Ann San Miguel

CS300 DSA: Analysis and Design

Professor Michael Susalla

August 6, 2025

Project One

**Vector Pseudocode**

Load Data: loadCoursesFromVectorFile(filename)  
Open file, read line by line.  
Parse line, check for errors, create a Course object.  
Add the Course to a Vector.  
**Menu:** displayVectorMenu()  
Present menu options: 1) Load, 2) Print All, 3) Find, 9) Exit.  
Read user input and call the appropriate function.  
**Print Sorted Courses:** printSortedVectorCourses(courses)  
Sort the courses vector (e.g., using quicksort).  
Iterate and print the alphanumeric course number and title.  
**Find Course:** findAndPrintVectorCourse(courses, courseNumber)  
Linearly search the courses vector for a matching courseNumber.  
If found, print the title and prerequisites; otherwise, print "not found."

**Hash Table Pseudocode**

**Load Data:** loadCoursesFromHashTableFile(filename)  
Open file, read each line, and create a Course object.  
Insert the Course into a HashTable using the course number as the key.  
**Menu:** displayHashTableMenu()  
Present menu options (same as vector menu).  
Read user input and call the appropriate hash table function.  
**Print Sorted Courses:** printSortedHashTableCourses(hashTable)  
Create a temporary vector by iterating through all hash table buckets.  
Sort the temporary vector.  
Print each course.  
**Find Course:** findAndPrintHashTableCourse(hashTable, courseNumber)  
Calculate the hash key from the courseNumber.  
Access the corresponding bucket to find the course.  
If found, print the details; otherwise, print "not found."

**Binary Search Tree (BST) Pseudocode**

**Load Data:** loadCoursesFromBSTFile(filename)  
Open file, read each line, and create a Course object.  
Insert the Course into a BST based on its alphanumeric course number.  
**Menu:** displayBSTMenu()  
Present menu options (same as others).  
Read user input and call the appropriate BST function.  
**Print Sorted Courses:** printSortedBSTCourses(bst)  
Perform an **in-order traversal** of the BST, which naturally prints the courses in sorted, alphanumeric order.  
**Find Course:** findAndPrintBSTCourse(bst, courseNumber)  
Traverse the tree, moving left or right depending on whether the courseNumber is less than or greater than the current node.  
If a match is found, print the details; otherwise, print "not found."

| **Task** | **Vector** | **Hash Table** | **Binary Search Tree** |
| --- | --- | --- | --- |
| **For loading the data** | O(n) | O(n) | O(n log n) |
| **Printing and sorting list** | O(n log n) | O(n log n) | O(n) |
| **Finding a Course** | O(n) | For O(1)= O(n) | For O(logn)=O(n) |
| **Memory Usage** | O(n) | O(n) | O(n) |

Advantages of vector, hash table, and BST would be that simple to implement, constant time lookup and performance, and efficient search natural sorting via in order. Disadvantages for all three vector, hash table, and BST would be that linear search time, not naturally ordered sorting is still needed in this case, and can degrade to O(n²) if not balanced.

My Final recommendation is that the hash table would be the best choice for this application due to the main strength is near-instantaneous search time (O(1) on average) for a specific course, which is the most frequent task of an academic advisor. While printing the sorted task list takes a long time than the BST, well this one is a less common operation. Hash table is more likely as an efficient data structure for this program.