An Introduction to Quantum Algorithms

Marianna Bonanome
City Tech

May 11, 2021

Goals:

1. The "intellectual core" of computer science - review of computational complexity.
2. Moving from the classical to the quantum
3. What is a quantum bit (qubit)?
4. Advantages of quantum computation vs. classical computation
5. Reversible logic Gates, quantum logic gates
6. Breaking RSA! - Shor's algorithm for the quantum factorization of integers
7. What else can we do? Grover's search procedure.
8. Sycamore vs. Summit: the fight for supremacy
9. Conclusions
10. References
11. Additional materials/notes

Computational Complexity - Hard, Harder, Hardest

Efficient Algorithms

Some polynomial time algorithms:

1. The linear search procedure: $\Theta(n)$.
2. The bubble sort procedure: $\Theta(n^2)$.
3. Matrix multiplication: $\Theta(n^3)$.

Tractability

A problem that is solvable using an algorithm with polynomial (or better) worst-complexity is called **tractable**, because the expectation is that the algorithm will produce the solution to the problem for reasonably sized input in a relatively short time.

Computational Complexity - Hard, Harder, Hardest

Efficient Algorithms

Some polynomial time algorithms:

1. The linear search procedure: $\Theta(n)$.
2. The bubble sort procedure: $\Theta(n^2)$.
3. Matrix multiplication: $\Theta(n^3)$.

Tractability

A problem that is solvable using an algorithm with polynomial (or better) worst-complexity is called **tractable**, because the expectation is that the algorithm will produce the solution to the problem for reasonably sized input in a relatively short time.

Problems that cannot be solved using an algorithm with worst-case polynomial time complexity are called **intractable**.

Some important classes of problems

P and NP

The class problems which are tractable is denoted by P.

Some important classes of problems

P and NP

The class problems which are tractable is denoted by P.

There are many problems for for which a solution, once found, can be recognized as correct in polynomial time - even though the solution itself might be hard to find (no poly-time algorithm to find the solution). The class of these problems is referred to as NP.

Some important classes of problems

P and NP

The class problems which are tractable is denoted by P.

There are many problems for for which a solution, once found, can be recognized as correct in polynomial time - even though the solution itself might be hard to find (no poly-time algorithm to find the solution). The class of these problems is referred to as NP.

A familiar NP problem

The factoring problem is in NP but outside of P because no known algorithm for a classical computer can solve it in only a polynomial number of steps - instead the number of steps increases exponentially as n increases. We will come back to this problem later!

Some important classes of problems

P and NP

The class problems which are tractable is denoted by P.

There are many problems for for which a solution, once found, can be recognized as correct in polynomial time - even though the solution itself might be hard to find (no poly-time algorithm to find the solution). The class of these problems is referred to as NP.

A familiar NP problem

The factoring problem is in NP but outside of P because no known algorithm for a classical computer can solve it in only a polynomial number of steps - instead the number of steps increases exponentially as n increases. We will come back to this problem later!

NP-complete

There is a class of problems with the property that if any of these problems can be solved by a an efficient algorithm, then all problems in the class can be solved by an efficient algorithm. They are in essence the "same" problem!!!

$P = NP$?

The million dollar question (literally)

An efficient algorithm for an NP-**complete** problem would mean that computer scientists' present picture of the classes P, NP and NP-**complete** was utterly wrong!!! It would mean that $P = NP$!

$P = NP$?

The million dollar question (literally)

An efficient algorithm for an *NP*-**complete** problem would mean that computer scientists' present picture of the classes *P*, *NP* and *NP*-**complete** was utterly wrong!!! It would mean that $P = NP$!

Does such an algorithm exist? This question carries a $1,000,000 reward from the Clay Math Institute in Cambridge, Mass.

$P = NP$?

The million dollar question (literally)

An efficient algorithm for an NP-**complete** problem would mean that computer scientists' present picture of the classes P, NP and NP-**complete** was utterly wrong!!! It would mean that $P = NP$!

