

Quantum Algorithms

Stephen Bartlett



The University of Sydney
AUSTRALIA
School of Physics

www.physics.usyd.edu.au/~bartlett

Outline

- Introduction to algorithms
 - ◆ Define algorithms and computational complexity
 - ◆ Discuss factorization as an important algorithm for information security
- Quantum algorithms
 - ◆ What they contribute to computing and cryptography
 - ◆ Shor's quantum algorithm for efficient factorization
 - ◆ Quantum search algorithms
 - ◆ Demonstrations of quantum algorithms
 - ◆ Ongoing quantum algorithms research

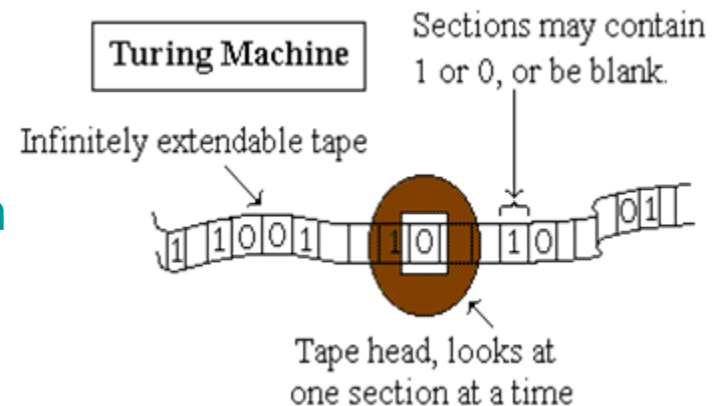
What is an algorithm?

- Consider a *problem* where each *instance* has a *solution*
 - ◆ Example of a problem: Is an integer p a prime number?
 - ◆ The instance: a particular choice of integer
 - ◆ The solution: either *yes* or *no* (a decision problem)
- *Algorithm*: a detailed step-by-step method for solving a problem
 - ◆ Example algorithm: a program $\text{PRIMALITY}(p)$ that runs on a computer and gives *yes* or *no* for any input integer p



Alan Turing

Computer: a universal machine that can implement any algorithm

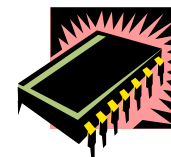


Example: discrete Fourier transform

- Problem: for a given vector (x_j) , $j=1,\dots,N$, what is the discrete Fourier transform (DFT) vector

$$y_j = \sum_{k=1}^N \exp(2\pi i(j-1)(k-1)/N) x_k$$

- Algorithm: a detailed step-by-step method to calculate the DFT (y_j) for any instance (x_j)
- With such an algorithm, one could:
 - ◆ write a DFT program to run on a computer
 - ◆ build a custom chip that calculates the DFT
 - ◆ train a team of children to execute the algorithm



Why are algorithms important?

■ Scientific applications

Simulating quantum systems

Database searching (e.g. genetic code)

Computer networking

■ Information security

Secure communication

Digital signatures

Privacy of information

Identification

e-commerce

■ Theory of computing

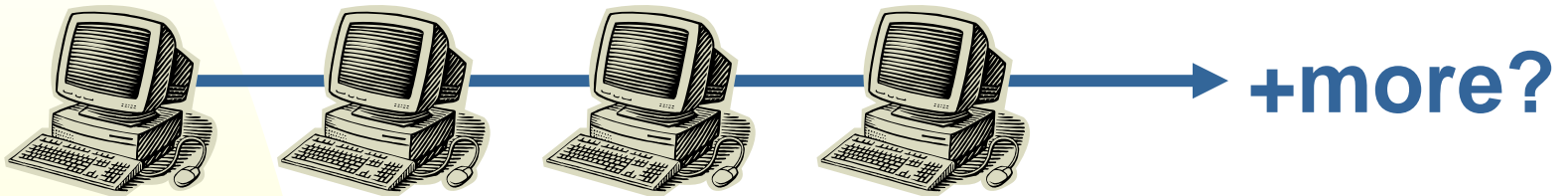
Limits of computation

Computational models

Complexity and computability

Computational complexity

- Consider an algorithm that solves a given problem
- Question: how much computing power do I need to execute this algorithm for a given input (instance) size?



