CSE 202
Basic Information: Winter, 2025

Instructor: Russell Impagliazzo

email: russell@cs.ucsd.edu or rimpagliazzo@ucsd.edu

TAs:

Russell's CSE 202 Office Hours: TBA

TA office hours: TBA

Useful websites: Gradescope Submit assignments, view graded assignments, and make regrade requests.

Canvas Lecture slides, background materials, study guides, old videos.

Piazza Discussion and announcements, finding study groups.

Podcasts podcast.ucsd.edu

Textbook in progress An undergraduate textbook in progress (by Ragesh Jaiswal):

https://sites.google.com/view/algobook/home?pli=1

Comments and errata appreciated.

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 will need to do a significant amount of 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. And see above textbook by Ragesh Jaiswal.

Assignments There will be six homework assignments, a project, and a take-home final exam.

For each homework assignment, there will be a corresponding ungraded homework assignment broken up into exercises (quick review questions to make sure you are following) and more difficult ungraded problems

similar to the homework problems (and often from a previous quarter's homework). We will give answers to the ungraded problems before the graded problems are due, for you to use as a template for the level of detail and rigor required for the graded homework. Most homework assignments 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 five. 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.) You can use the piazza site to help find a study group.

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.) The sincere effort grade will be 99% of the grade for the assignment. (If you complete all problems, your grade should be of the form $100.xy$ where xy is your percentage for correctness, allowing you to read off the score you would have gotten if this were a standard homework assignment.)

A new experiment this year is borrowed from submission guidelines for conferences. I will give each graded problem a "page limit". You should explain the main idea of your algorithm and its analysis within the page limit. You can go beyond the "page limit" as an "appendix" if needed, such as with more detailed pseudocode to make edge cases clearer, or present a simple idea within the page limit but explain how to improve it in the appendix. The graders will be guaranteed to read the part within the page limit, but will usually only read the appendix if the idea within the page limit is clear but they want to verify details. The page limit is meant as a major hint about how complicated algorithms need to be, and to get students to think more about big ideas before generating code. (In the past, some students get really carried away and waste quite a bit of time on over-elaborate solutions.) Please do not use small print or wide margins to try to evade page limits. We can see through that. As an experiment, these page limits might not be strictly enforced this year. Do your best to meet them, but don't worry if you go over by a bit. The limits may be broken down by sub-task, such as algorithm description, correctness proof, and time analysis, for some problems.

Evaluation: Homework will acount for 30 % of the grade (so 5 % per assignment), the project, 20%, and the final will account for the remaining 50 % of the grade. A B- or better on the final can be used as passing the MS Comprehensive Exam for the course.

We will use the following cut-offs to map numerical grades to letter grades:

A+ 98 %

A 91 %

A- 84 %

B+ 77 %

B 70 %

B- 60 %

At the end of the course, I may adjust the cutoffs slightly lower to make gaps between letter grades more significant. I cannot round all grades that are "close" to the cutoff, because with a large class, there will be close grades no matter what cutoffs are used.

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, if this is not already provided. Even if the problem is stated informally, you should give a more formal problem definition.

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. On the other hand, the main ideas of your algorithm should be clearly given in natural language, possibly as comments in your pseudo-code.

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. For different algorithmic paradigms, you'll have templates for the relevant types of correctness proofs. While it is not necessary to follow these templates, we encourage you to do so, since they avoid many of the fallacies and ambiguities students often fall into without guidance.

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. We have to evaluate your answer based on WHAT YOU WRITE not WHAT WE KNOW or WHAT WE THINK YOU MEANT. Communicating ideas in a clear, logical, and unambiguous fashion is a major part of this course. (This will be especially important for the project.)

Your answer will be graded on the following criteria:

1. Your algorithm must be clearly and unambiguously described. Communicating precisely is an important skill that this class is meant to teach.
2. You need to prove your algorithm correctly solves the problem. My rule is: an algorithm is incorrect until PROVED correct. We will use this rule in grading even if we know your algorithm is correct. If you don't explain why your algorithm works, in a manner that would convince someone paid to find flaws in algorithms, we 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 we 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. Algorithm design is a competitive sport. Whether your algorithm is "good enough" depends on what algorithms your competition is using. Yours cannot be substantially slower than theirs.

Some problems will give ballpark times that the algorithms should achieve. Others will not. This is intentional. When we give specific time complexities, students distort the algorithms to meet the given complexities, rather than think through approaches to the problem and pick the best one. This frequently leads to either false claims or artificially slow algorithms. (For example, we might know both a linear time and $O(n \log n)$ algorithm for a problem, but think the $O(n \log n)$ algorithm should be enough for full credit. If we say the algorithm must be $O(n \log n)$, students who came up with the linear time algorithm think they are wrong, and start slowing it down for no good reason. So we'd rather not be that precise.)

