

# **CS-300 Project One Final Part**

# **Zachariah Spencer**

## **Menu Pseudocode**

Open input file

WHILE not end of file

Read line from file

Create new Course object with data from line

Insert Course object into data structure

END WHILE

WHILE user selection not "4"

DISPLAY menu options:

*1. Load data structure*

*2. Print course list*

*3. Print course details*

*4. Exit program*

Get user choice

IF user chooses "Load data structure" THEN

Load data from file into chosen data structure

ELSE IF user chooses "Print course list" THEN

Retrieve list of courses from data structure

Sort courses alphanumerically

Print sorted list of courses

ELSE IF user chooses "Print course details" THEN

Prompt user for course number

Retrieve course object using course number

Print course title and prerequisites

ELSE IF user chooses "Exit program" THEN

Terminate program

END IF

END WHILE

**Vector Alphanumeric Printing Pseudocode**

// Sort the courses before printing

FOR i FROM 0 TO (size of courses - 2)

minIndex = i

FOR j FROM (i + 1) TO (size of courses - 1)

IF courses[j].courseNumber < courses[minIndex].courseNumber THEN

minIndex = j

END FOR

Swap courses[i] with courses[minIndex]

END FOR

// Print all the courses

FOR each course IN courses

Print course.courseNumber, course.courseTitle

END FOR

**Hashtable Alphanumeric Printing Pseudocode**// Hashtables cannot be sorted alphanumerically

// Extract the data from the hashtable into a vector and sort, then print

Initialize empty Vector SortedCourses

FOR each key-value pair in Courses DO

Insert value (course information) into SortedCourses

END FOR

Sort SortedCourses based on alphanumeric course number // Algorithm documented in prior section

FOR each Course IN SortedCourses DO

Print Course

END FOR

**Tree Alphanumeric Printing Pseudocode**

// This will be a PrintTreeCourses method that calls itself recursively

// Since my design involves a Binary Search Tree (BST), it can be assumed that it is already ordered

IF Node is not null THEN

PrintTreeCourses(Node.left) // Traverse left subtree

Print Node.value // Print the course at the current node

PrintTreeCourses(Node.right) // Traverse right subtree

END IF

**Evaluation**

*where n is the number of lines in the file, m is the average number of characters per line, and k is the average number of tokens per line.*

Vector: The most computationally expensive step is the token validation, where for each token, the program checks if there’s a corresponding line in the file. This results in an overall time complexity of .

Hashtable: The most time-consuming operations involve reading and splitting the file, as well as processing and validating tokens. Since reading/splitting the file and processing tokens both involve and respectively, the dominant term is . Thus, the overall time complexity is ,

Tree: The most time-consuming parts of the pseudocode are likely the reading/splitting of the file and the insertion into the binary search tree. The overall complexity will depend on how well the BST is balanced. If we assume the worst-case scenario for the BST (unbalanced), the insertion could take up to for each of the n courses, giving a total complexity of . Thus, considering all factors, the overall time complexity can be expressed as . Finalizing the analysis, the overall complexity is predominantly determined by the highest-order terms, leading us to conclude that in the worst-case scenario, the complexity is .

Advantages of Vectors:

* Vectors excel in sequential data access.
* They provide ample memory storage.

Disadvantages of Vectors:

* Inserting or deleting elements in the middle is costly, with a time complexity of \(O(n)\).
* Searching for elements is slower compared to hash tables or trees.

Advantages of Hash Tables:

* They offer fast lookup times.
* Ideal for quick data retrieval.

Disadvantages of Hash Tables:

* Their complex structure requires significant memory storage.

Advantages of Trees:

* Trees are efficient for operations like insertion and deletion.
* They are well-suited for sorting data.

Disadvantages of Trees:

* Their structure leads to higher memory usage.

**Recommendation**

Given the need for both order and efficient insertions and deletions, I would recommend a balanced binary search tree (BST) for this use case.