ECS122A Final Review

- Review material for final exam
  - Lecture notes (skip these “Extra LN”)
  - Solutions of homeworks (including “questions” on NPC), and midterms (available on canvas)

- What we have learned
  I. Basics and tools of trade
  II. Algorithm design techniques
  III. Graph algorithms
  IV. NP-completeness – brief introduction
I. Basics and tools of trade

1. Order of growth
   \( O, \Omega, \Theta \) definitions

2. Recurrence relations
   Linear recurrence relations
   Divide and conquer recurrence relations

3. The master theorem for solving the DC recurrence relations

4. Graph terminology and representations
   - graph, path, connected graph, connected component, cycle, acyclic,
     tree, spanning tree, sink, vertex-cover, clique, ...
   - adjacency list, adjacency matrix, incidence matrix.

5. Data structures:
   FIFO queue: enqueue, dequeue
   LIFO stack
   Priority queue: Insert(S,x), Extract-Min(S), Decrease-Key(S,x,k),
   ...
   Disjoint-set: Make-set(x), Union(x,y), Find-set(x)
II. Algorithm design techniques

Divide and Conquer

- Divide-and-Conquer algorithm – three steps:
  - **Divide** the problem into a number of (independent) subproblems
  - **Conquer** subproblems by solving them *recursively*. If the subproblem sizes are small enough, however, just solve them in a straightforward manner.
  - **Combine** the solutions to the subproblems into the solution of the original problem

- Examples:
  - MergeSort (vs. Insert sort)
  - The maximum and minimum values
  - The maximum subarray
  - Strassen’s algorithm for matrix multiplication
  - the closest pair of points in one dimension.
  - Searching for index $i$ such that $A[i] = i$ in a sorted array $A$
  - Integer multiplication
  - $k$-way merge operation
II. Algorithm design techniques

Greedy Algorithms

- Two elements of greedy algorithms
  - **The greedy-choice property**: a globally optimal solution can be arrived at by making a locally optimal (greedy) choice.
  - **The optimal substructure property**: an optimal solution to the problem contains within it optimal solution to subproblems.

- Examples (greedy works)
  - Activity selection
  - Huffman coding (data compression)
  - Job scheduling – minimizing the average completion time
  - MST

- Counter-examples (greedy does not work)
  - Knapsack problem
  - Money changing
II. Algorithm design techniques

Dynamic Programming

- Three key elements of dynamic programming
  - **The optimal substructure:** the optimal solution to the problem contains optimal solutions to subprograms ⇒ “recursion”
  - **Overlapping subproblems:** There are few subproblems in total, and many recurring instances of each. (unlike divide-and-conquer, where subproblems are independent)
  - **Memoization:** after computing solutions to subproblems, store in table, subsequent calls do table lookup.

- Examples:
  - Rod cutting
  - Matrix-chain multiplication
  - Longest common subsequence/substring
  - Edit distance
  - Knapsack problem
  - Change-making problem
III. Graph algorithms

- Elementary graph algorithms
  - Breadth-first search (BFS): FIFO queue, I/O, complexity
  - Depth-first search (DFS): LIFO stack, I/O, complexity

- Applications of BFS and DFS
  - sorting a dag
  - determining cycle
  - finding a sink
  - finding the connected components
III Graph algorithms

- Minimum Spanning Tree (MST)
  - Prim’s algorithm: priority queue, complexity
  - Kruskal’s algorithm: disjoint-set, complexity

- Shortest paths (single-source)
  - Bellman-Ford algorithm – “dynamic programming”
  - Dijkstra’s algorithm – priority queue – “greedy”
  - Bellman-Ford algorithm for DAG – only need a single pass

- Make sure to know how to precisely (correctly) illustrate a graph algorithm!
IV. NP-completeness – brief introduction

- Tractable and intractable problems
- Optimization problem and decision problem
- Polynomial reduction/transformation
- Formal definitions of P, NP, NP-complete and NP-hard
- Examples of intractable problems:
  Circuit-satisfiability (SAT), Graph-coloring, Hamiltonian-cycle (HC),
  Traveling-salesperson-problem (TSP), Knapsack-problem,
  Prime-testing, Subset-sum, Set-partition, Bin-packing, Vertex-cover,
  Clique problem.