**1. Pseudocode for Menu System**

1. Display menu options:
   * Option 1: Load course data.
   * Option 2: Print all courses in alphanumeric order.
   * Option 3: Search and print a specific course’s details.
   * Option 9: Exit the program.
2. Prompt user for an option.
3. If Option 1:
   * Load data into the selected data structure.
4. If Option 2:
   * Sort courses (Vector: sort(), Hash Table: extract and sort, BST: inOrderTraversal()).
   * Print sorted courses.
5. If Option 3:
   * Prompt user for a course number.
   * Search and print course details.
6. If Option 9:
   * Exit program.

**3. Pseudocode for Sorting and Printing Courses**

**Vector Implementation**

1. Sort the vector by courseNumber using a sorting algorithm.
2. Iterate through the vector and print each course's details.

**Hash Table Implementation**

1. Extract all courses from the hash table into a temporary vector.
2. Sort the vector by courseNumber.
3. Iterate through the sorted vector and print each course's details.

**Binary Search Tree (BST) Implementation**

1. Perform an in-order traversal of the BST.
2. Print each course’s details as they are visited.

**4. Runtime and Memory Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Structure** | **Insertion Complexity** | **Search Complexity** | **Memory Usage** |
| Vector | O(n) (unsorted) / O(log n) (sorted) | O(n) / O(log n) | Low |
| Hash Table | O(1) (average) / O(n) (worst) | O(1) (average) / O(n) (worst) | Moderate |
| BST | O(log n) (balanced) / O(n) (unbalanced) | O(log n) / O(n) | High |

**5. Evaluation of Data Structures**

**Vector**

* **Advantages**: Simple, efficient for small datasets.
* **Disadvantages**: Slow insertions/deletions.

**Hash Table**

* **Advantages**: Fast lookups, good for large datasets.
* **Disadvantages**: High memory usage, poor worst-case performance.

**Binary Search Tree (BST)**

* **Advantages**: Efficient searching, naturally sorted.
* **Disadvantages**: Poor performance if unbalanced.

**6. Recommendation**

Based on the requirements, a Binary Search Tree is recommended for this application due to its balanced performance in searching and sorting. If fast lookups are prioritized over sorting, a Hash Table might be a better alternative.

**7. Conclusion**

The presented pseudocode and complexity analysis provide a well-structured approach to handling course data efficiently using different data structures. Based on the analysis conducted above, we can see that the Binary Search Tree has the best overall case of searching as it is naturally sorted, thus making the usual case O(log n). Therefore, the choice of using a Binary Search Tree ensures that the most optimal searching and sorting algorithms are used, fulfilling ABCU’s requirements.