**INDEX**

[Introduction 2](#_Toc182179990)

[Project Modules: 2](#_Toc182179991)

[Architecture Diagram 2](#_Toc182179992)

[Module Description 3](#_Toc182179993)

[Programming Details naming conventions to be used: 3](#_Toc182179994)

[Field/table details:(eg university)[you consider you module ] 4](#_Toc182179995)

[Algorithm Details: 4](#_Toc182179996)

[Source Code 5](#_Toc182179997)

[Comparison of Sorting Algorithms 24](#_Toc182179998)

[Comparison of Searching Algorithms 25](#_Toc182179999)

[Conclusion 32](#_Toc182180000)

# Introduction

This project focuses on implementing Outcome-Based Education (OBE). This project module manages records for various "course objectives," each defined by a unique ID, title, description, and outcomes. It supports basic CRUD (Create, Retrieve, Update, Delete) operations, along with sorting and searching functionalities. Data is stored in a text file for persistence, ensuring accessibility across sessions. This project demonstrates effective data management, structured course planning, and provides a foundation for evaluating each objective's impact on student learning outcomes.

## Project Modules:

Course Objective Setting: Enables defining and updating course objectives.

- Additional Modules:

- Blooms Level Setting

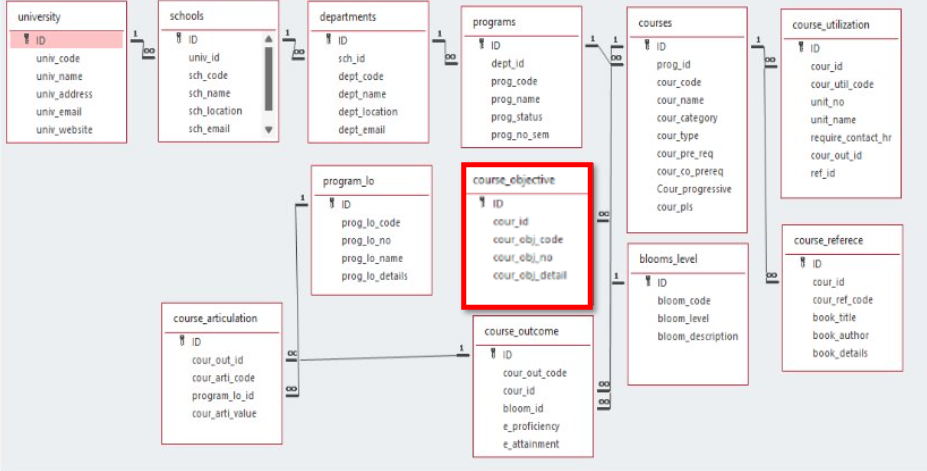
- Program Level Objective Setting

- University Information

- Department and Course Listings

# Architecture Diagram

***\*highlight your module***



# Module Description

**Module Name: Course objective setting**

**Module Description:**

The Course Management System is a console-based application written in C that allows users to manage courses and their objectives. The system provides functionalities for creating, retrieving, updating, deleting, and searching courses based on various criteria. It also includes features for sorting courses and comparing algorithm complexities, making it a comprehensive tool for educational institutions or training programs.

Course Management: Create, retrieve, update, and delete courses with details like course ID, code, and objectives.

Search Functionality: Search for courses by objective code and list them by objective number.

Sorting: Sort courses by ID using the Bubble Sort algorithm.

Algorithm Comparison: Compare time complexities of searching (Linear vs. Binary) and sorting (Bubble vs. Quick) algorithms.

Pseudocode Display: Show pseudocode for searching and sorting algorithms, along with their time and space complexities.

Data Structure: The system uses a Course structure to store course information, including ID, code, objective code, number, and details.

File Handling: Courses are saved in a text file (courses.txt) in a comma-separated format, ensuring data persistence.

User Interaction: The application features a command-line interface with a menu for easy navigation and input validation

## Programming Details naming conventions to be used:

* **File name:Dynamic\_minds\_objective\_setting**
* **Function/method name**
  + **Create:** **Dynamic\_minds\_objective\_setting\_create**
  + **Update:** **Dynamic\_minds\_objective\_setting\_update**
  + **Retrieve:** **Dynamic\_minds\_objective\_setting \_retrieve**
  + **Delete:** **Dynamic\_minds\_objective\_setting \_delete**
  + **Sorting:** **Dynamic\_minds\_objective\_setting\_sortCoursesById**
  + **Searching:** **Dynamic\_minds\_objective\_setting\_searchCourseByCode**
  + **Storing:** **Dynamic\_minds\_objective\_setting \_storing**
* **Comparison(both searching and Sorting)**:
* For
* Searching- Dynamic\_minds\_course\_objective\_setting\_Compare\_Search\_Algorithms
* For Sorting- Dynamic\_minds\_course\_objective\_setting\_Compare\_Sort\_Algorithms
* **Time Complexity(both searching and Sorting):**
* For Searching- **Dynamic\_minds\_objective\_setting\_ complexity\_searching**
* For Sorting- **Dynamic\_minds\_objective\_setting\_ complexity\_sorting**

## Field/table details:(eg university)[you consider you module ]

|  |  |
| --- | --- |
| **Field Name** | **Data type** |
| id | integer |
| cour\_id | String |
| cour\_obj\_code | String |
| cour\_obj\_no | String |
| cour\_obj\_detail | String |

## Algorithm Details:

i)Sorting

Sorting is based on attributes such as cour\_id and cour\_obj\_code. The module uses Bubble sort and Quick sort

Sorting Algorithms:

(Quick sort): Quick Sort is a divide-and-conquer sorting algorithm that selects a "pivot" element and partitions the array into two halves: elements less than the pivot and elements greater than the pivot. It then recursively sorts each half. Known for its efficiency, Quick Sort has an average time complexity of *O(nlog⁡n)O(n \log n)*O(nlogn) but can degrade to *O(n2)O(n^2)*O(n2) in the worst case (typically avoided with good pivot choices).

