**CS 300 Project One: Pseudocode and Runtime Analysis**

**Updated Pseudocode**

**Vector Implementation**

**Menu Pseudocode:**

FUNCTION displayMenu():

PRINT "1. Load Data Structure"

PRINT "2. Print Course List"

PRINT "3. Print Course"

PRINT "9. Exit"

RETURN GET\_USER\_INPUT()

FUNCTION main():

courses = EMPTY\_VECTOR

WHILE TRUE:

choice = displayMenu()

IF choice == 1:

filename = GET\_USER\_INPUT("Enter file name: ")

courses = loadCoursesFromFile(filename)

ELSE IF choice == 2:

printAllCoursesSorted(courses)

ELSE IF choice == 3:

courseNumber = GET\_USER\_INPUT("Enter course number: ")

searchCourse(courses, courseNumber)

ELSE IF choice == 9:

EXIT

ELSE:

PRINT "Invalid option"

**Print All Courses Sorted:**

FUNCTION printAllCoursesSorted(Vector<Course> courses):

sortedCourses = SORT(courses, BY courseNumber)

FOR EACH course IN sortedCourses:

PRINT course.courseNumber + ", " + course.name

**Hash Table Implementation**

**Menu Pseudocode:**  
FUNCTION displayMenu():

PRINT "1. Load Data Structure"

PRINT "2. Print Course List"

PRINT "3. Print Course"

PRINT "9. Exit"

RETURN GET\_USER\_INPUT()

FUNCTION main():

coursesHashTable = EMPTY\_HASH\_TABLE

WHILE TRUE:

choice = displayMenu()

IF choice == 1:

filename = GET\_USER\_INPUT("Enter file name: ")

coursesHashTable = loadCoursesFromFile(filename)

IF coursesHashTable == NULL:

PRINT "Failed to load courses from file."

ELSE IF choice == 2:

IF IS\_EMPTY(coursesHashTable):

PRINT "No courses loaded. Please load data first."

ELSE:

printAllCoursesSorted(coursesHashTable)

ELSE IF choice == 3:

IF IS\_EMPTY(coursesHashTable):

PRINT "No courses loaded. Please load data first."

ELSE:

courseNumber = GET\_USER\_INPUT("Enter course number: ")

searchCourse(coursesHashTable, courseNumber)

ELSE IF choice == 9:

PRINT "Exiting program."

EXIT

ELSE:

PRINT "Invalid option. Please try again."

**Print All Courses Sorted:**

FUNCTION printAllCoursesSorted(HashTable<Course> courses):

allCourses = GET\_ALL\_VALUES(courses)

sortedCourses = SORT(allCourses, BY courseNumber)

FOR EACH course IN sortedCourses:

PRINT course.courseNumber + ", " + course.name

**Binary Search Tree Implementation**

**Menu Pseudocode:**  
FUNCTION displayMenu():

PRINT "1. Load Data Structure"

PRINT "2. Print Course List (Alphanumeric Order)"

PRINT "3. Print Course Information"

PRINT "9. Exit"

RETURN GET\_USER\_INPUT()

FUNCTION main():

courseTree = NULL // BST root pointer initialized to null

WHILE TRUE:

choice = displayMenu()

IF choice == 1:

filename = GET\_USER\_INPUT("Enter file name: ")

courseTree = loadCoursesIntoTree(filename) // Returns BST root

IF courseTree == NULL:

PRINT "Failed to load courses from file."

ELSE IF choice == 2:

IF courseTree == NULL:

PRINT "No courses loaded. Please load data first."

ELSE:

printAllCoursesInOrder(courseTree) // In-order traversal

ELSE IF choice == 3:

IF courseTree == NULL:

PRINT "No courses loaded. Please load data first."

ELSE:

courseNumber = GET\_USER\_INPUT("Enter course number: ")

printCourseInfo(courseTree, courseNumber) // BST search

ELSE IF choice == 9:

PRINT "Exiting program."

EXIT

ELSE:

PRINT "Invalid option. Please try again."

