I. Core / A. Correctness and efficiency of algorithms

- Review of algorithms, correctness, and efficiency (week-before Friday and last week Wednesday)
- Asymptotics (last week Friday and **today**)

I. Core / B. Divide and Conquer

- General introduction (**today**)
- Solving recurrences (Wednesday)
- The master method (Friday)
- Quick sort (next week Monday)

Today:

- Review meaning and formal definition of big-Theta
- Theorem 3.1; Ex 3.1-(4 & 5)
- Properties of asymptotic notation
- Problem 3-1.d
- Problem 3-4.(a,b,c)
- Common functions (Section 3.2)
- Begin divide and conquer (Section 4.1)

Formal definition of big-Theta:

$$\Theta(g(n)) = \{f(n) \mid \exists\ c_1, c_2, n_0 \in \mathbb{N} \text{ such that } \forall\ n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

$g(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$.

Proof. *Let* $c_1 = \frac{1}{14}$, $c_2 = \frac{1}{2}$ *and* $n_0 = 7$. *Suppose* $n > 7$. *Then*

$$\begin{array}{rcccl} \frac{1}{14} & = & \frac{1}{2} - \frac{3}{7} < \frac{1}{2} & & \\ \frac{1}{14} & \leq & \frac{1}{2} - \frac{3}{n} & \leq & \frac{1}{2} \\ \frac{n^2}{14} & \leq & \frac{1}{2}n^2 - 3n & \leq & \frac{n^2}{2} \\ c_1 n^2 & \leq & g(n) & \leq & c_2 n^2 \end{array}$$

Therefore $g(n) = \Theta(n^2)$ *by definition.* $\square$

Formal definitions:

$$\Theta(g(n)) = \{f(n) \mid \exists\ c_1, c_2, n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

$$O(g(n)) = \{f(n) \mid \exists\ c, n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq f(n) \leq c\ g(n)\}$$

$$\Omega(g(n)) = \{f(n) \mid \exists\ c, n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq c\ g(n) \leq f(n)\ \}$$

$$o(g(n)) = \{f(n) \mid \forall\ c \in \mathbb{R}^+, \exists\ n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq f(n) < c\ g(n)\ \}$$

$$\omega(g(n)) = \{f(n) \mid \forall\ c \in \mathbb{R}^+, \exists\ n_0 \in \mathbb{R}^+ \text{ such that } \forall\ n \geq n_0, 0 \leq c\ g(n) < f(n)\ \}$$

Theorem 3.1. For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Theorem 3.1. For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Proof. *Suppose $f = \Theta(g(n))$. Then, by definition of Θ, there exist constants c_1, c_2, and n_0 such that for all $n \geq n_0$,*

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

Let $c = c_2$. Then $0 \leq f(n) \leq c \cdot g(n)$, hence $f(n) = O(g(n))$ by definition. Similarly, let $c = c_1$. Then $0 \leq c \cdot g(n)$, hence $f(n) = \Omega(g(n))$.

Conversely, suppose $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. By the definitions, there exist c, and n_1 such that for all $n \geq n_1$, $0 \leq f(n) \leq c \cdot g(n)$, and there exist c', and n_1' such that for all $n \geq n_1'$, $0 \leq c' \cdot g(n) \leq f(n)$.
Let $c_1 = c'$, $c_2 = c$, and $n_0 = max(n_1, n_1')$. Hence $f(n) = \Theta(g(n))$. $\square$

3.1-4. Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

3.1-4. Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

To see that $2^{n+1} = O(2^n)$, note that $2^{n+1} = 2 \cdot 2^n$. Thus 2 is the constant we're looking for, and we're done.

Let's attempt a proof that $2^{2n} = O(2^n)$. Does $\exists\ c, n_0 \mid \forall\ n \leq n_0, 2^{2n} \leq c \cdot 2^n$? If so, then

$$\begin{aligned} 2^n \cdot 2^n &\leq c \cdot 2^n \\ 2^n &\leq c \end{aligned}$$

. . . which is impossible.

3-1.d. If $k > d$, then $p(n) = o(n^k)$.

Proof. Suppose $k > d$ and suppose $c > 0$. Then

$$\begin{aligned} a_0 + a_1 n + \ldots + a_d n^d &< a_x + a_x n + \ldots + a_x n^d && \text{where } a_x = \max(a_0, a_1, \ldots a_d) \\ &< d \cdot a_x n^d && \text{(see why I chose } a_x \text{ instead of } a_m\text{?)} \\ &< c \cdot n^k && \text{if } n \text{ is big enough.} \end{aligned}$$

So, we want $d \cdot a_x < c \cdot n^{k-d}$. This holds as long as

$$n > \left(\frac{d \cdot a_x}{c} \right)^{\frac{1}{k-d}}$$

If $f(n) = O(g(n))$ and $\lg(g(n)) \geq 1$ and $f(n) \geq 1$ for sufficiently large n, then $\lg(f(n)) = O(\lg(g(n))$.

Scratch work: We need a d such that

$$\begin{aligned} \lg c + \lg g(n) &\leq d \lg g(n) \\ d &\geq \frac{\lg c}{\lg g(n)} + \frac{\lg g(n)}{\lg g(n)} \\ &\geq \lg c + 1 \\ (\lg c + 1)\lg g(n) &= \lg c \cdot \lg g(n) + \lg g(n) \end{aligned}$$

Proof. *Suppose $f(n) = O(g(n))$. Then there exist c, n_0 such that for all $n > n_0$, $f(n) \leq c \cdot g(n)$. Then*

$$\begin{aligned} lg\ f(n) &\leq lg\ c\ g(n) && \text{since lg is increasing} \\ &\leq lg\ c + lg\ g(n) && \text{by log property} \\ &\leq lg\ c \cdot lg\ g(n) + lg\ g(n) && \text{Since lg } g(n) \geq 1 \\ &\leq (lg\ c + 1) \cdot lg\ g(n) \end{aligned}$$

Thus for $n > n_0$, $\lg(f(n)) \leq (lg\ c + 1)\lg(g(n))$.

Big "morals" of §4.(1 & 2)

- Many problems have good divide and conquer solutions. The running time of a divide and conquer algorithm can be captured by a recurrence. So, let's make sure we can do recurrences.
- Sometimes it's divide-and-conquer even when it doesn't seem like it is.
- "Solving" a recurrence means finding an equivalent non-recursive formula.

"Normal" math induction:

"Normal" math induction:

$$\begin{aligned} & I(0) \\ & I(n) \rightarrow I(n+1) \\ \therefore \quad & \forall\ n \in \mathbb{N}, I(n) \end{aligned}$$

"Strong" math induction:

$$\begin{aligned} & I(0) \\ & (\forall\ i \leq n, I(i)) \rightarrow I(n+1) \\ \therefore \quad & \forall\ n \in \mathbb{N}, I(n) \end{aligned}$$

For next time

Read sections 4.(2 & 3). When reading about Strassen's method in Section 4.2, don't get bogged down in the details of matrix multiplication. If you are getting tired by pg 79, then skip ahead to the next section. It's much more important to give Section 4.3 your full attention.

Do 4.3-(1,2,9)