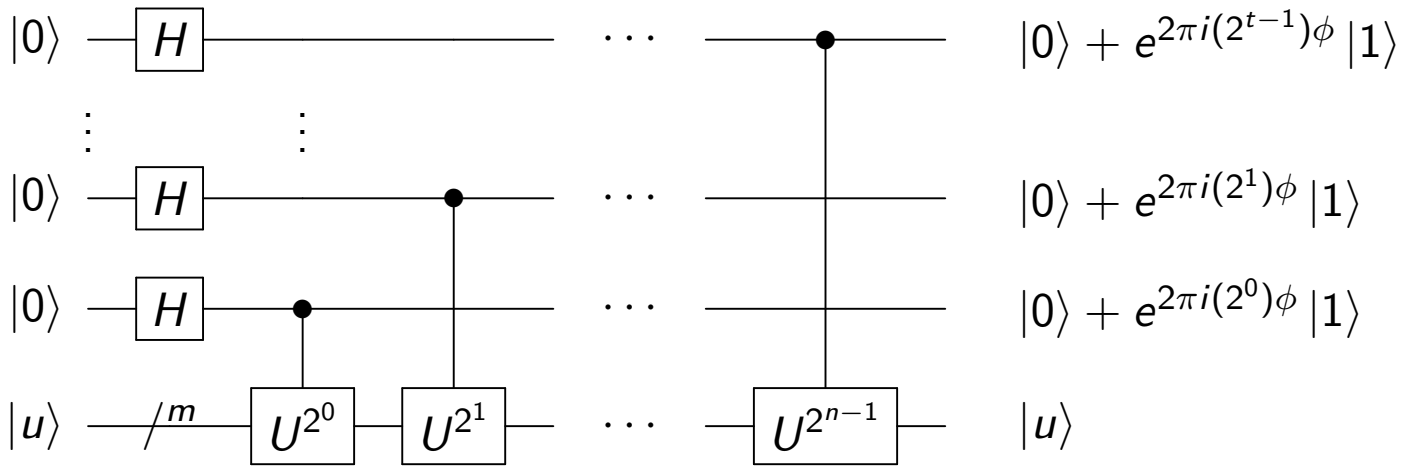
Phase Estimation

We focus on the dotted box first:



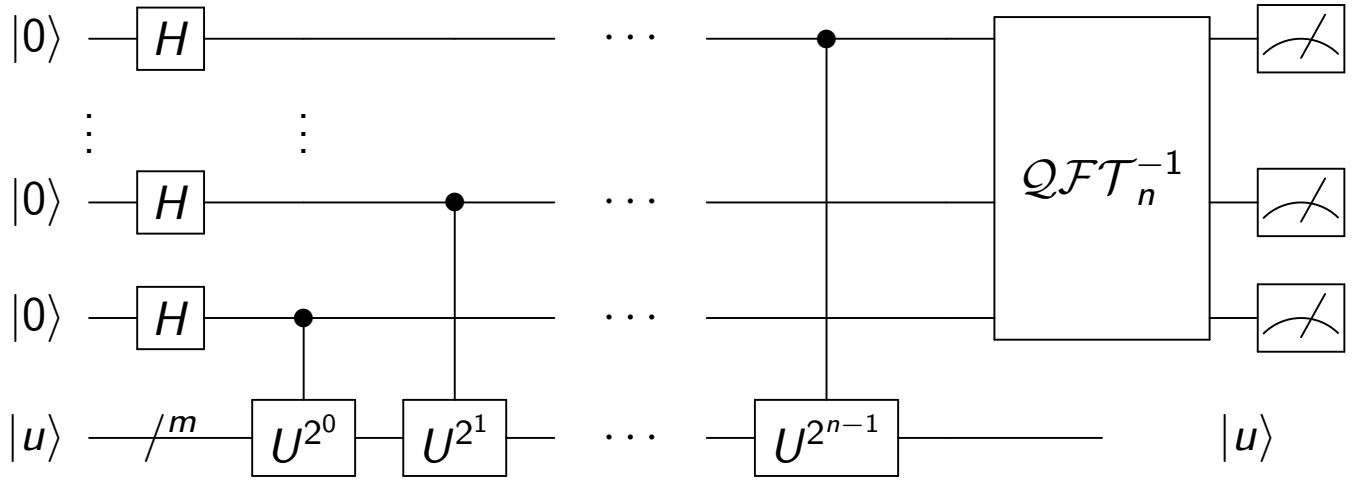
Phase Estimation

Here we have the circuit and output untill before applying the $\mathcal{QFT}_n^{-1}$. We see that the output is precisely that of the QFT if we had as input the computational qubits $|\phi_1 \cdots \phi_n\rangle$:

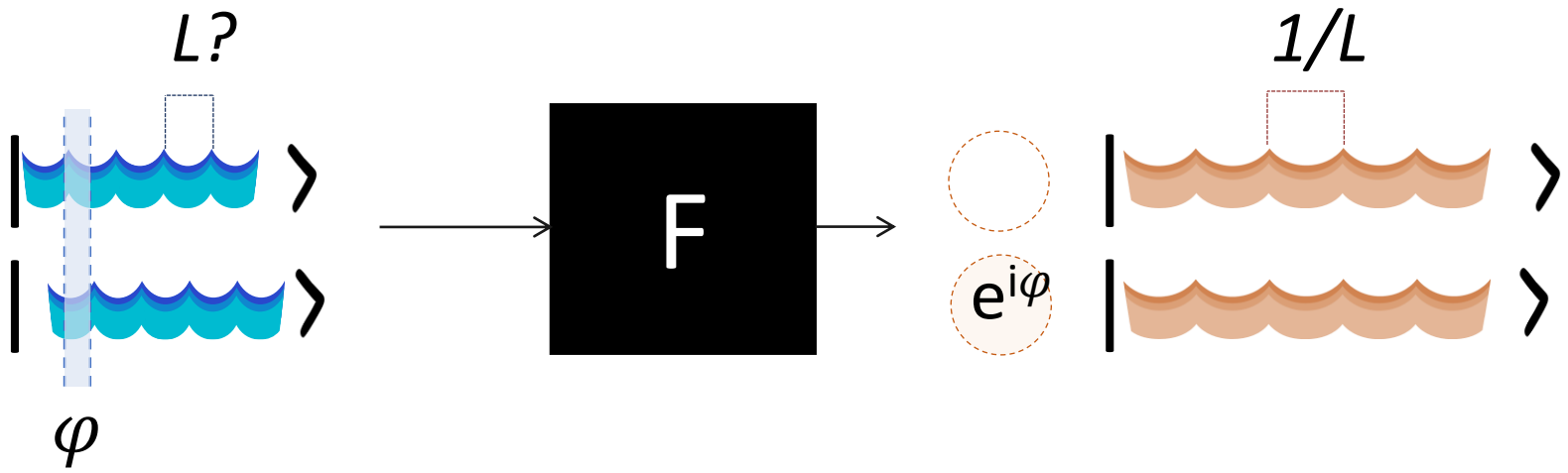


Phase Estimation

Therefore applying the inverse QFT and measuring in the computational basis will give us the value $2^n\phi = (\phi_1 \cdots \phi_n)$.



Quantum Factoring



Shor, FOCS 1994

Shor's Algorithm

- Invented by Peter Shor in 1994
- Is a **quantum algorithm** that solves **efficiently** the problem of factoring a large number to its prime factors

Shor's Algorithm

- Invented by Peter Shor in 1994
- Is a **quantum algorithm** that solves **efficiently** the problem of factoring a large number to its prime factors
- Factoring belongs in BQP
- There is no **known** polynomial classical algorithm for factoring
- An indication $\mathrm{BPP} \subset \mathrm{BQP}$. **Not a proof** since the classical conjectured hardness is **not** based on an impossibility proof

Shor's Algorithm

- Invented by Peter Shor in 1994
- Is a **quantum algorithm** that solves **efficiently** the problem of factoring a large number to its prime factors
- Factoring belongs in BQP
- There is no **known** polynomial classical algorithm for factoring
- An indication $\mathrm{BPP} \subset \mathrm{BQP}$. **Not a proof** since the classical conjectured hardness is **not** based on an impossibility proof
- Classical widely used cryptosystems (e.g. RSA) are based on the **assumption that factoring is hard**
- The **discrete logarithm problem** can also be **efficiently solved** using Shor's algorithm, breaking more used cryptosystems (e.g. ElGamal, Diffie-Hellman key exchange)

Shor's Algorithm

- Invented by Peter Shor in 1994
- Is a **quantum algorithm** that solves **efficiently** the problem of factoring a large number to its prime factors
- Factoring belongs in BQP
- There is no **known** polynomial classical algorithm for factoring
- An indication $\mathrm{BPP} \subset \mathrm{BQP}$. **Not a proof** since the classical conjectured hardness is **not** based on an impossibility proof
- Classical widely used cryptosystems (e.g. RSA) are based on the **assumption that factoring is hard**
- The **discrete logarithm problem** can also be **efficiently solved** using Shor's algorithm, breaking more used cryptosystems (e.g. ElGamal, Diffie-Hellman key exchange)

If a **scalable, fault tolerant, universal** quantum computer is built, most of current crypto breaks (from emails, bank transactions to national security secrets)!

