**OBE IMPLEMENTATION :COURSE OUTCOME**

**by**  
**Praneeth Reddy[AP23110011164]  
Lohith Kudeti[AP23110011152]  
Jayakrishna Panditi[AP23110011205]  
Bharat Bitra[AP23110010717]**

**Subhash Chembeti[AP23110011160]**A report for the CS204:Design and Analysis of Algorithmproject



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**SRM UNIVERSITY AP::AMARAVATI**

**INDEX**

**Introduction**

**Project Module**

**Architecture Diagram  
Module Description**Programming Details naming conventions to be used:

Field/Table details: Course Outcome

Algorithm Details:

(i)Sorting  
(ii)Searching  
(iii)Storing the details in a text file

**Source Code**  
OBE\_MAIN\_Team\_Unity.c  
**Conclusion**

**Introduction**

This project module manages records for "Course Outcomes," where each outcome is defined by attributes such as a unique outcome code, course ID, Bloom's Taxonomy level, expected proficiency, and attainment levels. The application supports fundamental CRUD (Create, Retrieve, Update, Delete) operations, allowing users to manage course outcome records effectively. Sorting and searching functionalities provide quick access and organization of outcomes, with data saved in a text file (course\_outcome.txt) for persistence, ensuring outcomes remain accessible across sessions. This project illustrates practical data management, basic algorithm comparisons, and efficiency evaluation in sorting and searching operations.

**Project Module:**

**Course Outcome Management**

The primary module in this project is Course Outcome Management, focused on handling records for various course outcomes.

**This module allows:**

**Outcome Code-based Identification**: Each course outcome is uniquely identified by its outcome code.

**Attribute-based Searching and Sorting**: Course outcomes can be searched and organized by attributes like course ID and Bloom's Taxonomy level, enabling efficient access and management.

**CRUD Operations:** Users can create, retrieve, update, and delete course outcome records with ease, facilitating effective outcome management.

The Course Outcome Management module utilizes Bubble Sort and Quick Sort for sorting, and Linear Search and Binary Search for searching, offering options for both simple and efficient record management.

**Architecture Diagram**

**A diagram of a program

Description automatically generated**

**Module Description**

**Module Name:**Course Outcome  
**Module Description:**  
 This Course Outcome Management System module is designed to perform operations on course outcomes data, which include creating, updating, retrieving, and deleting records (commonly referred to as CRUD operations). This module serves as a complete solution for managing course outcome data, with flexible options for CRUD operations, sorting, and searching, ensuring both efficiency and ease of use for users.

**Programming Details naming conventions to be used:**

* **File name:Team\_unity\_course\_outcome**
* **Function/method name**
  + **Create:** Team\_unity\_create
  + **Update:**Team\_unity\_update
  + **Retrieve:**Team\_unity\_retrive
  + **Delete:**Team\_unity\_delete
  + **Sorting**:(i)Team\_unity\_bubble\_sort
  + **(**ii)Team\_unity\_quick\_sort
  + **Searching**:(i)Team\_unity\_linear\_search
  + (ii)Team\_unity\_binary\_search
  + **Storing:**Team\_unity\_storing
  + **Comparison(both searching and Sorting):**
    - **For Searching**-(i)Team\_unity\_Compare\_Search\_linear

**(**ii)Team\_unity\_Compare\_Search\_binary

* + - **For Sorting-**(i)Team\_unity\_Compare\_sorting\_bubble

(ii)Team\_unity\_Compare\_sorting\_quick

* + **Time Complexity(both searching and Sorting):**
    - **For Searching-**Team\_unity\_complexity\_Search
    - **For Sorting-**Team\_unity\_complexity\_sorting
  + **Algorithm Details(pseudocode or steps)(both searching and Sorting):**
    - **For Searching-**Team\_unity\_Search\_algorithm\_details
    - **For Sorting-**Team\_unity\_sort\_algorithm\_details
  + **File name(for storing the details)**
    - **File name to be used is:-**course\_outcome.txt

|  |  |
| --- | --- |
| **Field Name** | **Data Type** |
| **cour\_out\_code** | **string** |
| **cour\_id** | **string** |
| **bloom\_id** | **string** |
| **e\_proficiency** | **float** |
| **e\_attainment** | **float** |

**Course Outcome:Field/Table Details**

**Algorithm details:**

**(i) Sorting**

The module provides two sorting options for sorting the course outcomes based on the cour\_out\_code. Users can choose between Bubble Sort and Quick Sort to compare their efficiencies.

**Primary Sorting Algorithm (Bubble Sort):**

This algorithm repeatedly steps through the list, compares adjacent items, and swaps them if they are in the wrong order.While straightforward, Bubble Sort is inefficient for larger datasets due to its O(n^2) complexity, as it involves multiple passes through the data.

**Comparison Algorithm (Quick Sort):**

Quick Sort is an efficient, divide-and-conquer algorithm that sorts data by partitioning around a pivot element, then recursively sorting the two halves.Quick Sort has an average time complexity of O(n log n), making it much faster for larger datasets than Bubble Sort.

**(ii) Searching**

The module supports two search methods to locate specific course outcomes by cour\_out\_code.

**Primary Searching Algorithm (Linear Search):**

Linear Search: Sequentially checks each item, making it O(n) and suitable for unsorted or small datasets.