**Print All Courses Sorted:**

FUNCTION printAllCoursesSorted(BinarySearchTree courseTree):

IN\_ORDER\_TRAVERSAL(courseTree.root):

IF node != NULL:

IN\_ORDER\_TRAVERSAL(node.left)

PRINT node.key + ", " + node.value.name

IN\_ORDER\_TRAVERSAL(node.right)

**Runtime Analysis**

File Loading and Course Creation

**Vector Analysis:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| OPEN\_FILE | 1 | 1 | 1 |
| WHILE NOT EOF | 1 | n | n |
| SPLIT line | 1 | n | n |
| Validate format | 1 | n | n |
| Create course object | 1 | n | n |
| Store in vector | 1 | n | n |
| **Total Cost** |  |  | 4n + 1 |
| **Runtime** |  |  | O(n) |

**Hash Table Analysis:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| OPEN\_FILE | 1 | 1 | 1 |
| WHILE NOT EOF | 1 | n | n |
| SPLIT line | 1 | n | n |
| Validate format | 1 | n | n |
| Create course object | 1 | n | n |
| Hash insert | 1 | n | n |
| **Total Cost** |  |  | 4n + 1 |
| **Runtime** |  |  | O(n) |

**BST Analysis:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| OPEN\_FILE | 1 | 1 | 1 |
| First pass WHILE | 1 | n | n |
| SPLIT line | 1 | n | n |
| Validate format | 1 | n | n |
| Second pass FOR | 1 | n | n |
| BST insert | log n | n | n log n |
| **Total Cost** |  |  | n log n + 4n + 1 |
| **Runtime** |  |  | O(n log n) |

Course Search Operations

**Vector Search:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| FOR all courses | 1 | n | n |
| IF match | 1 | n | n |
| Print info | 1 | 1 | 1 |
| FOR prerequisites | 1 | p | p |
| **Total Cost** |  |  | 2n + p + 1 |
| **Runtime** |  |  | O(n) |

**Hash Table Search:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Hash lookup | 1 | 1 | 1 |
| Print info | 1 | 1 | 1 |
| FOR prerequisites | 1 | p | p |
| **Total Cost** |  |  | p + 2 |
| **Runtime** |  |  | O(1) average case |

**BST Search:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| BST search | log n | 1 | log n |
| Print info | 1 | 1 | 1 |
| FOR prerequisites | 1 | p | p |
| **Total Cost** |  |  | log n + p + 1 |
| **Runtime** |  |  | O(log n) |

Print All Courses Sorted

**Vector Sort and Print:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Sort | n log n | 1 | n log n |
| FOR all courses | 1 | n | n |
| **Total Cost** |  |  | n log n + n |
| **Runtime** |  |  | O(n log n) |

**Hash Table Sort and Print:**  
(Same as vector since we must extract all values first)

**BST In-order Traversal:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Traversal | 1 | n | n |
| **Total Cost** |  |  | n |
| **Runtime** |  |  | O(n) |

Data Structure Analysis

Vector

**Advantages:**

* Simple implementation
* Fast insertion (O(1) amortized)
* Good cache locality

**Disadvantages:**

* Slow search (O(n))
* Slow sorting (O(n log n) each time)
* Inefficient for large datasets

Hash Table

**Advantages:**

* Very fast search (O(1) average case)
* Fast insertion (O(1) average case)
* Direct access to courses

**Disadvantages:**

* No inherent ordering (must sort for display)
* Potential collisions
* Memory overhead

Binary Search Tree

**Advantages:**

* Naturally maintains sorted order
* Fast search (O(log n))
* Efficient printing of sorted list (O(n))

**Disadvantages:**

* Slower insertion than hash table (O(log n))
* Can become unbalanced
* More complex implementation

Recommendation

Based on the analysis, I recommend using the Binary Search Tree data structure for this application. The key reasons are:

1. The BST provides efficient O(log n) search times while maintaining the courses in sorted order naturally.
2. Printing the sorted course list is extremely efficient (O(n)) with in-order traversal, which is a primary requirement.
3. While insertion is slightly slower than a hash table (O(log n) vs O(1)), we typically load the data once and perform many searches and print operations.
4. The BST provides a good balance between search efficiency and sorted order maintenance.

The hash table would be a close second choice if search speed was the absolute priority, but the need to frequently print the sorted list makes the BST more suitable overall. The vector implementation would be too inefficient for larger datasets due to its linear search time.