On nature of *Computing*

Subhashis Banerjee, Subodh Sharma

Department of Computer Science and Engineering
IIT Delhi

October 17, 2020

Computer programming is an art form, like the creation of poetry or music - Donald Knuth

If computers that you build are quantum,
Then spies everywhere will all want 'em.
Our codes will all fail,
And they'll read our email,
Till we get crypto that's quantum, and daunt 'em.

- Jennifer and Peter Shor

To read our E-mail, how mean
of the spies and their quantum machine;
be comforted though,
they do not yet know
how to factorize twelve or fifteen.

- Volker Strassen

Computing: Hardware models

- Abacus (*Greek Abakos; Hebrew Abaq*).
- Mechanical digital calculator: Blaise Pascal (1642), Leibnitz (1671). Leibnitz's version was commercially available in 1820.
- Charles Babbage's *Analytic engine* (1822). Could compute polynomial functions. Had *if-then-else* and *while-do*. Friend Ada Augusta is considered to be the first computer programmer.
- James Thomson's (1876) *mechanical wheel-and-disc integrator* became the foundation of *analog computation*. Were in use during World War I for gunnery calculations.

Computing: Hardware models

- Herman Hollerith's (1880) electro-mechanical sensing system. Used for the 1880 US census.
- Konrad Zuse's (1939) *Z1*, *Z2*, *Z3* and *Z4*. *Z3* was based on discarded telephone relays. Used George Bool's algebra. Programming notation *Plankalkul*.
- Howard Aiken's *Harvard Mark I* (1944; joint effort of IBM and Harvard). Based on electro-mechanical relays, very similar to *Z3* but larger.
- John Mauchly and Presper Eckert's (1946; Moore school, Pennsylvania) *ENIAC (Electronic Numeric Integrator and Automatic Calculator)*. Used vacuum tubes for switching.
- John von Neumann's (1945) *architecture* and the **RAM model**.

Early algorithms

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times \underline{(n-1)!} & \text{otherwise} \end{cases}$$

$O(n)$ multiplications. Euclid's *Elements*. 300 BC.

$$gcd(m, n) = \begin{cases} m & \text{if } n = 0 \\ \underline{gcd(n, m \bmod n)} & \text{otherwise} \end{cases}$$

$O(\log n)$ steps. Euclid's *Elements*. 300 BC.

Early algorithms

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x \times \underline{x^{n-1}} & \text{otherwise} \end{cases}$$

$O(n)$ multiplications. Dates back to the Egyptians. 2000 BC.

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x \times sqr(\underline{x^{n/2}}) & \text{if } odd(n) \\ sqr(\underline{x^{n/2}}) & \text{if } even(n) \end{cases}$$

$O(\log n)$ multiplications. Acharya Pingala in *Chandah Sutra*. 200 BC.

Models of computation

Recursive functions: Inductively defined functions $f : \mathbb{N}^n \to \mathbb{N}$

RAM model: Any programming language that supports **assignment**, **if-then-else**, **while-do**, an infinite array, 0 and $s \leftarrow s + 1$.

Turing machine: A mathematical model due to Alan Turing (1936). Consists of an **infinite tape**, a **finite state control**, a **read-write head** and **a program**.

Circuit model: *Acyclic* logic circuits of n input bits consisting of *NAND*, *FANOUT* and *CROSSOVER*; whose description can be generated by a *Turing machine*.

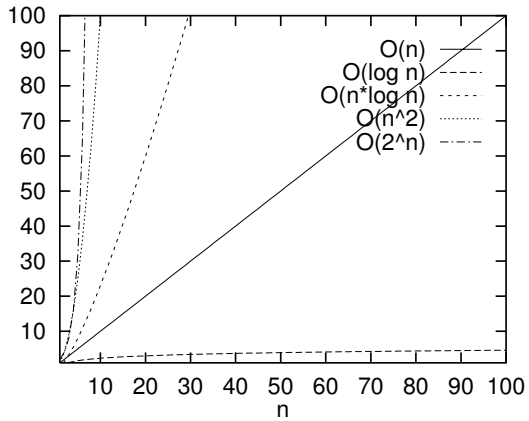
Church-Turing thesis

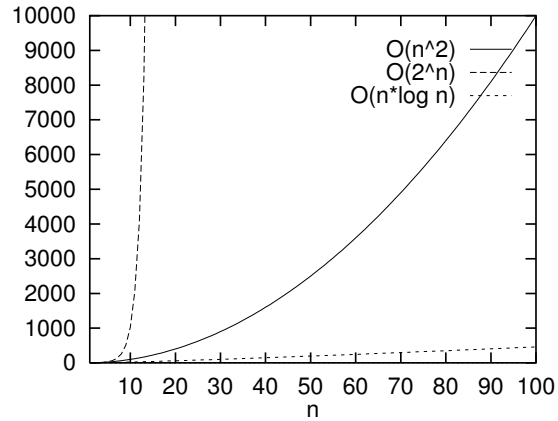
- All reasonable models of computations have turned out to be equivalent in terms of what they can compute.
- There can be a **Universal Turing machine** which can be used to simulate any Turing machine.
- The Universal Turing machine completely captures what it means to perform a computational task by algorithmic means.

