Connor Martin  
CS 300 Project One  
10/6/2025

**Pseudocode and Runtime Analysis**

**Data I’m using**

Course has: courseNumber, title, prerequisites (list of strings)

**Vector Pseudocode**

**LoadCourses(filename)**

Open the file.  
 If the file won’t open, print an error and stop.

Set courses = empty vector of Course.  
 Set seen = empty map from courseNumber → index in courses.  
 Set errors = empty list of strings.

For each line in the file:  
 Split by commas into tokens, trim spaces.  
 If tokens.size < 2, add “format error” to errors and continue.  
 Set courseNum = tokens[0].  
 Set title = tokens[1].  
 Set prereqList = tokens[2..end] (may be empty).  
 Make new Course(courseNum, title, prereqList).  
 Push course into courses.  
 Set seen[courseNum] = courses.size − 1.

For each course c in courses:  
 For each p in c.prerequisites:  
 If p not in seen, add “missing prerequisite p for c.courseNumber” to errors.

If errors is not empty, print each error.  
 Else print “file validated successfully”.

**SearchCourse(courses, target)**

For each course in courses:  
 If course.courseNumber == target:  
 Print course.courseNumber and course.title.  
 If course.prerequisites is empty, print “no prerequisites”.  
 Else print each prerequisite.  
 Stop.

If not found, print “course not found”.

**PrintAllCourses(courses)**

Sort courses by courseNumber ascending.  
 For each course in courses, print courseNumber and title.

**Hash Table Pseudocode**

**LoadCourses(filename)**

Open the file.  
If the file cannot be opened, print an error and stop.

Set courses = empty hash table.  
Set seen = empty set of courseNumbers.  
Set errors = empty list.

For each line in the file:  
 Read and split the line by commas into tokens.  
 If the number of tokens is less than 2, add a format error and continue.  
 Set courseNum = tokens[0].  
 Set title = tokens[1].  
 Set prereqs = tokens[2..end] (may be empty).  
 Make new Course(courseNum, title, prereqs).  
 Insert Course into hash table using courseNum as the key.  
 Add courseNum to seen.

For each course in courses:  
 For each p in course.prerequisites:  
 If p not in seen, add an error for missing prerequisite.

If errors is not empty, print each error.  
 Else print “file validated successfully”.

**SearchCourse(hashTable, target)**

Look up target in hashTable.  
 If not found, print “course not found”.  
 Else print courseNumber and title.  
 If prerequisites is empty, print “no prerequisites”.  
 Else print each prerequisite.

**PrintAllCourses(hashTable)**

Get all course keys from hashTable.  
 Sort the keys alphanumerically.  
 For each key in sorted keys, print courseNumber and title.

**Binary Search Tree (BST) Pseudocode**

**LoadCourses(filename)**

Open the file.  
 If the file cannot be opened, print an error and stop.

Set tree = empty Binary Search Tree (key: courseNumber → Course).  
 Set errors = empty list of strings.

For each line in the file (skip header if present):  
 Trim whitespace; if line is empty, continue.  
 Split the line by commas into tokens.  
 If len(tokens) < 2, append “Format error: missing fields” to errors and continue.  
 Set courseNum = tokens[0].  
 Set title = tokens[1].  
 Set prereqs = tokens[2..end] (may be empty).  
 Make new Course(courseNum, title, prereqs).  
 Call InsertCourse(tree, course).

If errors is not empty, print each error.  
 Else print “file validated successfully”.

**InsertCourse(tree, course)**

If tree.root is null, set tree.root = new Node(course) and return.

Set current = tree.root.  
 Loop:  
 If course.courseNumber < current.course.courseNumber:  
 If current.left is null, set current.left = new Node(course) and return.  
 Else set current = current.left.  
 Else:  
 If current.right is null, set current.right = new Node(course) and return.  
 Else set current = current.right.

**SearchCourse(tree, target)**

Set current = tree.root.  
 While current is not null:  
 If target == current.course.courseNumber, print course info and prerequisites; return.  
 If target < current.course.courseNumber, set current = current.left.  
 Else set current = current.right.  
 Print “course not found”.

**PrintAllCourses(tree)**

Call InOrderTraversal(tree.root).

**InOrderTraversal(node)**  
 If node is null, return.  
 InOrderTraversal(node.left).  
 Print node.course.courseNumber and node.course.title.  
 InOrderTraversal(node.right).

Menu (for all structures)

Print:

1. Load data
2. Print all courses (alphanumeric)
3. Print single course
4. Exit

Loop until exit:  
 Read user choice.  
 If 1, load data from file.  
 If 2, print all courses in order.  
 If 3, search for a course and print it.  
 If 9, exit.  
 Else print “invalid option”.

Runtime Analysis (Worst Case)

|  |  |  |
| --- | --- | --- |
| **Data Structure** | **Build Time (insert all n)** | **Memory Use and Notes** |
| Vector | O(n) total (O(1) amortized append) | Contiguous storage; low overhead |
| Hash Table | O(n) average, O(n²) worst if collisions | Extra memory for buckets |
| Binary Search Tree | O(n log n) average, O(n²) worst if sorted input | Node pointers per element |

**Advantages and Disadvantages**

**Vector**  
 Simple and memory efficient.  
 Easy to sort for alphanumeric output.  
 Lookup for a single course is O(n).

**Hash Table**  
 Fast direct lookup (O(1) average).  
 Not naturally ordered, must sort keys for output.  
 Uses more memory for buckets.

**Binary Search Tree**  
 In-order traversal naturally sorts output (O(n)).  
 Lookup and insert average O(log n).  
 Can degrade to O(n²) if tree becomes unbalanced.

**Recommendation**

The **Binary Search Tree** is the best option.  
 It supports efficient course lookup (O(log n)) and naturally produces the course list in alphanumeric order using in-order traversal (O(n)).  
 If a balanced BST like AVL or Red-Black were used, all operations would stay O(log n).  
 The Hash Table is a close second for lookup speed but needs extra sorting for output.