Does such an algorithm exist? This question carries a \$1,000,000 reward from the Clay Math Institute in Cambridge, Mass.

If we grant that $P \neq NP$, our only hope is to broaden what we mean by "computer."

$P = NP$?

The million dollar question (**literally**)

An efficient algorithm for an NP-**complete** problem would mean that computer scientists' present picture of the classes P, NP and NP-**complete** was utterly wrong!!! It would mean that $P = NP$!

Does such an algorithm exist? This question carries a $1,000,000 reward from the Clay Math Institute in Cambridge, Mass.

If we grant that $P \neq NP$, our only hope is to broaden what we mean by "computer."
NOTE: The factoring problem is neither known nor believed to be NP-**complete**.

Shifting focus: Classical to Quantum

Limits to Digital Computation

In 1965 Gordon Moore gave a law for the growth of computing power which states

Shifting focus: Classical to Quantum

Limits to Digital Computation

In 1965 Gordon Moore gave a law for the growth of computing power which states

Moore's Law

Computer power will double for constant cost roughly every two years.

Shifting focus: Classical to Quantum

Limits to Digital Computation

In 1965 Gordon Moore gave a law for the growth of computing power which states

Moore's Law

Computer power will double for constant cost roughly every two years.

Prediction

This dream will come to an end during this decade.

Shifting focus: Classical to Quantum

Limits to Digital Computation

In 1965 Gordon Moore gave a law for the growth of computing power which states

Moore's Law

Computer power will double for constant cost roughly every two years.

Prediction

This dream will come to an end during this decade.

Quantum effects are beginning to interfere with electronic devices as they are made smaller and smaller.

Solution

Move to a different paradigm!

Solution

Move to a different paradigm!

Can we use the counterintuitive laws of quantum mechanics to our advantage?

Solution

Move to a different paradigm!

Can we use the counterintuitive laws of quantum mechanics to our advantage?

Yes! In 1982 Richard Feynman and Paul Benioff independently observed that a quantum system can perform a computation.

Solution

Move to a different paradigm!

Can we use the counterintuitive laws of quantum mechanics to our advantage?

Yes! In 1982 Richard Feynman and Paul Benioff independently observed that a quantum system can perform a computation.

In 1985 David Deutsch defined quantum Turing machines, a theoretical model for quantum computing.

What is a qubit?

A quantum bit, or “qubit” stores information.

What is a qubit?

A quantum bit, or “qubit” stores information. Physically

- A qubit can be an ion which can occupy different quantum states. The atom in the ground state corresponds to the value “0” and the excited state, “1”.

What is a qubit?

A quantum bit, or “qubit” stores information. Physically

- A qubit can be an ion which can occupy different quantum states. The atom in the ground state corresponds to the value “0” and the excited state, “1”.
- A qubit can be a spin $\frac{1}{2}$ particle which can be in the spin up (“0”) or spin down (“1”) state.

What is a qubit?

A quantum bit, or “qubit” stores information. Physically

- A qubit can be an ion which can occupy different quantum states. The atom in the ground state corresponds to the value “0” and the excited state, “1”.
- A qubit can be a spin $\frac{1}{2}$ particle which can be in the spin up (“0”) or spin down (“1”) state.
- A qubit can be a photon in a vertically polarized state (“1”) or a horizontally polarized state (“0”).

Where do qubits "live"?

Disclaimer: These mathematical objects will become clearer to you once you have taken a class in LINEAR ALGEBRA! You are not expected to understand these statements YET.

A qubit is a quantum object (vector) whose state lies in a two dimensional Hilbert space.

Where do qubits "live"?

Disclaimer: These mathematical objects will become clearer to you once you have taken a class in LINEAR ALGEBRA! You are not expected to understand these statements YET.
A qubit is a quantum object (vector) whose state lies in a two dimensional Hilbert space.

The quantum state of N bits can be expressed as a vector in a space of $dim 2^N$.

