**Vector**

SET vector <courses> course

Open file

IF file does not open

PRINT invalid file

CLOSE program

PARSE file into vector

For i = 0; i < row.count ;i++

Courses course

Course.courseId [i][1]

Course.courseName[i][2]

For j = 3; j <= coloum.count; j++

Course.coursePre[i][j]

IF coursePre does not equal any courseId

Print invalid course

Close program

int numPrerequisiteCourses

GET user input of what course

Get Prerequisites for class store in temp vector

Then check Prerequisites for Prerequisites and store into same vector

while i not equals vector size

If vector [i] equals vector [i+1]

Remove vector [i]

increment totalPrerequisites by one

Print totalPrerequisites

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Compare user input to element in vector** | 1 | n | n |
| **Load** Prerequisites into vector **for each prerequisite of the course** | 1 | n | n |
| **Compare element in temp vecoter to courseId in vector** | 1 | n | n |
| **Load** Prerequisites into vector **for each prerequisite of the course** | 1 | n | n |
| **Increment through temp vector and remove repents** | 1 | n | n |
| **increment totalPrerequisites by one** | 1 | n | n |
| Print totalPrerequisites | 1 | 1 | 1 |
| **Total Cost** | | | 7n + 1 |
| **Runtime** | | | O(n) |

printSampleSchedule

if no Prerequisites in a course add to temp vector

print vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Check if course id has prerequisite if no prerequisite, add to temp vector** | 1 | n | n |
| Print temp vector | 1 | n | n |
| **Total Cost** | | | 2n |
| **Runtime** | | | O(n) |

printCourseInformation

get user input = course

Get Prerequisites for course and store in temp vector

Then check Prerequisites for Prerequisites and store into vector

Print user input with relevant course information

For i <= vector size

Print out relevant Prerequisite’s course information from vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Compare user input to element in vector** | 1 | n | n |
| **Print course information** | 1 | 1 | 1 |
| **Print temp vector** | 1 | n | n |
| **Total Cost** | | | 3n + 1 |
| **Runtime** | | | O(n) |

Int partition

Int low equals begin

Int high equals end

Int pivot = begin plus (end – begin) divide by two

Bool done equals false

While not done

While course.at(low).courseId less than course.at(pivot).courseId

Low plus 0ne

While course.at(pivot).courseId less than course.at(high).courseId

High minus 0ne

If low less than or equal to high

Done equals true

Else

Swap(course[low],course[high]

Low plus 0ne

High minus 0ne

Return high

Int InOrder()

Int mid equal zero

If begin greater than or equal to end

Return

Mid = call method partition( courses,begin , end)

InOrder (courses, begin , mid-1)

InOrder (courses, mid +1,end)

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Check current order** | 1 | n | n |
| **Load** Prerequisites into vector **for each prerequisite of the course** | 1 | n | n |
| **Call method partition** | 1 | 1 | 1 |
| **In order begin too mid** | 1 | n | n |
| **In order mid to end** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(N log N) |

**Hash Table**

SET vector <courses> course

SET int hash( int key)

Open file

IF file does not open

PRINT invalid file

CLOSE program

PARSE file into vector

For i = 0; i < row.count ;i++

Courses course

Course.courseId [i][1]

Course.courseName[i][2]

For j = 3; j <= coloum.count; j++

Course.coursePre[i][j]

Hash pointer insert(course)

IF coursePre does not equal any courseId

Print invalid course

Close program

int numPrerequisiteCourses()

GET user input of what course

Convert user input into key =hash(atoi(course.courseID.c\_str()));

Get Prerequisites for course from user input

While not null

Compare key and courseTable

If they match store node\* coursePre in a temp vector

Key = coursePre

while i not equals vector size

If vector [i] equals vector [i+1]

