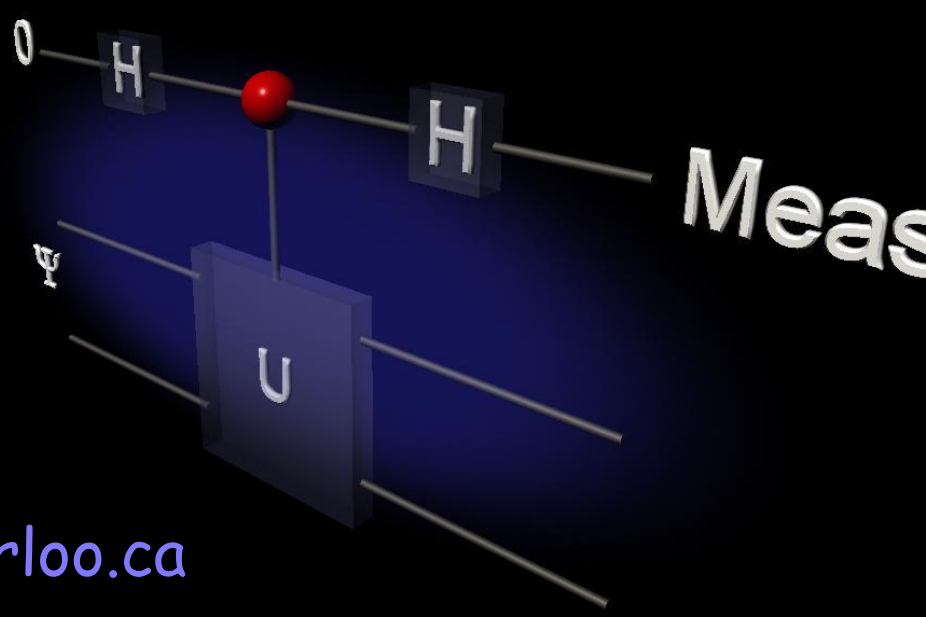
Introduction to Quantum Information Processing



CO481 CS467 PHYS467

Michele Mosca mmosca@iqc.uwaterloo.ca

Tuesdays and Thursdays 10am-11:15am

IQC Institute for Quantum Computing

UNIVERSITY OF WATERLOO

PI PERIMETER INSTITUTE FOR THEORETICAL PHYSICS

Overview

Lecture 8

- Computational complexity
- Big numbers…
- Polynomial complexity
- Simulating classical computation with quantum computation
- General purpose simulation algorithm

Computational Complexity

- *The (computational) complexity* of an **algorithm** refers to some measure of the resources *(e.g. time, space, basic operations, energy)* used by the algorithm.
- E.g. the traditional algorithm for multiplying two n-bit numbers takes $O(n^2)$ **time** steps (with pen and paper, or on a PC).
- E.g. the best-known rigorous probabilistic classical algorithms for factoring an n-digit number into its prime factors with high probability takes **time** in

$$e^{O\left(\sqrt{n \log n}\right)}.$$

- We measure the complexity as a function of the input size.
- Unless stated otherwise, we refer to the “worst-case” complexity, i.e. the running time on a worst-case input.

Computational Complexity

- *The (computational) complexity of a* ***problem*** refers to some measure of the resources (e.g. *time, space, basic operations, energy*) required to solve a problem.
- E.g. multiplying two *n*-bit numbers must take $\Omega(n)$ **time** (even just to write the answer).
- No known algorithm achieves that lower bound.
  - Until 2007, the best known upper bound was (Schönhage-Strassen, 1971) $O(n \log n \log\log n)$
  - In 2007, this was improved (Fürer) to $O(n \log n 2^{\log^* n})$
- Unless stated otherwise, we refer to the "worst-case" complexity, i.e. the resources required to solve the problem on any input of size *n*.

Aside : Computationally secure cryptography

Why do we believe the current cryptographic tools are secure?