Shor's Algorithm

It has a classical and a quantum part

- Classical Part: Reduces the factoring (and the discrete log) problem to the problem of Order-Finding

Shor's Algorithm

It has a classical and a quantum part

- Classical Part: Reduces the factoring (and the discrete log) problem to the problem of Order-Finding
- Quantum Part: Solves Order-Finding, it further requires:
  1. Quantum Fourier Transform
  2. Phase Estimation
  3. Efficient classical subroutines for: **reversible modular exponentiation** and the **continued fractions algorithm**

Shor's Algorithm

It has a classical and a quantum part

- Classical Part: Reduces the factoring (and the discrete log) problem to the problem of Order-Finding
- Quantum Part: Solves Order-Finding, it further requires:
  1. Quantum Fourier Transform
  2. Phase Estimation
  3. Efficient classical subroutines for: **reversible modular exponentiation** and the **continued fractions algorithm**
- We focus on Order-Finding, while we give at the end (for completeness) the classical part

Order Finding

Definition (Order): Given positive integers x and N that are **co-prime**, the **order** of x modulo N is the **least positive integer** r such that $x^r = 1 (\mathrm{mod}\, N)$

Order Finding

Definition (Order): Given positive integers x and N that are **co-prime**, the **order** of x modulo N is the **least positive integer** r such that $x^r = 1 (\mathrm{mod}\, N)$

The Problem

Determine the order r given the base number x and modulo N

Order Finding

Definition (Order): Given positive integers x and N that are **co-prime**, the **order** of x modulo N is the **least positive integer** r such that $x^r = 1 (\mathrm{mod}\, N)$

The Problem

Determine the order r given the base number x and modulo N

- It is believed to be difficult problem for classical computers (existing algorithms solve it in exponential time)
- The order-finding algorithm is (essentially) **phase estimation** applied to the following unitary operator $U_{x,N} |y\rangle = |xy (\mathrm{mod}\, N)\rangle$

Order Finding

Order-Finding can be decomposed into three steps:

1. Application of **Phase Estimation to the unitary**:

$$U_{x,N} |y\rangle = |xy (\mathrm{mod}\, N)\rangle$$

2. A method to **prepare the corresponding eigenstates** without knowing the order
3. A method to **recover the desired order** from the approximation of the phase/eigenvalue

Order Finding: Step 1 (Phase Estimation)

- The unitary $U_{x,N}|y\rangle = |xy(\mathrm{mod}\, N)\rangle$ corresponds to the function $f(x) = (xa) \bmod N$ which if iterated corresponds to **modular exponentiation**. Such a function (and unitary) can be efficiently in $O((\log N)^3)$ gates (see Nielsen & Chuang).
- The following state is an eigenstate of $U_{x,N}$:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp(\frac{-2\pi isk}{r}) |x^k \bmod N\rangle$$

- We can see this by evaluating $U_{x,N}|u_s\rangle$:

$$U_{x,N}|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \exp(\frac{-2\pi isk}{r}) |x^{k+1} \mathrm{mod} N\rangle = \exp(2\pi i \frac{s}{r}) |u_s\rangle$$

- The eigenvalue is $e^{2\pi is/r}$ and the phase $\phi = s/r$ has information about the order r which we need further steps to extract.

Order Finding: Step 2 (Preparing the Eigenstate)

- We **can't** prepare the eigenstates $|u_s\rangle$ since we don't know r.
- Note:

$$\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle = \frac{1}{r}\sum_{s=0}^{r-1}\sum_{k=0}^{r-1}\exp(\frac{-2\pi isk}{r})\,|x^k \bmod N\rangle = |1\rangle$$

- Running the $\mathcal{PE}$ algorithm on input $|0\rangle\,|u_s\rangle$:

$$\mathcal{PE}\left(|0\rangle\,|u_s\rangle\right) = |s/r\rangle\,|u_s\rangle$$

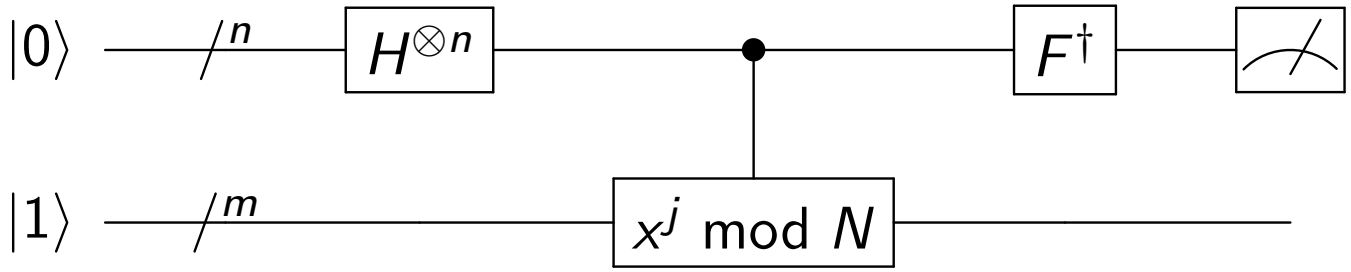
- By linearity, running the $\mathcal{PE}$ algorithm on input $|0\rangle\,|1\rangle$:

$$\mathcal{PE}\left(|0\rangle\,|1\rangle\right) = \mathcal{PE}\left(|0\rangle\,\frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|u_s\rangle\right) = \frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|s/r\rangle\,|u_s\rangle$$

- Measuring the first register collapses the state to one term: $|s'/r\rangle\,|u_{s'}\rangle$ for unknown both s', r.

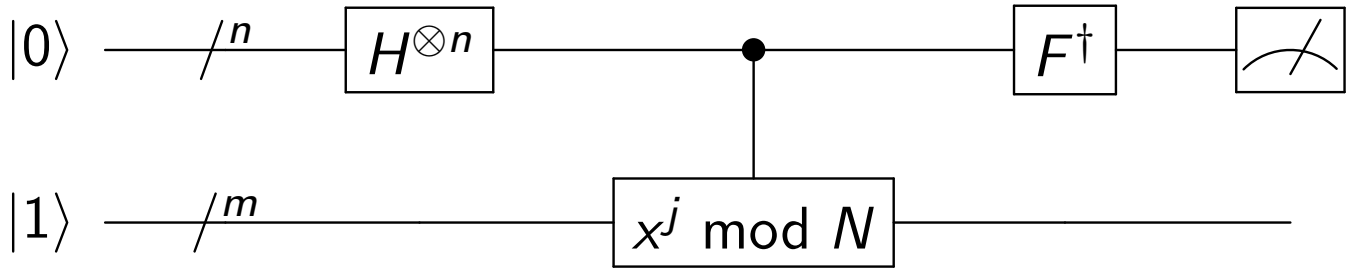
Order Finding: Summary

Circuit:



Order Finding: Summary

Circuit:

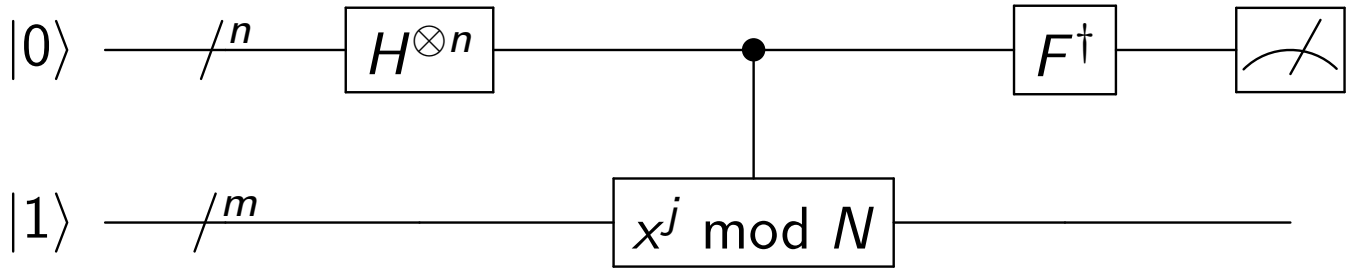


Steps:

1. Start with $|0\rangle\,|1\rangle$
2. Apply $\mathcal{PE}$, using the unitary $U_{x,N}$ in order to obtain: $\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |s/r\rangle\,|u_s\rangle$
3. Measure to obtain a n-bit approximation of s'/r for a randomly chosen s'
4. Use the continued fraction approximations to find the best guess for the order r
5. Check that is correct and terminate or repeat from step 1.

Order Finding: Summary

Circuit:



Steps:

1. Start with $|0\rangle\,|1\rangle$
2. Apply $\mathcal{PE}$, using the unitary $U_{x,N}$ in order to obtain: $\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} |s/r\rangle\,|u_s\rangle$
3. Measure to obtain a n-bit approximation of s'/r for a randomly chosen s'
4. Use the continued fraction approximations to find the best guess for the order r
5. Check that is correct and terminate or repeat from step 1.

Complexity: $O((\log N)^3)$ operations and succeeds w.h.p.

Grover's Search

Search Algorithms

- Searching (e.g. a name in a catalogue) is an important task with many applications
- Classical algorithms are not efficient, they require $O(N)$ queries in a database of size $N = 2^n$
- Quantum algorithms do better $O(\sqrt{N})$
- Quantum offers a **quadratic speed-up**. This is the optimal that quantum algorithms can achieve.
- It is practically important but much more moderate than an exponential speed-up (e.g. Shor's algorithm).

