Exam study sheet for CS6901

This test will cover the content of the course up to this moment (except NP-hardness lecture), especially assignments 1,2 and 3. To study, refer to the notes, as well as Kleinberg-Tardos book ("KT") and Cormen-Leiserson-Rivest-Stein ("CLRS") books, or any other books on algorithms.

For each algorithm design paradigm we studied, be able to give examples of algorithms adhering to this paradigm with problems they solve (e.g., Floyd-Warshall algorithm is dynamic programming, solves all-pairs shortest path). Know how to solve problems similar to the assignments (including proofs). For the applications, know how to model them as an abstract (e.g., graph) problem. Give the time complexity of the algorithms in O-notation, and mention, if relevant, data structures being used to achieve such performance. For example, Kruskal's algorithm for min. spanning tree uses Union-Find data structure, and with it can run in time $O(m \log n)$. Know which restrictions on the problem make it computationally easier (e.g., single-source shortest path with no weights, positive weights and any weights). **Make sure you know how to solve every problem on every submitted assignment.**

1. **Stable matching (stable marriage) problem**
   Know the statement of the problem, that the input is $2n$ preference lists of length n each. Know what is a matching, what is a stable/unstable matching). Be able to give examples. Know the algorithm to solve it (Gale-Shapley algorithm). Know some applications (e.g., medical interns assignment) and extensions; which data structures are needed to implement it.

2. **Background (complexity and data structures)**
   We are considering asymptotic worst-case performance of algorithms. Know O-notation. Know some lower bounds such as sorting, and the difference between an upper bound in O-notation (e.g., bubble sort and merge sort solve the same problem in different time), and lower bounds in Ω-notation (no comparison-based sorting faster than $\Omega(n \log n)$. When O and Ω coincide, use $\Theta(f(n))$ (e.g., sorting is $\Theta(n \log n)$). The definition of an "efficient" algorithm (polynomial-time and below are, exponential-time and above are not). For data structures, know about arrays, lists, priority queue based on heap data structure, and union-find data structure.

3. **Graph algorithms**
   Directed unweighted graphs: DFS, BFS, topological sort, strongly connected components, testing bipartiteness, algorithm on DAGs.
   Undirected weighted graphs: Minimal spanning trees: greedy algorithms by Kruskal, Prim-Jarnik, Boruvka.
   Directed weighted graphs: Dijkstra's single-source shortest-path (greedy, no negative weights). Bellman-Ford single-source shortest-path algorithm (allows negative weights, can detect a negative cycle, solves systems of difference constraints; use CLRS version, not KT). Floyd-Warshall (dyn. prog., all-pairs shortest-path).

4. **Greedy algorithms**
   Know the main idea (selecting a locally optimal solution at each step leads to a global optimum). Know how to design a greedy algorithm and prove its correctness, and be able to give examples of greedy algorithms.

5. **Dynamic programming**
   Know the steps of constructing a dynamic programming algorithm solution. Be able to define an array (as in "A(i,j) stores the value of... ", give a recurrence (as in "A(i,j) = ...", not pseudocode!), and recover the actual solution. Know when it runs in polynomial time and when it does not. Applications: all-pairs shortest path, scheduling with deadlines, profits, and durations, longest common subsequence, etc.

6. **Network flows.**
   Know what is a flow, how to compute a flow using Ford-Fulkerson, what is an augmenting path, what is a residual network. Give example where Ford-Fulkerson runs in exponential time; know about the example where it does not terminate (but it works on integers). Know how to make it polynomial-time (choose paths by breadth first search); you don't need to know the proof. Know the max flow min cut theorem, and applications to project selection/open pit mining (be able to name them). Know about circulations and dealing with lower bounds on flow. Use CLRS, KT and notes (especially KT).

7. **Restrictions**
   Know what a restriction on the data for a computational problem is. Be able to give examples of problems with restrictions that make the problem easier. Know that for stable matching, the restriction considered in the assignment, experimentally, makes the problem harder on average.