The CS166 Research Project

The research 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 research project. The first half of the document talks about what the project will consist of. The second half of the document goes over specific deadlines and expectations for those deadlines.

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

- **Proposal due Thursday, April 21st at 3:15PM**
- **Checkpoint 1 due Thursday, May 12th at 3:15PM**
- **Checkpoint 2 due Tuesday, May 24th at 3:15PM**
- **Project Writeup due Friday, June 3rd at 3:15PM**
- **Q&A Sessions run Wednesday, June 8th from 3:30PM – 6:30PM**
**Project Overview**

In a team of three or four, you'll 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 spend the remainder of the quarter doing your best to become an expert on the topic.

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:

- **Create an explanatory article on your topic.** You and your team will put together a (possibly interactive) article that helps people new to the topic build a deep understanding of how your topic works. You'll also describe your “interesting” component (detailed below) in this explanation.

- **Do something “interesting” with the topic.** 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.

- **Hold a Q&A session with the course staff.** Once you've submitted your written project, you'll schedule a time for your group to meet with the course staff for a small Q&A session. This will be a mix of us asking clarifying questions and just generally discussing what you learned in the course of doing the project.

There are three steps leading up to the final written project deadline.

- **A project proposal** in which you’ll tell us who you’re working with and list the topics you are most interested in exploring. Once this is submitted, we’ll assign project topics and send each team some questions to think about.

- **A first checkpoint** where you’ll submit a progress report, answer questions from the course staff, and outline areas that you’d like to learn more about. We’ll then offer feedback and guidance about how to proceed going forward.

- **A second checkpoint** where you’ll turn in a more polished draft of your explanatory article and outline the preliminary results from your “interesting” component. We’ll then offer more feedback and guidance as you finish things up.

Your project will be evaluated based on the checkpoints, your explanatory article, your interesting component, and your Q&A session. The two checkpoints are graded on a “making good progress” / “needs some work” basis, and the final project is graded based on your writeup and “interesting” component, as described later, along with how the Q&A session went. Specifically, your project grade is calculated as

- **Explanatory Article:** 45%
- **“Interesting” Component:** 35%
- **Q&A Session:** 15%
- **Checkpoints/Proposal:** 5%

The rest of this handout details the individual components of the project, as well as expectations for the checkpoints.
Your Written Deliverable: The Explanatory Article

Your major deliverable is an explanatory article on your topic. The goal of this article is to guide a reader on a journey from ignorance to understanding. This means that you'll want to explain your concepts using a mix of high-level intuitions, drawings, animations, interactive demos (if you'd like), and rigorous math.

Think of this as more of an exercise in good storytelling rather than an exercise in writing a journal article. Journal articles are designed to be compact. There’s an expectation that someone reading a paper will sit down with scratch paper and take notes, following along with the reasoning and pausing to verify intermediate results if necessary. In that sense, a journal article can be thought of as “knowledge concentrate” – you need to add some water and stir quite a bit to get something edible out of it.

On the other hand, what we’re asking you to do is to write something that is lucid, clear, well-motivated, and easy to follow. Be conversational. Draw lots of pictures. Provide motivation for the topic and each piece. Give intuitive explanations before going into more difficult math. Create some animations showing how things work, or give the user interactive controls they can use to step through things.

Here are some examples of the sorts of explanatory articles that, in our view, do a great job explaining how things work:

- **Pipes, Forks, & Dups: Understanding Command Execution and Input/Output Data Flow** by Roz Cyrus. This article explains highly technical concepts (I/O and redirection) using a blend of text, code, and pictures. I'm particularly impressed with how the visuals give a sense of “plug and play” that is inherent to how the code works but not directly manifested anywhere in the C source.

- **It's Triangles All the Way Down** by Diego Hernandez, Griffin Koontz, and Amy Liu. This blog shows an unexpected connection between binary search trees and triangulations of \((n+2)\)-vertex polygons, how they connect with tree rotations, and how they simplify our understanding of deleting from BSTs. This was a past CS166 project and the TAs and I were impressed by how easy this was to follow and how the pictures and animations make everything so easy to follow, and the included interactive demo makes it easy to play with the concepts.

- **The Burrows-Wheeler Transform and FM-Indices** by Karey Shi, Tim Ngo, and Varun Nambikrishnan. This project from Spring 2020 did an excellent job walking the reader through the intricacies of the Burrows-Wheeler transform, its history, its structure, and how it can then be compressed while remaining searchable through the FM-index data structure. The writeup is presented in a way that makes each step in the sequence seem totally reasonable and unpacks dense definitions through visuals that make things exceptionally simple to intuit.

- **Making Maps with Noise Functions** by Amit Patel. This article explores noise functions and how to use them to generate realistic terrains. The interactive slides make this a joy to explore and provide a powerful intuition for how everything fits together.

