The CS166 Final Project

The final project for CS166 will serve as a capstone for the course. It's a way for you to run wild with a topic, discover something interesting, and share it with everyone.

This handout goes over our expectations for the final project. Some of this is logistics (when are things due?), while some are more big-picture (what's expected of you?). If there's anything you're unclear or unsure about, please let us know – we're happy to clarify!

Here's a quick summary of the deadlines for the project components:

Proposal due Thursday, May 2nd at 2:30PM.

Checkpoint due Thursday, May 16th at 2:30PM.

Writeup due 24 hours prior to your presentation.
Presentations run June 3rd through June 5th.

Project Overview

In a team of three, you will choose a data structure, an algorithm pertaining to a specific data structure, or a theoretical result about data structures that will serve as the focus of your project. You'll then spend the remainder of the quarter doing your best to become an expert on the topic. To do that, you'll probably scour a bunch of research papers, blog posts, book chapters, etc. to see what you find.

Once you're up to speed with how the data structure works and how it sits in the larger ecosystem, you'll do three things:

- ***Do something "interesting" with the data structure***. This part of the project, which we refer to as the "interesting" component of the project, is fairly open-ended. You should aim to do something with the data structure that sheds some new light on it in some way. We'll discuss the "interesting" component in more detail later.
- ***Write a report.*** You and your team should produce a written report summarizing how the data structure works and detailing your "interesting" component. There are several ways you can write up your findings, as discussed in a later section.
- ***Give a presentation.*** You and your team will give a 15-20 minute presentation on your topic and "interesting" component to the course staff. These presentations are open to the public – you're welcome to invite anyone you'd like – and are your chance to show off everything you've been up to!

There are two milestones before the final project comes due. The first is a project proposal, in which you'll rank topics and scout out the landscape. The second is a checkpoint that serves as a progress report and check-in.

Your project will be evaluated based on your written report, your presentation, and your interesting component, each weighted roughly the same. The official deliverables are your written report and your presentation, and we'll learn more about your interesting component through your report and your presentation.

Step One: Submit a Proposal

To ensure that we don't end up with too many people covering the same topics in their final projects, your first step in working on the final project is to submit a list of project proposals.

First, determine who you'll be working with. You are required to work in a group of three unless you receive prior approval from the course staff (that is, we specifically authorize you to work individually or in a pair *before* you submit the project proposal). The requirement to work in a group of three is logistical; we simply don't have the staffing available to have more project groups than this. Sorry about that.

Next, we'd like you to list, in ranked order, ***four*** choices of topics for your final project. We'll give you a list of suggestions in another handout, but you're welcome to choose any data-structure-related topics you'd like as long as the following are true:

- ***The topic isn't something we've already covered***. However, you are welcome to pick topics that follow up on lecture or problem set topics. For example, feel free to tell us about progress on RMQ since the Fischer-Heun paper, or about improvements to SA-IS since the original 2008 paper, or on B$^+$-trees as an optimization of B-trees.
- ***That topic is focused on a specific data structure or research paper***. We want to make sure every project has a clear focus, and we've found that the best way to achieve this is to ground each project on a particular data structure or paper. For example, a project on "binary search trees" would be too general, since there are hundreds of variations on this theme. A project on the paper *The Geometry of Binary Search Trees* is very reasonable. A project on "purely functional data structures" is too broad, since that's a whole area of research. A project on "purely functional red/black trees" is at the right level of detail.

For each topic that you rank, we'd like you to find at least ***two*** sources on that topic as part of your project proposal. One of those sources must be an academic paper, and the other can be any source that you'd like. We recommend finding a set of course notes or a follow-up paper as your second source; these often do a good job summarizing the key ideas from the original paper.

Once we've received everyone's submissions, we'll run a matchmaking algorithm to assign topics to teams. To ensure that we have a wide array of topics presented, no two teams will be assigned the same topic. In case you're curious, we'll be using a stable marriage algorithm so that no two teams have a mutual incentive to swap project topics. Once your topic is assigned, you will need explicit permission from the course staff to change it – again, this is so we don't end up with duplicate topics.

