CS 300 Project One

John Pate

**Create pseudocode for a menu**.

1. Display a menu with options: "1. Load Data", "2. Print Sorted Courses", "3. Print Course Info", and "9. Exit".
2. Prompt the user to enter a choice from the menu.
3. If the user selects option 1, call the function to load the course data from a file into the data structure.
4. The load function opens the file, reads each line, parses the course information, and stores it in the selected data structure.
5. If the user selects option 2, call the function to print all courses sorted alphanumerically by course number.
6. The print function sorts the data by course number and prints each course's number and title.
7. If the user selects option 3, prompt the user for a course number to search for.
8. The course search function finds the course in the data structure and prints the title and prerequisites.
9. If the user selects option 9, print a message saying "Exiting Program" and terminate the program.
10. The program repeats displaying the menu until the user chooses to exit by selecting option 9.

**Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

1. Define the data structure (vector, hash table, or tree) to hold the course information.
2. Populate the structure with course data, including course numbers, titles, and prerequisites.
3. If using a vector, sort the courses by course number using an efficient sorting algorithm (e.g., quicksort or mergesort).
4. If using a hash table, extract all course entries and store them in a list, then sort the list by course number.
5. If using a tree (e.g., binary search tree), the courses will already be sorted by course number as they are inserted.
6. Once sorted, display a header: "Alphanumerically Ordered Courses".
7. Iterate through the sorted course data structure, printing each course’s number and title.
8. For each course, print its course number, title, and any prerequisites if needed.
9. Ensure that the courses are printed in ascending order of their course numbers.
10. After printing, return to the main menu for further actions, or exit the program.

For the hash table, adding courses is O(1) per course, so it’s O(n) in total. Checking prerequisites is O(1) per check, since we need to do this for each course it ends up being O(n \* m). Sorting the courses takes O(n log n), but since O(n \* m) is bigger, that’s the most important part, making the overall time complexity O(n \* m). For the binary search tree (BST), adding courses takes O(log n) per course, which adds up to O(n log n). For the binary search tree (BST), searching for prerequisites takes O(log n) per lookup, so for all the prerequisites, it adds up to O(m log n). Sorting the courses with an in-order traversal takes O(n), so the overall time complexity is O(n log n + m log n), and O(n log n). For Vector sorting adding courses is O(1) per course (O(n) in total). Checking prerequisites is O(n) per check, so it’s O(m \* n) in total for all the prerequisites. Sorting the courses takes O(n log n), since O(m \* n) is the biggest factor, the final time complexity is O(m \* n).

Hash tables are good for their efficient insertion and looking things up quickly. But bad for sorting and checking the prerequisites for classes. Binary search Trees are good for insertion searching and sorting data. However, searching for prereqs can be inefficient for BST’s as well. Vectors are good for their fast insertion and simple structure, but aren't as good for sorting and still inefficient at gathering prereqs.

Overall, I think the academic advisors should go for BSTs as they can provide both the sorted order and quick lookups for course details.