**Menu**

Int input = 0;

Int courseNumber;

While (input does not equal 9){

Print << 1. load file

Print << 2. Enter Specific course

Print << 3. Print sorted course list

Print << 4. Print Selected course

Print << 9. Exit Program

ReadInput << input

Switch (choice){

Case 1:

LoadData(File)

Print << File loaded

Case 2:

Print << Enter course number

ReadInput << courseNumber

Case 3:

sortCourses()

printSortedCourses()

//or for the tree data structure

Bst ->inOrder()

Case 4:

printCourseInformation(courseNumber)

Case 9:

Exit()

}

}

**Vector**

sortCourses

For i from 0 to courseVector.length -1

For j from i + 1 to courseVector.length

If courseVector[j].courseNumber < courseVector [i].courseNumber

Swap courseVector[i] with courseVector[j]

printSortedCourses

For each course in courseVector

Print << course.courseNumber + course.name + course.coursePrerequisite

**Hash Table**

sortCourses

courseList = courseHashTable Keys

For i from 0 to courseList.length -1

For j from i + 1 to courseList.length

If course.List[j].courseNumber < courseList [i].courseNumber

Swap courseList[i] with courseList[j]

printSortedCourses

For each course in courseList

Print << course.courseNumber + course.name + course.coursePrerequisite

**Tree**

inOrder(node)

inOrder(node.left)

Print << node.course.courseNumber + node.course.name + node.course.coursePrerequisite

inOrder(node.right)

**Run Time and Memory Evaluation**

* Vector: The vector will run at O(n^2) for its average and worst case.
* Hash Table: will run at O(n) on average and O(n^2) for its worst case.
* BST: The tree structure will run at O(n log n) on average and O(n^2) for its worst case.

**Data Structure Analysis**

* Vector: This data structure has the advantages of being simple to implement, having fast insertions at O(1), and handling memory efficiently. The downside is that it is not as efficient when searching through the data as the size of the data scales.
* Hash Table: This data structure also has the advantage of fast searches, but unlike vectors, it can also search efficiently. The downsides of this one are that it is more complex and can also run into collision issues, which can reduce the worst-case running time to O(n).
* BST: In a balanced tree, the run time is O(log n), giving it quick search performance. If the tree becomes unbalanced, however, that can reduce it to O(n). There is also no need to sort the data, as you can print it with an in-order traversal. It is still more complex to implement than a vector, though, and it can have a higher memory overhead.

**Recommendation**

My recommendation for the data structure will be a BST. While it is a bit more complex to implement, it has a few advantages that I think justify it. Even as more courses are added, the run-time will remain efficient, assuming it remains balanced. Traversing the nodes is simple once the data structure is created, which is helpful for printing specific courses and prerequisites. It also automatically maintains a sorted order, reducing the need to manually sort it.