Project One

*// Vector Pseudocode*

Vector<Course> readCourses(string csvPath) {

For each row in file

If less than 2 parameters on a line

Skip line (not valid)

Else

Add course number, description, and n prerequisites to vector

For each course

If prerequisite does not match a course number

Delete course from vector

Return course vector

}

Void printCourseInfo(Vector<Course> courses, String courseNumber) {

For each course in vector

If course is same as courseNumber

Print course information

For each prerequisite of the course

Print preqequisite course information

}

// Hash Table Pseudocode

Void loadCourses(string csvPath, HashTable\* courseTable) {

Calculate hash key based on number of rows in file

For each row in file

IF less than 2 parameters on a line

Skip line (not valid)

Else

If key is empty

Add course number, description, and ‘n’ prerequisites to table Else

Chain course node at key location

For each course in table

If prerequisite does not match a course number

Delete course from courseTable

}

Void printCourseInfo(HashTable\* courseTable, String courseNumber) {

For each course in table

If course is same as courseNumber

Print course information

For each prerequisite of the course

Print preqequisite course information

}

*// Binary Search Tree Pseudocode*

Void loadCourses(string csvPath, BinarySearchTree\* bst) {

For each row in file

If less than 2 parameters on a line

Skip line (not valid)

Else

If root is empty

Insert Node with course info as root

Else If courseNumber < rootNumber

If node is empty

Insert Node with course info

Else

Recursively repeat with node->left

Else

If node is empty

Insert Node with course info

Else

Recursively repeat with node->right

}

Void printCourseInfo(BinarySearchTree\* bst, String courseNumber) {

If no root

Print “Not Found”

While current node is not empty

If courseNumber matches

Print course info and any prerequisite course info

If courseNumber < root courseNumber

Assign current node with left child

Else

Assign current node with right child

}

**Menu Pseudocode**

Print menu options and wait for input

If input is valid (switch statement)

*// Option 1 – Load Courses*

Create the data structure

Call the loadCourses method for the appropriate data function

Print how many courses were loaded

*// Option 2 – Print courses*

Call the printCourseInfo method

*// Option 3 – Print Course*

Call search method for the correct data structure

If a course is found

Print course info plus prerequisites

Else

Print “Course not found”

*// Option 4 – Exit*

Exit the program

Else

Show menu again

**Alphanumeric Sort and Print**

*// Vector*

Use a quicksort function to sort vector alphanumerically

For each course in vector

Print course information

*// Hash Table*

A hash table is inherently “sorted” when insertion occurs based on the course number

void HashTable::PrintAll() {

For each key value smaller than table size

If there is one item in a bucket

Print course information

While there are multiple items in a bucket

Print course information

Set current node to the next node in the chain

}

*// Binary Search Tree*

A BST is sorted when insertion occurs as long the course number is the thing being compared

Void BinarySearchTree::inOrder(Node\* node) {

If next node is empty

End recursion

Recursively call inOrder with left child

Print course information

Recursively call inOrder with right child

}

*// Run-time analysis*

| **Vector Load Courses/Create Object** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| For all rows in file | 1 | n | n |
| If there are less than 2 parameters on a line | 1 | n | n |
| Skip line | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| Add course number, description, and n prerequisites to vector | 1 | 1 | 1 |
| For each course in vector | 1 | n | n |
| If prerequisite does not match a course number | 1 | n | n |
| Delete course from vector | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 4 |
| **Runtime** | | | O(n) |

| **Hash Table Load Courses/Create Object** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Calculate hash key | 1 | 1 | 1 |
| For all rows in file | 1 | n | n |
| If there are less than 2 parameters on a line | 1 | n | n |
| Skip line | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| If key is empty | 1 | n | n |
| Add course number, description, and n prerequisites to vector | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| Chain course node at key location | 1 | n | n |
| For each course in vector | 1 | n | n |
| If prerequisite does not match a course number | 1 | n | n |
| Delete course from table | 1 | 1 | 1 |
| **Total Cost** | | | 6n + 6 |
| **Runtime** | | | O(n) |

| **Binary Search Tree Load Courses/Create Object** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| For all rows in file | 1 | n | n |
| If there are less than 2 parameters on a line | 1 | n | n |
| Skip line | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| If root is empty | 1 | n | n |
| Insert node with course info | 1 | 1 | 1 |
| Else if courseNumber < rootNumber | 1 | 1 | 1 |
| If node is empty | 1 | n | n |
| Insert node with course info | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| Recursively repeat with node->left | 5 | n | n |
| Else | 1 | 1 | 1 |
| If node is empty | 1 | n | n |
| Insert node with course info | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| Recursively repeat with node->right | 5 | n | n |
| For each course in vector | 1 | n | n |
| If prerequisite does not match a course number | 1 | n | n |
| Delete course from table | 1 | 1 | 1 |
| **Total Cost** | | | 9n + 9 |
| **Runtime** | | | O(n) |

**Analysis**

The vector data structure’s primary advantages are that it can be dynamically resized if more courses are added to the catalog, they generally take less memory than other data structures, and they are simpler to code than the other two options. As for disadvantages, vectors are not initially sorted like a binary search tree is, therefore, in order to sort the course list, a separate sorting function needs to be used which takes more time and memory. One other thing about vectors is that the average and worst case runtime complexities are usually the same. The hash table data structure has the advantage of having an average time complexity of O(1) when it comes to searching and inserting elements which is the best of the three structures. A disadvantage is that space can be poorly used if the number of buckets is poorly chosen. For instance, if there were 10,000 buckets for 100 courses, there would be a lot of wasted memory, whereas a vector or BST cannot have that issue. As for a BST, one advantage is that the data is inherently sorted when it is first sorted into the BST (which is ultimately what makes it a BST). This eliminates the need for a separate sorting function, however, if the user wanted to order the BST based on a different criterion, then an entirely new BST has to be created rather than simply reordering the existing BST which would be unnecessary with a vector. As for runtime complexity, the BST has one of the best average runtimes of O(log(n)), however, that relies on the fact that the BST is balanced otherwise the runtime is more likely to be O(n).

**Recommendation**

Based on the analysis of the data structures, I would recommend the hash table as the ideal data structure for ABCU. This is primarily because it has an excellent runtime complexity, along with the fact that it is easily modified to suit the situation that the client may need. Although it has a similar, if not better, runtime complexity, a BST would not be ideal for this case due to the fact that it cannot be simply reordered like a vector or hash table can be. Therefore, it is less versatile as a data structure in this instance. A vector would be an okay choice as well, however, it lacks some functionality compared to a hash table and typically has a slower runtime based on the size of the data. If the course catalog were to jump in size, a hash table could easily accommodate the change by modifying the number of buckets, however, a vector would become n-times larger regardless of what you do.