- **Understanding the Inverse Ackermann Function** by Raimund Seidel. This talk, which was given by a computer scientist who published a new analysis of the union-find data structure, does a great job motivating both the result that he proved and explaining what on earth the Ackermann inverse function actually is. It blends technical rigor with a conversational tone and good use of diagrams.

- **The Evolution of Trust** by Nicky Case. This interactive exploration of the iterated prisoner's dilemma gives an intuitive feel for the game and the different strategies involved, and in doing so surveys the literature in a deep but approachable way. Is this an “article” in the conventional sense? Nope! But would we be okay with you submitting something like this? Oh yes, definitely!

- **The Essence of Calculus, Chapter 1** by Grant Sanderson. This video on integrals masterfully uses visualizations to motivate basic concepts of calculus in a way that both foreshadows how more advanced topics work and makes the whole idea of an integral intuitive and worthwhile. (And yes, if you’d like to submit a video explanation, go right ahead! But we’re still going to expect it to be rigorous. 😃)
To get to the point where you’re able to produce something lucid and clear like the works cited above, you will need to get very comfortable with your topic. We recommend reading a mix of scholarly papers and secondary sources (blog posts, slide decks, or course materials); the former will help you see how your topic is situated in the broader ecosystem, and the latter are be a great way to get the high-level ideas behind a concept or to see an alternate presentation of the topic. When reading primary sources (e.g. the original papers on your topic), you’ll likely need to do some reverse-engineering and unpacking of the topics you’re exploring. We recommend reading the metapaper *How to Read a Paper* by S. Keshav before diving into research papers if that’s new to you; it’s really helpful!

Here is our advice about how to interrogate a topic and better understand how it works:

- **Draw pictures!** This may seem like a simple piece of advice, but so many complicated concepts from CS theory become so much easier once you know the right picture to draw. This is especially helpful when looking at complicated data structures, mathematical analyses, etc. And don’t just draw one picture. Can you draw the same concept in multiple ways? Some data structures make so much more sense when you see them in action.

And if I may be so bold as to suggest this: **never copy figures from someone else. Always draw your own.** Why? Because when you have to draw everything yourself, you’ll quickly start discovering details that you previously wouldn’t have.

- **Switch from operational to mechanical definitions.** You’ve seen the benefits of having both mechanical and operational definitions for different data structures. Often, when you’re first reading up on a data structure, you’ll only see one of those two definitions. Finding the other can be incredibly helpful in seeing where things come from. For example, I personally found the rules for B-trees to be totally inscrutable until I realized they’re what happens if you obey the two simple rules of “add things to leaf nodes” and “split big nodes and kick keys higher up.” It’s common for papers on data structures to just give operational descriptions, and while it can be challenging to do so, it’s often worthwhile to reverse engineer things and come up with a mechanical description.

- **Perturb things.** Some design decisions in data structures are totally arbitrary. Others are completely necessary for things to work out the way they do. Think the choice of \( b = \frac{1}{4} \log_2 n \) in Fischer-Heun: there’s a balance between two competing forces (\( b \) must be small so that there aren’t too many block types, but large so that the sparse table can be built quickly). If you don’t see why something is there, change it and think about what would happen. You can learn a lot about a data structure by seeing how things break!

- **Break things into layers.** Some data structures are difficult to understand because there are many moving parts linked together. In many cases, what you’re looking at is really several independent ideas that are all stacked on top of one another.

- **Look for isometries.** Imagine you had stumbled upon the pseudocode for heapsort. The code for heapsort is purely done through a series of array operations based on multiplying and dividing by two. If you just see that code, it’ll be difficult to figure out what on earth the algorithm is doing and why it works. But if you realize that there array is a representation of another data structure (binary heaps), then the algorithm’s correctness becomes “obvious.”

- **Read voraciously.** If you’re hitting a dead end, see what other authors have had to say about your topic. Read the original paper, see what it’s citing, and see what cites it. Look for lecture slides or book chapters on the topic. (A single good lecture is worth 1,000 dense papers!) Chances are that seeing multiple perspectives might give you a better foothold to master the topic yourself.
• **Code things up.** Sometimes, things will only click when you code up a data structure and go through the process of debugging it. You might realize that something isn’t working as intended not because you have a bug in your code, but because you didn’t fully understand how the data structure works. (A note: there are some data structures that are so complex that coding them up would be a nightmarish experience. Don’t feel that you have to do this!)

• **Narrow your scope.** Some topics are just denser and more complicated than others, and if you’re overwhelmed by the scope of what’s in front of you, feel free to narrow your focus! Some of the best projects we’ve seen just took one aspect of one part of one data structure and ran with it. This doesn’t mean “pick the easiest part and just do that.” Rather, it means “find something about the topic you’re excited about, and go wild exploring just that part.”