Grover's Algorithm: General

- We assume that if we are given a solution to the search, we **can recognise the solution** (typical in database search)
- Modelled with a **quantum oracle** that identifies the solution
- The "cost" is the **number of queries** to the oracle

Grover's Algorithm: General

- We assume that if we are given a solution to the search, we **can recognise the solution** (typical in database search)
- Modelled with a **quantum oracle** that identifies the solution
- The "cost" is the **number of queries** to the oracle

The method:

1. Start with an equal superposition of all database elements
2. Perform a subroutine (Grover's iteration) that **amplifies the amplitude** of the database element that constitutes the solution to the search
3. Repeat sufficient times so that the amplitude of the solution is close to unity
4. Measure in the computational basis and obtain the answer to the search

Grover's Algorithm: General

- We assume that if we are given a solution to the search, we **can recognise the solution** (typical in database search)
- Modelled with a **quantum oracle** that identifies the solution
- The "cost" is the **number of queries** to the oracle

The method:

1. Start with an equal superposition of all database elements
2. Perform a subroutine (Grover's iteration) that **amplifies the amplitude** of the database element that constitutes the solution to the search
3. Repeat sufficient times so that the amplitude of the solution is close to unity
4. Measure in the computational basis and obtain the answer to the search

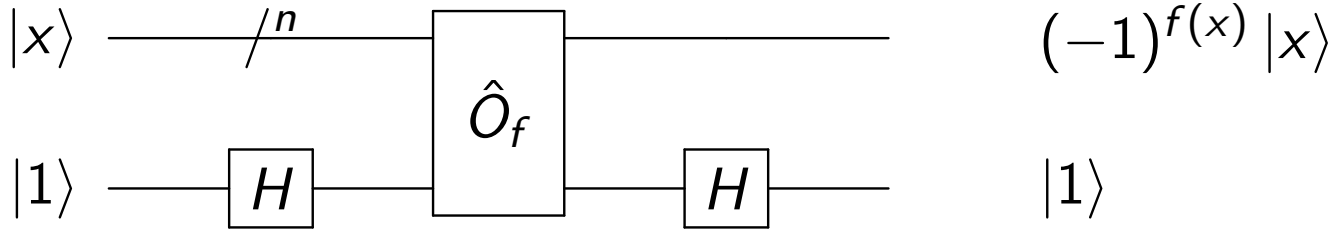
The algorithm can be generalised and is known as **amplitude amplification**

The Oracle

- We define the function $f : \{0,1\}^n \to \{0,1\}$ where $f(x) = 1$ iff x is the solution.
- The corresponding quantum oracle $\hat{O}_f$ acts (unitarily): $\hat{O}_f |x\rangle |q\rangle = |x\rangle |q \oplus f(x)\rangle$

The Oracle

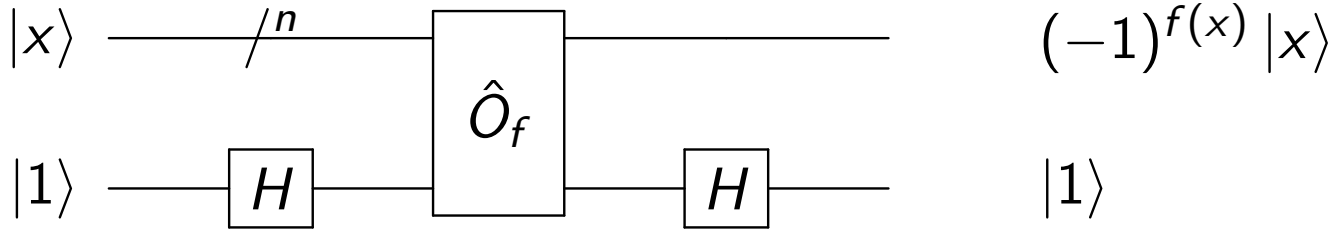
- We define the function $f : \{0,1\}^n \to \{0,1\}$ where $f(x) = 1$ iff x is the solution.
- The corresponding quantum oracle $\hat{O}_f$ acts (unitarily): $\hat{O}_f |x\rangle |q\rangle = |x\rangle |q \oplus f(x)\rangle$
- With the following circuit, the net result is that we multiply the input state with $(-1)^{f(x)}$, i.e. alter the overall sign only for the solution sought.



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

The Oracle

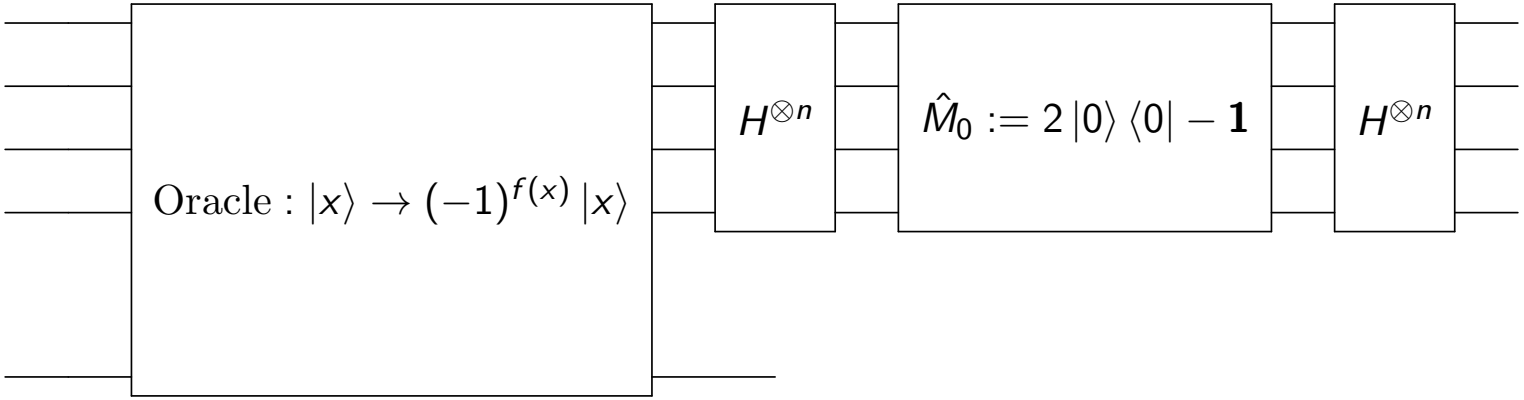
- We define the function $f : \{0,1\}^n \to \{0,1\}$ where $f(x) = 1$ iff x is the solution.
- The corresponding quantum oracle $\hat{O}_f$ acts (unitarily): $\hat{O}_f |x\rangle |q\rangle = |x\rangle |q \oplus f(x)\rangle$
- With the following circuit, the net result is that we multiply the input state with $(-1)^{f(x)}$, i.e. alter the overall sign only for the solution sought.



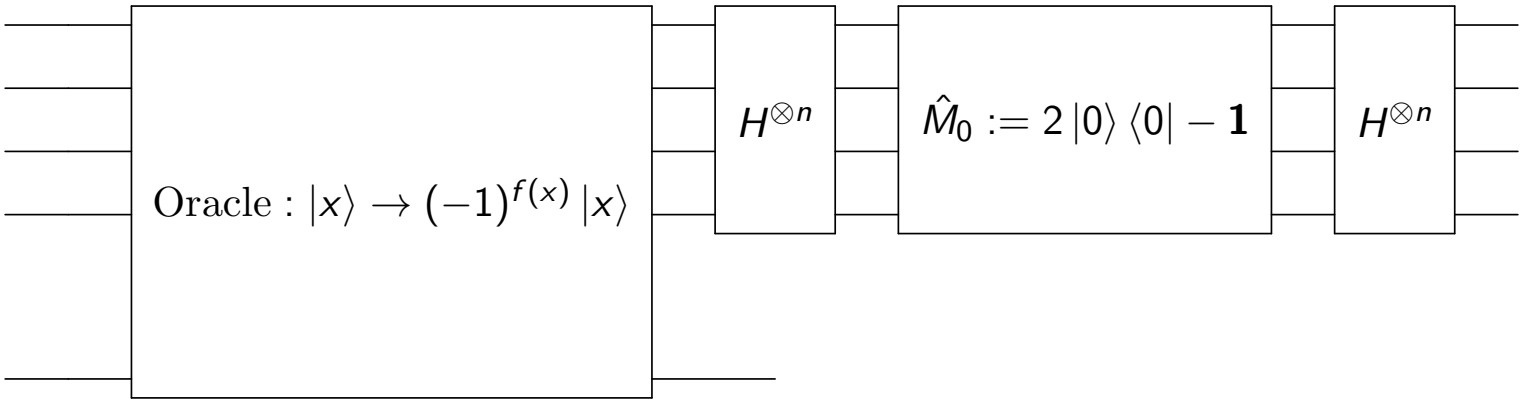
where we used
$\frac{1}{\sqrt{2}}(|f(x)\rangle - |1 \oplus f(x)\rangle) = (-1)^{f(x)} \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

- We can ignore the ancilla qubit and thus from here on, we replace the action of the oracle $\hat{O}_f$ with the circuit above.

