6.006 Review Session Topics

Session 1:

1. **Asymptotic Notation**
   - Big O, Big $\Omega$, $\Theta$, Little $\omega$, Little $o$

2. **Divide and Conquer**
   - Definition: create subproblems of same type and recursively run method
   - Runtime and recursive equation
   - 1D peak finding
   - 2D peak finding

3. **Master Theorem**
   - $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
   - What are the three different cases of the Master Theorem

4. **Sorting**
   - Insertion sort: $O(n^2)$
   - Merge sort: $O(n \log n)$; how it works, what is the recursion, how you obtain the given runtime
   - Comparison sorting lower bound
   - What is stable sorting
   - What is in-place sorting
   - Counting sort: what kind of sort, how it works, runtime
   - Radix sort

5. **Heaps**
   - Definition of a heap and why it serves as a Priority Queue
   - Heap representation in computer memory as an array and visual representation as a tree.
   - Heap operations that are $O(\log n)$
     - Insert(node $x$)
     - Min_heapify(node $x$)
     - Delete_min()
   - Find_min $O(\log n)$ if min is removed. Otherwise, $O(1)$
   - Build_heap(Array $A$): $O(n)$
   - Heap sort: how it works, $O(n \log n)$

6. **Trees**
   - Binary search trees: two child nodes, left node is $\leq$ parent node, right node is $\geq$ parent node
   - Components of a node
   - BST operations: $O(n)$ why?
     - Find($x$)
     - Insert($x$)
     - Delete($x$)
     - Successor($x$)
   - $O(n)$ operations introduce need for balanced binary search trees
• AVL trees
  o BST augmentation with property child heights differ by at most 1
  o Derive $O(\log n)$ upper bound on height
  o AVL rotations to maintain AVL property

7. **Hashing**
• What is hashing by chaining?
• What is open addressing?
• Cuckoo hashing
• Uniform hashing assumption (UHA) definition
• Simple uniform hashing assumption (SUHA) definition (note: different from uniform hashing assumption)
• Runtime of operations for all three (hashing with chaining, open addressing, cuckoo hashing)—average runtime (insert, delete, search...etc.)
• Hash functions (hashing with chaining)
• Probing methods (open addressing)
• Universal hashing
• Table doubling: algorithm, amortized analysis, amortized cost for insert and delete
• Rolling hash (Rabin-Karp): algorithm (hash function, sliding window concept, resolve collisions by comparing actual elements), runtime
• Open addressing: how deletions are handled, Insert(key), Delete(key), Search(key) functions and runtime
• Uniform hashing: what does it mean for a probe sequence?
• Different types of probing: linear (primary clustering), double hashing
• Runtime analysis of unsuccessful search, insertion and successful search

8. **Numerics**
• Newton’s method: describe the method, graphical representation, derive recursive formula
• Multiplication of large numbers: $O(n^2)$ naive method; Karatsuba’s method—better runtime
• High precision division

9. **Graphs**
• $G = (V, E)$—directed and undirected graphs
• Graph is defined by set of nodes and edges
• Data structure representation of graphs
  o Adjacency list
  o Adjacency matrix

Session 2:

1. **Graphs (unweighted):**
• Breadth first search (BFS):
  o Explain the algorithm
  o Pseudocode
  o Runtime: $O(V + E)$
  o May be used to find shortest path (unweighted graph)
• Depth first search (DFS):
  o Explain the algorithm
  o Pseudocode
  o Compare to BFS
  o What can it be used for? Connected components, cycles, topsort...etc.
• Topological sort: use DFS with finish times and then get the appropriate ordering $O(V + E)$
• Graph transformations

2. **Graphs (shortest paths)**
• Single source shortest paths
  o Edges now have different weights
  o Define shortest paths
  o Optimal substructure of shortest paths
  o Triangle inequality
• Bellman-Ford:
  o Explain algorithm
  o Pseudocode
  o Intuition of correctness
  o Runtime: $O(VE)$
  o Works with negative edges (no negative cycles)
  o Detects negative cycles
• Dijkstra’s algorithm
  o Explain algorithm
  o Pseudocode
  o Intuition of correctness
  o Runtime depends on data structure used: $O(V \times \text{extract min} + E \times \text{decrease key})$
  o Modify problem/graph and apply Dijkstra’s: how much time per modification? How much time does Dijkstra’s take on new graph?

3. **Dynamic Programming (DP)**
• Optimal substructure: The optimal solution can be produced by combining solutions of sub-problems, which are recursively produced by combining solutions of sub-sub-problems.
• Overlapping sub-problems: The total number of sub-problems arising recursively is polynomial. (Recursion + Memoization)
• Using a graph to represent a DP problem and finding shortest paths
• Review steps to take in solving a DP problem: (# subproblems, guess, time/guess...etc)
• Review all DP problems presented in lecture
• Knapsack problem: definition, structure, DP solution, pseudopolynomial runtime
• DP on trees: examples from lecture
• How do you calculate runtime from DP?

4. Complexity
• Halting problem (unsolvable in finite time)
• P, EXP, R definitions
• NP (non-deterministic polynomial): Given a candidate solution to the problem we can easily verify it (i.e. in polynomial time); NP does NOT mean “not polynomial”
• Most decision problems are uncomputable
• NP-complete problems
  o Problems in NP
  o Problems that are NP-hard (prove this with an appropriate reduction)
• Reductions:
  o If we reduce problem A to a problem B (A --> B), then B is at least as hard as A.
  o The reduction has to take at most polynomial time, otherwise there is no point in reducing problem A to B.
  o Given the above reduction, if B takes polynomial time, then A can be solved in polynomial time, too.
  o If A is NP-hard, then B is NP-hard, too.
  o These properties follows immediately from the definition of a reduction