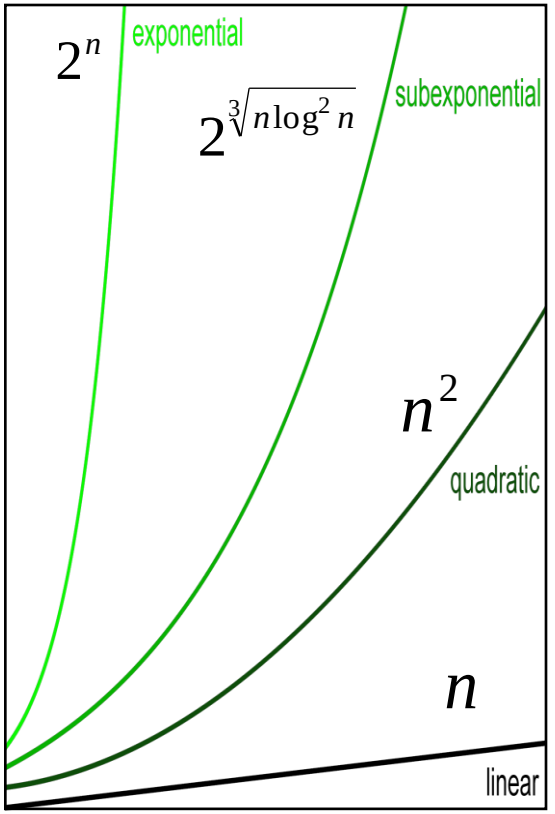
- It is easy to multiply two large numbers 9287137268623 **x** 61649017818402059 = 572542890955285156447699294757

- It takes between n and n^2 steps to multiply two n-digit numbers.

- It is not known how to easily factor an arbitrary large (non-prime) number (e.g. 652186080857407065896971) into a product of smaller numbers.

- Best-known heuristic classical methods take roughly $2^{\sqrt[3]{n\log^2 n}}$ steps.

- The best known classical methods for breaking elliptic curve cryptography with n digit keys take roughly 2^n steps.



Really big numbers …

A million GHz computers, running for 100 years, can perform about 2^{71} basic operations

10^6 computers x 10^9 cycles/second/computer x 31536000 seconds/year x 100 years
$\approx 2^{20} \times 2^{30} \times 2^{24} \times 2^7 = 2^{71}$ cycles

A billion THz computer, running for 1000 years, can perform about 2^{104} basic operations

10^9 computers x 10^{12} cycles/second/computer x 31536000 seconds/year x 10^3 years
$\approx 2^{30} \times 2^{40} \times 2^{24} \times 2^{10} = 2^{104}$ cycles

Approximate age of the universe: 2^{92} seconds

Big numbers …

www.keylength.com

The best known classical algorithm for cracking 256-bit Elliptic curve crypto requires about 2^{128} steps

The best known classical algorithm for cracking 2030-bit RSA requires about 2^{128} steps

The best known classical algorithm for cracking 128-bit AES (a widely used symmetric key cipher) requires about 2^{128} steps

“Polynomial” cost

- We say an algorithm A runs in “polynomial” time if there exists a polynomial p(n) such that the algorithm takes time at most p(n) on inputs of size n.

- One can similarly talk about algorithms that use polynomial space, or a polynomial number of logic gates.

- Algorithm A simulates algorithm B with “polynomial overhead” (in time e.g.) if there is some polynomial p(n) such that when algorithm B uses T time then algorithm A takes time at most p(T) when simulating algorithm B. (similarly for other resources)

Why polynomials ??

- Polynomials are closed under addition, multiplication and composition.

- Any known realistic classical computing model can be simulated with polynomial overhead by a classical (probabilistic) Turing machine (or a PC with arbitrary memory and random coins). Thus, in these cases, the Strong Church-Turing thesis holds.

