**Project One - Pseudocode and Analysis**

**Pseudocode for Menu**

DisplayMenu():

Display "Menu Options:"

Display "1. Load file data into data structure"

Display "2. Print an alphanumerically ordered list of Computer Science courses"

Display "3. Print course title and prerequisites"

Display "9. Exit program"

choice = GetUserInput()

if choice == 1:

LoadFileData()

else if choice == 2:

PrintAlphanumericCourseList()

else if choice == 3:

PrintCourseAndPrerequisites()

else if choice == 9:

ExitProgram()

else:

Display "Invalid choice. Please select a valid option."

DisplayMenu()

**Pseudocode for Printing Alphanumeric Course List**

**Vector Data Structure:**

PrintAlphanumericCourseList():

Sort courses in vector by alphanumeric course number

for each course in vector:

Print course information

**Hash Table Data Structure**

PrintAlphanumericCourseList():

Create a sorted list from hash table keys (course numbers)

Sort the list alphabetically

for each course number in sorted list:

Retrieve course information from hash table using course number

Print course information

**Tree Data Structure**

PrintAlphanumericCourseList(node):

if node is not null:

PrintAlphanumericCourseList(node.left)

Print course information for node

PrintAlphanumericCourseList(node.right)

**Runtime analysis for opening file, reading data, parsing, and error checking**

| **Code Line** | **Line cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open the file | 1 | 1 | 1 |
| Read each line from the file | n | n | n |
| Close the file | 1 | 1 | 1 |
| Split line by comma | 1 | n | n |
| Length check and error message | 1 | n | n |
| Extract courseNumber, courseTitle, prerequisites | 1 | n | n |
| Store tuple in list | 1 | n | n |
| Validate prerequisites | n\*k | n | n\*k |
| Insert course into tree | logn | n | nlogn |
| **Total Cost** |  |  | **nlogn + 5n + 2 + n\*k** |

**Runtime analysis for creating course objects and storing in data structure:**

| **Code Line** | **Line cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Split line by comma | 1 | n | n |
| Length check and error message | 1 | n | n |
| Extract courseNumber, courseTitle, prerequisites | 1 | n | n |
| Create Course object | 1 | n | n |
| Insert course into tree | logn | n | nlogn |
| **Total Cost** |  |  | **4n + nlogn** |

**Advantage and disadvantage of each structure:**

The Vector data structure is simple to use and works well for tasks where data needs to be sorted or accessed in order. It's great for situations where you need to go through data one by one. However, a downside is that adding or removing elements can be slow if the Vector needs to resize, which can slow down performance when handling dynamic data.

The Hash Table data structure is excellent for quickly looking up data using a unique identifier, like a course number. It's very efficient for adding and deleting data. However, Hash Tables can use more memory than other structures due to the extra space needed for the hash table. Also, sorting data in a Hash Table requires additional steps because it doesn't keep the elements in order.

The Tree data structure is great for keeping data in a sorted order which makes it very efficient for tasks that need ordered data retrieval or traversal. This built in order makes searching and accessing sorted data faster. However, Trees are more complex to set up and manage compared to simpler structures like Vectors or Hash Tables. They can also slow down for adding or deleting elements if the tree becomes unbalanced, which can affect performance.

**Recommendation:**

In this program, sorting and displaying courses alphabetically are key tasks. I would use a tree data structure because it keeps the course data automatically sorted, so no extra sorting is needed. It also provides fast and efficient access, updates, and searches even as the number of courses increases.