1. Randomized Algorithms

Randomized algorithms

If we have a table of algorithms vs input instances, and we want to introduce randomness to the solution we are working for, which part do we introduce randomness?

We can define a distribution on either **Algorithms** or **Inputs**.

Distribution of Inputs

A way to randomize inputs can be understood about a random graph of n vertices, and an edge between every pair of coins has a probability of $1/2$ for occuring. Hence, to create a random graph we can just toss a coin for every possible edge and include it based on the result.

There are $\binom{n}{2}$ possible edges, implying there are a total of

$$2^{\binom{n}{2}}$$

possible graphs of n vertices (assuming a simple graph i.e no self-loops or multiedges).

Distribution of algorithms

Say we have some algorithm A, we run the algorithm as

$$A(I, R)$$

Where I is an input instance, and R is some randomness.

This algorithm is deterministic. This algorithm takes a "random" string or advice either as input at the start or at an intermediate step.

Hence, instead of defining a randomized algorithm as a non-deterministic algorithm, we instead define it as a deterministic algorithm with random advice as input.

We want to minimize the use of randomness. - This is done as it makes the job easier of derandomizing algorithms. One of the most famous cases of derandomization is primality testing, which was earlier a randomized algorithm, and was made deterministic (the AKS primality test).

For this course, we can assume that the randomness comes from some other source that we don't have to worry about.

Why Study Randomized Algorithms?

In a lot of instances, randomized algorithms give a better guarantee (in expectation). This will be covered and explained in more detail in the next few lectures.

Understanding in terms of Quicksort, where we claim that picking a random pivot is a better algorithm than QS, which we defined earlier in this lecture.

Example of Generating Randomness

Let's see the case of quicksort, where we instead pick a pivot at random. This is equivalent to picking a random index from 1 to n. We can represent n with $\log n$ bits, hence we can randomly pick an index by flipping a coin to determine the value of each bit.