Pseudocode for All Data Structures

1. Vector Pseudocode

```
FUNCTION loadData(fileName):
    OPEN fileName FOR READING
    IF file cannot be opened:
        PRINT "Error: File not found."
        RETURN
    DECLARE courses AS VECTOR OF Course
    FOR each line in file:
        SPLIT line BY ',' INTO tokens
        IF number of tokens < 2:
            PRINT "Error: Invalid line format."
            CONTINUE
        SET courseNumber = tokens[0]
        SET courseTitle = tokens[1]
        SET prerequisites = tokens[2 TO END]
        // Check if all prerequisites exist in the dataset
        FOR each prerequisite IN prerequisites:
            IF prerequisite NOT IN courses:
```

```
PRINT "Error: Prerequisite " + prerequisite + " not found for course " + courseNumber
CONTINUE
// Create course object and add to vector
CREATE courseObject WITH courseNumber, courseTitle, prerequisites
ADD courseObject TO courses
CLOSE file
RETURN courses
FUNCTION printAllCourses(courses):
SORT courses BY courseNumber IN ASCENDING ORDER
FOR each course IN courses:
PRINT course.courseNumber + " | " + course.courseTitle
IF course.prerequisites IS NOT EMPTY:
PRINT "Prerequisites: " + JOIN(course.prerequisites, ", ")
ELSE:
PRINT "No prerequisites."

FUNCTION searchCourse(courses, courseNumber):
FOR each course IN courses:
IF course.courseNumber == courseNumber:
```

```
PRINT course.courseNumber + " | " + course.courseTitle
IF course.prerequisites IS NOT EMPTY:
PRINT "Prerequisites: " + JOIN(course.prerequisites, ", ")
ELSE:
PRINT "No prerequisites."
RETURN
PRINT "Course not found."
FUNCTION main():
DECLARE courses AS VECTOR OF Course
WHILE TRUE:
PRINT "1. Load Data"
PRINT "2. Print All Courses"
PRINT "3. Search Course"
PRINT "9. Exit"
READ userInput

IF userInput == 1:
PRINT "Enter the file name:"
READ fileName
courses = loadData(fileName)
```

```
ELSE IF userInput == 2:
    printAllCourses(courses)
ELSE IF userInput == 3:
    PRINT "Enter Course Number:"
    READ courseNumber
    searchCourse(courses, courseNumber)
ELSE IF userInput == 9:
    EXIT
ELSE:
    PRINT "Invalid option."
```

2. Hash Table Pseudocode

```
FUNCTION loadData(fileName):
    OPEN fileName FOR READING
    IF file cannot be opened:
        PRINT "Error: File not found."
        RETURN
    DECLARE courses AS HASH TABLE
    DECLARE courseList AS LIST
    FOR each line in file:
        SPLIT line BY ',' INTO tokens
        IF number of tokens < 2:
            PRINT "Error: Invalid line format."
            CONTINUE
        SET courseNumber = tokens[0]
        SET courseTitle = tokens[1]
        SET prerequisites = tokens[2 TO END]
        // Check if all prerequisites exist in the dataset
        FOR each prerequisite IN prerequisites:
            IF prerequisite NOT IN courses:
                PRINT "Error: Prerequisite " + prerequisite + " not found for course " + courseNumber
                CONTINUE
        // Create course object and insert into hash table
        CREATE courseObject WITH courseNumber, courseTitle, prerequisites
        INSERT courseObject INTO courses WITH KEY courseNumber
        ADD courseObject TO courseList
    CLOSE file
    RETURN courses, courseList
FUNCTION printAllCourses(courseList):
    SORT courseList BY courseNumber IN ASCENDING ORDER
    FOR each course IN courseList:
        PRINT course.courseNumber + " | " + course.courseTitle
        IF course.prerequisites IS NOT EMPTY:
            PRINT "Prerequisites: " + JOIN(course.prerequisites, ", ")
        ELSE:
            PRINT "No prerequisites."
FUNCTION searchCourse(courses, courseNumber):
    IF courses CONTAINS KEY courseNumber:
        SET course = courses[courseNumber]
        PRINT course.courseNumber + " | " + course.courseTitle
        IF course.prerequisites IS NOT EMPTY:
            PRINT "Prerequisites: " + JOIN(course.prerequisites, ", ")
        ELSE:
            PRINT "No prerequisites."
    ELSE:
        PRINT "Course not found."
FUNCTION main():
    DECLARE courses AS HASH TABLE
    DECLARE courseList AS LIST
```

```
WHILE TRUE:
    PRINT "1. Load Data"
    PRINT "2. Print All Courses"
    PRINT "3. Search Course"
    PRINT "9. Exit"
    READ userInput
    IF userInput == 1:
        PRINT "Enter the file name:"
        READ fileName
        courses, courseList = loadData(fileName)
    ELSE IF userInput == 2:
        printAllCourses(courseList)
    ELSE IF userInput == 3:
        PRINT "Enter Course Number:"
        READ courseNumber
        searchCourse(courses, courseNumber)
    ELSE IF userInput == 9:
        EXIT

    ELSE:
```

```
PRINT "Invalid option."
```