Some things to keep in mind:

- Really do take the source-gathering seriously. You will ultimately end up becoming an expert on a particular topic, and some topics are much easier to read up on than others. Don't just find a paper and call it a day – spend some time reading over that paper to make sure you have a sense of what you'd be signing up for and how good a resource it is. If you haven't read a research paper before, we highly recommend checking out Keshav's meta-paper *How to Read a Paper*.
- Because we're using stable marriage, if your top-ranked pick is something that no one else in the course proposed, you are *guaranteed* to get that topic. Every quarter we've had several new data structures proposed and discussed, and they're often quite exciting. However, be careful not to pick something obscure for the sake of its obscurity – often times, there are compelling reasons why some topic isn't more mainstream. ☺
- If you're unsure whether something would qualify as a good topic, please feel free to ask us! We're happy to weigh in and offer suggestions.

Your proposal will be graded on a 0 / 1 scale. If you turn in a project proposal that meets the above requirements, we'll give you one point. If not, we'll give you no points.

Step Two: Submit the Checkpoint

Once you have topics assigned, you'll need to start working to build a mastery of your topic. Your first deadline is the checkpoint, which consists of three milestones:

- ***Address questions from the course staff***. Around the time you receive your topic assignment, the course staff will reach out to you with questions about your particular topic. These questions are designed to help you focus your efforts as you're learning more about that topic. Your checkpoint should include the best answers to the questions that you're able to provide. If you're able to answer the questions, great! If you have a hunch about the answers but aren't sure, that's okay too! Write down what you've thought about so far and what you're blocked on. If you are completely stuck, no worries! Tell us, in detail, all the leads you tried out and why you keep getting stuck.
- ***Describe your planned "interesting" component***. Working through the writeup will help you get a much better sense of the topic that you've chosen. Based on your experience so far, you'll need to submit a proposal for what your "interesting" component will be. (There's a list of suggestions in the section on "interesting" components later in this handout.) This proposal should be detailed enough for us to have a sense of what it is that you're going to be doing. You don't necessarily need to give all the details about what you're going to do – after all, you'll have several weeks to actually work through the interesting component – but you should tell us

  - what you are planning on doing,
  - why that interesting component will shed some new light on your topic, and
  - what other work you've found that's in the same general area.

  There are two reasons we're asking you to submit your proposed "interesting" component here. The first is that we'd like to get a chance to offer input and suggestions about how to make it as awesome as possible. For example, if your "interesting" component is to code up a data structure and benchmark it, we might offer input about what to compare it against, or provide some questions to think through as you're doing the analysis. The second is to make sure that you're giving thought to what it is that you'd like to do. We've found that some of the best projects from past quarters arose because the team arrived at some big question they didn't understand, then used that to motivate the rest of their project. For example, a team once was presenting on a string-matching data structure that seemed fairly obscure and was curious why it was that it wasn't more popular. They proposed benchmarking it against other approaches on different workflows, and eventually discovered that there was a *very good reason* no one was using that data structure.
- ***Submit a progress report***. Write a brief (about 2,000-word) summary of your progress in understanding the topic. If you have a deep understanding of your topic, great! Write up an overview of the topic, highlighting key intuitions and describing how everything fits together. If you kinda sorta get it, then detail the parts that you're really solid on, and identify the points that you're still struggling with. And if you're finding that you're in over your head, tell us everything you've tried to do to get a better understanding and your best guesses of how everything fits together.

  The above paragraph shouldn't be interpreted to mean "it's okay to slack off." It's one thing to say "we did our best to understand things, here's what we managed to get, here's the areas that we're struggling with, here's what we've done to try to get unstuck, and here's our best guesses of what we're looking at." It's another thing to say "yeah, we just don't get it." If you're stuck and have no idea how to proceed, please reach out to us before the checkpoint deadline. We're happy to help!