Grover's Iteration

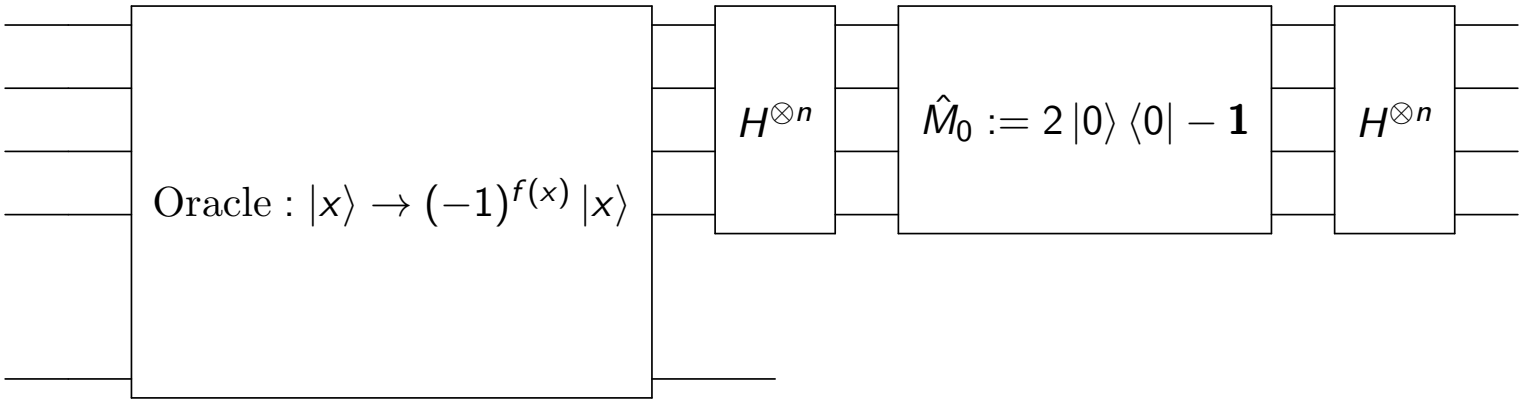


Grover's Iteration



1. Apply the oracle $\hat{O}_f$ (including the H at the ancilla qubit)
2. Apply H to the n-qubits
3. Reflect around the $|0\rangle$ state by applying $\hat{M}_0 := 2 |0\rangle \langle 0| - \mathbf{1}$
4. Apply H to the n-qubits

Grover's Iteration

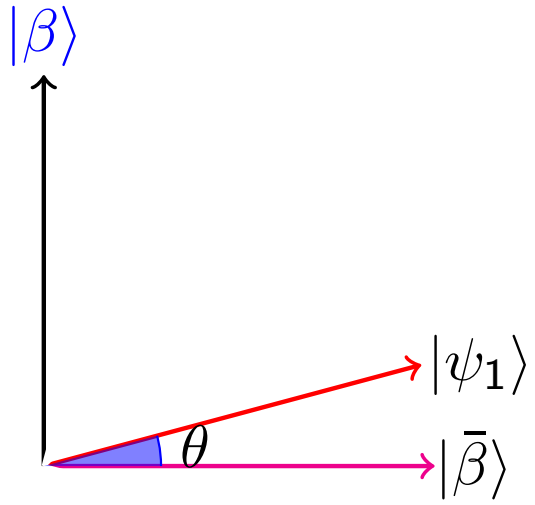


1. Apply the oracle $\hat{O}_f$ (including the H at the ancilla qubit)
2. Apply H to the n-qubits
3. Reflect around the $|0\rangle$ state by applying $\hat{M}_0 := 2 |0\rangle \langle 0| - \mathbf{1}$
4. Apply H to the n-qubits

Note that $H^{\otimes n} \hat{M}_0 H^{\otimes n} = \hat{M}_{+^n}$, where $|+^n\rangle := \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$:

$$\hat{G} = \hat{M}_{+n} \hat{O}_f = (2 |+^n\rangle \langle +^n| - \mathbf{1}) \hat{O}_f$$

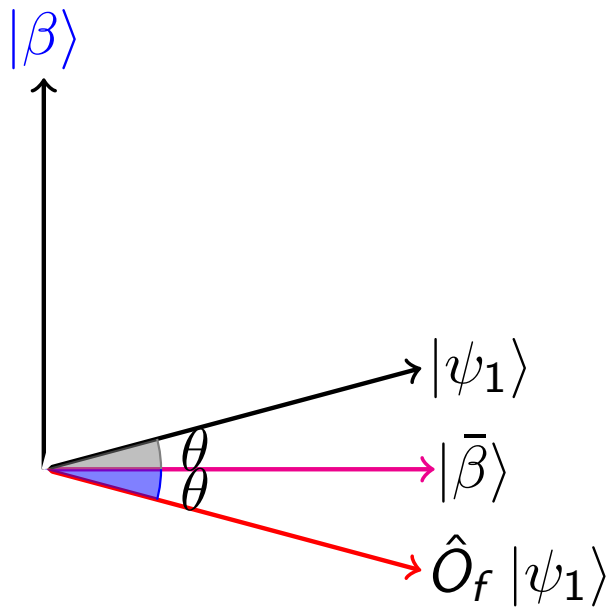
Grover's Iteration: Geometrically



Notation: State $|\beta\rangle$ is the solution, state $|\psi_1\rangle = |+^n\rangle$, state $|\bar{\beta}\rangle$ is in the same plane and $\langle\beta|\,\bar{\beta}\rangle = 0$, and $|\psi_{k+1}\rangle := \hat{M}_{+^n}\hat{O}_f\,|\psi_k\rangle$

Each iteration consists of two successive reflections:
(1) around $|\bar{\beta}\rangle$ and (2) around $|\psi_1\rangle$

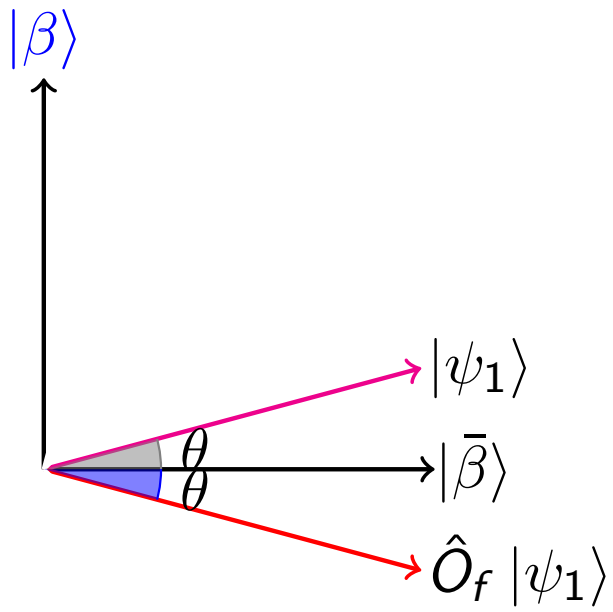
Grover's Iteration: Geometrically



Notation: State $|\beta\rangle$ is the solution, state $|\psi_1\rangle = |+^n\rangle$, state $|\bar{\beta}\rangle$ is in the same plane and $\langle\beta|\,\bar{\beta}\rangle = 0$, and $|\psi_{k+1}\rangle := \hat{M}_{+^n}\hat{O}_f\,|\psi_k\rangle$

Each iteration consists of two successive reflections:
(1) around $|\bar{\beta}\rangle$ and (2) around $|\psi_1\rangle$

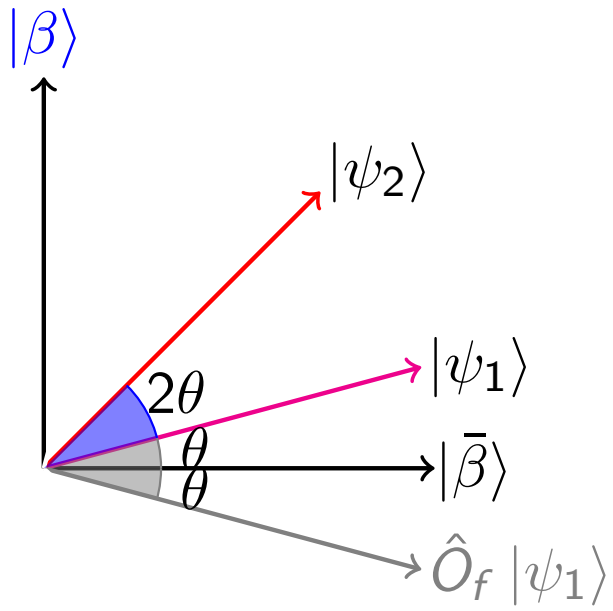
Grover's Iteration: Geometrically



Notation: State $|\beta\rangle$ is the solution, state $|\psi_1\rangle = |+^n\rangle$, state $|\bar{\beta}\rangle$ is in the same plane and $\langle\beta|\,\bar{\beta}\rangle = 0$, and $|\psi_{k+1}\rangle := \hat{M}_{+^n}\hat{O}_f\,|\psi_k\rangle$

Each iteration consists of two successive reflections:
(1) around $|\bar{\beta}\rangle$ and (2) around $|\psi_1\rangle$

