CS 170 Discussion 1 Reference Sheet

In this class, we care a lot about the runtime of algorithms. However, we don't care too much about concrete performance on small input sizes (most algorithms do well on small inputs). Instead we want to compare the *asymptotic (i.e. long-term)* growth of the runtimes.

Asymptotic Notation: The following are definitions for $\mathcal{O}(\cdot), \Theta(\cdot)$, and $\Omega(\cdot)$:

- $f(n) = \mathcal{O}(g(n))$ if there exists a $c > 0$ where after large enough n, $f(n) \leq c \cdot g(n)$. *(Asymptotically, f grows at most as much as g)*
- $f(n) = \Omega(g(n))$ if $g(n) = \mathcal{O}(f(n))$. *(Asymptotically, f grows at least as much as g)*
- $f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$. *(Asymptotically, f and g grow at the same rate)*

If we compare these definitions to the order on the numbers, $\mathcal{O}$ is a lot like $\leq$, Ω is a lot like $\geq$, and Θ is a lot like $=$ (except all are with regard to asymptotic behavior).

If we would like to prove asymptotic relations instead of just using them, we can use limits.

Asymptotic Limit Rules: If $f(n), g(n) \geq 0$:

- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, then $f(n) = \mathcal{O}(g(n))$.
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, for some $c > 0$, then $f(n) = \Theta(g(n))$.
- If $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$, then $f(n) = \Omega(g(n))$.

Note that these are all sufficient (and not necessary) conditions involving limits, and are not true definitions of $\mathcal{O}$, Θ, and Ω. We highly recommend checking on your own that these statements are correct!)

In the box below, we describe strategies of solving recurrence relations.

Master Theorem: If the recurrence relation is of the form $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$ for some constants $a > 0$, $b > 1$, and $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

Remember that if the recurrence relation is *not* in the form $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$, you can't use Master Theorem! If this is the case, you can try the following strategies:

1. **"Unraveling" the recurrence relation.**

   We try to recursively keep on plugging in the smaller subproblems (e.g. $T(n/2)$ or $T(n-1)$) to $T(n)$ to find a pattern or simply directly compute the entire expression.

2. **Draw a tree!**

   We use a tree representation to count the total number of calls on each subproblem, doing so by summing up the work per level.

3. **Change of Variables**

   We can make a change of variables (kind of like u-substitution for integration) to mold our recurrence into a nicer form to work with.

4. **Squeeze**

   For certain recurrence relations where we can't directly compute the solution, we can indirectly solve it by computing upper and lower bounds for the runtime based on known recurrence relations. In particular, if the upper and lower bounds have the same asymptotic growth, then we have our answer!

5. **Squeeze + Guess & Check**

   If we try using the previous method and can't end up with asymptotically equivalent upper/lower bounds, then we resort to guess-and-checking reasonable runtimes between the bounds to arrive at the solution (look up "The Substitution Method for Solving Recurrences – Brilliant" to see how to do this). For the purposes of this class, if you ever have to resort to using this method, the expression for $T(n)$ will always look "nice."

There are a lot more strategies that are out-of-scope for this class, and if you're curious we highly recommend you to read about them in the following link: `http://beta.iiitdm.ac.in/Faculty_Teaching/Sadagopan/pdf/DAA/new/recurrence-relations-V3.pdf`.