Where do qubits “live”?

Disclaimer: These mathematical objects will become clearer to you once you have taken a class in LINEAR ALGEBRA! You are not expected to understand these statements YET.
A qubit is a quantum object (vector) whose state lies in a two dimensional Hilbert space.

The quantum state of N bits can be expressed as a vector in a space of $dim 2^N$.

Dirac Notation - bras and kets

- The elements of Hilbert space H are called **ket vectors**, **state kets** or simply **kets**. Ex. $|0\rangle$

Where do qubits "live"?

Disclaimer: These mathematical objects will become clearer to you once you have taken a class in LINEAR ALGEBRA! You are not expected to understand these statements YET.
A qubit is a quantum object (vector) whose state lies in a two dimensional Hilbert space.

The quantum state of N bits can be expressed as a vector in a space of $dim 2^N$.

Dirac Notation - bras and kets

- The elements of Hilbert space H are called **ket vectors**, **state kets** or simply **kets**. Ex. $|0\rangle$
- Let $H^* = Hom_{\mathbf{C}}(H, \mathbf{C})$.

Where do qubits "live"?

Disclaimer: These mathematical objects will become clearer to you once you have taken a class in LINEAR ALGEBRA! You are not expected to understand these statements YET.
A qubit is a quantum object (vector) whose state lies in a two dimensional Hilbert space.

The quantum state of N bits can be expressed as a vector in a space of $dim 2^N$.

Dirac Notation - bras and kets

- The elements of Hilbert space H are called **ket vectors**, **state kets** or simply **kets**. Ex. $|0\rangle$
- Let $H^* = Hom_{\mathbf{C}}(H, \mathbf{C})$.
- The elements of H^* are called **bra vectors**, **state bras**, or simply **bras**. Ex. $\langle 0|$

How is a qubit different from a regular bit?

Qubits exist in a continuum of states between 0 and 1 (superposition) until they are observed.

How is a qubit different from a regular bit?

Qubits exist in a continuum of states between 0 and 1 (superposition) until they are observed.

$|\Psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ where $|\alpha_0|^2 + |\alpha_1|^2 = 1$

How is a qubit different from a regular bit?

Qubits exist in a continuum of states between 0 and 1 (superposition) until they are observed.

$|\Psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ where $|\alpha_0|^2 + |\alpha_1|^2 = 1$

We can generalize the concept of qubits to quantum registers. A state of a quantum register of size n is a *tensor product* of n qubits and can be written as

$$|\Psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x|x\rangle$$

where

$$\sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1$$

How is a qubit different from a regular bit?

Qubits exist in a continuum of states between 0 and 1 (superposition) until they are observed.

$|\Psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ where $|\alpha_0|^2 + |\alpha_1|^2 = 1$

We can generalize the concept of qubits to quantum registers. A state of a quantum register of size n is a *tensor product* of n qubits and can be written as

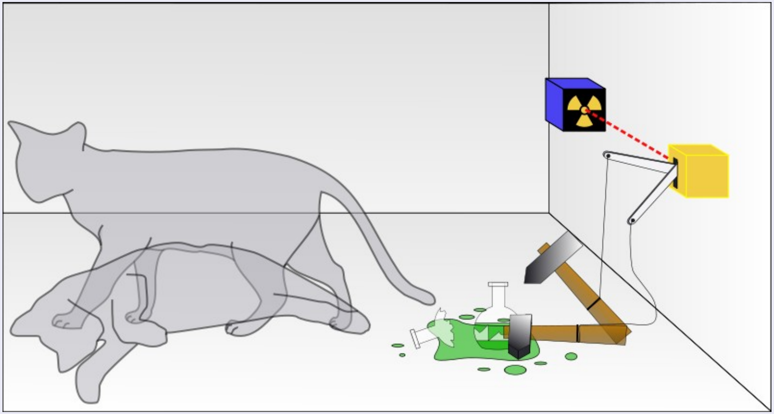
$$|\Psi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x|x\rangle$$