**Comparison Algorithm (Binary Search):**

Binary Search: Requires sorted data and uses a divide-and-conquer approach, making it faster with O(log n) time complexity.   
  
**(iii) Storing Course Outcomes in a Text File**

CRUD Operations on File (course\_outcome.txt):

Create: Adds a new entry.

Update/Delete: Modifies or removes entries in memory and then saves changes to the file.

Store: Saves all current data from memory to the file.  
  
**Source Code**#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_OUTCOMES 100

#define FILENAME "course\_outcome.txt"

typedef struct {

char cour\_out\_code[20];

char cour\_id[20];

char bloom\_id[20];

float e\_proficiency;

float e\_attainment;

} CourseOutcome;

CourseOutcome outcomes[MAX\_OUTCOMES];

int outcome\_count = 0;

void Team\_unity\_create() {

if (outcome\_count >= MAX\_OUTCOMES) {

printf("Cannot add more outcomes. Maximum limit reached.\n");

return;

}

CourseOutcome new\_outcome;

printf("Enter Course Outcome Code: ");

scanf("%s", new\_outcome.cour\_out\_code);

printf("Enter Course ID: ");

scanf("%s", new\_outcome.cour\_id);

printf("Enter Bloom ID: ");

scanf("%s", new\_outcome.bloom\_id);

printf("Enter Proficiency Level: ");

scanf("%f", &new\_outcome.e\_proficiency);

printf("Enter Attainment Level: ");

scanf("%f", &new\_outcome.e\_attainment);

outcomes[outcome\_count++] = new\_outcome;

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

if (file) {

fprintf(file, "%s,%s,%s,%.2f,%.2f\n", new\_outcome.cour\_out\_code, new\_outcome.cour\_id, new\_outcome.bloom\_id, new\_outcome.e\_proficiency, new\_outcome.e\_attainment);

fclose(file);

} else {

printf("Error opening file for writing.\n");

}

printf("Course Outcome created successfully.\n");

return;

}

void Team\_unity\_update() {

char cour\_out\_code[20];

printf("Enter Course Outcome Code to update: ");

scanf("%s", cour\_out\_code);

for (int i = 0; i < outcome\_count; i++) {

if (strcmp(outcomes[i].cour\_out\_code, cour\_out\_code) == 0) {

printf("Updating Course Outcome:\n");

printf("Enter New Course ID: ");

scanf("%s", outcomes[i].cour\_id);

printf("Enter New Bloom ID: ");

scanf("%s", outcomes[i].bloom\_id);

printf("Enter New Proficiency Level: ");

scanf("%f", &outcomes[i].e\_proficiency);

printf("Enter New Attainment Level: ");

scanf("%f", &outcomes[i].e\_attainment);

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

return;

}

}

printf("Course Outcome with code %s not found.\n", cour\_out\_code);

}

void Team\_unity\_retrieve() {

printf("Retrieving Course Outcomes:\n");

printf("----------------------------------------------------------------------------------------------------\n");

printf("| %-20s | %-20s | %-20s | %-12s | %-12s |\n", "Outcome Code", "Course ID", "Bloom ID", "Proficiency", "Attainment");

printf("----------------------------------------------------------------------------------------------------\n");

for (int i = 0; i < outcome\_count; i++) {

printf("| %-20s | %-20s | %-20s | %-12.2f | %-12.2f |\n",

outcomes[i].cour\_out\_code, outcomes[i].cour\_id, outcomes[i].bloom\_id, outcomes[i].e\_proficiency, outcomes[i].e\_attainment);

}

printf("----------------------------------------------------------------------------------------------------\n");

}

void Team\_unity\_delete() {

char cour\_out\_code[20];

printf("Enter Course Outcome Code to delete: ");

scanf("%s", cour\_out\_code);

for (int i = 0; i < outcome\_count; i++) {

if (strcmp(outcomes[i].cour\_out\_code, cour\_out\_code) == 0) {

for (int j = i; j < outcome\_count - 1; j++) {

outcomes[j] = outcomes[j + 1]; // Shift elements left

}

outcome\_count--;

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

return;

}

}

printf("Course Outcome with code %s not found.\n", cour\_out\_code);

}

void Team\_unity\_storing() {

FILE \*file = fopen(FILENAME, "w"); // Open the file in write mode

if (file) {

for (int i = 0; i < outcome\_count; i++) {

fprintf(file, "%s,%s,%s,%.2f,%.2f\n",

outcomes[i].cour\_out\_code,

outcomes[i].cour\_id,

outcomes[i].bloom\_id,

outcomes[i].e\_proficiency,

outcomes[i].e\_attainment);

}

fclose(file);

printf("Outcomes stored successfully.\n");

} else {

printf("Error opening file for writing.\n");

}

}

void display\_outcomes() {

printf("--------------------------------------------------------------------------\n");

printf("| %-15s | %-10s | %-10s | %-12s | %-12s |\n",

"Outcome Code", "Course ID", "Bloom ID", "Proficiency", "Attainment");

printf("--------------------------------------------------------------------------\n");

for (int i = 0; i < outcome\_count; i++) {

printf("| %-15s | %-10s | %-10s | %-12.2f | %-12.2f |\n",

outcomes[i].cour\_out\_code, outcomes[i].cour\_id, outcomes[i].bloom\_id,

outcomes[i].e\_proficiency, outcomes[i].e\_attainment);

