Syllabus for CSCI 5454
Design and Analysis of Algorithms
Spring 2011

Lectures: Mondays and Wednesdays from 4:00–5:15pm in ECCR 1B51

Lecturer: Aaron Clauset

Office:	ECOT 743
Email:	aaron.clauset@colorado.edu
Web Page:	http://tuvalu.santafe.edu/~aaronc/courses/5454/
Office Hours:	Tuesday 1:30–3:00 or by appointment

Description: This graduate-level course will cover a selection of topics related to algorithm design and analysis. Topics will include divide and conquer algorithms, greedy algorithms, graph algorithms, algorithms for social networks, computational biology, optimization algorithms, randomized data structures and their analysis. We will not cover any of these topics exhaustively. Rather, the focus will be on algorithmic thinking, efficient solutions to practical problems and understanding algorithm performance. Advanced topics will cover a selection of modern algorithms, many of which come from real-world applications.

Prerequisites: CSCI 2270 (Computer Science 2: Data Structures), calculus and discrete mathematics, or equivalent. I assume you are already familiar with basic asymoptotic analysis (Big-O, etc.) and can solve recurrence relations. The ability to quickly and correctly implement basic algorithms is essential. (If you are not confident on this point, you should not take the course.) If you don't have the necessary background, please see me.

Required Text: *Introduction to Algorithms* (3rd ed.), by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (available at the CU Bookstore).

Coursework and grading: Attendance to the lectures is required.

The first half of the course will be lecture driven. The second half of the class will revolve around crowd-sourced (student) lectures on advanced topics. There will be problem sets throughout the semester. The lecture and accompanying lecture notes are a major deliverable for the class, and I expect students to spend a considerable amount of time outside of class preparing. There are no written examinations in this class; the lecture can be viewed as a kind of oral examination.

The five problem sets will each have some mathematical and some programming problems. Programming problems can be done in any reasonable imperative language (`C++`, `Java`, `Python`, etc.), and run-able source code must be submitted for each solution. Unless specifically allowed, all parts of all algorithms and data structures must be implemented from scratch (that is, no libraries; if you use `Python` or another modern language, be sure you are not accidentally invoking non-trivial libraries; garbage collection features are okay). For the mathematical problems, you should assume a `RAM` model of computation, unless otherwise specified.

Problem sets will be due three weeks after they are assigned. Solutions must be in PDF format (e.g., typeset using LaTeX), should include all necessary details for me to follow the logic (see below for advice about writing up your solutions), and should be submitted via email by 11:59pm the day they are due. No late assignments will be accepted. Collaboration is allowed on the problem sets, but you may not copy (in any way) from your collaborators and you must respect University academic policies at all times. That is, you may discuss the problems verbally, but you are should write up your solutions separately. If you do work with someone, you must list and describe the extent of your collaboration in your solutions (a footnote is fine). Copying from any source in any way, including the Web but especially from another student (past or present), is strictly forbidden. If you are unsure about whether something is permitted, please see me before the assignment is due.

For the crowd-sourced lectures (CSLs below), students will form teams of two and will give a full-length (75 minute) technical lecture that

- motivates the problem the algorithm solves,
- describes the algorithm in appropriate detail (at least at the pseudocode level),
- analyzes its performance in appropriate detail, and
- describes interesting extensions of the technique.

Each team will also submit their lecture notes, in PDF via email by 11:59pm the day they give their lecture. No late submissions will be accepted. Students should form their own groups by February 7th and email me, by February 14th, their team's preferences in the form of a complete ranking of all 11 topics (brief descriptions of which are given below). Assignments to topics will be made using a matching algorithm. (The number of topics and the precise schedule may change slightly during the semester depending on the number of students in the class.) The grade for the lecture will be determined both by the in-class presentation and the quality of the lecture notes.

Most lectures will have an accompanying reading assignment, typically taken from the required textbook. I'll post the details of the readings on the class website.

The course grade will be 20% attendance, 30% crowd-source lecture and 50% problem sets.

Tentative schedule:

Week 1	Overview, why algorithms?
Week 2	Divide & conquer
Week 3	Randomized data structures
Week 4	Greedy algorithms
Week 5	Graph algorithms
Week 6	Minimum spanning trees
Week 7	Network flow
Week 8	Phylogenetic trees
Week 9	Optimization
Week 10	CSL: (i) stable matchings and (ii) disjoint sets and union find
Week 11	Spring break
Week 12	CSL: (i) multiple sequence alignment and (ii) Byzantine agreement
Week 13	CSL: (i) linear programming/simplex algorithm and (ii) guest lecture
Week 14	CSL: (i) expectation-maximization and (ii) fair division algorithms
Week 15	CSL: (i) approximation algorithms and (ii) link prediction in social networks
Week 16	CSL: (i) Sudoku and latin square solvers and (ii) PageRank algorithm

Crowd-sourced lecture topics:

1. Stable matchings: Given a set of n resources and a set of n people, each of whose preferences over the resources are represented by a complete ordering (a ranking), how can we assign people to resources such that the assignment has certain nice properties, e.g., being *stable* with respect to pairwise swaps?

2. Disjoint sets and union find: Given a set of n items divided into $k \leq n$ disjoint sets, how can we organize and maintain these items in a way that allows us to quickly find some item x and merge pairs of sets? For instance, if we were interested in routing information on a communication network, the sets could represent pairwise reachability within a graph G (i.e., the graph's components) and merging a pair of sets implies connecting two components.

3. Multiple sequence alignment: Suppose we are given $k \geq 2$ strings of potentially different lengths, drawn from a single alphabet Σ, e.g., the sequences could be DNA and the alphabet $\{A, T, C, G\}$. How can we identify an *alignment* of the strings that maximizes the total subsequence overlaps, potentially minus some penalty for unaligned subsequences?

