Introduction to algorithmic complexity

Introduction

We want to be able to compare the speed and space requirements of different algorithms. For example, we will want to ask questions like these: Which of two sorting algorithms runs the fastest on large arrays? Which uses more space? How can we tell that a particular sorting algorithm will be infeasible (takes far too much time), without actually running it, when attempting to sort arrays of size 10,000?

In analyzing different algorithms for speed and space, we should not have to depend on the computer on which it is running, the operating system being used, and so on. Our analysis of the algorithms should be independent of such considerations.

Also, we are interested in the speed and space requirements when the data gets *large*. You can use *any* algorithm to sort an array of size 10 or 50 —you probably won't notice any difference. But when sorting an array of size 1,000 or 10,000 or 100,000, you will see *huge* differences in the time different sorting algorithms take. Thus, we will talk about *asymptotic complexity*, the speed and space requirements as the size of the data gets large, even approaches infinity.

We will also be interested in *classes* of algorithms, depending on their time complexity. Suppose one algorithm takes *quadratic time* to sort an array of size `n`, taking, say. about $100n^2$ steps, while a second algorithm takes *cubic* time, for example, $20n^3$ steps. We would rather use the quadratic-time algorithm because $100n^2$ is a *lot* smaller than $20n^3$ when `n` is large.

Finally, we are most interested in (1) the *expected* or average time of an algorithm and (2) the *worst-case* time of an algorithm. One also hears talk of the *best-case* time, but it's of little importance compared to the other two.

You first step in the study of complexity will be to learn what a "basic step" is and to get some practice in counting the basic steps in execution of an algorithm.