* Apply non-coding development methodologies for outlining an algorithmic design
* Evaluate complex data structures that solve a given problem using advanced algorithmic designs

**Vector**

**// Load Courses Function**

Declare string variable line

Declare stream object myFile and open CSV file

If myFile does not open

Display error message

Return

While myFile is open and good

Get line from myFile with new line delimiter and save as line

If comma count in line is less than 1 then

Break

Else

Call function to add new Course vector object with line as parameter

If end of file then

Close CSV file

Call prerequisite check function

**// Add Course Function**

Declare a Course vector with course objects

Declare function that takes line from CSV file as parameter

Declare string stream ss and initialize with line

Create new course object called course

Declare string str

Let integer i = 0

While string stream is good

Find string in string stream with comma delimiter and save as str

If i = 0 then

Set str as course Number

Increase i

Else if i = 1 then

Set str as course Name

Increase i

Else

Set str as course Prerequisite

Add course to Course vector

**// Prerequisite Check Function**

Declare prerequisite check function that takes Course vector as parameter

Let integer i = 0

While i is less than Course length

If course object at position i has prerequisites

Check if prerequisite matches a course number in Course vector

If prerequisite does not match any course number in Course vector

Remove course from vector

Increase i

**// Print All Function**

Call quicksort function passing Course vector, beginning, and end as parameter

Partition courses into low and high

Implement quicksort logic over course ID until done

Return sorted Course vector

For course in Course vector

Display course

**// Search Course Function**

Get course name

Let integer i = 0

While i is less than course length

If course object name at position i matches course name from input then

Print course information and prerequisites

Return

Else

Increase i

If no course name matches then

Print “No Matches”

Return

**// Display Menu**

Define a vector to hold all courses

Declare integer variable choice and set to 0

Declare integer i and set = 0

While choice !=9

Display program menu

Get choice

If choice = 1

i = i +1

Call load courses function passing CSV file and course vector as parameter

If choice = 2

If i = 0

Display error must load courses first

Else

Call print all course function passing course vector as parameter

If choice = 3

If i = 0

Display error must load courses first

Else

Get course

Call search function passing course as parameter

If choice = 9

Display Goodbye

Exit

**Runtime Analysis**

 

**Hash Table**

**// Load Courses Function**

Declare string variable line

Declare stream object myFile and open CSV file

If myFile does not open

Display error message

Return

While myFile is open and good

Get line from myFile with new line delimiter and save as line

If comma count in line is less than 1 then

Break

Else

Call function to add new course hash table object with line as parameter

If end of file then

Close CSV file

Call prerequisite check function

**// Add Course Function**

Declare a Course hash table class with course objects

Define structure to hold courses with course number, course name, and prerequisites

Declare string stream ss and initialize with line

Create new course object called course

Declare string str

Let integer i = 0

While string stream is good

Find string in string stream with comma delimiter and save as str

If i = 0 then

Set str as course Number

Increase i

Else if i = 1 then

Set str as course Name

Increase i

Else

Set str as course Prerequisite

Call insert method passing new course object as parameter

Call hash method to calculate key based on course Number

Assign new node to node at key

If the node is empty

Allocate new node

Set new node next to empty

Set new node data to course object

Append node to hash table

**// Prerequisite Check Function**

Declare prerequisite check function that takes Course hash table as parameter

For each entry in hash table

If course object has prerequisites

Check if prerequisite matches a course number in Course hash table

If prerequisite does not match any course number in Course hash table

Remove course from hash table

**// Print All Function**

Declare new Sorted vector withsortedCourse objects

For each node in Course hash table

Copy node to new sortedCourse object

Push back new sortedCourse object to Sorted vector

Call quicksort function passing Sorted vector, beginning, and end as parameter

Partition courses into low and high

Implement quicksort logic over course ID until done

Return sorted Sorted vector

For sortedCourse in Sorted vector

Display sortedCourse

**// Search Course Function**

Get course number

Call hash method to determine key for course number

Assign new node to node at key

If node is not empty

Print course information and prerequisites

Return

Else

Print “No Matches”

Return

**// Display Menu**

Define a hash table to hold all courses

Declare integer variable choice and set to 0

Declare integer i and set = 0

While choice !=9

Display program menu

Get choice

If choice = 1

i = i +1

Call load courses function passing CSV file and course hash table as parameter

If choice = 2

If i = 0

Display error must load courses first

Else

Call print all course function passing course hash table as parameter

If choice = 3

If i = 0

Display error must load courses first

Else

Get course

Call search function passing course as parameter

If choice = 9

Display Goodbye

Exit

**Runtime Analysis**

 

