**//Menu (This is the same for all data structures)**

Initialize an instance of whatever data structure (vector, hash table, or binary tree) in use.

Initialize a new bid

Create an integer type variable and SAVE as userChoice

SET userChoice to 0

WHILE userChoice does not equal 9.

PRINT “Menu:” to screen.

PRINT “ 1. Load Data Structure” to screen.

PRINT “ 2. Print Course List” to screen.

PRINT “ 3. Print Course Information” to screen.

PRINT “ 9. Exit” to screen.

PRINT “Enter Choice” to screen.

GET input from user.

Create a switch based on userChoice

IF userChoice equals 1.

Call open and parse file function.

Create course objects for each line of the file and add to data structure.

IF userChoice equals 2.

Call the sort and print all courses function.

IF userChoice equals 3.

Call the print a course function

IF userChoice equals 9.

PRINT “Goodbye” to screen.

ELSE

PRINT “Invalid Choice” to screen.

SET userChoice to 0.

CLEAR input stream.

**Vector Pseudocode**

**//Open and parse file function creating course objects for each valid line**

Create a vector to hold a collection of course objects, save as programInfo

Create a integer type variable named “i” to iterate through loops.

Create a integer type variable named “j” to iterate through loops.

IF the input file is CSV file

Open input file

IF input file is open

WHILE the input file is good

FOR each line in the input file

GET the line from input file

IF line is not empty

IF the line has one comma (meaning at least 2 elements).

Create a course object and add to the collection of courses.

SAVE the first elements as courseNum

SAVE the second element as courseTitle

SET preReqs to null

ADD the course to the end of programInfo

ELSE IF the line has more than one comma

Count the number of commas and subtract one, SAVE as integer type variable named numPreReqs

Create a course object and add to the collection of courses.

SAVE the first element as courseNum

SAVE the second element as courseTitle

FOR each prerequisite (numPreReqs)

IF the prerequisite is equal to the first element on any row.

SAVE the elements as preReqs (1-n).

ELSE

PRINT “Prerequisite is not a valid course number.”

ADD the course to the end of programInfo

ELSE

PRINT “Error reading row”

CLOSE the input file

RETURN programInfo

**//Create course objects (this was addressed during parsing of the file but here is a general outline of the data the course object will hold.**

Create an object class named programCourse

The first string of the line is SAVED as the courseNum

The second string of the line is SAVED as the courseTitle

Count the number of prerequisites

For the number of prerequisites

IF the prerequisite matches the courseNum of any course

The subsequent strings of the line are SAVED as preReq(1-n)

ADD the programCourse to the vector called programInfo

**//Search and Print Course Info**

Create an integer type variable, save as “i” for loop variable

Create an integer type variable, save as “j” for inner loop variable

GET input from user for course number the user is looking for and SET as wantedCourse

FOR each programCourse in the vector programInfo

IF the course number is equal to the wantedCourse

print out the course information

for each prerequisite of the course

print the prerequisite course information

**//Partition function – accepts parameters of a vector to be sorted, an integer of low and an integer of high.**

SET low and high equal to begin and end

Pick the middle element as pivot point

WHILE not done

Keep incrementing low index while courseNum[low] < courseNum[pivot]

Keep decrementing high index while courseNum[pivot] is less than courseNum[high]

IF low is greater than or equal to high

SET done equal to true.

ELSE swap the low and high courses (built in vector method)

INCREMENT low

DEINCREMENT high

Return high

**// Sort function – accepts parameter of vector to be sorted, integer of start point and int of end point.**

Create an integer type variable for middle point of vector. SET to 0.

Base case:

IF there are 1 or zero courses to sort,

Partition is already sorted otherwise if begin is greater than or equal to end

Return

ELSE – Partition courses into low and high such that the midpoint is location of last element in low

SET middle point equal to partition of beginning and end of courses.

Recursively sort low partition (begin to mid).

Recursively sort high partition (mid+1 to end).

**// Sorted Print function – accepts vector of the courses to be displayed**

Call the sort function on the vector for programInfo

Loop and display the course info:

FOR all courses in programInfo

PRINT courseNum, courseTitle, and PreReqs

**Hash Table Pseudocode**

**//Open and parse file function creating course objects for each valid line**

Create a vector to hold a collection of course objects, save as programInfo

Create an integer type variable named “i” to iterate through loops.

Create an integer type variable named “j” to iterate through loops.

IF the input file is CSV file

Open input file

IF input file is open

WHILE the input file is good

FOR each line in the input file

GET the line from input file

IF line is not empty

IF the line has one comma (meaning at least 2 elements).

Create a course object and add to the collection of courses.