}

printf("--------------------------------------------------------------------------\n");

}

void load\_outcomes\_from\_file() {

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

if (file == NULL) {

perror("Error opening file.Starting with an empty record.");

return;

}

outcome\_count = 0;

while (fscanf(file, "%[^,],%[^,],%[^,],%f,%f\n",

outcomes[outcome\_count].cour\_out\_code,

outcomes[outcome\_count].cour\_id,

outcomes[outcome\_count].bloom\_id,

&outcomes[outcome\_count].e\_proficiency,

&outcomes[outcome\_count].e\_attainment) == 5) {

outcome\_count++;

}

fclose(file);

display\_outcomes();

}

void Team\_unity\_bubble\_sort() {

for (int i = 0; i < outcome\_count - 1; i++) {

for (int j = 0; j < outcome\_count - i - 1; j++) {

if (strcmp(outcomes[j].cour\_out\_code, outcomes[j + 1].cour\_out\_code) > 0) {

CourseOutcome temp = outcomes[j];

outcomes[j] = outcomes[j + 1];

outcomes[j + 1] = temp;

}

}

}

printf("Outcomes sorted using Bubble Sort.\n");

}

int Team\_unity\_partition(CourseOutcome arr[], int low, int high) {

CourseOutcome pivot = arr[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (strcmp(arr[j].cour\_out\_code, pivot.cour\_out\_code) < 0) {

i++;

CourseOutcome temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

CourseOutcome temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return (i + 1);

}

void Team\_unity\_quick\_sort(CourseOutcome arr[], int low, int high) {

if (low < high) {

int pi = Team\_unity\_partition(arr, low, high);

Team\_unity\_quick\_sort(arr, low, pi - 1);

Team\_unity\_quick\_sort(arr, pi + 1, high);

}

}

void Team\_unity\_Compare\_sorting\_bubble() {

Team\_unity\_bubble\_sort();

display\_outcomes();

}

void Team\_unity\_Compare\_sorting\_quick() {

Team\_unity\_quick\_sort(outcomes, 0, outcome\_count - 1);

display\_outcomes();

}

int Team\_unity\_linear\_search(char \*cour\_out\_code) {

for (int i = 0; i < outcome\_count; i++) {

if (strcmp(outcomes[i].cour\_out\_code, cour\_out\_code) == 0) {

return i;

}

}

return -1;

}

int Team\_unity\_binary\_search(char \*cour\_out\_code, int low, int high) {

while (low <= high) {

int mid = low + (high - low) / 2;

if (strcmp(outcomes[mid].cour\_out\_code, cour\_out\_code) == 0) {

return mid;

}

if (strcmp(outcomes[mid].cour\_out\_code, cour\_out\_code) < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1;

}

void Team\_unity\_Compare\_Search\_linear(char \*cour\_out\_code) {

int index = Team\_unity\_linear\_search(cour\_out\_code);

if (index != -1) {

printf("Course Outcome found using Linear Search:\n");

printf("--------------------------------------------------------------------------\n");

printf("| %-15s | %-10s | %-10s | %-12s | %-12s |\n",

"Outcome Code", "Course ID", "Bloom ID", "Proficiency", "Attainment");

printf("--------------------------------------------------------------------------\n");

printf("| %-15s | %-10s | %-10s | %-12.2f | %-12.2f |\n",

outcomes[index].cour\_out\_code, outcomes[index].cour\_id,

outcomes[index].bloom\_id, outcomes[index].e\_proficiency,

outcomes[index].e\_attainment);

printf("--------------------------------------------------------------------------\n");

} else {

printf("Course Outcome with code %s not found.\n", cour\_out\_code);

}

}

void Team\_unity\_Compare\_Search\_binary(char \*cour\_out\_code) {

Team\_unity\_quick\_sort(outcomes, 0, outcome\_count - 1);

int index = Team\_unity\_binary\_search(cour\_out\_code, 0, outcome\_count - 1);

if (index != -1) {

printf("Course Outcome found using Binary Search:\n");

printf("---------------------------------------------------------------------------\n");

printf("| %-15s | %-10s | %-10s | %-12s | %-12s |\n",

"Outcome Code", "Course ID", "Bloom ID", "Proficiency", "Attainment");

printf("---------------------------------------------------------------------------\n");

printf("| %-15s | %-10s | %-10s | %-12.2f | %-12.2f |\n",

outcomes[index].cour\_out\_code, outcomes[index].cour\_id,

outcomes[index].bloom\_id, outcomes[index].e\_proficiency,

outcomes[index].e\_attainment);

printf("---------------------------------------------------------------------------\n");

} else {

printf("Course Outcome with code %s not found.\n", cour\_out\_code);

}

}

void Team\_unity\_complexity\_Search() {

printf("Time Complexity for Linear Search: O(n)\n");

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

}

void Team\_unity\_complexity\_sorting() {

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

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

}

void Team\_unity\_Search\_algorithm\_details() {

printf("Linear Search Algorithm:\n");

printf("1. Start from the first element and compare it with the target.\n");

printf("2. If it matches, return the index.\n");

printf("3. If not, move to the next element and repeat.\n");

printf("4. If the end of the list is reached, return -1.\n\n");

printf("Binary Search Algorithm:\n");

printf("1. Ensure the array is sorted.\n");

printf("2. Set low and high pointers.\n");

printf("3. Calculate the mid index.\n");

printf("4. If the mid element matches the target, return the index.\n");

printf("5. If the target is less than the mid element, search the left half.\n");

printf("6. If the target is greater, search the right half.\n");

printf("7. Repeat until the target is found or the range is invalid.\n");

