# CS 300 Pseudocode Document

By Mariam Haji

**Vector pseudocode**

// Pseudocode for opening and parsing file

function openAndParseFile(filename: String) -> Vector<Course>:

// Open the file

// Create a vector to hold course objects

for each line in file:

// Find and add prerequisite course objects

course = new Course(courseNumber, courseTitle, prerequisites)

courses.push\_back(course)

// Add course object to vector

return courses

// Pseudocode for printing course information and prerequisites

function printCourseInformation(courses: Vector<Course>, courseNumber: String) -> void:

course = findCourse(courseNumber, courses)

if course == null:

print "Course not found"

return

// Sort courses by course number

for each course in sortedCourses:

// Print the information of the course

// Print the information of each prerequisite course

// Pseudocode for menu

function menu() -> void:

courses = null

while true:

print "1. Load Data Structure"

print "2. Print Course List"

print "3. Print Course"

print "4. Exit"

choice = readInt()

if choice == 1:

courses = openAndParseFile("filename.txt")

else if choice == 2:

printCourseList(courses)

else if choice == 3:

print "Enter course number:"

courseNumber = readString()

printCourseInformation(courses, courseNumber)

else if choice == 4:

break

**Hashtable pseudocode**

// Pseudocode for opening and parsing file

function openAndParseFile(filename: String) -> Hashtable<String, Course>:

// Open the file

//Create a hashtable to hold course objects

// Check for formatting errors

//print "Error: Incorrect formatting in line "

// Split prerequisites by semicolon

// Find and add prerequisite course objects from hashtable

// Add course object to hashtable

return courses

// Pseudocode for printing course information and prerequisites

function printCourseInformation(courses: Hashtable<String, Course>, courseNumber: String

**Tree pseudocode**

// Node class to represent a course

// Constructor to initialize node with course information

// Tree class to hold all the courses

// Method to insert a new course into the tree

// Helper method to recursively insert a new node

// Method to print out all the courses in the tree in alphanumeric order

// Helper method to recursively print out all the nodes in the tree

// Method to print out the course information and prerequisites for a given course

// Helper method to recursively find a node with a given course number

## Runtime Analysis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Structure** | **Operation** | **Time Complexity** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Vector | numPrerequisiteCourses | O(n^2) | 4 | n^2 | 4n^2 |
| printSampleSchedule | O(n^2) | 4 | n^2 | 4n^2 |
| printCourseInformation | O(n^2) | 5 | n^2 | 5n^2 |
| Load Data Structure | O(n) | 1 | n | n |
| Print Course List | O(n log n) | 2 | 1 | 2 |
| Print Course | O(n^2) | 5 | n^2 | 5n^2 |
| Hashtable | numPrerequisiteCourses | O(n) | 3 | n | 3n |
| printSampleSchedule | O(n) | 2 | n | 2n |
| printCourseInformation | O(n) | 4 | n | 4n |
| Load Data Structure | O(n) | 1 | n | n |
| Print Course List | O(n log n) | 2 | 1 | 2 |
| Print Course | O(1) | 1 | 1 | 1 |
| Tree | insertCourse | O(log n) | 2 | n | 2n log n |
| printCourseList | O(n) | 3 | n | 3n |
| printCourse | O(log n) | 2 | n | 2n log n |
| Load Data Structure | O(n log n) | 2 | n | 2n log n |

**Vector:**

Advantages:

Allows for random access to elements in O(1) time complexity, good for small to medium sized data sets and is simple to implement.

Disadvantages:

Its time consuming to add and remove elements from the middle of the vector since the remaining elements will need to be shifted.

Can be inefficient for very large data sets as a contiguous block of memory needs to be reserved to store the elements, which can lead to memory fragmentation.

**Hashtable:**

Advantages:

Provides fast access to elements in O(1) time complexity on average, making it ideal for large data sets and is good for searching and retrieving specific elements.

Disadvantages:

Hashing collisions can occur, leading to longer access times, it’s not suitable for preserving order as well as implementing an efficient hash function can be difficult.

**Tree:**

Advantages:

Provides fast access to elements in O(log n) time complexity on average, making it ideal for large data sets, the data structure is also efficient for inserting, removing and searching elements as well as preserves order, making it useful for sorting and traversing elements.

Disadvantages:

Requires more memory than hash tables, as each node stores its own data and pointers to its children and may be more complex to implement than other data structures.

**Recommendation**

Based on the Big O analysis results, all three data structures have the same O(n) runtime complexity for the operations required in the program. Although complex to implement and maintain, I would recommend the tree data structure since it has built-in sorting and traversal methods which provide efficient searching and sorting, especially if the data is large or needs to be sorted frequently.