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, for network flows represent the problem as a weighted graph, 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 all problems 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 (e.g., matching with indifferences, with more than one element on one side matched to the other side, etc).

2. **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, Huffman code, Kruskal and Prim algorithms for min. spanning. tree.

3. **Compression**
   Know the general idea how LZ77 works and how gzip uses Huffman codes to store dictionaries for LZ77.

4. **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?

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!). 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, general change making, sequence alignment (from bioinformatics lectures), etc.

6. **Bellman-Ford**
   Know the algorithm, understand why it works, running time, etc. Use the standard version that can detect negative cycles.

7. **Network flows.**
   Since the assignment is due after the test, I will only ask general questions about the setting. 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.

8. **Divide-and-conquer**
   Know the paradigm (split the problem in several pieces, solve each piece recursively, combine results). Be able to give examples (E.g., mergesort, finding a median, multiplying integers). Know the closed form for recurrences $T(n) = aT(n/b) + n^c$, at least for $b = 2$; look it up under "master theorem" (just know the three cases).

9. **FFT**
   Know what Discrete Fourier Transform is, and that polynomial coefficients form a convolution of the original coefficient vectors. Know the general idea of doing polynomial multiplication using FFT. Note: please read it in CLRS or elsewhere, rather than KT.

10. **Limits of tractability**
    Know what NP-complete and NP-hard is (NP-hard: any language in NP reduces to it. Can be optimization problem. NP-complete: a language in NP which is NP-hard. Has to be a language. E.g. MaxIndependentSet (G) is NP-hard; IndependentSet(G,k) is NP-complete).

    Methods of dealing with intractability: restrict the problem (e.g., Ind.Set on trees); consider approximations (e.g., 1/2 approx for knapsack), parameterize, use randomness (no proof that it actually helps, but can't disprove yet either; e.g. polynomial identity testing). Use local search heuristics (simulated annealing, gradient descent; genetic algorithms).