Vector:

**Open the file:**

**Utilize fstream and open file.**

**file = openfile(“”)**

**Make sure the file opened correctly:**

**If (file != null)**

**Else {**

**Return error(“file is null”)**

**Read file**

**Foreach (line in file)**

**Split lines into components for title, course #, and prereqs**

**Check if each file has a minimum of 2 components**

**If (course.length >= 2)**

**courseNum = course[0]**

**courseTitle = course[1]**

**coursePrereq = course[2]**

**Else If (course.Length < 2) {**

**Return Error (“file has less than 2 components”)**

**}**

**Else**

**Return Error (“File has more than 3 components”)**

**Create course objects**

**newCourse = createNewCourse (courseNum, courseTitle, prereq)**

**Store course into a vector**

**courseVector.Add(newCourse)**

**else{**

**print(“Error: file is incorrect in line “, line)**

**Close the file**

**CloseFile(file)**

**Else {**

**Print(“Error: Not able to open file”)**

**Create a function that is able to search and print course info**

**New function PrintCourseInfo(courseCode)**

**Found = False;**

**Ask for input from user**

**Input = courseCode**

**Foreach(course in courseVector)**

**If (course.course.Number == courseCode)**

**{**

**Print out info:**

**Print(“The course number is: “ , course.CourseNum)**

**Print(“The course title is: “ , course.courseTitle)**

**Print(“The prerequisites are: “ , course.coursePrereq)**

**Found = True;**

**}**

**If(found = False) {**

**Print(“The course was not found”)**

Hash Tables:

**Open the file:**

**Utilize fstream and open file.**

**file = openfile(“”)**

**Make sure the file opened correctly:**

**If (file != null)**

**Else {**

**Return error(“file is null”)**

**Read file**

**Foreach (line in file)**

**Split lines into components for title, course #, and prereqs**

**Check if each file has a minimum of 2 components**

**If (course.length >= 2)**

**courseNum = course[0]**

**courseTitle = course[1]**

**coursePrereq = course[2]**

**Else If (course.Length < 2) {**

**Return Error (“file has less than 2 components”)**

**}**

**Else**

**Return Error (“File has more than 3 components”)**

**Create course objects**

Create Hash Table class

Define private structures that hold bids

Create constructors

Initialize with a bid

Initialize with a bid and a key

Resize Hash tables size to the table size given.

Call up methods for Insert, PrintAll, Remove, and Bid Search

Create Insert bid method

Create keys for the given bids

Retrieve node based on key

If no entry node was found for the key

Assign the current node to the key position

Else

If no node is used

Assign old node to UNIT\_MAX

Set to key

Set old not to the bid and the old node next to nullptr

Else

While loop (find next open node)

Add new node to end

Create PrintAll Method

For(nodes from beginning to end)

If(key is != UINT\_MAX)

Output key, bidId, title, amount, and fund

Node = next iter

While (node != nullptr)

Output key, bidId, title, amount, and fund

Node = next node

Create Remove Method

Set the key = to the hash atoi bidId cstring

Erase node begin and the key

Create Search Method

Create key for given bid

If (entry is found for the key)

Return node bid

If(no entry is found for the key)

Return bid

While (node1!= nullptr)

Return node->bid

Return bid

Tree Data:

Create Hash Table class

Define private structures that hold bids

Create constructors

Initialize with a bid

Initialize with a bid and a key

Resize Hash tables size to the table size given.

