CSE 202

Basic Information: Fall, 2001

Instructor: Russell Impagliazzo

email: russell@cs.ucsd.edu

webpage: www-cse.ucsd.edu/classes/fa01/cse202

Class: Peterson 104 Tuesdays and Thursdays, 9:35-10:55;

Prof. Office Hours: Monday, Wednesday, 2:30-4, 4111 AP&M, or by appointment (subject to change)

TA: TBA

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. 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. A helpful text for proofs is Solow: How to Read and Do Proofs.

Text Book: Corman, Liesserson Rivest and Stein, An Introduction to Algorithms (2nd ed.) I think first edition (no Stein) will be fine; but you'll need to find the corresponding page numbers for problems.

Optional Supplementary Texts: Neapolitan and Naimipour, Foundations of Algorithms; Jeff Edmonds, Thinking About Algorithms Abstractly, www.yorku.ca/j̃eff/notes

Assignments There will be one "callibration" homework, three 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.

Evaluation: Homework will acount for 30 % of the grade, the project, 20%, and the final will account for the remaining 50 % of the grade. The callibration homework is optional and does not count towards the grade.

Standards for evaluation: Most problems will be theoretical in nature. Answers to these problems must include, implicitly or explicitly, a proof of all claims. Grading of all problems (homework and exam) will be both on the basis of correctness and on logical consistency and completeness, i.e., "mathematical style". It is your obligation to provide 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 mentally address your answer to a bright undergraduate just starting an algorithms course; 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.

More particularly, many questions will be of the form , "Provide an efficient algorithm for the following computational problem". Your answer will be graded on the following criteria:

1. Your algorithm must be clearly and unambiguously described. 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.

2. Your algorithm needs to solve the problem given. To certify this, you need to give a correctness proof for your algorithm. 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. In some cases, correctness is easy or trivial; in this case, your correctness argument can be a short English explanation of Other times, correctness is highly non-trivial and requires a medium-sized mathematical argument. It's your job to distinguish these two cases.
3. Your algorithm must be efficient. This means, usually, polynomial-time, but it could mean sub-linear time in the case of data-base operations, or almost linear time when the problem is trivially poly-time. Again, unless your answer includes a well-reasoned time analysis, the presumption is that it is NOT poly-time. 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 relatively efficient. That means that, even if your algorithm is correct and reasonably fast, you may lose points if there is a FASTER algorithm. 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.

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.

Project The project for the class is to find a well-specified algorithmic problem that is important for some computer science area, preferably in your future research area or some other area you are interested in and knowledgable about. You should 1. define the problem rigorously, 2. present and analyze algorithms in the literature to solve the problem, and 3. discuss how well the theoretical analysis captures the real-life performance of these algorithms. Your project should have at least one non-trivial mathematical proof in it (correctness proof, approximation ratio, non-trivial time analysis), and at least a page of non-mathematical discussion of the application. If you have trouble thinking of a topic, talk to me (and we can have trouble thinking of a topic together.) You may work in groups up to 4, but a team project should be somewhat longer, and everyone in the group must write up part of the project. Individual projects should be 5 (word-processed) pages or so. Tentative due date: Nov. 15.

Academic Honesty Students will be allowed to solve and write up all homework assignments and the project in groups of size up to 4. All names should appear on the assignment. 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.

Students should not look for answers to homework problems in other texts. However, 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 during 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.

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 You are expected to do a **lot** of independent reading for this course. I will be presenting selected topics from the text, but much of what I'll be going over in class will require background not covered explicitly in class, either from an undergraduate course or from independent reading. The text tends to go over material in immaculate detail, and can frequently be skimmed so the reading assignments are not quite as heavy as they may seem at first glance. However, they are substantial and you are responsible for all material included.

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:

1. Background: Skim the Appendices and Chapter I (Foundations). Try some exercises. If you have **any** problems, go back and read the chapter thoroughly. All of this material is important and none of it will be covered in class. Do the callibration homework.
2. First 11 lectures: Basic algorithm design techniques: Divide-and-Conquer, Backtracking, Depth-first Search, Breadth-first Search, Dynammic Programming, Greedy Algorithms.

   Divide-and-Conquer : Mergesort(pp. 28-36); Quick-sort (Chapter 7); Depth-first and Breadth-first search in trees (Chapter 22); Strassen's matrix multiply (Chapter 28.2); Integer multiplication and Fast Fourier Transforms (Ch. 30); Divide-and-conquer in parallel computing: Valiant's parallel comparison model(paper to be handed out). 4 lectures.

   Back-tracking, Depth-first Search, Breadth-first Search, Branch-and-Bound Not in text. Example problems: Maximum Independent set (p. 1018), Graph Coloring (p. 1019), Hamiltonian cycle(34.5.3); addition chains (not in text). 2 lectures.

   Greedy algorithms Chapter 16; additional examples: Minimum Spanning Tree(Chapter 23); Dijkstra's algorithm (Chapter 24), greedy approximation algorithms (Chapter 35, e.g., 35.1, 35.3). 2 lectures.

   Dynammic Programming: Chapter 15, additional examples: All-pairs shortest paths (Chapter 25); DP-based approximation algorithms (Chapter 35, e.g. 35.5). 3 lectures.
3. Next 4 lectures: Applying Data Structures to Algorithm Design (Chapters 6, 10-14, 18, 21. A lot of reading, but much of this should be familiar.) Refers to all algorithms mentioned above, in particular greedy algorithms.
4. Network flows and applications . (Chapter 26; see also 29).
5. Reductions and NP-completeness (Section 34) 2 lectures.

6. Heuristics and approximations (Section 35 and additional reading) 1 lecture.
7. Time and opportunity permitting, we'll have one guest lecture or lecture on a topic requested by the class.

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

1. Oct. 2 : Callibration homework due.
2. Oct. 18: First homework due.
3. Nov. 1 : Second homework due.
4. Nov. 15 : Project due.
5. Nov. 29: Third homework due
6. Dec. 6: final exam due