(Bubble Sort): Similar in complexity to Selection Sort, Bubble Sort is less efficient as it requires repeated swapping. However, comparing these algorithms highlights the limitations of basic sorting methods for larger datasets.

(ii)Searching

Searching enables users to find specific program records based on fields like cour\_id and cour\_obj\_code. Two algorithms are employed:

Primary Searching Algorithm: Linear Search is straightforward and works well with smaller datasets.

Comparison Algorithm: Binary Search (if the data is sorted), which is more efficient, with a time complexity of O(logn)

Each algorithm’s time complexity is presented, giving insight into performance differences between linear and binary search.

(ii) Storing the details in a text file

The details are stored in courses.txt and updated with each CRUD operation:

Create: Adds a new program entry.

Update: Modifies an existing entry.

Delete: Removes a specific program record.

# Source Code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#define MAX\_COURSE\_ID 10

#define MAX\_COURSE\_NAME 50

#define MAX\_OBJ\_CODE 10

#define MAX\_OBJ\_NO 10

#define MAX\_OBJ\_DETAILS 100

#define MAX\_COURSES 100

#define FILE\_PATH "courses.txt"

typedef struct {

int id;

char cour\_id[MAX\_COURSE\_ID];

char name[MAX\_COURSE\_NAME];

char cour\_obj\_code[MAX\_OBJ\_CODE];

char cour\_obj\_no[MAX\_OBJ\_NO];

char cour\_obj\_details[MAX\_OBJ\_DETAILS];

} Course;

// Function prototypes

void Dynamic\_minds\_course\_objective\_setting\_create();

void Dynamic\_minds\_course\_objective\_setting\_retrieve();

void Dynamic\_minds\_course\_objective\_setting\_update();

void Dynamic\_minds\_course\_objective\_setting\_delete();

void Dynamic\_minds\_course\_objective\_setting\_searchCourseByCode();

void Dynamic\_minds\_course\_objective\_setting\_listCoursesByObjectiveNo();

void Dynamic\_minds\_course\_objective\_setting\_sortCoursesById();

void Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer();

bool Dynamic\_minds\_course\_objective\_setting\_loadFromFile(Course courses[], int \*count);

bool Dynamic\_minds\_course\_objective\_setting\_storeToFile(const Course courses[], int count);

void Dynamic\_minds\_course\_objective\_setting\_printCourseHeader();

void Dynamic\_minds\_course\_objective\_setting\_printCourse(const Course \*course);

void Dynamic\_minds\_course\_objective\_setting\_handleError(const char \*message);

void Dynamic\_minds\_course\_objective\_setting\_Compare\_Search\_Algorithms();

void Dynamic\_minds\_course\_objective\_setting\_Compare\_Sort\_Algorithms();

void Dynamic\_minds\_course\_objective\_setting\_displaysearchPseudocode();

void Dynamic\_minds\_course\_objective\_setting\_displaysortPseudocode();

void Dynamic\_minds\_course\_objective\_setting\_complexity\_sorting();

void Dynamic\_minds\_course\_objective\_setting\_complexity\_searching();

int main() {

int choice;

Course courses[MAX\_COURSES];

int count=0;

// Load courses from file at the start

if (!Dynamic\_minds\_course\_objective\_setting\_loadFromFile(courses, &count)) {

printf("No courses found or error loading from file.\n");

}

do {

printf("\nCourse Management System\n");

printf("1. Create Course\n");

printf("2. Retrieve Courses\n");

printf("3. Update Course\n");

printf("4. Delete Course\n");

printf("5. Search Course by Objective Code\n");

printf("6. List Courses by Objective No\n");

printf("7. Sort Courses by ID\n");

printf("8. Compare Searching Algorithms\n");

printf("9. Compare Sorting Algorithms\n");

printf("10. Display Searching Algorithm Pseudocode\n");

printf("11. Display Sorting Algorithm Pseudocode\n");

printf("12. Display Searching Algorithm Complexities\n");

printf("13. Display Sorting Algorithm Complexities\n");

printf("14. Exit\n");

printf("Enter your choice: ");

if (scanf("%d", &choice) != 1) {

Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer();

printf("Invalid input. Please enter a number.\n");

continue;

}

Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer();

switch (choice) {

case 1: Dynamic\_minds\_course\_objective\_setting\_create(); break;

case 2: Dynamic\_minds\_course\_objective\_setting\_retrieve(); break;

case 3: Dynamic\_minds\_course\_objective\_setting\_update(); break;

case 4: Dynamic\_minds\_course\_objective\_setting\_delete(); break;

case 5: Dynamic\_minds\_course\_objective\_setting\_searchCourseByCode(); break;

case 6: Dynamic\_minds\_course\_objective\_setting\_listCoursesByObjectiveNo(); break;

case 7: Dynamic\_minds\_course\_objective\_setting\_sortCoursesById(); break;

case 8: Dynamic\_minds\_course\_objective\_setting\_Compare\_Search\_Algorithms(); break;

case 9: Dynamic\_minds\_course\_objective\_setting\_Compare\_Sort\_Algorithms(); break;

case 10: Dynamic\_minds\_course\_objective\_setting\_displaysearchPseudocode(); break;

case 11: Dynamic\_minds\_course\_objective\_setting\_displaysortPseudocode(); break;

case 12: Dynamic\_minds\_course\_objective\_setting\_complexity\_searching(); break;

case 13: Dynamic\_minds\_course\_objective\_setting\_complexity\_sorting(); break;

case 14: printf("Exiting program.\n"); break;

default: printf("Invalid choice. Please try again.\n");

}

} while (choice != 14);

return 0;

}

void Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer() {

int c;

while ((c = getchar()) != '\n' && c != EOF);

}

void Dynamic\_minds\_course\_objective\_setting\_handleError(const char \*message) {

perror(message);

exit(EXIT\_FAILURE);