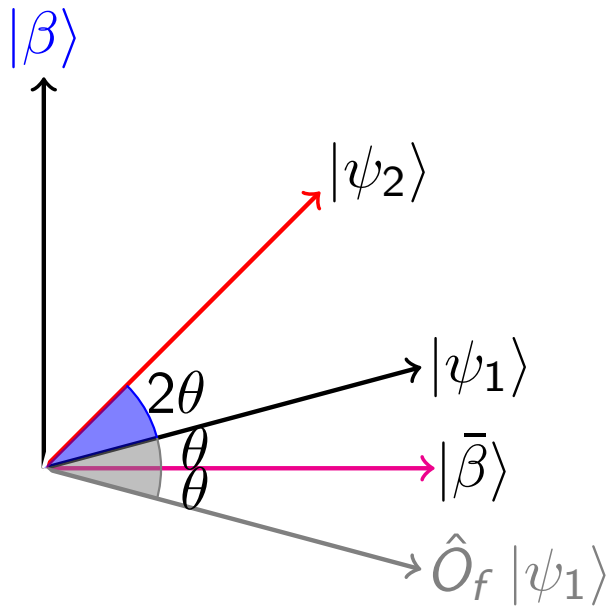
Grover's Iteration: Geometrically



Notation: State $|\beta\rangle$ is the solution, state $|\psi_1\rangle = |+^n\rangle$, state $|\bar{\beta}\rangle$ is in the same plane and $\langle\beta|\,\bar{\beta}\rangle = 0$, and $|\psi_{k+1}\rangle := \hat{M}_{+^n}\hat{O}_f\,|\psi_k\rangle$

Each iteration consists of two successive reflections:
(1) around $|\bar{\beta}\rangle$ and (2) around $|\psi_1\rangle$

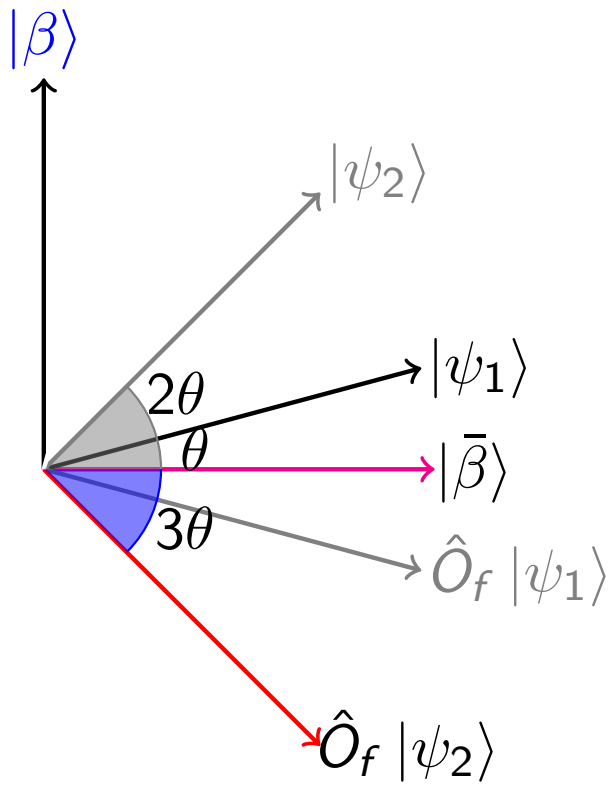
Grover's Iteration: Geometrically



Notation: State $|\beta\rangle$ is the solution, state $|\psi_1\rangle = |+^n\rangle$, state $|\bar{\beta}\rangle$ is in the same plane and $\langle\beta|\,\bar{\beta}\rangle = 0$, and $|\psi_{k+1}\rangle := \hat{M}_{+^n}\hat{O}_f\,|\psi_k\rangle$

Each iteration consists of two successive reflections:
(1) around $|\bar{\beta}\rangle$ and (2) around $|\psi_1\rangle$

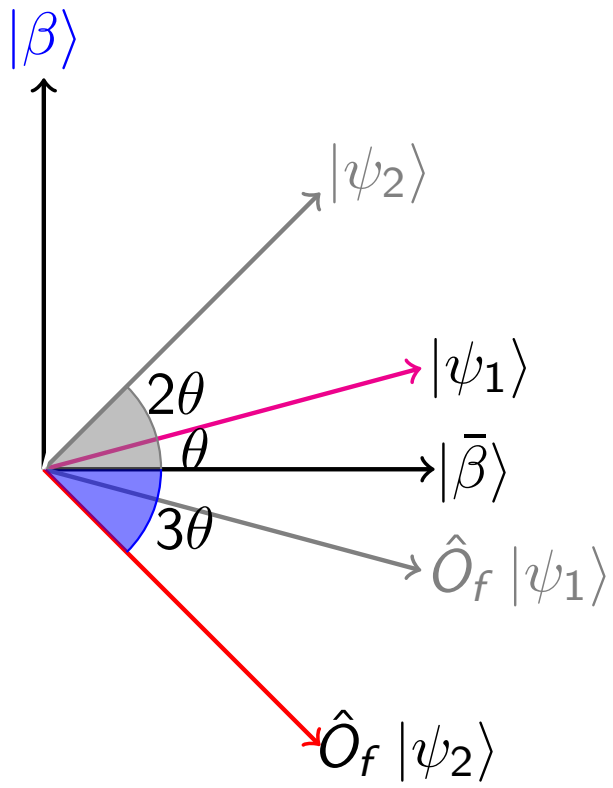
Grover's Iteration: Geometrically



Notation: State $|\beta\rangle$ is the solution, state $|\psi_1\rangle = |+^n\rangle$, state $|\bar{\beta}\rangle$ is in the same plane and $\langle\beta|\,\bar{\beta}\rangle = 0$, and $|\psi_{k+1}\rangle := \hat{M}_{+^n}\hat{O}_f\,|\psi_k\rangle$

Each iteration consists of two successive reflections:
(1) around $|\bar{\beta}\rangle$ and (2) around $|\psi_1\rangle$

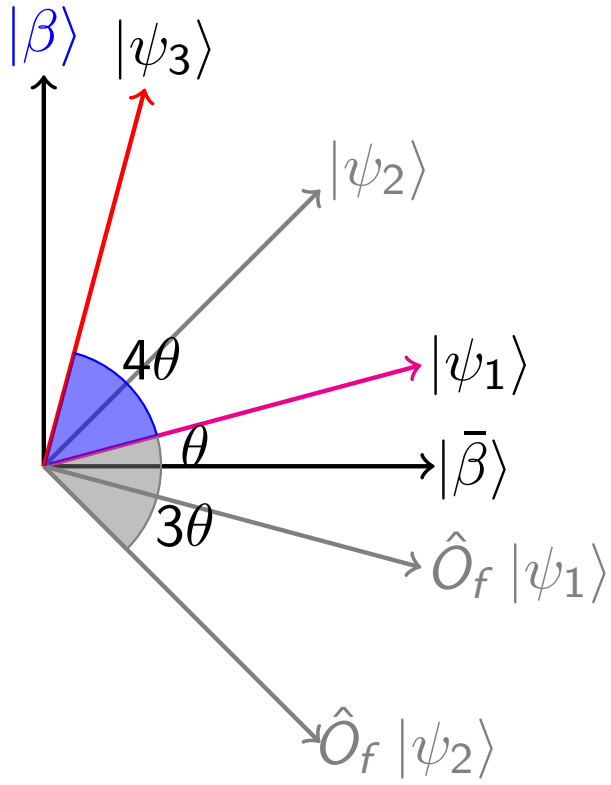
Grover's Iteration: Geometrically



Notation: State $|\beta\rangle$ is the solution, state $|\psi_1\rangle = |+^n\rangle$, state $|\bar{\beta}\rangle$ is in the same plane and $\langle\beta|\,\bar{\beta}\rangle = 0$, and $|\psi_{k+1}\rangle := \hat{M}_{+^n}\hat{O}_f\,|\psi_k\rangle$

Each iteration consists of two successive reflections:
(1) around $|\bar{\beta}\rangle$ and (2) around $|\psi_1\rangle$

Grover's Iteration: Geometrically



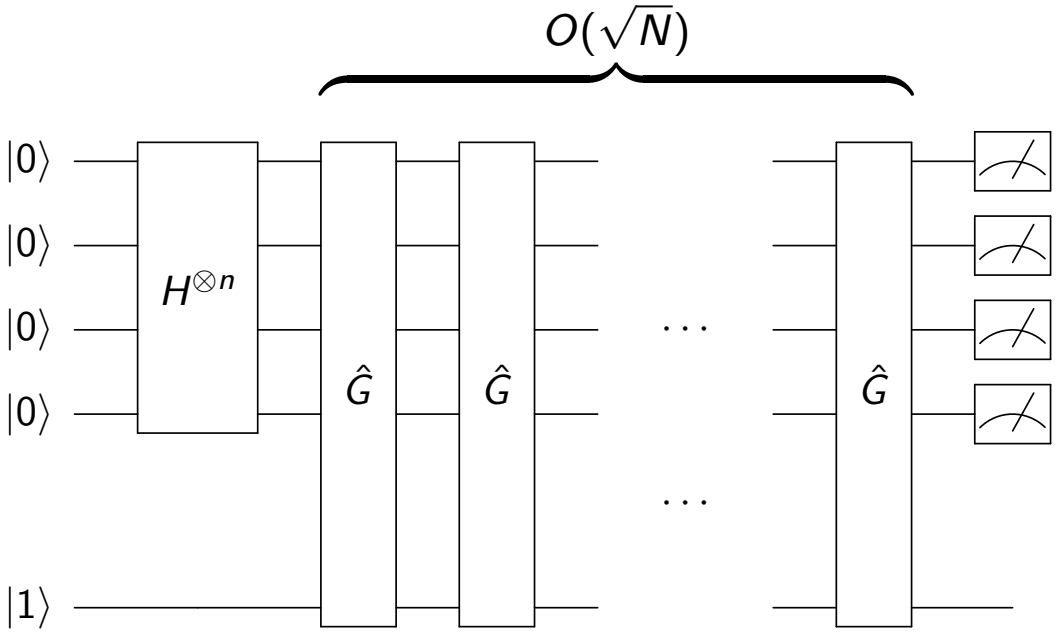
Notation: State $|\beta\rangle$ is the solution, state $|\psi_1\rangle = |+^n\rangle$, state $|\bar{\beta}\rangle$ is in the same plane and $\langle\beta|\,\bar{\beta}\rangle = 0$, and $|\psi_{k+1}\rangle := \hat{M}_{+^n}\hat{O}_f\,|\psi_k\rangle$