where

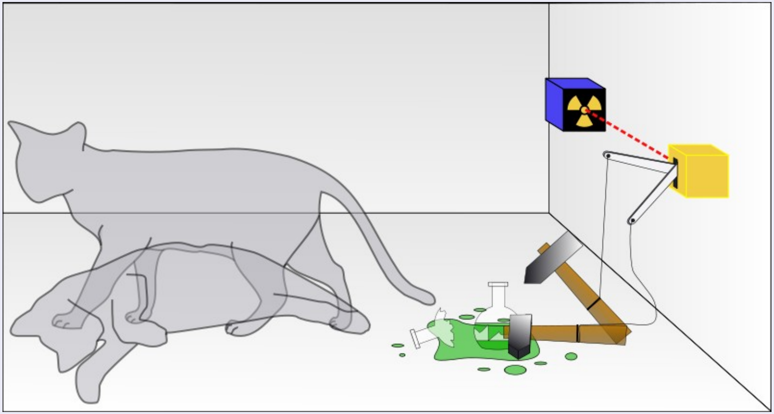
$$\sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1$$

The size of the computational state space of a quantum register is exponential in the physical size of the system.

How can a cat be both dead and alive?



How can a cat be both dead and alive?



Main advantage of quantum computation

- In 1985 David Deutsch found the quantum parallelism principle. This principle allows one to evaluate a function f for distinct inputs simultaneously.

Main advantage of quantum computation

- In 1985 David Deutsch found the quantum parallelism principle. This principle allows one to evaluate a function f for distinct inputs simultaneously.
- Quantum parallelism exploits the superposition of states.

Main advantage of quantum computation

- In 1985 David Deutsch found the quantum parallelism principle. This principle allows one to evaluate a function f for distinct inputs simultaneously.
- Quantum parallelism exploits the superposition of states.
- `https://www.youtube.com/watch?v=IrbJYsep45E`

Reversible Logic Gates

- In 1961 Landauer showed that the only logical operations that require dissipation of energy are irreversible ones.
- Logic gates are typically irreversible.
- The AND-gate is obviously irreversible since given the output $a_f = 0$ we cannot say if the input (a_i, b_i) is equal to $(0, 0)$, $(0, 1)$ or $(1, 0)$.

Reversible Logic Gates

- In 1961 Landauer showed that the only logical operations that require dissipation of energy are irreversible ones.
- Logic gates are typically irreversible.
- The AND-gate is obviously irreversible since given the output $a_f = 0$ we cannot say if the input (a_i, b_i) is equal to $(0,0)$, $(0,1)$ or $(1,0)$.
- The same is true for the OR, XOR, or NOR-gates.

Reversible Logic Gates

- In 1961 Landauer showed that the only logical operations that require dissipation of energy are irreversible ones.
- Logic gates are typically irreversible.
- The AND-gate is obviously irreversible since given the output $a_f = 0$ we cannot say if the input (a_i, b_i) is equal to $(0,0)$, $(0,1)$ or $(1,0)$.
- The same is true for the OR, XOR, or NOR-gates.
- In 1973 Bennett found that any computation can be performed using only reversible steps.

Example of a logic gate

The following is the truth table for the three-bit CONTROL-CONTROL-NOT gate (or CCN-gate).

a_i	b_i	c_i
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Example of a logic gate

The following is the truth table for the three-bit CONTROL-CONTROL-NOT gate (or CCN-gate).

a_i	b_i	c_i
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

a_f	b_f	c_f
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	1
1	1	0

Quantum Logic Gates

- Why are they reversible?

Quantum Logic Gates

- Why are they reversible?
- Both classical and quantum mechanics in the Hamiltonian formulation describe only reversible processes.

Quantum Logic Gates

- Why are they reversible?
- Both classical and quantum mechanics in the Hamiltonian formulation describe only reversible processes.
- Quantum logic gates generally act on a superposition of digital states. They can be represented by operators (matrices).

