CS4102, Algorithms, Spring 2010

Dr. Tom Horton

horton.uva@gmail.com

- Course Mechanics
- Homework issues to resolve
- Course content
  - Topics from earlier classes
  - CS4102 course learning objectives
- What's the course all about? A quick tour

CS 432, Algorithms: General Info

- See beginning of course memo for general information
  - It's a draft! We may alter HW information, exam dates, grade breakdown,…
- Pre-requisites:
  - CS 2150 is <u>absolutely</u> required (with C- or better)

CS 432, Algorithms: General Info

- Required Textbook:
  - *Algorithms*,
    by Richard Johnsonbaugh and Marcus Schaefer
  - On reserve in library
- Other references (also on reserve):
  - Your CS2150 textbook
  - The "old" textbook by Cormen et al
- Other readings may be assigned
  - Definitely some handouts
  - Possibly web articles, PDFs

Expectations

- For a given week/unit, I will post in advance:
  - What I expect you to remember/review from earlier
  - What I expect you to read <u>before</u> class sessions
  - A set of problems you ought to be able to solve at the end of that week/unit
- This will allow us to:
  - Have more interesting class sessions than just lecturing
  - A more successful educational experience (better grades)
- Let's all live up to our expectations (the high ones)!

Expectations:

- Of you:
  - When asked, prepare for things in advance
  - Participate in class activities, some out-of-class activities
  - Act mature, professional.
  - Plan ahead.
  - Don't take advantage. Follow the Honor Code. (See the BOCM.)
- Of me:
  - Be fair, open, and considerate.
  - Seek and listen to your feedback.
  - Not to waste your time.
  - Be effective in letting you know how you're doing

Exam Info:

- Three exams:
  - Exam 1, 22%. Thursday, Feb. 25
  - Exam 2, 22%. Tuesday, Apr. 13
  - Final exam, 22%.   May 6, 2-4pm.
  - Final is half on topics after April 13, and half on earlier topics.
- Issues?
  - Exam days: prefer Tuesdays to Thursdays?
  - Move exam(s) up to be a week earlier?

Other Class-work

- Homework (worth 34% of your grade):
  - Could be a combination of:
  - **Problem sets.** Traditional homework problems. Proofs, math, algorithms, etc.
  - **Programming-based assignments.** Possibly experiment oriented. Involving programming. Groups or pairs. Probably done in parallel to other problems sets, with longer deadlines. One or two or ?.

- Now, let's talk about this…

Topic for discussion: Homework

- The Big Question: How does homework help you learn better?
  - "Learn better" might mean doing better on exams.
- Possible student points of view:
  - Working through problems is better than just reading solutions.
  - Being required to turn it in makes me do it.
  - I do better on HW than exams so I need a HW score component to help my grade.
  - I learn better working through problems with others.
  - I prefer to work alone. (Perhaps: I don't like seeing others "slide by" when working groups or pairs.)

Class Activities using Think/Pair/Share

- Think/Pair/Share is a common “active/cooperative” learning technique:
  - Instructor poses a question or topic.
  - Individuals think alone about it for, say, two minutes.
  - People pair up and explain their thoughts to each other.
    - Communication helps one organize one’s thought.
    - Explaining to another helps both learn.
  - Instructor calls on some pairs to share their combined answers or ideas with the class.

Think/Pair/Share and then Discussion

- How does homework help me learn better?
    - **I'm willing to hear you input about how we should organize the problem-set part of the HW assignments.**
- **Issues:**
    - **Lots of required problems to turn in, or a few? How many problem sets?**
    - **Write programs, or not? Want more experience?**
    - **Do them alone, or in pairs? In groups? If >1, turn in individual answers?**
    - **Getting help or answers? From TA? From solutions?**
    - **Types of problems:**
      Like exam? Bigger/more challenging? Programs?
    - **Rules about late homework?**

Will Your Input Change Things?

