Almera Wardle

Professor Saba Jamaliannasrabadi, M.S

CS – 300 DSA: Analysis and Design

October 19, 2024

**Evaluation**

**Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Worst case running time of reading the file | 1 | n | n |
| Worst case running time of creating course objects | 1 | n | n |
| Total cost |  |  | 2n |
| Runtime |  |  | O(n) |

**Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Worst case running time of reading the file | 1 | n | n |
| Worst case running time of creating course objects | O(1) | n | O(n) |
| Total cost |  |  | O(n) |
| Runtime |  |  | O(n) |

**Binary Search Tree**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Worst case running time of reading the file | 1 | n | n |
| Worst case running time of creating course objects | O(log n) | n | O(n log n) |
| Total cost |  |  | O(n log n) |
| Runtime |  |  | O(n log n) |

**Advantages and Disadvantages**

Vectors are simple data structures that provide efficient sequential access to elements, making them suitable for applications where order matters and data is accessed by position. However, they are inefficient for insertion and deletion operations, as these require shifting elements, resulting in O(n) complexity. Searching in a vector is also inefficient, with a linear O(n) time complexity, which can be a drawback when fast lookups are needed.

Hash tables offer fast O(1) average-time complexity for insertions, deletions, and lookups, making them ideal for quick access to data. However, they do not maintain order, which can be a limitation when sorted access is needed. While collisions are rare with a good hash function, they can cause performance degradation to O(n), and the memory overhead due to extra space for collision handling can be higher compared to other data structures.

Binary search trees (BSTs) provide efficient O(log n) operations for insertions, deletions, and searches when balanced, and they maintain data in a sorted order, which is useful for range queries. However, an unbalanced tree can degrade performance to O(n), and the memory overhead is higher due to the need to store child node pointers. Maintaining a balanced tree also adds complexity to the implementation.

**Recommendation**

Based on my analysis of vectors, hash tables, and binary search trees, I recommend using hash tables for the pseudocode implementations. Hash tables offer the best overall performance, with an average-case time complexity of O(1) for insertion, deletion, and lookup operations. Given that the pseudocode frequently involves tasks like searching for specific courses and validating prerequisites, the fast lookup time of a hash table makes it the most efficient choice.

Vectors, while simple and effective for sequential access, have a linear O(n) complexity for searching and insertion operations, which would result in slower performance as the number of courses increases. Binary search trees (BSTs), on the other hand, offer O(log n) time complexity for balanced trees, but they require additional overhead to maintain balance. In the worst-case scenario (an unbalanced tree), the time complexity could degrade to O(n), making it less reliable in terms of performance.

The hash table's constant-time operations, combined with the ability to handle course lookups and validations quickly, make it the most suitable data structure for this scenario. The small memory overhead and occasional collision handling are outweighed by the significant performance benefits, especially when compared to the other two structures.