}

void Team\_unity\_sort\_algorithm\_details() {

printf("Bubble Sort Algorithm:\n");

printf("1. Compare adjacent elements and swap them if they are in the wrong order.\n");

printf("2. Repeat the process for each element in the array.\n");

printf("3. Continue until no swaps are needed, indicating the array is sorted.\n\n");

printf("Quick Sort Algorithm:\n");

printf("1. Choose a pivot element from the array.\n");

printf("2. Partition the array into two halves: elements less than the pivot and elements greater than the pivot.\n");

printf("3. Recursively apply the same process to the left and right halves.\n");

printf("4. Combine the sorted halves to get the final sorted array.\n");

}

int main() {

int choice;

char cour\_out\_code[20];

while (1) {

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

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

printf("2. Update Course Outcome\n");

printf("3. Retrieve Course Outcomes\n");

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

printf("5. Sort Outcomes (Bubble Sort)\n");

printf("6. Sort Outcomes (Quick Sort)\n");

printf("7. Search Outcome (Linear Search)\n");

printf("8. Search Outcome (Binary Search)\n");

printf("9. Show Time Complexity (Searching)\n");

printf("10. Show Time Complexity (Sorting)\n");

printf("11. Show Search Algorithm Details\n");

printf("12. Show Sort Algorithm Details\n");

printf("13. Store Outcomes\n");

printf("14. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

Team\_unity\_create();

break;

case 2:

Team\_unity\_update();

break;

case 3:

Team\_unity\_retrieve();

break;

case 4:

Team\_unity\_delete();

break;

case 5:

printf("Enter Course Outcome Code to sort: ");

scanf("%s", cour\_out\_code);

Team\_unity\_Compare\_sorting\_bubble();

break;

case 6:

printf("Enter Course Outcome Code to sort: ");

scanf("%s", cour\_out\_code);

Team\_unity\_Compare\_sorting\_quick();

break;

case 7:

printf("Enter Course Outcome Code to search: ");

scanf("%s", cour\_out\_code);

Team\_unity\_Compare\_Search\_linear(cour\_out\_code);

break;

case 8:

printf("Enter Course Outcome Code to search: ");

scanf("%s", cour\_out\_code);

Team\_unity\_Compare\_Search\_binary(cour\_out\_code);

break;

case 9:

Team\_unity\_complexity\_Search();

break;

case 10:

Team\_unity\_complexity\_sorting();

break;

case 11:

Team\_unity\_Search\_algorithm\_details();

break;

case 12:

Team\_unity\_sort\_algorithm\_details();

break;

case 13:

Team\_unity\_storing();

break;

case 14:

printf("Exiting the program.\n");

exit(0);

default:

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

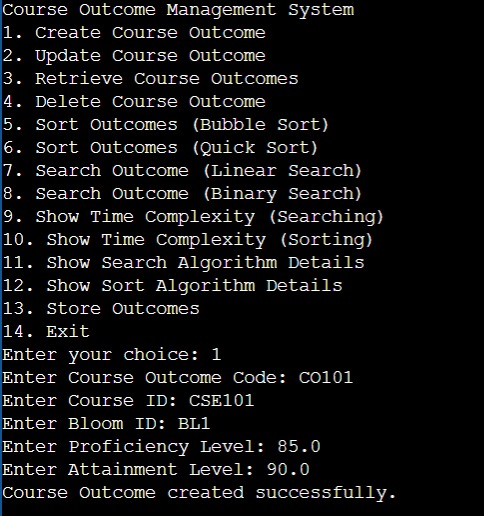
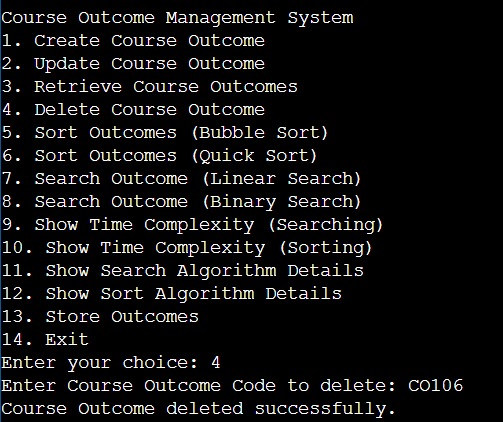
}

