##### Menu Pseudocode

1. Display the following options to the user:
   * 1: Load the file data into the data structure
   * 2: Print an alphanumerically ordered list of all courses
   * 3: Print the course title and the prerequisites for a specific course
   * 9: Exit the program
2. Read the user's choice.
3. Based on the user's choice, perform the corresponding action:
   * If the user chooses 1:
     + Call the function to load the file data into the data structure.
   * If the user chooses 2:
     + Call the function to print an alphanumerically ordered list of all courses.
   * If the user chooses 3:
     + Prompt the user to enter the course number.
     + Call the function to print the course title and prerequisites for the given course number.
   * If the user chooses 9:
     + Exit the program.
   * If the user enters an invalid option:
     + Display an error message and show the menu again.

##### Pseudocode for Loading Data into a Vector

1. Define a Course Structure:
   * A course has the following attributes:
     + courseNumber: String
     + courseTitle: String
     + prerequisites: List of Strings
2. Load Data Function:
   * Open the file named "courses.txt" for reading.
   * If the file can't be opened, display an error message and stop the program.
   * Create an empty list called courseList.
   * While there are lines to read in the file:
     + Read a line from the file.
     + Split the line into parts using commas.
     + If the line has fewer than two parts, display an error message and stop the program.
     + Extract the course number, title, and prerequisites.
     + Create a new course object with the extracted data.
     + Add the course object to the list.
   * Close the file.
3. Print Sorted List Function:
   * Sort the courseList by courseNumber.
   * For each course in the sorted list:
     + Print the course number and title.
4. Print Course Information Function:
   * Prompt the user to enter a course number.
   * Search the courseList for the course with the given number.
   * If the course is found:
     + Print the course number, title, and prerequisites.
   * If the course is not found, print "Course not found."

##### Pseudocode for Loading Data into a Hash Table

1. Define a Course Structure:
   * A course has the following attributes:
     + courseNumber: String
     + courseTitle: String
     + prerequisites: List of Strings
2. Load Data Function:
   * Open the file named "courses.txt" for reading.
   * If the file can't be opened, display an error message and stop the program.
   * Create an empty hash table called courseTable.
   * While there are lines to read in the file:
     + Read a line from the file.
     + Split the line into parts using commas.
     + If the line has fewer than two parts, display an error message and stop the program.
     + Extract the course number, title, and prerequisites.
     + Create a new course object with the extracted data.
     + Add the course object to the hash table with the course number as the key.
   * Close the file.
3. Print Sorted List Function:
   * Create a list of all courses from the hash table.
   * Sort the list by courseNumber.
   * For each course in the sorted list:
     + Print the course number and title.
4. Print Course Information Function:
   * Prompt the user to enter a course number.
   * Search the hash table for the course with the given number.
   * If the course is found:
     + Print the course number, title, and prerequisites.
   * If the course is not found, print "Course not found."

##### Pseudocode for Loading Data into a Binary Search Tree (BST)

1. Define a Course Structure:
   * A course has the following attributes:
     + courseNumber: String
     + courseTitle: String
     + prerequisites: List of Strings
2. Load Data Function:
   * Open the file named "courses.txt" for reading.
   * If the file can't be opened, display an error message and stop the program.
   * Create an empty binary search tree called courseTree.
   * While there are lines to read in the file:
     + Read a line from the file.
     + Split the line into parts using commas.
     + If the line has fewer than two parts, display an error message and stop the program.
     + Extract the course number, title, and prerequisites.
     + Create a new course object with the extracted data.
     + Insert the course object into the binary search tree with the course number as the key.
   * Close the file.
3. Print Sorted List Function:
   * Traverse the binary search tree in-order.
   * For each course in the traversal:
     + Print the course number and title.
4. Print Course Information Function:
   * Prompt the user to enter a course number.
   * Search the binary search tree for the course with the given number.
   * If the course is found:
     + Print the course number, title, and prerequisites.
   * If the course is not found, print "Course not found."

#### Runtime Analysis

1. Vector:
   * Load Data: O(n) to read and add courses.
   * Print Sorted List: O(n log n) to sort and O(n) to print.
   * Print Course Information: O(n) for linear search.
2. Hash Table:
   * Load Data: O(n) to read and add courses.
   * Print Sorted List: O(n) to gather courses, O(n log n) to sort, and O(n) to print.
   * Print Course Information: O(1) for hash table lookup.
3. Binary Search Tree (BST):
   * Load Data: O(n log n) to insert courses.
   * Print Sorted List: O(n) for in-order traversal.
   * Print Course Information: O(log n) for search.

#### Advantages and Disadvantages

1. Vector:
   * Advantages: Simple to implement, direct indexing.
   * Disadvantages: Inefficient search and sorting for large datasets.
2. Hash Table:
   * Advantages: Fast lookups, average O(1) search time.
   * Disadvantages: No inherent order, requires additional steps to sort.
3. Binary Search Tree (BST):
   * Advantages: Keeps elements sorted, efficient search and traversal.
   * Disadvantages: Can become unbalanced, leading to O(n) worst-case operations.

#### Recommendation

Based on the runtime analysis, the hash table is recommended for its average O(1) search time, making it highly efficient for frequent course lookups. Although it requires additional steps to sort for printing, its overall performance for large datasets is superior. The binary search tree is a close second due to its balanced efficiency in both searching and sorting.