The above has led to the assertion called the **Church-Turing thesis:**
If an algorithm can be performed on any piece of hardware (including a modern computer) then there is an equivalent algorithm for a Universal Turing machine which performs the same task.

What about efficiency?

- Roughly speaking, an efficient algorithm is one which runs in time **polynomial** in the size of the input.
- In contrast, an inefficient algorithm takes super-polynomial (typically exponential) time.





- Strengthened version of **Church-Turing thesis:** Any algorithmic process can be simulated **efficiently** using a Turing machine.

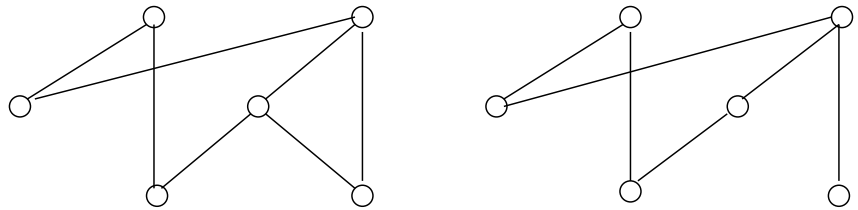
Big O

$f(n)$ is $O(g(n))$ if *there exist* $c > 0$, $n_0 \geq 0$ such that $f(n) \leq cg(n)$ whenever $n \geq n_0$.

Decision problems and complexity classes

Decision problems:

- Given a composite integer m and $l < m$, does m have a non-trivial factor less than l?
- Does a given graph have a Hamiltonian cycle?



Complexity classes:

- **P** is the class of decision problems that a UTM can solve in polynomial time.
- **NP** is the class of decision problems whose solutions a UTM can verify in polynomial time.

Does "coin toss" help?

- Consider a function $f : \{0, \ldots, 2^n - 1\} \rightarrow \{0, 1\}$.
- Suppose we are given that $f(x)$ is either *constant* (0 or 1 for all values of x) or *balanced* (0 for exactly half for all possible x and 1 for the other half).
- Our problem is to decide what type f is?
- Clearly, any deterministic algorithm will take at least $2^{n-1} + 1$ queries in the worst case.
- Alternatively, we can choose k (fixed) values of x *uniformly at random*. If $f(x)$ is different for any two conclude *balanced*, else conclude *constant*. In the latter case there is a non-zero probability of error, equal to 2^{-k}.
- The probability bound is arbitrary. *Chernoff bound* can be used to amplify the probability to near 0 with only a few (logarithmic) repetitions. [Suppose $X_1, X_2, \ldots X_n$ are independent and identically distributed random variables, each taking the value 1 with probability $1/2 + \epsilon$ and the value 0 with probability $1/2 - \epsilon$. Then

$$p\left(\sum_{i=1}^{n} X_i \leq n/2\right) \leq e^{-2\epsilon^2 n}$$

]

Randomized algorithms

- Solovay and Strassen showed, in mid 1970's, that a *randomized algorithm* could determine whether a number n is a prime (with an arbitrarily low probability 2^{-k}) or a composite (with certainty) in $O(k \log^3 n)$ time.
- *No efficient deterministic algorithm was known for the problem till Manindra Agarwal et. al. in 2003.*
- Strengthened version of **Church-Turing thesis:** Any algorithmic process can be simulated **efficiently** using a *probabilistic* Turing machine.
- **BPP** is the class of problems that can solved *efficiently* using a probabilistic TM.

What is (not) known about complexity?

- Some other complexity classes: **L**, **PSPACE**, **EXP**.
- It is known that $\mathbf{L} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}$.

  Is $\mathbf{P} = \mathbf{NP}$?
- It is also known that $\mathbf{P} \subset \mathbf{EXP}$ and $\mathbf{L} \subset \mathbf{PSPACE}$. Hence at least one of the inclusions above must be strict. Which one?
- Also, clearly, $\mathbf{P} \subseteq \mathbf{BPP}$
- If an **NP**-Complete problem can be solved in time t, then all problems in **NP** can be solved in time $poly(t)$.

Where does quantum computing fit?

- Peter Shor (1994) gave an $O(n^3)$ quantum algorithm for *factoring* an n bit number. The best known classical algorithm for the problem is *number field sieve* which works in $exp(O(n^{1/3} \log^{2/3} n))$.
- Lov Grover (1995) gave an $O(\sqrt{n})$ quantum algorithm for *search* in an *unstructured search space* of size n.
- If an $O(\log n)$ algorithm could be found for search it would have established that **NPC** problems can be solved efficiently on quantum computers.
  Not to be - Grover's algorithm has been proved to be optimal.
- It is known that $\mathbf{P} \subseteq \mathbf{BQP} \subseteq \mathbf{PSPACE}$.
- Is $\mathbf{P} \subset \mathbf{BQP}$?.