**Vector Sorting Pseudocode:**

# Bidding System Pseudocode

# This program helps you manage and sort bids.

# Start the program

Load bids from a file into the system

# We begin by reading a list of bids from a data file.

Display a menu for the user

# We present you with a menu of options to choose from.

Let the user choose an option

# You can select what you want to do by entering a number.

If the user chooses to see the bids:

# If you pick this option, we will display all the bids you loaded.

Show the list of bids

If the user chooses to sort by title using selection sort:

# If you want to sort the bids alphabetically by their titles,

# we'll do that for you using a method called "selection sort."

Sort the bids by title alphabetically

If the user chooses to sort by title using quicksort:

# If you prefer a faster way to sort the bids by title,

# we'll use a method called "quicksort" to do it more efficiently.

Sort the bids by title more efficiently

End the program

# That's it! You can explore your bids, sort them, and then we say goodbye.

**Hash Tables Pseudocode:**

Loading Bids into the HashTable

Function LoadBidsFromFile(fileName):

Open the file with fileName for reading

If the file doesn't exist:

Print an error message

Return an empty hash table

Create an empty hash table called bidTable

For each line in the file:

Trim extra spaces at the beginning and end of the line

Extract bidId, title, fund, and amount from the line

If any required parameter is missing:

Print an error message

Continue to the next line

Create a new bid with bidId, title, fund, and amount

Insert the bid into the bidTable

Close the file

Return bidTable

Creating and Inserting Bid Objects into the HashTable

Function CreateAndInsertBid(bidId, title, fund, amount, bidTable):

Create a new bid object with bidId, title, fund, and amount

Insert the bid into the bidTable

Return

Searching and Printing Bid Information

Function SearchAndPrintBidInfo(bidTable, targetBidId):

For each bid in bidTable:

If the bid's ID matches targetBidId:

Print "Bid ID: ", bid's ID

Print "Title: ", bid's title

Print "Fund: ", bid's fund

Print "Amount: ", bid's amount

Return

Print "Error: Bid not found"

Return

**Tree Pseudocode:**

class Course:

attributes: course\_number, course\_title, prerequisites

function parse\_course\_info(line):

tokens = split\_line(line)

course\_number = tokens[0]

course\_title = tokens[1]

prerequisites = tokens[2:] if length(tokens) > 2 else []

return course\_number, course\_title, prerequisites

function load\_courses(file\_name):

root = None

file = open\_file(file\_name)

for line in file:

course\_number, course\_title, prerequisites = parse\_course\_info(line)

new\_course = Course(course\_number, course\_title, prerequisites)

root = insert\_course(root, new\_course)

close\_file(file)

return root

function insert\_course(root, new\_course):

if root is null:

return new\_course

else if new\_course.course\_number < root.course\_number:

root.left = insert\_course(root.left, new\_course)

else:

root.right = insert\_course(root.right, new\_course)

return root

function print\_course\_info(node):

if node is not null:

print\_course\_info(node.left)

print(node.course\_number, node.course\_title, node.prerequisites)

print\_course\_info(node.right)

**Menu Pseudocode:**

# ABCU's Course Information System

# This program manages and displays detailed information about Computer Science courses.

# Start the program

Load course data from a file into the system

# The system reads and organizes course details from a data file.

# Display a menu for the user

# Users are presented with options for managing and exploring course information.

Display menu options:

1. Show Alphabetical Course List

# Display an alphabetically sorted list of all Computer Science courses.

2. Show Course Details

# View detailed information for a specific course.

3. Exit

# Let the user choose an option

# Users can select tasks by entering a corresponding number.

Get user input for choice

# If the user chooses to display the course list

If choice == 1:

# Show all Computer Science courses in alphabetical order.

ShowAlphabeticalCourseList(dataStructure)

# If the user chooses to show course details

Else If choice == 2:

# Prompt the user to input a course code.

Get user input for course code

# Show detailed information for the specified course.

ShowCourseDetails(courseCode, dataStructure)

# If the user chooses to exit the program

Else If choice == 3:

# End the program gracefully.

ExitProgram()

# If the user enters an invalid choice

Else:

# Display an error message for invalid input.

Display "Invalid choice. Please select a valid option."

# Repeat the menu loop until the user chooses to exit.

Repeat until user selects to ExitProgram

**Runtime Analysis:**

The pseudocode loops through each input file line during the file reading operation. The temporal complexity of reading the file is O(n), assuming that it has 'n' lines. O(n) is the time complexity for constructing course objects since the loop that parses each line to build course objects iterates 'n' times. Since fundamental operations are performed on each line processed in this loop, the cost per line is 1. As a result, O(n) may be used to represent the entire time complexity for reading files and creating course objects.

Inserting 'n' course objects would have the following runtime difficulties given the data structures involved (vector, hash table, and tree): Vector's insertion time is O(n), as it could need to resize the underlying array. When taking into account potential collisions and resizing, the average insertion time for a hash table is O(n). In terms of tree topologies, during insertion, it exhibits O(n log n) for balanced trees. It might, however, degenerate to O(n^2) for insertion if it is imbalanced. This research helps to evaluate how well various data structures work in handling course data by illuminating the operational efficiency of each structure.