1. My project simulation:
   1. My project was a Java project that simulated three different types of scheduling algorithms. One algorithm I’ve implemented was the round robin algorithm with a quantum size of 1, the shortest remaining time algorithm, and the feedback algorithm with a quantum size of 1. I started out by creating a class to read a file, given it has no errors, then storing the process, starting time, and duration into three different hashmaps, with key/values of starting time/process, process/starting time, and process/duration respectively. Finally, I’ve created another class containing the logic of the three scheduling algorithms; to which I’ve created an object in my main class, passing in necessary data structures to make my algorithms work. Finally, I’ve enabled user input, to show results of round robin, shortest remaining time, feedback, or all three algorithms at once.
   2. For all three algorithms, I’ve realized that I’ve needed to keep track of the time, number of processes that have completed, and a data structure to keep track of time spent running. Finally, I’ve also needed a hashmap to keep track of the output result of each process, followed by a method to print necessary results.
   3. I’ve implemented the round robin algorithm by a loop that runs as long as there is at least one incomplete process. When a process comes in, I’ve signaled it to run if the queue is empty, or put it in the queue if it’s another process’s turn to run. If a current process is set to run, I’d show output of all processes that didn’t start or are waiting in the queue (denoted by a space), and an X for the current running process. As a process runs, I’d keep track of how long it’s spent running, and if it completes, my code would signal its completion by not putting it back in the queue.
   4. I’ve implemented the shortest remaining time algorithm by a loop that runs as long as there is at least one incomplete process. When a process comes in, I’d put it in a list that evaluates each process, until it finds the one with the shortest remaining time left, which would be taken out of the list and set as the current process. If a current process is set to run, my code would show output of all processes that didn’t start or are waiting in the queue (denoted by a space), and an X for the current running process. If a process ends, I’d set the current process to blank, so the process with the shortest remaining time has to be found again. If the current process didn't end, my code would put it back inside the list.
   5. I’ve implemented the shortest remaining time algorithm by a loop that runs as long as there is at least one incomplete process. When a process comes in, my code would keep it on top of the first queue (the current process), and move the existing element on the first queue down to the second queue, and the existing element down to the third queue. If a current process is set to run, my code would show output of all processes that didn’t start or are waiting in the queue (denoted by a space), and an X for the current running process. If a process completed, I’d just simply remove it from its existing queue, rather than moving it down a queue, or just to the back of queue 3 if already in queue 3.
2. Difficulties encountered:
   1. I’ve encountered a difficulty with all three of the algorithms because the iterator was the time, and I’d need a hash map with time as a key and process as a value to avoid the hassle of having another loop to find the corresponding process.
   2. I’ve encountered a difficulty with shortest remaining time because I originally used a queue to hold suspended processes, and this algorithm’s next process to run isn’t always guaranteed to be the first suspended one since ones down the data structure may take priority. Therefore, I needed to use an array list instead of a queue.
   3. I’ve encountered a difficulty with feedback because I’ve shown all processes inside the queues to output empty space. This was problematic because the current processes I’ve left in the first queue. Also, I realized that my written interpretation of this algorithm looked at the top priority element, showing output as a running process, before demoting it, so I realized that I needed a flag to signal which queue a process that was taken out of following execution for a bit of time.
   4. I’ve encountered a difficulty when handling gaps of time when no process is running. I’ve handled this by only printing an X for a process if and only if a current process exists. Also, I handled this by resetting the current process to nothing when every data structure is empty and a process has ended, showing an inevitable gap of time.
   5. I’ve encountered a difficulty when I wasn’t able to account for multiple processes that enter at the same time. I originally used a hashmap for an entry time to index a single process, but I changed the value in the key/value pair to a string array list in order to count for two or more processes coming in at the same time.
3. What was learned:
   1. The main thing I’ve learned from this assignment was that while data structures and algorithms based programs can be broken down as much as possible, by getting test cases, writing steps by hand (like what each variable is, what is on this data structure), but implementation in code is different. For example, one process may need to take out the current process, and put it into data structure, or another can be to identify the current process, keep track of data structure it can be put into, and move it to that data structure. I’ve also learned that a bottom up approach works best for me when visualizing algorithms because it helps me consider multiple test cases, and each iteration and transition between each iteration.
   2. I also learned that I need to make my own test cases, and sometimes the most generic test case may come in. I learned to account for gaps of time and multiple processes coming in at once, and despite not being given those processes, I realized I had to tamper with jobs.txt to make my own test cases for gaps of time and multiple processes starting at once.
4. Results:
   1. After all, I’ve successfully implemented all three algorithms to the best of my ability, having it work properly with the input file and a few more similar test cases I’ve created myself. I’m proud of myself for completing it with more than one week to spare, as I have time to study for upcoming exams and do other things, rather than having to play catch up like I did for Project 2. Although this project was the easiest of all three projects in this course, it definitely was not easy, and I was forced to review and build upon the application of my existing data structures and algorithms skills.