MAT 302: LECTURE SUMMARY

MORE ON APPROXIMATION

We began lecture with a discussion of the $O(\cdot)$ notation introduced last time. Consider the following chain of approximations:

$$\begin{aligned} x - 1000\sqrt{23x+40} + 10\sin x &= x - 1000\sqrt{23x+40} + O(1) \\ &= x + O(\sqrt{x}) \\ &= \left(1 + O\Big(\frac{1}{\sqrt{x}}\Big)\right) x \qquad (\dagger) \\ &= \Big(1 + o(1)\Big)\, x \qquad (\ddagger) \end{aligned}$$

Note that with the exception of (†), each approximation is weaker than the last (in the sense that we lose some information at each stage). The first two equalities are discussed in the January 13th lecture summary, so I won't say much about them, other than to mention Kiavash's nice idea for showing that $O(\sqrt{23x+40}) = O(\sqrt{x})$: he first observed that

$$\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$$

(prove this!), from which he deduced that

$$\begin{aligned} \sqrt{23x+40} &\leq \sqrt{23x} + \sqrt{40} \\ &= \sqrt{23}\,\sqrt{x} + O(1) \\ &= O(\sqrt{x}) + O(1) \\ &= O(\sqrt{x}). \end{aligned}$$

Make sure you understand why each of these transitions is legal.

The approximation (†) is fairly straightforward to justify; it illustrates that one can factor into or out of a $O(\cdot)$. In (‡), $o(1)$ means a quantity which tends to 0 as $x \to \infty$. Note that any function which equals $\big(1 + o(1)\big)x$ is asymptotic to x.

SECURITY AND COMPLEXITY

We spent the end of the lecture discussing security and complexity. There were two notions of security we mentioned:

(1) A practical approach, in which a cryptosystem is theoretically breakable but takes too long to do so in practice; and

Date: January 18th, 2011.

(2) A theoretical notion: a cryptosystem is truly (information-theoretically) secure if there does not exist a way to crack the system.

Next time, you will discuss an example satisfying the second criterion. In the meantime, we talked a bit about the first. What is it that makes a cryptosystem difficult to break, even if you know encryption algorithm? As we saw earlier with the substitution cipher, having a large key space doesn't imply that a system is hard to crack.

The real question is to figure out a precise meaning for the word 'hard'. How do we measure how hard a computation is? This is the realm of complexity theory in computer science, and we will talk more about it later. For now, we developed a bit of an intuition.

How hard is addition? You evaluated $7 + 8$ instantly, $37 + 68$ with some hesitation, $437 + 668$ after a longer pause, etc. Of course, as the numbers get bigger, it gets harder to add them – no surprise there. Still, addition is pretty easy.

It's a similar story with multiplication, but the difficulty increases a little faster than with addition. For example, 7×8 you knew instantaneously, but the next harder example 17×43 you had to work out on paper, and 417×743 is already too tedious to work out. Still, in principle, it wouldn't take that much work: more or less three step to evaluate 417×3, another three to evaluate 417×4, another three to evaluate 417×7, and finally you have to add shifts of the three numbers together to finish. By contrast, in adding two three-digit numbers you would have to make roughly 3 calculations.

In the above examples, we saw that the complexity of the problem grew with the size of the input. But what does size mean, exactly? This point is somewhat subtle. Here are two different questions, each relating to the size of a number N:

(1) How long does it take to count up to N?
(2) How long does it take to write N down?

The first is what we normally think of as the size of N; the answer is $\asymp N$. The second question is more relevant to studying the complexity of a calculation; the answer in this case is $\asymp \log N$. (Make sure you understand why this is the case! Note that all $\log$s in this course are base e.)