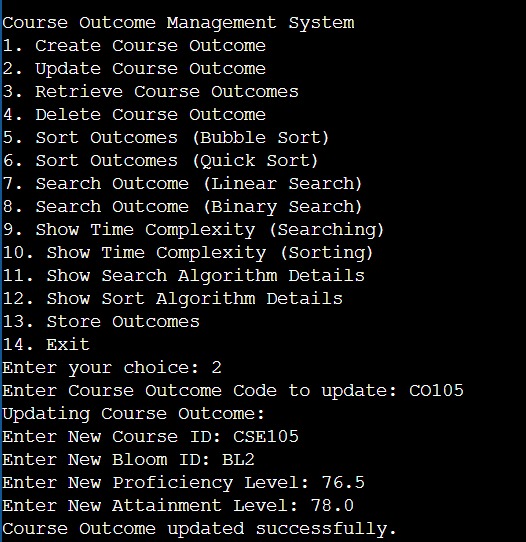
}

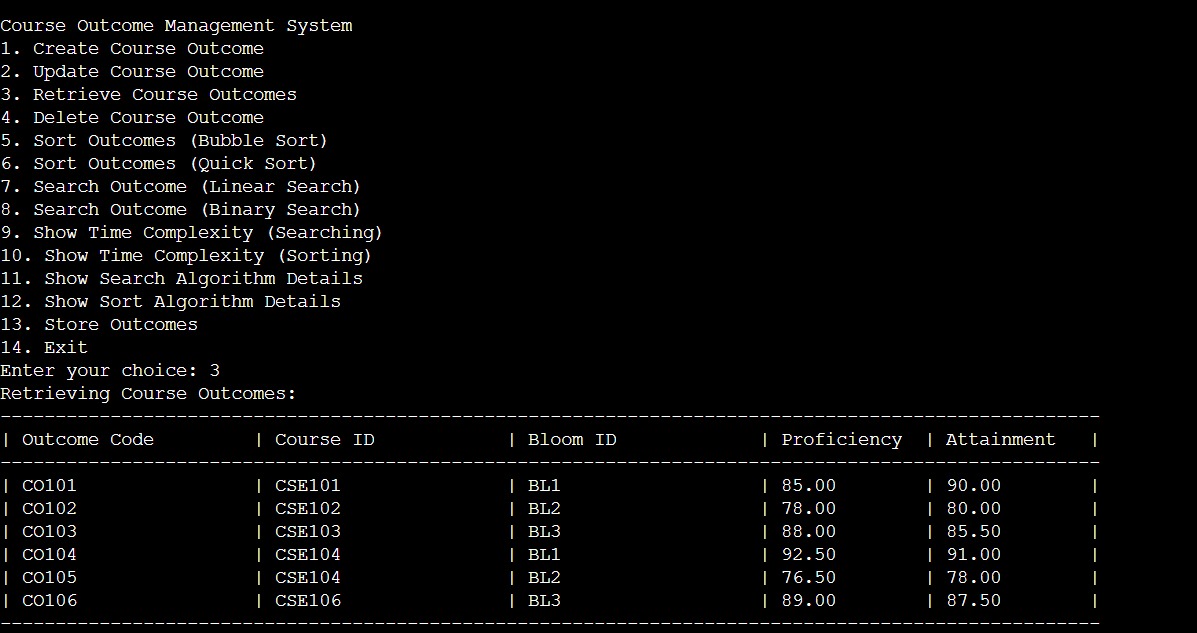
return 0;