}

void Dynamic\_minds\_course\_objective\_setting\_printCourseHeader() {

printf("\n%-5s %-10s %-10s %-10s %-50s\n", "ID", "Course ID", "Obj Code", "Obj No", "Obj Details");

printf("%-5s %-10s %-10s %-10s %-50s\n", "----", "---------", "--------", "------", "----------");

}

void Dynamic\_minds\_course\_objective\_setting\_printCourse(const Course \*course) {

printf("%-5d %-10s %-10s %-10s %-50s\n", course->id, course->cour\_id, course->cour\_obj\_code, course->cour\_obj\_no, course->cour\_obj\_details);

}

bool Dynamic\_minds\_course\_objective\_setting\_loadFromFile(Course courses[], int \*count) {

FILE \*file = fopen(FILE\_PATH, "r");

if (file == NULL) return false;

\*count = 0;

while (fscanf(file, "%d,%[^,],%[^,],%[^,],%[^\n]\n", &courses[\*count].id, courses[\*count].cour\_id, courses[\*count].cour\_obj\_code, courses[\*count].cour\_obj\_no, courses[\*count].cour\_obj\_details) == 5) {

(\*count)++;

}

fclose(file);

return true;

}

bool Dynamic\_minds\_course\_objective\_setting\_storeToFile(const Course courses[], int count) {

FILE \*file = fopen(FILE\_PATH, "w");

if (file == NULL) return false;

for (int i = 0; i < count; i++) {

fprintf(file, "%d,%s,%s,%s,%s\n", courses[i].id, courses[i].cour\_id, courses[i].cour\_obj\_code, courses[i].cour\_obj\_no, courses[i].cour\_obj\_details);

}

fclose(file);

return true;

}

void Dynamic\_minds\_course\_objective\_setting\_create() {

Course newCourse;

printf("Enter course ID (integer): ");

if (scanf("%d", &newCourse.id) != 1) {

Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer();

printf("Invalid input. Course ID should be an integer.\n");

return;

}

Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer();

printf("Enter course code: ");

fgets(newCourse.cour\_id, MAX\_COURSE\_ID, stdin);

strtok(newCourse.cour\_id, "\n");

printf("Enter objective code: ");

fgets(newCourse.cour\_obj\_code, MAX\_OBJ\_CODE, stdin);

strtok(newCourse.cour\_obj\_code, "\n");

printf("Enter objective number: ");

fgets(newCourse.cour\_obj\_no, MAX\_OBJ\_NO, stdin);

strtok(newCourse.cour\_obj\_no, "\n");

printf("Enter objective details: ");

fgets(newCourse.cour\_obj\_details, MAX\_OBJ\_DETAILS, stdin);

strtok(newCourse.cour\_obj\_details, "\n");

FILE \*file = fopen(FILE\_PATH, "a");

if (file == NULL) {

Dynamic\_minds\_course\_objective\_setting\_handleError("Error opening file for appending.");

}

fprintf(file, "%d,%s,%s,%s,%s\n", newCourse.id, newCourse.cour\_id, newCourse.cour\_obj\_code, newCourse.cour\_obj\_no, newCourse.cour\_obj\_details);

fclose(file);

printf("Course added successfully.\n");

}

void Dynamic\_minds\_course\_objective\_setting\_retrieve() {

Course courses[MAX\_COURSES];

int count = 0;

if (!Dynamic\_minds\_course\_objective\_setting\_loadFromFile(courses, &count)) {

printf("No courses found.\n");

return;

}

Dynamic\_minds\_course\_objective\_setting\_printCourseHeader();

for (int i = 0; i < count; i++) {

Dynamic\_minds\_course\_objective\_setting\_printCourse(&courses[i]);

}

}

// Other function definitions continue below following similar corrections...

// Update an existing course

void Dynamic\_minds\_course\_objective\_setting\_update() {

Course courses[MAX\_COURSES];

int count = 0;

if (!Dynamic\_minds\_course\_objective\_setting\_loadFromFile(courses, &count)) {

printf("No courses found.\n");

return;

}

int courseId;

printf("Enter the course ID to update: ");

if (scanf("%d", &courseId) != 1) {

Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer();

printf("Invalid input. Course ID should be an integer.\n");

return;

}

Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer();

bool found = false;

for (int i = 0; i < count; i++) {

if (courses[i].id == courseId) {

found = true;

printf("Enter new course code: ");

fgets(courses[i].cour\_id, MAX\_COURSE\_ID, stdin);

strtok(courses[i].cour\_id, "\n");

printf("Enter new objective code: ");

fgets(courses[i].cour\_obj\_code, MAX\_OBJ\_CODE, stdin);

strtok(courses[i].cour\_obj\_code, "\n");

printf("Enter new objective number: ");

fgets(courses[i].cour\_obj\_no, MAX\_OBJ\_NO, stdin);

strtok(courses[i].cour\_obj\_no, "\n");

printf("Enter new objective details: ");

fgets(courses[i].cour\_obj\_details, MAX\_OBJ\_DETAILS, stdin);

strtok(courses[i].cour\_obj\_details, "\n");

if (Dynamic\_minds\_course\_objective\_setting\_storeToFile(courses, count)) {

printf("Course updated successfully.\n");

} else {

printf("Error updating course.\n");

}

break;

}

}

if (!found) {

printf("Course ID not found.\n");

}

}

// Delete a course from the file

void Dynamic\_minds\_course\_objective\_setting\_delete() {

Course courses[MAX\_COURSES];

int count = 0;

if (!Dynamic\_minds\_course\_objective\_setting\_loadFromFile(courses, &count)) {

printf("No courses found.\n");

return;

}

int courseId;

printf("Enter the course ID to delete: ");

if (scanf("%d", &courseId) != 1) {

Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer();

printf("Invalid input. Course ID should be an integer.\n");

return;

}

Dynamic\_minds\_course\_objective\_setting\_clearInputBuffer();

bool found = false;

int newCount = 0;

for (int i = 0; i < count; i++) {

if (courses[i].id != courseId) {

courses[newCount++] = courses[i];