- Let N be an integer describing the size of our instance
 - ◆ Example: N could be the number of bits needed to write the input in memory
- How does the number of steps in our algorithm depend on N ? (Definition of “steps” is a bit arbitrary, but the choice doesn’t affect scaling)

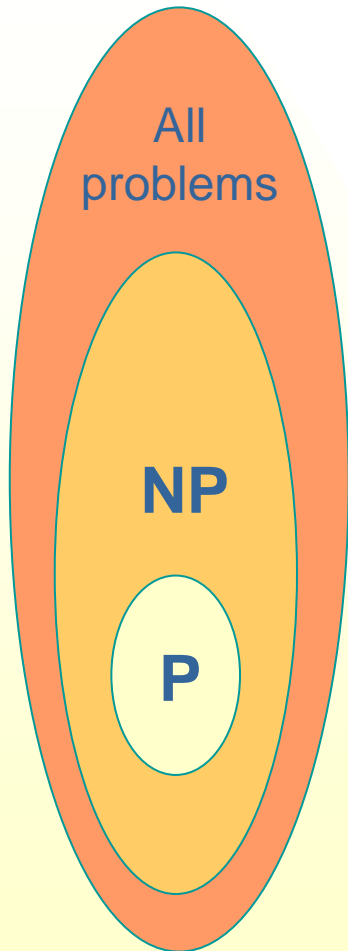
Computational complexity of DFT

- For the DFT, N could be the dimension of the vector

$$y_j = \sum_{k=1}^N \exp(2\pi i(j-1)(k-1)/N)x_k$$

- To calculate each y_j , must sum N terms
- This sum must be performed for N different y_j
- Computational complexity of DFT: requires N^2 steps
- DFTs are important → a lot of work in optical computing (1950s, 1960s) to do fast DFTs
- 1965: Tukey and Cooley invent the Fast Fourier Transform (FFT), requires $N \log N$ steps
- FFT much faster → optical computing almost dies overnight

Complexity classes - P and NP



Naively categorise problems:

- **P**: the set of problems with an algorithm that requires resources that are polynomial in the size of the problem
 - ◆ Problems in **P** are considered “solvable”
 - ◆ Not the whole story: an algorithm that scales as N^{100} is not easy in practice
 - ◆ Both DFT and FFT are in **P** but FFT requires fewer resources
- **NP**: the set of problems for which a “guessed” solution can be checked using polynomial resources
 - ◆ Some problems in **NP** can be used for cryptography (data encryption, secure communication, etc.)

P = NP ?

Example: Factoring

- Factoring: given a number, what are its prime factors?
- Considered a “hard” problem in general, especially for numbers that are products of 2 large primes

Example: $4633 = 41 \times 113$

RSA-129

1143816257578888676692357799761466120102182 96721242362562561842935706935245733897830597123563958705058989075147599290026879543541 =
3490529510847650949147849619903898133417764638493387843990820577 x 32769132993266709549961988190834461413177642967992942539798288533

- Best factoring algorithm requires resources that grow exponentially in the size of the number (RSA-129 took 17 years)
- Example: factor a 300-digit number
 - ◆ Best algorithm: takes 10^{24} steps
 - ◆ On computer at THz speed: 150,000 years
- Difficulty of factoring is the basis of security for the RSA encryption scheme used, e.g., on the internet
- Information security of interest to private and public sectors



Quantum algorithms



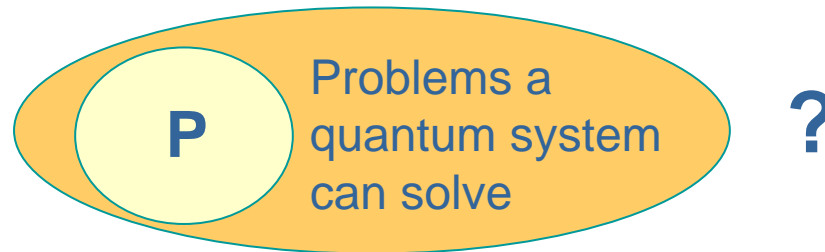
Richard Feynman

- Feynman (1982): there may be quantum systems that cannot be simulated efficiently on a “classical” computer
- Deutsch (1985): proposed that machines using quantum processes might be able to perform computations that “classical” computers can only perform very poorly



David Deutsch

- Concept of *quantum computer* emerged as a universal device to execute such quantum algorithms