Your deliverable for this section is your progress report, which should also include the "interesting" proposal and your answers to our questions. It's graded on a 0 / 1 / 2 scale, where 2 means "you're at or above where you need to be," 1 means "you're on the right track but not quite there yet," and 0 means "you're not where you need to be."

Step Three: Do the "Interesting" Component

You have wide latitude in choosing your "interesting" component. Here's a sampling of some of the stronger "interesting" components we've seen in the past. This list is meant to help you brainstorm ideas and is not exhaustive. Every year we've seen something new that totally blew us away!

- ***Apply the data structure to solve a problem, then analyze and interpret the results.***
  - Using persistent B-trees (a variation on B-trees where all previous versions of the data structure are accessible after any update) to represent a series of snapshots of webpages, then analyzing in what cases they work well for compression.
  - Implementing an optimized version of the Burrows-Wheeler transform using processor intrinsics as part of a data compression pipeline, then analyzing the efficiency gains.
- ***Implement the data structure, compare its performance against benchmarks or against theoretically-predicted behavior, and analyze the results.***
  - Creating and optimizing a learned index structure (a hybrid of a machine learning model and a Bloom filter) and determining what parameters and inputs lead to the best results.
  - Developing an image compression algorithm based on splay trees and Hilbert curves, then comparing that compression algorithm against standard lossless compression schemes.
  - Comparing cache-oblivious binary search trees (BSTs that gain the caching advantages of B-trees without being aware of the cache line size) against other BST representations and determining how well the theoretical bounds match real-world performance.
- ***Prove a theoretical result about a data structure and use that proof to improve understanding.***
  - Proving that Robin Hood hash tables (hash tables where elements are pushed around so that few elements are in bad places) satisfy a key structural invariant, then using that invariant to simplify the implementations of the key hash table operations.
  - Determining that the costs of accesses in a k-d tree (a multidimensional search tree) are related to parabolas in metric spaces, and using this to explain empirical performance data.
- ***Generalize a data structure, or explore why such a generalization isn't typically used.***
  - Exploring how rewrite rules used to simplify majority/inverter graphs (used to represent logic circuits in synthesis software) can be generalized from 3-input gates to 5-input gates, and why doing so loses some of the elegance of the 3-input case.
  - Investigating how cuckoo hash table performance degradation thresholds change as the number of tables and slots change.
- ***Design a visualization of the data structure that makes it significantly easier to understand.***
  - Visualizing finger trees, a purely functional data structure formed by turning the leaves of binary search trees into new "roots," as a central spoke with small branches containing the elements, dramatically simplifying the presentation.
  - Encoding binary search trees as triangulations of convex polygons, providing new intuitions for tree rotations and search tree deletion.
- ***Explore the progression of related algorithms and data structures and explain how the data structure fits into a broader context.***
  - Exploring the history of concurrent priority queues and how the evolution of machine hardware has changed the costs and benefits of different parallel data structures.
  - Comparing one particular data structure for nearest neighbor lookup (ball trees) against older (quadtrees) and newer (locality-sensitive hashing) strategies, showing how different key insights fundamentally shifted how data structures were designed.

Your "interesting" component will be evaluated on three metrics:

- ***Creativity:*** How original is your idea? Are you breaking new ground, or are you doing something fairly standard? Simply coding up a data structure is likely a low-creativity endeavor, though if you use that implementation in a novel way, that could be high-creativity.

  A good way to come up with a high-creativity project is to ask questions like "why isn't this data structure used in this place I would have expected to have seen it?" or "why don't we do this some other way?" These questions typically lead in promising directions, as answering them requires you to dive deeper into an area you might not have explored.

- ***Difficulty:*** How challenging was your project? If you arrived at a novel result or produced a new perspective on a topic, that's almost certainly high-difficulty (if not, someone else would have already done it!). If you took off-the-shelf implementations of different data structures and ran them against each other, it's a little bit less impressive.

  "Difficulty" doesn't mean that your final project is necessarily *complicated*. It just needs to be *non-trivial*. Doing a lot of work to arrive at a simple result will not result in you getting a low difficulty score. Picking something you know is going to be easy from the get-go likely will.