Quantum Logic Gates

- Why are they reversible?
- Both classical and quantum mechanics in the Hamiltonian formulation describe only reversible processes.
- Quantum logic gates generally act on a superposition of digital states. They can be represented by operators (matrices).
- Unitary matrices represent the time evolution of quantum mechanical systems.

Quantum Logic Gates

- Why are they reversible?
- Both classical and quantum mechanics in the Hamiltonian formulation describe only reversible processes.
- Quantum logic gates generally act on a superposition of digital states. They can be represented by operators (matrices).
- Unitary matrices represent the time evolution of quantum mechanical systems.
- So quantum logic gates can be represented by unitary matrices.

The Quantum N-gate

$$N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

transforms $|0\rangle \to |1\rangle$ and $|1\rangle \to |0\rangle$.

Shor's Algorithm

is a quantum algorithm that factors an integer in polynomial time (1994). If the integer has L digits, Shor's algorithm takes $O(L^2)$ to factor it.

Shor's Algorithm

is a quantum algorithm that factors an integer in polynomial time (1994). If the integer has L digits, Shor's algorithm takes $O(L^2)$ to factor it.

Why should we care?

- The security of the RSA cryptosystem is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem.

Shor's Algorithm

is a quantum algorithm that factors an integer in polynomial time (1994). If the integer has L digits, Shor's algorithm takes $O(L^2)$ to factor it.

Why should we care?

- The security of the RSA cryptosystem is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem.
- For instance, RSA-250, the largest number to be factored to date, has 250 decimal digits (829 bits), and was factored in February 2020. The CPU time spent on finding these factors amounted to more than 900 core-years on a 2.1 Ghz Intel Xeon Gold 6130 CPU!

  See `https://en.wikipedia.org/wiki/RSA_numbers`
- On a classical computer the best known method for factorization of a number with 300 digits takes $5 \cdot 10^{24}$ steps or with terahertz speed **150,000 years**!

Shor's Algorithm

Shor's algorithm hinges on being able to quickly compute the period of the following periodic function (using a quantum algorithm):

$$f(x) = y^x \mod (n)$$

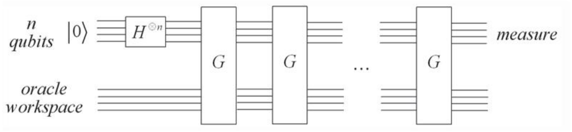
for $x = 0, 1, 2, \ldots$. Where $n = pq$, p and q prime factors.

To begin, select y randomly so that $1 < y < n$ and $(y, n) = 1$. Next, find the period T of the function $f(x) = y^x \mod (n)$. Compute $z = y^{T/2}$. Lastly, to find factors of n, compute $(z+1, n)$ and $(z-1, n)$ (can use the Euclidean Algorithm for this).

The Quantum Search Algorithm

- In 1996 Lov K. Grover gave a quantum algorithm for searching an unsorted list with N entries.
- By having the input and output in superpositions of states one can find an object in $O(N^{1/2})$ quantum mechanical steps instead of $O(N)$ classical steps for the worst-case linear search.

The Quantum Search Algorithm

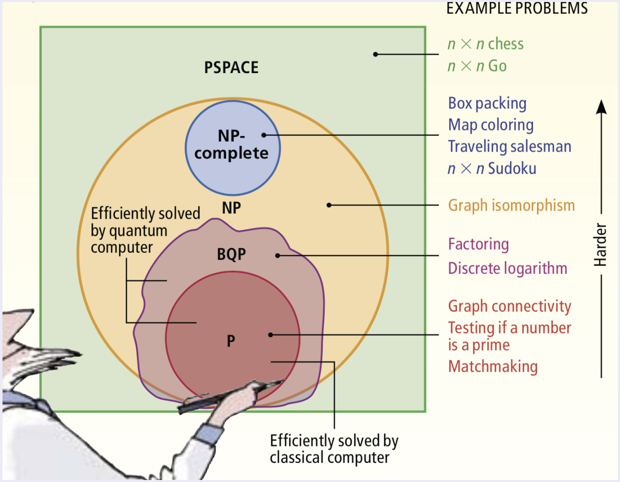


In order to achieve the correct solution with probability near 1, one must apply the Grover iteration $O\left(\sqrt{N}\right)$ times if there is a single solution.

