Data Structure	Total Cost Formula	Big-O	Memory Usage	Advantages	Disadvantages
Vector	$(6 + k) \times n$	$O(n)$	$O(n)$	• Very simple to implement • Low per-item overhead	• Lookup by course → $O(n)$ • To list in order requires an explicit sort → $O(n \log n)$
Hash Table	$(6 + k) \times n$ (avg. case)	$O(n)$ (avg. case)	$O(n)$	• Average-case $O(1)$ insert & lookup • Fast prerequisite validation	• No inherent ordering (must sort keys → $O(n \log n)$) • Extra memory for buckets • Worst-case collisions
BST	**Unbalanced:** $(6 + k + n) \times n = O(n^2)$ **Balanced:** $(6 + k + \log n) \times n = O(n \log n)$	$O(n^2)$ worst-case $O(n \log n)$ if balanced	$O(n)$	• Inorder traversal gives sorted list in $O(n)$ • $O(\log n)$ insert & lookup if balanced	• Unbalanced tree can degrade to $O(n^2)$ overall • Pointer overhead & more complex to implement

Recommendation:
I recommend using the hash table implementation for this application. Although a BST provides a convenient way to list courses in order, the hash table offers fast average-case insertions and lookups, which is highly beneficial when an advisor is querying for a course's details and prerequisites. The cost of sorting the keys for the "print all courses" operation ($O(n \log n)$) is acceptable given the relatively small and fixed number of courses in a typical Computer Science curriculum. Overall, the hash table strikes the best balance between performance for frequent lookups and acceptable performance for occasional sorting, with predictable $O(n)$ loading time and $O(1)$ average lookup performance.