Legend For Runtime Analysis Tables

| Term | Meaning |
| --- | --- |
| Code | Step or operation from the pseudocode being analyzed |
| Line Cost | Constant time cost for the step when executed once |
| Number of TImes Executed | Number of times the step runs *n* courses in the input file |
| Total Cost | Total work for that step calculated as *Line Cost x Number of Times Executed* |

Vector Runtime Analysis

| Code | Line Cost | # of Times Executed | Total Cost |
| --- | --- | --- | --- |
| Open File | 1 | 1 | 1 |
| While not the End of File | 1 | *n* | *n* |
| Split line and trim fields | 1 | *n* | *n* |
| Create course object | 1 | *n* | *n* |
| Append course to end of vector | 1 | *n* | *n* |
| Close file | 1 | 1 | 1 |
| Total Cost |  |  | 4*n* +2 |

Runtime: *O(n)*

Load is linear because append is constant on average. Exact course lookup is linear. A full ordered print requires a separate sort; radix is O(k·n) with small fixed k for these IDs, so close to linear in practice. Validating prerequisites by scanning the vector during load, that separate validation can push load toward O(n²), which is outside the loader analysis.

Hash Table Runtime Analysis

| Code | Line Cost | #of Times Executed | Total Cost |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| While not at end of file | 1 | n | n |
| Split line and trim fields | 1 | n | n |
| Create course object | 1 | n | n |
| Hash key and insert into table | 1 avg  n worst | n | n avg  n2 worst |
| Close file | 1 | 1 | 1 |
| Total Cost |  |  | 4n+2 avg n2+3n+2 worst |

Runtime: O(n) average O(n2) worst case

Average inserts and lookups are constant, so single course queries and prerequisite checks are fast. Order is not preserved, so printing an ordered list requires extracting all items and sorting them. Collisions and a high load factor can push worst case toward O(n²).

Binary Search Tree Runtime Analysis

| Code | Line Cost | #of Times Executed | Total Cost |
| --- | --- | --- | --- |
| Open file | 1 | 1 | 1 |
| While not at end of file | 1 | n | n |
| Split line and trim fields | 1 | n | n |
| Create course object | 1 | n | n |
| Hash key and insert into table | Log n avg  n worst | n | n log n avg  n2 worst |
| Close file | 1 | 1 | 1 |
| Total Cost |  |  | n log n + 3n + 2 avg n2+3n+2 worst |

Runtime O(n log n) average O(n2) worst case

Insertion traverses to the sorted position, giving n log n average and n² worst case if the tree skews. In-order traversal prints a fully sorted list in O(n) with no extra sort step. If input arrives nearly sorted and the tree is not self-balancing, height can degrade toward linear.

I recommend implementing a Binary Search Tree keyed by courseNumber. It satisfies both advising requirements directly; an in order traversal prints the complete course list in alphanumeric order in O(n) with no separate sort, and searching by courseNumber is O(log n) on average. Vector and hash table implementations do not maintain order, so printing the full list requires extracting all courses and sorting each time. For the runtime comparison, the loader, which refers to the part that opens the course file, reads n lines, builds Course objects, inserts them into the chosen structure, and closes the file. The tables measure only this loader path, and these tables justify choosing the tree for this assignment.