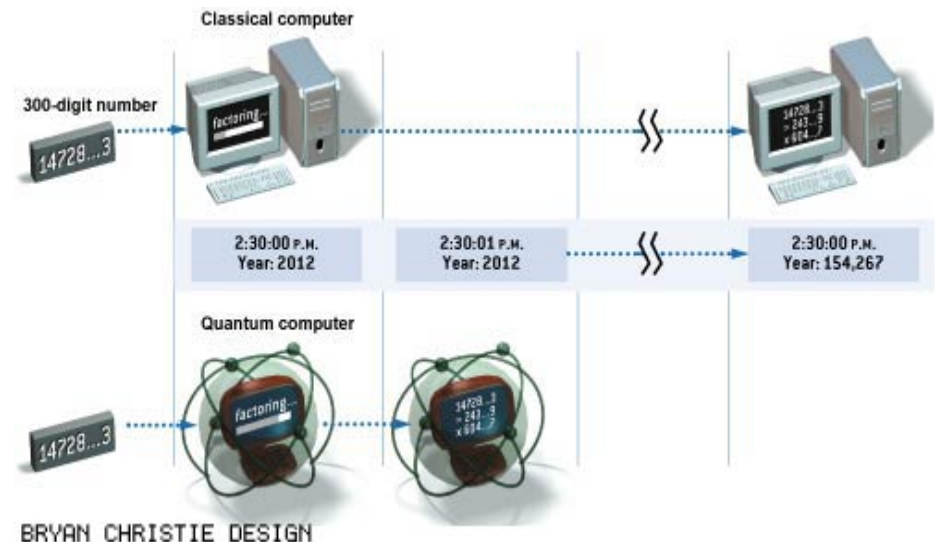


Factoring with quantum systems

- Shor (1995): quantum factoring algorithm

Example: factor a 300-digit number

Best classical algorithm: 10^{24} steps	Shor's quantum algorithm: 10^{10} steps
On classical THz computer: 150,000 years	On quantum THz computer: <1 second



Scientific American, Nov 2002

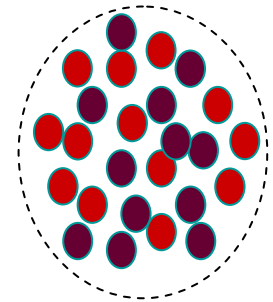
- To implement Shor's algorithm, one could:
 - run it as a program on a "universal quantum computer"
 - design a custom quantum chip with hard-wired algorithm
 - find a quantum system that does it naturally! (?)

Implications

- Information security and e-commerce are based on the use of **NP** problems that are not in **P**
 - ◆ must be “hard” (not in **P**) so that security is unbreakable
 - ◆ requires knowledge/assumptions about the algorithmic and computational power of your adversaries
- Quantum algorithms (e.g., Shor’s factoring algorithm) require us to reassess the security of such systems
- Lessons to be learned:
 - ◆ algorithms and complexity classes can change!
 - ◆ information security is based on assumptions of what is hard and what is possible → better be convinced of their validity

Search problems

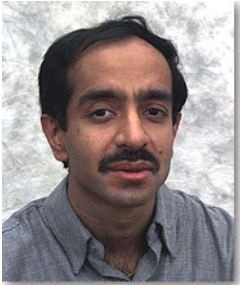
- Problem 1: Given an unsorted database of N items, how long will it take to find a particular item x ?
 - ◆ Check items one at a time. Is it x ?
 - ◆ Average number of checks: $N/2$
- Problem 2: Given an unsorted database of N items, each either red or black, how many are red?
 - ◆ Start a tally
 - ◆ Check items one at a time. Is it red?
 - ★ If it is red, add one to the tally
 - ★ If it is black, don't change the tally
 - ◆ Must check all items: requires N checks
- Not surprisingly, these are the best (classical) algorithms
- We can define quantum search algorithms that do better



Oracles

- We need a "quantum way" to recognize a solution
- Define an *oracle* to be the unitary operator
$$O : |x\rangle|q\rangle \rightarrow |x\rangle|q \oplus f(x)\rangle$$
where $|q\rangle$ is an ancilla qubit
- Could measure the ancilla qubit to determine if x is a solution
- Doesn't this "oracle" need to know the solution?
 - ◆ It just needs to recognize a solution when given one
 - ◆ Similar to **NP** problems
- One oracle call represents a fixed number of operations
- Address the complexity of a search algorithm in terms of the number of oracle calls → separates scaling from fixed costs