- **I want your input, but I also really want:**
  - **You to learn more, better.**
  - **Distinguish student performance for grading.**
  - **Maintain some level of course standards.**
    - **I can't let you off easy.**
  - **Must make the course run smoothly**
    - **E.g. can't grade HWs if all of them turned in at the end of term!**

Homework: Your Thoughts and Ideas

- Like test problems
- one slip day
- at least one problem from each section/topic
- Book solutions (not always good) then have similar problems with solutions/process
- Large set of problems, you select some of them
- Optional problems in addition to required problems to cover important things
- Collaboration good, but option to work alone
- Assigning partners causes problems
- Working out solutions in class (watching process)

Homework: Your Thoughts and Ideas (2)

- Partnering and using time wisely, time shortage, working problems in class
- Schedule so you know in advance
- Turn in one set (maybe in class stuff too)
- Ack. Partners
- If typed, gets shared, so turn in one copy for group (graphs and such)
- CS so of course! But not labor/time intensive. Good to see mechanically works.
- But, can add overhead. Many are looking for practical view of algorithms into code.
- Typesetting: too time intensive! Needed for publication, but not for learning. Maybe one assignment – good to know!
- Electronic submission vs. paper

Homework: Your Thoughts and Ideas (2)

- Electronic submission vs. paper
- Yes – electronic submission!
- Drop a HW instead of slip days. (One for 5-6 HWs, maybe 2 for >)
- Have things due right before they're going to be graded
- Grace period, late penalty
- Slip days make more sense than a drop
- Concrete due dates good for many people.

What you know already from CS2150

- Definition of an algorithm
- Definition of algorithm “complexity”
- Measuring worst-case complexity
- Cost as a function of input size
- Asymptotic rate of growth: Big-Oh, Big-Theta
- Relative ordering of rates of growth
- Analyzing an algorithm's cost:
  - sequences, loops, if/else, functions, recursion
- Focus on counting one particular statement or operation; don’t count all statements

What you know already from CS2150 (2)

- Problems and their solutions:
  - Linear data structures vs. tree data structures
  - Searching: linear/sequential search, binary search (?), hashing
  - Sorting: quicksort in CS2110 (?), mergesort
  - Priority Queue ADT and Heap Implementation
  - Graphs: basic definitions, data structures
  - Shortest-path: undirected and directed
  - Depth-first and breadth-first search, topo. sorting
  - Minimum spanning trees: two algorithms
  - All-pairs shortest path (Floyd-Warshall)
  - Huffman Coding

What you know already from CS2150 (3)

- Examples of Algorithm design methods:
  - (?) Divide and Conquer (quicksort, mergesort)
  - Greedy (Shortest path, MST, Huffman coding)
  - Dynamic programming (fibonacci numbers, Floyd-Warshall)

What you know already from Discrete Math and CS3102…

- From CS2102:
  - Proofs: induction, contradiction
  - Counting, probability, combinatorics, permutations
  - Graphs,...

- From CS3102 (maybe)
  - Maturity in mathematics and computing theory
  - Ability to do proofs
  - Abstract models of computation

Um, but… Which of these are fuzziest?

- Huffman
- Counting, limits
- QS, MS
- MST
- Finding worst-case

Major Concepts in Our Course

- Topics list includes:
  - Basics of algorithm analysis and design.
    - Asymptotic growth rate. Lower bounds. Recurrence relations. Mathematical techniques.
  - Search (some) and graph searching.
    - Depth-first search, breadth-first search, exhaustive search
  - Divide and Conquer approach to algorithm design
  - Sorting, Selection, more on lower bounds
  - Greedy algorithms
    - Spanning trees, shortest paths, others
  - Dynamic programming
  - NP-complete problems
  - (Perhaps) Algorithms and intellectual property

Course Learning Objectives

At the end of the course, students will:

- Comprehend fundamental ideas in algorithm analysis, including:
  - time and space complexity; identifying and counting basic operations; order classes and asymptotic growth; lower bounds; optimal algorithms.
- Apply these fundamental ideas to analyze and evaluate important problems and algorithms in computing, including:
  - search, sorting, graph problems, and optimization problems.
- Apply appropriate mathematical techniques in evaluation and analysis, including:
  - limits, logarithms, exponents, summations, recurrence relations, lower-bounds proofs and other proofs.

At the end of the course, students will:

- Comprehend, apply and evaluate the use of algorithm design techniques such as:
  - divide and conquer, the greedy approach, dynamic programming, and exhaustive or brute-force solutions.
- Comprehend the fundamental ideas related to the problem classes NP and NP-complete, including:
  - their definitions, their theoretical implications, Cook's theorem, etc. Be exposed to the design of polynomial reductions used to prove membership in NP-complete.

OK… But What's It Really All About?

- Let's illustrate some ideas you'll see throughout the course
  - Using one example
- Concepts:
  - Describing an algorithm
  - Measuring algorithm efficiency
  - Families or types of problems
  - Algorithm design strategies
    - Alternative strategies
  - Lower bounds and optimal algorithms
  - Problems that seem very hard

Everyone Already Knows Many Algorithms!