Conclusions:

- Until today there are only two basic quantum algorithmic methods known.
  - **Shor's Algorithm** for the factorization of integers.
  - **Grover's Search Algorithm**.
- Quantum algorithms have the potential to demonstrate that for some problems quantum computation is more efficient than classical computation.
- A goal is to determine for which problems quantum computers are faster than classical computers.
- Two important quantum complexity classes are BQP and QMA which are the bounded-error quantum analogues of P and NP.
- Goal: Find out where these classes lie with respect to classical complexity classes such as P, NP, PP (problems solvable by probabilistic Turning machine in poly-time), PSPACE (problems that can be solved by a Turing machine using a poly-space) and other complexity classes.

Our best guess:



The diagram depicts the class of problems a quantum computer would solve efficiently, BQP ("bounded error, probabilistic, polynomial time"), might relate to other fundamental classes of computational problems.

Google Sycamore vs. IBM Summit: a battle for supremacy

- A paper by Google's quantum computer research team published in October 2019 in *Nature*, claims the project has reached quantum supremacy.

Google Sycamore vs. IBM Summit: a battle for supremacy

- A paper by Google's quantum computer research team published in October 2019 in *Nature*, claims the project has reached quantum supremacy.
- The task: Compute the probability distribution of all of the possible outputs from a quantum psuedo-random number generator.

Google Sycamore vs. IBM Summit: a battle for supremacy

- A paper by Google's quantum computer research team published in October 2019 in *Nature*, claims the project has reached quantum supremacy.
- The task: Compute the probability distribution of all of the possible outputs from a quantum psuedo-random number generator.
- Quantum operations were used to generate random numbers - this is VERY difficult to simulate with a classical machine.

Google Sycamore vs. IBM Summit: a battle for supremacy

- A paper by Google's quantum computer research team published in October 2019 in *Nature*, claims the project has reached quantum supremacy.
- The task: Compute the probability distribution of all of the possible outputs from a quantum psuedo-random number generator.
- Quantum operations were used to generate random numbers - this is VERY difficult to simulate with a classical machine.
- Sycamore, Google's 53 qubit quantum computer performed the task in 200 seconds.

Google Sycamore vs. IBM Summit: a battle for supremacy

- A paper by Google's quantum computer research team published in October 2019 in *Nature*, claims the project has reached quantum supremacy.
- The task: Compute the probability distribution of all of the possible outputs from a quantum psuedo-random number generator.
- Quantum operations were used to generate random numbers - this is VERY difficult to simulate with a classical machine.
- Sycamore, Google's 53 qubit quantum computer performed the task in 200 seconds.
- Google claims that Summit, IBM's supercomputer at Oak Ridge National Labs, would greater than 10,000 years to perform the same task i.e. the task intractable classic ally.

Google Sycamore vs. IBM Summit: a battle for supremacy

- A paper by Google's quantum computer research team published in October 2019 in *Nature*, claims the project has reached quantum supremacy.
- The task: Compute the probability distribution of all of the possible outputs from a quantum psuedo-random number generator.
- Quantum operations were used to generate random numbers - this is VERY difficult to simulate with a classical machine.
- Sycamore, Google's 53 qubit quantum computer performed the task in 200 seconds.
- Google claims that Summit, IBM's supercomputer at Oak Ridge National Labs, would greater than 10,000 years to perform the same task i.e. the task intractable classic ally.
- IBM claims it would take 2.5 days by running a different algorithm.

