CS-300 Project One

Keir Charlton-Molloy

Runtime & Memory Analysis (Worst Case)

Vector: File-Load Phase

Code Line	Line Cost	# Times Executes	Total Cost
open file	1	1	1
read each line (pass 1)	1	n	n
split line into tokens	1	n	n
validate token count (≥ 2)	1	n	n
collect course numbers	1	n	n
reset file pointer	1	1	1
read each line (pass 2)	1	n	n
split line into tokens (pass 2)	1	n	n
check each prerequisite in HashSet	1	n^2	n^2
create Course object	1	n	n
push Course into vector	1	n	n
Total Cost			**$2 + 8n + n^2$**
Worst-Case Big-O			**$O(n^2)$**

Memory note: A vector stores all Course objects in one contiguous block and overhead is limited to any unused capacity the dynamic array reserves.

Hash Table: File‑Load Phase

Code Line	Line Cost	# Times Executes	Total Cost
open file	1	1	1
read each line (pass 1)	1	n	n
split line into tokens	1	n	n
validate token count (≥ 2)	1	n	n
collect course numbers	1	n	n
reset file pointer	1	1	1
read each line (pass 2)	1	n	n
split line into tokens (pass 2)	1	n	n
check each prerequisite in HashSet	1	n^2	n^2
create Course object	1	n	n
insert Course into hash table	1	n	n
Total Cost			**$2 + 8n + n^2$**
Worst‑Case Big‑O			**$O(n^2)$**

Memory note: Requires an array of buckets plus one pointer per element. With a 0.75 load factor the table typically holds about 1.3–1.5× the raw data size.

Binary Search Tree: File-Load Phase

Code Line	Line Cost	# Times Executes	Total Cost
open file	1	1	1
read each line (pass 1)	1	n	n
split line into tokens	1	n	n
validate token count (≥ 2)	1	n	n
collect course numbers	1	n	n
reset file pointer	1	1	1
read each line (pass 2)	1	n	n
split line into tokens (pass 2)	1	n	n
check each prerequisite in HashSet	1	n^2	n^2
create Course object	1	n	n
insert Course into balanced BST	log n	n	n log n
Total Cost			**$2 + 7n + n \log n + n^2$**
Worst-Case Big-O			**$O(n^2)$**

Memory note: Each node stores the Course plus two child pointers (and sometimes a parent pointer), giving ~3 pointers of overhead per course but no bucket array.

Advantages and Disadvantages (Vector, Hash Table, Binary Search Tree)

An advantage of a vector is that it keeps every Course object in one block of memory, so traversing the list to print the full catalogue is cache‑friendly and simple to code. Sorting is also straightforward, as the standard library's sort function can produce a fully ordered alphanumeric list with minimal effort. The memory overhead is also minimal since the vector only holds the course objects themselves. However, disadvantages occur when the advisors need to insert new courses, remove existing ones, or search for a single course since each of those operations can require shifting many elements and therefore runs in linear time. If the advisors load data once and then ask for the list repeatedly, the vector must either maintain a permanently sorted copy (duplicating storage) or re‑sort on every request, adding an *O(n log n)* cost each time.

An advantage of a hash table is that it excels at constant‑time, average‑case insertions and look‑ups. This makes it well-suited for quickly finding one specific course together with its prerequisites. Load performance stays linear because each insertion costs *O(1)* on average, and extra bucket capacity allows it to grow without needing to resize right away. The disadvantage to hash tables is that it does not preserve order. To meet the advisor's requirement for an alphanumeric catalogue, the program must extract every key, place them in a temporary vector, and sort that vector. This adds an additional *O(n log n)* step every time the list is printed. Hash collisions can also reduce the constant‑time performance to linear time in the worst case, and bucket arrays often use more memory than necessary for the data. These trade‑offs make hash tables less suitable for programs that need to produce sorted output frequently and reliably.

An advantage of a binary search tree (BST) is that it stores courses in a way that naturally produces a fully sorted list through an in-order traversal, so no additional sorting is needed after loading. Finding a specific course, such as when an advisor enters a course number, takes O(log n) time, which is faster than the linear search required by a vector. Insertions during the load phase also takes O(log n) per insertion, resulting in a total load time of O(n log n). While this is slower than a hash table's average-case load time, it's only a one-time cost at startup. Memory usage is moderate since each node carries two child pointers (and sometimes a parent pointer) in addition to the Course object. The main disadvantage is if the tree becomes unbalanced it can hurt performance. However, maintaining balance is well understood and relatively efficient using modern self-balancing trees like AVL or red-black trees.

Recommendation

After the one-time file load, the program must repeatedly do two main things. First, it must print the entire Computer Science catalogue in alphanumeric order. Second, it must retrieve a single course together with its prerequisites. A balanced BST is the best choice because it satisfies both tasks with no extra sorting and with logarithmic-time search. Although its initial load phase is *O(n log n)*, which is slower than the hash table's linear insertions, the advisors pay that cost only once. In contrast, a vector or hash table would repay the sort penalty every time they ask for the course list. Vectors also have slower, linear-time searches, and hash tables don't maintain order, so sorting is still needed that cancels out their speed advantage. Overall, the balanced BST has the fastest overall performance during typical advising sessions, uses a reasonable amount of memory, and provides reliable results, making it the best choice for the final implementation.