Syllabus for CSCI 5454
Design and Analysis of Algorithms
Spring 2013

Lectures: Mondays and Wednesdays from 1:00–2:15pm in ECCS 1B12 (and via CAETE)

Lecturer: Aaron Clauset

Office:	ECOT 743
Email:	aaron.clauset@colorado.edu
Course Web Page:	http://tuvalu.santafe.edu/~aaronc/courses/5454/
Office Hours:	Wednesdays 10:00–11:30am or by appointment

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

Prerequisites: Undergraduate algorithms (CSCI 3104), data structures (CSCI 2270), discrete mathematics (CSCI 2824) and two semesters of calculus, or equivalents. This class assumes familiarity with asymptotic analysis (Big-O, etc.), recurrence relations and the correct implementation of basic algorithms. Students without the required background may struggle to keep up with the lectures and assignments.

Required Text: *Introduction to Algorithms* (3rd ed.), by Cormen et al.

Overview:

- Problem sets (5 total) will be due every 3 weeks throughout the semester.
- The first half of the course will be lecture driven. The second half will revolve around (i) team lectures by students or (ii) an independent project (see below).
- The team lecture or independent project (see below) is the major deliverable for the class. I expect students to commit considerable outside time to their completion (> 20 hours). There are no formal exams; however, the lecture should be treated as a kind of oral examination and the project as a kind of written examination.

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) knapsack problems and (ii) multiple sequence alignment
Week 13	CSL: (i) randomized min-cut and (ii) Sudoku and latin square solvers
Week 14	CSL: (i) Byzantine agreement and (ii) fair division algorithms
Week 15	CSL: (i) NP-hard problems and (ii) approximation algorithms
Week 16	CSL: (i) link prediction in social networks and (ii) PageRank algorithm

CAETE students: An educational officer (EO) is required for the first day of class.

Assignments & Deadlines:

assignment	assigned	deadline
Problem set 1	Jan. 14 (Monday)	Feb. 4 (Monday)
CLS vs. IP lottery results		Jan. 28 (Monday)
CSL, submit team prefs.		Feb. 3 (Sunday)
IP, choose topic		Feb. 3 (Sunday)
Problem set 2	Feb. 5 (Tuesday)	Feb. 25 (Monday)
Problem set 3	Feb. 26 (Tuesday)	Mar. 18 (Monday)
CSL, lecture notes		day of lecture
Problem set 4	Mar. 19 (Tuesday)	Apr. 8 (Monday)
Problem set 5	Apr. 9 (Tuesday)	Apr. 29 (Monday)
IP write up		May 3 (Friday)

Course work and grading:

Grading

5454-002 (or by permission): attendance (0.2), lecture / project (0.3), problem sets (0.5).
5454-740: independent project (0.4), problem sets (0.6).

Problem sets

- There will be 5 problem sets; each will include mathematical and programming problems. Problem sets are due 3 weeks after they are assigned.
- Any reasonable imperative language (`C/C++`, `Java`, `Python`, etc.) may be used to complete the programming problems.

  Run-able source code must be included at the *end* of your solutions file. *Failure to submit your source code will result in no credit for the programming questions.*

  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 and static arrays are okay; "dictionary" data structures are not).
- Solutions to mathematical problems should assume a `RAM` computation model (unless otherwise specified).
- Your complete solution file must be submitted as a **single** PDF via email by 11:59pm the day they are due. *Late or improperly formatted solutions will receive no credit.* (I recommend using LaTeXto produce your solutions. Remember to include your source code *at the end* of the file.
- Your solutions must be detailed and clear. Explain in words how you set up your analysis and explain in detail why your solution is correct. (Advice for doing this can be found at the end of this document.)
- Figures and graphs must be labeled correctly. *Figures with unlabeled axes or data series will receive no credit.*
- 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. You may discuss the problems verbally, but you must write up your solutions separately.

If you discuss a problem with another student, you must list and describe the extent of your collaboration (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.

There will be a zero-tolerance policy to violations of this requirement. Violators will be removed from the class and given a failing grade.

- Some topics will only be covered through the problem sets.