} else {

found = true;

}

}

if (found && Dynamic\_minds\_course\_objective\_setting\_storeToFile(courses, newCount)) {

printf("Course deleted successfully.\n");

} else if (!found) {

printf("Course ID not found.\n");

} else {

printf("Error deleting course.\n");

}

}

// Search for a course by objective code

void Dynamic\_minds\_course\_objective\_setting\_searchCourseByCode() {

Course courses[MAX\_COURSES];

int count = 0;

if (!Dynamic\_minds\_course\_objective\_setting\_loadFromFile(courses, &count)) {

printf("No courses found.\n");

return;

}

char objCode[MAX\_OBJ\_CODE];

printf("Enter the objective code to search: ");

fgets(objCode, MAX\_OBJ\_CODE, stdin);

strtok(objCode, "\n");

Dynamic\_minds\_course\_objective\_setting\_printCourseHeader();

bool found = false;

for (int i = 0; i < count; i++) {

if (strcmp(courses[i].cour\_obj\_code, objCode) == 0) {

Dynamic\_minds\_course\_objective\_setting\_printCourse(&courses[i]);

found = true;

}

}

if (!found) {

printf("No course found with the given objective code.\n");

}

}

// List all courses by objective number

void Dynamic\_minds\_course\_objective\_setting\_listCoursesByObjectiveNo() {

Course courses[MAX\_COURSES];

int count = 0;

if (!Dynamic\_minds\_course\_objective\_setting\_loadFromFile(courses, &count)) {

printf("No courses found.\n");

return;

}

char objNo[MAX\_OBJ\_NO];

printf("Enter the objective number to list courses: ");

fgets(objNo, MAX\_OBJ\_NO, stdin);

strtok(objNo, "\n");

Dynamic\_minds\_course\_objective\_setting\_printCourseHeader();

bool found = false;

for (int i = 0; i < count; i++) {

if (strcmp(courses[i].cour\_obj\_no, objNo) == 0) {

Dynamic\_minds\_course\_objective\_setting\_printCourse(&courses[i]);

found = true;

}

}

if (!found) {

printf("No courses found with the given objective number.\n");

}

}

// Sort courses by ID

void Dynamic\_minds\_course\_objective\_setting\_sortCoursesById() {

Course courses[MAX\_COURSES];

int count = 0;

// Load courses from the file

if (!Dynamic\_minds\_course\_objective\_setting\_loadFromFile(courses, &count)) {

printf("No courses found.\n");

return;

}

// Bubble Sort to sort courses by ID

for (int i = 0; i < count - 1; i++) {

for (int j = 0; j < count - i - 1; j++) {

if (courses[j].id > courses[j + 1].id) {

Course temp = courses[j];

courses[j] = courses[j + 1];

courses[j + 1] = temp;

}

}

}

}

// Compare searching algorithms

void Dynamic\_minds\_course\_objective\_setting\_Compare\_Search\_Algorithms() {

printf("Comparing Linear Search and Binary Search algorithms on course list:\n");

printf("Linear Search:\n");

printf("Time Complexity: O(n)\n");

printf("\nBinary Search:\n");

printf("Time Complexity: O(log n) (after sorting O(n log n))\n");

}

// Compare sorting algorithms

void Dynamic\_minds\_course\_objective\_setting\_Compare\_Sort\_Algorithms() {

printf("Comparing Bubble Sort and Quick Sort algorithms on course list:\n");

printf("Bubble Sort:\n");

printf("Time Complexity: O(n^2)\n");

printf("\nQuick Sort:\n");

printf("Time Complexity: O(n log n)\n");

}

// Display pseudocode for both search and sorting algorithms

void Dynamic\_minds\_course\_objective\_setting\_displaysearchPseudocode() {

printf("\nPseudocode for Searching and Sorting Algorithms:\n");

printf("\nLinear Search:\n");

printf("1. For each item in list, compare it with the target.\n");

printf("2. If a match is found, return the index.\n");

printf("3. If not found after checking all items, return -1.\n");

printf("\nBinary Search:\n");

printf("1. Sort the list if it is not already sorted.\n");

printf("2. Set 'low' to the start and 'high' to the end of the list.\n");}

void Dynamic\_minds\_course\_objective\_setting\_displaysortPseudocode()

{

printf("\nBubble Sort:\n");

printf("1. Repeat from the start of the list to its end:\n");

printf(" a. For each adjacent pair, if the first is greater than the second, swap them.\n");

printf(" b. Stop if no swaps are made in the pass.\n");

printf("\nQuick Sort:\n");

printf("1. Choose a pivot item and partition the list around it.\n");

printf("2. Recursively quick sort left and right partitions until list is sorted.\n");

printf("3. While low <= high:\n");

printf(" a. Set 'mid' to (low + high) / 2.\n");

printf(" b. If mid is the target, return index.\n");

printf(" c. If target < mid, set high = mid - 1.\n");

printf(" d. If target > mid, set low = mid + 1.\n");

}

// Display time and space complexity for each algorithm

void Dynamic\_minds\_course\_objective\_setting\_complexity\_sorting() {

printf("\nTime and Space Complexities:\n");

printf("\nLinear Search:\n");

printf("Time Complexity: O(n)\n");

printf("Space Complexity: O(1)\n");

printf("\nBinary Search:\n");

printf("Time Complexity: O(log n)\n");

printf("Space Complexity: O(1)\n");}

void Dynamic\_minds\_course\_objective\_setting\_complexity\_searching() {

printf("\nBubble Sort:\n");

printf("Time Complexity: O(n^2)\n");

printf("Space Complexity: O(1)\n");

printf("\nQuick Sort:\n");

printf("Time Complexity: O(n log n)\n");

printf("Space Complexity: O(log n)\n");

