CS161 - Course Review

David Kauchak

- Sorting algorithms
  - Algorithms
    - Mergesort
    - Insertion sort
    - Selection sort
    - Bubblesort
    - Quicksort (randomized quicksort)
    - Heapsort
  - algorithm features (in place, stable, etc)ps2
  - how they operate
  - runtimes

  Runtime, how they operate
- Big O
  - O, Θ, Ω
  - Proving bounds
  - Proving bounds don't hold
  - Comparing functions (rules of thumb)
- Recurrences
  - Recursion-tree method
  - Substitution method
  - Master method
- Search trees

  - Binary search trees
  - B-trees

  Operations and runtime

- Data structures
  - Linked lists
  - Stacks
  - Queues
  - Binary heaps
  - Priority queues

  Operations, how they operate and runtime

- Hashtables
  - Hash functions
  - Collision by chaining
  - Open addressing
    * Linear probing
    * Quadratic probing
    * Double hashing

  Operations, how they operate, collision analysis, runtime

- Graph algorithms
  - Types of graphs (directed, undirected, weighted, trees, dags, complete, bipartite, ...)
  - Representations (adjacency list, adjacency matrix)
  - BFS
  - DFS
  - Topological sort
  - Single source shortest paths
    * Dijkstra - positive weights
    * Bellman-Ford - general graphs

  - Minimum spanning trees
    * Cut property
    * Kruskal's algorithm
    * Prim's algorithm

  Besides knowing the algorithms, you should also be able to problem solve generic graph problems.

- Greedy algorithms

  - What makes for a greedy problem/solution?
  - Identifying greedy problems
  - proving greedy algorithms are optimal
  - Examples
    * Interval scheduling
    * horn formulas
    * Fractional knapsack problem
    * Huffman coding algorithm

  How to recognize greedy algorithms? Solving greedy problems

- Dynamic programming

  - identifying DP solutions
  - Defining optimal solution wrt solution to optimal subproblems
  - Bottom-up approach
  - Examples
    * Fibonacci
    * Counting binary search trees
    * LCS
  - Memoization

  identifying and solving DP problems

- String algorithms

  - Edit distance

  - String operations
  - String matching
    * Naive approach
    * FSA (building an FSA from a pattern, FSA operations)
    * Rabin-Karp algorithm

  how they operate. runtimes

- Linear programming problems
  - Definition of an LP
  - Normal form
  - Graphical representation of constraints
  - Creating LPs from word problems
  - Solving LPs
    * What makes a program solvable
    * Simplex method (finding neighbors, determining if we're at the max)
    * Graphically solving
    * runtime analysis

- Amortized analysis

  java.util.ArrayList: Supports standard array operations, but does not have a fixed size.

  - set - $\Theta(1)$
  - get - $\Theta(1)$
  - add - $\Theta(1)$???

  The underlying implementation is an array of some initial capacity. As long as the number of elements is less than the initial capacity, all operations performed as expected. However, when the initial capacity is met and another element is added, a new array is generated with a larger initial capacity (e.g. double the size) and the elements from the original array are copied over.

What is worst case cost of the add operation? $O(n)$

In **amortized** analysis we analyze the average performance of each operation in the worst case.

Consider $n + 1$ insertions into an array of capacity n. What is the average cost of an insertion?

$n + 1$ adds

n copies

$2n$ - allocating memory (depending on the implementation)

$4n+1$ overall operations for $n+1$ insertions $= (4n+1)/(n+1) = 4 = \Theta(1)$

- Interview questions (how far have we come...)
  - Describe the algorithm for a depth-first graph traversal.
  - Write a function f(a, b) which takes two character string arguments and returns a string containing only the characters found in both strings in the order of a. Write a version which is order N-squared and one which is order N.
  - You're given an array containing both positive and negative integers and required to find the sub-array with the largest sum (O(N) a la KBL). Write a routine in C for the above.
  - Reverse a linked list
  - Insert in a sorted list
  - Write a function to find the depth of a binary tree