COMP 285 (NC A&T, Spr '22) Lecture 0

Adapted From Virginia Williams' lecture notes. Additional credits: J. Su, W. Yang, Gregory Valiant, Mary Wootters, Aviad Rubinstein, Sami Alsheikh.

Introduction

1 Logistics

The course website is at https://comp285.ml. All course information is available on the website.

2 Why are you here?

Most of you are here because this class is required. But why is it required?

1. **Algorithms are fundamental to all areas of CS:** Algorithms are the backbone of computer science. Wherever computer science reaches, an algorithm is there, and the classical algorithmic design paradigms that we cover re-occur throughout all areas of CS.

   For example, COMP 350 (operating systems) leverages scheduling algorithms and efficient data structures, COMP 322 (internet systems) crucially uses shortest-path algorithms, COMP 365 (AI & Machine Learning) leverages fast geometric algorithms and similarity search, COMP 420 (applied network security) leverages fast number theoretic and algebraic algorithms. We'll discuss applications to all these subjects in this class.

   Algorithms and the computational perspective (the "computational lens") has also been fruitfully applied to other areas, such as physics (e.g. quantum computing), economics (e.g. algorithmic game theory), and biology (e.g. for studying evolutions, as a surprisingly efficient algorithm that searches the space of genotypes).

2. **Algorithms are useful:** Algorithms are useful: Much of the progress that has occurred in tech/industry is due to the dual developments of improved hardware (a la "Moore's Law"—a prediction made in 1965 by the co-founder of Intel that the density of transistors on integrated circuits would double every year or two), and improved algorithms. In fact, the faster computers get, the bigger the discrepancy is between what can be accomplished with fast algorithms vs what can be accomplished with slow algorithms .... Industry needs to continue developing new algorithms for the problems of tomorrow, and you can help contribute.

3. **Algorithms are fun!** The design and analysis of algorithms requires a combination of creativity and mathematical precision. It is both an art and a science, and hopefully at

least some of you will come to love this combination. One other reason it is so much fun is that algorithmic surprises abound. Hopefully this class will make you re-think what you thought was algorithmically possible, and cause you to constantly ask "is there a better algorithm for this task?". Part of the fun is that Algorithms is still a young area, and there are still many mysteries, and many problems for which (we suspect that) we still do not know the best algorithms. This is what makes research in Algorithms so fun and exciting, and hopefully some of you will decide to continue in this direction.

3 Etymology of the world "Algorithm"

As a round-about way of describing the etymology of the word "Algorithm", pause for a minute and consider how remarkable it is that 3rd graders can actually multiply large numbers. Its really amazing that anyone, let alone an 8-yr old, can multiply two 10-digit numbers. One reason multiplication is so easy for us is because we have a great data structure for numbers—we represent numbers using base-10 (Arabic) numerals, and this lends itself to easy arithmetic.

Why were romans so bad at multiplication? Well, imagine multiplying using roman numerals. What is LXXXIX times CM? The only way I can imagine computing this is to first translate the numbers into Arabic numerals [$LXXXIX = 50 + 10 + 10 + 10 + (-1) + 10 = 89$, and $CM = (-100) + 1000 = 900$] then multiplying those the standard way. Roman numerals seem like a pretty lousy data structure if you want to do arithmetic.

The word "Algorithm" is a mangled transliteration of the name "al-Khwarizmi." Al-Khwarizmi was a 9th-century Persian polymath, born in present-day Uzbekistan, who studied and worked in Baghdad during the Abbassid Caliphate; around 820 AD he was appointed as the astronomer and head of the library of the House of Wisdom in Baghdad. He wrote several influential books, including one with the title [something like] "On the Calculation with Hindu Numerals", which described how to do arithmetic using Arabic numerals (aka Arabic-Hindu,or just Hindu numerals). The original manuscript was lost, though a Latin translation from the 1100's introduced this number system to Europe, and is responsible for why we use Arabic numerals today. (You can imagine how happy a 12th century tax collector would have been with this new ability to easily do arithmetic....) [The old French word algorisme meant "the Arabic numerals system", and only later did it come to mean a general recipe for solving computational problems.]

4 Karatsuba Integer Multiplication

4.1 The problem

Suppose you have two large numbers, and you want to multiply them. Of course, you all know how to solve this problem: you learned an algorithm (which we'll call the

"grade-school algorithm") when you were in grade school. The question is, **can we do better?**

In order to understand this, we need to talk at least a little bit about what we mean by better. How do we measure the running time of an *algorithm*? It's tempting to measure it in units of time—say, milliseconds on a computer. However, this doesn't really capture the running time of an algorithm. Rather, it captures the running time of an algorithm, with a particular implementation, on a particular piece of hardware. For example, grade-school multiplication is much faster on a computer than by hand, but it's still the same algorithm in both cases.

Instead, we'll focus on how fast the running time *scales* as a function of the input. We will be a bit more precise about this in the next lecture, but for now, we'll define this notion by example. Suppose we use the grade-school algorithm to multiply two n-digit numbers. The bulk of the work is taken up by multiplying every pair of digits together. For example, in 1234×6789, we have to multiply 9×4, 9×3, 9×2, 9×1, 8×3, etc. There are n^2 such pairs, so we'll say that this algorithm has a running time that scales like n^2.

Why should we care about this measure of complexity? We'll talk about this more next lecture, but intuitively, this scaling behavior is the thing that really matters as n gets large. Suppose we had two algorithms, one of which had running time that scaled like n^2 and one which scaled like $n^{1.6}$. Suppose that running an algorithm by hand is 10000 times slower than running an algorithm on a computer. For large enough n, $10000n^{1.6} < n^2$, and intuitively this means it would actually be faster to run the $n^{1.6}$ algorithm by hand than the n^2 algorithm on a computer. So we can definitively say that the $n^{1.6}$ algorithm is "faster," because for large n, it will be faster, no matter how the algorithm is implemented and no matter what hardware it's running on.

With that in mind, our question is now this: can we multiply two n-digit integers faster than the grade-school algorithm? That is, with a running time that scales faster than n^2?

One try might be to store the answers ahead of time, or at least store partial answers. For example, we could store the products of all pairs of n-digit numbers, and then just look up the pair we need. This does result in performance gains, however, and also leads to exponential storage costs. (For example, if $n = 100$, we would need to store a table of $10^{2n} = 10200$ products. Note that the number of atoms in the universe is *only* $\approx 10^{80}$. . . .) So we'll have to do something more clever.