Probabilistic Analysis and Randomized Algorithms

ECE 1762 Algorithms and Data Structures
Spring Semester, 2004 — U Toronto

1 Probabilistic Analysis and Randomized Algorithms

There are several standard ways in which probability theory can arise in the analysis of algorithms:

Randomized algorithms: A randomized algorithm is one in which the algorithm itself makes random choices, and hence the time/space used by the algorithm is a random variable that depends on these random selections. The analysis of RANDOMIZED QUICKSORT given in CLR is a classic example. There are various flavors of randomized algorithms. If we assume that we deal with algorithms that solve decision problems only (*i.e.,* the output of the algorithm is an answer either "yes" or "no" for a given problem) then we have the following two types of randomized algorithms:

Las Vegas. A Las Vegas algorithm is one which does not *lie*. In other words, the answer it gives is always correct. The distinguishing characteristic of Las Vegas algorithms is that sometimes the sequence of random decisions that it makes may lead to a stage where it is impossible to find a solution. In that sense, for a fixed time bound, the algorithm might return the answer answer "I don't know".

Monte Carlo. Monte Carlo algorithms are popular in simulations of physical systems, for example. For decision problems, Monte Carlo algorithms always return with a solution but they may "lie" with a small probability, *i.e.* they may answer "yes" when the answer is actually "no". No warning is usually given when the algorithm makes a mistake (lies). A Monte Carlo algorithm is said to be *p–correct* if it returns a correct solution with probability not less than p.

Various types of results are useful. Often, an *expected time* result is given. (This is often the easist to obtain.) However, you will generally also want to know something about the deviation of the actual time from the expected time. Instead of computing standard deviations — which are often hard in this context — we generally look for *high confidence* results: "algorithm A uses time $O(T(n))$ with probability $\geq 1 - 1/n^c$."

Probabilistic analysis. Sometimes a deterministic algorithm is given a probabilistic analysis. In this case, we generally put a probability distribution on the inputs (dependent upon the expected frequency of inputs, although often the inputs are just given the uniform distribution). The expected time is then just the average time used, when time is averaged over all inputs (of a fixed size).

CLR analyzes the deterministic algorithms for hash tables, bucket sort and *deterministic* Quicksort in this manner.