ABCU

Derricko Swink

Vector Sorting

Prerequisites

```
FUNCTION loadCourses(filePath)
    OPEN filePath FOR READING
    WHILE NOT EndOfFile
        READ line
        SPLIT line INTO fields using ',' as delimiter

        CREATE course : Course
        course.courseNumber = fields[0]
        course.name = fields[1]
        course.prerequisites = []

        FOR i FROM 2 TO LENGTH(fields) - 1
            IF isValidCourseNumber(fields[i]) THEN
                ADD fields[i] TO course.prerequisites
            ELSE
                PRINT "Invalid prerequisite:", fields[i]
            END IF
        END FOR

        STORE course IN coursesCollection
    END WHILE
    CLOSE file
```

```
END FUNCTION
```

Menu

```
FUNCTION displayMenu()
    WHILE true
        PRINT "1. Load Course Data"
        PRINT "2. Display All Courses"
        PRINT "3. Display Course Details"
        PRINT "4. Exit"

        READ choice
        SWITCH(choice)
            CASE 1:
                CALL loadCourses(filePath)
            CASE 2:
                CALL displayAllCourses()
            CASE 3:
                PRINT "Enter Course Number:"
                READ courseNumber
                CALL displayCourseDetails(courseNumber)
            CASE 4:
                PRINT "Exiting program."
                BREAK
            DEFAULT:
                PRINT "Invalid choice. Try again."
        END SWITCH
    END WHILE
END FUNCTION
```

Sorted Course List

```
FUNCTION displayAllCourses()
    IF usingVector THEN
        SORT courses ALPHABETICALLY by courseNumber
    ELSE IF usingHashTable THEN
        EXTRACT courseNumbers, SORT them, and RETRIEVE courses
    ELSE IF usingBST THEN
        CALL inOrderTraversal(root)
    END IF

    FOR EACH course IN sorted list or traversal result
        PRINT course.courseNumber, course.name
    END FOR
END FUNCTION
```

Runtime Evaluation

Runtime Analysis Chart

Operation	Vector	Hash Table	Binary Search Tree (BST)
Insertion	O(1) (end) / O(n) (mid)	O(1) (average) / O(n) (worst)	O(log n) (average) / O(n) (worst)
Seach (by Course Number)	O(n)	O(1) (average) / O(n) (worst)	O(log n) (average) / O(n) (worst)
Deletion	O(n)	O(1) (average) / O(n) (worst)	O(log n) (average) / O(n) (worst)
Traverse All (Print All)	O(n)	O(n)	O(n)
Sort	O(n log n)	Not directly applicable	In-order traversal: O(n)
Memory Usage	Dynamic (resizable)	Dynamic (depends on hash size)	Depends on tree balance

Vector:

- Insertion: O(1) (end) or O(n) (specific index)
- Search: O(n)
- Sort: O(n log n)

Hash Table:

- Insertion/Search: O(1) average, O(n) worst-case (collisions)
- Traversal: O(n)

Binary Search Tree (BST):

- Insertion/Search: O(log n) average, O(n) worst-case
- Traversal (in-order): O(n)

Advantages & Disadvantages

Vector:

Advantages: Simple, good for small datasets, easy to implement.

Disadvantages: Inefficient search and insert operations for large datasets.

Hash Table:

Advantages: Fast lookups, ideal for large datasets with infrequent sorted operations.

Disadvantages: Collision handling required, unordered storage.

Binary Search Tree (BST):

Advantages: Maintains sorted order, efficient searches with balanced trees.

Disadvantages: Can be inefficient if unbalanced; more complex implementation.

Recommendation

For fast lookups and frequent access by course number, I recommend using a Hash Table.

For frequent sorted traversal and display, I recommend using Binary Search Tree (BST).

For simplicity (and if the dataset is small), I recommend using Vector if sorted retrieval isn’t frequent.