1 An Overview

In COMP252, you developed algorithms to solve a variety of problems and analyse the complexity of these algorithms, usually by bounding their (worst case) running time. You performed such analyses of algorithms for sorting, selection, graph searching, and other problems in COMP252. 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 say an algorithm is *efficient* if its running time is a sufficiently slowly growing function of the input size.

To a large extent, in previous courses the focus has been on developing efficient algorithms for problems for which such algorithms exists. In this course, we consider how to show that a given problem is hard in that there is unlkely to be an efficient algorithm to solve it. We then discuss how to handle such problems.

One techniquae for showing that problems are hard is also a technique for solving problems which is central to computer science: programming or reducing one problem to another. If we have a fast algorithm to reduce problem π to problem π' and a fast algorithm to solve problem π' then we have a fast algorithm to solve problem π. So a fast reduction allows us to use a fast algorithm for one problem to efficiently solve a different problem. Note that if we have such a fast reduction and there is no fast algorithm for problem π then there is no fast algorithm for problem π'. Thus fast reductions also allow us to use a proof that some problem is hard to prove another problem is hard.

Another new topic we intoduce is the notion of *certificates* which allow us to verify quickly that the output of an algorithm is correct. Thus, for example,it is much easier to verify that an algorithm has correctly determined that n is prime, if it provides us with two integers f_1 and f_2 both less than n whose product is n. For then, we need only multiply f_1 and f_2 together and check that the product is n.

It is natural to ask for certificates from algorithms for solving hard problems so that we can efficiently verify that they have provided a correct solution . As we shall see, the existence, or apparent non-existence, of certificates appears to be intimately related to whether or not a problem can be solved efficiently.

In the first part of the course, we introduce the notion of reductions and certificates via a discussion of Mathematical Programming and more specifically Linear and Integer Linear Programming.

In the second part of the course, we discuss the use of reductions in proving that problems are hard. This is the theory of NP-completeness.

In the third part of the course we discuss a number of techniques for handling

hard problems algorithmically. For example we may settle for approximate not exact solutions, restrict our attention to easy instances of the problem, or settle for a randomized algorithm which we expect to work quickly.