ECS122A Final Review Checklist

Here are a list of concepts, definitions, algorithms, design and analysis techniques that you should know from the class, including homeworks and midterms. This is not meant to be comprehensive. It is merely a reminder of what we need to review for the upcoming final exam.

I. Algorithms, design and analysis techniques, examples

1. Divide and Conquer
   (a) 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
   (b) 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

2. Greedy Algorithms
   (a) 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.
   (b) Examples
   - Activity selection problems
   - Huffman coding
   - Minimum spanning tree (MST)
   (c) Counterexamples
   - knapsack problem
   - Money changing
3. Dynamic Programming

(a) 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.

(b) Examples:

- Rod cutting
- Matrix-chain multiplication
- Longest common subsequence/substring
- Edit distance
- Knapsack problem
- Change-making problem

4. Graph algorithms

(a) Elementary graph algorithms

- Breadth-first search (BFS)
- Depth-first search (DFS)
- Applications
  - Topological sort of a DAG
  - Finding a sink
  - Finding connected components of a undirected graph
  - Detecting a cycle

(b) Minimum spanning tree

- Prim’s algorithm
- Kruskal’s algorithm

(c) Shortest paths (from single-source)

- Bellman-Ford algorithm
- Dijkstra’s algorithm
- DAG

II. NP-completeness

1. Tractable and intractable problems

2. Optimization problem and decision problem

3. Polynomial reduction/transformation

4. Formal definitions of P, NP, NP-complete and NP-hard

5. 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.
III. Definitions, concepts and data structures

1. Growth of functions and asymptotic notations: $O(f(n)), \Omega(f(n)), \Theta(f(n))$

2. Linear recurrence relations

3. Divide-and-conquer recurrence relations

4. The master method/theorem for solving divide-and-conquer recurrence relations

5. Graph, path, connected graph, connected component, cycle, acyclic, tree, spanning tree, ...

6. Graph representations: adjacency list, adjacency matrix, incidence matrix.

7. Data structures:
   - FIFO queue and LIFO stack – enqueue, dequeue
   - Priority queue – Insert(S,u), Extract-Min(S), Decrease-Key(S,u,k), ...
   - Disjoint-set – Make-set(u), Union(u,v), Find-set(u)