Class Project

The basic goal of the course project is for you to independently apply some of the thinking you have (hopefully) been developing in this class. In particular, the project will involve you independently tackling new material and presenting it. There are three general types of project.

Reading Project: Find a few interesting but challenging theoretical papers on a coherent topic related to randomized algorithms or probabilistic analysis, and write up an exposition that synthesizes their ideas. The goal here is for you to develop an exposition that is both clearer and more informative than that of the original papers (and this will be the main component of your grade). The papers should therefore be quite new; it is not enough to take some already well-known topic from journals or textbooks and re-describe it. As the emphasis here is on synthesis, you need not give all proofs in full detail, but the reader should come away with a good understanding of why it all works. Unless the paper is extensive, one paper is usually not sufficient; two or three usually are. And simply summarizing each one isn’t sufficient; I am not asking for an enumeration of results. Focus instead on explaining the key ideas, insights, and techniques from the papers.

Implementation project: Algorithms papers tend to leave out a lot of little implementation details that turn out not to be so little in practice. Find a randomized algorithm that interests you – either from class or (like a reading project) from a recent paper – and implement and test it. This kind of project will be graded on how well you explain the algorithm you are implementing, what kind of interesting heuristics or implementation level “improvements” you developed, what interesting test cases you run the algorithm against, and how well you interpret the results. Note that the reader (me) may well not be familiar with the algorithms you are implementing; thus, your paper must give an adequate description of the algorithm, not just jump to the implementation results.

Note that you may also explore implementation heuristics that have “not theoretically valuable” but may lead to enormous improvements in practice. Once you have an implemented algorithm, you can test it to see how it performs in practice, and make such improvements.

Be careful – implementations can take a lot of time. Make sure what you are trying to implement is a suitably small, self-contained algorithm. Give serious thought to test inputs; ideally, they will come from real-world problems, or will be specially designed to stress some aspect of the algorithm. You probably also need a control (i.e. dumb) algorithm to compare yours to. Finally, be aware that there is prior literature in this area: before you implement algorithm X, hunt for papers about implementations of algorithm X. It’s OK to replicate previous results, but you should be clear that’s what you’re doing.

Theoretical research: Develop an interesting new solution to an algorithmic problem; this can be more efficient in some dimension or just simpler. You will need to do background reading. I really can’t recommend this type of project, since, as with all research, you need to be prepared for the risk of not making any progress. Instead, I’d recommend planning a reading or implementation project, and if you make some theoretical progress, that can take its place (and will be duly impressive). Note it is not appropriate to submit some theory project that you’ve already been working on. It would be appropriate to take ideas from class and apply them in a novel way to your existing research.
Regardless of which route you take, the end result should be a roughly 10-12 page paper describing the results. Presentation quality will be a factor in your grade. The paper should be written to be understood by any other student in this class: you should not assume the reader has more knowledge than an undergraduate CS curriculum plus this class. I strongly recommend that you trade readings with someone else in the class; you’ll be surprised at how incomprehensible other students find your writeup, and a good editor will help a lot.

FAQ:

- **Due date?** Let’s say May 5 for now.

- **Is collaboration allowed?** Yes, you can work with one other person if you like. I’ll expect a bit less on a solo project than a pair project, but maybe not too much less, so pairing up is probably a good idea. However, if you’re doing a reading project, keep in mind what a project is supposed to be like – a synthesis. Two people splitting up some papers and summarizing them without any synthesis is not going to be a good project. Similarly, for the other projects, make sure if you are working as a pair your output reflects a shared work product.

- **Which kind of project should I choose?** A reading project is by far the least risky. On the other hand, if you can cope with the risk and want to do a project that stands out, an implementation or research project is the way to go. Perhaps you’ll get (the start of) a paper from the project.

- **Does my project need to relate to the class?** Yes. It should involved randomized algorithms or probabilistic analysis in a non-trivial way. If you have questions on this, see me.

- **What topic should I work on?** The best topic to pick is one you are interested in anyway. Many of you are already involved in some research project; some thought may reveal an algorithmic component. Perhaps your system is presently using a naive algorithm for X and could be improved by using a more sophisticated one. You can do background reading on the topic (reading project), develop a theoretical model of the problem and devise a solution (theoretical project) or take some extant algorithm and try it out (experimental project). Note that you are not limited to what we’ve done in class.

- **Where can I find papers?** Google Scholar is your friend. Use it. The main theory conferences are STOC, FOCS, and SODA (Symposium on the Theory of Computing, Symposium on Foundations fo Computer Science, Symposium on Discrete Algorithms) but there are many more, some on specialized topics (World Wide Web conference, Graph Drawing) and some smaller workshops/conferences (Workshop on Algorithms and Data Structures, European Symposium on Algorithms).