# CS 300 Pseudocode Document

**Victor Udeh**

**M 6-2**

**Vector Pseudocode**

Procedure LoadCoursesFromFileToVector(filename)

Initialize vector courses

Open filename for reading

While not end of file

Read line and create Course object

Add Course to vector courses

EndWhile

Sort courses in vector by course number

Close file

EndProcedure

Procedure PrintAllCoursesFromVector(courses)

For each course in courses

Print course number and title

EndFor

EndProcedure

Procedure PrintCourseFromVector(courses, courseNumber)

For each course in courses

If course's number matches courseNumber

Print course's title and prerequisites

EndIf

EndFor

EndProcedure

**Hash Table Pseudocode**

Procedure LoadCoursesFromFileToHashTable(filename)

Initialize hash table coursesHashTable

Open filename for reading

While not end of file

Read line and create Course object

Insert Course into coursesHashTable with course number as key

EndWhile

Close file

EndProcedure

Procedure PrintAllCoursesFromHashTable(coursesHashTable)

Initialize list courseNumbers from coursesHashTable keys

Sort courseNumbers

For each number in courseNumbers

Access course in coursesHashTable using number

Print course's number and title

EndFor

EndProcedure

Procedure PrintCourseFromHashTable(coursesHashTable, courseNumber)

If coursesHashTable contains courseNumber

Access course using courseNumber

Print course's title and prerequisites

EndIf

EndProcedure

**Tree Pseudocode**

Procedure LoadCoursesFromFileToTree(filename)

Initialize binary search tree coursesTree

Open filename for reading

While not end of file

Read line and create Course object

Insert Course into coursesTree sorted by course number

EndWhile

Close file

EndProcedure

Procedure PrintAllCoursesFromTree(node)

If node is not null

PrintAllCoursesFromTree(node.left)

Print node's course number and title

PrintAllCoursesFromTree(node.right)

EndIf

EndProcedure

Procedure PrintCourseFromTree(node, courseNumber)

If node is null

Return

If node's course number equals courseNumber

Print node's title and prerequisites

ElseIf courseNumber is less than node's course number

PrintCourseFromTree(node.left, courseNumber)

Else

PrintCourseFromTree(node.right, courseNumber)

EndIf

EndProcedure

**Menu System Pseudocode**

Procedure MainMenu()

While true

Display options: Load Data Structure, Print Course List, Print Course, Exit

Read user's choice

Select user's choice

Case Load Data Structure

Call corresponding load procedure

Case Print Course List

Call corresponding print all courses procedure

Case Print Course

Prompt for course number

Call corresponding print course procedure

Case Exit

Exit program

EndSelect

EndWhile

EndProcedure

**POINTS**

**1. Evaluate the run-time and memory of data structures:**

* + Vector: Linear searches O(n), sorting O(n log n), and adding an element O(1) or O(n) if resizing is needed. Memory use is efficient but requires reallocating and copying when resizing.
  + Hash Table: Insertion, deletion, and access are O(1) on average, but worst-case O(n) due to collisions. Memory overhead is higher due to the hash table structure.
  + Tree (BST): Insertion, deletion, and searching are O(log n) in balanced trees but can degrade to O(n) if unbalanced. Memory usage is proportional to the number of nodes, with additional overhead for pointers.

1. **Analysis of each data structure:**
   * Vector: Simple and efficient for ordered data but suffers from slow search times and potential overhead from resizing.
   * Hash Table: Excellent for quick lookups but may have inefficiencies in memory use and collision handling.
   * Tree: Provides a balance with ordered storage and efficient lookups, but performance depends on tree balance.
2. **Recommendation:**  
   Given the requirements for alphanumeric ordering and efficient access to course details, a balanced tree structure like an AVL or Red-Black tree is recommended. These structures maintain order and ensure O(log n) access times consistently, offering a balanced compromise between the efficiency of hash tables for access and the ordered nature of vectors, with better worst-case performance guarantees.

**Advantages and Disadvantages**

* Vector:
  + Advantages: Direct access to elements, efficient iteration, and dynamic resizing.
  + Disadvantages: Slow searches for unsorted data, potentially costly resizing operations, and inefficient insertion/deletion except at the end.
* Hash Table:
  + Advantages: Fast access, insertion, and deletion operations on average; efficient for lookups.
  + Disadvantages: Memory overhead, performance degradation in case of many collisions, and unordered data storage.
* Tree (BST):
  + Advantages: Ordered data structure, providing efficient searches, insertions, and deletions when balanced.
  + Disadvantages: Performance can degrade to linear in unbalanced trees, and requires more memory for node pointers.