Chapter 1

Introduction

These lecture notes cover the key ideas involved in designing *algorithms*. We shall see how they depend on the design of suitable *data structures*, and how some structures and algorithms are more *efficient* than others for the same task. We will concentrate on a few basic tasks, such as storing, sorting and searching data, that underlie much of computer science, but the techniques discussed will be applicable much more generally.

We will start by studying some key data structures, such as arrays, lists, queues, stacks and trees, and then move on to explore their use in a range of different searching and sorting algorithms. This leads on to the consideration of approaches for more efficient storage of data in hash tables. Finally, we will look at graph based representations and cover the kinds of algorithms needed to work efficiently with them. Throughout, we will investigate the computational efficiency of the algorithms we develop, and gain intuitions about the pros and cons of the various potential approaches for each task.

We will not restrict ourselves to implementing the various data structures and algorithms in particular computer programming languages (e.g., *Java*, *C*, *OCaml*), but specify them in simple *pseudocode* that can easily be implemented in any appropriate language.

1.1 Algorithms as opposed to programs

An *algorithm* for a particular task can be defined as "a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time". As such, an *algorithm* must be precise enough to be understood by *human beings*. However, in order to be *executed* by a *computer*, we will generally need a *program* that is written in a rigorous formal language; and since computers are quite inflexible compared to the human mind, programs usually need to contain more details than algorithms. Here we shall ignore most of those programming details and concentrate on the design of algorithms rather than programs.

The task of *implementing* the discussed algorithms as computer programs is important, of course, but these notes will concentrate on the theoretical aspects and leave the practical programming aspects to be studied elsewhere. Having said that, we will often find it useful to write down segments of actual programs in order to clarify and test certain theoretical aspects of algorithms and their data structures. It is also worth bearing in mind the distinction between different programming paradigms: *Imperative Programming* describes computation in terms of instructions that change the program/data state, whereas *Declarative Programming*