Run-time and Memory Analysis for Data Structures

**Pseudocode for a Menu:**

**Function** Menu()

{

**WHILE** true

{

**Print** "Select an option:"

**Print** "1: Load Data Structure"

**Print** "2: Print Course List"

**Print** "3: Print Course"

**Print** "4: Exit"

**Initialize** variable choice

**Take** user input for choice

**SWITCH** choice

{

CASE 1:

**Call** appropriate loading function (e.g. Store() for vector)

**BREAK**

CASE 2:

**Call** Print() to print the course list

**BREAK**

CASE 3:

**Call** Search() to print a specific course

**BREAK**

CASE 4:

**Exit** the program

**BREAK**

DEFAULT:

**Print** "Invalid option, please try again."

**BREAK**

}

}

}

**Main** Function

{

**Call** Menu() function

}

**Pseudocode to Print Courses in Alphanumeric Order:**

**Function** SortAndPrintCourses()

{

**Sort** courses vector based on courseNumber (implement or use sorting function)

**Call** Print () function

}

Using the given pseudocodes, we have looked at three different data structures: the Vector, the Hash Table, and the Tree. All of these are used to manage course information. Each of these structures has its own pros and cons when it comes to performance, how well it uses memory, and how hard it is to apply.

**Vector:** A vectoris a dynamic list that keeps allocating memory. It's easy to use, but some tasks may not work as well as they could.

**Hash table**: A structure that links keys to values. This makes it easy to look up information by key, but it might use more memory.

**Tree**: A hierarchical data structure that sorts data in a way that makes it easy to find, add, and remove data.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Vector** | **Hash Table** | **Tree** |
| Load Data | O(n log n) | O(n) | O(n log n) |
| Search Course | O(n) | O(1) avg | O(log n) avg |
| Print Courses | O(n) | O(n) | O(n) |
| Insert course | O(n) | O(1) avg | O(log n) avg |
| Delete Course | O(n) | O(1) avg | O(log n) avg |
| Memory Efficiency | **Moderate** | **Low** | **High** |

**Evaluation:**

* **Vector**:
  + Pros: Simple implementation; constant time access.
  + Cons: Slower search, insert, and delete operations.
* **Hash Table**:
  + Pros: Fast average-case time complexity for search, insert, and delete.
  + Cons: Can be inefficient in terms of memory; performance depends on the quality of the hash function.
* **Tree (Balanced)**:
  + Pros: Logarithmic time complexity for search, insert, and delete; efficient memory usage.
  + Cons: More complex to implement.

**Analysis:**

**For Large Data Sets:**

Hash tables may give the best search, insert, and delete speed in the average case.

Trees offer a good mix between speed and the amount of memory they use.

Vectors may not be as good for big data sets, especially for operations like adding and removing data.

**For Small Data Sets:**

The changes may be less important, and the fact that vectors are easy to understand may be a plus.

**Recommendation:**

**If speed is important**, use hash tables, but be aware of how they might waste room and what could go wrong.

**If memory efficiency is really important**, you might want to use a balanced tree.

**if simplicity is important**, Vectors could be a good choice for small to medium-sized data sets