Each iteration consists of two successive reflections:
(1) around $|\bar{\beta}\rangle$ and (2) around $|\psi_1\rangle$

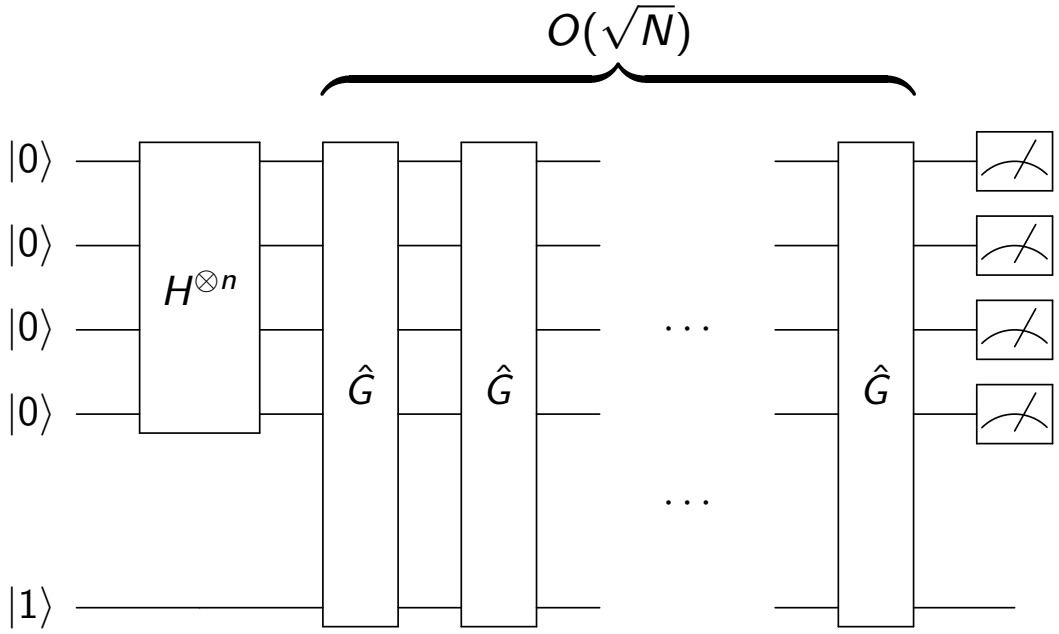
Gover's Iteration: Geometrically

- The angle between $|\bar{\beta}\rangle$ and $|\psi_k\rangle$ is: $(2k-1)\theta$
- To evaluate θ note that: $\sin\theta = \langle\beta|\,\psi_1\rangle = \frac{1}{\sqrt{2^n}}$ and for sufficient large n we have $\theta \approx \frac{1}{\sqrt{2^n}}$ which is very small!
- The starting state (equal superposition) is almost orthogonal to the solution
- We need $k = O(2^{n/2})$ iterations to reach a number $O(1)$, since the angle scales as $2k\theta$. Then we can obtain total angle $\approx \pi/2$ and can get with high probability the solution.

Grover's Algorithm



Grover's Algorithm



1. Apply H to the n-qubits to create equal superposition $|+^n\rangle$
2. Apply the Grover operator $\hat{G}$ to amplify the amplitude of the solution
3. Repeat $O(\sqrt{N})$ times
4. Measure the n-qubits in the computational basis

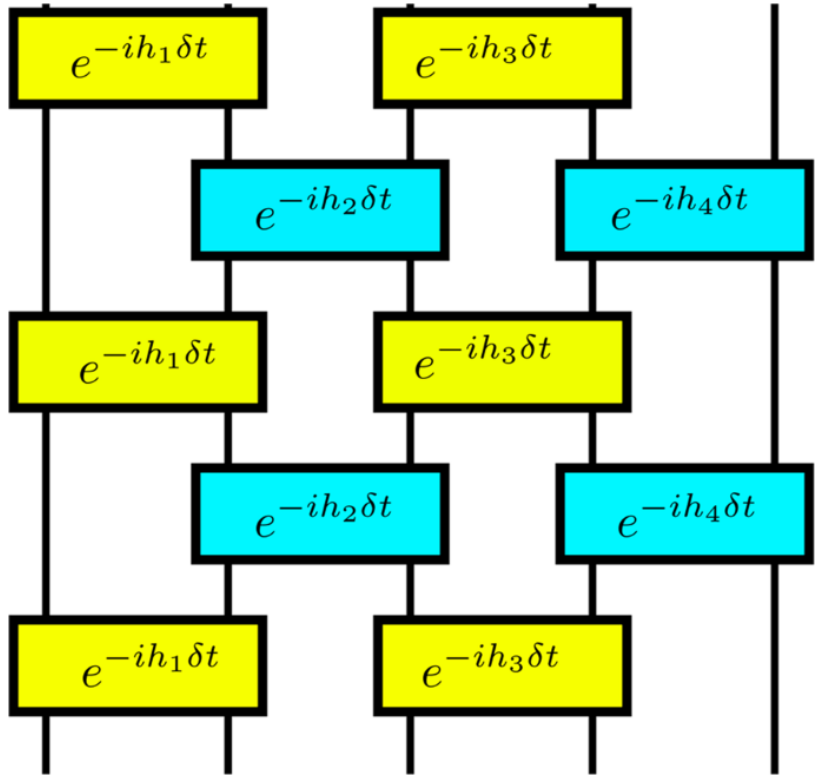
Grover's Algorithm

- We need one call to the oracle per Grover iteration, i.e. $O(\sqrt{N})$ queries
- We can generalise for M solutions to the search problem, with $O(\sqrt{N/M})$ queries
- For the algorithm to succeed need to know **how many iterations are required** (performing more iterations will eventually decrease the probability of success)
- If the number of solutions is unknown, there is an algorithm (**quantum counting**) that efficiently determines the number of solutions

More quantum algorithms

Hamiltonian Simulation

$$e^{-it\hat{H}} = e^{-it\sum_n \hat{H}_n} = \lim_{K\to\infty}\left(\prod_n e^{-i\frac{t}{K}\hat{H}_n}\right)^K$$



Quantum Algorithm for Linear equations

HHL algorithm for "solving" large linear systems

- Solving large linear systems $Ax = b$ is one of the most important problems in science and engineering.

  Goal: given matrix A and vector b, find vector x

- Harrow-Hassidim-Lloyd'09: "solves" this problem exponentially faster by preparing state $|x\rangle$ **IF** system is well-behaved:

  Assumptions

  (1) state $|b\rangle$ easy to prepare;

  (2) A is well-conditioned: $\lambda_{max}/\lambda_{min}$ not too big;

  (3) unitary operation e^{iA} is easy to apply (sparseness suffices)

How does the Harrow-Hassidim-Lloyd algorithm work?

- Input: Hermitian matrix $A \in \mathbb{R}^{N \times N}$ and vector $b \in \mathbb{R}^N$
  Goal: approximately prepare $|x\rangle$, where $Ax = b$
- Let $v_1, \ldots, v_N, \lambda_1, \ldots, \lambda_N$ be eigenvectors, eigenvalues of A
- HHL algorithm:
  1. Prepare quantum state $|b\rangle = \sum_{i=1}^{N} \beta_i |v_i\rangle$
     NB: applying A^{-1} corresponds to multiplying with λ_i^{-1}
  2. Use eigenvalue estimation: $\sum_{i=1}^{N} \beta_i |v_i\rangle|\lambda_i\rangle$
  3. Make new qubit $\sum_{i=1}^{N} \beta_i |v_i\rangle|\lambda_i\rangle \left(\lambda_i^{-1}|0\rangle + \sqrt{1 - \lambda_i^{-2}}|1\rangle \right)$
  4. Uncompute $|\lambda_i\rangle$ by inverting eigenvalue estimation
  5. Amplify the $|0\rangle$-part to end with $\sum_{i=1}^{N} \beta_i \lambda_i^{-1} |v_i\rangle = |x\rangle$

Read the fine print

Scott Aaronson

New quantum algorithms promise an exponential speed-up for machine learning, clustering and finding patterns in big data. But to achieve a real speed-up, we need to delve into the details.

