In the projects for this course, I focused on two main challenges. The first was analyzing the run-time and memory efficiency of different data structures, like stacks, queues, and hash maps, to determine which was best suited for specific operations. The second involved writing a program to sort and print a list of Computer Science courses in alphanumeric order. Both tasks required me to think critically about how data is stored, accessed, and managed, and they reinforced how important it is to pick the right data structure for the job.

To solve these problems, I started by breaking them down into smaller steps. For the analysis in Project One, I looked at how operations like insertion, deletion, and searching would perform in different structures, comparing their efficiency in real-world scenarios like managing a student enrollment system. For Project Two, I focused on implementing a sorting algorithm, choosing bubble sort initially to get a basic version working before optimizing it with something more efficient, like quicksort. This approach made it easier to understand the process and gave me a better handle on debugging when issues came up.

There were definitely roadblocks along the way. One challenge was debugging why my sorting algorithm wasn’t ordering strings correctly; it turned out to be a simple oversight with case sensitivity in the data. Another was understanding when to use dynamic memory structures versus static ones. To get past these issues, I leaned on online tutorials, course discussions, and trial and error. Honestly, just stepping away and coming back later with fresh eyes helped a lot, too.

Working on these projects has really changed how I approach software design. I’ve started thinking more about the long-term impact of my choices, like how easy the code will be to update or scale in the future. For example, instead of hardcoding values or writing messy loops, I made sure my solutions used clean, reusable functions and meaningful variable names. It’s a small thing, but it makes a huge difference when you revisit the code after a while.

Overall, these projects pushed me to write code that’s not just functional but also clear and maintainable. I made an effort to keep things organized, use consistent formatting, and add comments where it mattered. These might seem like little details, but they’re exactly what I’d want if someone handed me this code in a group project or at a job. Looking back, I feel more confident about balancing efficiency with readability, and I know these skills will stick with me as I keep learning and building more complex programs.