**Pseudocode and Runtime Analysis**

**Kyler Legrone**

This document includes the finalized pseudocode and runtime analysis for the Vector, Hash Table, and Binary Search Tree implementations as part of Project One.

**Pseudocode Functions**

**Vector Implementation**

void searchCourse(Vector<Course> courses, String courseNumber) {  
 for all courses  
 if the course is the same as courseNumber  
 print out the course information  
 for each prerequisite of the course  
 print the prerequisite course information  
}

**Hash Table Implementation**

void searchCourse(HashTable<Course> courses, String courseNumber) {  
 if courseNumber exists in hash table  
 print out course information  
 for each prerequisite of the course  
 if prerequisite exists in hash table  
 print the prerequisite course information  
 else  
 print prerequisite not found  
 else  
 print course not found  
}

**Binary Search Tree Implementation**

void searchCourse(Tree<Course> courses, String courseNumber) {  
 search for course in BST  
 if course is found  
 print out course information  
 for each prerequisite of the course  
 search prerequisite in BST  
 if found, print prerequisite course information  
 else print prerequisite not found  
 else  
 print course not found  
}

**Runtime Analysis**

**Vector Runtime Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| for all courses | 1 | n | n |
| if the course is the same as courseNumber | 1 | n | n |
| for each prerequisite of the course | 1 | 1 | 1 |
| print the prerequisite course information | 1 | n | n |
| Total |  |  | 4n + 1 |
| Runtime |  |  | O(n) |

**Hash Table Runtime Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| check if course exists in hash table | 1 | 1 | 1 |
| print course info | 1 | 1 | 1 |
| for each prerequisite of the course | 1 | m | m |
| check if prerequisite exists in hash table | 1 | m | m |
| print prerequisite course information | 1 | m | m |
| Total |  |  | 3m + 2 |
| Runtime |  |  | O(m) |

**Binary Search Tree Runtime Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| search for course in BST | log n | 1 | log n |
| print course info | 1 | 1 | 1 |
| for each prerequisite of the course | 1 | m | m |
| search prerequisite in BST | log n | m | m log n |
| print prerequisite info | 1 | m | m |
| Total |  |  | m log n + m + log n + 1 |
| Runtime |  |  | O(m log n) |

**Evaluation and Recommendation**

Vector is simple and easy to implement but inefficient for searching and sorting (O(n) for search, O(n log n) with sorting). It's best for small datasets but not ideal for quick lookups.

Hash Table offers the fastest search time (O(1)) and handles direct access well. However, it doesn’t maintain order, so sorting must be done separately. It's memory-efficient but not ideal when sorted output is constantly needed.

The Binary Search Tree provides a great balance, with O(log n) searches and automatic alphanumeric sorting through in-order traversal. Though slightly more complex to implement, it supports all advisor requirements efficiently.

I recommend using the Binary Search Tree (BST) for the final implementation due to its sorted structure and efficient search performance.