NOTES FOR SEPTEMBER 26, 2002

This class is the study of the analysis (and to a lesser extent, the design) of algorithms. The analysis of an algorithm is a proof of that algorithm's completeness, of it's correctness, and of a set of properties that govern the algorithm's running time.

Algorithms can be divided up, roughly, into four classes. These are

Obvious algorithms: An algorithm exists and is obvious. It is not necessarily efficient. In many cases, however, it is the best that you can do—life does not always cooperate. The obvious algorithm is generally the one that can be derived off of the problem statement directly. For example, we might use Bozosort to sort a list; bozosort is to try every combination of entries until you find one that is in order.

Methodical algorithms: An algorithm exists and is not obvious, but isn't too obscure either. These algorithms are generally derived by taking a problem and applying a general method of algorithm design. This often gives a faster set of steps. An example of a methodical algorithm might be a naive implementation of quicksort—quicksort uses the notion of divide-and-conquer, which is a standard algorithm design technique.

Clever algorithms: Algorithms that involve manipulating or tweaking some parameter to exploit part of the problem are clever. Cleverness generally involves using intuition and trickery. It is difficult to teach trickery, but by imitating it we may become more likely to be clever. This course will actually concentrate on the transition from Methodical algorithms to Clever algorithms. An example of a clever algorithm might be some of the boundary conditions used in quicksort, combined with other sorting algorithms to provide an overall smaller upper bound.

Miraculous algorithms: Sometimes algorithms appear out of nowhere and give surprisingly good solutions to a problem that has been studied for years. These are generally referred to as miraculous because nobody is certain how they spring into being. An example of this might be "fusion tree sort", which I know nothing about but that Russell mentioned in class.

An interesting point was brought up which is that people often believe that as computers get faster and cheaper, and as memory gets larger we do not actually need to worry about algorithmic efficiency to the extent that we needed to worry in the days of Univac. To some extent, this is true; certain problem sizes are bounded and do not grow with the size of a computer (eg, the genome isn't getting any larger, so it is theoretically possible that we can handle many genomic questions with suitably large computers). However, there are at least two other axes involved in this argument, rendering it somewhat specious. First, one must consider the family of problems that have no efficient solution (i.e., exponential time algorithms) so tractable problem size remains remarkably small; second, one must consider the fact that our computational appetite grows with our capabilities. If we have super-fast graphics, we want better and more realistic textures. If

we have million-processor arrays, we want more detail in our scientific simulations and in our artificial intelligence programs. Thus, as the computer technology improves, we need to think more about efficiency, not less, to keep the improvements in computing commensurate with improvements in hardware.

The rubric that we should follow in this class is fairly straightforward, but I will state it so that I remember to use it when I do a homework set.

- Is the algorithm clearly described in English and Mathematical (i.e., non-code) terms?
- Does the algorithm actually solve the stated problem? This involves a correctness proof, which may take one of two forms: a short English description of why the algorithm works, or a moderately long mathematical argument with the intent to force a reader to agree that the algorithm works.
- Is the algorithm efficient? Efficiency is generally considered to be polynomial time, but for algorithms with obvious polynomial solutions, it may be linear or sub-linear time. Again, this will require some form of proof, be it a short and well-motivated English description of the running efficiency (eg, outer loop executes n times, while the inner loop executes m times, leading to $\mathcal{O}(nm)$ efficiency) or an involved mathematical argument.
- Is the algorithm nearly optimal, according to the state of the art? This may seem unfair, but algorithms won't get used if they're not the fastest or simplest available. This will generally require a literature search, followed by a brief citation of related algorithms and how they compare.