Roger Newton
CS 300

Project One Pseudocode

-Vector-

File Input
Open file

If the file doesn't open
    Print “error opening file”
    return

While not the end of file
    Read line
    Split line by commas

    If less than two parts
        Print “error missing data”
        Go to next line

    Get courseNumber from first part
    Get courseName from second part

    If more parts
        Add to prerequisite list

    Check each prerequisite
        Look through all courses

If prerequisite not found

Print " error not found"

Close files

// Added a course object to store in the vector

For each of the validated course data

Create a new course object

courseNumber = first part

courseName = second pary

For the remaining parts

Add to the course prerequisite

Add course Object to course vector

Print "course has been stored"

<u>Course Object</u>

Courses = empty vector

While reading lines from the file

Split lines into two parts

Create new course object

courseNumber = first part

courseName = second part

For the remaining parts

Add to course prerequisite

Add course object to course vector

Print " course stored"

Print Course Info

// using the example

Void searchCourse(Vector<course> courses, String courseNumber) {

    For all the courses

        If the course is the same as the couseNumber

            Print " course info"

            For each prerequisit of the course

                Print " course prerequisite info"

Print all sorted courses

Void printSortedCourses(Vector <course> courses)

    Sort courses vector by courseNumber alphanumerically

    For each course in the courses

        Print courseNumber and courseName

-Hash Table-

File Input

Open File

If the file doesn't open

    Print "Error opening file"

    Return

Create empty hash table for courses

Create empty list for validation

While not the files end

    Read line

    Split line by commas

    If less than two parts

        Print " Error There is missing data"

        Go to the next line

    Get CourseNumber from first part

    Get CourseNumbver from second part

    Add CourseNumber to validation list

    If more parts exist

        For each remaining part

            Add to the prerequisite list in course

    Store course data for the hash table (temporarily)

Close file

//Now check if the prerequisites exist

For each course in the data

    For each prerequisite in course

        Look through the valid list

        If prerequisite not found in the list

            Print " error not found"

Return

// Added a course object to store in the hash table

For eachvalidated course Data

Create newCourse data

courseNumber = first part

courseName = second part

For the remaining parts

Add to the course prerequisite

Position = hash function of the courseNumber

Courses position = course object

Print " course stored"

Course Object

Courses = empty hash table

Function HashFunction for courseNumber

NumberPart = numbers from courseNumber

Return NumPart from table size

Structure Course

courseNumber = string

courseName = string

Prerequisites = empty vector

While reading valid data

Create newCourse object

CourseNumber = first part

CourseName = second part

For the remaining parts

Add to the course prerequisite

Position = Hash Function of courseNumber

courses(position) = course object

Print " Courses successfully loaded into the table"

Printing Course Info

Find the course position using the hash function

Get course from the position in the table

If course matches the courseNumber

Print the course number and name

If the course has prerequisites

Print Prerequisites:

For each prerequisite of the course

Print prerequisite course infor

Else

Print "No prerequisites required"

Else

Print “ Course couldn't be found”

Print All Sorted Courses

Void PrintSortedCourses(HashTable Courses)

Create an empty vector for sorting

For each position in the table

If position has a course

Add course to the vector

Sort the vector by courseNumber alphanumerically

For each course in the vector

Print courseNumber and the courseName

- Binary Search Tree -

File Input

Open file

If the file doesn't open

Print "error opening file"

Return

Create an empty list for validation

```
    Read line and split
    If less than two parts
        Print "error missing data"
        Go to next line

    Get courseNumber from first part
    Get courseName from the second part

    Add courseNumber to validation list
    If more parts exist
        For each remaining part
            Add to prerequisite list

Close file

// This is to check prerequisites exist as courses
For each course data
    For each prerequisite in the course
        Look through the valid list
        If the prerequisite not found
            Print "error the prerequisite not found"
            Return

// Added a course object to store in the tree
For eachvalidated course Data
            Create newCourse data
            courseNumber = first part
            courseName = second part

            For the remaining parts
                    Add to the course prerequisite
```

```
If the tree is empty
        Create root node with the course object
Else
        Start at the root node
        While it isnt inserted
                If courseNumber is > current node courseNumber
                        If the left child is empty
                                Insert the course as a left child
                        Else
                                Move to the left child
                Else
                        If the right child is empty
                                Insert course as the right child
                        Else
                                Move to the right
Print " course stored"
```

