Brycen McEuen

Professor Susalla

CS-300

June 15, 2025

**Project One Pseudocode**

**Menu:**

Int Main

Define menu switch options

Define timer variable

Define data structure used to hold course information

Initialize data structure

Int choice equals zero

WHILE choice does not equal nine

Print menu options

Case 1:

Initialize timer variable

Call method to load course file

Calculate elapsed time and print results

Break

Case 2:

Call method to print courses in alphabetical order

Break

Case 3:

Initialize timer variable

Call method to search for course

Start timer

IF course ID is not empty

Call method to display course information

ELSE

“Course is not found.”

Break

Case 9:

Print goodbye message

Break

**Print Courses for vector:**

Use std::sort to sort courses alphabetically

FOR each course in course vector structure

Print course and prerequisites

**Print courses for hash table:**

Int J equals zero

Create loop to print all courses

Current node equals integer I

IF current node is not at maximum value

Increment int J by one

Print course information

WHILE next node is not NULL

Increment int J by one

Go to next node

Print course information

**Print courses for tree:**

IF node is not NULL

Traverse left side

Print course information

Traverse right side

**Big O Analysis:**

Vector:

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Initialize vector** | 1 | 1 | 1 |
| **For each line** | 1 | n | n |
| **For first and second value** | 1 | n | n |
| **Push back value to vector** | 1 | n | n |
| **If more than two values** | 1 | n | n |
| **Push back until newline** | 1 |  |  |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

Hash Table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Declare course structure** | 1 | 1 | 1 |
| **Create hash table class** | 1 | 1 | 1 |
| **Create structure for nodes** | 1 | 1 | 1 |
| **Constructor** | n | 1 | n |
| **Declare hash function** | 1 | 1 | 1 |
| **Declare insert method** | 1 | 1 | 1 |
| **Hash key using hash function** | n | n | n |
| **Check if bucket is empty** | n | n | n |
| **If empty, create new node and add item** | n | n | n |
| **Else, handle collision** | n | n | n |
| **Set cursor to hash location** | n | n | n |
| **Make newNode pointer** | n | n | n |
| **Find last node in chain** | n | n | n |
| **Append new node** | n | n | n |
| **Total Cost** | | | 9n + 5 |
| **Runtime** | | | O(n) |

Binary Search Tree:

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Create BST class** | 1 | 1 | 1 |
| **Declare variables** | 1 | 1 | 1 |
| **Constructor** | 1 | 1 | 1 |
| **Destructor** | n | n | n |
| **Remove node** | n | n | n |
| **Total Cost** | | | 2n + 3 |
| **Runtime** | | | O(n) |

As you can see, the vector and BST are fairly similar in terms of cost and speed, whereas the hash table is quite a bit bigger. Due to the slower speed and complexity of hash tables, I would recommend against them for this scenario. Another consideration is that vectors don’t sort automatically and will have to be sorted separately after being loaded into the data structure. So, if the courses need to be sorted alphabetically, that means you’ll have to do more work when building a vector-based storage system. For these reasons, I would recommend using a Binary Search Tree structure to handle the courses. These will provide fast lookups by course number and come automatically sorted by default.