SRM UNIVERSITY AP

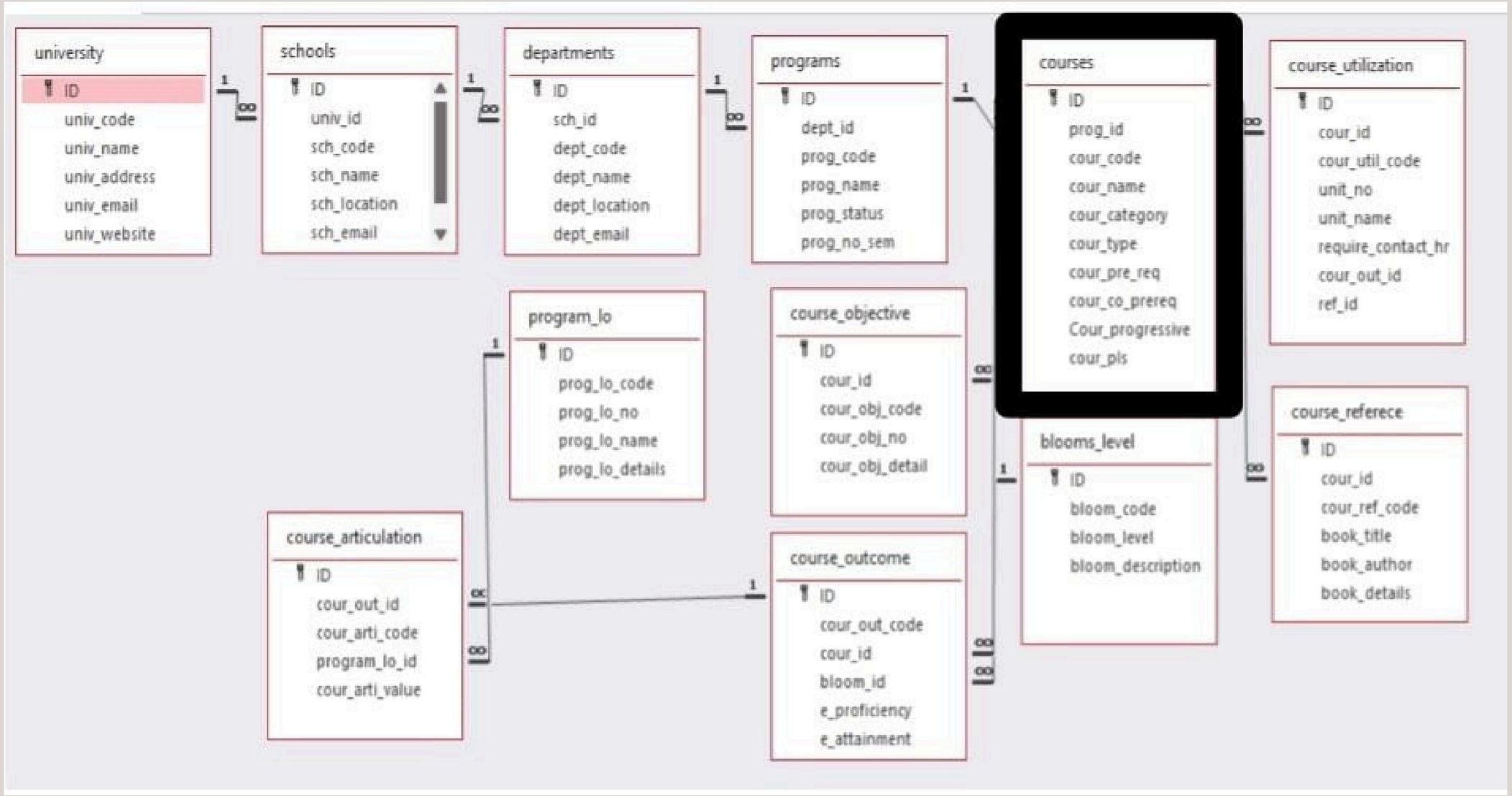
courses

Madhu Swapnika-AP23110011011
Abhighna-AP23110011051
Nikitha- AP23110011062
Akhil sai -AP23110011013
Losishram-AP23110010703

INTRODUCTION

The Course Management System is a C++ program that efficiently manages course details like course ID, program ID, course code, course name, prerequisites , corequisites, and progression. t supports adding, retrieving, searching, sorting, and deleting courses.
Binary Search is used for fast course search by course code with a time complexity of O(log n).
Merge Sort organizes courses by code, ensuring an efficient sort with O(n log n) complexity.
Ideal for educational institutions, this system handles large datasets, providing quick access and updates to course information.
.