- In general, measuring complexity up to a polynomial factor gives a certain degree of robustness (e.g. against reasonable changes in computing model, and other “details”)

Polynomial complexity ≈ Efficient

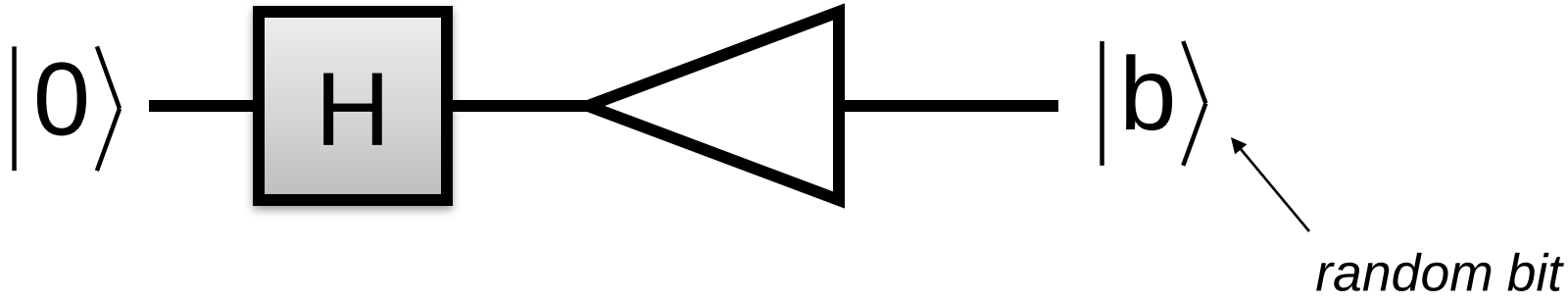
"It should not come as a surprise that our choice of polynomial algorithms as the mathematical concept that is supposed to capture the informal notion of 'practically efficient computation' is open to criticism from all sides. […]

Ultimately, our argument for our choice must be this: **Adopting polynomial worst-case performance as our criterion of efficiency results in an elegant and useful theory that says something meaningful about practical computation, and would be impossible without this simplification**" – Christos Papadimitriou

Simulating classical computation with quantum computation

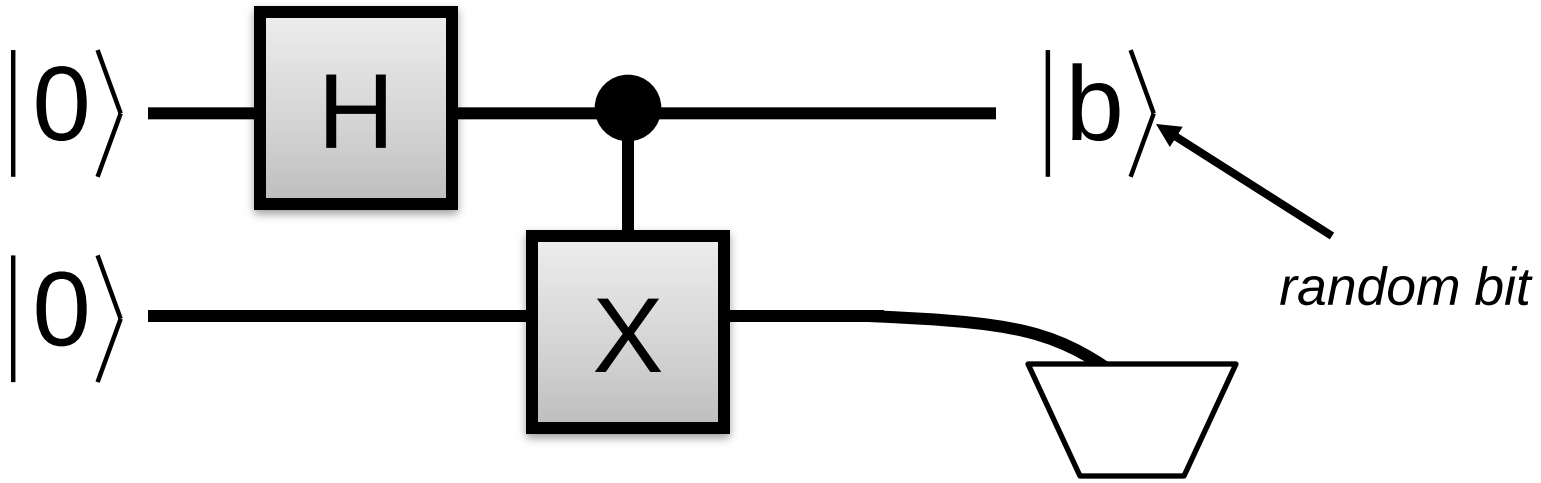
Probabilistic classical computation can be simulated efficiently by reversible quantum computation, by discarding (or measuring) some qubits (Exercise 6.1.1).