SAVE the first elements as courseNum

SAVE the second element as courseTitle

SET preReqs to null

ADD the course to the end of programInfo

ELSE IF the line has more than one comma

Count the number of commas and subtract one, SAVE as integer type variable named numPreReqs

Create a course object and add to the collection of courses.

SAVE the first element as courseNum

SAVE the second element as courseTitle

FOR each prerequisite (numPreReqs)

IF the prerequisite is equal to the first element on any row.

SAVE the elements as preReqs (1-n).

ELSE

PRINT “Prerequisite is not a valid course number.”

CALCULATE the hash key of the courseNum

ADD the course to the end of programInfo at the location of the hash key

ELSE

PRINT “Error reading row”

CLOSE the input file

RETURN programInfo

**//Create course objects (this was addressed during parsing of the file but here is a general outline of the data the course object will hold.**

Create an object class named programCourse

The first string of the line is SAVED as the courseNum

The second string of the line is SAVED as the courseTitle

Count the number of prerequisites

For the number of prerequisites

IF the prerequisite matches the courseNum of any course

The subsequent strings of the line are SAVED as preReq(1-n)

CALCULATE the hash key of the courseNum.

ADD the programCourse to the vector called programInfo at the location of the hash key.

**//Search and Print Course Info**

Create an integer type variable, save as “i” for loop variable

Create an integer type variable, save as “j” for inner loop variable

GET input from user for course number the user is looking for and SET as wantedCourse

CALCULATE hash of the courseNum of wantedCourse

FOR each programCourse in the vector programInfo at the location of wantedCourse hash value.

IF the course number is equal to the wantedCourse

PRINT out the course information

FOR each prerequisite of the course

CALCULATE the hash value of the courseNum

Locate the information of the prerequisite course by searching the values at the hash key

PRINT the prerequisite course information

**//Sort and Print function – Hash tables are unordered data structures. To sort data list must be made.**

Create a linked list for courses listed in hash table.

FOR all hash values in table programInfo

IF hash value is not empty.

IF linked list is empty

Add first course listed in hash value to linked list as head and tail.

Iterate through hash value adding courses in to list in order by courseNum

ELSE

For each course in hash value add course to list in order by courseNum

FOR all courses in linked list

PRINT courseNum, courseTitle and preReqs.

**Binary Tree Pseudocode**

**//Open and parse file function creating course objects for each valid line**

Create a binary tree to hold a collection of course objects, save as programInfo

Create an integer type variable named “i” to iterate through loops.

Create an integer type variable named “j” to iterate through loops.

IF the input file is CSV file

Open input file

IF input file is open

WHILE the input file is good

FOR each line in the input file

GET the line from input file

IF line is not empty

IF the line has one comma (meaning at least 2 elements).

Create a course object and add to the collection of courses.

SAVE the first elements as courseNum

SAVE the second element as courseTitle

SET preReqs to null

ADD the course to programInfo at the location of courseNum.

ELSE IF the line has more than one comma

Count the number of commas and subtract one, SAVE as integer type variable named numPreReqs

Create a course object and add to the collection of courses.

SAVE the first element as courseNum

SAVE the second element as courseTitle

FOR each prerequisite (numPreReqs)

IF the prerequisite is equal to the first element on any row.

SAVE the elements as preReqs (1-n).

ELSE

PRINT “Prerequisite is not a valid course number.”

Create a new Node named newCourse

Create a temporary pointer named currentCourse

SET currentCourse to root node.

IF tree is empty.

ADD the new Node to the binary tree called programInfo as the root node

ELSE if tree is not empty

WHILE the currentNode is not equal to null pointer

IF newCourse course number is less than the root node course number

Recurse down the left subtree

IF newCourse course number is greater than or equal to the root node

Recurse down the right subtree

ADD newCourse to the binary tree at the currentNode location

ELSE

PRINT “Error reading row”

CLOSE the input file

RETURN programInfo

**//Create course objects (this was addressed during parsing of the file but here is a general outline of the data the course object will hold.**

Create an object class named programCourse

The first string of the line is SAVED as the courseNum

The second string of the line is SAVED as the courseTitle

Count the number of prerequisites

For the number of prerequisites

IF the prerequisite matches the courseNum of any course

The subsequent strings of the line are SAVED as preReq(1-n)

Create a new Node named newCourse

Create a temporary pointer named currentCourse

SET currentCourse to root node.

IF tree is empty.