ARCHITECTURE DIAGRAM



MODULE DESCRIPTION

COURSE CLASS:

- Stores course details such as ID, program ID, code, name, category, type,prerequisites, corequisites, progression, and PLS.

->CourseManager Class: addCourse(): Add new courses.
->removeCourse(): Remove courses by their code.
->displayCourses(): Display all course details.
->findCourseByCode(): Search for a course using binary search.
->sortCoursesByCode(): Sort courses by code using merge sort.
->Binary Search Module: Efficient method to search for courses by code.
->Merge Sort Module: Sorts courses efficiently by their code.
Main Program: Brings together all functionalities—adding, sorting, searching, and displaying courses.

COURSES: FIELD DETAILS

course_id	integer
program_id	integer
course_code	string
course_name	string
course_category	string
course_type	string
pre_requisite	string
co_requisite	string
progessive	string
course_pls	string

ALGORITHM DETAILS

SEARCHING:

1. Course Search (Binary Search)

Function: Finds a course by its code.

Algorithm: Binary Search

Time Complexity: O(log n) (efficient search, requires sorted data)

SORTING:

2. Course Sorting (Merge Sort)

Function: Sorts courses by course code.

Algorithm: Merge Sort

Time Complexity: O(n log n) (efficient sorting)

STORING:

3. Storing Courses (File Handling)

Function: Saves courses to a file for persistence.

Time Complexity: O(n) (depending on file I/O)

SOURCE CODE:

```cpp
#include <iostream>
# C:\Program Files (x86)\Dev-Cpp\MinGW64\lib\gcc\x86_64-w64-mingw32\4.9.2\include\c++\iostream - Ctrl+Click for

using namespace std;

const int MAX_EXECUTIVES = 100; // Maximum number of executives
struct Executive {
    int executive_id;
    int program_id;
    string executive_code;
    string executive_name;
    string executive_category;
    string executive_type;
    string pre_requisite;
    string co_requisite;
    string progressive;
    string pls; // Program Learning Statement
};

// Array to store executives
Executive executives[MAX_EXECUTIVES];
int executive_count = 0; // Current number of executives

// Function prototypes
void create_executives();
void retrieve_executives();
void delete_executives();
void merge_sort_executives(int low, int high);
void binary_search_executives();
void merge_executives(int low, int mid, int high);

// Function to create a new executive
void create_executives() {
    if (executive_count >= MAX_EXECUTIVES) {
        cout << "Cannot add more executives. Maximum limit reached.\n";
        return;
    }

    Executive new_executive;
    cout << "Enter Executive ID: ";
    cin >> new_executive.executive_id;
    cout << "Enter Program ID: ";
    cin >> new_executive.program_id;
    cout << "Enter Executive Code: ";
    cin >> new_executive.executive_code;
    cout << "Enter Executive Name: ";
    cin.ignore();
    getline(cin, new_executive.executive_name);
    cout << "Enter Executive Category: ";
    getline(cin, new_executive.executive_category);
    cout << "Enter Executive Type: ";
    getline(cin, new_executive.executive_type);
    cout << "Enter Pre-requisites: ";
    getline(cin, new_executive.pre_requisite);
    cout << "Enter Co-requisites: ";
    getline(cin, new_executive.co_requisite);
    cout << "Enter Executive Progression: ";
    getline(cin, new_executive.progressive);
    cout << "Enter PLS (Program Learning Statement): ";
    getline(cin, new_executive.pls);

    executives[executive_count++] = new_executive;
    cout << "Executive created successfully.\n";
}

// Function to retrieve all executives
void retrieve_executives() {
    for (int i = 0; i < executive_count; i++) {
        Executive& e = executives[i];
        cout << "ID: " << e.executive_id << ", Program ID: " << e.program_id << ", Code: " << e.executive_code
             << ", Name: " << e.executive_name << ", Category: " << e.executive_category
             << ", Type: " << e.executive_type << ", Pre-requisite: " << e.pre_requisite
             << ", Co-requisite: " << e.co_requisite << ", Progression: " << e.progressive
             << ", PLS: " << e.pls << endl;
```

