**//Code for opening reading and loading into a structure**

void ReadFile(String pathtofile, Structure<Course> courses){

LoadFile(pathtofile)

Try:

For each row in the file at pathtofile

Add row[0] = Course.number

Add row[1] = Course.title

Add row[2] = Course.prerequisite

. . . . (until null in the row)

Courses.push\_back(Course)

Catch: Throw an error likely if the file is not found or we go out of range.

}

**//Code for main menu**

void main()

int choice = 0

while (choice not equal to 9){

print menu options to screen

switch(choice){

// whenever structure is passed in that is choosing to use // a structure, not needed in final program since analysis // will show which is best.

case 1:

ReadFile(pathtofile, structure)

Case 2:

printSampleSchedule(Structure)

Case 3:

printCourseInformation(Structure, course num)

case 9:

Not needed choosing 9 will exit the loop

}

}

**// Vector pseudocode**

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

void printSampleSchedule(Vector<Course> courses) {

Vector<Course> temp vector

Loop through list read in by file until all rows are in temp vector

If any value after position 1 in the row is null (no pre-requisites)

Add to end of temp vector

Else If not null but prerequisite is already in courses

Add to end of temp vector

Else

Pass row for now

Print temp vector

}

void printCourseInformation(Vector<Course> courses, String courseNumber) {

for all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the course

print the prerequisite course information

}

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| For all courses | 1 | N | N |
| If the course is the same as courseNumber | 1 | N | N |
| Print out the course information | 1 | 1 | 1 |
| For each prerequisite of the course | 1 | N | N |
| Prinmt the prerequisite course information | 1 | N | N |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

**// Hashtable pseudocode**

int numPrerequisiteCourses(Hashtable<Course> courses) {

if course is the same as courseNumber

print the prerequisite course information

}

void printSampleSchedule(Hashtable<Course> courses) {

For a course at the beginning (CSCI100 or MATH201)

Print that course then remove from beginning

Move to next position in hash table – print all objects pointed at then remove

Continue until you reach the node that points at nothing(CSCI400)

}

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

find appropriate hash bucket

If the course is the same as the course number

Print out the course information

Else

Print not found

}

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Find appropriate hash bucket | 1 | 1 | 1 |
| If course is found in hash bucket | 1 | n | n |
| Print course information | 1 | 1 | 1 |
| Else | 1 | 1 | 1 |
| Print Not found | 1 | 1 | 1 |
| **Total Cost** | | | N + 4 |
| **Runtime** | | | O(n) |

**// Tree pseudocode**

int numPrerequisiteCourses(Tree<Course> courses) {

Starting at the root directory until you find a node that points to nothing (the bottom of the Tree)

Count the keys of each child node you cross.

}

void printSampleSchedule(Tree<Course> courses) {

Loop through all nodes until a node that points at nothing is found

Print that course (should be bottom of the tree)

Then move up to the parent and print that and repeat.

}

void printCourseInformation(Tree<Course> courses, String courseNumber) {

Start at root

While (node doesn’t point to null) {

If (node = courseNumber) {

Print node info

}

Else {

Next node

}

}

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Start at root | 1 | 1 | 1 |
| While node doesn’t point to null | 1 | n | N |
| If node matches courseNumber | 1 | n | N |
| Print node info | 1 | 1 | 1 |
| Else next node | 1 | n | N |
| **Total Cost** | | | 3n + 1 |
| **Runtime** | | | O(logN) |

**// Advantages and Disadvantages**

**Vector:**

**Advantages:**

* **Good for small data sets**
* **Intuitive and easy to manipulate**

**Disadvantages:**

* **Worst case is the longest search time, must iterate through all elements.**
* **Have to resize whenever elements are added.**

**Hashtable:**

**Advantages:**

* **Best case is the fastest search O(1)**
* **Averages better run time than trees.**

**Disadvantages:**

* **Potential for collisions**
* **Worst case run-time is the same as vector iterates through all elements.**
* **Difficult to get elements in order.**

**Trees:**

**Advantages:**

* **Easy to get elements in order.**
* **Can always keep the cost of operations to O(logN)**
* **Good for large data sets**

**Disadvantages:**

* **Very complicated and not necessary for small numbers of lists**
* **Only works well on lists that are sorted.**