- Worked retail? You know how to make change!
- Example:
  - My item costs $4.37. I give you a five dollar bill. What do you give me in change?
  - Answer: two quarters, a dime, three pennies
  - Why? How do we figure that out?

Making Change

- The problem:
  - Give back the right amount of change, and…
  - Return the fewest number of coins!
- Inputs: the dollar-amount to return
  - Also, the set of possible coins. (Do we have half-dollars? That affects the answer we give.)
- Output: a set of coins

- Note this problem statement is simply a transformation
  - Given input, generate output with certain properties
  - No statement about how to do it.
- Can you describe the algorithm you use?

A Change Algorithm

1. Consider the largest coin
2. How many go into the amount left?
3. Add that many of that coin to the output
4. Subtract the amount for those coins from the amount left to return
5. If the amount left is zero, done!
6. If not, consider next largest coin, and go back to Step 2

Code

```
def make_change(amt, coin_vals):
    val = amt
    i = 0
    coin_cts = [ ] # how many of each?
    while amt > 0:
        c = coin_vals[i]
        num = amt / c
        coin_cts.append(num) # add to list
        amt = amt - (num * c)
        i = i + 1
    return coin_cts
```

Is this a “good” algorithm?

- What makes an algorithm “good”?
  - Good time *complexity*. (Maybe space complexity.)
  - Better than any other algorithm
  - Easy to understand
- How could we measure how much work an algorithm does?
  - Code it and time it. Issues?
  - Count how many “instructions” it does before implementing it
  - Computer scientists count basic operations, and use a rough measure of this: order class, e.g. O(n lg n)

Evaluating Our Greedy Algorithm

- How much work does it do?
  - Say C is the amount of change, and N is the number of coins in our coin-set
  - Loop at most N times, and inside the loop we do:
    - A division
    - Add something to the output list
    - A subtraction, and a test
  - We say this is O(N), or linear in terms of the size of the coin-set
- Could we do better?
  - Is this an *optimal algorithm*?
  - We need to do a proof somehow to show this

You’re Being Greedy!

- This algorithm an example of a family of algorithms called *greedy algorithms*
- Suitable for optimization problems
  - There are many *feasible answers* that add up to the right amount, but one is *optimal* or best (fewest coins)
- Immediately greedy: at each step, choose what looks best now. No “look-ahead” into the future!

- What’s an optimization problem?
  - Some subset or combination of values satisfies problem constraints (feasible solutions)
  - But, a way of comparing these. One is best: the *optimal solution*

Does Greed Pay Off?

- Greedy algorithms often efficient.
- Are they always right? Always find the optimal answer?
  - For some problems.
  - Not for checkers or chess!
  - Always for coin-changing problem? Depends on coin values
    - Say we had a 11-cent coin
    - What happens if we need to return 15 cents?
  - So how do we know?
- In the real world:
  - Many optimization problems
  - Many good greedy solutions to some of these

Another Change Algorithm

- Give me another way to do this?
- Brute force:
  - Generate all possible combinations of coins that add up to the required amount
  - From these, choose the one with smallest number
- What would you say about this approach?
- There are other ways to solve this problem
  - *Dynamic programming:* build a table of solutions to small subproblems, work your way up

Some Problems Seem Very Hard

- Some problems we know seem hard (intractable)
  - We can't find good solutions
    - Our solutions work, but they're like the "brute force" method in terms of efficiency
  - But, we can't <u>prove</u> that it's <u>impossible</u> to solve this more quickly
    - Can't find good solution, can't say one doesn't exist
  - Do you know of any example problems like this?

- Families of problems: NP-hard and NP-complete
  - Some interesting mathematical properties
  - The Big Question in Computer Science: Does P = NP?

Expectations: Chapter 1 and 2:

- Chapter 1:
  - Read. Nature of algorithms, pseudo-code convention
- Chapter 2:
  - I'll lecture on Sections 2.3 and 2.4 next
  - Read all sections by end of the week.
  - Next class: In sections 2.1, 2.2, 2.5, and 2.6, tell me what topics issues are new, confusing, need review.
- Also, think back or review topics listed on slide "What you know already from CS2150" earlier

Course slide credits

- Textbook publisher makes some slides available for this book (ugh)
- I have slides from:
  - Earlier courses by Dave Luebke, Jim Cohoon
  - Slides originally for the Baase text created by:
    Dr. Ben Choi, Louisiana Tech University
- I'll modify these, of course…