CS 300 Final Project Part I: Pseudocode and Runtime Analysis

**Pseudocode Section**

**Vector Implementation**

**1. Load and Parse File:**

FUNCTION LoadCourses(filename):

OPEN file

IF file fails THEN

DISPLAY "Error"

RETURN

CREATE empty vector courseLines

FOR each line in file DO

SPLIT line into tokens

IF token count < 2 THEN

DISPLAY "Invalid line"

CONTINUE

ADD line to courseLines

CLOSE file

**2. Create Course Objects:**

FUNCTION ParseCourses(courseLines):

CREATE empty vector courses

FOR each line in courseLines DO

SPLIT line into tokens

SET id = tokens[0]

SET title = tokens[1]

CREATE empty vector prerequisites

FOR i = 2 to tokens.length - 1 DO

ADD tokens[i] to prerequisites

CREATE Course object

SET fields using id, title, prerequisites

ADD course to courses

RETURN courses

**3. Print Course Info (Vector):**

FUNCTION PrintCourse(courses, id):

FOR each course in courses DO

IF course.id == id THEN

DISPLAY course.id, title, prerequisites

RETURN

DISPLAY "Course not found"

**4. Sort and Print All Courses:**

FUNCTION PrintAllCourses(courses):

SORT courses by id

FOR each course in sorted list DO

DISPLAY course.id, course.title

**Hash Table Implementation**

**1. Load and Parse File:** *(Same as Vector)*

**2. Create Course Objects in Hash Table:**

FUNCTION ParseCoursesToHash(courseLines):

CREATE empty hashTable

FOR each line in courseLines DO

SPLIT line into tokens

SET id = tokens[0], title = tokens[1]

CREATE empty list prereqs

FOR i = 2 to tokens.length - 1 DO

ADD tokens[i] to prereqs

CREATE Course object

SET fields

INSERT course INTO hashTable with key = id

RETURN hashTable

**3. Print Course Info (Hash Table):**

FUNCTION PrintCourse(hashTable, id):

IF id in hashTable THEN

DISPLAY course info

ELSE

DISPLAY "Course not found"

**4. Print All Courses:**

FUNCTION PrintAllCourses(hashTable):

CREATE vector from hashTable values

SORT vector by id

FOR each course in sorted list DO

DISPLAY course.id, title

**Tree Implementation**

**1. Load and Parse File:** *(Same as Vector)*

**2. Create Course Objects in Tree:**

FUNCTION ParseCoursesToTree(courseLines):

CREATE empty BST courseTree

FOR each line in courseLines DO

SPLIT line into tokens

SET id = tokens[0], title = tokens[1]

CREATE empty list prereqs

FOR i = 2 to tokens.length - 1 DO

ADD tokens[i] to prereqs

CREATE Course object

SET fields

INSERT course INTO courseTree

RETURN courseTree

**3. Print Course Info (Tree):**

FUNCTION PrintCourse(courseTree, id):

SEARCH course in courseTree by id

IF found THEN

DISPLAY course info

ELSE

DISPLAY "Course not found"

**4. In-Order Print of All Courses:**

FUNCTION PrintAllCourses(courseTree):

PERFORM in-order traversal

DISPLAY course.id, title for each

**Menu System (All Structures)**

REPEAT UNTIL user chooses 9:

DISPLAY menu options

GET user input

IF input = 1 THEN Load and Parse file

IF input = 2 THEN Print all courses

IF input = 3 THEN Prompt for ID, Print course info

IF input = 9 THEN Exit

| **Operation** | **Vector** | **Hash Table** | **Tree (BST)** |
| --- | --- | --- | --- |
| Load & Parse File | O(n) | O(n) | O(n log n) |
| Insert Course | O(1) append | O(1) avg | O(log n) avg |
| Search by ID | O(n) | O(1) avg | O(log n) avg |
| Print All (Sorted) | O(n log n) | O(n log n) | O(n) |
| Memory Usage | Low | Medium | High |

**Analysis and Recommendation**

* **Vector:** Easiest to use and low memory, but search and sort are slow for large datasets.
* **Hash Table:** Extremely fast lookup and insert, but data must be extracted and sorted for ordered output.
* **Tree (BST):** Maintains order naturally and supports fast search/insert, but uses more memory and requires careful memory management.

**Recommendation:** Use the **Binary Search Tree (BST)**. It provides the best performance for both inserting and retrieving courses in sorted order without needing extra operations. BSTs offer efficient average-case performance for all operations needed in ABCU’s advising program, making them the most balanced and scalable choice.