Your explanatory article will be evaluated on the following 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 spelling, grammatically, or formatting 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-Building:** 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.

• **Correctness:** It’s important to ensure that your analysis and exposition are correct. Double-check the details before submitting. If you aren’t sure about whether something is true or how something works, come talk to us! We’re happy to help out.

• **Difficulty:** Some topics are much harder to explore than others. For example, the soft heap, which seems like it’s not all that tricky, is surprisingly complex to fully explain and build an intuition for. Others, like WAVL trees, have a nicer motivation. This isn’t to say that you need to pick a difficult topic. Rather, depending on the difficulty of the topic you pick, we may raise or lower our expectations accordingly.
A Note on Citing Sources

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.

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 to the sources you're following.

A few specifics to watch out for:

- You need to include citations not just if you're taking text directly from another source, but also if the general layout and organization of your explanation follows the general outline of another source. So, for example, if your writeup includes a more detailed analysis of something you read, you must cite that original source.

- If you include a figure or table in your writeup that you took from another source, you must cite that source. I would generally advise against copying figures or tables, though. Often times, you'll learn a lot more if you either draw the figure yourself based on the description you're reading or rerun an experiment to get your own numbers.

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 “Interesting” Component
The other major aspect of your project is the “interesting” component, where you’ll take your topic and do something “interesting” with it. There is no separate deliverable for your “interesting” component. Instead, you’ll describe it and report your findings in your explanatory article.

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.
  - Developing an image compression algorithm based on splay trees and Hilbert curves, then comparing that compression algorithm against standard lossless compression schemes.
- **Formulate a hypothesis, run an experiment, and interpret the results.**
  - Explaining the unusual choice of the coefficient 4.5 in a paper on splay trees by running experiments on actual splay trees and observing 4.5 popping up in the empirical results.
  - Optimizing a learned index structure (a hybrid of a machine learning model and a Bloom filter) and running experiments to determine what makes it tick and what its limits are.
- **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.

As a note, regardless of what you choose to do,

> **prioritize depth over breadth.**

In other words, we would much rather see a deep dive into a narrow area than a broad but shallow survey. Some of the best “interesting” components we’ve seen have looked at just one small part of a data structure, but in a way that totally changed our conception of how that data structure behaved.
Here are some specific pieces of advice if you are planning on coding up your data structure as part of your “interesting” component, based on past experience:

• **Be careful when coding up ideas from Theoryland.** As you’ve seen from this quarter, some data structures are designed to be used heavily in practice, while others are primarily of theoretical interest and aren’t really meant to be used in industrial code. Tournament heaps, for example, are known to have high constant factors, and Fischer-Heun is easily outperformed by other strategies. If your topic is something meant to establish a theoretical result or introduce a new theoretical technique, pause before coding it up. Is doing so going to teach you anything new? If not, hold off on coding it up and look for another project topic.

• **Formulate hypotheses before you start.** Simply coding up a data structure on its own is nice, but that by itself isn’t “interesting” if doing so doesn’t lead to new insights. Treat this as an experimental design problem. What are you interested in learning about the data structure? What specific tests will you run to assess that? If you know what you’re looking for before you get started, you are far more likely to get interesting results when you’re done.

• **Write clean code.** When writing your implementation, comment the heck out of it, pick beautiful variable names, and, more generally, aim to produce something so beautiful that if it were posted online, it would become the de facto, “go to” resource for the topic. If in trying to do this you realize you don’t fully understand why something is done the way it is, stop coding and spend some time reviewing your topic to make sure you really understand it.

• **Make comparisons fair.** We often see projects where people compare one data structure against another in practice to see how they compare, and that’s fine provided they have some reason for doing so. However, if you do compare data structures this way, make sure your comparison is fair. Don’t compare an optimized C++ implementation of a data structure against a Python version of another, for example. Similarly, if you make optimizations to your implementation, make sure you similarly invest a lot of time improving the competition. If possible, use well-tested, commonly-used libraries.

• **Watch your workflows.** There’s an excellent paper entitled *A Back-to-Basics Empirical Study of Priority Queues* by Larkin et al that analyzes many types of priority queues to see how well they hold up in practice. It’s a great read and gives a sense of the gap between theory and practice in this area. One of its major points, though, was that the workflows often used to test priority queues have intrinsic biases in them that can easily lead to inferring the wrong conclusion from the data. If you’re planning on measuring the performance of a data structure, make sure you’ve thought through why you’re using the specific workflows you’ve chosen.

• **Interpret your results.** Don’t just generate plots and call it a day; instead, look at your data and do your best to explain what’s causing it. Ideally, validate your hypotheses by instrumenting the code to get more fine-grained data, or run a secondary set of experiments to see where they take you.
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 conclusion or observation 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.
The Q&A Session

The last part of your project is a Q&A session to be held with the course staff. This will be a 15-20 minute discussion where we'll ask some questions of your team based on your writeup. We may ask to hear more about your interesting component, or about why the data structure is designed the way it is.

