Jasmine Garcia

CS 300: Data Analysis and Design

Instructor Mike Susalla

June 2025

Project One

Below is pseudocode for Project One:

**Main Menu**

Display menu:

1. Load Courses

2. Print All Courses

3. Search for a Course

9. Exit

Prompt the user to select…

If the user enters ‘1’, call load courses

If the user enters ‘2’, call print courses

If the user enters ‘3’, call search courses

If the user enters ‘9’, exit program

**Vector Implementation**

Load all courses from the CSV file using the vector

Open CSV file at CSV path

For each row in file:

Load course ID, course title, and prerequisites

Store course in course list vector

Print all courses from the vector

Sort course list by course ID using quicksort

For each course in course list:

Print course ID, title, and any prerequisites

Search for a specific course in the vector

For each course in course list

If course ID matches course ID

Print course info and prerequisites

**Hash Table Implementation**

Load courses the CSV file using a hash table

Open CSV file at CSV path

For each row in file:

Load course ID, title, and prerequisites

Create hash key using full course ID (not numeric portion)

Store course in hash table at key index

Print all courses from the hash table

For each bucket in course table

If bucket is *not* empty:

Print course ID, title, and prerequisites

Search for a specific course in the hash table

Compute hash key for course ID

Navigate list at hash index

If course ID matches course ID:

Print course info and prerequisites

**Binary Search Tree Implementation**

Load all courses from the CSV file using a BST

Open CSV file at csvPath

For each row in file:

Load course ID, title, and prerequisites

Insert course into BST based on course ID

Print all courses from the BST in order

If node is *not* null:

Call the left subtree

Print course ID, title, and prerequisites

Call the right subtree

Search for a course in the BST

If node *is* null, return not found

If course ID matches node:

Print course info and prerequisites

Else if course ID is less than node course ID

Call left subtree

Else:

Call right subtree

**Runtime Analysis Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | **Vector** | **Hash Table** | **Binary Search Tree** |
| Load Courses | O(n) | O(n) [with collisions] | O(n log n) [balanced] |
| Search Courses | O(n) | O(1)-O(n) | O(log n)-O(n) |
| Print Sorted | O(n log n) | O(n)+sort step? | O(n) [in order] |

**Analysis**

Vectors load quickly when data doesn’t need to be sorted and ultimately, they are really easy to use. However, they are slower when it comes to searching, and users must sort them manually prior to printing.

The pros of hash tables are that they’re very efficient in the event that collisions are minimal, and also they have a fast lookup at O(1). The cons of hash tables are that they are hard to hash full course IDs accurately, and they’re not sorted. Therefore, they require more structure to print in the correct order.

Binary search trees, or BSTs, are great for printing and searching quickly, especially when they’re balanced. The cons of BSTs are that if the input is already sorted, they can become inefficient. Additionally, they’re a bit more complex to set up than a vector.

**Recommendation**

For ABC University, they need to quickly search and print courses in alphabetical order, likely multiple times each session. For that reason, I think a binary search tree is the best option for ABCU. BSTs make searching efficient, they keep data sorted, and handle memory better than vectors. Lastly, BSTs avoid the difficulties of handling collisions and sorting like in a hash table.