ADD the new Node to the binary tree called programInfo as the root node

ELSE if tree is not empty

WHILE the currentNode is not equal to null pointer

IF newCourse course number is less than the root node course number

Recurse down the left subtree

IF newCourse course number is greater than or equal to the root node

Recurse down the right subtree

ADD newCourse to the binary tree at the currentNode location

**//Search and Print Course Info**

Create an integer type variable, save as “i” for loop variable

Create an integer type variable, save as “j” for inner loop variable

GET input from user for course number the user is looking for and SET as wantedCourse

Create a temporary pointer, SAVE as currentCourse.

SET currentCourse equal to the root course node.

IF tree is not empty.

IF currentCourse course number is not equal to wantedCourse course number

WHILE wantedCourse course number is less than currentCourse

Recursively search down left subtree

WHILE wantedCourse course number is greater than currentCourse

Recursively search down right subtree.

IF currentCourse is equal to wanted Course

PRINT out the course information

FOR each prerequisite of the course

Set wantedCourse to prerequisite

SET currentCourse to root node

Search tree for match

When match found

PRINT the prerequisite course information

**//Sorted Print Function – includes the following 2 functions**

**Function to Traverse the tree in order**

call inOrder fuction and pass root

**The inOrder function – accepts a node pointer as parameter**

IF the node passed is not equal to null ptr

Recurse down left

PRINT courseNum, courseTitle and preReqs

Recurse down right

**EVALUATION**

| **Code for parsing input and using a vector for data structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create a vector to hold a collection of course objects, save as programInfo | 1 | 1 | 1 |
| Create a integer type variable named “i” to iterate through loops. | 1 | 1 | 1 |
| Create a integer type variable named “j” to iterate through loops. | 1 | 1 | 1 |
| IF the input file is CSV file | 1 | 1 | 1 |
| Open input file | 1 | 1 | 1 |
| IF input file is open | 1 | 1 | 1 |
| WHILE the input file is good | 1 | n | n |
| FOR each line in the input file | 1 | n | n |
| GET the line from input file | 1 | n | n |
| IF line is not empty | 1 | n | n |
| IF the line has one comma (meaning at least 2 elements). | 1 | n | n |
| Create a course object and add to the collection of courses. | 1 | n | n |
| SAVE the first elements as courseNum | 1 | n | n |
| SAVE the second element as courseTitle | 1 | n | n |
| SET numPreReqs to Null | 1 | n | n |
| ADD the course to the end of programInfo | 1 | n | n |
| ELSE IF the line has more than one comma | 1 | n | n |
| Count the number of commas and subtract one, SAVE as integer type variable named numPreReqs | 1 | n | n |
| Create a course object and add to the collection of courses. | 1 | n | n |
| SAVE the first element as courseNum | 1 | n | n |
| SAVE the second element as courseTitle | 1 | n | n |
| FOR each prerequisite (numPreReqs) | 1 | n | n |
| IF the prerequisite is equal to the first element on any row. | 1 | n | n |
| SAVE the elements as preReqs (1-n). | 1 | n | n |
| ELSE | 1 | n | n |
| PRINT “Prerequisite is not a valid course number.” | 1 | n | n |
| ADD the course to the end of programInfo | 1 | n | n |
| ELSE | 1 | n | n |
| PRINT “Error reading row” | 1 | n | n |
| Close the input file | 1 | 1 | 1 |
| Return programInfo | 1 | 1 | 1 |
| **Total Cost** | | | 22n + 8 |
| **Runtime** | | | O(n) |

| **Code for parsing input and using a hash table for data structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create a vector to hold a collection of course objects, save as programInfo | 1 | 1 | 1 |
| Create a integer type variable named “i” to iterate through loops. | 1 | 1 | 1 |
| Create a integer type variable named “j” to iterate through loops. | 1 | 1 | 1 |
| IF the input file is CSV file | 1 | 1 | 1 |
| Open input file | 1 | 1 | 1 |
| IF input file is open | 1 | 1 | 1 |
| WHILE the input file is good | 1 | n | n |
| FOR each line in the input file | 1 | n | n |
| GET the line from input file | 1 | n | n |
| IF line is not empty | 1 | n | n |
| IF the line has one comma (meaning at least 2 elements). | 1 | n | n |
| Create a course object and add to the collection of courses. | 1 | n | n |
| SAVE the first elements as courseNum | 1 | n | n |
| SAVE the second element as courseTitle | 1 | n | n |
| SET numPreReqs to Null | 1 | n | n |
| ADD the course to the end of programInfo | 1 | n | n |
| ELSE IF the line has more than one comma | 1 | n | n |
| Count the number of commas and subtract one, SAVE as integer type variable named numPreReqs | 1 | n | n |
| Create a course object and add to the collection of courses. | 1 | n | n |
| SAVE the first element as courseNum | 1 | n | n |
| SAVE the second element as courseTitle | 1 | n | n |
| FOR each prerequisite (numPreReqs) | 1 | n | n |
| IF the prerequisite is equal to the first element on any row. | 1 | n | n |
| SAVE the elements as preReqs (1-n). | 1 | n | n |
| ELSE | 1 | n | n |
| PRINT “Prerequisite is not a valid course number.” | 1 | n | n |
| CALCULATE the hash key of the courseNum | 1 | n | n |
| ADD the course to the end of programInfo at the location of the hash key | 1 | n | n |
| ELSE | 1 | n | n |
| PRINT “Error reading row” | 1 | n | n |
| Close the input file | 1 | 1 | 1 |
| Return programInfo | 1 | 1 | 1 |
| **Total Cost** | | | 23n + 8 |
| **Runtime** | | | O(n) |

