CS211, Lecture 28
Course Summary

Announcements:

- end of regular consulting: Fri, 5/1
- special hours to finish regrades, pick up work will be announced on the website
- consulting: forms in 303 Upson, starting 1PM today
- final exam info: Final Exam (on website) (review info, prior exam posted soon)
- final exam conflicts? see website; due Friday 5/2!

Overview:

- final exam review
- summary of what we did cover
- summary of what we didn't cover
- where to go from here…

1. Final Exam Review

1.1 Concepts

1.2 Topics

2. The Summary

2.1 The Nature of Programming

- programming: automating problem solving
- OOP: modeling of information and action find the nouns! find the verbs!
- OOP fundamentals
- OOP style
  - information hiding
  - abstraction
  - generic programming
- overall theme of CS211: improve your OOP style

2.2 Advanced Math

- discrete math
  - logic
  - some set theory
  - summations
- induction
  - base case(s)
  - inductive hypothesis
  - inductive step
  - conclusion

2.3 Advanced Flow Control

- flow control to design action
- how?
  - iteration/looping
  - recursion
- which is better?

2.4 Advanced OOP

- inheritance
  - inheritance to extend classes to ease reuse
  - inheritance to "tweak" functionality
- subtyping and polymorphism
  - interfaces to force programmers to be consistent
  - interfaces to hide details of implementation to develop better code
  - interfaces to provide a mechanism to supply different implementations but not change code

- polymorphism and upcasting vs downcasting
  - upcast always legal: supertype var gets subtype obj
  - downcast depends: subtype var get supertype obj requires a cast!
- compile time vs run time
- static binding vs dynamic binding
  - static binding: compile time assignments, like lookup of method names -- Java looks at variable type
  - dynamic binding -- during run time Java looks at object's actual type
- inner classes
  - member level
  - statement level
  - expression level (anon classes)

2.5 Advanced I/O

- namely, GUIs: want to improve interface for user
- event driven programming: user actions trigger actions
- GUI statics
  - components
  - containers
  - layout managers
- GUI dynamics
  - events
  - event listeners

2.6 Generic Programming

- Inheritance: extend classes, use interfaces, absrtact classes
- API
  - Collections
  - JFC
  - util, lang, …
- **`Object`**
  - **`toString`**, **`hashcode`**, **`equals`**, **`clone`**, …
- Cloning: alias, shallow, deep
- Iterators
  - array
  - list
  - more?
- **`Comparable`**
- more? Comparators, …

2.7 Algorithm Analysis

- need way to quantify choice in d.s.
  - style
  - ease
  - suitability
  - time/space
- techniques:
  - best/worst/average case
  - big Oh (different kinds of "o")
- big-oh (aymptotic notation)
  - measure aspect of program as function of input size n
  - gives upper (perhaps extreme upper) bound estimate
  - determine T (or S) as function of n
  - drop constants and lower order terms or just count the dominant operation
  - more formal: f(n)=O(g(n)) if f(n)<=cg(n) for some c0>0 and there's an n >= n0
  - to prove, must find (c0,n0); to disprove, must show contridiction

2.8 Sorting and Searching

- searching:
  - chaotic/random O(maybe forever)
  - linear: O(n), but easy to rem!
  - binary: O(log n), but need sorted array
- sorting:
  - select sort O(n^2), but easy to rem!
  - merge sort O(nlog n), but creates extra space
  - quick sort O(nlog n), could be O(n^2), but no extra arrays to create

2.9 Abstract Data Type

- information
- set of operations
- only a specification!

2.10 Data Structure

- collection of information with operations…
- actually, the implementation of the ADT!

2.11 Foundational Data Structures

- variables
- strings
- arrays
- lists
  - different ways to build
  - head, tail, head&tail, next/prev links
- array-lists
  - “indexable” lists
  - “growable” arrays
- trees
  - hierarchical, like graphs but no cycles
  - binary trees

2.12 Search Structures

- Why?
  - want to find information
  - need quick/efficient search time
- Basic interface
  - search
  - insert
  - delete
- Linear
  - list: O(n)
  - sorted array: binary search: O(log n), but need to sort

- Hierarchical
  - binary search tree: O(log n), but could be O(n) (why?)
  - better binary search tree?
- hashing
  - key-value pairs (see exercise 3 sol for good example!)
  - convert key to hashcode for index into array
  - store k-v pairs in linked list buckets
  - collisions and space/time issues (load capacity)
  - O(1) for retrieval, but could be as bad as O(n)

2.13 Sequence Structures

- Why?
  - want structure with quick storage and retrieval time
  - could use for searching, but might give poor time
- Basic interface
  - put
  - get
- Linear
  - stack: LIFO (put puts last, get takes last); O(1)
  - queue: FIFO (put puts list, get takes 1st); O(1)
  - PQ: LIFO, but min/max priority reorders the Q

- Hierarchical
  - heap: sim to BST but ordering on each level
  - use heap for PQ
  - put, get: O(log n)
  - most efficient way to implement is with array
  - handy for PQ --> need to store PQ elements

2.14 Graphs

- pull things together! use search and sequence structures to do your bidding
- theory: G = {V, E}; V is set of nodes, E is set of edges
  - adjacent edges
  - adjacent nodes
- types
  - undirected vs directed
  - unweighted vs weighted

- building graphs
  - adjacency matrix
  - adjacency list (we did a lot with this)
- problems:
  - traversals: DFS vs BFS
  - SSSP: single source shortest path
    unweighted
    weighted: Dijkstra's Algorithm
  - minimum spanning trees: Prim's algorithm

3. What's Next?

3.1 What we didn't cover this time…

- other kinds of timing analysis (best, average)
- recurrence relations (running time of recursive algorithms)
- balancing binary search trees

3.2 Where to go from here?

- cover what we missed (see the textbooks…)
- implement in different languages (CS312…)
- learn about algorithms (CS482…)
- build wonderful software!