**Pseudocode for a menu:**

*displayMenu(){*

**display "Menu:"**

**display "1 - Load Course Data"**

**display "2 - Print All Courses (Sorted)"**

**display "3 - Print Course Information"**

**display "9 - Exit"**

}

*menu(){*

**create datastructure object (dst)**

**bool dataLoaded = false**

**while loop**

**DisplayMenu()**

**Get choice value from user**

**Switch(choice):**

**CASE 1:**

**loadCourses(dst)**

**set dataLoaded = TRUE**

**display confirmation of data loaded**

**BREAK**

**CASE 2:**

**if dataLoaded is false:**

**display error**

**else:**

**printAllCourses(dst)**

**BREAK**

**CASE 3:**

**if dataLoaded is false:**

**display error**

**else:**

**prompt user to enter courseNumber**

**printCourseInformation(dst, courseNumber)**

**BREAK**

**CASE 9:**

**Display program exit**

**End while loop**

**DEFAULT:**

**Display error an ask for new inout**

}

*}*

*loadCourses(dst){*

**prompt for file path**

**get filepath**

**loadCoursesFromFile(filepath, dst)**

*}*

**Pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order:**

*Vector\_DisplaySortedCourses(courseVector){*

**sort courseVector by course.courseNumber**

**for each course in vector**

**display course.courseNumber and course.courseTitle**

*}*

*Hash\_displaySortedCourses (hashTable{*

**Create empty course list**

**for each bucket in hashTable:**

**for each course in bucket:**

**add course to courseList**

**sort courseList by course.courseNumber**

**for each course in courseList:**

**display course.courseNumber and course.courseTitle**

*}*

*Bst\_displaySortedCourses (node){*

**IF node IS NOT NULL:**

**Ttraverse left subtree with recursive call displaySortedCourses (node.left)**

**display node.course.courseNumber and node.course.courseTitle**

**Ttraverse left subtree with recursive call displaySortedCourses(node.right**

*}*

**Runtime evaluation:**

**Vector**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open the file** | 1 | 1 | 1 |
| **Read each line in the file** | 1 | n | n |
| **Parse the line into fields** | 1 | n | n |
| **Create a Course object** | 1 | n | n |
| **Add the Course object to the vector** | 1 | n | n |
| **Close the file** | 1 | 1 | 1 |
| **Total Cost** | | | 3n + 2 |
| **Runtime** | | | O(n) |

Advantages: Vectors are advantageous for access which is direct and iteration which is beneficial in our program for printing.

Disadvantages: Vectors are disadvantageous when inserting data because it requires adjusting the positions of data in order to insert meaning there are extra steps needed. Vectors are also disadvantages when searching because it has to be done sequentially.

**Hash Table**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open the file** | 1 | 1 | 1 |
| **Read each line in the file** | 1 | n | n |
| **Parse the line into fields** | 1 | n | n |
| **Create a Course object** | 1 | n | n |
| **Compute hash and insert into hash table** | O(1) | n | n |
| **Close the file** | 1 | 1 | 1 |
| **Total Cost** | | | 3n + 2 |
| **Runtime** | | | O(n) |

Advantages: Hash tables are advantageous because they have constant time complexity for inserting, deleting, and searching because of the hashing directly maps keys to indices.

Disadvantages: Hash tables are disadvantageous becasue collisions can occur and handling them lower the algorithms efficiency and memory.

**Binary Search Tree**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open the file** | 1 | 1 | n |
| **Read each line in the file** | 1 | n | n |
| **Parse the line into fields** | 1 | n | n |
| **Create a Course object** | 1 | n | n |
| **Insert Course into BST** | O(logn) | n | n\*O(logn) |
| **Close the file** | 1 | 1 | 1 |
| **Total Cost** | | | n⋅O(logn)+3n+2 |
| **Runtime** | | | O(nlogn) |

Advantages: BST are advantageous when retrieving course details and maintain their sorted order due to balancing when implementing. Insertions are efficient with a cost of n\*O(logn).

Disadvantages: BST are disadvantageous because they are more complex to implement and maintain. Maintainace includes balancing the BST in order to keep its integrity as a BST.

**Recommendation:**

I would recommend the use of a binary search tree for my code. Although it is more of a challenge to implement, it will maintain sorted order when inserting without the need to resort like with a vector or hashtable.