}

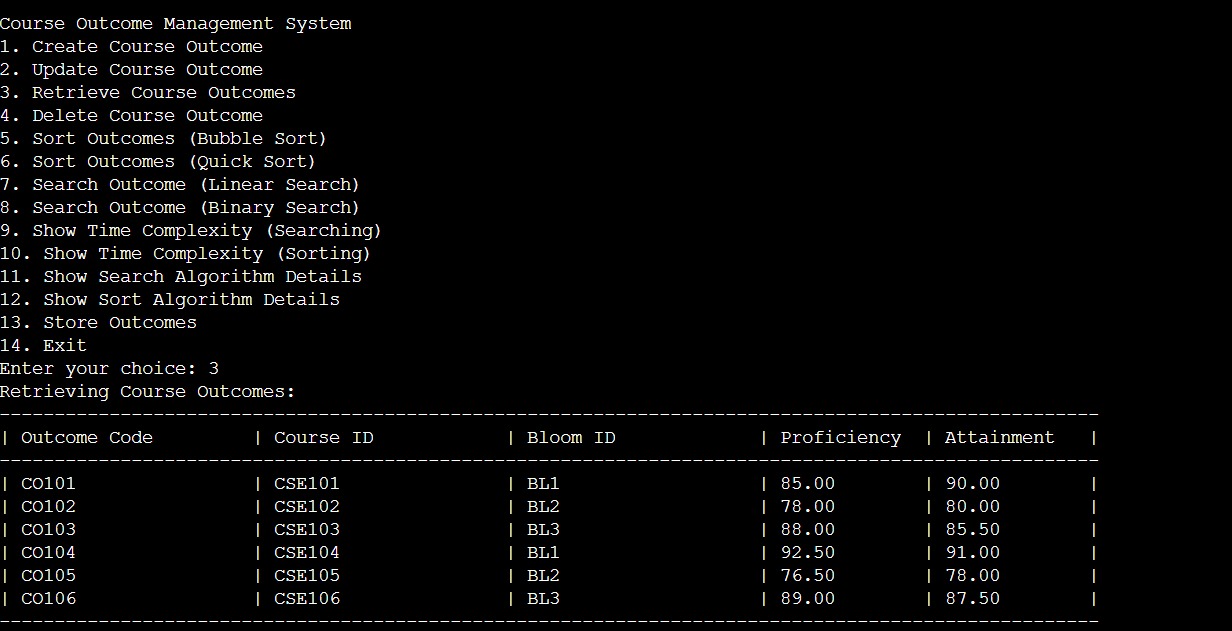
**Output**

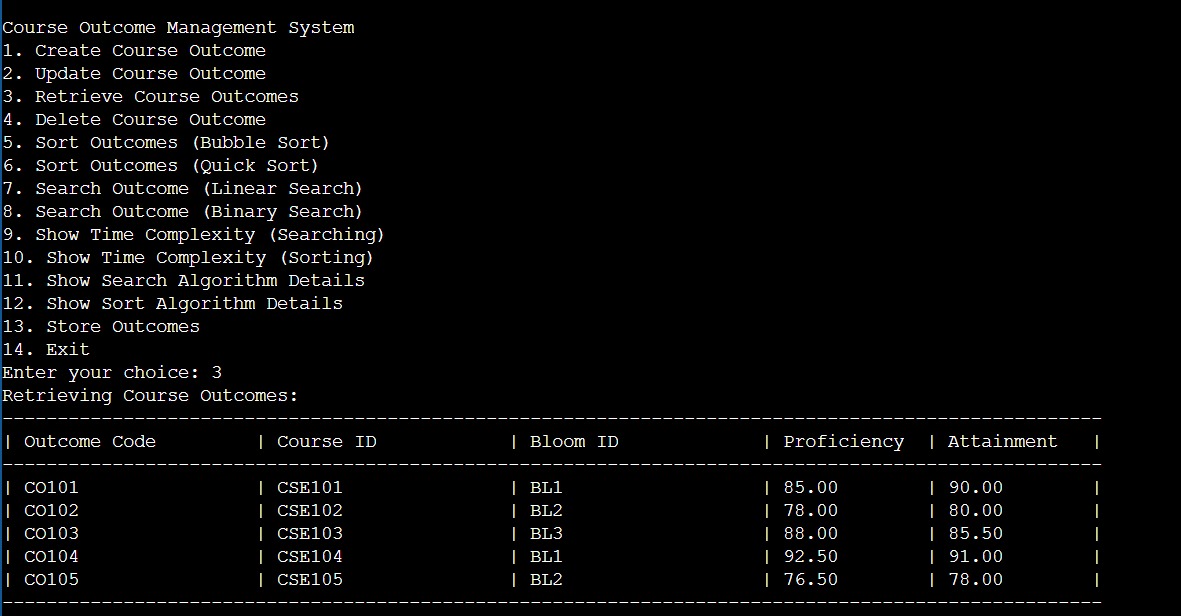
**Create Delete**

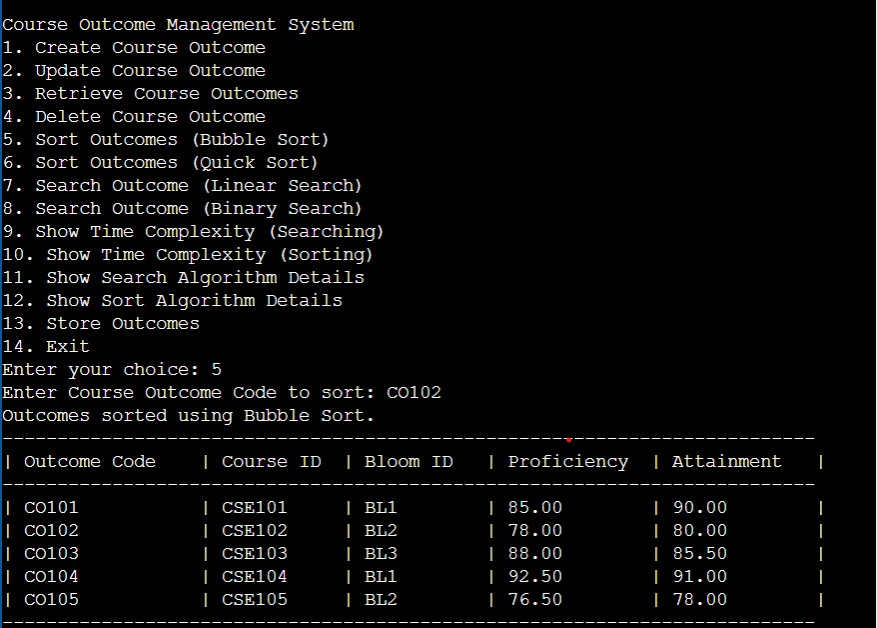
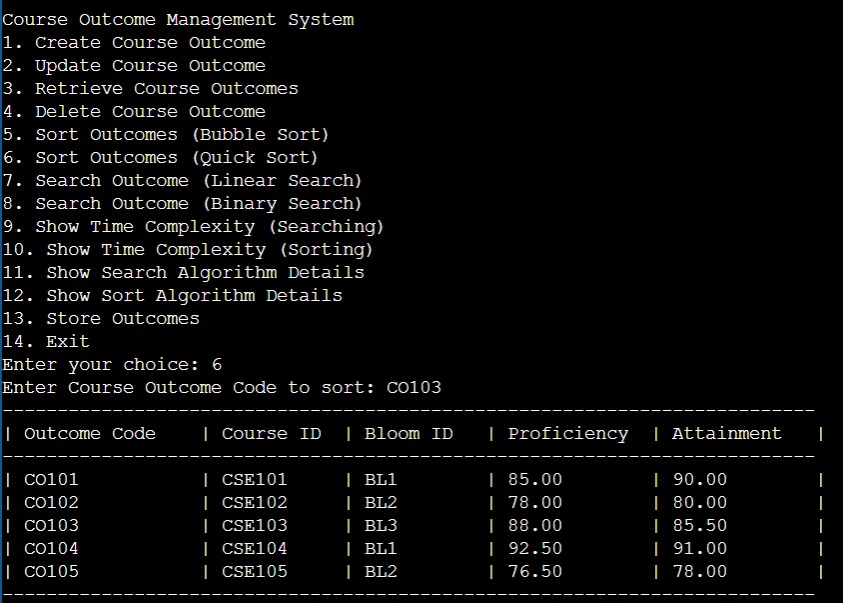
**Update**

