Administrative

Team Name:

ANALYSEE

Team Members (GitHub Usernames):

Ryan Goodman (ryan-goodman), James Wilcox (JamesWilcox-git), and Shane Cross (Shanecross23).

GitHub Repository:

https://github.com/ryan-goodman/Final-Project-Group-4-ANALYSEE

Video Demo:

https://youtu.be/JF3yoIQxbp4

Extended and Refined Proposal

Problem:

While first learning about various data structures, algorithms, and problems/questions related to a DSA curriculum, many students struggle to understand topics at a conceptual level. Though there are several reasons why students may have trouble understanding content, an inability to visualize the particular data structure, algorithm, or problem usually has a notable impact on their ability to fully understand the concept. As a result, visual examples, such as those provided in Professor Kapoor's lectures/slides, are an invaluable part of learning DSA content.

Motivation:

In a perfect world, every student has a fantastic understanding of all content within a course, scores perfectly on every assessment, and passes the class successfully. Unfortunately, this is not a reality and struggles related to learning a topic can often

cause frustration, loss of motivation, etc. which can sometimes result in students abandoning a subject rather irrationally. Any tool that allows students to understand content more effectively and results in a higher success rate by any amount is extremely valuable to educators.

Features:

Our project allows users to run various sorting algorithms on either randomly generated integers or strings. After running the sort, the user can see the amount of time taken, comparisons done, and swaps executed. The purpose of displaying the comparisons and swaps is to give users a better understanding of which operations certain sorting algorithms require more or less of. For example, hardware that struggles with comparisons may want to utilize a sort with less average comparisons. Additionally, the project allows users to compare the metrics of two sorts at once by having a left and right side. Lastly, the code was written with modularity in mind to allow the project to be easily expanded upon in the future (either by us or other students).

Description of Data:

We randomly generated our own integers and strings; the amount of data generated as well as the length of the strings can be easily changed. We plan to add to the project moving forward to allow a wider variety of data sets to be used.

Tools/Languages/APIs/Libraries Used:

Ultimately, our final product was done using Processing (Java). However, we plan to move the project to openFrameworks (C++) soon as it simply offers more tools and better modularity.

Algorithms Implemented:

Quick, Shell, Bubble, and Insertion sorts. (We understand bubble and insertion do not count towards the requirements but created them for testing and more options).

Other Data Structures/Algorithms Used:

N/A.

Distribution of Responsibilities:

Ryan Goodman - Wrote proposals and first version of the project using openFrameworks.
James Wilcox - Wrote majority of final project in Processing.
Shane Cross - Has not contributed.

Analysis

Changes From Initial Proposal:

The most significant changes were completing the project in Processing and not providing animations at the intermediary stages of each sort. Our group was unable to agree on what framework to use which resulted in miscommunication. As a result, entire sorts were implemented instead of iterative versions which allow for a draw then sort cycle. However, we made everything into a side-by-side style application to allow the metrics of different sorts to be compared.

Time Complexity of Notable Functions/Features:

The user's input determines whether to sort integers or strings and which sort to use. The sorts use extremely common implementations, so their time complexities match with the complexity they are usually considered to have. Quick sort performs in around nlog(n) time when the data is NOT sorted. Due to the nature of quick sort, performing it on a sorted array results in stack overflow due to the heavy recursion. To combat this, we check if the items are sorted before performing quick sort and then begin the timer. Shell sort tends to perform in $n^{3/2}$ time when the items are NOT sorted and in nlog(n) time when the items are already sorted. Bubble sort performs in n^2 time due to the many comparisons. Further, the large amount of items requiring sorting results in bubble sort taking extremely long on unsorted data. Insertion sort performs in n^2 time in the worst case when the data is sorted in reverse order. However, it performs significantly better than bubble due to being less repetitive. Additionally, generating the data performs in

O(1) time since the amount of data is always the same and doesn't take long as it is not a heavy operation.

Reflection

Overall Experience:

The overall experience of this project was helpful and insightful. We learned about group dynamics, managing work, resolving disagreements, and common software development practices. The hands on work using a repository to work on and upload files is something that none of us had really done. However, we do feel like the overall project was slightly too small to warrant a repository where people work on different things at the same time. Additionally, it was extremely beneficial to receive input from different team members when making decisions. Coming up with optimizations and modularity practices becomes significantly easier when you have multiple opinions. Ultimately, it was relatively enjoyable to collaborate with each other for a common project.

Challenges:

The biggest challenge our group faced was organization. Various members had other responsibilities going on due to it being close to the end of the semester. Because of this, it was difficult to properly allocate time, convey ideas, and exchange work. We were able to properly come together in the end, but the initial and intermediary stages were definitely rough in terms of organization. Additionally, it was hard to give important updates to each other as Slack is not necessarily checked very often.

Changes if Starting Over:

If we could start over, we would place a large emphasis on clear and proper communication. Most of the stress/worry associated with this project stemmed from a lack of communication; sometimes, it was hard to get a hold of people to converse and be on the same page. Further, we would definitely utilize more of GitHub's features that promote workflow. The issues tab is a fantastic way to stay organized, and we wish we

found it sooner. As for the project, we hope to partially rebuild it to improve the quality and expand it further. We would not make any major changes to the current features, but we do intend to allow more support for interactive aspects that promote learning.

Thoughts from Each Member:

I learned a lot about the importance of making a plan early and how difficult it can be to make multiple moving parts work together. Working with teammates, while a very powerful tool, also requires large amounts of coordination and early planning. If I am ever involved in a different team project, I will have a lot of insight on how to proceed and carry myself. Additionally, I learned about making resolutions and compromises between team members; I realized a large part of working in a group is allowing yourself to be a moving part of the larger machine. Lastly, my understanding of using GitHub for version control and software development has improved significantly thanks to this project. Before, I knew some information about how the process works but nowhere near the amount I know now. - Ryan Goodman

This project taught me the value of establishing early communication with teammates, as well as deciding on an IDE that either everyone can use, or that is easy to learn. In the initial implementation, Ryan made a sophisticated looking project in visual studio using open frameworks, but I was super unfamiliar with how open frameworks functioned and struggled tremendously, so we eventually ended up switching over to the Processing IDE. Essentially, we ended up writing the final project twice, once by each of us (our third teammate Shane did not contribute). Had we worked this out earlier, we would have needed to do a lot less work. Overall I had fun on this project, and I really felt like a full fledged programmer when I spent several full days in a row slamming out code and crying over devious logic problems that I myself had created. - Bowen Wilcox