- ***Insight:*** How much did we learn from your "interesting" component? If you've exposed a hidden structure inside of a well-known data structure, or demonstrated convincingly why one particular algorithm runs quickly, or fundamentally shifted our perception about how to think about problem-solving, that's high-insight. If you confirmed that a slow data structure is slow without diving into *why* it's slow, that's low-insight.

  Want to come up with a high-insight final project? Before you decide what to do, make a list of questions you haven't answered to your own level of satisfaction about your topic. Then ask – what could we do to answer this question? Projects chosen this way typically end up with some very cool insights at the end.

You'll detail the results of your "interesting" component in both your writeup and your presentation, so there are no explicit deliverables for this section.

Step Four: Put Together Your Writeup

The project writeup is one of your two deliverables. This is your chance to describe your topic in detail and to talk us through what you did for your "interesting" component.

In your writeup, you should describe the data structure or topic you chose, explain it and any key results, and to give a detailed account of your "interesting" component. ***Don't forget to describe your interesting component here***; this is one of the major places we're going to look to hear about what you did and what you learned in the process!

You have some latitude in how you produce your writeup. You could write it as a standard technical report, aimed primarily at theory-minded folks. If you go down this route, feel free to draw pictures, make digressions to explain concepts clearly, etc. The gold standard here would be something like Tim Roughgarden's lecture notes (here's an example of what that looks like), which are both rigorous and lucid.

Alternatively, you could put together a long-form article (say, a detailed blog post) describing the data structure to practicing programmers with a solid math background. A great example of a technical topic explained in a blog post is this article by Stanford MS student Roslyn Cyrus about I/O redirection and pipes in Linux – it's a great mix of intuition and rigor, with great use of diagrams to clarify things.

We don't have fixed length requirements for the writeup, but we expect that most writeups will be between 5,000 and 10,000 words. Figure that you'll likely have a lot to say – you'll want to explain your topic in detail and describe what you did in your "interesting" component – so we don't expect that meeting the minimum word count will be too tricky. If you find that what you have is way shorter or way longer than the numbers quoted above, feel free to reach out to us! Our perspective is that as long as what you have is lucid and complete, the length isn't super important.

Regardless of how you choose to structure your writeup, it will be evaluated on three metrics:

- ***Clarity:*** How clear is your exposition on the topic? Does it flow naturally in a way that guides the reader through a difficult idea, or does it force the reader to keep pausing to refer back to concepts that are described earlier? Does it contain speling, grammatically, or $\text{f}^{\text{or}}{}_{\text{matt}}\text{i}^{\text{n}}\text{g}$ errors, or is it well-proofread? Remember that your goal isn't to just brain-dump everything you've learned; it's to explain how something works and why it's worth learning about in the first place.
- ***Intuition:*** How well does your writeup communicate the intuition behind your topic? For example, if there's some key technical lemma necessary for things to work properly, did you just present the lemma "as-is," or did you help the reader contextualize what it says and understand why it's true? If there's some unusual choice of a constant, can you account for why that constant is chosen that way? And if there's something that's surprising, can you provide an explanation as to why, in retrospect, it might not be as surprising as initially perceived?
- ***Rigor:*** This is a theory class, after all, and so we will be looking at how rigorous you are in your writeup. You don't need to prove everything, but you shouldn't hand-wave away every result. Pick a few interesting bits here and there and really dive deep into them! This is something that, hopefully, you'll already have done when reading up on the topic in the first place.

Although none of the following questions need to be addressed in your writeup, we've found that, historically, writeups that do go into these details typically are much easier to read:

- Where does your topic sit in the broader landscape of data structures? Is it primarily of theoretical interest, or is it something that you'll likely encounter IRL?
- What other work has been done in the same space? Was your topic a groundbreaking innovation, or was it the continuation of a long line of other ideas? And has anyone else improved upon it?
- Why are things the way they are and not some other way? That is, why are certain invariants there, what happens if you perturb the constants, and what would occur if you tried doing things differently? Would things still work? If so, why? If not, why not?

