CS 300 – Final Project Part I: Pseudocode and Runtime Analysis

Student: Nicole Hutto  
Course: CS 300 – Data Structures and Algorithms

# Pseudocode Summary

## Vector

* - Load file and store course objects in a vector after validating format and prerequisites.
* - Sort the vector and print all course information.
* - Search the vector linearly to print a single course and its prerequisites.

## Hash Table

* - Load course data into a hash table using course numbers as keys.
* - Efficiently look up a course and print its details using constant-time access.
* - Requires additional logic to sort courses if needed.

## Binary Search Tree (BST)

* - Insert course objects into the tree by course number.
* - Use in-order traversal to print courses in sorted order.
* - Search for a course using log(n) comparisons in a balanced tree.

# Pseudocode for Each Data Structure

## 1. Vector

### Load and Parse File

Function LoadCourses(filePath)  
 Open file  
 If cannot open:  
 Print "Error"  
 Exit  
 For each line in file:  
 Split line by comma into tokens  
 If tokens.length < 2:  
 Print "Invalid format"  
 Continue  
 Add tokens[0] to knownCourses  
 Store tokens in tempCourseList  
  
 For each line in tempCourseList:  
 Extract courseNumber, title, prerequisites  
 If any prerequisite not in knownCourses:  
 Print "Invalid prerequisite"  
 Skip this course  
 Create Course object with number, title, prerequisites  
 Append to courseVector

### Print All Courses (Sorted)

Function PrintAllCourses(vector)  
 Sort vector by courseNumber  
 For each course in vector:  
 Print courseNumber, name, prerequisites (if any)

### Print Single Course

Function SearchCourse(vector, input)  
 For each course in vector:  
 If course.courseNumber == input:  
 Print course details  
 Return  
 Print "Course not found"

## 2. Hash Table

### Load and Parse File

Function LoadCourses(filePath)  
 Open file  
 If cannot open:  
 Print "Error"  
 Exit  
 Initialize HashTable courses  
 For each line in file:  
 Split into tokens  
 If tokens.length < 2:  
 Print "Invalid format"  
 Continue  
 For each prerequisite:  
 If not in knownCourses:  
 Print "Invalid prerequisite"  
 Continue  
 Create Course object  
 Insert into HashTable using courseNumber as key

### Print All Courses

Function PrintAllCourses(hashTable)  
 For each bucket in hashTable:  
 For each course:  
 Print courseNumber, name, prerequisites

### Print Single Course

Function SearchCourse(hashTable, input)  
 If input exists in hashTable:  
 Print course details  
 Else:  
 Print "Course not found"

## 3. Binary Search Tree (BST)

### Load and Parse File

Function LoadCourses(filePath)  
 Open file  
 If cannot open:  
 Print "Error"  
 Exit  
 For each line:  
 Parse tokens  
 If invalid format:  
 Print error  
 Continue  
 Validate prerequisites  
 Create Course object  
 Insert into BST

### Insert into BST

Function Insert(root, course)  
 If root is null:  
 Return new node  
 If course.courseNumber < root.course.courseNumber:  
 root.left = Insert(root.left, course)  
 Else:  
 root.right = Insert(root.right, course)  
 Return root

### Print All Courses (In-Order Traversal)

Function PrintAllCourses(node)  
 If node is not null:  
 PrintAllCourses(node.left)  
 Print course info  
 PrintAllCourses(node.right)

### Search Course

Function Search(node, input)  
 While node is not null:  
 If courseNumber == input:  
 Print course info  
 Return  
 Else if input < node.courseNumber:  
 node = node.left  
 Else:  
 node = node.right  
 Print "Course not found"

# Menu System (Common to All)

While userChoice != 9:  
 Print Menu  
 If userChoice == 1:  
 LoadCourses()  
 If userChoice == 2:  
 PrintAllCourses()  
 If userChoice == 3:  
 Prompt for course number  
 SearchCourse()  
 If userChoice == 9:  
 Exit

# Runtime Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operation | Vector | Hash Table | BST | Memory Usage | Best Use Case |
| Load & Parse File | O(n) | O(n) | O(n log n)\* | Moderate | Small data |
| Search Course | O(n) | O(1) | O(log n)\* | High | Fast lookup |
| Print All (Sorted) | O(n log n) | O(n) + sort | O(n) | High | Sorted view |

\*Assuming a balanced tree.

# Evaluation & Recommendation

Vector is simple and effective for small datasets and maintains insertion order. However, searching is linear and sorting adds overhead.  
Hash Table is the best for fast lookups, but lacks natural ordering. It consumes more memory and requires extra logic for sorting.  
BST offers both ordered storage and efficient searching, but performance can degrade if unbalanced.  
  
✅ Recommendation: Vector  
Based on the requirements, Vector is recommended due to:  
- Simpler implementation  
- Acceptable runtime (O(n log n) sort, O(n) search)  
- Lowest total cost among all three options  
- Successfully implemented and validated in earlier milestones

# Detailed Runtime Cost Tables

## Vector

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| for all courses | 1 | n | n |
| if the course is the same as courseNumber | 1 | n | n |
| print out the course information | 2 | 1 | 1 |
| for each prerequisite of the course | 1 | n | n |
| print the prerequisite course information | 2 | n | n |

Total Cost: 6n + 1

Runtime: O(n)

## Hash Table

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| for all courses | 2 | n | n |
| if the course is the same as courseNumber | 1 | n | n |
| print out the course information | 1 | 1 | 1 |
| for each prerequisite of the course | 2 | n | n |
| print the prerequisite course information | 4 | n | n |

Total Cost: 9n + 1

Runtime: O(n)

## Tree

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| for all courses | 1 | n | n |
| if the course is the same as courseNumber | 1 | n | n |
| print out the course information | 2 | 1 | 1 |
| for each prerequisite of the course | 1 | n | n |
| print the prerequisite course information | 4 | n | n |

Total Cost: 8n + 1

Runtime: O(n)