For twenty years, quantum computing has been catnip to science journalists. Not only would a quantum computer harness the notorious weirdness of quantum mechanics, but it would do so for a practical purpose: solving certain problems exponentially faster than we know how to solve them with any existing computer. But, there's always been a catch — and I'm not even talking about the difficulty of building practical quantum computers. Supposing we had a quantum computer, what would we use it for? The 'killer apps' — the applications for which a quantum computer would promise huge speed advantages over classical computers — have struck some people as inconveniently narrow. Using a quantum computer, one could dramatically accelerate the simulation of quantum physics and chemistry (the original application advocated by Richard Feynman in the 1980s), break almost all of the public-key cryptography currently used on the Internet (for example, by quickly factoring large numbers with the famous Shor's algorithm[1]) and maybe achieve a modest speed-up for solving optimization problems in the infamous NP-hard class (but no one is sure about the last one). Alas, as interesting as that list might be, it's difficult to argue that it would transform civilization in anything like the way classical computing did in the previous century.

Recently, however, a new family of quantum algorithms has come along to challenge this relatively narrow view of why a quantum computer might be useful. Not only do these new algorithms promise an exponential speed-up over classical algorithms, but they do so for eminently practical problems involving machine learning, clustering, classification and finding patterns in huge amounts of data. So, do these algorithms live up to the claims? That's a simple question with a complicated answer.

One algorithm at the centre of the 'quantum machine learning' mini-revolution is called HHL after Aram Harrow, Avinatan Hassidim and Seth Lloyd, who invented it in 2008[2]. Many, though not all, of the subsequent quantum learning algorithms extend HHL or use it as a subroutine[3].

HHL attacks one of the most basic problems in all of science: solving a system of linear equations. Given an $n \times n$ real matrix, A, and a vector, $\mathbf{b}$, the goal of HHL is to (approximately) solve the system $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, and to do so in an amount of time that scales only logarithmically with n, the number of equations and unknowns. Classically, this goal seems hopeless, as n^2 steps would be required even to examine all of the entries of A, and n steps would be needed even to write down the solution vector, $\mathbf{x}$. In contrast, by exploiting the exponential character of the wave function, HHL promises to solve a system of n equations in only about $\log n$ steps. But does it really do so? This is one case where it's important to read the fine print.

> Do these algorithms live up to the claims? That's a simple question with a complicated answer.

Briefly, the HHL algorithm solves $A\mathbf{x} = \mathbf{b}$ in logarithmic time, but it does so with four caveats (Box 1), each of which can be crucial in practice. To make a long story short, HHL is not exactly an algorithm for solving a system of linear equations in logarithmic time. Rather, it's an algorithm for approximately preparing a quantum superposition of the form $|x\rangle$, where $\mathbf{x}$ is the solution to a linear system $A\mathbf{x} = \mathbf{b}$, assuming the ability to rapidly prepare the state $|x\rangle$, and to apply the unitary transformation e^{-iAt}, and using an amount of time that grows roughly like $\kappa s(\log n)/\varepsilon$, where n is the system size, κ is the system's condition number, s is its sparsity and ε is the desired error.

For all that, couldn't the HHL algorithm still be useful for something? Absolutely — as long as one can address all of the caveats, and explain why they're not fatal for the desired application. To put it differently, we could see HHL less as a quantum algorithm in its own right than as a template for other quantum algorithms. One fills out the template by showing how to prepare $|b\rangle$, apply e^{-iAt}, and measure $|x\rangle$ in a specific case of interest, and then carefully analyses the resulting performance against that of the best-known classical algorithm for that case.

To my knowledge, so far there have been two attempts to work out potential applications of the HHL template from start to finish. Clader, Jacobs, and Sprouse[5] argued that HHL could be used to speed up the calculation of electromagnetic scattering cross-sections, for systems involving smooth geometric figures in three-dimensional space. To do this, Clader *et al.*[5] needed to verify a number of things, in addition to the fact that solving Maxwell's equations can be reduced to solving a linear system $A\mathbf{x} = \mathbf{b}$. First, the appropriate matrix A is sparse. Second, at least from numerical data, the condition number κ is bounded, after careful 'preconditioning' is performed. Third, provided the object is regular, one can calculate the entries of A and prepare the state $|b\rangle$ explicitly, avoiding the need for a quantum RAM. And finally, given that one really is interested in specific observables of the solution vector $\mathbf{x}$, one doesn't need the entire $(x_1, ..., x_n)$.

Crucially, Clader *et al.*[5] could not rule out the possibility that, once the problem of solving a linear system has been restricted in all these ways, there is also a classical algorithm that provides the answer in nearly the same amount of time as HHL. The most they could say is that they couldn't find such a classical algorithm. The difficulty here is a general one: in quantum algorithms research we always want to compare to the fastest possible classical algorithm that performs the same task. But if we are honest, the 'task' here cannot be defined as solving an arbitrary linear system $A\mathbf{x} = \mathbf{b}$, because HHL can't do that either. The task, rather, needs to be defined as estimating certain observables of $\mathbf{x}$ for certain special systems $A\mathbf{x} = \mathbf{b}$, such that HHL can efficiently estimate the same observables. And that makes it far less obvious whether there might be a clever classical solution as well.

Similarly, in ref. 6, HHL was used to give a quantum algorithm for approximating effective resistances in electrical networks. Again, this algorithm could give an exponential speed-up over classical

algorithms, but only under special conditions: for example, if the electrical network has small degree but a large amount of interconnectivity and if a description of the electrical network can be quickly loaded into the quantum computer (for example, because the network has a regular pattern). It is not yet clear whether there are real-world examples in which these conditions are satisfied, but we also lack a fast classical algorithm to approximate the effective resistances.

One important result indicates that there are at least some cases where HHL runs in logarithmic time, whereas any possible classical algorithm requires polynomial time[2]. Namely, HHL was shown to be universal for quantum computation[2]. What this means is that we can encode any quantum algorithm — say, Shor's factoring algorithm — into a system of roughly 2^n linear equations with 2^n unknowns, and then use HHL to 'solve' the system (simulate the algorithm) in polynomial time. Thus, provided we believe any quantum algorithm achieves an exponential speed-up over the best possible classical algorithm, HHL can in principle achieve such a speed-up as well. On the other hand, the linear systems produced by this reduction will be extremely artificial. The essential insight behind the exponential speed-up, one might argue, is provided by Shor's algorithm or whatever other quantum algorithm we are simulating, rather than by HHL itself.

In the years since HHL, quantum algorithms achieving exponential speed-up over classical algorithms have been proposed for other major application areas, including *k*-means clustering[7], principal component analysis[8], support vector machines[9], data fitting[10] and even computing Google PageRanks[11]. Although not all of these algorithms face exactly the same caveats as HHL, it is fair to say that there are related issues in getting quantum speed-ups from them. With each of them, one faces the problem of how to load a large amount of classical data into a quantum computer (or else compute the data on the fly), in a way that is efficient enough to preserve the quantum speed-up. With each, one faces the problem that, even if the output state $|\psi\rangle$ implicitly encodes all the data one wants in its amplitudes, a measurement of $|\psi\rangle$ reveals only a tiny probabilistic sketch of the information. Most importantly, with each algorithm one faces uncertainty about whether, after one has properly accounted for all the caveats, one could then find a classical sampling algorithm that achieves similar performance to that of the quantum algorithm.

To illustrate these issues, let's consider the algorithm in ref. 7 for supervised machine learning. In this task, we are given a quantum state $|u\rangle$ of $\log_2 n$ qubits — or equivalently, a vector of n amplitudes — as well as two 'clusters' of other states $(|v_1\rangle, ..., |v_m\rangle)$ and $(|w_1\rangle, ..., |w_m\rangle)$. The problem is to classify $|u\rangle$ into one of the two clusters, by deciding which mean it is closer to: $|v\rangle = (|v_1\rangle + ... + |v_m\rangle)/m$ or $|w\rangle = (|w_1\rangle + ... + |w_m\rangle)/m$. Lloyd *et al.*[7] have provided an extremely fast quantum algorithm for this problem: in particular, they showed how to estimate the inner products $\langle u|v\rangle$ and $\langle u|w\rangle$ with an accuracy of ε, using only about $(\log mn)/\varepsilon$ steps. They argue that this represents an exponential speed-up over classical inner-product computation.

The trouble is, where did we get these states $|u\rangle$, $|v_k\rangle$ and $|w_k\rangle$ in the first place? The most obvious possibility would be that we prepared the states ourselves, using lists of amplitudes written out in a quantum RAM. In that case, however, a straightforward preparation strategy would work only if the amplitudes are relatively uniform — if there aren't a few amplitudes that dominate all the others in magnitude. But if the amplitudes are uniform in the required way, then it's not hard to show that we can also estimate the inner products $\langle v|u\rangle$ and $\langle u|w\rangle$ using a classical random sampling algorithm, in about $(\log mn)/\varepsilon^2$ steps. In other words, once we carefully spell out conditions under which the quantum algorithm would be useful, we find — at least in this example — that a classical algorithm exists that is at most quadratically slower.

Admittedly, the above argument doesn't rule out that there could be other ways to generate the states $|u\rangle$, $|v_k\rangle$ and $|w_k\rangle$, and that with those other ways the quantum speed-up would stand. And indeed, it can be shown that if our task was to estimate $\langle u|F|v\rangle$, where F is an $n \times n$ unitary matrix that applies a Fourier transform, then any classical algorithm would need at least about

Box 1 | HHL checklist of caveats.

