CS 240 Midterm Review (Module 1–7)

Asymptotic Analysis

- Problem instance (I) – *input* for the specified problem
- Problem solution – *output* for the specified problem instance
- Problem size – Size(I) = size of instance I
- Algorithm - a step-by-step process for carrying out a series of computations
  - An algorithm A solves a problem P if, for every instance I of P, A computes a valid solution for I in <u>finite</u> time
- RAM model
  - Assume any memory access & primitive operation is constant time
  - Assume infinite amount of memory
  - Sequential operation
  - Running time is determined by the # of memory accesses & primitive operations
- Order notations
  - $\underline{f(n) \in O(g(n))}$ if $\exists\ c > 0$ and $n_0 > 0$ such that $0 \le f(n) \le cg(n)\ \forall\ n \ge n_0$
    - f "grows no faster than" g
    - f is "upper-bounded" by g ($\le$)
  - $\underline{f(n) \in \Omega(g(n))}$ if $\exists\ c > 0$ and $n_0 > 0$ such that $0 \le cg(n) \le f(n)\ \forall\ n \ge n_0$
    - f "grows no slower than" g
    - f is "lower-bounded" by g ($\ge$)
  - $\underline{f(n) \in \Theta(g(n))}$ if $\exists\ c_1, c_2 > 0$ and $n_0 > 0$ such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)\ \forall\ n \ge n_0$
    - f and g grow at the same rate
  - $\underline{f(n) \in o(g(n))}$ if $\forall\ c > 0, \exists\ n_0 > 0$ such that $0 \le f(n) < cg(n)\ \forall\ n \ge n_0$
    - f is "*strictly* upper-bounded" by g ($<$)
  - $\underline{f(n) \in \omega(g(n))}$ if $\forall\ c > 0, \exists\ n_0 > 0$ such that $0 \le cg(n) < f(n)\ \forall\ n \ge n_0$
    - f is "*strictly* lower-bounded" by g ($>$)
  - Suppose $L = \lim_{n\to\infty} \dfrac{f(n)}{g(n)}$
    - If $L = 0$ then $f \in o(g)$
    - If $0 < L < \infty$ then $f \in \Theta(g)$
    - If $L = \infty$ then $f \in \omega(g)$
  - If $f \in O(g)$ and $f \in \Omega(g)$, then $f \in \Theta(g)$
- Loop analysis
  - Begin from the innermost nested loop; use $\sum$ for each outer loop

- Recurrence relations analysis
  - e.g. mergesort:
  - Step 1: split array of length n into two subarrays, of lengths $\lceil \frac{n}{2} \rceil$ and $\lfloor \frac{n}{2} \rfloor$ ($T = \Theta(n)$)
  - Step 2: recursively run mergesort on subarrays ($T = T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor)$)
  - Step 3: merge sorted subarrays into a single sorted array ($T = \Theta(n)$)
  - Thus the <u>recurrence relation</u> is

$$
\begin{aligned}
T(n) &= \Theta(1) && \text{if } n = 1 \\
T(n) &= T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n) && \text{if } n > 1 \\
&= 2T(\frac{n}{2}) + cn \\
&= 2(2T(\frac{n}{4}) + \frac{cn}{2}) + cn \\
&= \ldots \\
&= 2^k T(\frac{n}{2^k}) + kcn && \text{where } k = \log n \\
&= nT(1) + \log n(cn) \\
&\in \Theta(n \log n)
\end{aligned}
$$

  - **In general**, $\{T(n) = T(n/2) + c\} \in \Theta(n \log n)$

Priorty Queues

-