CSE 202
Basic Information: Winter, 2014

Instructor: Russell Impagliazzo

email: russell@cs.ucsd.edu

webpage: www-cse.ucsd.edu/classes/wi14/cse202-a

Class:

TA: Eric Christiansen

Russell's CSE 202 Office Hours: M: 11-12 and 5-6. Th: 11-12. CSE 4248.

TA office hours: Tu,Th: 3:30-4:30, W: 2:30-3:30, CSE 4144

Prerequisites: We assume some undergraduate exposure to discrete mathematics, and to algorithms and their analysis, and the ability to read, recognize and write a valid proof. For example, CSE 20, CSE 21, and CSE 101 cover the prerequisite material. We will cover many of the same topics from an undergraduate algorithms courses. However, after a quick review of the basics, we will move on to related advanced material for each topic. If your background in these areas is weak, you may need to do some extra work to catch up. Please talk to me about this.

Text Book: Kleinberg and Tardos, Algorithm Design

Optional Supplementary Texts: Neapolitan and Naimipour, Foundations of Algorithms; Jeff Edmonds, How to Think About Algorithms A helpful text for proofs is Solow: How to Read and Do Proofs.

Assignments There will be five homework assignments, a project, and a take-home final exam. Each homework assignment will have four theoretical problems and one implementation problem. You should budget 10-20 hours for each homework assignment, including time spent in office hours, and 20-40 hours for the project. Homework and the project can be done in groups up to four. Most people stay with the same group for the quarter, but you are free to switch groups (or kick people out of your group if they aren't pulling their weight.) If you have trouble finding a group, or need another person for your group, send me email and I'll put an announcement on the class website.

The first homework assignment is on pre-requisite material and should be started immediately. It will be given two grades, a grade for "sincere effort" in which every problem attempted is given full credit (which will be the grade recorded), and a standard grade with the same grading criteria as all future assignments (which is only for your use.)

Evaluation: Homework will acount for 30 % of the grade, the project, 20%, and the final will account for the remaining 50 % of the grade.

Standards for evaluation: Most problems will be algorithm design problems, where you are given a computational problem and asked to design and analyze an efficient algorithm for the problem. Any solution to an algorithm design question MUST contain the following:

Problem statement A clear unambiguous statement of the problem to be solved.

Algorithm description A clear, unambiguous description of the algorithm. This can be in (well-documented, clear) pseudocode (with explanations in English) or in (precise, mathematical, well-defined) English. There should be no room for interpretation in the steps carried out by your algorithm. Any mathematically and computer literate individual should be able to follow the steps presented, or to implement the algorithm as a computer program. Such implementations by different people should still be essentially identical.

Correctness proof: A convincing mathematical argument that the algorithm described solves the computational problem described. Sometimes this can be brief, but it must always be given.

Time analysis: A time analysis of the algorithm, up to] order, in terms of all relevant parameters. You must prove this analysis is correct. Frequently, this will be brief, but occasionally the time analysis will be the heart of the question, and can be quite challenging.

What is a proof? A proof is a compelling argument that forces the reader to believe the result, not just notes from which a proof could be reconstructed. Note that I will mark off for any unsubstantiated and non-trivial claims in a proof, even correct claims.

Don't assume knowledge or sophistication on the part of the reader. A good rule of thumb is to pretend you are teaching an undergraduate class in algorithms, and write in a way that explains what is going on to the students in the class. Don't think of your reader as a trained algorithms professor who already knows the answer. I have to evaluate your answer based on WHAT YOU WRITE not WHAT I KNOW.

Your answer will be graded on the following criteria:

1. Your algorithm must be clearly and unambiguously described.
2. You need to prove your algorithm correctly solves the problem. My rule is: an algorithm is incorrect until PROVED correct. I will use this rule in grading even if I KNOW your algorithm is correct. If I think you don't KNOW why your algorithm works, I will grade it as if your algorithm does NOT work.

3. Your time analysis must be proved correct. A time analysis is usually an upper bound on worst-case time. You get credit for what you claim and prove about the running time of your algorithm, even if the algorithm is actually faster. Logic is as important as calculations in a time-analysis. At a minimum, a time analysis requires an explanation of where the calculations come from. If the analysis is "easy" (e.g., with a simple nested loop algorithm), these explanations can be brief (e.g., "The outside loop goes from 1 to n, and each iteration, the inside loop iterates m times, so the overall time is $O(nm)$."). Other times, the time analysis is a tricky, mathematical proof. If you give just calculations or just a short explanation, and I think the time bound is NOT easy and clear from what you wrote, you will lose points even if you give the correct time.