Reversible quantum computation can thus efficiently simulate probabilistic classical computation.



Alternatively …

Probabilistic classical computation can be simulated efficiently by reversible quantum computation, by discarding (or measuring) some qubits (Exercise 6.1.1).



Simulating quantum computation with classical computation

There are classical algorithms that can simulate special types of quantum algorithms.

Example:

Gottesman-Knill theorem: Quantum circuits using only **CNOT, H, X, Y, Z** gates (which generate the “Clifford group”), and starting with computational basis states, can be efficiently simulated on a classical computer.

Simulating quantum computation with classical computation

However, there are no known classical algorithms that will efficiently simulate any quantum algorithm.

It is widely believed that no such efficient simulation exists.

Simulating quantum computation with classical computation

One can however, simulate a quantum computation (starting with computational basis states) using exponential time on a classical computer.

General purpose simulation algorithm

***Input**:*

- A reasonable description of a quantum circuit with *T* gates in $G = \{\mathbf{H}, \mathbf{T}, \mathrm{CNOT}\}$ with all-zeroes input, using *n* qubits of memory.

- A precision parameter ε .

Output:

The probability (with precision ε) that measuring the rightmost qubit (in computational basis) will yield outcome “0”.

For convenience, label the *T* gates $G_1, G_2, \ldots, G_T$.

General purpose simulation algorithm

Algorithm (sketch):

Initialize a list of $N=2^n$ numbers, $a_0, a_1, \ldots, a_{N-1}$, each with k bits of precision, where $k = 100n\log(T/\epsilon)$

Set $a_0 = 1$, $a_1 = a_2 = \ldots = a_N = 0$. These represent the initial amplitudes of the N basis states (where a_i is the amplitude of $|i\rangle$, and i is represented in binary).

For i = 0,1,2, …, N-1:

Update the list of amplitudes according to the action of gate G_i.

General purpose simulation algorithm

For i = 0,1,2, …, N-1:

Update the list of amplitudes according to the action of gate G_i.

NB: *Perform every addition and multiplication with round-off error in*

$$O\left(\frac{\varepsilon}{2^n T}\right)$$

Each such computation can be done with complexity polynomial in

$$n + \log\left(\frac{T}{\varepsilon}\right)$$

General purpose simulation algorithm

Denote the final amplitudes by: b_0, b_1, …, b_{N-1}.

Compute

$$\sum_{\text{even } i} |b_i|^2$$

This simulation involved $O(2^n T)$ arithmetic operations involving complex numbers represented with $O(n + \log(T/\varepsilon))$ bits of precision.

Overall time cost:

$$O\left(2^n T \text{poly}\left(n, \log T, \log \frac{1}{\varepsilon}\right)\right)$$

General purpose simulation algorithm

Overall time cost:

$$O\left(2^n T \text{poly}\left(n, \log T, \log \frac{1}{\varepsilon}\right)\right)$$

Memory (“space”) used is in

$$O\left(2^n \text{poly}\left(n, \log T, \log \frac{1}{\varepsilon}\right)\right)$$

One can actually reduce the space required to polynomial in

$$n, \log T, \log \frac{1}{\varepsilon}$$

“Effective” vs “Efficient” computation

Note that this kind of simulation means that quantum computers can’t compute anything not “computable” by a classical computer (i.e. what is “effectively” computable).

Quantum computers likely change what is “efficiently” computable.