# What I know *after* taking CS 31

a. **Worst Case Running Time.**

- Computational problem $\Pi$ has instances/inputs $I$; each input $I$ has solution/output $S$.
- An algorithm $\mathcal{A}$ for $\Pi$ takes $I \in \Pi$ and returns its solution $S$.
- Each instance $I \in \Pi$ has a notion of size $|I|$.
  Often, this is the number of bits required to describe $I$.
- The running time of algorithm $\mathcal{A}$ on $I$ is denoted as $T_{\mathcal{A}}(I)$.
- The ***worst case running time*** of $\mathcal{A}$ as a function of size is defined to be

$$T_{\mathcal{A}}(n) := \max_{I \in \Pi : |I| \le n} T_{\mathcal{A}}(I)$$

b. **The Big-Oh Notation.**

- Useful notation to tell the "big picture" without worrying about annoying details.
- $g(n) \in O(f(n))$ if $\exists \ a, b > 0$ such that for all $n \ge b$, $g(n) \le a \cdot f(n)$.
- $g(n) \in \Omega(f(n))$ if $\exists \ a, b > 0$ such that for all $n \ge b$, $g(n) \ge a \cdot f(n)$.
- $g(n) \in \Theta(f(n))$ if $g(n) \in O(f(n))$ and $g(n) \in \Omega(f(n))$.
- Often the $\in$ is replaced by $=$; so we would say $T(n) = O(n^2)$ to imply $T(n) \in O(n^2)$.
- $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ often tells us the relation.
- Beware: limits may not exist. In that case, the definition is what we must revert to.

c. **Recursive Algorithms and Recurrence Inequalities.**

- One big theme of algorithms design is recursion: break into smaller subproblem, solve the smaller subproblem recursively, and pop-up.
- Recursive algorithms are analyzed using recurrence inequalities.
- Recurrence inequalities relate the running time $T_{\mathcal{A}}(n)$ to $T_{\mathcal{A}}(m)$ for smaller $m < n$.
- There is a base case involved.
- Solved using the "kitty method."

d. **Divide and Conquer.**

- Break a problem into two (or more), recursively solve, combine solutions.
- Often works for speeding up algorithms for which a not-so-bad naive solutions exist.
- Problems seen: MERGE SORT, COUNTING INVERSIONS, POLYNOMIAL MULTIPLICATION, CLOSEST PAIR OF POINTS, many others in the Psets.
- Analysis Tool : Master Theorem.

e. **Dynamic Programming.**

- Smart Recursion / Recursion with Memory.
- Think of optimum solution; see if solution can be built by combining solutions of smaller subproblems.
- Smaller subproblems should be "succinctly representable". The value should be defined by a "function" on not too many parameters. Function should have a recurrence relation.
- Six-Step Solution Presentation
  (a) Definition of the function.
  (b) Base Cases.
  (c) Recurrence.
  (d) Proof of Recurrence.
  (e) Pseudocode (including recovery)
  (f) Runtime and space.
- Problems Seen: SUBSET SUM, KNAPSACK, EDIT DISTANCE, LONGEST INCREASING SUBSEQUENCE, and many others in the Psets.

f. **Depth First Search.**
   - Revisiting an old algorithm.
   - Lots of power in the first and last's returned.
   - Applications: CONNECTIVITY, CYCLE?, TOPOLOGICAL ORDER of DAGs, STRONGLY CONNECTED COMPONENTS: all in *linear* $O(n+m)$ time!
   - Topological Order solves many problems in DAGs via dynamic programming: LONGEST PATH, and others in PSets.

g. **Breadth First Search.**
   - Shortest hop-length walks in $O(n+m)$ time.
   - Distance Labels as a *certificate* of optimality.
   - Queue implementation leads to fast implementation.
   - Useful for checking if a graph is BIPARTITE? (in UGP)
   - The "weighted" generalization also finds shortest cost paths (explored in UGP).

h. **Dijkstra.**
   - Clever generalization of BFS which works when graphs have positive cost edges.
   - *Doesn't* necessarily work with negative cost edges. Beware!
   - Main idea : don't add vertex in queue once distance label updated. Only one vertex with the smallest distance label is added.
   - Runs in $O(m+n\log n)$ time using Fibonacci heaps. Or in $O(m\log n)$ time using usual heaps.
   - Same idea solves the *maximum capacity* path from source to vertex.
   - Can also be used to find shortest length cycles (this was done in problem set).

i. **Bellman-Ford.**
   - In graphs with possibly negative cost edges, this algorithm either detects negative cost cycles, or figures out shortest paths.
   - Finds shortest cost walks whose lengths are bounded. In case of no negative cost cycles, shortest walks are shortest paths.
   - Dynamic program. Runs in $O(mn)$ time.
   - We did this problem on ***directed*** graphs. The problem can also be solved in undirected graphs, but that's a story for another rainy day.
   - All pairs shortest paths can be found in $O(n^3)$ time (this was done in a problem set.)