# Algo Final Exam Study Guide

What is Dynamic programming?

- Requires **optimal substructure -** solution to larger problems is made up of solutions to smaller ones

Steps:

1. Identify a recursive structure of the problem
    a. What is the "last thing" done?
2. Save solution to each subproblem in memory
3. Select a good order for solving subproblems

Difference between top down and bottom up?
Asymptotically the same, bottom up works faster IRL bc not dealing with the stack

Top-Down: Recursion
Bottom-Up: Iterative

Dynamic Programming Examples

- Domino Tiling
- Log Cutting
- Matrix Chaining
- Seam Carving
- Longest Common Subsequence
- Gerrymandering

Log Cutting

1. Array for cut(n) of length n+1
    a. keep track of all the cuts you've made
2. for loop, recursively compute a solution
3. Keep track of memory for solutions

Solve from smaller to larger logs, find the cut + price of cut
At the largest log, find the max of all the previously computed values

What are Greedy Algorithms?

- Requires **optimal substructure -** solution to larger problems is made up of solutions to smaller ones
- **Only one subproblem to consider!**

Steps:

1. Identify a **greedy choice property**
2. Repeatedly apply the choice property until no subproblems remain

Greedy Algorithms

- Changemaking (briefly)
- Interval Scheduling
- Huffman Coding
- Belady Cache Replacement
- Questions:
    - Why log cutting not greedy

Choice Properties
- Shortest Interval
- Fewest conflicts
- Earliest Start
- Earliest end

How are greedy algorithms different from dynamic programming?
- Dynamic programming
    - Several choices for which small subproblem
    - Best last choice, look at all last choices & use recursive structure to find the best
    - Examples: log cutting, largest common subsequence, seam carving
- Greedy
    - Must only consider one choice for small subproblem
    - **Have to show the choice we make is always best choice**
    - Recurse on 1 subproblem, find 1 best choice with only 1 smaller subproblem to solve

Von Neumann Bottleneck
- memory/space tradeoff
- Processes are faster, looking up memory is slow
- Relationship between space/time complexity
- Hierarchical memory (registers → cache → disk)

Graphs
- Kruskal
- Prim
- Dijkstra's
- Bellman Ford
- Floyd-Warshall

Network Flow
- Ford-Fulkerson
    - Residual
    - Augmenting Path
- Edmunds Karp
- Max Flow, Min Cut
- Edge-Disjoint Path

- Vertex-Disjoint Path
- Max Bipartite Matching

Reduction

- Maximum Independent Set
- Minimum Vertex Cover
- NP Hard
    - 3-SAT
    - K-Independence Set
    - K-Vertex Cover
    - K-Clique