A major point that we need to address: ***your writeup should be your own words and your own presentation***. Your writeup should *not* be a paraphrase of your sources. That is, if we were to line up the sources you've cited and the paper you wrote, we shouldn't feel as though we're reading the same thing twice. You should be aiming for *synthesis* rather than *recall*. A good way to avoid rephrasing your sources is to try to explain things from scratch. Can you reverse engineer the thought process that led to the discovery of your topic? Or can you at least tell a good story about how, with the right insights and hunches, you can get to the finished version?

Regardless of what you do, ***you must cite your sources*** in your writeup, and not just if you're taking a passage from somewhere else. Remember, it's plagiarism to take another person's work and present it as your own! This applies even if you aren't literally copying text or figures. If your writeup essentially follows along with someone else's explanation, you need to make that clear through appropriate citations.

You're welcome to share drafts of your writeup with other CS166 students to get feedback, and in fact you're encouraged to do so. Just make a note of it somewhere in your writeup.

The writeup is due 24 hours before your group gives its presentation. This will give us time to give a quick read of your writeup before you give your presentation. For example, if your group is presenting on Monday, June 3 at 3:30PM, your writeup is due on Sunday, June 2 at 3:30PM.

Step Five: Give Your Presentation

Your second deliverable is a 15- to 20-minute presentation that you'll give during the last week of class.

As with the writeup, you should use this time to introduce your data structure, describe it, and present the “interesting” component of your project. Unlike the writeup, during your presentation you do not need to give a complete description of your data structure. Instead, present the basics (what does it do? how does it work? what are the time bounds?) in a way that makes it accessible and understandable. Your job should be to communicate the data structure and present your “interesting” component in a way that makes the structure seem as interesting and exciting as possible. ***Again, don’t forget to cover your “interesting” component!*** Otherwise, we won’t know about what you were working on.

At the end of your presentation, we'll open for questions from the course staff and anyone else present. You should be prepared, in particular, to discuss your “interesting” component (we’ll likely have questions about it from your writeup). We may ask questions on topics that are purely in your writeup, though we’ll try to keep things focused on the presentation at hand and the interesting component.

These presentations are open to the public. You can probably expect that two or three members of the course staff will be there, along with other CS166 students and generally interested members of the CS community. Accordingly, you can assume that the audience has a technical background at the level of CS166, but you should not assume that they already know what your data structure is or how it works. In particular, even though the course staff will have likely already taken a quick look at your writeup, we won't necessarily have a full command of the data structure at the time you present.

Your presentation will be graded on the following metrics:

- ***Clarity:*** The best presentations are ones where you can come in knowing nothing, or the merest contours of the topic, and walk out feeling smarter. How well can you lead us from ignorance into understanding?
- ***Scoping:*** Your writeup should have so much information in it that it couldn’t be presented in twenty minutes, and your presentation should accordingly not attempt to cover everything. You won’t have time to present everything in the time allotted, so you’ll need to make some decisions about what to prioritize. If you substantially run over time, or if you have to rush through the most interesting parts of your talk, that’s likely due to poor scoping. If you’ve rehearsed your talk and evaluated what it is that you want to cover, you shouldn’t have too much trouble getting high marks here.
- ***Questions:*** We’ll be asking you questions about your project at the end of the presentation. These questions come from a place of genuine curiosity – we’re excited to hear what you’ve learned, and we love getting to ask people about their areas of expertise! If you have a solid command of the topic, have really given thought to your “interesting” component, and generally put a good amount of time and energy into the project, chances are you’ll do great here. If, on the other hand, you never really understood some key aspects of the topic, you may find this part more challenging.

Because presentations are spaced apart at roughly 30-minute intervals and we want to have five minutes for questions at the end, we may have to cut off early any presentations that run over the time limit. Additionally, please arrive at least five minutes early so that you have time to get set up.

As a note, ***all members of the team are expected to speak at the presentation***. After all, we want to hear about what you learned!

Once your presentation is over, that’s it! You’re done! Congratulations on finishing your project!