Remove vector [i]

increment totalPrerequisites by one

Print totalPrerequisites

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Convert user input into key** | 1 | 1 | 1 |
| **Compare key to coursetable** | 1 | 1 | 1 |
| **Increment through temp vector and remove repents** | 1 | n | n |
| **increment totalPrerequisites by one** | 1 | n | n |
| Print totalPrerequisites | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 3 |
| **Runtime** | | | O(1) |

PrintSampleSchedule()

Int key =hash(atoi(course.coursePre.c\_str()));

If key is not null

Print course

Increment to next node.coursePre

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Convert user input into key** | 1 | 1 | 1 |
| **If no** Prerequisites **Print course** | 1 | n | n |
| **Total Cost** | | | 2n + 1 |
| **Runtime** | | | O(n) |

printCourseInformation()

GET user input of what course

Convert user input into key =hash(atoi(course.courseID.c\_str()));

If key equal courseTable

Print course

while key at coursePre not null

if course has Prerequisites store in temp vector

increment node to tempVector[i]

while i not equals vector size

If vector [i] equals vector [i+1]

Remove vector [i]

Print user input with relevant course information

For i <= vector size

Print out relevant Prerequisite’s course information from vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Convert user input into key** | 1 | n | n |
| **Print course** | 1 | 1 | 1 |
| **Compare element in temp vecoter to courseId in vector** | 1 | n | n |
| **Load** Prerequisites into vector **for each prerequisite of the course** | 1 | n | n |
| **Increment through temp vector and remove repents** | 1 | n | n |
| Print Prerequisites | 1 | n | 1 |
| **Total Cost** | | | 6n + 1 |
| **Runtime** | | | O(n) |

InOrder()

For i equal nodes begin(); i not equal node.end(); add one to i

If i pointer not equal to UINT\_MAX

Print node pointer course.courseId

Print node pointer course.courseName

while node pointer course.courseId has coursePre

Print node pointer course.coursePre

Node\* i pointer next

While node not null

Print node pointer course.courseId

Print node pointer course.courseName

while node pointer course.courseId has coursePre

Print node pointer course.coursePre

Node = node next

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Print** course.courseId | 1 | n | n |
| Print course.courseName | 1 | n | n |
| while node pointer course.courseId has coursePre  Print node pointer course.coursePre | 2 | n | n |
| **Total Cost** | | | 4n |
| **Runtime** | | | O(n) |

**Tree Data Structure**

SET vector <courses> course

IF file does not open

PRINT invalid file

CLOSE program

PARSE file into vector

For i = 0; i < row.count ;i++

Courses course

Course.courseId [i][1]

Course.courseName[i][2]

For j = 3; j <= coloum.count; j++

Course.coursePre[i][j]

bst pointer insert(course)

IF coursePre does not equal any courseId

Print invalid course

Close program

int numPrerequisiteCourses()

GET user input of what course

New node current = root

Set vector temp

Get Prerequisites for course from user input

While current not null

Compare userInput and current

If current pointer course.courseId equals userInput

while current pointer course.courseId has coursePre

temp.append(coursePre)

if userInput less than current pointer course.courseId

current = current pointer left

else

current = current pointer right

while i not equals vector size

If vector [i] equals vector [i+1]

Remove vector [i]

increment totalPrerequisites by one

Print totalPrerequisites

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Get user input** | 1 | 1 | 1 |
| **Load** Prerequisites into vector **for each prerequisite of the course** | 6 | n | n |
| **Compare element in temp vecter to courseId in vector** | 1 | n | n |
| **Increment through temp vector and remove repents** | 1 | n | n |
| **increment totalPrerequisites by one** | 1 | n | n |
| Print totalPrerequisites | 1 | 1 | 1 |
| **Total Cost** | | | 11n + 2 |
| **Runtime** | | | O(n) |

PrintSampleSchedule()

While current not null

If current pointer course.courseId does not have coursePre

Print course

if userInput lest than current pointer course.courseId

current = current pointer left

else

current = current pointer right

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Check if each course has** Prerequisites | 3 | n | n |
| **Print course** | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

printCourseInformation()

