EECS 3101 M: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/3101M`
Also on Moodle

Administrivia

- Lectures: Tue-Thurs 1:00-2:30 pm (HNE 038)
- Tests (35%): 3 tests, 15% each (worst test to be scaled to 5%)
- final (50%)
- Homework (15%)
- Office hours: Mon-Wed 3-4 pm or by appointment at LAS 3043

Textbook: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Introduction to Algorithms, 3rd edition, MIT Press and McGraw-Hill, 2009.

Homework, Grades

- We will be paperless, except for tests and final examination
- All course information will be online – Moodle and on the public course webpage
- All homework MUST be typed. You will get a zero if you submit handwritten solutions. You may use Office, Google Docs, LaTeX, or other packages but all submissions must be in pdf format.
- We will use crowdmark for grading. Follow instructions for re-appraisal requests.
- All returned work will be in electronic form (including tests).
- I would like to use iClicker
- Grades will be on moodle

Tutorials and Other Administrivia

- Tutorials (1.5 hours/week) are **mandatory**. These will be led by a TA. Most tutorials will be on problem solving
- Missed tests cannot be made up. If you have a valid medical reason, the weight will be transferred to the final
- If you have serious non-medical reasons (having work is not one), talk to me. We will deal with those on an ad hoc basis
- Plagiarism: Will be dealt with very strictly. Read the detailed policies on the webpage

Resources

- We will follow the textbook closely
- There are more resources than you can use – including books, lecture slides and notes, online texts, video lectures, assignments
- Problems in Algorithms by Parberry is downloadable at `https://larc.unt.edu/ian/books/free/poa.pdf`; This is an invaluable resource for testing your understanding
- Coding interview sites
- Jeff Edmonds' (`www.cse.yorku.ca/~jeff`) textbook has many, many worked examples
- Andy Mirzaian (`http://www.cse.yorku.ca/~andy`) has very good notes and slides for this course

The Big Picture

- The design and analysis of algorithms is a FOUNDATIONAL skill – needed in almost every field in Computer Science and Engineering.
- Programming and algorithm design go hand in hand.
- Coming up with a solution to a problem is not of much use, if you cannot argue that the solution is
  - Correct, and
  - Efficient

The Big Picture - 2

Previous courses (1012,1020,2030, maybe 2011):
Given a problem:

- Figure out an algorithm
- Code it, debug, test with "good" inputs
- Some idea of running time, asymptotic notation
- Study some well known algorithms: e.g. Binary Search, QuickSort, (maybe) Depth-first-search of graphs
- Possibly: some idea of lower bounds for it

Course Objectives

- Problem-solving: Design of algorithms – paradigms
  - Divide-and-Conquer
  - Greedy
  - Dynamic Programming
  - Graph Algorithms
  - Review some very simple data structures; e.g., Heaps
- Reasoning about ALGORITHMS
  - Correctness proofs: Loop invariants, induction
  - Efficiency analysis.
  - Comparison of algorithms
- Reasoning about PROBLEMS
  - Lower bounds. "Is your algorithm the best possible?"
  - Intractability: "The problem seems to be hard – is it provably intractable?"
  - Complexity classes "Are there inherently hard problems?"

Secondary Course Objectives

A new way of thinking – abstracting out the algorithmic problem(s):

- Extract the algorithmic problem and ignore the "irrelevant" details
- Focuses your thinking, more efficient problem solving
- Programming contest problems teach this skill more effectively than exercises in algorithms texts.

Role of Mathematics

- Needed for correctness proofs: Pre-condition – post-condition framework; similar ideas used in program verification, Computer-aided design.
- Needed for performance analysis: Computing running time
- Specific topics
  - (Very) elementary logic.
  - Simple proofs: Induction, proof by contradiction
  - (Rarely) Elementary calculus
  - Summation of series.
  - Simple counting techniques.
  - Elementary graph theory

My Expectations

- You will attend classes and tutorials regularly
- Want to solidify your algorithmic foundations
- Ask for help when needed
- Follow academic honesty regulations (see the class webpage for more details on policies).

To do well in this class

- Study with pen and paper
- Ask for help early
- Practice, practice, practice. The Parberry book on problems and coding interview sites are good sources for exercises
- Follow along in class rather than take notes
- Ask questions in class or outside class
- Keep up with the class
- Read the book, not just the slides
- Be timely

Examples

- Sorting a set of numbers (seen before)
- Finding shortest paths in weighted graphs (seen before?)
- Optimal matrix multiplication – compute $A_1 A_2 \ldots A_n$ using the fewest number of multiplications; e.g.:
  $A_1 = 20x30$, $A_2 = 30x60$, $A_3 = 60x40$,
  $((A_1 A_2)A_3) : 20x30x60 + 20x60x40 = 84000$
  $(A_1(A_2 A_3)) : 20x30x40 + 30x60x40 = 96000$
- Traveling Salesman Problem: Find the minimum weight cycle in an weighted undirected graph which visits each vertex exactly once and returns to the starting vertex
  Brute force: find all possible permutations of the vertices and compute cycle costs in each case. Find the maximum.

Q: This is exponential time. Can we do better?

Pseudocode

- Machine/language independent statements; similar to C/C++, Java, Python
- Very simple commands: assignment, equality tests, branch statements, for/while loops, function calls
- No objects/classes (usually)
- Comments, just like in real programs
- Should be at a level that can be translated into a program easily
- As precise as programs, without the syntax headaches (may contain lines of English)
- Not concerned with software engg issues like data abstraction, modularity and error handling

Pseudocode Conventions

- assignment: $=$, equality testing: $==$
- if, for, while uses indentations rather than begin-end or parentheses
- ∞ used as a symbol
- Array notations: $A.length$, $A[1\,..\,j-1]$.
- My notation can vary slightly from the book

You can use pseudocode, English or a combination

Pseudocode Example (page 18)

INSERTION-SORT(A)

```
1   for j = 2 to A.length
2       key = A[j]
3       // Insert A[j] into the sorted sequence A[1..j − 1].
4       i = j − 1
5       while i > 0 and A[i] > key
6           A[i + 1] = A[i]
7           i = i − 1
8       A[i + 1] = key
```

Next: Correctness of Algorithms

QUESTIONS?