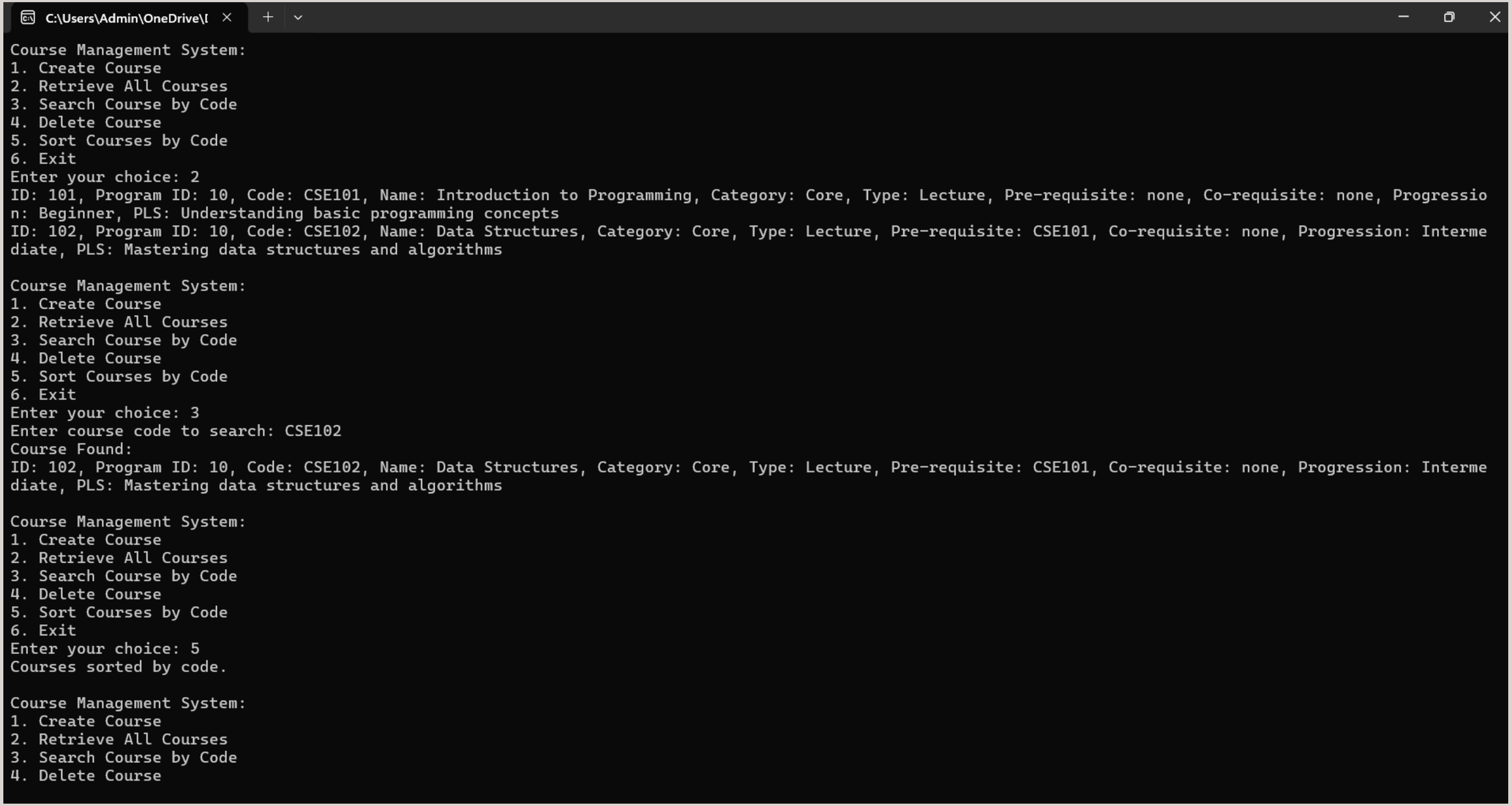
```cpp
// Merge function
void merge_executives(int low, int mid, int high) {
    int n1 = mid - low + 1;
    int n2 = high - mid;

    Executive left[n1], right[n2];
    for (int i = 0; i < n1; i++)
        left[i] = executives[low + i];
    for (int i = 0; i < n2; i++)
        right[i] = executives[mid + 1 + i];

    int i = 0, j = 0, k = low;
    while (i < n1 && j < n2) {
        if (left[i].executive_code <= right[j].executive_code) {
            executives[k++] = left[i++];
        } else {
            executives[k++] = right[j++];
        }
    }
    while (i < n1) {
        executives[k++] = left[i++];
    }
    while (j < n2) {
        executives[k++] = right[j++];
    }
}

// Merge Sort function
void merge_sort_executives(int low, int high) {
    if (low < high) {
        int mid = low + (high - low) / 2;
        merge_sort_executives(low, mid);
        merge_sort_executives(mid + 1, high);
        merge_executives(low, mid, high);
    }
}
```

```cpp
inary Search function
l binary_search_executives() {
string code;
cout << "Enter executive code to search: ";
cin >> code;

int low = 0, high = executive_count - 1;
bool found = false;

while (low <= high) {
    int mid = low + (high - low) / 2;
    if (executives[mid].executive_code == code) {
        cout << "Executive Found:\n"
             << "ID: " << executives[mid].executive_id << ", Program ID: " << executives[mid].program_
             << ", Code: " << executives[mid].executive_code << ", Name: " << executives[mid].executiv
             << ", Category: " << executives[mid].executive_category << ", Type: " << executives[mid].
             << ", Pre-requisite: " << executives[mid].pre_requisite << ", Co-requisite: " << executiv
             << ", Progression: " << executives[mid].progressive << ", PLS: " << executives[mid].pls <
        found = true;
        break;
    } else if (executives[mid].executive_code < code) {