Call up methods for Insert, PrintAll, Remove, and Bid Search

Create Insert bid method

Create keys for the given bids

Retrieve node based on key

If no entry node was found for the key

Assign the current node to the key position

Else

If no node is used

Assign old node to UNIT\_MAX

Set to key

Set old not to the bid and the old node next to nullptr

Else

While loop (find next open node)

Add new node to end

Create PrintAll Method

For(nodes from beginning to end)

If(key is != UINT\_MAX)

Output key, bidId, title, amount, and fund

Node = next iter

While (node != nullptr)

Output key, bidId, title, amount, and fund

Node = next node

Create Remove Method

Set the key = to the hash atoi bidId cstring

Erase node begin and the key

Create Search Method

Create key for given bid

If (entry is found for the key)

Return node bid

If(no entry is found for the key)

Return bid

While (node1!= nullptr)

Return node->bid

Return bid

Menu:

Int inputValue = 0;

Create Bid;

While (inputValue != 4)

{ cout<< “1. Load Data Structure” << endl;

cout<< “1. Load Data Structure” << endl;

cout<< “2. Course List” << endl;

cout<< “3. Course” << endl;

cout<< “4. Exit” << endl;

switch(inputValue)

case 1:

cout << “loadBids(bid)” << endl;

break;

case 2:

cout << “Course List “<< endl;

break;

case 3:

cout << “Course ” << endl;

break;

case 4:

cout << “Exit” << endl;

break;

default:

Cout << error<< endl;

1. Evaluate the run-time and memory of data structures that could be used to address the requirements. In a previous assignment, you created pseudocode to do the following:
   1. Define how the program opens the file, reads the data from the file, parses each line, and checks for formatting errors.

A program opens the file and then reads each line while parsing them for course information. While it is checking, it makes sure that each line has each parameter and if not it will spit out an error message.

* 1. Show how to create course objects, so that one course object holds data from a single line from the input file.

Using this pseudocode written for the previous assignments, analyze the worst-case running time of each, reading the file and creating course objects, which will be the Big O value. This should not include the pseudocode written for the menu or the sample schedule above. To do this, do the following:

* 1. Specify the cost per line of code and the number of times the line will execute. Assume there are n courses stored in the data structure.
  2. Assume the cost for a line to execute is 1 unless it is calling a function, in which case the cost will be the running time of that function.

To read the file, it is a O(1) \* ‘m’. This is because the number of lines is m and it has to read each line.

To create the course objects, it is a the same ratio (a O(1) \* ‘m’), but m is the courses in each line.

1. Based on the advisor’s requirements, analyze each data structure (vector, hash table, and tree). Explain the advantages and disadvantages of each structure in your evaluation.

Vector:

Advantages:

You get instant access to the elements by utilizing the numbering their locations and finding them by their numbers. This is because vectors order the items in a straight line by item number.

Disadvantages

Vectors are slow to reorder. You cannot just add a new number into it, you need to move the values around to get it into the correct location. Another disadvantage is that it is a fixed size, so you need to plan ahead on how much data it will hold.

Hash Tables:

Advantages:

Search is very fast in Hash tables, this is because it holds/sorts values together where you can quickly find and access them. Another advantage is that they are not fixed sized, so it’s easy to add and remove items.

Disadvantages:

A big flaw is that it uses up a lot of space. It also it difficult to keep sorted because it can quickly be overfilled in one section. Because it is easy to sort items into one section, it takes a good plan to make sure it is set up correctly.

Linked Tree:

Advantages:

These are easy to keep neat and sorted. The values are split in half, so it is easy to sort and locate items. Because it is sorted by half, it is quick to find the item as well as easy to input/remove values.

Disadvantages

The problem with Linked trees is that is takes up a large amount of data. It also takes a long time because it has to set up the values in a specific way.

1. Now that you have analyzed all three data structures, make a recommendation for which data structure you will plan to use in your code. Provide justification for your recommendation, based on the Big O analysis results and your analysis of the three data structures.

I would use a Hash Table because we need to find the course information quickly, sort them (based on their values), and add/remove courses as they come. The Hash table is a good choice to read the information from a file, create course objects from the file, and then sort them where they go. In this instance each line would be ‘m’ for (O(1) \* ‘m’). Another reason why Hash tables would be best for this is because once you plan and sort the hash tables correctly, it is easy to add and remove new values.