Computational Complexity

Lecture 0: Getting started

Ronald de Haan
me@ronalddehaan.eu

University of Amsterdam

2024

What is Computational Complexity?

- The study of what you can compute with limited resources
  - E.g.: time, memory space, random bits
    but also: nondeterminism, oracles

- *Computability theory* studies what can be computed **in principle**
- *Computational complexity theory* studies what can be computed **realistically**

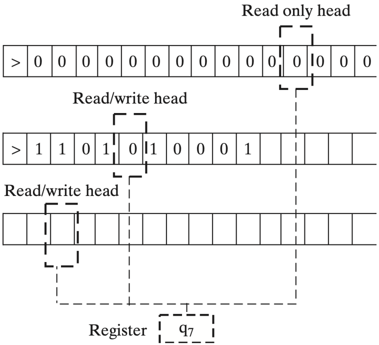
What is Computational Complexity? (ct'd)

- Main methodology: distinguish different degrees of difficulty (complexity classes)
  - There is an entire 'zoo' of complexity classes:
    `https://www.complexityzoo.net/`
    (currently listing 546 classes)
- One central question: the **P versus NP problem**
  (one of the $1M *Millennium Prize Problems*)

Definition (Turing machines; TMs)

A *Turing machine* $\mathbb{M}$ is a tuple (Γ, Q, δ), where:

- Γ is the *alphabet*: a finite set of symbols, including 0, 1, $\square$ (the blank symbol), and $\triangleright$ (the start symbol)
- Q is a finite set of *states*, including a designated start state q_{start} and a designated halting state q_{halt}
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathsf{L}, \mathsf{R}, \mathsf{S}\}^k$ is a *transition function*, for some $k \geq 2$ (the number of tapes of the machine)



Model of computation (ct'd)

Definition (TM computing a function)

A TM $\mathbb{M}$ computes the following (partial) function f, where for each $x \in \Sigma^*$:

- $f(x) = y$ if $\mathbb{M}$ halts on input x with output y,
- $f(x) =$ undefined if $\mathbb{M}$ does not halt on input x

Definition (running time)

Let $\mathbb{M}$ be a TM and $g : \mathbb{N} \to \mathbb{N}$ be a function. Then $\mathbb{M}$ *runs in time* $g(n)$ if for each input $x \in \Sigma^n$ of length n, the machine $\mathbb{M}$ halts after (at most) $g(n)$ steps.

- **Note:** we will switch (often implicitly) between
  the conceptual level ("algorithms")
  and the fully formal level ("Turing machines")

- Typically, we are interested in how (roughly) the running time scales, not in all the details
- We use what is called asymptotic analysis

Definition (Big O)

Let $f, g : \mathbb{N} \to \mathbb{N}$. We say that f is $O(g)$ if there exists a constant $c \in \mathbb{N}$ and an $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

- **Note:** in addition to "f is $O(g)$", the following are also used: "$f = O(g)$", "$f \in O(g)$", "$f(n)$ is $O(g(n))$", etc.

For example,
$4n^2 + 3n + 10$ is $O(n^2)$

Take $c = 8$
and $n_0 = 4$

Exercise session 1

- Big-O and little-o notation
- Describing algorithms at different abstraction levels