        low = mid + 1;
    } else {
        high = mid - 1;
    }
}

if (!found) {
    cout << "Executive with code " << code << " not found.\n";
}


unction to delete an executive
```

```
void delete_executives() {
    int executive_id;
    cout << "Enter executive ID to delete: ";
    cin >> executive_id;

    bool found = false;
    for (int i = 0; i < executive_count; i++) {
        if (executives[i].executive_id == executive_id) {
            for (int j = i; j < executive_count - 1; j++) {
                executives[j] = executives[j + 1];
            }
            executive_count--;
            found = true;
            break;
        }
    }

    if (found) {
        cout << "Executive deleted successfully.\n";
    } else {
        cout << "Executive with ID " << executive_id << " not found.\n";
    }
}

// Main function
int main() {
    int choice;

    do {
        cout << "\nExecutive Management System:\n";
        cout << "1. Create Executive\n";
        cout << "2. Retrieve All Executives\n";
        cout << "3. Search Executive by Code\n";
        cout << "4. Delete Executive\n";
        cout << "5. Sort Executives by Code\n";
        cout << "6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                create_executives();
                break;
            case 2:
                retrieve_executives();
                break;
            case 3:
                merge_sort_executives(0, executive_count - 1); // Ensure the list is sorted befo
                binary_search_executives();
                break;
            case 4:
                delete_executives();
                break;
            case 5:
                merge_sort_executives(0, executive_count - 1);
                cout << "Executives sorted by code.\n";
                break;
            case 6:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice, please try again.\n";
        }
    } while (choice != 6);

    return 0;
```

