For simple problems, it is often easy to see that a particular algorithm will always work, i.e. that it satisfies its specification. However, for more complicated specifications and/or algorithms, the fact that an algorithm satisfies its specification may not be obvious at all.

In this case, we need to spend some effort verifying whether the algorithm is indeed correct. In general, testing on a few particular inputs can be enough to show that the algorithm is incorrect.

However, since the number of different potential inputs for most algorithms is infinite in theory, and huge in practice, more than just testing on particular cases is needed to be sure that the algorithm satisfies its specification. We need correctness proofs.

Although we will discuss proofs in these notes, and useful relevant ideas like invariants, we will usually only do so in a rather informal manner (though, of course, we will attempt to be rigorous. The reason is that we want to concentrate on the data structures and algorithms. Formal verification techniques are complex and will normally be left till after the basic ideas of these notes have been studied.

Finally, the efficiency or performance of an algorithm relates to the resources required by it, such as how quickly it will run, or how much computer memory it will use. This will 6 usually depend on the problem instance size, the choice of data representation, and the details of the algorithm. Indeed, this is what normally drives the development of new data structures and algorithms.

1.3 Data structures, abstract data types, design patterns

The term data structure is used to denote a particular way of organizing data for particular types of operation. These notes look at numerous data structures ranging from familiar arrays and lists to more complex structures such as trees, heaps and graphs.

Often, we want to talk about data structures without having to worry about all the implementational details associated with particular programming languages, or how the data is stored in computer memory. We can do this by formulating abstract mathematical models of particular classes of data structures or data types which have common features. These are called abstract data types and are defined only by the operations that may be performed on them. Typically, we specify how they are built out of more primitive data types (e.g., integers or strings), how to extract that data from them, and some basic checks to control the flow of processing in algorithms. Abstract data types are defined only by the operations that may be