Reading assignments: Most lectures will have an accompanying reading assignment, typically taken from the required textbook. I expect you to read these outside of class and come prepared to discuss the material.

Crowd-source lecture and *Independent project*

Due to the large size of the class, a maximum of 24 on-campus students will complete the Crowd-Source Lecture (CSL) assignment; all other on-campus students and all off-campus CAETE students will complete the Independent Project (IP) assignment.

For on-campus students, assignment to the CSL or IP will be done by lottery. I will announce the results of the lottery in class on Monday, January 28th.

- *Crowd-source lecture* (on campus only)

  CSL students will form teams of 2, choose a topic from the list below. This assignment consists of (i) an in-class presentation and (ii) a set of lecture notes.

  The in-class presentation consists of a 60 minute technical lecture that

  1. motivates the problem the algorithm solves,
  2. describes the algorithm in appropriate detail (at least at the pseudocode level) and gives appropriate (small) worked examples,
  3. proves its correctness and time/space usage, and
  4. describes interesting extensions, variations or applications.

  The last 15 minutes of the 75 minute class will be devoted to class discussion, which I will lead, with the presenters' help.

Lecture notes for the chosen topic must be submitted to me as a PDF via email by 11:59pm the day the lecture is given. No late submissions will be accepted. The lecture notes will be posted on the class website. Lecture notes must be fully referenced (include citations), and must include your name.

Students should form their own groups and email me, by 11:59pm, Sunday, February 3rd, their team's preferences in the form of a complete ranking of the topics listed below. I will match teams to topics using a matching algorithm. The joint grade will be determined both by the quality of the in-class presentation and the lecture notes.

Students wishing to do a practice run of the lecture with me should contact me at least one week in advance to arrange a time.

- *Independent project*

  Students will work independently to

  1. implement *from scratch* in `C/C++` or `Java` a non-trivial algorithm or data structure,
  2. give a detailed mathematical analysis of its correctness, space and time usage,
  3. identify and explain the inputs that result in worst-, average- and best-case performance,
  4. numerically characterize its worst- and average-case space and time usage,
  5. write up these results in a 10-page report, with figures and citations.

  A PDF of the writeup is due via email by 11:59pm on Friday, May 3rd (last day of class). No late submissions will be accepted. Students must email me, by 11:59pm, Sunday February 3rd, the project topic, which must be chosen from the list given below. The grade will be determined by the quality of the analysis, experimental results and overall written presentation. Original code must be submitted as an appendix (does not count toward 10 page limit). The presence of any code that is not original to the submitting student will result in a failing grade. All submitted code is subject to Googling.

  In numerically characterizing the space and time performance, you must implement an appropriate randomized input generator, describe it in your writeup, and use it to demonstrate that your implementation achieves the claimed asymptotic bounds on both space and time across input sizes that vary over several orders of magnitude.

  Your writeup should explain clearly the type of problems the algorithm solves, the idea behind the algorithm, your analytic results, and it should both describe and comment

on the results of the numerical tests. You should close with a brief discussion of extensions, improvements and recent work in its general area.

If you would like feedback about your project, please come to office hours.

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. (Note: must cover path compression.)

3. Knapsack problems: Suppose you are given a set of n item types where the ith item has weight w_i and value v_i. If our knapsack can support a maximum weight of W, which items should we take in order to maximize the value in the knapsack? (Note: must cover both fractional and 0-1 versions.)

4. 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?

5. Randomized min-cut: Finding the minimum-weight cut of a graph can be done using an algorithm, which we will learn in class, in $O(n^3)$ time. However, a randomized algorithm can beat this deterministic algorithm. How does this randomized algorithm work? How can guessing be better than knowing? (Note: must cover both the Karger algorithm and the Karger-Stein algorithm.)

6. 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? (Note: must cover backtracking.)

7. 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.

8. 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? (Note: must cover Sperner's Lemma.)

9. NP-hard problems: Most algorithms problems we cover will have polynomial-time solutions (i.e., are in the computational complex lass P). But for many important interesting and important problems, we don't know if a polytime solution exists. If we can verify that a solution exists in polynomial time, these problems are called NP-complete. How do we know a problem is in this class? What are examples of such problems? Why are they hard? (Note: must explain what is a reduction, and must show a reduction of Subset Sum to Vertex Cover.)

