Lecture #22: Application — The Analysis of Algorithms
Key Concepts

Average-Case Analysis of Deterministic Algorithms

Consider a deterministic algorithm that solves a given computational problem. Let n be a non-negative integer.

- A ***sample space*** consists of instances of this problem of size n — or of *sets* of these inputs (such that the inputs in any one of these sets would cause the algorithm to behave in the same way).
- A ***probability distribution*** is formed, to model assumptions about the *likelihood* of the (types of) inputs included in the sample space.
- The number of steps used by the algorithm, on each (type of) input in the sample space, is used to define a ***random variable***.
- The ***expected running time*** of the algorithm, for input size n, is the ***expected value*** of this random variable with respect to this probability distribution.

Examples from CPSC 331: Analysis of algorithms for operations in hash tables; analysis of the deterministic `Quicksort` algorithm.

Randomized Algorithms

Randomized algorithms use random number generators during the execution — so that neither the output generated, not the number of steps used to generate it, are necessarily *fixed*, even when the algorithm's input is.

Analysis for a Fixed Input

Consider the execution(s) of a randomized algorithm, for a given computational problem, on a given instance of this problem.

- A ***sample space*** consists of sequences values that are produced during the algorithm's execution, using the random number generator.
- A ***probability distribution*** is formed, to model assumption about the random number generator that is being used. It is usually assumed that this is a "perfect" source of random values: When a value from a finite, nonempty set is requested then each element of the set is produced with the same probability, and this is independent of all choices that have been made before this (or anything else).
- The number steps that are used by the algorithm, for this input and for this sequence of random variables, is used to define a ***random variable***.
- The ***expected running time***, for this input, is the expected value of this random variable, with respect to this probability distribution.

Measuring Running Time as a Function of Input Size

- Suppose that the *expected running time* of a randomized algorithm, for a given instance of the computational problem, is as defined above. Then the ***worst-case running time*** of this algorithm (with respect to this probability distribution) is a function $T : \mathbb{N} \to \mathbb{R}$.

  If there are finitely many instances of the problems with size n — or only finitely many values that that the "expected running time" can assume, for instances of the problem with size n, then the worst-case expected running time, $T(n)$, is the *maximum* of the expected running times of the algorithm for all inputs of size n.[1]

Example from CPSC 331: Analysis of a randomized `Quicksort` algorithm.

Randomized Algorithms That Can Fail

A ***decision problem*** is a computational problem that answers a "Yes-or-No" question, so that its output is always either `true` (corresponding to the answer "Yes") or `false` (corresponding to the answer "No").

[1] This to be defined to be the *supremum* of this set of values, when infinitely many inputs must be considered.

- A ***Las Vegas*** algorithm for a decision problem is a randomized algorithm that can never give a wrong answer. Las Vegas algorithms are of interest when their worst-case expected running times are at most polynomial in the size of the algorithm's input.
- A ***Monte Carlo*** algorithm for a decision problem is a randomized algorithm that can, sometimes, fail — but that does so with small probability:
  - The algorithm only returns true, when executed on a given input, if this is the correct answer for that input. That is, the algorithm always returns the corrected answer when it is executed on an input for which correct answer is "No".
  - If the algorithm is executed on an input for which the correct answer is "Yes" (so that the algorithm *should* return `true`) then the probability that the algorithm *does not* return `true` is at most $\frac{1}{2}$.
  - Other kinds of algorithms that can fail are studied in the literature too. This will be considered, further, in the tutorial exercise for this topic.

What Really Happens

Randomized algorithms are almost always analyzed using the assumption that "perfect" sources of random values are available — that is, when a value is to be chosen from a finite non-empty set then every value is returned with the same probability, and this is independent of all previous uses of the random number generator.

- Current programming environments currently provide ***pseudorandom number generators*** instead. These are *deterministic* processes that generate sequences of values, starting from an initial value (generally called the "seed").
- Thus the *assumption*, used to define probability distributions in order to analyze the performance of randomized algorithms, is ***false***.
- Nevertheless, experiments (involving monitoring the performance of randomized algorithms, as they are used in progress) generally given results that are consistent with the results that these flawed analyses indicate.
- A huge amount of additional information about this is available — and this is beyond the scope of this course.