4. Byzantine agreement: Suppose we have n computationally constrained agents, connected to each other all-to-all, but some fraction c of which are corrupt. Communications between pairs of agents are private. How can we specify a communication protocol such that the $(1-c)n$ non-corrupt agents can reach consensus on the value of a single bit of information? Assume the corrupt agents are controlled by a malicious, computationally powerful adversary who knows the details of the communication protocol, is not bound to follow it and whose sole goal is to prevent consensus. For instance, the agents could be military commanders trying to decide whether or not to attack their enemy.

5. Linear programming and the simplex algorithm: Given a linear objective function, i.e., a set of linearly independent constraints such as inequalities that form a convex polytope in the solution space, how can we find a setting of variables $\mathbf{x}$ that optimizes the objective? A wide variety of problems can be framed in this way, e.g., multicommodity flow, microeconomic production functions, assignment of job candidates to jobs, etc.

6. Expectation-maximization: Given a statistical model that can be expressed as a *likelihood function* with parameters θ, some of which are hidden or latent, and observed data $\mathbf{x} = \{x_i\}$, how can we find a parameterization $\hat{\theta}$ that maximizes the likelihood of the data, given the model? The E-M algorithm can do this quickly and efficiently, and is widely used in machine learning.

7. Fair division algorithms: Given some resource, e.g., a "cake" or a set of household chores, and a set of k individuals, each of which prefers certain parts of the resource over others, how can we divide up the resource in a *fair* way, i.e., so that each individual believes they received a fair portion?

8. Approximation algorithms: Even though we may not be able to solve all instances of an NP-Hard problem in polynomial time, can we develop algorithms that yield *approximately* optimal solutions, i.e., solutions that are within some factor of the optimum, to these problems in polynomial time?

9. Link prediction in social networks: Given an existing social network G, composed of n nodes and m edges denoting "friendships", how can we accurately predict which friendships are missing? That is, can we make accurate recommendations to some node i of its true but not yet identified (latent or hidden) friendships based on the patterns of friendships we have already identified?

10. Sudoku and latin square solvers: The general Sudoku problem provides a board of $n^2 \times n^2$ cells arranged in n^2 blocks of $n \times n$ cells. The values within each of the

blocks, and along each row and column, must contain exactly the values $1 \ldots n$ with no duplicates. Given a partially specified instance, how can we efficiently fill in the unspecified cells such that we satisfy all the constraints?

11. PageRank algorithm: The classic web search algorithm. Given a set of webpages and hyperlinks between them, which we can represent as a directed graph G, how can we quickly and accurately return a set of relevant pages given a set of search terms?

Assignments & Deadlines:

assignment	assigned	due
Problem set 1	January 12	February 1
CSL, form lecture team		February 7
CSL, topic preferences		February 14
Problem set 2	February 2	February 22
Problem set 3	February 23	March 15
CSL, lecture notes		day of lecture
Problem set 4	March 16	April 5
Problem set 5	April 6	April 26

Advice for writing up your solutions:
Your solutions for the problem sets should have the following properties. I will be looking for these when I grade them:

1. **Clarity**: All of your work and answers should be clear and well separated from other problems. If I can't quickly identify and understand your solution, I can't evaluate it. I will not spend much time looking at any particular solution, so the easier and more clear you make your work, the most likely you are to get full credit.

2. **Completeness**: Full credit for all problems is based on both sufficient intermediate work (the lack of which often produces a "justify" comment) and the final answer. There are many ways of doing most problems, and I need to understand exactly how *you* chose to solve each problem. Here is a good rule of thumb for deciding how much detail is sufficient: if you were to present your solution to the class and everyone understood the steps, then you can assume it is sufficient.

3. **Succinctness**: The work and solutions that you submit should be long enough to convey exactly why the answer you get is correct, yet short enough to be easily digestible by someone with a basic knowledge of the material. If you find yourself doing

more than half a page of dense algebra, generating more than a dozen numeric values or using more than a page or two per problem, you're probably not being succinct. Clearly indicate your final answer (circle, box, underline, etc.). Note: It's usually best to rewrite your solution to a problem before you hand it in. If you do this, you'll find you can usually make the solution much more succinct.

4. **Numerical experiments**: Some programming problems will require you to conduct numerical experiments. For instance, to show that an algorithm takes $O(n \log n)$ time, you will need to measure the running time at multiple values of n, plot the measured running time versus those, and then plot the asymptotic function showing that the function matches the data. Plotting the average measured running time for a given value of n will often improve the results (by averaging over system-induced fluctuations). To get a good trend, I recommend using a dozen or so exponentially spaced values of n, e.g., $n = \{2^4, 2^5, 2^6, \ldots\}$. When presenting your results, you must label your axes and your data series, and explain your experimental design.

5. **Source code**: Your source code for all programming problems must be included at the end of your solutions. It should be appropriately commented so that I can understand what you are doing and why and it must be run-able – that is, if I try to compile and run it, it should work as advertised.

Suggestions: Suggestions for improvement are welcome at any time. Any concern about the course should be brought first to my attention. Further recourse is available through the office of the Department Chair or the Graduate Program Advisor, both accessible on the 7th floor of the Engineering Center Office Tower.

Special Accommodations: Requests for special accommodations (e.g., religious holidays, learning disabilities, special needs, etc.) should be brought to my attention within the first two weeks of class. I will do my best to accommodate all reasonable requests, per CU guidelines.

Honor Code: As members of the CU academic community, we are all bound by the CU Honor Code. I take the Honor Code very seriously, and I expect that you will, too. Any violations will result in a failing grade for the course and will be reported.