Course Wrap-up (!!)

CSC148, Introduction to Computer Science
Fall 2017
Diane Horton
Administrative Stuff

- We will announce on Piazza when A2 results are available.
- We will contact you individually about any outstanding remarks etc.
Office hours

• Help Centre hours continue through Friday:
  – 1-3: first-year only
  – 4-6: first and second year

• David and I have a new schedule starting tomorrow:
  – Thu 10-12: Diane, BA 1220
  – Fri 11-1: David, in the Help Centre
Preparing for the final

• Re-solve parts of the assignment that your partner did.
• Re-do / complete lab exercises.
• Run examples from class. Modify them and explore the impact.
• Test out your “what if” questions / theories.
• Make up your own problems.
• Focus on topics you aren’t fully confident in.
• Solve old tests and finals.
  – NB: Some use content or corners of Python we didn’t touch
The final

• Comprehensive (covers the whole term), including:
  – Class design
  – Inheritance
  – ADTs, stack, queue, priority queue
  – Trees, BSTs, linked lists
  – Recursion
  – Time-efficiency and big-oh analysis
  – Techniques for testing; choosing good test cases
  – “Design-by-contract concepts”; use of preconditions, assertions
The final

• As before, we provide an aid sheet. It’s posted.
• There is very little to memorize.
  – It’s about understanding, not regurgitation.
• Helpers are always welcome, unless we specifically say otherwise.
• Comments are not necessary, unless we specifically ask for them.
• We have posted the front page.
What have we learned?
Abstraction

- If a function provides a consistent interface
  - Client code can ignore implementation and think abstractly.
  - Implementation details change with no impact on client code.

- The same holds for classes and methods.

- ADTs take abstraction up a notch
  - They are above the level of any implementation or language.
  - We saw stack, queue, priority queue, collection.
  - A tree can be though of as an ADT. E.g., AbstractTree and its child classes represent hierarchical information.
  - Or it can be thought of as an implementation details. E.g., a BST can merely be a good implementation of collection.
Good design

• Inheritance can capture what classes share.
  – Specifies what any future descendant class must do.
  – Specifies what every descendant class has.
  – Allows client code to call methods on an object without knowing which kind it is

• Allows plug-out plug-in compatibility.

• Think hard about the interfaces to your classes.
  – If you have to change the interface, all client code must change.

• Much more on this in csc207, Software Design.
Common structures

• Some new data structures are in your repertoire:
  – Standard: Linked list, tree, binary search tree
  – Designed by us: Block

• There are many more:
  – AVL tree, red-black tree, trie, heap, hash table, ...

• More on this in csc263, Data Structures and Analysis.

• We also saw a new algorithm structure:
  – Greedy algorithms. This is a whole category of algorithms

• More on this in csc373, Algorithm Design, Analysis & Complexity
Design by Contract

• A way of thinking about code
  – Each piece assumes certain pre-conditions
  – And guarantees certain post-conditions

• Some algorithms require significant reasoning to invent / understand / assess for correctness.

• Assertions can help us express that reasoning.

• More in reasoning about correctness in csc236, Introduction to the Theory of Computation.
Run-time analysis

• Some algorithms are much faster than others.
• Some algorithms are so slow they are infeasible.
• Big-oh helps us express such things.
• More on this in csc236, Introduction to the Theory of Computation.
• Sometimes the problem itself is infeasible.
• More on that in csc438, Computability and Logic
Trade-offs

• Speed vs accuracy
  – E.g., our “smart” player vs finding the optimal solution

• Time vs space
  – E.g. from A2?

• Time vs time
  – E.g., sorted lists require slower insert and delete than unsorted lists, but have faster search.
  – E.g., balanced search trees require slower insert and delete than regular search trees, but have faster search.
Gaining confidence in our code

• Doctest examples are very important.
• But we need to design thorough test cases.
• For complex code, this is not easy.
• Tools like unittest help implement testing.
• Property-based testing lets us easily check more cases, but only check for properties.
• Assertions and proofs are valuable for very tricky algorithms.
Your software tools

• You’ve started to gain proficiency with
  – A professional IDE
  – A unit testing framework
  – The debugger

• This is part of what makes you a professional.

• More on this in csc207, Software Design.
Your thinking tools

• You’ve also added new tools for thinking about:
  – Data
  – Algorithms
  – Code design
  – Analysis of correctness
  – Analysis of algorithms

• Knowing when and how to use them is part of what makes you a computer scientist.

• The journey is just beginning!
Foundations outside CS

- **Calculus** is important, for example, for
  - Machine learning, computer graphics, computational finance

- **Linear algebra** is important, for example, for
  - Machine learning, computer vision, medical imaging

- **Statistics** is important, for example, for
  - Machine learning, data mining, computational finance, user-experience (UX) studies, CS education research
Foundations outside CS

• Embrace opportunities to strengthen skills in
  – communication
  – teamwork

• Employers value these greatly
Payoff in 3rd and 4th year

- csc320, 420 / 418: image processing / graphics
- csc321, 411: machine learning
- csc369, 469: operating systems
- csc343, 443: databases
- csc488: compilers
- csc436, 446, 456, 466: numerical computing
- csc404: computer games
- csc490+454@DCSIL: entrepreneurship
- Etc . . . .
Maximizing your experience

Research opportunities

– http://web.cs.toronto.edu/program/ugrad/research.htm
– For course credit: csc490, csc494/5
– For pay: summer NSERC “USRA” program
– Research seminars: web.cs.toronto.edu/news/events.htm

• Career Mentorship program
  – web.cs.toronto.edu/program/ugrad/mentor.htm

• CSSU events: cssu.cdf.toronto.edu

• Professional Experience Year (PEY): pey.utoronto.ca

• Read @DCS (emailed Fridays) + the BA screens

• #1 advice: Get to know your profs!
fin