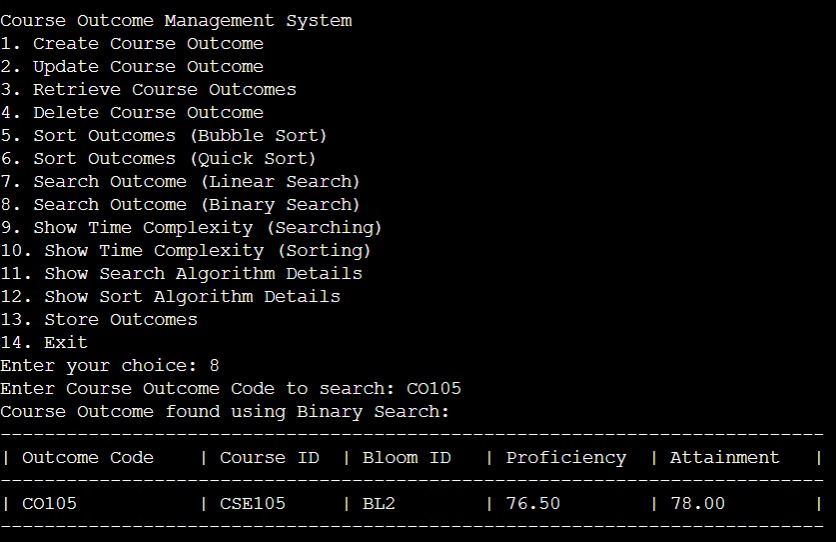
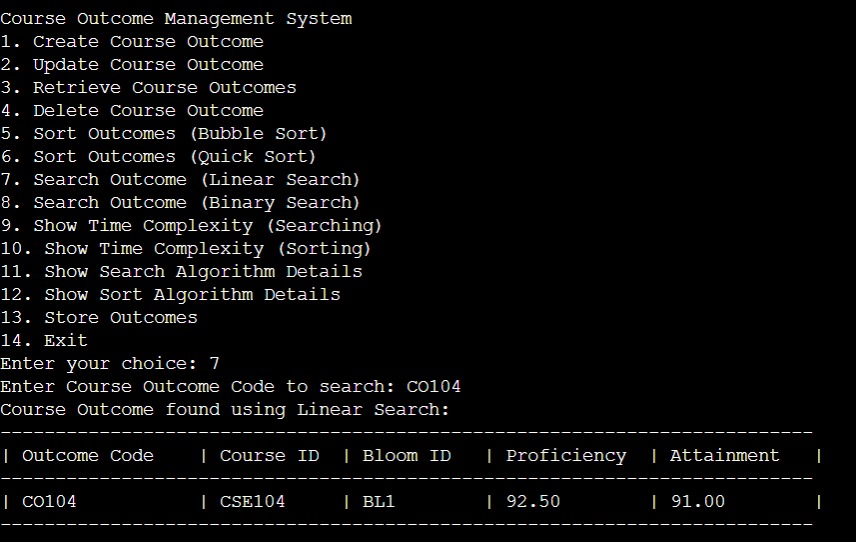
**After Creating (Retrieve)**

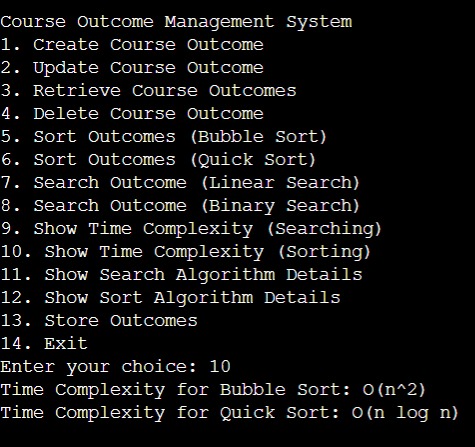
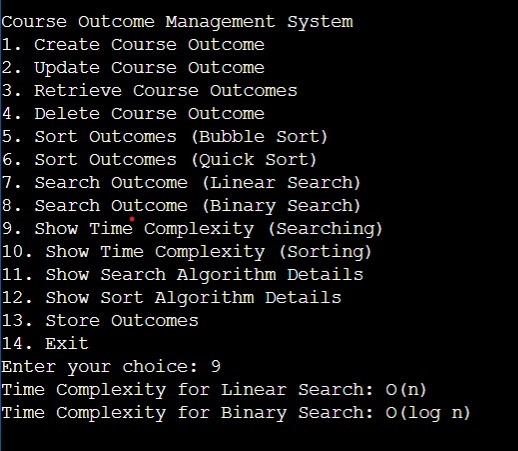
**After Updating(Retrieve)**

