Lecture 1: Algorithms and Computation

Admin

- Course information handout
- Websites (Stellar, ALG, Gradescope, Piazza)
- Email `6.006-questions@mit.edu`
- Grading {PS: 25%, Q1: 20%, Q2: 20%, F: 34%, R: 1%}
- Recitations MW, please confirm or adjust by Fri. 9/14
- Prerequisites

6.042 (Discrete Mathematics)	6.0001 (Introduction to Python)
Sets, Sequences, Relations	Language Syntax
Sums, Counting	Basic Features
Proofs	Basic Recursion
Number Theory (Modular arithmetic)	
Probability	
Graphs	

Computation

- **Problem**
  - Binary relation from problem inputs to correct outputs
  - Usually don't specify every correct output for all inputs
  - Provide a verifiable predicate (a property) the output must satisfy
  - Bounded inputs
    - In this room, is there a pair of students with same birthday?
  - Unbounded inputs (generalization, arbitrarily large)
    - For any set of students, is there a pair of students with same birthday?
    - Size of input: often n, but not always!
- **Algorithm**
  - Procedure mapping inputs to **single** outputs (for a deterministic algorithm)
  - Solves a problem if returns correct output for any problem input

  - Algorithm should have constant size
  - Must prove correctness
    - For **bounded** inputs, can use case analysis to prove
    - For **unbounded** inputs, must prove by induction
    - Algorithm must be recursive or loop in some way (Recurrences next week)

- **Efficiency**
  - Produces a correct output in **polynomial time** with respect to input size n
  - Sometimes no efficient algorithm exists for a problem
  - Asymptotic Notation
    - Upper bounds (O), lower bounds (Ω), tight bounds (Θ)
    - $\in, =$, is, order
    - Particles in universe estimated $< 10^{100}$

input	constant	logarithmic	linear	log-linear	quadratic	polynomial	exponential
n	$\Theta(1)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n^c)$	$2^{\Theta(n^c)}$
1000	1	≈ 10	1000	$\approx$ 10,000	1,000,000		$2^{1000} \approx 10^{301}$

- **Model of Computation (Word-RAM)**
  - Memory: Bit storage of 0 or 1
  - Word: chunk of w bits for some fixed w (word size)
  - Treat memory as a **random access array** of words
    - Stores data in a sequence of integer-labeled, equally-sized chunks
    - Supports read, write from any bucket in $O(1)$ time (random access)
  - **Data Structure:** Way of storing data supporting set of operations
  - Computer has constant # word registers to perform operations
  - Model operations: (constant time)
    - access: read, write (other inputs/outputs)
    - bitwise: and, or, negate, shift
    - arithmetic: add, subtract, mod, compare, ...
  - Assumption: $w \geq$ # bits to represent largest memory address $(\log n)$
  - Memory address must be able to access every place in memory
  - 32-bit words $\rightarrow$ max 4 GB memory
  - 64-bit words $\rightarrow$ max 10^{10} GB of memory
  - **Python** is another model of computation implemented on a Word-RAM

How to Solve an Algorithms Problem

- Reduce to a problem you already know (use data structure or algorithm)

Search Data Structures	Sort Algorithms	Shortest Path Algorithms
Array	Insertion Sort	Breadth First Search
Sorted Array	Selection Sort	Depth First Search/Topo. Sort
Linked List	Merge Sort	Dijkstra
Dynamic Array	Heap Sort	Bellman-Ford
Binary Heap	AVL Sort	Floyd-Warshall
Binary Search Tree / AVL	Counting Sort	Johnson
Direct-Access Array	Radix Sort	
Hash Table		

- Design your own (recursive) algorithm
  - Model function calls as nodes in a graph, directed edge from $A \to B$ if A calls B
  - Classify algorithms based on (1) shape of dependency graph and (2) nodes visited

Class	Graph	Visited
Brute Force	Star	All
Decrease & Conquer	Chain	All
Divide & Conquer	Tree	All
Dynamic Programming	DAG	All
Greedy / Incremental	DAG	Some

  - Analysis requires proof by induction (Greedy requires additional proof)