Reminders

- Exams will be returned next Tuesday
- Professor LaBerge will be guest speaker Thursday
- No office hours on Thursday
- Homework **will** still be assigned on Thursday

CMSC 203: Lecture 11

Algorithms and Complexity

Algorithms

- An **algorithm** is a procedure that follows a (finite) sequence of steps that solves a problem
  - Also includes mathematical / computational model
- Searching, sorting, min, max, transpose, etc.
- Algorithms may be written in **psuedocode**
  - Mix of English and traditional programming

Properties of Algorithms

- **Input**: Input must be a set
- **Output**: Input must *map* to a set
- **Definiteness**: Steps of algorithms must be precise
- **Correctness**: Should produce correct input → output
- **Finiteness**: Should output in finite steps
- **Effectiveness**: Each step must be finite time
- **Generality**: Applicable for all problems of desired form

Searching Algorithms

- Finding element in ordered list
  - Given element *x* in list *A*, return index of element or 0 if element is not in the list
- Linear search
  - Compares *x* to *A* element-by-element
- Binary search
  - Works if monotonically increasing
  - Based on subdividing search space

Sorting Algorithms

- Ordering elements of a list
  - Monotonically increasing / decreasing
- Many different kinds of sorts
  - Each solves a distinct problem
  - There is no such thing as a free lunch
- Bubble sort
  - Compare elements next to each other and swap
- Insertion sort
  - “Inserts” each element into correct sorted order

Greedy Algorithms

- Used to solve *optimization* problems
  - Minimize / maximize some parameter
- Greedy algorithms take the “best” choice at each step
- Can prove algorithm is optimal, or show nonoptimal counter example
  - Generally proven using proof by contradiction

Halting Problem

- Given a computer program as input, determine if the program will eventually stop
- Proof by contradiction shows it is impossible
  - By Alan Turing
  - Also created the definition of a Turing Machine
- This proves that not every problem may be programmed
- Also proves that finding an infinite loop is NP-Hard

Complexity

- We want to know how *efficient* an algorithm is
  - Number of steps algorithm takes and how long each step takes
  - Amount of memory used by each algorithm
- Two forms of analysis
  - **Empirical analysis**: Actual runtime
  - **Asymptotic analysis**: “Growth Function”

Asymptotic Analysis

- Estimate for how many operations are performed
- Can't do run-time or steps in an algorithm
  - Differences in hardware and software
- We want to compare algorithms while ignoring constant multipliers and smaller order terms

Big-O

- **Big-O notation** is used to make these estimates
  - Interested in algorithm as input size increases
  - Compare efficiency of two algorithms
- Provides an “upper-bound” for the algorithm
- Given two functions f(x) and g(x), we say that f(x) is O(g(x)) if: $\exists c,k$ such that $|f(x)| \leq c|g(x)|$ for all $x > k$
  - g(x) is an upper bound for f(x) for all $x > k$
- Proven using constructive proof