6-2 Project One: Pseudocode and Runtime Analysis

Sylvia Davis  
Southern New Hampshire University

CS-300-10554-M01 DSA: Analysis and Design

Prof. Aline Yurik

June 11, 2025

**Pseudocode – Vector**

OpenFile(fileName)  
 Open the file for reading  
 For each line in file  
 Split line by comma into tokens  
 If token count < 2  
 Display error message  
 Continue to next line  
 For each token in tokens[2:]  
 Check if token is a valid course in file  
 Create Course object with course number, title, and prerequisites  
 Append Course object to vector  
Close file

PrintCourse(courseNumber)  
 For course in vector  
 If course.courseNumber == courseNumber  
 Print course.title  
 Print course.prerequisites

**Pseudocode – Hash Table**

OpenFile(fileName)  
 Open the file for reading  
 For each line in file  
 Split line by comma into tokens  
 If token count < 2  
 Display error message  
 Continue to next line  
 For each token in tokens[2:]  
 Check if token is a valid course in file  
 Create Course object  
 Insert Course into Hash Table using courseNumber as key  
Close file

PrintCourse(courseNumber)  
 Use courseNumber to retrieve Course from Hash Table  
 If found  
 Print course.title and prerequisites

**Pseudocode – Binary Search Tree**

OpenFile(fileName)  
 Open file for reading  
 For each line in file  
 Split line into tokens  
 If token count < 2  
 Display error  
 Continue  
 Create Course object  
 Insert course into BST based on courseNumber  
Close file

PrintCourse(courseNumber)  
 Search BST for courseNumber  
 If found, display title and prerequisites

**Pseudocode – Menu System**

DisplayMenu()  
 Display Options:  
 1. Load course data  
 2. Print course list (sorted)  
 3. Print course information  
 9. Exit  
 Read user choice  
 If choice == 1  
 OpenFile and load data  
 Else if choice == 2  
 Print sorted course list  
 Else if choice == 3  
 Prompt for course number  
 Call PrintCourse(courseNumber)  
 Else if choice == 9  
 Exit

**Runtime Analysis Chart**

|  |  |  |  |
| --- | --- | --- | --- |
| Data Structure | Load Time (Big O) | Search Time (Big O) | Space Complexity |
| Vector | O(n) | O(n) | O(n) |
| Hash Table | O(n) | O(1) | O(n) |
| Binary Search Tree | O(n log n) | O(log n) | O(n) |

**Evaluation and Recommendation**

The vector offers a straightforward and simple implementation but is less efficient in searching due to its linear time complexity. Hash tables provide constant time access, which is optimal for retrieving individual course information, but they do not maintain order, making it more difficult to print a sorted course list. Binary search trees offer a balanced trade-off: they provide logarithmic time for both insertion and retrieval and support ordered traversal.  
  
Given the requirements to both retrieve specific course information and print courses in alphanumeric order, the binary search tree is the most appropriate structure. It allows for efficient lookup and maintains order without the complexity of sorting the entire list after insertion.  
  
Therefore, I recommend using the Binary Search Tree for the final implementation.