1 Algorithms

Algorithm is a solution to a *computational problem* that specifies (in general terms) a desired *input-output* relationship. It describes a specific step-by-step procedure for solving the problem in a **finite** amount of time. The input is an *instance* of the computational problem, and the output is the corresponding solution.

2 Partial vs total correctness

It is conventional to distinguish between *partial* and *total* correctness.
An algorithm that produces the desired output for all inputs **when the algorithm terminates** is said to be *partially correct*.
An algorithm that produces the desired output for all inputs **and is guaranteed to terminate** is said to be *totally correct*.
Our definition of an algorithm, implicitly assumes total correctness.

2.1 example

An algorithm that "sorts a sequence of n numbers":

- **input**: a sequence of n numbers $(a_1, a_2 ... a_n)$
- **output** a permutation of the input sequence $(a'_1, a'_2 ... a'_n)$, such that $(a'_1 \leqslant a'_2 \leqslant ... \leqslant a'_n)$

3 Correctness

An algorithm for a computational problem is *correct* if, for every legal input instance, the equired output is produced.

3.1 Correctness vs testing

Correctness is not the same as testing. Testing is applied to an *implementation of an algorithm*- we need something runnable in order to perform the tests. Each test shows the implementation is correct for a **particular** input and output. Many algorithms have an **infinite** number of possible inputs, but we can only run a finite number of tests. Testing can increase conidence, but we **cannot** show an algorithm is correct by testing.
Correctness is (usually) a **property** of an algorithm rather than an implementation of an algorithm

3.2 Approaches to correctness

- Static analysis
- proof-based
- model-based (model checking)
- program synthesis (program is guaranteed correct by construction)

4 Assertions

An *assertion* is a statement about what holds at a particular state in a computation. Since we reason with them, they must be **precise**.
Can also be though as a **partial** description of a computational state.

4.1 Why use assert statements

- Can provide useful documentation (so long as they are not over-used)
- Can be placed on every code branch (it can be difficult to devise unit tests that force execution down every code branch)
- With well-chosen conditions you can get a useful increase in assurance at low computational cost

Reference section

placeholder
placeholder