# CS 300 Pseudocode Document

**//Vector - Milestone 1**

|  |
| --- |
| loadCourses(filename): open file, parse each line into Course object, add to vector  validatePrerequisites(): for each course/prereq, check if prereq exists in vector, print warning if not  printAllCourses(): sort vector by courseNumber, print each courseNumber + name  searchCourse(courseNumber): for each course, if match found, print course info + prerequisites  displayMenu(): loop → options:  1 → loadCourses + validatePrerequisites  2 → printAllCourses  3 → searchCourse  9 → exit |

**//Hash Table - Milestone 2**

|  |
| --- |
| loadCourses(filename): open file, parse each line into Course object, add to temporary list  buildHashTable(): for each course/prereq in list, check if prereq exists, print warning if not, insert course into hash table (key = courseNumber)  printAllCourses(): gather all courses from hash table, sort by courseNumber, print each courseNumber + name  searchCourse(courseNumber): search hash table, if found print course info + prerequisites  displayMenu(): loop → options:  1 → loadCourses + buildHashTable  2 → printAllCourses  3 → searchCourse  9 → exit |

**//Binary Search Tree – Milestone 3**

|  |
| --- |
| loadCourses(filename): open file, parse each line into Course object, add to temporary list  buildCourseTree(): for each course/prereq in list, check if prereq exists, print warning if not, insert course into BST (key = courseNumber)  printAllCourses(): perform in-order traversal of BST, print each courseNumber + name  searchCourse(courseNumber): search BST, if found print course info + prerequisites  displayMenu(): loop → options:  1 → loadCourses + buildCourseTree  2 → printAllCourses  3 → searchCourse  9 → exit |

**Runtime Analysis Chart**

The Runtime Analysis Chart shows the time complexity of reading the file and creating course objects for each data structure. For both the Vector and Hash Table, this process is O(n), as each of the *n* courses must be read and inserted. Individual insertions are O(1) in the Hash Table, but the process remains linear. For the Binary Search Tree (BST), inserting *n* courses takes O(n log n), since each insertion is O(log n). This analysis focuses only on the file reading and object creation, not menu or search/print operations.

| **Data Structure** | **Line Cost** | **# Times Executes** | **Total Cost** | **Big O** |
| --- | --- | --- | --- | --- |
| **Vector** | 1 | n | n | O(n) |
| **Hash Table** | 1 | n | n | O(n) |
| **Binary Search Tree** | 1 | n\*log n | n log n | O(n log n) |

**Advantages and Disadvantages of Each Data Structure**

The Vector data structure is simple to implement. It allows fast iteration and easy sorting using built-in functions. It offers straightforward data storage and enables efficient list printing. However, searching for a specific course requires O(n) time unless the vector is sorted and a binary search is used. In addition, vectors may not scale as well for very large datasets due to their linear search time when unsorted.

Hash Tables provide fast lookup, insertion, and deletion with average-case O(1) performance, making them highly efficient for accessing individual courses. However, they do not store data in sorted order, so additional steps are required to print courses alphanumerically. Hash Tables are ideal when quick lookups are the primary requirement but are less suited for applications that rely on ordered traversal.

Binary Search Trees (BST) naturally maintain data in sorted order, allowing the list to be printed efficiently through in-order traversal. BSTs also provide good search, insertion, and deletion performance, with O(log n) efficiency when balanced. They are well-suited for supporting dynamic updates if new courses are added or modified over time. However, care must be taken to ensure the tree remains balanced to avoid degraded performance. Given the project’s requirement to frequently display an alphanumeric list of courses, the BST offers strong advantages. It provides the best overall balance of functionality and performance, supporting efficient ordered printing and dynamic data updates.