There will also be some implementation problems on the homework. The goal for these problems is to conduct a meaningful algorithmic experiment and present data and conclusions in a clear format. Implementing the algorithms in question is necessary for these problems, but isn't the goal in itself. Since the implementation is a means rather than a goal, you can use existing libraries or modify existing code, as long as you acknoweldge your sources and maintain enough control over how the algorithms work that your experiment is valid. (For example, you need to know what data structures any algorithm you are using calls, and if the data structures have been modified in any way from the standard ones mentioned in class.)

For these problems, you should describe clearly what the algorithms you implemented were, what their asymptotic time analyses are, your results from the challenge instances described in the problems, and timing information about various problem instances. Then you need to make a reasonable inference and reach a conclusion from this data about the question asked. (Since this is not well-defined, there might be multiple reasonable inferences possible, and possibly the best conclusion is "There is no statistically significant conclusion possible from this experiment".)

WE 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. Often a fast implementation won't be part of the problem statement, but if you don't use an optimized implementation, you will not be able to collect enough data for large instances to make meaningful conclusions.

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 meant to develop the ability to model applications as algorithmic problems. In order to avoid looking at applications that are already the focus of intense research, where many formulations are already known, one format will be to pick a somewhat frivilous application based on a hobby, puzzle or game, and to avoid those addressed by large amounts of previous research.

Because artificial intelligence involves many complex interrated algorithms, and AI applications are increasingly using a huge amount of resources, minimizing the resources used in AI component algorithms is incredibly important. So a second format for the project will be to discuss such a component sub-procedures used in AI applications. The goal will be to find the most efficient possible algorithms to implement these sub-procedures. You need to identify and define a very specific sub-procedure (in formal terms). You should look not just at the standard or most obvious way of implementing this sub-procedure, but creatively consider alternative solutions. You need to identify the relevant resources used by such procedures (which may go beyond cycles), constraints on the algorithm due to the settings in which they must be used (e.g., distributed data), and give at least some upper bound analysis for the resources used by your algorithms. Note that "using existing AI tools to do X" will not be considered an appropriate project. We want to enable more efficient AI tools, not utilize existing ones. This will usually be a more ambitious and self-guided type of project, but one that could potentially synergize with other class work or research. (Please get permission from your adviser or other instructors before basing it on research or other course work.)

Either type of project will have four milestones. The first few reports are intended not to evaluate your work, but to give you feedback that might help shape an appropriate project and prevent you from picking an over-ambitious project.

Proposal A two to three page description in English of the hobby, game, or sub-procedure your project will focus on, and what related computational questions you want to address. Do not try to define these questions mathematically and do not make any conclusions about algorithmic techniques you might use.

Formulation A mathematically clear formulation of computational problems related to the above. Give a brief justification of how your formal description captures the aspects you described in the first part. Your formulation should have: What is the format for input to your algorithm, including any restrictions or constraints on the input? What is the format for outputs to your algorithm? What constraints , expressed in logical or mathematical terms, must the output satisfy? What objective function or functions (again, expresssed in mathematical terms) define when one solution is better than another? Are there any unusual limitations on the type of algorithm you can use (e.g., on-line, memory restricted, streaming, distributed, parallel, etc.)?

Do not reach conclusions about the types of algorithm you will use to solve the problems. Try to avoid using known algorithms or techniques in the definition of the problem. In other words, don't define the problem after coming up with a solution, and don't restrict your options for what algorithms you will consider.

Algorithmic Solutions: Descriptions and analysis of algorithms you have come up with to solve the problems in your formulation, or other results about the difficulty of these problems (e.g,, NP-completeness). Your algorithms might be exact algorithms or approximation algorithms, or heuristics if all else fails, but you should definitely give time analyses and correctness proofs as applicable.

Experiments: A presentation of results from implementing your algorithm and performing some experiments using the implementation. Submit a summary of the results, and a conclusions about the applicability of the algorithm to the game or hobby.

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. It makes no difference the format other

people's ideas are in, whether an in-person discussion, a textbook chapter, a website, or an AI generated summary. You must acknowledge any source of ideas external to yourself.

(The exceptions are those being renumerated for providing you ideas for this course: myself, the TA's, the textbook, class materials on the website or piazza site for this quarter's section. These do not need to be credited. Any other perspon, site, or source does.)