| **Code for parsing input and using a binary tree for data structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Create a binary tree to hold a collection of course objects, save as programInfo | 1 | 1 | 1 |
| Create a integer type variable named “i” to iterate through loops. | 1 | 1 | 1 |
| Create a integer type variable named “j” to iterate through loops. | 1 | 1 | 1 |
| IF the input file is CSV file | 1 | 1 | 1 |
| Open input file | 1 | 1 | 1 |
| IF input file is open | 1 | 1 | 1 |
| WHILE the input file is good | 1 | n | n |
| FOR each line in the input file | 1 | n | n |
| GET the line from input file | 1 | n | n |
| IF line is not empty | 1 | n | n |
| IF the line has one comma (meaning at least 2 elements). | 1 | n | n |
| Create a course object and add to the collection of courses. | 1 | n | n |
| SAVE the first elements as courseNum | 1 | n | n |
| SAVE the second element as courseTitle | 1 | n | n |
| SET numPreReqs to Null | 1 | n | n |
| ADD the course to the end of programInfo | 1 | n | n |
| ELSE IF the line has more than one comma | 1 | n | n |
| Count the number of commas and subtract one, SAVE as integer type variable named numPreReqs | 1 | n | n |
| Create a course object and add to the collection of courses. | 1 | n | n |
| SAVE the first element as courseNum | 1 | n | n |
| SAVE the second element as courseTitle | 1 | n | n |
| FOR each prerequisite (numPreReqs) | 1 | n | n |
| IF the prerequisite is equal to the first element on any row. | 1 | n | n |
| SAVE the elements as preReqs (1-n). | 1 | n | n |
| ELSE | 1 | n | n |
| PRINT “Prerequisite is not a valid course number.” | 1 | n | n |
| Create a new Node named newCourse | 1 | n | n |
| Create a temporary pointer named currentCourse | 1 | n | n |
| SET currentCourse to root node. | 1 | n | n |
| IF tree is empty. | 1 | n | n |
| ADD the new Node to the binary tree called programInfo as the root node | 1 | n | n |
| ELSE if tree is not empty | 1 | n | n |
| WHILE the currentNode is not equal to null pointer | 1 | n | n |
| IF newCourse course number is less than the root node course number | 1 | n | n |
| Recurse down the left subtree | 1 | Best – logN  Worst - n | Best – logN  Worst - n |
| IF newCourse course number is greater than the root node course number | 1 | n | n |
| Recurse down the right subtree | 1 | Best – logN  Worst - n | Best – logN  Worst - n |
| ADD newCourse to the binary tree at the currentNode location | 1 | n | n |
| ADD the course to the end of programInfo | 1 | n | n |
| ELSE | 1 | n | n |
| PRINT “Error reading row” | 1 | n | n |
| Close the input file | 1 | 1 | 1 |
| Return programInfo | 1 | 1 | 1 |
| **Total Cost** | | | 35n + 8 |
| **Runtime** | | | O(n) |

**Analysis**

While hash tables allow for real time indexing since it uses a key to find the data needed; it is an unordered structure and does not allow for sorting without first copying the courses into a different data structure which seems wasteful. The binary tree and vector structures both have O(n) runtimes for adding the data to the structure but the binary tree takes more memory to do so as it must save pointers for both left and right nodes. While searching the binary tree is faster than the searching the unsorted vector (albeit slower than the hash table which has already been discarded as an option). The binary tree, however, can run into issues is the data is not in a random order before being stored, instead of a tree you get a linked list negating any benefit in search speed and possibly leading to bugs that crash the program. Based on my understanding of the requirements the academic advisors at ABCU have for the program I recommend using a binary tree to store the course data. The binary tree sorts as it inserts which makes printing the course list in order and searching much faster.