**3-3 Milestone: Vector Data Structure Pseudocode**

openFile("file.txt") //O(1)

courseObjects = new Vector<Course>() //O(1)

for each line in file //O(N)

// Split the line into tokens

tokens = split(line, ",") //O(N)

if (tokens.length < 2) //O(N)

print("Not enough parameters on line.") //O(1)

continue

courseCode = tokens[0] //O(1)

courseName = tokens[1] //O(1)

prerequisiteCourseCode = (tokens.length > 2) ? tokens[2] : "" //O(1)

if (!prerequisiteCourseCode.isEmpty() && !courseCodeExists(courseObjects, prerequisiteCourseCode)) // O(N)

print(Prerequisite course not found in file.") //O(1)

continue //O(1)

course = new Course(courseCode, courseName, prerequisiteCourseCode) //O(1)

courseObjects.add(course) //O(1)

closeFile() //O(1)

function searchCourseAndPrint(courseObjects, targetCode)

found = false //O(1)

for each course in courseObjects //O(1)

if course.courseCode == targetCode ) //O(N)

print("Course Code: " + course.courseCode) ) //O(1)

print("Course Name: " + course.courseName) ) //O(1)

if !course.prerequisiteCourseCode.isEmpty() ) //O(N)

print("Prerequisite Course: " + course.prerequisiteCourseCode) ) //O(1)

found = true ) //O(1)

break ) //O(1)

if !found ) //O(1)

print("Course not found.") ) //O(1)

**4-3 Milestone: Hash Table Data Structure Pseudocode**

openFile("file.txt") //O(1)

courseObjects = new Hashtable<String, Course>() //O(1)

for each line in file //O(N)

// Split the line into tokens

tokens = split(line, ",") //O(N)

if (tokens.length < 2)

print("Not enough parameters on line.") //O(1)

continue //O(1)

courseCode = tokens[0] //O(1)

courseName = tokens[1] //O(1)

prerequisiteCourseCode = (tokens.length > 2) ? tokens[2] : ""//O(1)

if (!prerequisiteCourseCode.isEmpty() && !courseObjects.containsKey(prerequisiteCourseCode))

print("Prerequisite course not found in file.") //O(1)

continue

course = new Course(courseCode, courseName, prerequisiteCourseCode) //O(1)

courseObjects.put(courseCode, course) //O(1)

closeFile()//O(1)

function searchCourseAndPrint(courseObjects, targetCode) //O(1)

if courseObjects.containsKey(targetCode) //O(N)

course = courseObjects.get(targetCode) //O(1)

print("Course Code: " + course.courseCode) //O(1)

print("Course Name: " + course.courseName) //O(1)

if !course.prerequisiteCourseCode.isEmpty()//O(n)

print("Prerequisite Course: " + course.prerequisiteCourseCode) //O(1)

else//O(1)

print("Course not found.") //O(1)

**5-3 Milestone: Tree Data Structure Pseudocode**

Readfile(filename) //O(1)

Open file //O(1)

If file can not be opened//O(1)

Print Error//O(1)

Return//O(1)

Initalize a new binary tree//O(1)

While not at end of file //O(N)

Line = read next line//O(1)

Tokens = split by comma //O(N)

If length of tokens < 2

Print not enough parameters //O(1)

CourseCode = token[0] //O(1)

courseName = token[1] //O(1)

prerequisiteCourseCode = “ “ //O(1)

if prerequsitecourseCode is not empty and not in binary tree //O(N)

print not found//O(1)

course = create new course with courseCode, courseName, and prequsiteCourseCode //O(1)

close file //O(1)

insert(binarytree, course) //O(1)

if binary tree is empty //O(N)

create new node with course//O(1)

else

currentNode = binarytree //O(1)

while//O(n)

if course.courseCode < currentnode.course.courseCode //O(N)

if current node is null//O(N)

currentNode.left = new node //O(1)

else

currentNode = currentNode.left //O(1)

else

if currentNode.right is null //O(1)

curentNode.right = create new node //O(1)

else

curentNode = currentNode.righ //O(1)

searchCourseandPrint(BinaryTree, i) //O(1)

currentNode = BinaryTree,root //O(1)

while currentNode is not null //O(N)

if currentNode.course.CourseCode == I //O(1)

print course code and name //O(1)

else if//O(1)

i< currentNode.course.courseCode //O(1)

currentNode = currentNode.left //O(1)

else

currentNode = current.Node.right //O(1)

print course not found //O(1)

**Pseudocode for Menu**

mainMenu()

While

Print “1. Load data structure”

Print “2. Print course list”

Print “3. Print course”

Print “4. Exit”

Choice = get user input

if choice == 1

Load data structure

Else If choice == 2

Print course list

Else If choice == 3

Print “Enter course code”

Coursecode = get user input

printCourse(coursecode)

else if choice == 4

exit

else

print “Invalid choice”

End loop

**Pseudocode for Course list**

void printCourseInformation()

For each course in data structure

If course.title equals courseTitle

Print “Course: “ course.title

Print “Prerequisites: “ + course.prerequisites

Return

End

Print “course not found.”

**Explain the advantages and disadvantages of each structure in your evaluation.**

The choice between using a vector, a hash table, or a tree for data storage is dependent on factors like the size of the data set, the type of data, and the big O. Vectors are straightforward. They allow for random access to elements. Vectors can also grow as needed. They are efficient for checking if an item exists, like a prerequisite. However, they are inefficient for large data sets as they have a big O of linear searches which is O(N). Hash tables are very good at inserting items with a big O of O(1) and would be great for checking prerequisites as well. Hash tables would also be a good data storage for this project because the search to find specific courses based on course codes is O(1). Tree’s on the other hand I believe would be the worst because the big O can degrade if the tree is unbalanced which due to how prerequisites are set it is possible. Furthermore, a tree data structure has a lot of memory overhead since each node in a tree requires additional memory to point to its children.

**Make a recommendation for which data structure you will plan to use in your code.**

Based on the above factors, and the fact that I personally have more experience with hash tables and find them easier to implement, I will be using a hash table to store the data in this project. A hash table is efficient in handling inserting and searches with a big O of O(1). A hash table will help ensure that the searches are quick.