1 An Overview

In this course, we develop algorithms to solve a variety of problems and analyse the complexity of these algorithms, usually by bounding their running time.

You performed similar analyses of algorithms for sorting, selection, graph searching, and other problems in 308-251. Many of the tools developed in that course will be used in this course. (eg. big O notation, various data structures, and the sorting, selection and graph searching algorithms themselves).

We use the order of growth of the running time as a measure of the speed of an algorithm. We can compare the performance of algorithms using this measure. Thus we say one algorithm is *faster* than another if its time complexity has smaller growth rate.

We would also like to have a way of comparing the difficulty of the problems we consider. A natural measure would be the order of growth of the running time of the fastest algorithm for the problem. Unfortunately, it is very difficult to get a handle on this measure. Indeed, it is hard to prove any lower bound on the running time of the fastest algorithm for a problem, as this necessitates dealing with all of the (possibly infinitely many) algorithms which solve it.

In this course, we introduce a technique for comparing the difficulty of problems directly which does not require us to compute the measure discussed in the last paragraph. This is an important new direction which gives this course a flavour very different from earlier courses.

Another new topic is verifying that the output of an algorithm is correct. Obviously it is trivial to check whether the output of a program for sorting integers is correct: we just scan the output and make sure each number is less than or equal to its successor. But how could we check the correctness of, say, the output of a program that computes a shortest path between two nodes in a network, or a minimum weight spanning tree of a weighted graph? For more complicated problems we will see that the addition of some extra data, known as a *certificate*, can facilitate the checking process. We will see that the existence, or apparent non-existence, of certificates is intimately related to whether or not a problem can be solved efficiently.

This course also differs markedly from 308-251 in that we consider algorithms for hard problems. We no longer want to just bake cakes we also want to find needles in haystacks. For this reason, we need to significantly alter how we think about algorithms. We no longer think of streamlining the transformation from a well-defined set of inputs to a well defined set of outputs (" I can speed things up by beating the eggs while the chocolate is melting"). Instead, we think of our algorithm as a search procedure ("Is this the needle?-no a piece of hay, Is this the needle? -no a piece of hay, Is this the needle?..."). We hasten to assure the reader that all algorithms can be viewed from this new perspective. For each input to an algorithm there are a number of acceptable output strings; the algorithm has to find one such string from amongst all possible strings it might output.

Of course, the input-output streamlining paradigm is also valid for every algorithm. Furthermore, it is natural when discussing certain types of approaches for designing algorithms, e.g. divide and conquer. However, the search paradigm is more appropriate when discussing most of the new algorithm design techniques we discuss and for discussing our approach to comparing problem difficulty. Indeed, once one formally defines a problem, one is led naturally to this point of view.

Although we discuss algorithms for hard problems, we want to stay away from those which are too hard. For example, in 308-330 you will see/have seen that there are problems which are undecidable, i.e. for which no algorithm exists at all. Certainly we would not want to work on such a problem. It turns out that the notion of a certificate is also a useful tool for deciding if it is reasonable to attempt to solve a given problem.