CSE 373: Data Structures and Algorithms

Lecture 24: Course Victory Lap

Instructor: Lilian de Greef
Quarter: Summer 2017

Announcements

- Final Exam on Friday
  - Will start at 10:50, will end promptly at 11:50 (even if you're late) *so be early*
  - Anything we've covered is fair game
  - Only bring pencils and erasers
  - Turn off / silence and put away any devices (e.g. phone) before exam
- Section
  - Will go over solutions for select problems from practice set
  - Practice set posted on course webpage (under Sections)
  - Recommendation: do the practice problems, then use section to go over the questions you found hardest (there isn't enough time to cover all of them)
  - You're welcome to go to both sections if you want!

A Plug for Course Evaluations

- I'd super appreciate it!

- Why did I do things that worked well for you in this class?
  - Because students in the past gave feedback to instructors
  - The instructors could then act on it and pass that information on (e.g. to me)
  - You can think of evals as a way to pay forward the improvements enabled by past students
- Comments/ideas for improvements?
  - Help future students of 373!
  - Help future students of Lilian!
- Liked it?
  - Help me land a future job in teaching, should I choose it pursue it as my carrier
  - And/or nudge and encourage me to do so
- Important to have as many student as possible fill it out: avoid sample bias

(link closes tomorrow, so do it today!)

>>>>>>>>>>>> https://uw.iasystem.org/survey/179902 <<<<<<<<<<<<<<

Victory Lap!

A celebratory lap around the track or field by the victors (that’s us)!

Wow, we covered a *lot*!

Wowza! Blimey! Holy mackerel! Hot dang!

- **Abstract Data Types (ADTs)** and **Data Structures**
- **Stacks** and **Queues**
  - Linked list implementation
  - Array implementations (including circular arrays)
- **Asymptotic Analysis**
  - Big-O of code snippets
  - Inductive Proofs
  - Recurrence Relations (and when to apply them)
  - Formal definition of Big-O
  - Big-O and -Omega, Theta, little-o and -omega
  - Amortized Analysis
- **Dictionary ADT**
- **Hash Tables**
  - Hash functions, hash values, and indexing
  - insert, find, remove
  - Collisions
  - Separate chaining
  - Open addressing / probing
  - Linear probing
  - Quadratic probing
  - Double hashing
  - Rehashing
- **Generic trees**
  - Terminology
- **Binary trees**
  - Terminology
  - Representation
  - Calculating the height
  - Traversals
- **Binary Search Tree (BST)**
  - find
  - insert
  - delete (3 cases)
  - buildTree
  - Terminology (e.g. successor, predeccessor)
  - Balanced vs unbalanced trees
- **AVL Trees**
  - Balance conditions
  - AVL balance condition
  - Rotations
  - insert (4 cases)
- **Priority Queue ADT**
- **Heaps**
  - insert & delete
  - Percolations
  - Array representation/implementation
  - buildTree (client version and Floyd's Method /heapify)
  - d-heaps
- **For each data structure**
  - Ways to implement
  - Pros, Cons, and other reasons to choose one over the other

Graphs

- General knowledge & terminology
  - Mathematical representation (G = (V, E), etc.)
  - Undirected & Directed Graphs
  - Self Edges
  - Weights
  - Paths
  - Cycles
  - Connectedness
  - Trees as graphs
  - DAGs
  - Density & Sparsity
- Graph data structures
  - Adjacency Matrix
  - Adjacency List
  - When to use which and why

Graph algorithms

- Topological Sort
  - What it is
  - Necessary conditions
  - Two algorithms for topological sort
- Traversals
  - Depth First Search (DFS)
  - Breadth First Search (BFS)
  - When to use which
- Shortest path
  - For unweighted graphs
  - For weighted graphs (Dijkstra's algorithm)
  - Two approaches to Dijkstra's, when to use which
- Spanning Trees
  - Approach #1: DFS
  - Approach #2: Add acyclic edges
- Minimum Spanning Tree (MST)
  - Prim's Algorithm
  - Kruskal's Algorithm

