Design and Analysis of Algorithms – CSE 101

Basic Information: Spring, 2014

Instructor: Russell Impagliazzo

Class: Monday and Wednesday, Peterson 108, 5:00 - 6:20

Mandatory discussion section: Wednesday 4-5;

101 Professor Office Hours: Monday, 12:10-2:00 and Thursday 4-6. 4248 or if it’s crowded, we’ll
move to 4262

e-mail: russell@cs.ucsd.edu

webpage: www-cse.ucsd.edu/classes/sp14/cse101-a

TAs: Stefan Schneider, Yan Yan, Udbahv Singh

TA Office Hours: Stefan: Tuesdays 3-6, room TBA. Udbahv: Wed. 10:30-11:30 and 1-3, room TBA. Yan: Friday morning, details TBA.

Prerequisites: CSE 21, CSE 100.

Text Book: Dasgupta, Papadimitriou, and Vazirani, Algorithms

Assignments There will be eight homework assignments, four practice quizzes, four quizzes, and a final
exam.

Evaluation: Homework will account for 32% of the grade, the quizzes 32%, and the final will account
for the remaining 36% of the grade.

Homework should be done in groups of up to five members. Each group turns in a single set of
answers written collectively by the group. (See the section on grading policies for details.) The first
homework assignments will have two grades. The first, which will actually be used to determine
your average, gives full credit for any serious attempt to answer the problem. The second, for your
use only, gives an actual grade using the same standards as the rest of the course. This is your
chance to see what is required for the class without taking a risk.

Each of the four quizzes will be on a Wednesday in class time, generally every other week. There
will always be a practice quiz in discussion section one week earlier, the Wednesday of the previous
week. The practice quiz will have the same types of questions as the quiz, and be graded by the
same standards and handed back by Monday. Your grade for each quiz will be the better of your
grades on the practice quiz and actual quiz.

The final exam will be a three hour in class exam, at the scheduled time.

All quizzes and finals are open book and open notes. However, you may not use any devices capable
of communication, such as computers, smart phones, anything with a wireless connection, texting,
smoke signals or coughing in Morse code.

Academic Integrity Most people who fail this course fail due to academic dishonesty. Thus, I have
found it necessary to enforce some strict rules to guard against academic dishonesty.

I do not grade on a curve. The standards reflect my experience at what UCSD undergraduates
have been capable of in this course in the past, not by your peers this year. It is possible for all
students to get A’s or all students to get C’s, and the actual averages vary significantly from year
to year. In particular, this means that any other student seeming to get ahead by cheating will
not affect your grade, and is not a reason to start cheating yourself. Often, it takes time to gather
convincing proof that a student is cheating, so just because a student seems to be getting away
with it, this can be misleading and is certainly not a reason to cheat yourself.
Students should solve and write up all homework assignments in groups of size two to five. All names should appear on the assignment, and all will usually get the same grade. Members of a group are responsible for all parts of any assignment with their names on it. Problems should be solved by the group, not divided up between group members. Each member of a group should participate in discussions about each problem. Even proof-reading a solution of another student in your group counts as participation, but if there is a problem where you did not participate at all, even in verifying someone else’s solution, you should put a disclaimer to that effect next to your name on the assignment. You will not receive credit for these problems, but you also will not bear responsibility for them. If you are claiming credit for a problem, you do bear responsibility for the integrity of the group in obtaining the answer.

Students should not look for answers to homework problems in other (i.e., other than the course text, the optional text, and class notes) texts or other sources (e.g. wikipedia, Internet discussion groups). However, students may use other texts or sources 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 this text or source, and should give an acknowledgement of the text or source on the first page of their solutions. Such a solution may be given partial or no credit if it too closely follows the text. **Not giving an acknowledgment is academic dishonesty, and will be treated as such.** This rule applies to any material found on the internet, and to conversations with or written material from other people, whether or not they are students in the class. However, it does not apply to material in either text, handed out in class or on the class web-page for this year, or to conversations with the instructor or teaching assistants.

Be sure to follow the following guidelines:

1. Do not discuss problems with people outside your group (except during office hours, with the TAs or me).
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 sources that had solutions to homework problems.

### 1 Standards for assignments

Most required assignments will be mathematical or theoretical in nature, involving designing and analyzing an algorithm. Although some assignments will require students to design and analyze pseudo-code programs, no implementation will be needed to complete these assignments. However see below for algorithm experiments.

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 an argument could be constructed. In particular, the correct formulas or pseudo-code are not a complete solution by themselves; their significance and the logic of their application need to be explained.

When giving an algorithm, the following three things should **always** be included, unless the problem explicitly says not to: a clear and complete description of your algorithm; a correctness proof, showing why the algorithm solves the problem in question; and a time analysis, giving the worst-case runtime (up to order, in \( O \)-notation). Descriptions need to be unambiguous. You should be able to give the description to any other student and have them easily implement a correct program from it. Proofs need to be completely clear, completely unambiguous, and logically compelling, but can be in high-level mathematical English. Time analysis needs to give a true upper bound
on the time taken by the algorithm, in $O$ notation. You need not argue that the bound is tight, but your grade will be partially based on how fast you show your algorithm is, not just how fast it really is. So if your algorithm takes time $O(n^2)$ and you claim it is $O(n^3)$, this is also correct, but you will be graded on efficiency as if your algorithm really took $\Omega(n^3)$ time.

Some relaxation of this rule will apply to problems of a computational nature, where you are merely expected to present a solution and give some informal justification. Such problems will be designated by key phrases such as “Find a solution and justify your answer.”