Google Sycamore vs. IBM Summit: a battle for supremacy

- A paper by Google's quantum computer research team published in October 2019 in *Nature*, claims the project has reached quantum supremacy.
- The task: Compute the probability distribution of all of the possible outputs from a quantum psuedo-random number generator.
- Quantum operations were used to generate random numbers - this is VERY difficult to simulate with a classical machine.
- Sycamore, Google's 53 qubit quantum computer performed the task in 200 seconds.
- Google claims that Summit, IBM's supercomputer at Oak Ridge National Labs, would greater than 10,000 years to perform the same task i.e. the task intractable classic ally.
- IBM claims it would take 2.5 days by running a different algorithm.
- What does this mean?

Google Sycamore vs. IBM Summit: a battle for supremacy

- A paper by Google's quantum computer research team published in October 2019 in *Nature*, claims the project has reached quantum supremacy.
- The task: Compute the probability distribution of all of the possible outputs from a quantum psuedo-random number generator.
- Quantum operations were used to generate random numbers - this is VERY difficult to simulate with a classical machine.
- Sycamore, Google's 53 qubit quantum computer performed the task in 200 seconds.
- Google claims that Summit, IBM's supercomputer at Oak Ridge National Labs, would greater than 10,000 years to perform the same task i.e. the task intractable classic ally.
- IBM claims it would take 2.5 days by running a different algorithm.
- What does this mean?The jury is out.

Thank you!

References:

- Preskill's podcast and notes: https://quantumfrontiers.com/author/preskill/
- A timeline on quantum computation: https://en.wikipedia.org/wiki/Timeline_of_quantum_computing
- An article, "The Limits of Quantum Computers" https://www.cs.virginia.edu/~robins/The_Limits_of_Quantum_Computers.pdf
- The "bible" for learning quantum computation: http://csis.pace.edu/ctappert/cs837-18spring/QC-textbook.pdf
- Another perspective: https://www.worldscientific.com/worldscibooks/10.1142/3808

Additional materials:

RSA cryptosystems

In the RSA (Rivest, Shamir and Adleman)cryptosystem, each individual has an encryption key (n, e) where $n = pq$ the modulus is the product of two large primes p and q, (say with 200 digits each), and an exponent e that is relatively prime to $(p-1)(q-1)$

Additional materials:

RSA cryptosystems

In the RSA (Rivest, Shamir and Adleman)cryptosystem, each individual has an encryption key (n, e) where $n = pq$ the modulus is the product of two large primes p and q, (say with 200 digits each), and an exponent e that is relatively prime to $(p-1)(q-1)$

To encrypt a message, translate the text to numerical equivalents, break into blocks m_i, and then use fast modular exponentiation to compute they encrypted blocks c_i using the function $c_i = m_i^e \mod n$.

Additional materials:

RSA cryptosystems

In the RSA (Rivest, Shamir and Adleman)cryptosystem, each individual has an encryption key (n, e) where $n = pq$ the modulus is the product of two large primes p and q, (say with 200 digits each), and an exponent e that is relatively prime to $(p-1)(q-1)$

To encrypt a message, translate the text to numerical equivalents, break into blocks m_i, and then use fast modular exponentiation to compute they encrypted blocks c_i using the function $c_i = m_i^e \mod n$.

In order to *decrypt* a message sent in this scheme, once must be able to find the decyrption key d which is the inverse of $e, \mod (p-1)(q-1)$. Without knowing p and q, and only knowing n (with possibly 400 digits), this system is difficult to break!

Shor's Algorithm: An example

Factoring $n = 30$ using Shor's Algorithm

- Given $n = 30$, choose a y, so that $1 < y < 30$ and $(y, 30) = 1$. I will use $y = 11$ (but I could have chosen: 19, 29, 7, 13 or 23).
- Compute values of $f(x)$ to find the period of $f(x)$ (by hand):

$$f(0) = 11^0 \mod 30 = 1$$

$$f(1) = 11^1 \mod 30 = 11$$

$$f(2) = 11^2 \mod 30 = 1$$

$$f(3) = 11^3 \mod 30 = 11$$

$$f(4) = 11^2 \mod 30 = 1 \ldots$$

- T is obvious 2. Now compute $z = y^{T/2} = 11^{2/2} = 11^1 = 11$.
- Finally factors of $n = 30$ are $(z + 1, 30) = (12, 30) = 6$ and $(z - 1, 30) = (10, 30) = 10$. Both 6 and 10 are factors of 30.