Sorting Algorithms

- Terminology
  - Stable sort
  - In-place sort
  - External sort
- Comparison Sort
  - Insertion Sort
  - Selection Sort
  - Heapsort (including in-place version)
  - Merge Sort (including time- & space-saving versions)
  - Quicksort (including different pivot rules)
  - Using cutoffs
- Other Sorts
  - Conditions that let you use them
  - Bucket Sort (a.k.a. Bin Sort)
  - Radix Sort
- How to sort massive data
  - What algorithms make the most sense and why
  - How to sort
- For each algorithm:
  - Worst- best-case scenarios & run times
  - Other pro's/con's of each
  - When to use which

General Algorithms Knowledge

- Analyzing algorithms
  - Correctness (less emphasis here)
  - Efficiency
- Several algorithm types
  - Greedy algorithms
  - Dynamic programming
  - Divide-and-conquer
- P vs NP

Software Design: Preserving Abstractions

- Abstraction (what it is, why it's important)
- Memory representation (call stack, heap space, program counter, etc.)
- Aliasing and mutations, how they're problematic
- Copy-in
- Copy-out
- Immutability (e.g. using the 'final' keyword)
- Deep copies & deep immutability (and why)

Parallelism

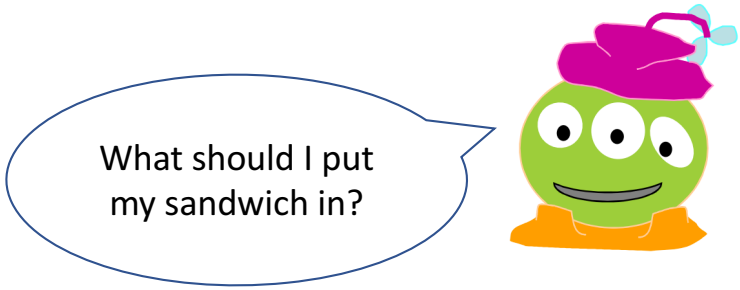
- Terminology
- Parallelism vs Concurrency
- Shared memory & race conditions
- Threads / Fork-join programming
  - How to use in Java (subclass, create 'thread' object, start(), join())
  - What happens under the hood
- Divide-and-conquer approach and why
- Map & Reduce
- Analysis (including Amdahl's Law)

Design decisions

- Ability to ask questions about problem to inform solution
- How to analyze/justify a decision
  - Time efficiency
  - Space efficiency
  - How parallelizable (in a few cases)
- Fluency with data structures & algorithms concepts/knowledge
  - Purposes a data structure is well-suited for and why
    - Available operations
    - Efficiency of basic operations
    - Space usage (conceptually)
  - Pro's and con's of different algorithms

Copy-pasted Lecture 1 slide!

What is a Data Structure?



- On super high level: a container for data
- Real-world examples of containers:








Copy-pasted Lecture 1 slide!

The crux of this course

- Understanding your data structures and algorithms to choose the right one for the job.
- Fundamental CS skill
- After this course, I want you to be able to
  - Make good design choices
  - Justify and communicate design decisions

Tool to aid us: Asymptotic Analysis

- For & while loops
- Recursive Methods
- Formal definition of worst-case
- Average Case

Stack and Queue ADTs

Dictionary ADT

Priority Queue ADT

Graphs

Graph Algorithms

- DFS
- BFS
- Dijkstra's
- Spanning Trees
- MSTs
  - Prim's
  - Kruskal's

Sorting Algorithms

- Insertion
- Selection
- Heap
- Merge
- Quick
- Bucket/Bin
- Radix

Types of Algorithms

- Greedy
- Dynamic Programming
- Divide-and-Conquer
- P and NP classes of algorithms

Other things

- Coding Style
- Preserving Abstractions
- Parallelism

Where next?

At UW, lots of upper-division CSE courses available!
(https://www.cs.washington.edu/prospective_students/undergrad/admissions/nonmajor)

- CSE 154: Web Programming
  Developing Websites and client and server side software
- CSE 374: Intermediate Programming Concepts and Tools
  Concepts of lower-level programming (C/C++) and explicit memory management
- CSE 415: Introduction to Artificial Intelligence
  Knowledge representation, logical and probabilistic reasoning, learning, language understanding, intro to game theory
- CSE 417: Algorithms and Computation Complexity
  NP Complete problems, undecidable problems, graph theory and complexity
- … and more!

Tons of resources outside UW, like **free classes**!

