**Alondra Paulino Santos**

**Run Time and Memory Analysis**

**Vector**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **open file / error check** | 1 | 1 | 1 |
| **loop over lines** | 1 | n | n |
| **split + trim tokens** | 1 | n | n |
| **create Course object** | 1 | n | n |
| **add to temp map by courseNumber** | 1 | n | n |
| **validate prerequisites** | 1 | n | n |
| **append course to vector** | 1 | n | n |
| **Total Cost** | | | 6n + 1 |
| **Worst-case Runtime** | | | O(n) |

**Vector** is the most memory-efficient because it stores data in a contiguous block with no extra pointers. Its runtime is O(n) for most operations, or O(nk) if you include prerequisite validation. It’s simple and efficient, but not great for fast searching.

**Hash Table**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **open file / error check** | 1 | 1 | 1 |
| **loop over lines** | 1 | n | n |
| **split + trim tokens** | 1 | n | n |
| **create Course object** | 1 | n | n |
| **add to temp map by courseNumber** | 1 | n | n |
| **validate prerequisites** | 1 | n | n |
| **insert into hash table (worst-case collisions)** | n | n | n^2 |
| **Total Cost** | | | n^2 + 5n + 1 |
| **Worst-case Runtime** | | | O(n^2) |

**Hash Table** uses more memory due to buckets and per-node pointers. On average, it gives very fast lookups (close to O(1)), but in the worst case, with many collisions, it can degrade to O(n²) or O(nk). The extra memory tradeoff is worth it for speed in most practical cases.

**Binary Search Tree**

| **open file / error check** | 1 | 1 | 1 |
| --- | --- | --- | --- |
| **loop over lines** | 1 | n | n |
| **split + trim tokens** | 1 | n | n |
| **create Course object** | 1 | n | n |
| **add to temp map by courseNumber** | 1 | n | n |
| **validate prerequisites** | 1 | n | n |
| **BST insert (unbalanced)** | n | n | n^2 |
| **Total Cost** | | | n^2 + 5n + 1 |
| **Worst-case Runtime** | | | O(n^2) |

**Binary Search Tree (BST)** uses the most memory since each node stores two pointers (left and right). If balanced, it offers efficient searching and traversal (O(log n) per insert/search, or O(n log n) total with prerequisites). If unbalanced, performance drops to O(n²).

**Comparison Summary:**

Vector is the most memory-efficient and simple, but has slower lookups. Hash Table trades extra memory for the fastest average lookups, though it can degrade in the worst case. Binary Search Tree uses the most memory, and while it performs well if balanced, it becomes inefficient when unbalanced. Overall, the Hash Table is the best choice for this advising program because it balances memory use with fast performance while still allowing a sorted view when needed.