(1) The vector $\mathbf{b} = (b_1, ..., b_n)$ somehow needs to be loaded quickly into the quantum computer's memory, so that we can prepare a quantum state $|b\rangle = \sum_{i=1}^{n} b_i |i\rangle$, of $\log_2 n$ quantum bits, whose n amplitudes encode the entries of $\mathbf{b}$. Here, I assume for simplicity that $\mathbf{b}$ is a unit vector. At least in theory, this can be accomplished using a 'quantum RAM' — a memory that stores the classical values b_i, and that allows them all to be read at once, in a quantum superposition. Even then, however, it's essential either that $\mathbf{b}$ is relatively uniform, without a few values of b_i that are vastly larger than the others, or else that the quantum RAM contains (say) the partial sums $\sum_{i=1}^{j} b_i^2$, or pointers to the large entries of $\mathbf{b}$, rather than just values of b_i^2. Alternatively, if $\mathbf{b}$ is described by a simple, explicit formula, then the quantum computer might be able to prepare $|b\rangle$ quickly by itself, without needing to consult a quantum RAM. Either way, though, if preparing $|b\rangle$ already takes n^c steps for some constant c, then the exponential speed-up of HHL vanishes in the very first step.

(2) The quantum computer also needs to be able to apply unitary transformations of the form e^{-iAt}, for various values of t. If the matrix A is sparse — it contains at most s nonzero entries per row, for some $s \ll n$ — and if there is a quantum RAM that conveniently stores, for each i, the locations and values of the nonzero entries in row i — then it is known that one can apply e^{-iA} in an amount of time that grows nearly linearly with s (ref. 4). There are other special classes of matrix A for which a quantum computer could efficiently apply e^{-iAt}. Again, though, if applying e^{-iAt} takes n^c time for some constant c, then the exponential speed-up of HHL disappears.

(3) The matrix A needs to be not merely invertible, but robustly invertible, or 'well-conditioned'. More precisely, let $\kappa = |\lambda_{max} / \lambda_{min}|$ be the ratio in magnitude between the largest and smallest eigenvalues of A. Then the amount of time needed by HHL grows linearly with κ. If κ grows like n^c, then the exponential speed-up is gone.

(4) The limitation noted earlier — that even writing down $\mathbf{x} = (x_1, ..., x_n)$ already requires n steps — also applies in the quantum world. When HHL is finished, its output is not $\mathbf{x}$ itself, but rather a quantum state $|x\rangle$ of $\log_2 n$ qubits, which (approximately) encodes the entries of $\mathbf{x}$ in its amplitudes. The quantum computer user can then measure $|x\rangle$ in a basis of her choice, to reveal some limited statistical information about $\mathbf{x}$: for example, the locations of any extremely large entries of $\mathbf{x}$, or the approximate value of an inner product $\langle x|z\rangle$, where $\mathbf{z}$ is a fixed vector. However, learning the value of any specific entry x_i will, in general, require repeating the algorithm roughly n times, which would once again kill the exponential speed-up.

$(\sqrt{n})/(\log n)$ steps, nearly matching an upper bound of $\sqrt{n}$ steps[12,13]. On the other hand, a quantum computer can estimate $\langle u|F|v\rangle$ just as easily as it can estimate $\langle u|v\rangle$, in only a logarithmic number of steps. To put it differently, if one of two vectors that we want to compare is given to us, not directly, but only through its Fourier transform, then we really do get an exponential quantum advantage in comparing the vectors.

Even here, however, care is needed. The theorem in ref. 12 — which rules out a superfast classical algorithm to compare one vector **u** written in memory against the Fourier transform of a second vector **v** written in memory — applies only if **u** and **v** are both completely random when considered individually. Furthermore, we still have the issue mentioned before: that a quantum computer can estimate $\langle u|F|v\rangle$ in logarithmic time only if it can quickly prepare the states $|u\rangle$ and $|v\rangle$. And to do that, starting from lists of entries $(|u_1\rangle, ..., |u_n\rangle)$ and $(|v_1\rangle, ..., |v_n\rangle)$ in a quantum RAM, we need the vectors to be reasonably uniform: neither **u** nor **v** can have sharp 'peaks'. Take away either of these conditions, and the exponential quantum speed-up can once again disappear.

The general rule here is that if we want an exponential quantum speed-up, then we need there to be an extremely fast way to create a certain kind of superposition, and we also need there not to be a fast way to create an analogous probability distribution using a classical computer. The Fourier transform is one convenient way to achieve both of these goals: it can foil classical sampling methods because of its 'global' nature, while causing no problems for a quantum computer. But quantum algorithm designers have other tricks up their sleeves.

Recently, a polynomial-time quantum algorithm has been proposed[14] for estimating certain topological features of data: most notably Betti numbers, which count the number of holes and voids of various dimensions in a scatterplot. What is notable about the algorithm used in ref. 14 is that its input consists of only $n(n-1)/2$ real numbers — namely, the distances between each pair of points out of n — and using that data the algorithm constructs a quantum superposition over up to 2^n simplices in a simplicial complex. In other words, much like the algorithm of Clader *et al.*[5] for electromagnetic cross-sections, the new algorithm avoids the expensive need to load a huge amount of data from a quantum RAM. Instead, it computes the data as needed from a much smaller input. The Betti number algorithm still has a significant drawback: preparing a superposition over simplices takes a number of steps that grows like $\sqrt{(2^n/S)}$, where S is the total number of simplices in the complex. So, if S is much smaller than 2^n, as it will often be in practice, the algorithm can require exponential time. Even so, the ability to estimate Betti numbers efficiently when $S \approx 2^n$ provides a good example of how the limitations of HHL-type algorithms can sometimes be overcome.

So in summary, how excited should we be about the new quantum machine-learning algorithms? To whatever extent we care about quantum computing at all, I'd say we should be excited indeed: HHL and the later algorithms represent real advances in the theory of quantum algorithms, and in a world with quantum computers, they would probably find practical uses. But along with the excitement, we ought to maintain a sober understanding of what these algorithms would and wouldn't do: an understanding that the original papers typically convey, but that often gets lost in second-hand accounts.

The new algorithms provide a general template, showing how quantum computers might be used to exponentially speed up central problems like clustering, pattern-matching, and principal component analysis. But for each intended application of the template, one still needs to invest a lot of work to see whether the application satisfies all of the algorithm's 'fine print' — and if, once we include the fine print, there's also a fast classical algorithm that provides the same information. This property makes the quantum machine-learning algorithms quite different from, say, Shor's factoring algorithm. Having spent half my life in quantum-computing research, I still find it miraculous that the laws of quantum physics let us solve any classical problems exponentially faster than today's computers seem able to solve them. So maybe it shouldn't surprise us that in machine learning, like anywhere else, nature will still make us work for those speed-ups. ❐

Scott Aaronson is in the Electrical Engineering and Computer Science Department, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139-4307, USA.
e-mail: aaronson@csail.mit.edu

References

1. Shor, P. W. *SIAM J. Comput.* **26,** 1484–1509 (1997).
2. Harrow, A. W., Hassidim, A. & Lloyd, S. *Phys. Rev. Lett.* **103,** 150502 (2009).
3. Childs, A. M. *Nature Phys.* **5,** 861 (2009).
4. Berry, D. W., Childs, A. M., Cleve, R., Kothari, R. & Somma, R. D. *Phys. Rev. Lett.* **114,** 090502 (2015).
5. Clader, B. D., Jacobs, B. C. & Sprouse, C. R. *Phys.Rev. Lett.* **110,** 250504 (2013).
6. Wang, G. Preprint at http://arxiv.org/abs/1311.1851 (2013).
7. Lloyd, S., Mohseni, M. & Rebentrost, P. Preprint at http://arxiv.org/abs/1307.0411 (2013).
8. Lloyd, S., Mohseni, M. & Rebentrost, P. *Nature Phys.* **10,** 631–633 (2014).
9. Rebentrost, P., Mohseni, M. & Lloyd, S. *Phys. Rev. Lett.* **113,** 130503 (2014).
10. Wiebe, N., Braun, D. & Lloyd, S. *Phys. Rev. Lett.* **109,** 050505 (2012).
11. Garnerone, S., Zanardi, P. & Lidar, D. A. *Phys. Rev. Lett.* **108,** 230506 (2012).
12. Aaronson, S. Preprint at http://arxiv.org/abs/0910.4698 (2009).
13. Aaronson, S. & Ambainis, A. Preprint at http://arxiv.org/abs/1411.5729 (2014).
14. Lloyd, S., Garnerone, S. & Zanardi, P. Preprint at http://arxiv.org/abs/1408.3106 (2014).

Quantum optics route to market

Jürgen Stuhler

Research in quantum optics has already led to commercial technologies, but the gap between the lab and market products is still large. Looking from the industrial side, one can see ways of bridging this gap.

Quantum optics research has flourished over the past twenty years. Driven by scientific curiosity and enthusiasm, potential applications and — last but not least — available funding, many groups around the world have published fundamental insights and technologically promising results. During these two decades, the Nobel Prize in Physics was awarded to more than ten scientists for their work on quantum optics phenomena or tools. This resonates with the statement of the European Commission that the twenty-first century will be the century of the photon — much like the twentieth century was the century of the electron. And 2015 has been declared the International Year of Light and Light-based Technologies. Undoubtedly, quantum optics research is in vogue and expectations are high. But, as in any other scientific field, simple economics