- Coursera (https://www.coursera.org/browse/computer-science)

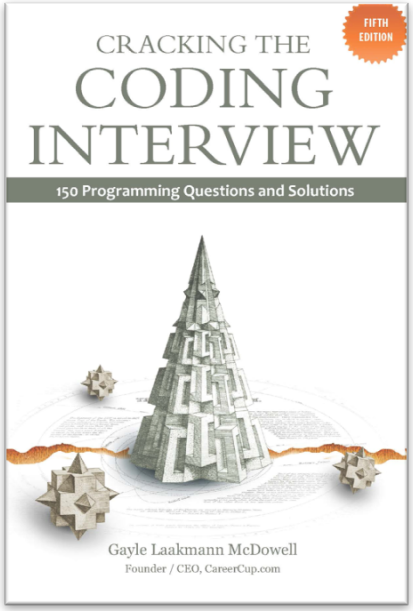
  - Machine Learning
  - Mobile / Web / Game Development
  - Data Science
  - Cybersecurity
  - Networks
  - R Programming
  - Neural Networks
  - Interaction Design
  - Python
  - More theory (algorithms, principles, etc.)
  - Computational Neuroscience
  - … and more!

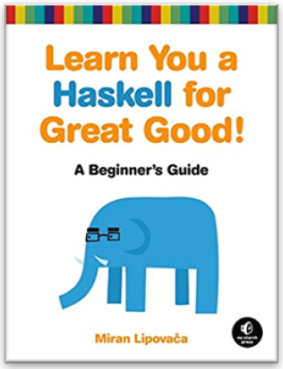
- Codecademy (https://www.codecademy.com/)

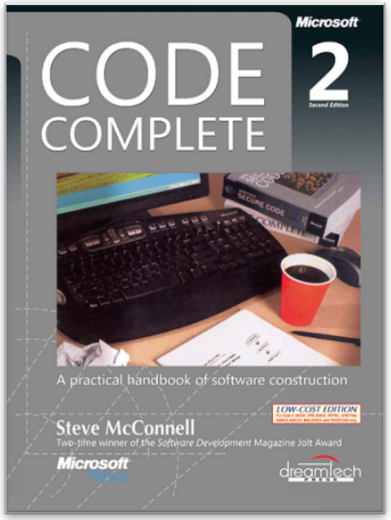
  - HTML & CSS
  - Making websites
  - SQL
  - Git
  - JavaScript
  - Python
  - Ruby
  - … and more!

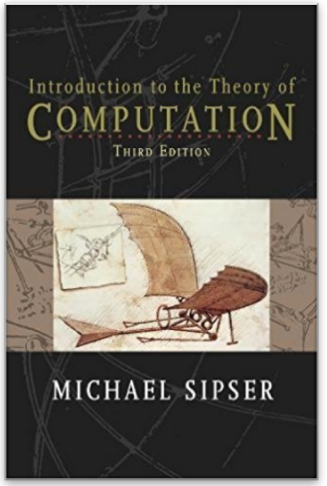
- Interactive, game-like way to learn Git with visuals: http://learngitbranching.js.org/

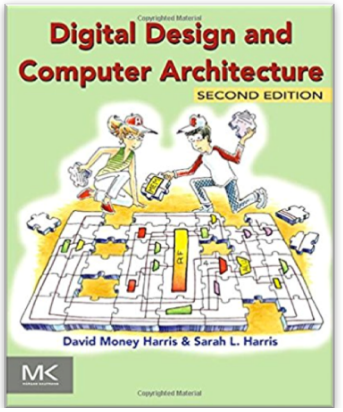
Books! So many books!











Learn a new language!

- **Python:** https://www.learnpython.org/
- **Haskell**: http://learnyouahaskell.com/chapters
- **C++**: http://www.learncpp.com/
- **Scala**: http://www.scala-lang.org/documentation/
- **Ruby**: https://www.codecademy.com/learn/ruby
- **PHP**: https://www.codecademy.com/learn/php
- **Racket**: https://learnxinyminutes.com/docs/racket/

There are resources of 100's of languages online. Pick one and mess with it!

