Algorithms

An algorithm is a method that can be applied in order to accomplish a particular task. Novices to computing often regard the programming language as the most important focus in learning to *program*. This is far from the case; it would be like saying that the use of words and grammar are the most important aspect of a novel, rather than the characters or the plot. An experienced programmer must focus on *design*, and design is about choice and one choice is the selection and implementation of appropriate algorithms.

An algorithm normally consists of a set of instructions. We are used to following sets of instructions in everyday life. Typical examples are the set of instructions that come with a piece of flat-pack furniture, or a recipe in cooking. Let us consider an algorithm for making a cup of tea:

Algorithm 1

fill kettle

boil kettle

put tea in teapot

fill teapot with boiling water

pour tea into cup

add milk

Algorithm 2

fill kettle

boil kettle

put teabag in cup

fill cup with boiling water

remove teabag

add milk

We note that both algorithms accomplish the same task. In situations in which we have a choice over a set of different algorithms it would help to consider their other properties. For example we may point out that the second algorithm may be more efficient, or that the first algorithm may easily provide a second cup of tea. We could consider that one algorithm is “better” than another. However, it is more appropriate to think in terms of the relative properties of algorithms, rather than make general value

judgements about them. Different algorithms that perform the same task may be fit for purpose in different situations. Often the more efficient algorithms are more complex and more difficult to program.

Algorithms are developed so that they may be implemented on computer. For illustration, a simple example of an algorithm, to find the average of a list of 100 numbers, is given:

```
sum=0;
for (i=0; i<100; i++)
{
sum=sum+x[i];
}
average=sum/100;
```

The algorithm is in a c-like or java-like coding. We note that x is an array of numbers, which is a form of data structure[1], and algorithms often have are written to work with data structures, for example to find information from them or to modify them.

We may think that an algorithm like the one above computes so fast on a modern computer that it is not worthwhile to worry about the computer processing time of algorithms. It is true that such an algorithm would compute fast and that whether a computer program takes 1 microsecond or 10 microseconds is not going to be a cause for concern. However, in the real world algorithms often have to work on large data structures and often algorithms are much more complex than the example. In reality we need techniques for evaluating algorithms.

The properties of algorithms can be determined experimentally, for example by implementing it on a computer and for example measuring the time taken for the algorithm to complete. Alternatively we inspect and analyse the algorithm to determine some measure of the time it takes to run.

Finally, some of the steps in an algorithm can often be carried out simultaneously, or *in parallel*. Returning to the example of making a cup of tea, in Algorithm 1 the tea could be put in the teapot while we are waiting for the kettle to boil. There is no advantage (in terms of efficiency) in running tasks in parallel if a computer only has one processor. However, if a computer has several processors, then the *parallelization* of code will often reduce the processing time, especially if algorithms are written to take advantage of them.

[1] Data Structures