You should prepare to answer questions about the following:

- Basic details about the data structure, its operation, and its analysis.
- Specifics about your writeup, claims you’ve made there, or intuitions you’ve built up.
- The broader context of your data structure, including how it relates to others.
- “Meta” questions about the design of the data structure, such as how things would change if different design decisions were made.
- Details about your “interesting” component.

How should you prepare for the Q&A? Our recommendation is to simply review your notes from when you were researching your topic and be comfortable discussing how you approached things. After all, by the time you’re at the Q&A session, you’ll have already completed the project writeup and done your “interesting” component!

We will evaluate your Q&A session by the depth of understanding that you demonstrate when answering questions and discussing the topic. The better you've intuited your topic and know the surrounding landscape, the better.
Deliverables: The Project Proposal

Your first step in working on the project is to submit a project proposal.

First, determine who you’ll be working with. You are required to work in a group of three or four unless you receive prior approval from the course staff (that is, we authorize you to work individually or in a smaller group before you submit the project proposal). This requirement is logistical – we simply don’t have the capacity for everyone to work in smaller groups. By working in a group of three or four, you also give us the flexibility to support students who, due to their circumstances, don’t have the option to do so.

Next, give a rank-ordered list of your four top project topics. 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 or will cover.** For example, you can’t pick the Fischer-Heun structure from lecture, since we already covered it as a group. 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.

- **The topic is a specific data structure, algorithm, or theoretical result.** Every project needs 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, “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. “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. 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:

- **Take the source-gathering seriously.** You will ultimately end up becoming an expert on your 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 ask us!** We’re happy to weigh in and offer suggestions. Just give us enough lead time to do some reading before getting back to you.

The project proposal is graded on a “meets these requirements” / “doesn’t meet these requirements” basis. There is no length requirement; just make sure to include all of the requested information.
Deliverables: The First 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 four milestones:

- **Submit a progress report.** Write a brief (1,000 – 2,000 words) summary of your progress in understanding the topic. In particular, explain the parts of your topic that you understand as well as you can explain them, and identify the areas you don’t yet understand and talk through what you’ve done to try to understand them. The goal of this report is to give us a sense of where you stand and what specifically you’ll need help on.

- **Answer 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.

- **Ask the course staff questions.** In the course of doing the above steps, you will likely have come across some questions you don’t have the answer to. Write out a list of technical questions you’re not sure about. We can then review them and offer some input and advice as you’ve moving forward.

After you’ve submitted your checkpoints, the TAs and I will review what you’ve submitted and set up a time to meet to discuss it. That will let us offer suggestions and advice about how to proceed and provide an opportunity for you to chat with us as well.

The first checkpoint is graded on a “good enough” / “needs more work” basis. We’re expecting to see a solid effort and some good progress toward understanding your topic. You are not required to have a mastery of your topic, nor are you required to have figured out all the details. If you’ve put in a solid effort into each part and taken things seriously, you should be in good shape even if you’re struggling to understand your topic.

If we think your project needs more work, we will outline specific areas to focus and improve on before the next project checkpoint rolls around.
Deliverables: The Second Checkpoint

The second checkpoint is designed to let you show us what you’ve done for your “interesting” component. Your deliverables are

- **A draft of your explanatory article.** You should submit a rough draft of your explanatory article. It will likely be incomplete because you’ll still be working on your “interesting” component, but should start to take on the contours of its final version. Aim to do the best job you can here; the TAs and I will offer feedback on this draft and the more work you put in the more useful our feedback will be.

- **A progress report on your “interesting” component.** This should be a brief (∼1,000-word) writeup summarizing what you’ve done so far for your “interesting” component, along with plans for your next steps. This will give us an opportunity to check in with what you’re doing and offer some suggestions and guidance before you finalize things.

As before, we’ll review your checkpoint and meet with your team to offer feedback and suggestions going forward. Our expectation here is that you’ve made nontrivial progress on your “interesting” component and have gotten your explanatory article into reasonable shape. If that’s the case, great! If not, we’ll give specific action items to focus on between now and the final submission.
Deliverables: Your Explanatory Article
On Friday of the last week of class, you’ll submit your final explanatory article. This should include a summary of your “interesting” component in addition to the general description of your topic. The course staff will read over it almost as soon as it comes in, and in doing so we’ll start to formulate a list of questions to ask you when we meet in person.

Deliverables: The Q&A Session
Toward the end of the quarter, we’ll coordinate times to meet with each project team for a Q&A session. Once the Q&A sessions have wrapped up, it’s time to celebrate! You’re done! Congratulations!