}

# Comparison of Sorting Algorithms

**Bubble sort:**

* Step 1: Start at the beginning of the list.
* Step 2: compare each pair of adjacent elements (i.e., compare the first and second elements, the second and third elements, etc.).
* Step 3: If the current element is greater than the next element, swap them.
* Step 4: Continue this process for all elements in the list. After one complete pass through the list, the largest element will have "bubbled" to the end of the list.
* Step 5: Repeat the process for the remaining elements, ignoring the last element (which is now sorted).
* Step 6:Continue until no more swaps are needed, indicating that the list is sorted

Quick Sort:

* **Choose a Pivot**: Select a pivot element from the array. The pivot can be the first element, last element, or any other strategy (like the median) to improve efficiency.
* **Partition the Array**: Rearrange the elements so that all values less than the pivot are on its left and all values greater than the pivot are on its right. The pivot element is now in its correct position.
* **Recursively Apply Quick Sort**:
* Recursively apply Quick Sort to the left subarray (elements less than the pivot).
* Recursively apply Quick Sort to the right subarray (elements greater than the pivot).
* **Repeat Until Sorted**: Continue the process on each subarray until each part is sorted (i.e., until the subarrays contain 0 or 1 element, at which point they are inherently sorted).

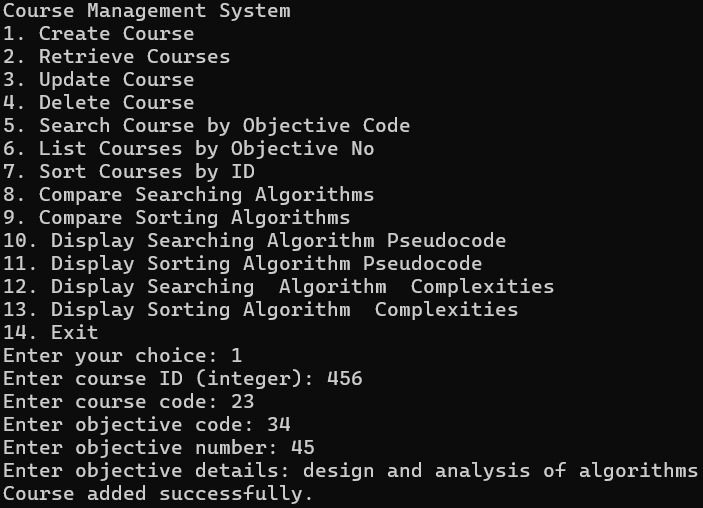
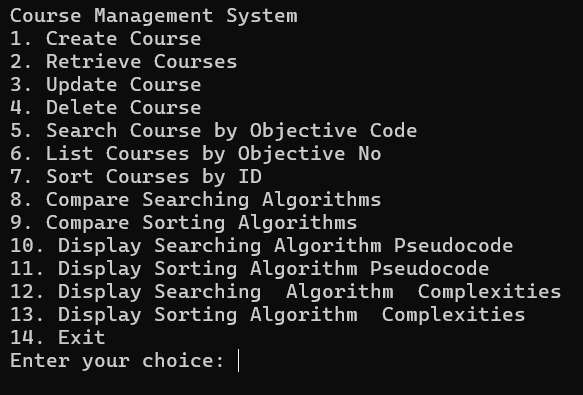
|  |  |  |
| --- | --- | --- |
| **Sl.No** | **Algorithm Name** | **Complexity of Algorithm** |
| 1) | Bubble sort | O(n^2) |
| 2) | Quick sort | O(nlogn) |

