**Vector Pseudocode:**

START

Create method for loading courses

Open files (with parameters)

WHILE file is not at end of file

Read and parse each line in CSV to check for course title & ID

Check values, if no errors, read parameters

IF correct

Check for course prerequisites

Create function for Course objects (with parameters)

Declare variables for courses, files; initialize variables

Open and read CSV file

Store course data

WHILE open

Store course data in vector

Create function for course search (with parameters)

Declare variables to open CSV file; initialize variables

Open and read CSV file

Output course information

WHILE open

Output course information

Store course information in vector

END

**Hash Table Pseudocode:**

START

Create node struct course

Open and read file

Parse each line in CSV to check for title

IF title is present

Check for course ID

IF present

Check for course prerequisites

IF present

Add data to array

Check values in parameters

IF less than 2

Skip course, print error message

ELSE

Add course name, number, prereq to table

IF prerequisite present

Add information to table

ELSE if no prerequisite present

Skip course, print error message

Create Course

Declare and initialize variables for course

WHILE open

FOR each course included

IF course & course ID match

Output course information

FOR each course in hash table

Output prerequisite information

END

**Binary Tree Pseudocode:**

START

Read CSV file

Function for total quantity, course prerequisites

FOR each prerequisite “p” in total number of prerequisites

Add left and right nodes to total number of prerequisites

Output total

Function to output course schedule

FOR each course node

Output course name

Check for course prequisites

IF course node left is present

Output course prerequisite

IF course node right is present

Output course prerequisite

Function to output course information

FOR each course node

IF course matches course number

Output course prerequisites

Check for course prerequisites

IF course node left present

Output course prerequisite & information

IF course node right present

Output course prerequisite & information

ELSE if node left available

Move information to node

ELSE if node right available

Move information to node

END

**Menu Pseudocode:**

START

Display menu and take user input

WHILE user input does not equal 9

Output “Choose an option”

Output “1. Load Data”

Output “2. Print Course List”

Output “3. Print Course”

Output “9. Exit”

Switch for menu, depending on user input

IF user input is 1

Output course data

IF user input is 2

Output course information

IF user input is 3

Output “Enter course ID:”

User input is taken

Output course number, title, prerequisites

IF user input is 9

Ouput “Goodbye”

Terminate program

END

**Output Sorted List:**

START

Create vector sorting string with s parameter

Partition method

Initialize low element, assign as first in list

Initialize high element, assign as last in list

Find midpoint; midpoint is low + (high – low) ÷ 2

Set pivot, assign as midpoint

WHILE pivot is < high element

High is assigned -1

Swap values to distribute values high to left and low to right; use temp variable val

Set low, assign as high

Set high to temp variable

Quick sort

Set mid, assign as equal to 0

Set low, assign as start

Set high, assign as finish

IF start is greater than or equal to finish

Return

Partition for low finish

Use recursive call for quick sort

Quick sort: courses array, low index, low index (end)

Quick sort: courses array, low index (end) +1, high index

Output display of course information

FOR all information available

Output course ID & prerequisite(s)

END

**Evaluation:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector Code** | **Line Cost** | **Executions** | **Total Cost** |
| Open and parse all courses | 1 | n | n |
| Check for course information and store | 1 | n | n |
| Output course information | 2 | 1 | 1 |
| Check for course prerequisite and store | 1 | N | n |
| Output prerequisite information | 2 | N | n |
| Total Cost = 5(n) + 1 | | | |
| Run Time = O(n) | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Hash Table Code** | | **Line Cost** | **Executions** | **Total Cost** |  |
| Open and parse all courses | | 2 | N | N |  |
| Check for course information and store | | 1 | N | N |  |
| Output course info | | 1 | 1 | 1 |  |
| Check for course prerequisite and store | | 2 | N | n |  |
| Output prerequisite information | | 4 | N | n |  |
|  | Total Cost = 9(n) + 1 | | | |
|  | Run Time = O(n) | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Binary Tree Code** | **Line Cost** | **# Executions** | **Total Cost** |  |
| Open and parse all courses | 1 | N | N |  |
| Check for course information and store | 1 | N | N |  |
| Output course info | 2 | 1 | 1 |  |
| Check for course prerequisite and store | 1 | N | N |  |
| Output prerequisite info | 4 | N | n |  |
| Total Cost = 7(n) + 1 | | | |  |
| Run Time = O(n) | | | |  |

For this assignment, we are tasked with creating a program to parse information from CSV file to read, organize, store, and output information regarding courses, course data and course prerequisites, and to find the most efficient way to create a satisfactory end product while reducing memory usage. Evaluating three different strategies for code structure, including vector, binary tree and hash table data structures, we are to choose based on the criteria listed above.

Each data structure has advantages and disadvantages, and it is our task to take the data collected with this evaluation and together with knowledge of the nature of the available structures, give a recommendation on which will be best suited for achieving our customer’s objectives.

Vectors came out with the fastest runtime, there are possible speed limitation factors for searching for specific data, as it performs an item by item search in an array. However, this structure does possess positive attributes such as resizing capabilities for the arrays, which can actually work to reduce runtime in some instances.

With the hash table method, keys are mapped to values as a way to organize within the structure. However, the search function feature of this method is slower in implementation. Each items must be parsed, sorted, and then printed. So the time to perform an alphanumeric search and sort can be lengthy.

Binary trees are fast, but require more code in the end to distribute between left and right nodes. The time to determine whether or not to distribute to one or the other node can add run time. However, binary tree searching can be more efficient if there are updates needed to the data inserted and deleted, as it is an effective way to distribute and organize the data.

Our analysis did yield a slight advantage for the vector method, and even with added benefits of the binary method including future insertion, deletion and maintenance of the list order, I recommend vector search based on the given criteria.