Factoring using Shor's Algorithm (cont.)

- This factoring method fails sometimes, for instance, when the period T is odd. HOWEVER,

Factoring using Shor's Algorithm (cont.)

- This factoring method fails sometimes, for instance, when the period T is odd. HOWEVER,
- If y ($1 < y < n$) is randomly selected, Ekert and Jozsa showed that the probability that two numbers have $gcd = 1$ is greater than $1/\log_2(n)$ so the probability of failure is small. See `https://doi.org/10.1103/RevModPhys.68.733`.
- Also note, that for $y = 11, 19$, and 29, $T = 2$ and for $y = 7, 13$, and 23, $T = 4$.

Your turn to try!

Use Shor's algorithm to factor $n = 21$. You may use our Python code for fast modular exponentiation to help you find the period of $f(x)$ and the code for computing the gcd if you would like.
`https://trinket.io/python/653108cbc9`
`https://trinket.io/python/2ee4b2d236`

The Quantum Search Algorithm: more details

Suppose one wants to search through N elements, indexed $0, 1, 2, \ldots,$ $N - 1$.

The Quantum Search Algorithm: more details

Suppose one wants to search through N elements, indexed $0, 1, 2, \ldots,$ $N-1$.

Assume $N = 2^n$ so the index is stored in n bits and that the search problem has one solution.

Define a function $f(x)$, such that for $x \in \{0, 1, \ldots, N-1\}$

$$f(x) = \begin{cases} 0 & \text{if } x \text{ is not a solution} \\ 1 & \text{if } x \text{ is a solution.} \end{cases}$$

The Quantum Search Algorithm: more details

An oracle is supplied which has the ability to recognize the solutions for the search problem. Recognition is signalled by the use of an oracle qubit $|q\rangle$.

The oracle, O, is a unitary operator defined on the computational basis $|0\rangle$, $|1\rangle$.
The action of the oracle is

$$O|x\rangle|q\rangle = |x\rangle|q \oplus f(x)\rangle$$

where x is the index register and $|q\rangle$ is the oracle qubit.

Grover's Algorithm: geometric visualization

One can view the Grover iteration as a rotation in the two dimensional Hilbert space spanned by the initial vector $|\psi\rangle$ and the state that consists of the superposition of the solutions to the search problem.

- Let the notation $\sum_x{}'$ indicate the sum over all the x which are solutions to the search problem.
- Let $\sum_x{}''$ indicate the sum over all the x which are not solutions.

Then one can describe the following two states:

- $|\alpha\rangle = \frac{1}{\sqrt{N-M}}\sum_x{}''|x\rangle$ and
- $|\beta\rangle = \frac{1}{\sqrt{M}}\sum_x{}'|x\rangle$

The Quantum Search Algorithm: geometric visualization

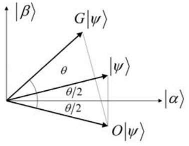


Figure: The action of a single Grover iteration G.

- No matter how many times G is applied to $|\psi\rangle$, the vector remains in the plane spanned by $|\alpha\rangle$ and $|\beta\rangle$.
- In fact, one also knows the angle of rotation since $\cos\left(\frac{\theta}{2}\right) = \sqrt{\frac{(N-M)}{N}}$.
- Each application of G rotates the vector $|\psi\rangle$ closer to alignment with the vector $|\beta\rangle$.

- When this occurs, a measurement in the computational basis produces one of the outcomes superimposed on $|\beta\rangle$ with high probability.
- This is a solution to the search problem.