SAMPLE OUTPUT

```
Course Management System:
1. Create Course
2. Retrieve All Courses
3. Search Course by Code
4. Delete Course
5. Sort Courses by Code
6. Exit
Enter your choice: 1
Enter Course ID: 101
Enter Program ID: 10
Enter Course Code: CSE101
Enter Course Name: Introduction to Programming
Enter Course Category: Core
Enter Course Type: Lecture
Enter Pre-requisites: none
Enter Co-requisites: none
Enter Course Progression: Beginner
Enter PLS (Program Learning Statement): Understanding basic programming concepts
Course created successfully.

Course Management System:
1. Create Course
2. Retrieve All Courses
3. Search Course by Code
4. Delete Course
5. Sort Courses by Code
6. Exit
Enter your choice: 1
Enter Course ID: 102
Enter Program ID: 10
Enter Course Code: CSE102
Enter Course Name: Data Structures
Enter Course Category: Core
Enter Course Type: Lecture
Enter Pre-requisites: CSE101
Enter Co-requisites: none
Enter Course Progression: Intermediate
Enter PLS (Program Learning Statement): Mastering data structures and algorithms
Course created successfully.
```


C:\Users\Admin\OneDrive\[
Course Management System:
1. Create Course
2. Retrieve All Courses
3. Search Course by Code
4. Delete Course
5. Sort Courses by Code
6. Exit
Enter your choice: 2
ID: 101, Program ID: 10, Code: CSE101, Name: Introduction to Programming, Category: Core, Type: Lecture, Pre-requisite: none, Co-requisite: none, Progressio
n: Beginner, PLS: Understanding basic programming concepts
ID: 102, Program ID: 10, Code: CSE102, Name: Data Structures, Category: Core, Type: Lecture, Pre-requisite: CSE101, Co-requisite: none, Progression: Interme
diate, PLS: Mastering data structures and algorithms
Course Management System:
1. Create Course
2. Retrieve All Courses
3. Search Course by Code
4. Delete Course
5. Sort Courses by Code
6. Exit
Enter your choice: 3
Enter course code to search: CSE102
Course Found:
ID: 102, Program ID: 10, Code: CSE102, Name: Data Structures, Category: Core, Type: Lecture, Pre-requisite: CSE101, Co-requisite: none, Progression: Interme
diate, PLS: Mastering data structures and algorithms
Course Management System:
1. Create Course
2. Retrieve All Courses
3. Search Course by Code
4. Delete Course
5. Sort Courses by Code
6. Exit
Enter your choice: 5
Courses sorted by code.
Course Management System:
1. Create Course
2. Retrieve All Courses
3. Search Course by Code
4. Delete Course

```
Course Management System:
1. Create Course
2. Retrieve All Courses
3. Search Course by Code
4. Delete Course
5. Sort Courses by Code
6. Exit
Enter your choice: 4
Enter course ID to delete: 101
Course deleted successfully.

Course Management System:
1. Create Course
2. Retrieve All Courses
3. Search Course by Code
4. Delete Course
5. Sort Courses by Code
6. Exit
Enter your choice: 2
ID: 102, Program ID: 10, Code: CSE102, Name: Data Structures, Category: Core, Type: Lecture, Pre-requisite: CSE101, Co-requisite: none, Progression: Interme
diate, PLS: Mastering data structures and algorithms

Course Management System:
1. Create Course
2. Retrieve All Courses
3. Search Course by Code
4. Delete Course
5. Sort Courses by Code
6. Exit
Enter your choice: 6
Exiting...

--------------------------------
Process exited after 306.5 seconds with return value 0
Press any key to continue . . .
```

Comparison of searching algorithms:

LINEAR SEARCH	BINARY SEARCH
It has a time complexity of O(n) , meaning in the worst case, it may need to check all n elements in the dataset.	It has a time complexity of O(log n), meaning it dramatically reduces the number of comparisons by dividing the search space in half with each step.
Checks each element one by one sequentially.	Divides the dataset into halves and focuses on the relevant half.
Less efficient as the dataset size increases.	Highly efficient for large, sorted datasets.

comparision of sorting algorithm:

MERGE SORT	QUICK SORT
o(n log n) in all cases.	O(n2) in the worst case.
Provides consistent performance across all datasets.	Performance can degrade with poor pivot choice.
Effective for sorting large datasets that don't fit into memory.	Less efficient with large, external datasets

Conclusion:

The course management system streamlines course handling by efficiently managing creation, updating, deletion, and retrieval. With Merge Sort for sorting and Binary Search for quick retrieval, it ensures fast and accurate access to course data. The system also manages essential course relationships like prerequisites and progression, making it an effective tool for organizing and maintaining academic information.

THANK YOU