About every other homework assignment will have an algorithmic experiment problem. This will involve implementing an algorithm and running it on test examples. However, the point is to run the experiment; it is assumed that you can implement the algorithm successfully. The experiment may involve comparing the time used by or performance of several algorithms or versions of an algorithm. You can use any programming language, and do not need to turn in your code. What you need to turn in is a clear description of the experiment you ran (e.g., which algorithm you implemented, which programming language you used, which program library you used, what the programming environment you ran on is like.) Then clearly present the results, giving a table or graph of the results in terms of input sizes. Often, a log-log scale (log of the input size vs. log of the time used) is useful in giving a clear picture of algorithm performance.

2 Presentation

We strongly prefer homework to be typeset using an editor such as LaTeX. We prefer physical printouts of the assignments to be turned in in person, but will accept a .pdf file mailed to us before the due date if you cannot turn it in on time in class.

3 Lateness Policy

Homework will usually be due at the start of class each Monday. (At some point in the quarter, there might be a skipped week or an extension until Wednesday.) Late homework will be accepted until I give out an answer key and no later. So you have to be no later than me. I can give some leeway if you have an emergency, especially if you email me before the due date with a request for an extension.

Reading Schedule We will not be able to cover every example on each topic in the text in class. You are expected to read the other sub-sections independently. In particular, we will not cover the material in Chapter 2 explicitly, since it should have been covered in CSE 20 and 21. We will be using it extensively, however; so reading this chapter in advance is a good plan.

To help you plan your reading, here is a tentative schedule of topics to be covered in class, and the corresponding sections of the text to be read. I reserve the option to change the schedule at a later point.

**Background**: Order notation, time analysis, recurrence relations: See chapter 0. This material should be covered in CSE 20 and 21. Read it and try some exercises. If you have any problems, go back and read the chapter thoroughly. Basic data structures: lists, arrays, graph representations (adjacency matrix vs. adjacency list), heaps binary search trees, and hash tables. These should have been covered in CSE 100. All of this material is important and none of it will be explicitly covered in class.

**Graph search** (Chapters 3 and 4: 3 lectures). High-level goals: Going from a high-level strategy to an efficient algorithm; algorithm analysis with multiple parameters; tight analysis of loops; reviewing proofs of correctness; the role of data structures in both defining inputs and as sub-procedures in algorithms; how to generalize an algorithm technique to a new problem;
how to use an algorithm for one problem as a “sub-routine” in an algorithm for another. Example problems: Graph connectivity, breadth-first vs. depth-first search, shortest paths (Dijkstra’s algorithm), maximum bandwidth paths, topological search.

**More on going from high level strategies to efficient algorithms**, 3 lectures, but intermingled with graph search, greedy algorithms. High level goals: Using the high-level algorithmic strategy to prove correctness; Using pre-processing, restructuring, and incorporating data structures to make efficient versions of algorithms. Amortized analysis of algorithms. Examples: heap sort, skylines, element distinctness, efficient versions of Dijkstra’s algorithm, strongly connected components, the orbit problem.

**Greedy Algorithms, Chapter 5** 3 lectures. High-level goals: Why greedy algorithms are dangerous; why they are useful; techniques for proving greedy algorithms are correct: greedy stays ahead, exchange arguments, achieving a bound. Example problems: business plan, one-machine interval scheduling (4.1); Minimum spanning trees (4.5); minimum machines event scheduling.

**Divide and Conquer. Chapter 2.** 3 class lectures. High-level goals: Recursive algorithms. Proofs of correctness using strong induction. Recurrence relations and time analysis of recursive algorithms; “Master Theorem”, when is divide-and-conquer good?, How to make it better?, surprising algorithms. Examples: Mergesort (5.1); All-pairs distances in trees; Multiplication of large integers (5.5); Closest pair of points (5.4), Strassen Matrix Multiplication, Tree isomorphism. Quicksort (13.5);


**Dynamic Programming, Chapter 6.** 3 lectures. High-level goals: Finding and exploiting repeated sub-problems in recursions. Memoization, dynamic programming. Systematic conversion of recursive algorithms to DP. Finding alternative recursive algorithms. Example problems: Longest increasing sub-sequence (not in text); weighted interval scheduling (6.1), edit distance, RNA secondary structure (6.5), Shortest paths (6.8-6.10), Load balancing problems and subset sum (6.4).

**Network Flow**, Chapter 7, sections 7.1, 7.2 and 7.3. 2 lectures. High-level goals: Learn the Ford-Fulkerson method; hill-climbing algorithms; When one search problem “codes” another: reductions between problems.

**NP-completeness**, 2 lectures. Chapter 8, 9. High-level goals: definition of NP as class of natural search problems; understanding how to use reductions to show problems hard, existence of NP-complete problems, how to show new problems NP-complete, ubiquity of NP-completeness, how to cope with NP-completeness.

Assignment and Exam schedule: To help you plan, here is the tentative assignment and exam schedule:

1. April 7, Homework 1 due
2. April 14, Homework 2 (basic graph algorithms) due
3. April 16, First practice quiz (order, correctness proofs, graphs) (discussion)
4. April 21, Homework 3 (paths in graphs) due
5. April 23, First quiz (order, correctness proofs, graphs) (class)
6. April 28, Homework 4 (using data structures in algorithms) due
7. April 30, Second practice quiz (paths in graphs, data structures in algorithms) (discussion)
8. May 5, Homework 5 (greedy algorithms) due
9. May 7, Second quiz (paths in graphs, using data structures) (class)
10. May 12, Homework 6 (divide-and-conquer) due
12. May 19, Homework 7 (back-tracking) due
13. May 21, Third quiz (greedy algorithms, divide-and-conquer), class
14. May 26, Homework 8 (dynamic programming) due
15. May 28, Fourth practice quiz (back-tracking, dynamic programming) in discussion
16. June 4, Fourth quiz (back-tracking, dynamic programming) in class
17. June 13, 7-10 PM, Final exam