4. Your algorithm must be efficient. Although we sometimes use "polynomial-time" as a benchmark for "efficient", this isn't a hard and fast rule. An algorithm is efficient for a problem if there is no competing algorithm that is much faster. To be efficient might require being sub-linear time for applications where input sizes are huge, or almost linear time when the problem is trivially poly-time, or an improved exponential time algorithm for an NP-hard problem.

   If there is a faster algorithm, then you may not get full credit. Nothing you said is false, but I'll still deduct some points because you COULD have been cleverer. How can you avoid this? There's no guaranteed way, but if the correct algorithm seems easy, you should still try to think of improvements or other approaches. Is this fair? Not really, but it is the facts of life. Your algorithm won't be used if there is a faster equivalent algorithm.

   A good algorithm designer considers a variety of approaches and picks the best one, rather than prematurely converging to the first adequate algorithm.

There will also be some implementation problems on the homework. For these problems, you should present an outline of the basic algorithm you used, a discussion of implementation problems or where you didn't use the fastest asymptotic algorithm and why, results from the challenge instances described in the problems, and timing information about various problem instances. I WILL NOT READ ACTUAL CODE SO DON'T BOTHER HANDING IT IN. If the actual times seemed different from the asymptotic analysis, give a short discussion of possible reasons. Give the programming language used and your computer's speed rating so that I can normalize. Your grade will be based on your answer's completeness, and your success and time for challenge problems.

One handy tool is to graph performance on a log-log scale, the log of

the problem size on one axis and the log of the time taken on the other. This allows you to compare running times on a variety of scales, and to show the results in a comprehensible way. More importantly, it converts polynomials to lines, with the slope of the line revealing the exponent of the polynomial.

Project The project for the class is to implement an algorithm for an application that you are interested in. Submit a two to five page summary, describing the algorithm, the application, issues that arose in implementation, and data comparing benchmark performance (time taken, quality of solutions found, or other measurement) to the theoretical analysis or to competing approaches. Discuss the suitability of the algorithm to that application. Is it what you would use? Why or why not?

Academic Honesty **Golden rule of Academic Honesty** The point of all research is to get people to share ideas. In order to properly reward people for sharing ideas, it is necessary to give them credit when they do so. So the golden rule is: Whenever you use someone's ideas, you MUST give them credit.

Working in groups Students will be allowed to solve and write up all homework assignments and the project in groups of size from 2 to 5. This needs to be a *collective* effort on the part of the group, not a *division of labor*. Students are responsible for the correctness and the honesty of all problems handed in with their names attached. If a group member did not participate in discussion for one of the problems handed in by the group, the group must write a note on the front page of the solution set to that effect. However, verifying someone else's solution counts as a contribution.

The point of working in groups is:

Practicing communication skills: A major part of algorithm design and analysis is clearly communicating what your algorithm is and why it works. You need to practice with your group explaining algorithms to others and verifying claims about algorithms.

Brainstorming: Your group needs to come up with a variety of approaches to problems. Working by yourself, it is too easy to get stuck on one approach. Other people are often better at catching our mistakes, too.

Preparing for later careers: Whether in academia, research lab, government or industry, almost all work is collective, not individual. Learning to work as part of a team is essential.

Reducing our workloads: This is a big class. Fjola and I would not be able to properly grade individual assignments in a timely way. So hand in one assignment per group.

Other sources: Students should not look for answers to homework problems in other texts or on the internet. Yes, it is interesting to find the best known solutions. **However, do the literature search after you hand in your own solution.** Students may use other texts as a general study tool, and may accidentally see solutions to homework problems. In this case, the student should write up the final solution without consulting the text, and should give an acknowledgement of the text on the first page of their solutions. Such a solution may be given partial or no credit if it too closely follows the text. This includes all material found on the web (except on this year's class webpage), discussion with others who are either students or not (except the instructor or TA, or other students as part of office hours), or written notes from others, whether students or not, (except class notes).

