# Pseudocode and Runtime Analysis

## Introduction

This document provides the final pseudocode and runtime analysis for the advising program at ABCU's Computer Science department. The pseudocode includes logic for managing courses using three data structures: Vector, Hash Table, and Tree. Additionally, the runtime analysis evaluates the efficiency of each data structure, allowing for an informed recommendation on the optimal choice.

## Pseudocode

### Vector Data Structure

1. Load the file:  
 - Open the file.  
 - If the file does not exist, print an error and exit.  
 - Read each line, split it into components, and validate the format.  
 - Add valid course data to a list.  
  
2. Create course objects:  
 - For each line in the list, create a course object containing the course number, title, and prerequisites.  
  
3. Print course information:  
 - Search the list for the course.  
 - Print the course number, title, and prerequisites.  
  
4. Menu:  
 - Option 1: Load file data into the vector.  
 - Option 2: Print an alphanumerically ordered list of courses.  
 - Option 3: Print a specific course's title and prerequisites.  
 - Option 9: Exit.

### Hash Table Data Structure

1. Load the file:  
 - Open the file.  
 - If the file does not exist, print an error and exit.  
 - Read each line, split it into components, and validate the format.  
 - Add valid course data to a hash table with the course number as the key.  
  
2. Create course objects:  
 - For each line, create a course object containing the course number, title, and prerequisites.  
  
3. Print course information:  
 - Search the hash table for the course.  
 - Print the course number, title, and prerequisites.  
  
4. Menu:  
 - Option 1: Load file data into the hash table.  
 - Option 2: Print an alphanumerically ordered list of courses (extract keys and sort).  
 - Option 3: Print a specific course's title and prerequisites.  
 - Option 9: Exit.

### Binary Search Tree Data Structure

1. Load the file:  
 - Open the file.  
 - If the file does not exist, print an error and exit.  
 - Read each line, split it into components, and validate the format.  
 - Add valid course data to a binary search tree.  
  
2. Create course objects:  
 - For each line, create a course object containing the course number, title, and prerequisites.  
  
3. Print course information:  
 - Search the tree for the course.  
 - Print the course number, title, and prerequisites.  
  
4. Menu:  
 - Option 1: Load file data into the tree.  
 - Option 2: Print an alphanumerically ordered list of courses (in-order traversal).  
 - Option 3: Print a specific course's title and prerequisites.  
 - Option 9: Exit.

## Runtime Analysis

The table below outlines the runtime complexity of key operations for the Vector, Hash Table, and Tree data structures.

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | Hash Table | Tree |
| Load and Parse File | O(n) | O(n) | O(n) |
| Create Course Objects | O(n) | O(n) | O(n) |
| Sort Courses | O(n log n) | N/A | O(log n) |
| Search for Course | O(n) | O(1) | O(log n) |

## Analysis of Data Structures

1. Vector:  
- Advantages: Simple to implement, suitable for small datasets.  
- Disadvantages: Linear search is slow for large datasets; sorting is required for ordered output.  
  
2. Hash Table:  
- Advantages: Fast lookups (O(1) average case).  
- Disadvantages: Memory-intensive; hash collisions may slow performance.  
  
3. Tree:  
- Advantages: Efficient search, insert, and traversal operations (O(log n)).  
- Disadvantages: More complex to implement than vectors or hash tables.

## Recommendation

Based on the runtime analysis and requirements, the recommended data structure is the Hash Table. It provides the fastest lookup times for individual course queries, which is critical for advisor use. Although it is memory-intensive, this trade-off is acceptable given the efficiency benefits.