**Module Three**

// Vector pseudocode

Use fstream to open file

Call to open file

If return value > -1

Display “file not found”

Else file is found

If not end of file (EOF)

Read each line

Parse each line

If fewer than two values in a line

Display “ERROR”

Else read parameters

Close file

Search for SpecificCourse

Initialize variables to open file

Read file

Read each line

Parse each line

Check for courseTitle

Check for courseNumber

If file free of errors

Check for prerequisites for course

Create CourseObj

Initialize variables for courses

Call file to read

Store CourseObj in vector data structure

Search for SpecificCourse

Initialize varables to open file

Call file to read

Print course information

Store data in vector data structure

**Module Four**

//Hash table

Use fstream to open file

Call to open file

If return value > -1

Display “file not found”

Else file is found

If not end of file (EOF)

Read each line

If fewer than two values in a line

Display “ERROR”

Else read parameters

Close file

Create HashTable Class

Create Insert method to insert items into HashTable

Loop through the file

If not EOF

For each line in file

For 1st and 2nd values

Create temp\_item to hold values

If 3rd value exists

Add to current value

Call insert method for each value

Request Input

Assign Input to key

If key found

Print course information

For each prerequisite of course

Print prerequisite course information

**Module Five**

//Tree

**Menu**

Pseudocode for menu

Create integer for switch statement

Integer = uInput

uInput = 0

Create Bid variable

While input ≠ 4

Print: 1. Load Data Structure

Print: 2. Course List

Print: 3. Course

Print: 4. Exit

Switch(uInput)

Case 1:

loadBids(bid)

Case 2:

Print: Course List alphanumerically

Case 3:

Print: Course title

Print: Prerequisites

Case 4:

Print: “Thank you. Goodbye.”

Case 5:

Print: “No input found for user.”

**Alphanumeric order**

Create sorted print method

printSorted(courses)

Create partition method

int partition

Set lowIndex to first element

Set highIndex to last element

Set midPoint to lowIndex

Create quicksort method

quicksort

Set mid to 0

lowIndex to begin

highIndex to end

If begin >= end

Return

Recursive call to quicksort

Create display course method

void displayCourse

Loop through vector to display courses

Display courses

Create inOrder method

Void BinarySearchTree inOrder

If node != null

Check left leaf

node->left

Check right leaf

node->right

**Runtime analysis**

**Pros and cons of each data structure**

Vector data structures are used in C++ to store elements of similar data types. One advantage of using vector data structures is it is easy to add and remove the last element from the index. It is also easy to access the target element using the index. With few advantages, come disadvantages as well. Vector data structures are inefficient when inserting or deleting elements in the middle since the rest of the elements would require adjustments after the insertion/deletion. Vector data structures are also not good when working with large amounts of data because memory consumption is more resulting in slower loading times.

Hash table data structures use a hash function to compute indexes for a key, allowing one to store the correct value in the correct location. A pro of using hash tables is better synchronization because they lack the time complexity found in binary search trees. Another advantage is how fast and efficient hash tables are for looking up, creating, and deleting data. Hash tables are also usually best for large amounts of data. A disadvantage of using hash tables is when there are too many collisions, hash tables become less efficient. Hash tables reject “null” as a key’s value because “null” is not a unique value. In fact, it isn’t a value at all.

Binary search tree data structures are binary trees that contain a left subtree with keys that are lesser than the key of the node and a right subtree with keys that are greater than the key of the node. An advantage of using binary search trees is the code use is simple compared to other data structures. Binary search trees also are more efficient for insertion and deletion. A disadvantage of using binary search trees is accessing the element is usually slower than other methods. When binary search trees are imbalanced, complexity can increase.

**Recommendation**

Because vector data structures require more memory and are not good with larger amounts of data, I would completely cancel that one out. This leaves hash tables and binary search trees. Hash tables require the proper hash function in order to work. There is no special function for binary search trees. One must also know the input data size before creating a hash table. This isn’t always possible and is not a requirement for binary search trees. Runtime is typically faster for binary search trees than hash tables, too. With this information, I would recommend a binary search tree.