Quantum searching



Lov Grover

Bell Labs

- Grover (1996): quantum search algorithm
- For M solutions in a database containing N elements:

Classical search	Quantum search
N/M oracle calls	$(N/M)^{1/2}$ oracle calls

- Quantum search algorithm works by applying the oracle to superpositions of all elements, and increases the amplitude of solutions (viewed as states)
- Quantum search requires that we know M/N (at least approximately) prior to the algorithm, in order to perform the correct number of steps
- Failure to measure a solution \rightarrow run the algorithm again

Quantum counting

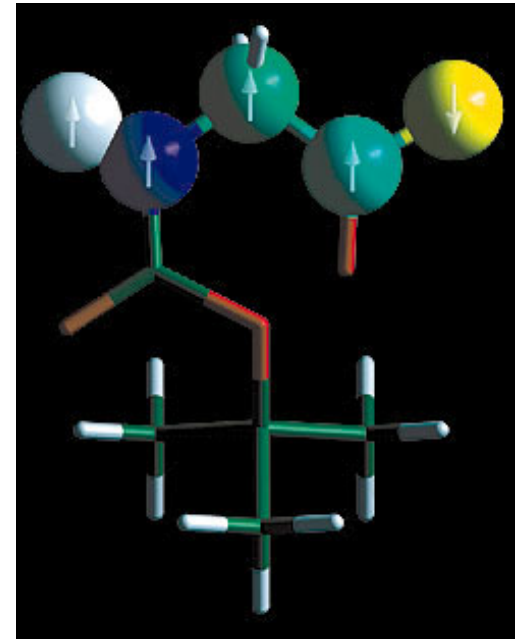
- What if the number of solutions M is not known?
- Need M in order to determine the number of iterations of the Grover operator
- Classical algorithm requires N steps
- Quantum algorithm: Use phase estimation techniques
 - ◆ based on quantum Fourier transform (Shor)
 - ◆ requires $N^{1/2}$ oracle calls
- For a search with unknown number of solutions:
 - ◆ First perform quantum counting: $N^{1/2}$
 - ◆ With M , perform quantum search: $N^{1/2}$
 - ◆ Total search algorithm: still only $N^{1/2}$

Can we do better?

- Quantum search algorithm provides a quadratic speedup over best classical algorithm
 - Classical: N steps
 - Quantum: $N^{1/2}$ steps
- Maybe there is a better quantum search algorithm
- Imagine one that requires $\log N$ steps:
 - ◆ Quantum search would be exponentially faster than any classical algorithm
 - ◆ Used for **NP** problems: could reduce them to **P** by searching all possible solutions
- Unfortunately, NO: Quantum search algorithm is "optimal"
- Any search-based method for **NP** problems is slow

Demonstrations of quantum algorithms

- Initial demonstrations of quantum algorithms have been performed using NMR quantum computing
 - ◆ 1997: Grover's quantum searching algorithm on a 2-qubit quantum computer
 - ◆ 2001: Shor's factorization algorithm on a 7-qubit quantum computer to factorize 15
- NMR quantum computing demonstrates the principle, but cannot "scale up" beyond a few qubits
- New scalable architectures (e.g., silicon-based, photonic) are necessary to perform useful computations



A molecule containing
5 NMR spin qubits
(Munich NMR QC group)

How do quantum algorithms work?

- What makes a quantum algorithm potentially faster than any classical one?
 - ◆ Quantum parallelism: by using superpositions of quantum states, the computer is executing the algorithm on all possible inputs at once
 - ◆ Dimension of quantum Hilbert space: the “size” of the state space for the quantum system is exponentially larger than the corresponding classical system
 - ◆ Entanglement capability: different subsystems (qubits) in a quantum computer become entangled, exhibiting nonclassical correlations
- We don't really know what makes quantum systems more powerful than a classical computer
- Quantum algorithms are helping us understand the computational power of quantum vs classical systems

Summary of quantum algorithms

- It may be possible to solve a problem on a quantum system much faster (i.e., using fewer steps) than on a classical computer
- Factorization and searching are examples of problems where quantum algorithms are known and are faster than any classical ones
- Implications for cryptography, information security
- Study of quantum algorithms and quantum computation is important in order to make assumptions about adversary's algorithmic and computational capabilities
- Leading to an understanding of the computational power of quantum vs classical systems