Course Object

```
binarySearchTree = empty tree
// This is the node structure for the tree
Struct Node
   course = Course object
   left = null
   right = null

// now the course structure
Struct Course
   courseNumber = string
   courseName = string
```

```
    prerequisites = empty vector

While processing course data
    Create a new course object
    courseNumber = first part
    courseName = second part

    For the remaining parts
        Add to the course prerequisite vector

    // now insert into binary search tree
    If the tree is empty
        Create root node with course object
    Else
        Start at the root node
        While it is not inserted
            If courseNumber is less than the current node courseNumber
                If the left child is empty
                    Insert course as left child
                Else
                    Move to the left child
            Else
                If the right child is empty
                    Insert course as the right child
                Else
                    Move to the right child

    Print "course stored"
```

Print Course Info

```
Void searchCourse(Tree<Course> courses, String courseNumber)
```

```
Start at the root node

While the node does exist
    If courseNumber matches the current node courseNumber
        Print out the course info

        For each prerequisite of the course
            Print the prerequisite course information
        Return

    If courseNumber is less than the current node courseNumber
        Move to the left child
    Else
        Move to the right child

Print "Course was not found"
```

Print All Sorted Courses

```
Void PrintSortedCourses(tree <course> courses)
        Start at root node
        Perform in order traversal

        If the current node isnt full
                Go to the left subTree
                Print current node courseNumber and courseName
                Go to right subTree
```

Menue

// This can work with all the data structures

Display the menu options

While the user doesnt choose exit(9)

    Print "(1) Load Data Structure"

    Print "(2) Print Course List"

    Print "(3) Print Course"

    Print "(9) Exit"

    Print "What would you like to do?"

    Get user choice

    If the choice is 1

        Print "Enter file name"

        Get he file name

        Call the load data function

        Print "Data loaded successfully"

    Else if the choice is 2

        If the data structure is empty

            Print "No data loaded"

        Else

```
        Call the print sorted courses function
    Else if the user chooses 3
        If the data structure is empty
            Print "No data loaded"
        Else
            Print "What course would you want to know about?"
            Get courseNumber
            Call the search course function with courseNumber
    Else if option 9
        Print "Thank you, Goodbye"
        Exit program


    Else
        Print choice "not a valid option"
Print "Thank you,Goodbye"
```

RunTime Analysis

Vector	Line Cost	Executions	Total
Open file	1	1	1
Read and parse the lines	1	n	n
Validate the prerequisites	1	n * n	n * n
Create and store the couse object	1	n	n
Total	4		n^2 + 2n + 1
Run Time			O(n^2)

Vector	Line Cost	Executions	Total
Open file	1	1	1
Read and parse the lines	1	n	n
Validate the prerequisites	1	n * n	n * n
Create and hash insert the objects	1	n	n
Total	4		n^2 + 2n + 1
Run Time			O(n^2)

Binary Tree	Line Cost	Executions	Total
Open file	1	1	1
Read and parse the	1	n	n

lines			
Validate the prerequisites	1	n * n	n * n
Create and insert into the tree	Log n	n	N log n
Total	3 + log n		n^2 + n log n + n + 1
Run Time			O(n^2)

After creating the Big O analysis charts, I can see that they all share performance when loading data from the validation loop. This is because they all make sure each prerequisite is checked against all the courses with an O(n^2). Given that I chose to evaluate the data structures based on the other features for sorting and searching to narrow down which would fit the requirements best. Each of the three data structures for the course system has its own strengths and weaknesses. The vector DS would be the simplest to work with, but it would be slower when searching because you would go through every course one by one(O(n)). The hash table DS is rather fast at finding specific courses (O (1)) look up time. The downside is that it won't keep the courses in order so displaying a sorted list would require copying everything to a vector for sorting, which would be O(n log n). Lastly the Binary tree DS keeps the courses in a sorted order automatically when they get inserted which can take more time initially, but saves time when printing the list O(n).

The advantages and disadvantages basically come down to what the system needs the most. The vector's advantage is offering all-around simplicity due to the built in sorting functions with a disadvantage of slow searching that gets worse as the number of courses grows. The hash

table's advantage is having the fastest lookup time for specific courses with the disadvantage being that it isn't naturally ordered, requiring extra steps to sort and display courses alphabetically. This would defeat the purpose of having a fast look up time since a sorted display is the main requirement. The Binary Tree advantage is that it stays sorted automatically and gives a O(n) time for printing the sorted list. The disadvantage of the Binary Tree would be that it can make the code base more complex and take a longer set up time.

I would recommend using the Binary Search Tree for the advising programs software. I came to this conclusion after reviewing the Big O analysis of each Data structure I worked with over the past couple of weeks. This highlighted that Binary Tree offers a O(n) time for displaying the sorted courses wich was faster than the time for vector and Hashtable. I figure that since the advisors will want to view the complete alphabetically sorted course list on a regular basis that this would be the most important feature to optimize. The Binary Tree might not have as fast of an individual course look-up time compared to the hash table, but it makes up for it in the sorting. I found that this would be the perfect balance to meet all the requirements needed from the advisors using the software.