# Computational complexity

ian.mcloughlin@gmit.ie

***Abstract*—Notes on computational complexity.**

## I. Languages

Start with any set, call it an alphabet and denote it by $A$. For example:

$$A = \{0, 1\}$$

We define the strings of length $i$ over $A$ recursively:

$$A^0 = \{\epsilon\}$$
$$A^1 = A$$
$$A^i = \{as \mid a \in A, s \in A^{i-1}\}$$

The Kleene star $A^*$ of the alphabet $A$ is then union of all $A^i$:

$$\bigcup_{i \in \mathbb{N}_0} A^i$$

Here $\mathbb{N}_0$ is the set of natural numbers including zero:

$$A^* = \mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$$

A language $L$ over an alphabet $A$ is a subset of $A^*$:

$$L \subseteq A^*$$

## II. Turing machines

Turing machines encapsulate the concept of a computer. They are simple machines, consisting of a single read/write head and an infinite tape of cells, all but a finite number of which are blank. The head is over a single cell of the tape and is in any one of a finite number of states at a given point in time. Turing machines perform one small step at a time, which involves reading the symbol in the current cell, overwriting it with a symbol, moving to the cell to the left or right, and changing state.

We define a (deterministic) Turing machine $M$ by a 7-tuple:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_f)$$

where: $Q$ is a set of states; $\Sigma$ is the input alphabet, not containing the blank symbol $\sqcup$; $\Gamma$ is the tape alphabet, a superset of $\Sigma$ containing $\sqcup$; $\delta$ is a map:

$$\delta : (Q \setminus \{q_a, q_f\}) \times \Gamma \to \Gamma \times \{L, R\} \times Q;$$

$q_0$ is the initial states; $q_a$ is the accept state, which is a terminal state; $q_f$ is the reject state, also a terminal state.

Before the Turing machines begins its operation, a finite number of consecutive non-blank cells contain symbols and the head is over the left-most of these. These non-empty cells form the Turing machine's input and can be viewed as a string.

When the Turing machine operates on an input, there are three possibilities. The first two are similar: the machine can accept the input by ending in the accept state or the machine can reject the input by ending in the reject state. In these cases the machine stops upon entering these states, which is usually called halting. The third possibility is that the machine might never stop operating.

The subset of $\Sigma^*$ (the Kleene star of the input alphabet) containing all elements that are accepted by the Turing machine $M$ is called the language of the Turing machine $L(M)$:

$$L(M) \subseteq \Sigma^*$$

A Turing machine that halts on all inputs is called a decider and is said to decide its language.

## III. Decision problems

A decider makes a binary choice for a given input: accept or reject. The Turing machine $M$ in this case performs a map:

$$f : \Sigma^* \to \{q_a, q_f\}$$

Sometimes this map is called an indicator function, as it indicates which elements of $\Sigma^*$ are in $M$. In general, a map from a set to any set with two elements is called a decision problem. So, a decider performs a decision problem.

## IV. Complexity

A decider takes a finite number of steps before halting on a given input. We are typically interested in knowing the maximum number of steps $f(n)$ the decider takes on all inputs of length $n$.

We can often express $f(n)$ using a simple formula, such as $f(n) = 2n + 2$. We are interested in knowing how $f(n)$ grows in relation to $n$. For this, we use big-O notation:

**Definition.** *A function $f(n)$ is said to be $O(g(n))$ if there exist two positive integers $c$ and $n_0$ such that $cg(n) \geq f(n)$ for all $n \geq n_0$.*

In the previous example, $f(n) = 2n+2$ is $O(n)$ which can be seen by setting $g(n) = n$, $c = 3$ and $n_0 = 2$ as per Fig. 1.

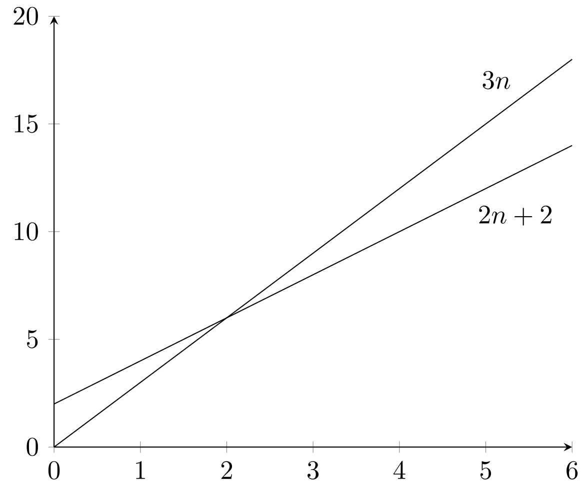

Fig. 1: $2n+2$ is $O(n)$