10. 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? (Note: must cover Vertex Cover.)

11. 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?

12. 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?

Independent project topics:

- Ford-Fulkerson max-flow algorithm (graphs)
- Back-tracking Sudoku solver (search)
- Bron-Kerbosch max-clique algorithm (search)
- Fortune's algorithm for Voronoi diagrams (computational geometry)
- Christofides' algorithm for Traveling Salesman Problem (approximation)
- AVL tree (data structures)
- Fibonacci heap (data structures)
- Algorithm of your choosing (requires instructor approval)

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 more clear you make your work, the more likely you are to get maximum 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 number of atomic operation at multiple values of n, plot the measured values versus n, and then plot the asymptotic function showing that the function matches the data. Plotting the *average* number of operations for a given value of n will almost always improve your results. To get a good trend, I recommend using a dozen or so exponentially spaced values of n, e.g., $n = \{2^4, 2^5, \ldots, 2^{16}, \ldots\}$. When presenting your results, you must 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.

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 significant violation will result in a failing grade for the course and will be reported. Here is the University's statement about the matter:

All students of the University of Colorado at Boulder are responsible for knowing and adhering to the academic integrity policy of this institution. Violations of this policy may include: cheating, plagiarism, aid of academic dishonesty, fabrication, lying, bribery, and threatening behavior. All incidents of academic misconduct shall be reported to the Honor Code Council (`honor@colorado.edu`; 303-735-2273). Students who are found to be in violation of the academic integrity policy will be subject to both academic sanctions from the faculty member and non-academic sanctions (including but not limited to university probation, suspension, or expulsion). Other information on the Honor Code can be found at `http://www.colorado.edu/policies/honor.html` and at
`http://www.colorado.edu/academics/honorcode/`

Special Accommodations: If you qualify for accommodations because of a disability, please submit to your professor a letter from Disability Services in a timely manner (for exam accommodations provide your letter at least one week prior to the exam) so that your needs

can be addressed. Disability Services determines accommodations based on documented disabilities. Contact Disability Services at 303-492-8671 or by e-mail at dsinfo@colorado.edu.

If you have a temporary medical condition or injury, see Temporary Injuries under Quick Links at Disability Services website and discuss your needs with your professor.

Campus policy regarding religious observances requires that faculty make every effort to deal reasonably and fairly with all students who, because of religious obligations, have conflicts with scheduled exams, assignments or required attendance. In this class, I will make reasonable efforts to accommodate such needs if you notify me of their specific nature by the end of the 3rd week of class. See full details at
http://www.colorado.edu/policies/fac_relig.html

Classroom Behavior: Students and faculty each have responsibility for maintaining an appropriate learning environment. Those who fail to adhere to such behavioral standards may be subject to discipline. Professional courtesy and sensitivity are especially important with respect to individuals and topics dealing with differences of race, color, culture, religion, creed, politics, veterans status, sexual orientation, gender, gender identity and gender expression, age, disability, and nationalities. Class rosters are provided to the instructor with the student's legal name. I will gladly honor your request to address you by an alternate name or gender pronoun. Please advise me of this preference early in the semester so that I may make appropriate changes to my records. See policies at
http://www.colorado.edu/policies/classbehavior.html and at
http://www.colorado.edu/studentaffairs/judicialaffairs/code.html#student_code

Discrimination and Harrassment: The University of Colorado at Boulder Discrimination and Harassment Policy and Procedures, the University of Colorado Sexual Harassment Policy and Procedures, and the University of Colorado Conflict of Interest in Cases of Amorous Relationships policy apply to all students, staff, and faculty. Any student, staff, or faculty member who believes s/he has been the subject of sexual harassment or discrimination or harassment based upon race, color, national origin, sex, age, disability, creed, religion, sexual orientation, or veteran status should contact the Office of Discrimination and Harassment (ODH) at 303-492-2127 or the Office of Student Conduct (OSC) at 303-492-5550. Information about the ODH, the above referenced policies, and the campus resources available to assist individuals regarding discrimination or harassment can be obtained at
http://www.colorado.edu/odh