Be sure to follow the following guidelines:

1. Do not discuss problems with people outside your group whether students or not (except the TA and me, of course).
2. Do not share written solutions or partial solutions with other groups.
3. Prepare your final written solution without consulting any written material except class notes and the class text.
4. Acknowledge all supplementary texts or other sources that had solutions to homework problems. Acknowledge anyone who helped with assignments, except the TA and myself.
5. Do not discuss the final exam with anyone except the instructor and TA.

Remember the Golden Rule: if you use someone's ideas (whether that is an anonymous Wikipedia editor or your spouse), you MUST give them credit. The exceptions are for standard ideas that everyone knows and the people whose job it is to give you ideas (Eric, myself, the textbooks).

Lateness Policy Late homework will be accepted until I give out an answer key and no later. So you have to be no later than me.

Reading Schedule When we cover a topic in class, I will not always use the textbook examples, or follow the textbook descriptions. The point is not that you shouldn't read the textbook, but to give you MORE examples and viewpoints (the ones in class AND the textbook's). You are expected to be reading the relevant sections of the textbook as or before we cover them in class. Much of the prerequisite background will never be explicitly covered in class, but if you aren't already familar wtih it, you are expected to teach yourself from the textbook (or undergraduate textbooks in Algorithms).

To help you plan, I will give a tentative schedule of general topics to be covered in class, and the corresponding sections of the text to be read: Our milage may vary as we actually cover the topics in class.

Background: Chapters 1-3. Also, recurrence relations from 5.1, 5.2. The callibration homework is on background, so if you are having trouble with understanding that, you need to spend more time on background. That includes exercises.

Introduction 1/2 lecture.

Graph search Chapter 3. Breadth-first and depth-first search, applications. Dijkstra's algorithm. Incorporating data structures and preprocessing into algorithms. Reductions and how to use them. Analogies and why they need to be reproved. 2 1/2 lectures.

Greedy algorithms Chapter 4. Some examples: Interval scheduling (4.1), Minimum Spanning Tree(4.5); (4.4), greedy approximation algorithms (11.1-11.3). Methods for proving greedy algorithms optimal. Analogs for proving approximation ratios. Continuing discussion of optimizing data structures for algorithms, amortized analysis. 3 lectures.

Divide-and-Conquer : Chapter 5. Mergesort (5.1, in passing), Integer Multiplication (5.5), Closest Pair of points (5.4). Median Finding and selection (13.5). Recurrences, the Master Theorem, and what to do when the Master Theorem doesn't apply. 2 lectures.

Search and Optimization problems, and the class NP See section 8.3 (1/2 lecture)

Back-tracking, Depth-first Search, Breadth-first Search, Branch-and-Bound Not in text, but see Chapter 10. Example problems: Maximum Independent set (p. 16), Graph Coloring (8.7), Hamiltonian cycle(8.5)addition chains (not in text). (2 lectures).

Dynamic Programming: Chapter 6. Weighted interval scheduling (6.1), edit distance (6.6), All-pairs shortest paths (6.8); Subset sum (6.5) and DP-based approximation algorithms (11.8). 3 lectures.

Network Flows and Hill-climbing: Network flows and the Ford-Fulkerson algorithm. (Chapter 7.1-7.3). Hill-climbing as an algorithmic technique (not in text, but see Chapter 1, stable marriage, and Chapter 12, local search) . Metropolis and simulated annealing. (2 lectures).

Linear programming: Linear programming, duality, simplex algorithm, ellipsoid algorithms, approximation algorithms using LP. (2 lectures) Note: may be condensed if time runs short.

Reductions and NP-completeness (Chapter 8.2, 8.4-8.7). (1 lecture)

Coping with intractibility: Heuristics, average-case analysis, and approximations (Chapter 11, 12). 1 lecture.

Assignment Schedule To help you plan, here is a tentative list of when assignments will be due:

1. Tuesday, January 14: Homework 1 (background material) due.
2. Tuesday, January 28: Homework 2 (graph search, data structures, reductions) due.
3. Tuesday, February 11: Homework 3 (greedy algorithms, approximation algorithms, divide-and-conquer) due.
4. Tuesday, February 25: Homework 4 (back-tracking, dynammic programming) due.
5. Thursday, March 6: Project due.
6. Tuesday, March 11: Homework 5 (network flow, linear programming, reductions revisited) due.
7. Thursday, March 13: Final exam available.
8. Wednesday, March 18: Final exam due.