**CS 300 6-2 Project One**

Caleb Leavell

caleb.leavell@snhu.edu

Southern New Hampshire University

**Pseudocode**

**Menu**

BEGIN Menu

PRINT "Select an option:"

PRINT "1: Load course data"

PRINT "2: Print all courses in alphanumeric order"

PRINT "3: Print course title and prerequisites"

PRINT "9: Exit"

READ userOption

IF userOption == 1 THEN

CALL LoadCourseData()

ELSE IF userOption == 2 THEN

CALL PrintAllCourses()

ELSE IF userOption == 3 THEN

CALL PrintCourseDetails()

ELSE IF userOption == 9 THEN

EXIT

ELSE

PRINT "Invalid option. Please select again."

END IF

END Menu

**Load Course Data**

BEGIN LoadCourseData

OPEN "courses.txt"

WHILE NOT EndOfFile("courses.txt") DO

READ line FROM "courses.txt"

PARSE line INTO courseNumber, courseTitle, prerequisites

CREATE Course object with courseNumber, courseTitle, prerequisites

ADD Course object TO dataStructure

END WHILE

CLOSE "courses.txt"

END LoadCourseData

**Print All Courses in Alphanumeric Order (for Vector, Hash Table, Tree)**

**Vector**

BEGIN PrintAllCoursesVector

SORT vector BY courseNumber

FOR EACH course IN vector DO

PRINT course.courseNumber, course.courseTitle

END FOR

END PrintAllCoursesVector

**Hash Table**

BEGIN PrintAllCoursesHashTable

CREATE list FROM hashTable values

SORT list BY courseNumber

FOR EACH course IN list DO

PRINT course.courseNumber, course.courseTitle

END FOR

END PrintAllCoursesHashTable

**Tree**

BEGIN PrintAllCoursesTree

CALL InOrderTraversal(treeRoot)

END PrintAllCoursesTree

FUNCTION InOrderTraversal(node)

IF node IS NOT NULL THEN

CALL InOrderTraversal(node.left)

PRINT node.courseNumber, node.courseTitle

CALL InOrderTraversal(node.right)

END IF

END FUNCTION

**Print Course Details**

BEGIN PrintCourseDetails

PRINT "Enter the course number:"

READ courseNumber

course = FIND course IN dataStructure WITH courseNumber

IF course IS NOT NULL THEN

PRINT "Course Title:", course.courseTitle

PRINT "Prerequisites:", course.prerequisites

ELSE

PRINT "Course not found."

END IF

END PrintCourseDetails

**Runtime Analysis Charts**

**Load Course Data**

|  |  |  |
| --- | --- | --- |
| **Data Structure** | **Load Data Time Complexity** | **Total Complexity** |
| **Vector** | O(n) | O(n) |
| **Hash Table** | O(n) | O(n) |
| **Tree** | O(n log n) | O(n log n) |

**Print All Courses in Alphanumeric Order**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Structure** | **Sorting Time Complexity** | **Printing Time Complexity** | **Total Complexity** |
| **Vector** | O(n log n) | O(n) | O(n log n) |
| **Hash Table** | O(n log n) | O(n) | O(n log n) |
| **Tree** | O(n) | O(n) | O(n) |

**Print Course Details**

|  |  |
| --- | --- |
| **Data Structure** | **Lookup Time Complexity** |
| **Vector** | **O(n)** |
| **Hash Table** | **O(1)** |
| **Tree** | **O(log n)** |

**Analysis of Advantages and Disadvantages**

**Vector**

* **Advantages**:
  + Simple to implement.
  + Fast sequential access.
* **Disadvantages**:
  + Inefficient for searches: O(n).
  + Sorting required for ordered print: O(n log n).

**Hash Table**

* **Advantages**:
  + Very fast lookups: O(1) on average.
  + Efficient insertion: O(1).
* **Disadvantages**:
  + Sorting required for ordered print: O(n log n).
  + Inefficient memory usage if the load factor is not managed.

**Tree**

* **Advantages**:
  + Efficient ordered traversal: O(n).
  + Reasonable lookup time: O(log n).
* **Disadvantages**:
  + More complex to implement.
  + Insertions can be slower: O(log n).

**Recommendation**

Based on the runtime analysis, I recommend using a **Hash Table** for the following reasons:

* It provides the fastest average-case insertion and lookup times: O(1).
* The complexity of sorting for alphanumeric order is balanced by its fast lookup capability.
* Memory usage is efficient for large datasets, assuming proper load factor management.