Exam study sheet for CS6783

Here is the list of topics we have covered. We talked about algorithm design paradigms and specific problems with algorithms that solve them.

For each paradigm, please be able to do the following

1. Give an example of a problem and an algorithm from this paradigm (e.g., a greedy algorithm for min. spanning. tree, divide-and-conquer algorithm for sorting)
2. Know, roughly, the type of problems solvable within it (e.g., dynamic programming and greedy algorithms can sometimes approximate and sometimes solve restricted versions of optimization problems)
3. Know the basic steps of designing an algorithm within this paradigm (for greedy: sort, then go through elements without retracing to previously considered elements; for dynamic programming create an array, etc.)

For the applications, please be able to name and describe algorithms solving a specific problem. Give the time complexity of the algorithms in O-notation, and mention, if relevant, data structures being used. For example, Kruskal's algorithm for min. spanning tree uses Union-Find data structure, and with it can run in time $O(m \log n)$.

Make sure you know how to solve every problem on assignments 1 and 2.

List of topics

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**
   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 $\Theta(n \log n)$).
3. **Basic data structures**
   Arrays, lists, priority queue based on heap data structure, and union-find data structure.
4. **Graph algorithms**
   Directed unweighted graphs: DFS, BFS, topological sort, strongly connected components, testing bipartiteness.
   Undirected weighted graphs: Minimal spanning trees: greedy algorithms by Kruskal and Prim.
   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). Floyd-Warshall all-pairs shortest path (dynamic programming).
5. **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. Know examples of both exact and approximation greedy algorithms. List of problems we considered: interval scheduling, scheduling with deadlines and profits, 1/2 approximating Knapsack, Dijkstra's algorithm, Huffman code, Kruskal's and Prim's algorithms for min. spanning. tree.
6. **String matching**
   Know Rabin-Karp algorithm and Knuth-Morris-Pratt. Which of them would you use for matching multiple patterns? Which is related to finite automata?
7. **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!). Know when it gives a polynomial time solution and when it does not. Applications: interval scheduling, all-pairs shortest path, scheduling with deadlines, profits, and durations, longest common/increasing subsequence, sequence alignment (from bioinformatics lectures), etc. Give an example of a situation when a dynamic programming algorithm does not run in polynomial time. What would be another approach in this case? (backtracking).