Let's Start At the Very Beginning ...

Robin Dawes

September 9, 2021

That's a very good place to start.
When you read you begin with A - B - C
In Computing we start with com-plex-i-ty.

Com-plex-i-ty! Com-plex-i-ty!
The first big thing just happens to be
Com-plex-i-ty! Com-plex-i-ty!

Ancient History

When computers first came into general use, people everywhere started to write programs to solve problems. Many common problems - such as computing the square root of a number, or storing and retrieving telephone numbers - could be solved in many different ways. It was natural to try to identify the best way to solve these problems. But in order to choose a best solution, we need a measurable criterion that lets us compare solutions.

The basic criterion that we use today is based on time-requirements. If algorithms A and B both solve the same problem but B takes less time than A, then we prefer to use B.

by "solve" we mean "always give the correct answer to"

The difficulty with this is that if we want to compare the time requirements of two algorithms, we need to ensure a level playing field: the two algorithms need to be tested under identical conditions. The same hardware, the same OS, the same programming language, the same environment (including the same background processes being run by the OS), etc. etc. Research papers used to be published listing the execution time of new algorithms as evidence of improvement. But without a common frame of execution, these numbers were rarely meaningful.

or new implementations of old algorithms

Slightly Less Ancient History

TO MOVE AWAY from the ephemeral details of hardware and software and focus on the fundamental properties of the algorithms, two abstractions are applied:

1. Instead of measuring clock-time, we compute the number of operations executed by an algorithm as a function of the size of the input.
2. Instead of focusing on the exact number of operations, we look at the *rate of growth* of the number of operations.

The justification for this simplified method of comparing algorithms is this: if we can show that the number of operations for algorithm B grows more slowly than the number of operations for algorithm A, then there **must be** some value n_0 such that for all input of size $\geq n_0$, algorithm B will use fewer operations than algorithm A.

In other words: once the size of the input gets large enough, algorithm B will be faster than algorithm A.

I wrote some fairly detailed notes on computational complexity back at the beginning of 2020 (pre-pandemic!) Rather than rewrite them, I will just include them here.