**Run-Time and Memory Analysis of Data Structures**

**1. Overview of Data Structures Used**

The pseudocode primarily uses the following data structures:

* **Vector**: A dynamic array (std::vector in C++) to store Course objects.
* **Strings**: To represent course numbers, titles, and prerequisites.
* **Struct**: A lightweight structure to organize course data into objects.

**2. Run-Time Analysis**

The pseudocode's key operations and their time complexities are as follows:

**File Processing**

* **Opening the File**:
  + Checking file existence and opening the file are constant-time operations: O(1).
* **Reading Lines**:
  + Reading each line from the file takes O(n), where n is the number of lines (courses) in the file.

**Data Parsing and Validation**

* **Splitting Lines into Tokens**:
  + Splitting a string (using commas) takes O(k), where K is the number of characters in the line.
* **Validating Prerequisites**:
  + For each course, the program validates the prerequisites by iterating through the prerequisites vector. This takes O(m)O(m)O(m), where mmm is the number of prerequisites for a given course.
  + Overall, this operation contributes O(n⋅m) complexity, where n is the number of courses and mmm is the average number of prerequisites.

**Storing Data in a Vector**

* **Creating and Storing Course Objects**:
  + Creating a Course object takes O(1) for memory allocation and assignment.
  + Adding the object to the vector (amortized) has O(1) time complexity.

**Searching for a Course**

* **Linear Search**:
  + The search iterates through the courseList vector to find a specific course. This is a linear operation with O(n), where n is the number of courses.

**Overall Run-Time Complexity**

The pseudocode has the following overall time complexity:

* **File Processing**: O(n)
* **Data Parsing and Validation**: O(n⋅m)
* **Inserting into Vector**: O(n)
* **Searching for a Course**: O(n)O(n)O(n)

**Total Complexity**: O(n⋅m) (dominant term due to validation).

**3. Memory Usage Analysis**

The memory usage depends on the following components:

**Course Struct**

Each Course object includes:

* courseNumber (string): O(s1), where s1 is the length of the course number.
* courseTitle (string): O(s2), where s2 is the length of the course title.
* prerequisites (vector of strings): O(m⋅s3), where mmm is the number of prerequisites and s3 is the average string length of each prerequisite.

**Memory for a Single Course**: O(s1+s2+m⋅s3)

**Vector Data Structure**

The vector used to store n courses dynamically allocates memory:

* Each Course object requires space: O(s1+s2+m⋅s3).
* The vector's overhead includes a pointer to the allocated array, size, and capacity fields.

**Total Memory for the Vector**:  
O(n⋅(s1+s2+m⋅s3)), where:

* n = total number of courses
* s1,s2 = string lengths of course number and title
* m = average number of prerequisites per course
* s3 = average string length of prerequisites

**4. Summary Table**

| **Operation** | **Run-Time Complexity** | **Memory Complexity** |
| --- | --- | --- |
| File opening | O(1) | O(1) |
| File reading (n lines) | O(n) | O(1) |
| Splitting strings (k characters) | O(k) | O(k) |
| Validating prerequisites | O(n⋅m) | O(1) |
| Creating Course objects | O(1) | O(s1+s2+m⋅s3) |
| Inserting into vector | O(1) (amortized) | O(n) |
| Searching for a course | O(n)O(n)O(n) | O(1) |
| **Total (dominant term)** | O(n⋅m) | O(n⋅(s1+s2+m⋅s3) |

**5. Optimizations to Consider**

1. **Use a Hash Map for Faster Searching**:
   * Instead of a linear search (O(n)), you can use a hash map (unordered\_map in C++) to store courses by their course number.
   * This reduces search time to O(1), improving efficiency.
2. **Reduce Memory Overhead**:
   * Use references or pointers for strings in the prerequisites vector to avoid redundant memory allocations.

**Conclusion**

The current implementation has an overall run-time complexity of O(n⋅m) and memory complexity of O(n⋅(s1+s2+m⋅s3)). While the vector structure works well for small datasets, using a hash map can optimize search operations for larger datasets.