MAT 302: LECTURE SUMMARY

First, motivated by a problem from the assignment, we discussed a technique for sizing up a nasty integral. Consider $\displaystyle\int_1^x \frac{\log t}{\sqrt{t}}\,dt$. Although this integral cannot be evaluated exactly, we can get an idea of how it grows with x. The idea is to split the integral into two pieces, one of which contains the bulk of the original integral, and the other of which is insignificant by comparison. More precisely:

$$\text{(*)} \qquad \int_1^x \frac{\log t}{\sqrt{t}}\,dt = \int_1^{\sqrt{x}} \frac{\log t}{\sqrt{t}}\,dt + \int_{\sqrt{x}}^x \frac{\log t}{\sqrt{t}}\,dt.$$

We begin by estimating the size of the second integral on the right hand side:

$$\int_{\sqrt{x}}^x \frac{\log t}{\sqrt{t}}\,dt \le \int_{\sqrt{x}}^x \frac{\log x}{\sqrt{t}}\,dt \le C\sqrt{x}\log x$$
$$\int_{\sqrt{x}}^x \frac{\log t}{\sqrt{t}}\,dt \ge \int_{\sqrt{x}}^x \frac{\log \sqrt{x}}{\sqrt{t}}\,dt \ge c\sqrt{x}\log x$$

for some positive constants c and C. In other words, we have shown that

$$\int_{\sqrt{x}}^x \frac{\log t}{\sqrt{t}}\,dt \asymp \sqrt{x}\log x.$$

The first integral on the right hand side of (*) is small by comparison:

$$\int_1^{\sqrt{x}} \frac{\log t}{\sqrt{t}}\,dt \le \int_1^{\sqrt{x}} \frac{\log \sqrt{x}}{\sqrt{t}}\,dt = O(x^{1/4}\log x) = o(\sqrt{x}\log x)$$

It follows that

$$\int_1^x \frac{\log t}{\sqrt{t}}\,dt \asymp \Big(1+o(1)\Big)\sqrt{x}\log x \asymp \sqrt{x}\log x.$$

So, even though we cannot evaluate the integral, we can get a decent sense of how quickly it grows with x.

From here, we moved on to the topic of how easy or hard a problem is. In computer science, this has a distinct meaning from the mathematical one: a problem is easy to solve if it can be solved *quickly*, and hard otherwise. For example, evaluating the sum of two D-digit numbers requires $O(D)$ steps, so even for enormous numbers with 100 digits, it requires relatively few steps to sum them. Similarly, multiplying two D-digit numbers requires $O(D^2)$ steps; again, even if the numbers in question are enormous (e.g. 100 digits long each), the computation of their product takes on the order of 10,000 steps, a very manageable number. We also figured out that evaluating an exponential of the form x^N is easy: it takes $O(\log N)$ steps. In general, when you see $\log N$, you should think of it as the number of digits of N.

Date: March 15th, 2011.

By contrast, the problem of factoring $N = PQ$ (where P and Q are primes) seems to be hard: the best algorithm we've come up with is essentially the brute force one, in which we search through all the odd numbers smaller than $\sqrt{N}$ for the prime divisors of N. If P and Q are 100-digit numbers, such a search would take on the order of 10^{100} computations, which is entirely undoable. Of course, this doesn't prove that factoring N is a hard problem – it's quite likely that there are significantly faster ways to factor N which humans haven't yet discovered.

The number of steps required to solve an easy problem (in the above sense) is polynomial in the number of digits of the input. We will say that a problem can be solved in *polynomial time* if there exists an algorithm and a constant k such that the algorithm solves the problem in $O\Big((\log N)^k\Big)$ steps, where N is the input. (Remember that intuitively, $\log N$ represents the number of digits of N.) Let **P** denote the set of all problems which can be solved in polynomial time.

Factoring is an example of a problem which may or may not belong to **P** – nobody knows for sure. On the other hand, factoring is easy to *verify*. Suppose I give you $N = PQ$, as well as P. It's then a trivial matter to factor N – simply divide it by P and you're done! The set of all problems which can be verified in polynomial time is called **NP**, which stands for *nondeterministic polynomial time*. Thus, factoring is in **NP**.

One of the most famous open questions in computer science is: does $\mathbf{P} = \mathbf{NP}$? In other words, if a problem's solutions are easy to verify, does this necessarily imply that it was easy to solve in the first place? If someone were to discover a polynomial-time algorithm for factoring, this would prove that $\mathbf{P} \neq \mathbf{NP}$. On the other hand, if someone proves that $\mathbf{P} \neq \mathbf{NP}$, that wouldn't say anything about the specific problem of factoring. Make sure you understand this point.

We also discussed a couple of other open questions, including:

- Can $\varphi(N)$ be computed in polynomial time?
- Can $\mu(N)$ be computed in polynomial time?