- Cost Analysis: pick a measure of interest (e.g # of times a certain operation is called)
  - ↳ Recursive: construct a recurrence relation for the measure of interest, use master theorem
  - ↳ Iterative: do loop analysis to show the cost of each loop & thus total cost
- Correctness/Termination
  - ↳ Recursive (strong or weak induction): show that the algo returns correct result for base case, assume it returns the correct result for the recursive calls, show it returns the correct result after recursive calls
  - ↳ Iterative (loop invariant) construct some loop invariant P(K) that shows that the algo has built a "correct partial solution after K iterations. Show by induction that it holds $\forall n \in \mathbb{N}$
  - ↳ Termination: show that the recursive calls are happening on instances of smaller sizes or that the while-loop's continuation condition will eventually be false
- Master Theorem (for $T(n) = aT(n/b) + O(n^d)$, consider a vs. b^d)
  - ↳ $a < b^d \Rightarrow T(n) \in \Theta(n^d)$
  - ↳ $a = b^d \Rightarrow T(n) \in \Theta(n^d \log n)$
  - ↳ $a > b^d \Rightarrow T(n) \in \Theta(n^{\log_b a})$
- Divide and Conquer Generic Strategy (n = size of input)
  - ↳ If base case (typically $n=1$), return trivial solution
  - ↳ Else:
    1. Break input into a parts of size (n/b)
    2. Recursively call algorithm on each of these a parts to get a results
    3. Take the a results and merge them to get final answer, return it
- Generic Greedy Strategy
  1. Sort the input according to a certain attribute (ascending or descending)
  2. Initialize some answer to be returned after 3.
  3. Loop over the sorted input
     - ↳ If the current selection can be added to the answer, add it to the answer
     - ↳ Otherwise, skip over the current selection
- Generic Greedy Correctness Proof
  - ↳ Show feasibility of solution — prove that the solution doesn't violate the constraints of the problem
  - ↳ Greedy "stays ahead"
    - ↳ Let $\Omega = \{\omega_{1,2,\dots m}\}$ be some optimal solution and $G = \{g_{1,2,\dots k}\}$ be the solution made by the greedy algorithm
    - ↳ Come up with some fxn $f(S)$ = some numeric aspect of solution S, use induction to show:
      $P(n) = [f(\{g_{1,2,\dots n}\}) \leq f(\{\omega_{1,2,\dots n}\})] \; \forall n \in \mathbb{N}$
    - ↳ Use above relationship to show $G \equiv \Omega$ (usually contradiction)
  - ↳ Exchange argument: let Ω be some opt. soln, G be solution made by greedy algorithm
    - ↳ Type 1: Swapping within the solution
      - ↳ Assume there is a pair of elements in Ω whose order violates the greedy heuristic
      - ↳ Show that by switching this pair of elements to obey the greedy heuristic, the cost of this modified optimal solution is no worse than the "cost" of the original optimal solution
    - ↳ Type 2: Swapping out mismatches
      - ↳ Assume that $e_1 \notin \Omega$, $e_1 \in G$, and $e_2 \in \Omega$, $e_2 \notin G$
      - ↳ Show that by swapping $e_1 \leftrightarrow e_2$, the "cost" of G doesn't increase beyond the "cost" of Ω
    - ↳ Use induction to show that if we keep doing these switches to modify optimal soln $\rightarrow$ greedy soln, $\Omega \rightarrow G$ at no additional "cost"
- Dynamic Programming Solution Elements
  - ↳ Lookup Table (array where all solutions are stored)
    - ↳ For an N input problem, let C be the N-dimensional array that stores optimal sub-solutions
  - ↳ Semantic Array (English description of lookup table)
    - ↳ $C[i_1, i_2, \dots i_N]$ = value of the optimal solution for problem w/ $att_1 = i_1$, $att_2 = i_2$, ... $att_N = i_N$
    - ↳ The answer will be located @ $C[j_1, j_2, \dots j_N]$, where the initial input has $att_1 = j_1$, $att_2 = j_2$, ... $att_N = j_N$
  - ↳ Computational Array (Mathematical description of lookup table)
    - ↳ $C[i_1, i_2, \dots i_N]$ = operation that uses sub-solutions to obtain solution to input w/ $att_1 = i_1$, $att_2 = i_2$, ... att
  - ↳ Demonstrate the equivalence between semantic & computational arrays to prove correctness