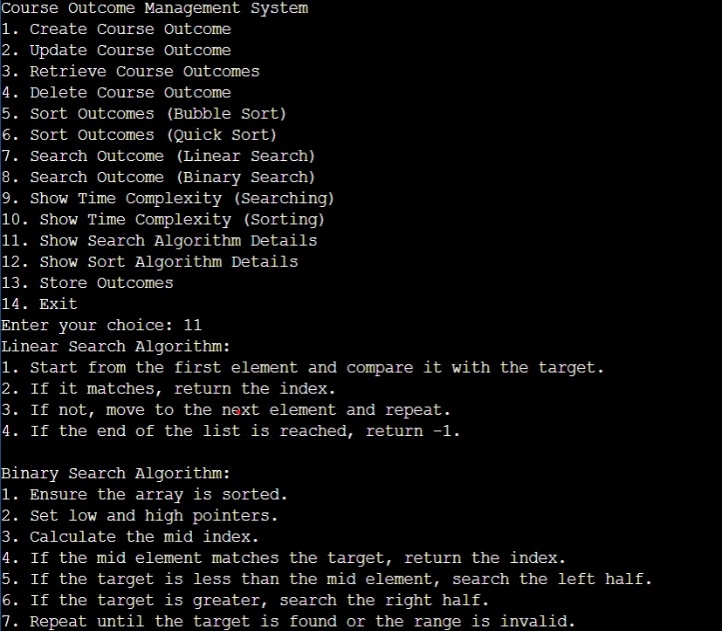
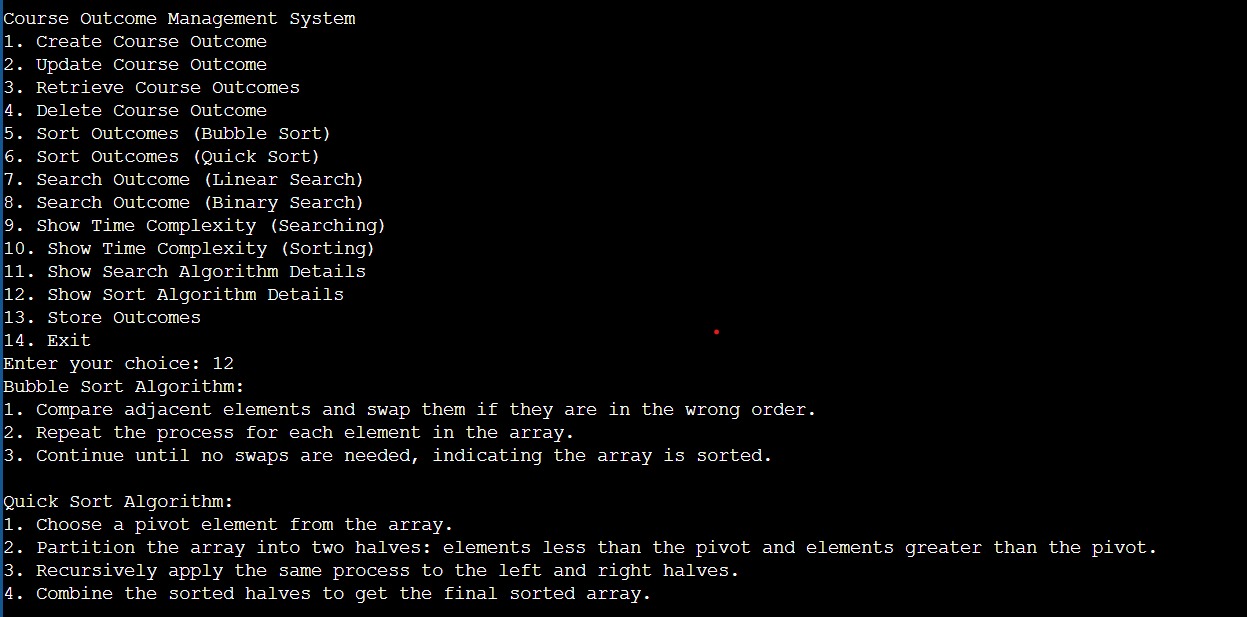


**After Deleting(Retrieve)**  


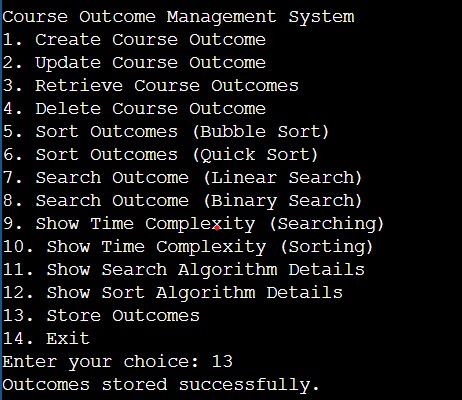
**Sorting**

**Searching**

**Time Complexity**

**Algorithm Details(Pseudocode)**

**Storing**



**Conclusion**

In conclusion, the Course Outcome Management module provides an efficient way to handle and maintain records of course outcomes. With features like CRUD operations, sorting, and searching, it simplifies data organization and access, ensuring quick retrieval and streamlined updates. The data persistence in course\_outcome.txt supports consistent access across sessions, and the project showcases basic algorithm comparisons, highlighting their strengths and limitations. This module effectively demonstrates practical data management for academic records, making it a valuable tool for course outcome tracking.