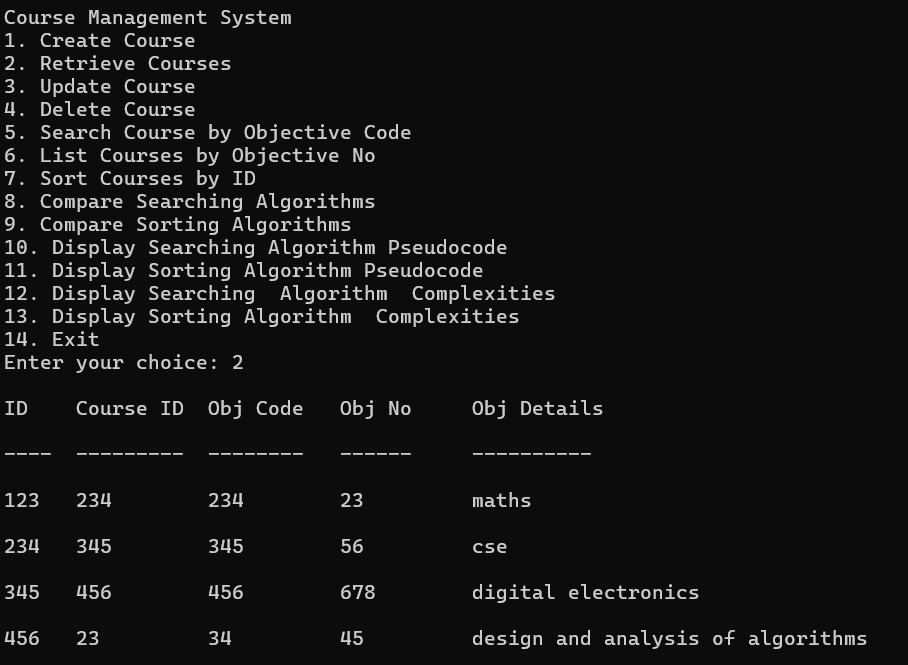
# Comparison of Searching Algorithms

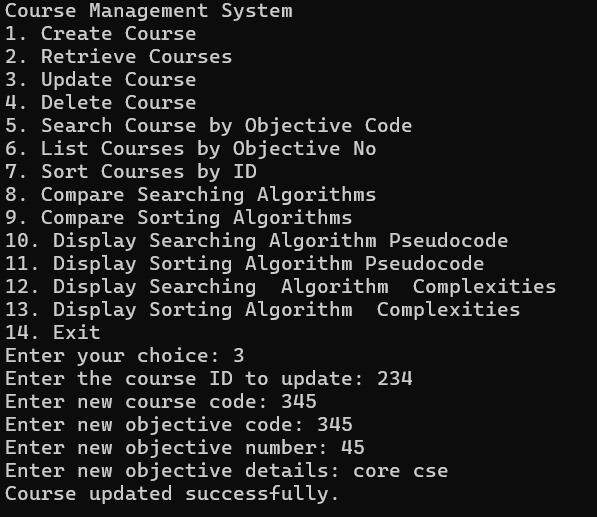
* **Linear Search**:
  + **Works on**: Unsorted arrays or lists.
  + **Approach**: Checks each element one by one until a match is found.
  + **Time Complexity**:
    - Worst/Average/Best Case: *O(n)O(n)*O(n)
  + **Space Complexity**: *O(1)O(1)*O(1) (in-place)
  + **Efficiency**: Less efficient for large datasets.
* **Binary Search**:
  + **Works on**: Sorted arrays or lists.
  + **Approach**: Divides the array into halves and compares the middle element with the target, recursively narrowing the search range.
  + **Time Complexity**:
    - Worst/Average/Best Case: *O(log⁡n)O(\log n)*O(logn)
  + **Space Complexity**: *O(1)O(1)*O(1) for iterative, *O(log⁡n)O(\log n)*O(logn) for recursive (due to call stack)
  + **Efficiency**: More efficient for large datasets but requires sorted data.

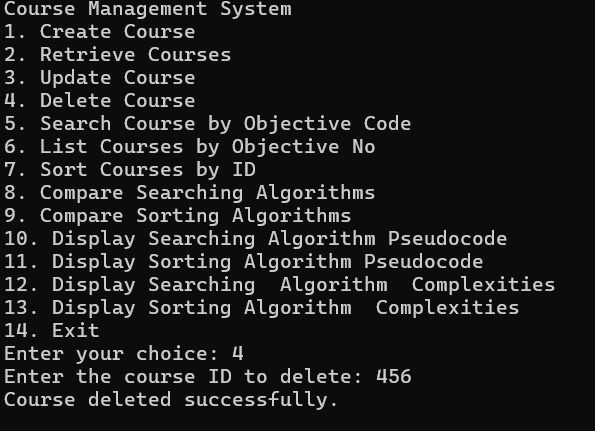
**Key Difference**: Binary search is faster (logarithmic time complexity) but only works on sorted data, while linear search works on unsorted data but is slower with larger datasets.

