VECTOR PSEUDOCODE

function main()

string file = ‘courses.txt’

vector<Course> courses = readFromFile(file)

class Course

String number

String title

String prerequisites

Constructor(number, title, prerequisites)

function readFromFile(file)

Create vector<Course> courses

Open file

While open file

read from file

split at space

number

title

create course object with the new attributes

append to courses vector

Return courses

PrintCourseListVector(courses)

Sort

For each course

Print(course)

HASH TABLE PSEUDOCODE

ReadFromFile(file)

If file cant open

Error

For each line

Remove whitespace and split lines

Assign contents to variables to hold data

ValdiateFile(id, name, courses)

CreateCourse(id, name, courses)

Close()

ValidateFile()

If empty

Print error

For each course in courseList

If empty

Print error

CreateCourse(id, name, courses)

Id

Name

Courses

AddCourseToTable()

Calculate index and store

If collision

Add object at hash index

PrintCourses()

For each bucket

For each course

Print details

PrintCourseListHashTable()

Create list

For each course

Add to list

Sort(list)

For each course

Print(course)

BINARY TREE PSEUDOCODE

LoadFile(file)

Open file

If not opened

Error

Create list courses

While loop through file

Read from file

Split data into different variables (courseNumber, title, prerequisite)

For each prerequisaite iteration

If prerequisaite not present

Error

Add (courseNumber, title, prerequisite)

InsertCourse(tree, course)

If null

Return node

If courseNumber < tree root course number

leftTree insertCourse

else

rightTree insertCourse

BuildTree(courses)

New binary tree

For each course iteration

newTree = insertCourse(newTree, course)

PrintTree(tree)

Print leftTree

Print rootTree

Print rightTree

PrintCourse(course)

Print(number)

Print(title)

If empty

None

Else

Print prerequisite

PrintCourseListTree(tree)

Initialize root

If null

Return

Traverse left node

Print(course)

Traverse right node

MENU PSEUDOCODE

Menu()

Print 1 -load data

Print 2 -print alphanumerically

Print 3 -print course and prerequisites

Print 9 -exit

Input = getInput

if input = 1

file = getinput

loadFile(file)

else if input = 2

printCourses

else if input = 3

course = getInput

printCourseInfo(course)

else if input = 9

exit

break

else

error

EVALUATION

Different data structures offer different advantages in different situations. As for implementation, Vector is the most straightforward for small amounts of data,. It has a O(n) runtime for loading and printing. It is less efficient with sorting as O(n log n) leaves far too much overhead. The most efficient for searching would be (O(1)) for individual queries. We still do need to consider a better option for sort time, because the Hash Table still needs O(n log n) time for the sort. After considering this, I think the binary search tree is the most suitable, primarily because of its ability to maintain a sorted order during insertion and a search with a time of (O (log n)) and its inorder printing of (O(n)). The Binary tree provides balance between performance and functionality, making it the best choice for our system.