If you acknowledge others contributions, you will not be guilty of an AI violation. However, we will attempt to grade your work based on the amount your own ideas have added to it. If that amount is zero, you may still get a zero for the assignment.

Other sources: Students should not look for answers to homework problems in other texts or on the internet, and should above all not ask an AI to do their homework for them because AIs are lying twits. 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 algorithms to others and verifying claims about algorithms. By text, I mean any person, site, video, program or materials that was useful to your solution.

To avoid academic integrity violations, *cite* every source you use. If you *cite* a source in a way that allows us to compare the source to your assignment, you will not be cheating. However, we will grade based on the amount you *improve* over the source, so a verbatim repetition or even summary of the source would not be worth much credit.

Artificial intelligence Just as with other collaborations, any use of AI must be acknowledged. As part of the acknowledgement, you should give a summary of prompts you used, or make such prompts available to us if we request them. As with other collaborations, your grade will be determined by the amount your ideas improved on the AI's. So submitting AI generated algorithm descriptions or proofs will be graded as a zero even if acknowledged.

If you use AI as a general study tool, be aware that it is not always accurate or may not reflect the specific terminology or approaches used in this class. Just as with supplementary texts and web sources, do not look for solutions to specific problems from AI tools.

Part of this class is developing your own communication skills. Over reliance on AI editing prevents you from developing your skills (and the AI's technical writing style is often very poor.) Write the final drafts of every assignment yourselves and have human partners proofread them.

If you have any doubts about appropriate use of AI, ask for explicit permission.

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. The TA's and I would not be able to properly grade individual assignments in a timely way. So hand in *one assignment per group*.

Limit discussions of problems Do not discuss graded problems with people outside your group whether students or not (except the TA and me, of course). Ungraded problems may be discussed freely.

Do not share answers Do not share written solutions or partial solutions with other groups.

Final is individual Prepare your final written solution without consulting any written material except class notes and the class text. Do not discuss the final exam with anyone except the instructor and TAs.

Acknowledgement Acknowledge all supplementary texts or other sources that had solutions to homework problems. Acknowledge anyone who helped with assignments, except the TA and myself.

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. If you ask for an extension, I may delay giving out the answer key until you submit, so if

the answer key is not given promptly, this is likely the reason If you ask for an extension, I may delay giving out the answer key until you submit, so if the answer key is not given promptly, this is likely the reason.

Reading Schedule This is a schedule of topics covered in class, and the most relevant textbook sections. Not every class topic will be in the textbook, and not every important textbook example will be covered in class. The textbook and classroom presentations are intended to together give you more perspectives and examples for each topic than each would individually. You are expected to be reading the relevant sections of the textbook around the same time 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).

Our milage may vary as we actually cover the topics in class, and updated schedules may be issued as needed.

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

Introduction, formulating problems 1 lecture (Jan 6).

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. 4 lectures (Jan. 8-15) .

Greedy algorithms Chapter 4, some of chapters 11 and 12. 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. 5 lectures. (Jan 17-29)

Divide-and-Conquer : Chapter 5. Mergesort (5.1, in passing), Integer Multiplication (5.5) and Fast Fourier Transformation, 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. Multiple parameters. 3 lectures. (Jan. 31- Feb. 5))

Randomized and Streaming algorithms See paper on website. Feb. 7

Parallel computation. (see paper on website) 2 lectures (Feb. 10-12)

Search and Optimization problems, and the class NP See section 8.3 . Context for next few topics.

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, second merges with DP: Feb. 14-19).

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). 5 lectures (Feb. 19 - 28).

Network Flows, Hill-climbing, gradient descent, linear programming: 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. (4 lectures, March 3-10)

Reductions and NP-completeness (Chapter 8.2, 8.4-8.7). (March 12 lecture)

Coping with intractibility: Heuristics, average-case analysis, and approximations (Chapter 11, 12). 1 lecture (March 14).

Assignment Schedule **Jan. 14** Homework 1 (background) due

Jan 21 Project proposal due

Jan. 28 Homework 2 (path-finding, reductions, using data structures) due

Feb. 4 Project problem specification due

Feb. 11 Homework 3 (greedy algorithms, approximation algorithms) due

Feb. 18 Homework 4 (divide and conquer) due

Feb 25 Project algorithms and analyis due

Mar. 4 Homework 5 (BT, DP) due

Mar. 11 Project implementation and experiments due

March 12 Take home final available

March 13 Homework 6 (NF, GD) due

March 18 Take home final due