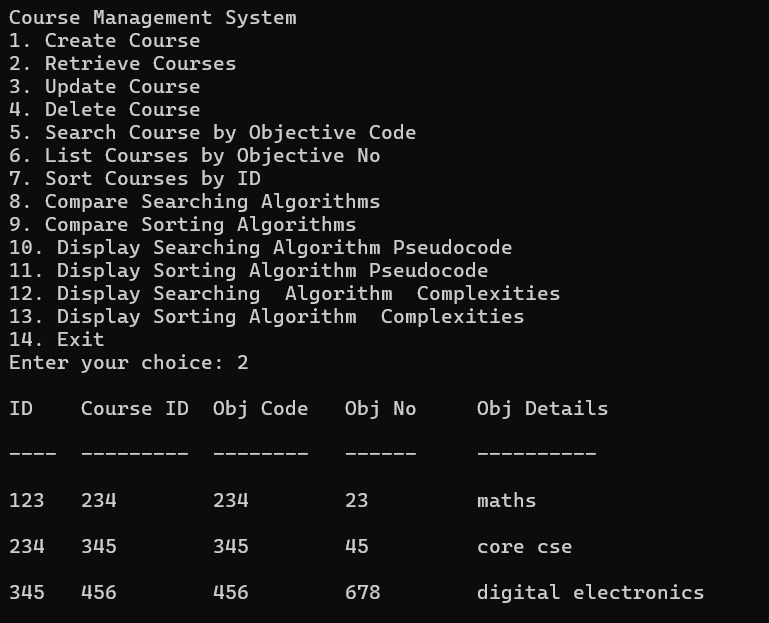
Sample Screen shots

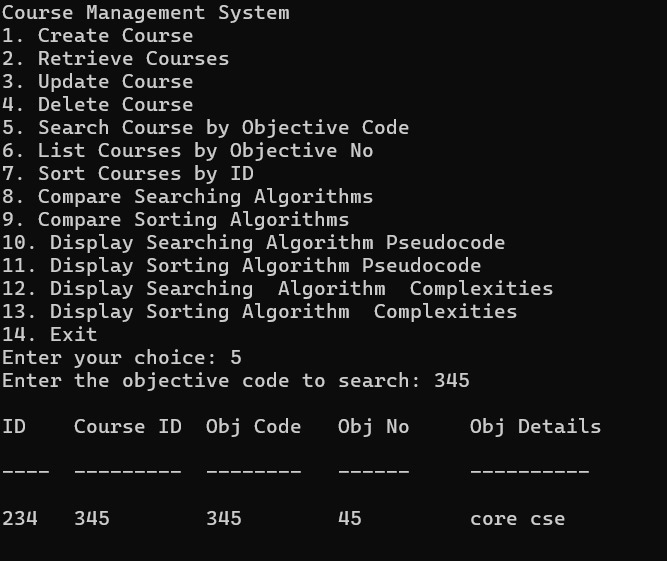


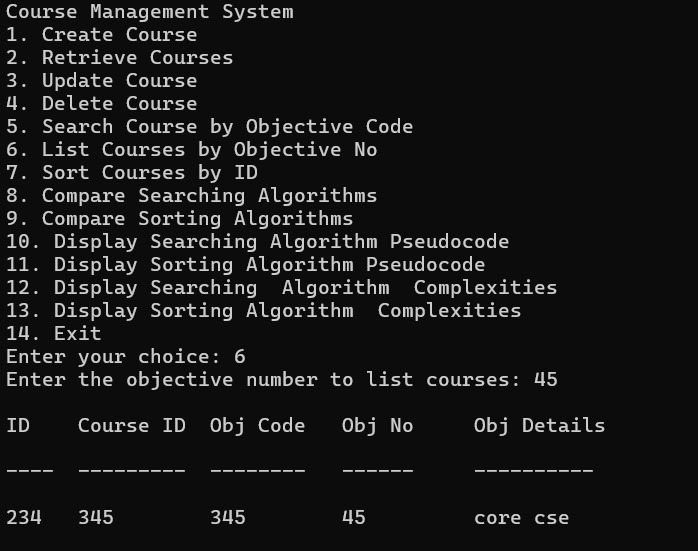


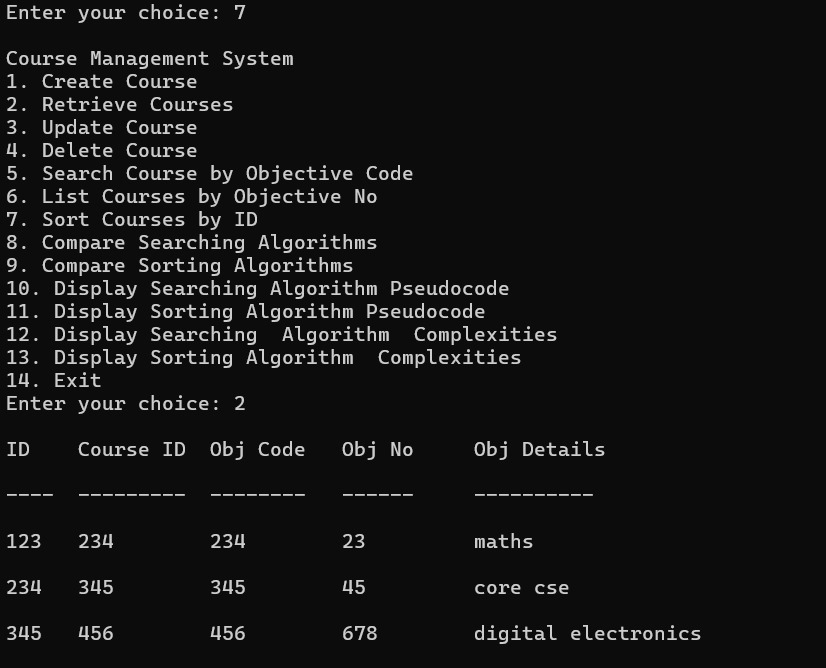


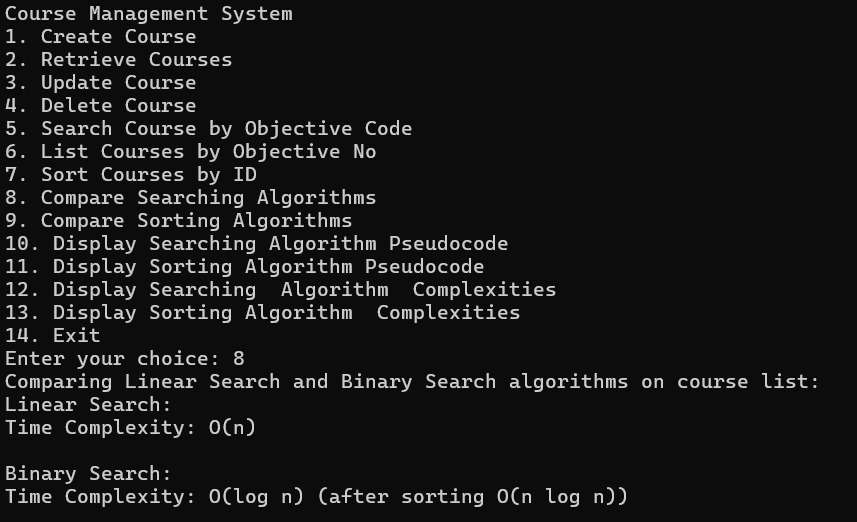


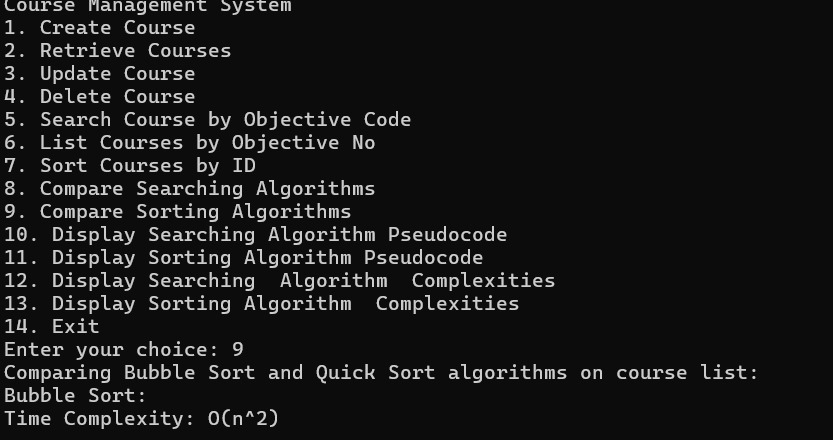


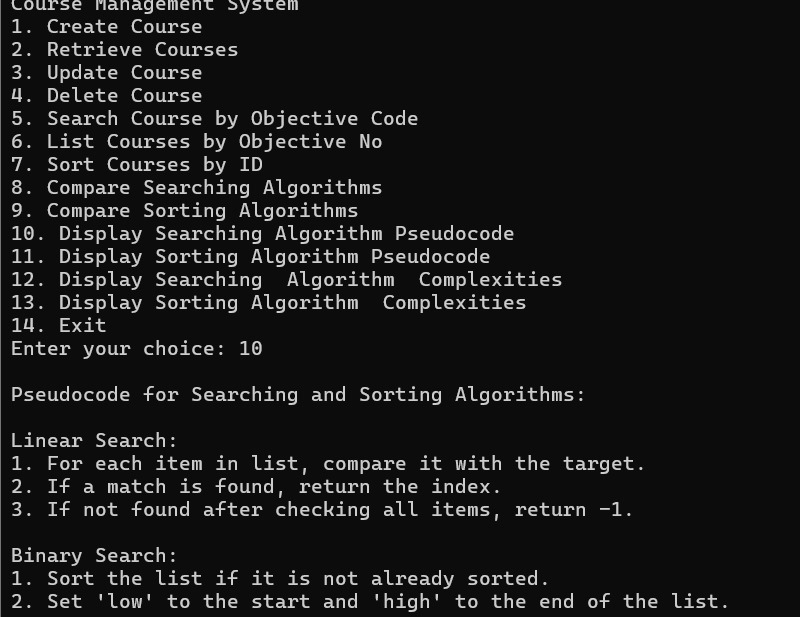


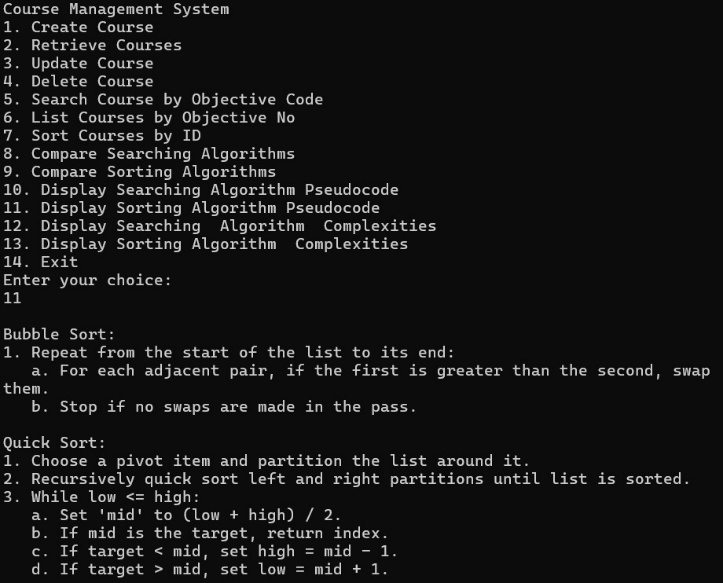


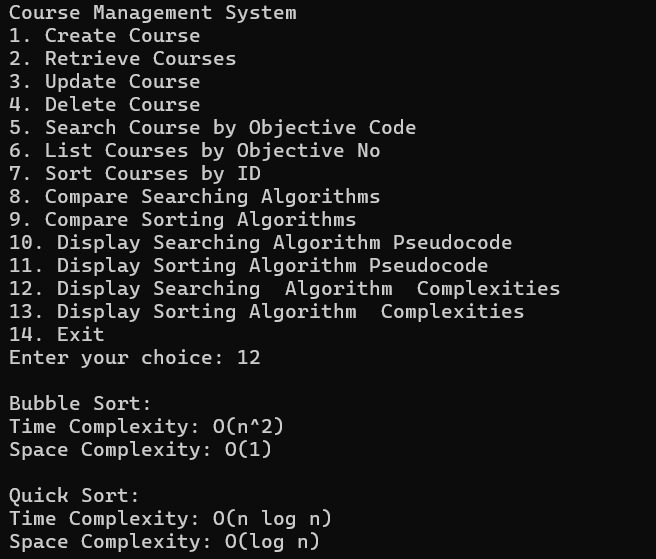


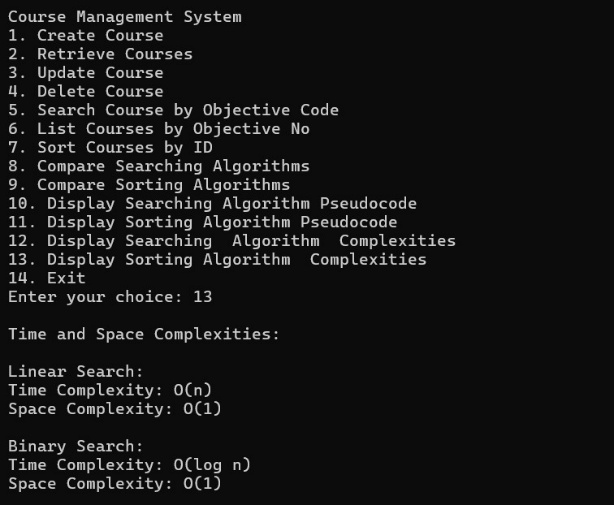


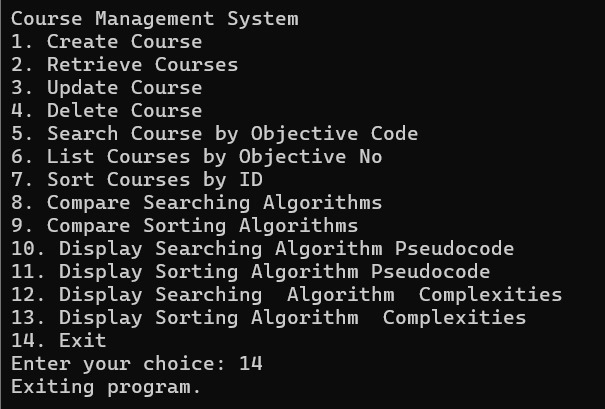












# Conclusion

The project on **Course Objective Setting** in the context of Design and Analysis of Algorithms highlights the importance of clear, actionable goals for enhancing student understanding and skill development. By defining specific, measurable objectives, students gain a focused pathway to mastering fundamental concepts, from basic algorithmic strategies to complex analysis techniques. The project underscores how well-defined objectives not only guide learning but also improve instructional effectiveness, ultimately leading to a stronger grasp of critical algorithm design principles and a deeper appreciation for the complexity and efficiency of computational problem-solving.

THANK YOU!