Display menu options:

Print “Menu:”

Print “1: Load file data”

Print “2: Computer Science Department alphanumerically”

Print “3: Course Title and Prerequisites for individual course”

Print “9: Exit”

Input user choice

If user choice = 1

Call function to load file data

Print “Data loaded successfully”

Else if user choice = 2

Verify data is loaded

If data is loaded

Call function to sort and print courses

Else

Print “Error: no data loaded”

Else if user choice = 3

Verify data is loaded

If data is loaded

Input course identifier from user

Call function to search for and print course title and prerequisites

Else

Print “Error: no data loaded”

Else if user choice = 9

Exit program

Else

Print “Error: Invalid option, please select a valid option”

**Sorting in alphanumeric order:**

**VECTOR:**

Sort(vector\_courses)

For each course in vector\_courses:

Print(course)

**HASH TABLE:**

Create list\_courses from hash\_table\_courses

Sort(list\_courses)

For each course in list\_courses:

Print(course)

**BINARY SEARCH TREE:**

InOrderTraversal(tree\_courses)

For each course in traversal\_result:

Print(course)

**RUN TIME ANAYLSIS**

**VECTOR:**

Loading Data: O(1)

Search: O(N)

Sort/Print: O(N log N) – assuming quick sort is used

**HASH TABLE:**

Loading Data: O(1)–O(N) – depending on collisions

Search: O(1)–O(N) – depending on collisions

Sort/Print: O(N) – assuming the table is created in order

**BINARY SEARCH TREE**

Loading Data: O(log N)

Search: O(log N)-O(N) – depending on tree balance

Sort/Print: O(N) – using in order traversal

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vector | Hash Table | Binary Search Tree |
| Loading Data | O(1) | O(1)–O(N) | O(log N) |
| Search | O(N) | O(1)–O(N) | O(log N)-O(N) |
| Sort/Print | O(N log N) | O(N) | O(N) |

**Advantages, Disadvantages, Recommendations**

When it comes to the three options presented, they all have some advantages and disadvantages. Vectors are particularly fast at loading data into an unsorted vector; however, sorting that data later becomes much slower than other options. Hash tables can be high performing, assuming they are large enough to avoid collisions. Since this is unlikely, the table needs to be able to handle collisions which slows performance. Binary search trees can be the most consistent with performance overall, but there is always the issue of how the data is read in. The tree can become unbalanced very quickly, creating a large slowdown especially to the searching and sorting that data. As far as a recommendation goes, it truly depends on the needs of the data. If the data is meant to be constantly rearranged and sized, something like a binary search tree is much easier to do that with. If the searching speed is a concern, the real question would be the difference between the hash table and the binary search tree. If the hash table is designed well, it can prove to be much faster than a binary search tree that is not perfectly balanced. Both options in the end will be much preferable than the vector. Assuming the binary search tree is not unbalanced, I would recommend this as the preferred option as it is able to be sized and sorted much faster than a hash table that is not created in order. Alternatively the hash table would be the second recommendation, but only in the event hat the table size is optimized to limit collisions and not much additional data is expected to be added.