Section 3.3, selected answers
Math 114 Discrete Mathematics

D Joyce, Spring 2018

8. Consider "Horner's method" for evaluating polynomials as described in the text.

a. Evaluate $3x^2 + x + 1$ at $x = 2$ by working through each step of the algorithm. Here, we're treating the polynomial $3x^2 + x + 1$ as if it were written $(3x + 1)x + 1$ and computing first what's inside the parentheses. More generally, a polynomial

$$a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x^1 + a_0$$

is computed as if it were written

$$(\ldots((a_n x + a_{n-1})x + a_{n-2})x + \ldots + a_1)x + a_0.$$

The details of computation for the specific example look like this:

Multiply 3 by 2 to get 6
Add 1 to get 7
Multiply that by 2 to get 14
Add 1 to get 15, the answer.

b. How many multiplications and additions are needed?

n multiplications; n additions. Note that there were only half as many multiplications needed as in problem 7. Multiplications take the most time, so this algorithm is about twice as fast as the naive algorithm in problem 7.

9. How large a problem can be solved in 1 second using an algorithm that requires $f(n)$ bit operations, where each bit operation is carried out in 10^{-9} seconds, with the following values for $f(n)$?

A nanosecond is 10^{-9} seconds. So 10^9 operations can be executed in a second. So the question becomes How large is n so that $n = 10^9$.

a. $\log n$? If $\log n$ is a billion (10^9), then what is n? These logs are base 2. In general,

$$\log_2 n = x \quad \text{iff} \quad n = 2^x$$

Therefore, $n = 2^{1000000000}$. That's a huge number. Since 2^{10} is about 10^3, therefore $2^{1000000000}$ is about $10^{300000000}$. That's 1 followed by 300 million 0s. It's too big to imagine! Algorithms that are $\mathcal{O}(\log n)$ are really speedy.

b. n? Then n is a billion, of course. $\mathcal{O}(n)$ is pretty speedy.

c. $n \log n$? Well, $n \log n = 1000000000$ isn't easily solved for n. Using a calculator, you can figure it out to be about $n = 40000000$, 40 million. $\mathcal{O}(n \log n)$ is still pretty speedy.

d. n^2? Well, $n^2 = 1000000000$ gives $n = \sqrt{1000000000}$, which is about 31623. Not such a big number. $\mathcal{O}(n^2)$ is not nearly as good as $\mathcal{O}(n \log n)$.

e. 2^n? So, $2^n = 1000000000$. To solve for n, take $\log_2$ of each side. Then $n = \log_2 1000000000$ which is about 30 (since $\log_2 1000$ is about 10). So exponential algorithms only work for small n.

f. $n!$? Even when n is pretty small, its factorial can be pretty big. For instance, $12! = 479001600$, which is about half a billion, while $13! = 6227020800$, which is about 6 billion. So $n = 12$ is the largest value of n that can be treated in less than a second. Factorial algorithms are so slow, they only work for very small n.

Math 114 Home Page at `http://math.clarku.edu/~djoyce/ma114/`