3. Binary Search Tree (BST) Pseudocode

```
STRUCTURE Course:
    STRING courseNumber
    STRING courseTitle
    LIST OF STRINGS prerequisites

STRUCTURE Node:
    Course course
    Node left
    Node right

FUNCTION insertNode(Node root, Course course):
    IF root IS NULL:
        RETURN NEW Node(course)
    IF course.courseNumber < root.course.courseNumber:
        root.left = insertNode(root.left, course)
    ELSE:
        root.right = insertNode(root.right, course)
    RETURN root

FUNCTION loadData(fileName):
```

```
OPEN fileName FOR READING
IF file cannot be opened:
    PRINT "Error: File not found."
    RETURN
DECLARE root AS Node = NULL
FOR each line in file:
    SPLIT line BY ',' INTO tokens
    IF number of tokens < 2:
        PRINT "Error: Invalid line format."
        CONTINUE
    SET courseNumber = tokens[0]
    SET courseTitle = tokens[1]
    SET prerequisites = tokens[2 TO END]

    // Check if all prerequisites exist in the dataset
    FOR each prerequisite IN prerequisites:
        IF prerequisite NOT IN root:
            PRINT "Error: Prerequisite " + prerequisite + " not found for course " + courseNumber
            CONTINUE
        // Create course object and insert into BST
        CREATE courseObject WITH courseNumber, courseTitle, prerequisites
        root = insertNode(root, courseObject)
    CLOSE file
    RETURN root

FUNCTION inOrderTraversal(Node root):
    IF root IS NOT NULL:
        inOrderTraversal(root.left)
        PRINT root.course.courseNumber + " | " + root.course.courseTitle
        IF root.course.prerequisites IS NOT EMPTY:
            PRINT "Prerequisites: " + JOIN(root.course.prerequisites, ", ")
        ELSE:
            PRINT "No prerequisites."
        inOrderTraversal(root.right)

FUNCTION searchCourse(Node root, STRING courseNumber):
    IF root IS NULL:
        PRINT "Course not found."
        RETURN
    IF courseNumber == root.course.courseNumber:
        PRINT root.course.courseNumber + " | " + root.course.courseTitle
```

```
IF root.course.prerequisites IS NOT EMPTY:
PRINT "Prerequisites: " + JOIN(root.course.prerequisites, ", ")
ELSE:
PRINT "No prerequisites."
ELSE IF courseNumber < root.course.courseNumber:
searchCourse(root.left, courseNumber)
ELSE:
searchCourse(root.right, courseNumber)


FUNCTION main():
DECLARE root AS Node = NULL
WHILE TRUE:
PRINT "1. Load Data"
PRINT "2. Print All Courses"
PRINT "3. Search Course"
PRINT "9. Exit"
READ userInput
IF userInput == 1:
PRINT "Enter the file name:"
READ fileName
```

```
    root = loadData(fileName)
ELSE IF userInput == 2:
    inOrderTraversal(root)
ELSE IF userInput == 3:
    PRINT "Enter Course Number:"
    READ courseNumber
    searchCourse(root, courseNumber)
ELSE IF userInput == 9:
    EXIT
ELSE:
    PRINT "Invalid option."
```

Runtime Analysis

For the analysis of the runtime, first, I looked at how long each of the operations took for the vector, hash table, and BST. I used the Big O notation, which tells us how the runtime grows as the number of the course (n) increases. After all the analysis this is what I found:

1. Loading Data:

This is the part where we read the file, create course objects, and also to store all of them in the data structure.

- Vector: Reading the file and creating course objects takes O(n) time because we process each course once.
- Hash Table: Similar to the vector, reading the file and inserting courses into the hash table takes O(n) time.
- BST: Inserting each course into the BST takes O(log n) time on average. Since we insert n courses, the total time is O(n log n).

2. Sorting:

The next part is where we sort the courses in alphanumeric order.

- Vector: Sorting the vector takes O(n log n) time because we use an efficient sorting algorithm like merge sort or quicksort.
- Hash Table: To sort the courses, we first extract them from the hash table (which takes O(n) time) and then sort them (which takes O(n log n) time). So the total time is O(n log n).
- BST: The BST is already sorted because of its structure and printing the courses in order takes O(n) time (in-order traversal).

3. Searching:

This part is where we search for a specific course by its course number or the course ID.

- Vector: Searching in a vector takes O(n) time because, in the worst case, we might have to check every course.
- Hash Table: On average, searching in a hash table takes O(1) time because we can directly access the course using its course number as the key.
- BST: Searching in a BST takes O(log n) time on average because we divide the search space in half with each step.

Summary Table:

Operations	Vector	Hash Table	BST
Loading Data	O(n)	O(n)	O(n log n)
Sorting	O(n log n)	O(n log n)	O(n) (already sorted)
Searching	O(n)	O(1) average, O(n) worst case	O(log n) average, O(n) worst case

So, for this, I assumed that each line of the pseudo-code takes 1 unit of time unless it calls a function and if it does call a function the cost is the run time of that function. For example, inserting a course into a BST takes O(log n) time, so I counted that as the cost for that line. I also analyzed the worst-case runtime for each operation. For example, O(n) time can be taken for searching in a vector in a worse case if the course we are looking for is at the end of the vector.

This all matters because vector is very easy and simple to use but at the same time it is slow in terms of searching and it also requires sorting. Moving on to the Hash table it is fast for searching but at the same time, it doesn't keep the course sorted. BST automatically sorts the courses and is efficient for searching which is important for this program.

Advantages and Disadvantages

Data Structure	Advantages	Disadvantages
Vector	Simple to implement, easy to sort	Slow for searching (O(n)), requires sorting after loading
Hash Table	Fast search time (O(1) on average), scalable for large datasets	Collisions can degrade search performance, requires sorting for ordered output
BST	Automatically sorted, efficient search (O(log n) on average)	Slower insertion time (O(log n)), can become unbalanced in worst case

Recommendation

I recommended using the Hash Table for this program because I feel like it’s the fastest among the others in terms of searching, taking only O(1) time on average to find a course. This is perfect for quickly showing courses, sorting can be done efficiently also in O(n log n) time. Vector in terms of searching is slow and in the BST it is slow for inserting courses while sorted. In conclusion, the hash table is the best choice because it is fast, scalable, and balances performance well for this application.