New node current = root

Set vector temp

GET user input of what course

While current not null

Compare userInput and current

If current pointer course.courseId equals userInput

while current pointer course.courseId has coursePre

temp.append(coursePre)

if userInput less than current pointer course.courseId

current = current pointer left

else

current = current pointer right

while temp is not null

if course has Prerequisites store in temp vector

increment node to tempVector[i]

while i not equals vector size

If vector [i] equals vector [i+1]

Remove vector [i]

Print user input with relevant course information

For i <= vector size

Print out relevant Prerequisite’s course information from vector

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Compare user input to element in vector** | 1 | n | n |
| **Load** Prerequisites into vector **for each prerequisite of the course** | 1 | n | n |
| **Compare element in temp vecoter to courseId in vector** | 1 | n | n |
| **Load** Prerequisites into vector **for each prerequisite of the course** | 1 | n | n |
| **Increment through temp vector and remove repents** | 1 | n | n |
| **increment totalPrerequisites by one** | 1 | n | n |
| Print totalPrerequisites | 1 | 1 | 1 |
| **Total Cost** | | | 7n + 1 |
| **Runtime** | | | O(n) |

InOrder()

This pointer inOrder root

inOrder if node is not null

inOrder node pointer left

Print node pointer course.courseId

Print node pointer course.courseName

while node pointer course.courseId has coursePre

Print node pointer course.coursePre

inOrder node pointer right

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| inOrder node pointer left | 1 | n | n |
| Print node pointer course.courseId | 1 | n | n |
| Print node pointer course.courseName | 1 | n | n |
| while node pointer course.courseId has coursePre  Print node pointer course.coursePre | 2 | n | n |
| inOrder node pointer right | 1 | n | n |
| **Total Cost** | | | 6n |
| **Runtime** | | | H=log2n |

**Menu**

While not equal 9

Print Menu

Print 1. Load Data structure

Print 2. Course List

Print 3. Print Course

Print 9. Exit

Get userInput

Switch statement using userInput

Case one

Call loadCouse method

Case two

Call InOrder method

Print “ Here is a sample schedule.”

Call PrintSampleSchedule()

Case three

Print “what course would you like to know about?”

Call printCourseInformation()

Case nine

Print “ Thank you for using the course planner!”

Default

Print “Not a valid option”

**Advantages and Disadvantages**

**Vector:**

Using vectors to store data is rather straightforward and to manipulate the data does not need to be converted to compare elements. Vectors are a go way to quickly store data but is very time consuming to and expensive to maintain for the long term. In terms of algorithmic use, it is the slowest and can be very complex with large data sets. Vectors are much better used in conjunction with other algorithms. In terms of the big O at best the speed for a quick sort is and the worst speed would be .

**Binary Search Tree:**

Binary search trees are way to store data in a tree format where the value of the data will indicate where the data located on the tree. In general BST are much faster than link lists and vectors. The tree one problem with a BST is that you need a separate tree for each item. So in terms of this assignment, you would need a separate tree for courseId, And each Prerequisite. In terms of the big O while doing a search through a tree at best the speed is and at worst the speed is .

**Hash Table:**

Of the three data structures that we are using these are the fastest and the most efficient data structure. They tend to have a constant time complexity of O(1) but with large lists they have a potential of collisions and are limited to the size of the memory. This problem can be mitigated by rehashing or resizing. This is very time consuming with the liner O(N). Hash tables also provide simple security as the key can be encrypted.

**Recommendation:**

For the most part my program is very linear with exception of certain specialize functions of each data structure. I have decided to use the hash table data structure since it is the fastest with the constant time of O(1). I also feel very confident in creating this program with the guidance of the hash table assignment and the internet. I will also be using vectors to store temporary lists to print out to the console. This provides a lot of linearity to my program but I feel it is the best way to meet the expectations of project two.