Project One

Pseudo-Code:

Vector:

file = open(file)

courseVector = Vector()

for line in file:

courseDetails = parseCourseDetails

if validateFormat():

course = createCourseObject()

courseVector.append(course)

file.close()

function searchCourse:

for course in courseVector:

if course.code == courseCode:

printCourseInfo()

printPrerequisites()

function printCourseInfo:

print("Course Code:", course.code)

print("Course Title:", course.title)

function printPrerequisites:

if course.prerequisites:

print("Prerequisites:")

for prereqCode in course.prerequisites:

prereqCourse = findCourse()

print("- Course Code:", prereqCourse.code)

print(" Course Title:", prereqCourse.title)

function findCourse:

for course in courseVector:

if course.code == courseCode:

return course

function parseCourseDetails:

tokens = split()

courseCode = tokens[0]

courseTitle = tokens[1]

if length(tokens) > 2:

prerequisites = tokens[2:]

else:

prerequisites = null

return

function validateFormat:

if length(courseDetails) < 2:

print("Error: Insufficient parameters on a line.")

return false

if courseDetails[2]:

for prereqCode in courseDetails[2]:

if not findCourse():

print("Error: Prerequisite", prereqCode, "not found.")

return false

return true

function printCoursesInAlphanumericOrder:

mergeSort(courseVector)

for course in courseVector:

printCourseInfo(course)

Hash Table:

function load\_courses\_from\_file(file):

Open file

Initialize hash table course\_hash\_table

For each line in file:

Split the line course\_number, course\_title, prerequisites

If the number of elements < 2:

Print error message

If prerequisites not empty:

For each prerequisite:

If prerequisite does not exist in the hash table:

Print "Error: Prerequisite not found for course"

Create course object with course\_number, course\_title, and prerequisites

Calculate the hash value for course\_number

Insert the course object into hash table using hash value

Close file

Return course\_hash\_table

function print\_course\_information(course\_hash\_table):

For entry in the hash table:

Print "Course Number:", entry.key

Print "Course Title:", entry.value.course\_title

If entry.value.prerequisites not empty:

Print "Prerequisites:"

For each prerequisite in entry.value.prerequisites:

Print prerequisite

function printCoursesInAlphanumericOrder:

mergeSort(course\_hash\_table)

for course in course\_hash\_table:

printCourseInfo(course)

Tree:

function load\_courses\_from\_file(file):

Open file

Initialize binary tree course\_binary\_tree

For each line in file:

Split the line to get course\_number, course\_title, and prerequisites

If the number of elements < 2:

Print error message

If prerequisites not empty:

For each prerequisite:

If prerequisite does not exist in the binary tree:

Print "Error: Prerequisite not found for course"

Create course object with course\_number, course\_title, and prerequisites

Insert the course object into binary tree based on course\_number

Close file

Return course\_binary\_tree

function print\_course\_information(course\_binary\_tree):

In-order traversal of binary tree:

Print "Course Number:", current\_node.course\_number

Print "Course Title:", current\_node.course\_title

If current\_node.prerequisites not empty:

Print "Prerequisites:"

For each prerequisite in current\_node.prerequisites:

Print prerequisite

function printCoursesInAlphanumericOrder:

mergeSort(course\_binary\_tree)

for course in course\_binary\_tree:

printCourseInfo(course)

Menu:

function main\_menu:

display "1. Load Data Structure"

display "2. Print Course List"

display "3. Print Course"

display "4. Exit"

get user\_input

if user\_input == 1:

load\_data\_structure()

else if user\_input == 2:

print\_course\_list()

else if user\_input == 3:

print\_course()

else if user\_input == 4:

exit\_program()

else:

display "Invalid input. Please choose again."

main\_menu()

Evaluation:

One advantage of the vector data structure is that it allows a programmer to access values based on their index. One disadvantage of the vector data structure is that inserting and deleting items in the data structure is not as efficient as others.

One advantage of the hash table data structure is that it can insert and delete items very efficiently and handle the changing of the number of elements within the structure well. One disadvantage of the hash table data structure is that the efficiency may be worse if the key is not designed in a way that helps prevent collisions.

One advantage of the binary tree data structure is that it is efficient when inserting, deleting, and searching for elements. One disadvantage of the binary search tree is that is much more difficult to implement than the other data structures previously mentioned.

After analyzing all three of the data structures the recommendation, I have for which one should be used in this code is the hash table data structure. My reason for this is that it provides efficient insertion and deletion capabilities. Furthermore, it has an efficient worst-case run time compared to the vector and tree data structures.