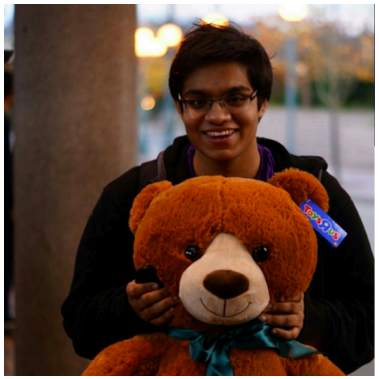
Learn to code games!

- Using **Unity**: https://www.udemy.com/unitycourse/
- Using **ActionScript**: https://www.siteground.com/tutorials/actionscript/
- Make an **Android App** (using mostly Java): http://developer.android.com/training/basics/firstapp/index.html

... and so much more!

- Create cool/useful things with code
  - And even post/maintain it on GitHub for others to see/contribute
- Fork peoples projects on GitHub and read their code
- Contribute to open source projects
- Participate in a hackathon
- Create an account on StackOverflow
  - Ask and answer questions!
- Learn how to write scripts to automate things you don’t like spending time on!

Thank you, TAs!



A. What is helping you learn in this class?

Office Hours

Accessible Teacher and TAs during office hours [90 %]

- "Lots of office hours!" (G1)
- "Office hours are awesome!" (G2)
- "Homework, a lot of office hours" (G3)
- "Section and office hours" (G8)
- "Lots of office hours, really accessible piazza -> lots of resources" (G10)

Thank you, students!

For

- Participating in class (Questions! Answers! Follow-up questions!)
  - Takes willingness and courage!
- Participating in polls and disucssions
- Attending section and office hours
  - For the staff, that makes it worth our while to put in the effort ☺
- Occasionally laughing at my jokes
  - or groaning or head-shaking or at least putting up with them
- Attendance (for a summer class especially!)
- Putting effort into learning the material
- Great attitude!

Question 1:

Given a list of integers, find the highest value obtainable by concatenating them together.

For example: given [9, 918, 917], result = 9918917

For example: given [1, 112, 113], result = 1131121

Given a list of integers, find the highest value obtainable by concatenating them together.

For example: given [9, 918, 917], result = 9918917

For example: given [1, 112, 113], result = 1131121

Question 2:

Given a very large file of integers (more than you can store in memory), return a list of the largest 100 numbers in the file

Given a very large file of integers (more than you can store in memory), return a list of the largest 100 numbers in the file

Question 3:

Given an unsorted array of values, find the 2nd biggest value in the array.

(Harder alternative: Find the k'th biggest value in the array)

Given an unsorted array of values, find the 2nd biggest value in the array.

Question 4:

Given a list of strings, write a method that returns the frequency of the word with the highest frequency.

(Harder version)

Given a list of strings, write a method that returns a sorted list of words based on frequency

Given a list of strings, write a method that returns the frequency of the word with the highest frequency.

Question 5:

Your task is to store a directory of employees who work at a company. Important operations include the ability to add an employee to the directory, to determine whether someone works at the company (based on name), and be able to print all of the employees in alphabetical order. What data structure would you use and why?

Question 6:

You have recipes that each have a list of ingredients and instructions. Although most recipes do an okay job of listing ingredients in the same order as the instructions use them, they don’t always and often have mistakes. How would you fix their ordering?

Question 7:

You later decided that you'd rather have the ingredients listed by what kind of measuring spoons they use (so you can measure everything and changing spoons as little as possible) but otherwise keep the ordering the same. Conveniently, each ingredient also lists a quantity and the correct measuring spoon size. How would you re-order the ingredients?

Example input:

(1, half-tsp, salt),
(1, cup, sugar),
(2, Tbsp, vanilla),
(1, cup, butter),
(2, cup, flour),
(1, tsp, baking powder),
(3, Tbsp, egg)

Output would have order of:

half-tsp, tsp, Tbsp, cup

You later decided that you'd rather have the ingredients listed by what kind of measuring spoons they use (so you can measure everything and changing spoons as little as possible) but otherwise keep the ordering the same. Conveniently, each ingredient also lists a quantity and the correct measuring spoon size. How would you re-order the ingredients?

Example input:

(1, half-tsp, salt),

(1, cup, sugar),

(2, Tbsp, vanilla),

(1, cup, butter),

(2, cup, flour),

(1, tsp, baking powder),

(3, Tbsp, egg)

Output would have order of:

half-tsp, tsp, Tbsp, cup