Lecture 22: Analyzing Algorithms

In the previous lecture, we discussed proving *correctness* of an algorithm via techniques such as mathematical induction.

We now briefly discuss how to analyze an algorithm's *efficiency*. We already saw an example before (in the context of the Tower of Hanoi). There, we counted the number of steps (or moves) required to solve the Hanoi problem involving n disks, and discovered that this number equals $2^n - 1$.

The algorithm we discussed for solving the Tower of Hanoi is an example of a rather inefficient algorithm. The reason is because the exponential function 2^n grows very quickly as a function of n (for example, for $n = 250$, we already exceed the number of atoms in the known universe.)

On the other hand, consider *bubble sort*. Given an unsorted array of (say) integers $x_1, x_2, \ldots, x_n$, the following algorithm produces a sorted array:

for $i \in \{1, \ldots n-1\}$:

for $j \in \{1, \ldots, n-i\}$:

if $x_j > x_{j+1}$ then swap x_j, x_{j+1}

You have probably seen this algorithm before (and/or even coded it at some point). The question of efficiency now arises: how many steps does bubble sort take to execute in terms of n? The answer could be of importance in real-life applications: no matter how and in which software environment we implemented this algorithm, the time taken to run the algorithm would proportionally depend on some function of n.

Assuming that each $>$ comparison takes 1 step, observe that each inner loop takes $n - i$ steps; here, i is the outer loop index that starts with 1, then becomes 2, and increments by 1 each time until it reaches $n - 1$. Therefore, the total number of steps is given by:

$$(n-1) + (n-2) + \ldots + 2 + 1$$

which is nothing but the sum of the first $n - 1$ integers. By induction (or other means), we can prove that this equals:

$$\frac{(n-1)n}{2}.$$

We won't go too much further, but suffice to say that you'll do much more of such analyses of algorithms in 311.

Big-Oh

Let us conclude by thinking of the following question: among the two, which algorithm takes "more" time – the Hanoi algorithm from last class, or Bubble sort?

Of course, time taken is hard to measure since it depends on lots of factors – how fast the computing hardware is, what's the OS used, etc. But let's assume that everything is run in exactly the same environment.

Even then, these two algorithms are not directly comparable – one requires comparing numbers, while the other involves moving disks, but let us assume that either operation requires 1 unit of computation.

Hanoi requires $2^n - 1$ steps, while bubble sort requires $n(n-1)/2$ steps. Typically, we are not interested in leading constants, and instead wish to focus on the *growth* of either expression as a function of n, so we say that Hanoi takes $O(2^n)$ steps (or "big-oh 2^n steps"), while bubble-sort takes $O(n^2)$ steps. The former is an exponential function, which dominates the latter (which is a quadratic). Therefore, as a rule, the Hanoi algorithm is *much* slower. We will revisit big-Oh towards the end of the class.