CS 300 Project 1

**Pseudocode for vector:**

Initialize empty vector of Course objects

Function openAndReadFile(fileName):

try:

Open file with given fileName

For each line in file:

Parse line to extract course information

Create Course object with parsed information

Add Course object to vector

catch FileError:

Print "Error reading file"

Function printCourseList():

Sort vector based on course number

For each course in vector:

Print course information

Function printCourse(courseNumber):

For each course in vector:

If course.number == courseNumber:

Print course information

Print prerequisites

Function menu():

Display options:

1. Load Data Structure

2. Print Course List

3. Print Course

4. Exit

Read user choice

Execute corresponding function

**Pseudocode for Hash Table:**

Initialize empty hash table of Course objects

Function openAndReadFile(fileName):

try:

Open file with given fileName

For each line in file:

Parse line to extract course information

Create Course object with parsed information

Add Course object to hash table with course number as key

catch FileError:

Print "Error reading file"

Function printCourseList():

Get all keys from hash table

Sort keys

For each key in sorted keys:

Print course information for hash table[key]

Function printCourse(courseNumber):

If courseNumber exists in hash table:

Print hash table[courseNumber] information

Print prerequisites

Function menu():

Display options:

1. Load Data Structure

2. Print Course List

3. Print Course

4. Exit

Read user choice

Execute corresponding function

**Pseudocode for Tree:**

Initialize empty binary search tree of Course objects

Function openAndReadFile(fileName):

try:

Open file with given fileName

For each line in file:

Parse line to extract course information

Create Course object with parsed information

Insert Course object into binary search tree based on course number

catch FileError:

Print "Error reading file"

Function printCourseList(treeNode):

If treeNode is not null:

In-order traversal to print course information

Function printCourse(treeNode, courseNumber):

If treeNode is not null:

If treeNode.course.number == courseNumber:

Print treeNode.course information

Print prerequisites

Else if treeNode.course.number < courseNumber:

Recursively search right subtree

Else:

Recursively search left subtree

Function menu():

Display options:

1. Load Data Structure

2. Print Course List

3. Print Course

4. Exit

Read user choice

Execute corresponding function

**Big O Analysis**

Vector:

Open and read file: O(n)

Create Course Objects: O(1) per line

Print Course List: O(n log n) (sorting)

Print Course: O(n)

Hash Table:

Open and read file: O(n)

Create Course Objects: O(1) per line

Print Course List: O(n log n) (sorting)

Print Course: O(1) average (O(n) worst case due to hash collisions)

Tree:

Open and read file: O(n)

Create Course Objects: O(1) per line

Print Course List: O(n) (in order of traversal)

Print Course: O(log n) average (O(n) worst case for unbalanced tree)

**Evaluation**

Based on the Big O analysis vectors are good for sequential access and maintains insertion order but has higher time complexity for searching. Hash tables can be advantageous to look things up quickly on average, but have potential worst-case scenarios with hash collisions. Trees can be used for efficient lookups, especially if the tree remains balanced.

Considering the requirements, both vector and hash tables require O(n log n) time for sorting courses. Hash tables have an advantage with O(1) time for lookups to search for a course. Hash tables may also consume more memory due to hash table collisions.

Hash Table seems to be the best fit due to its average O(1) time complexity for lookups, which aligns well with the program's requirements. The potential for hash collisions can be mitigated with a good hash function and proper resizing strategies.