The Role of Algorithms in Computing

1 Algorithms

An *algorithm* is a computational procedure that solves a well-specified *computational problem* by taking some *input* (or set of inputs) and producing some *output* (or set of outputs) in finite time. An example of a well-specified problem is the *sorting problem*:
Input: A sequence of n numbers $\langle a_1, a_2, ..., a_n \rangle$
Output: A permutation of the input sequence $\langle a'_1, a'_2, ..., a'_n \rangle$ where $a'_1 \leq a'_2 \leq ... \leq a'_n$.

An *instance* of a problem consists of an input needed to compute a solution that satisfies the given constraints of the problem. For example, $\langle 41, 32, 55 \rangle$ is an instance of the sorting problem.

An algorithm for a computational problem is *correct* if it produces a correct solution in finite time (that is, it *halts*) for all instances of the problem. Thus, an incorrect algorithm would be one that only works for some instances and doesn't work for others, or one for which a solution can only be produced in infinite time for at least one instance. We can say that a correct algorithm *solves* the computational problem.

A *data structure* is a way to store and organise data in order to facilitate access and modifications.

Algorithms that receive their inputs over time, rather than it all being present at the start, are called *online algorithms*.

Differences in algorithmic efficiency can have as big an impact on total system performance as fast hardware, for example. In light of this, it is worth thinking of algorithms as a *technology*. To illustrate this, consider two sorting algorithms: *insertion sort* and *merge sort*. These have efficiencies $c_1 n^2$ and $c_2 n \log n$, respectively. Even if $c_1 \ll c_2$, eventually the form of the dependence on n will have a greater impact.