**Binary Search Tree**

**// Load Courses Function**

Declare string variable line

Declare stream object myFile and open CSV file

If myFile does not open

Display error message

Return

While myFile is open and good

Get line from myFile with new line delimiter and save as line

If comma count in line is less than 1 then

Break

Else

Pass line to create new course Binary Search Tree object

If end of file then

Close CSV file

Call prerequisite check function

**// Add Course Function**

Declare a Course binary search tree with course objects

Set BST root equal to nullptr

Define course objects with course number, course name, and prerequisites

Declare BST node structure with course object

Set node structure left child and right child equal to nullptr

Declare string stream ss and initialize with line

Create new course object called course

Declare string str

Let integer i = 0

While string stream is good

Find string in string stream with comma delimiter and save as str

If i = 0 then

Set str as course Number

Increase i

Else if i = 1 then

Set str as course Name

Increase i

Else

Set str as course Prerequisite

Call insert function passing new course object as parameter

If root node is equal to nullptr

Set root node to new bid

Else

Call add node passing root node and new course object as parameters

If course number of new course object is smaller than current node course number then

If left child is empty then

Set left child to new course object

Else set current node to left child

Recurse - Call add node passing left child and new course object as parameters

If course number of new course object is greater than current node course number then

If right child is empty then

Set right child to new course object

Else set current node to right child

Recurse - Call add node passing right child and new course object as parameters

**// Prerequisite Check Function**

Declare prerequisite check function that takes Course binary search tree as parameter

For each node in binary search tree

If course object at node has prerequisites

Check if prerequisite matches a course number in Course binary search tree

If prerequisite does not match any course number in Course binary search tree

Call remove function passing node as parameter

**// InOrder Function**

Call inOrder function passing root as parameter

**// inOrder Function**

If node is != nullptr

Call inOrder passing left child of node as parameter

Display course

Call inOrder passing right child of node as parameter

**// Search Course Function**

Get course number

Call search function passing course number as parameter

Set current node to root node

While current node is not equal to nullptr

If course number matches current node course number

Display course information and prerequisites

Return

Else If course number is less than current node course number

Set current node to current node left child

Else If course number is greater than current node course number

Set current node to current node right child

If course number not found

Display course not found

Return

**// Display Menu**

Define a binary search tree to hold all bids and set = nullptr

Declare integer variable choice and set to 0

Declare integer i and set = 0

While choice !=9

Display program menu

Get choice

If choice = 1

i = i +1

Call load courses function passing CSV file and course BST as parameter

If choice = 2

If i = 0

Display error must load courses first

Else

Call in order function passing course BST as parameter

If choice = 3

If i = 0

Display error must load courses first

Else

Get course

Call search function passing course as parameter

If choice = 9

Display Goodbye

Exit

**Runtime Analysis**

 

**Advantages and Disadvantages**

While although each data structure can be used to facilitate the needs of the academic advisors in the Computer Science department at ABCU, there are advantages and disadvantages that distinguish them. A vector is the easiest data structure to implement and requires fewer lines of code, highlighting its advantages. However, the time complexity associated with using a vector does not make it an ideal data structure for this project. The search, inserting, and removing operation for a vector would execute linearly and proportionally to the size of the input. The more courses, the longer the runtime. Another disadvantage is evident in the print all function. In order to sort the vector to print alphanumerically, a quicksort operation must be implemented and called prior to displaying courses. Quicksort has an average runtime of O(n log(n)) and worst of O(n^2).

Hash tables have an advantage with time complexity for searching, inserting, and removing (deleting) an item. For these operations, the average runtime while using a hash table is constant at O(1) meaning it executes within the same time regardless of the size of the input. The drawback of using a hash table is also apparent in the print all function. In order to sort the hash table, data from each entry must be copied into a vector or list and then sorted. This inconvenience has a time complexity of O(n) since all entries must be accessed.

Binary search trees require the greatest number of lines of code but performs the best out of the three data structures. The average time complexity to search, insert, or remove a course for this project would be O(log(n)). It recursively calls functions to traverse either the right or left subtree. Nodes to the left of the root are less in value and nodes to the right are greater. The impact of this organization can be seen in the runtime. Printing all courses in alphanumeric order will still have a time complexity of O(n) simply because each course must be accessed, but the order of the course objects does not need to be changed nor does another data structure need to be used. Making recursive calls to the inOrder function applies simple logic to traverse the left subtree first, followed by the root, and then the right subtree.

**Recommendation**

Based on the Big O complexity and analysis of the three data structures, I recommend using a binary search tree to fulfill the needs of the ABCU Computer Science department. The average O(n) time complexity for many vector operations and the difficulty with sorting a hash table are the main reasons a binary search tree is better suited for this project. All three data structures are similar in the way the way they open a CSV file, read data parsing each line, and check for format errors. Though there are some differences in the way a course objects are created, the runtime analysis for all three equated to O(n). When we further explore the runtime for other operations and the steps necessary to fulfill ABCU’s requirements, we can see that a binary search tree outperforms the others. The average time complexity for search, insertion, and deletion is O(log(n)) which is faster than a vector and, unlike a hash table, sorting alphanumerically does not require data to be copied into another structure.