**Department of Computer science and Engineering**

**CS 204:Design and Analysis of Algorithm**

**Project Title:OBE Implementation**

***Team Deatails:***

**Team Name : Power Rangers**

**Team project : Course Reference**

**Team leader: N.Thanvitha – AP23110011199**

**Team members:**

**V.L.Nikitha - AP23110011195**

**M.Olive Chaitanya - AP23110011192**

**L.Neelima - AP23110011190**

**K.Narendra Kumar - AP23110011169**

**B.Raja Vardhan - AP23110011177**

***SUBMITTED TO***

**Designed by**

**GAVASKAR S,**

**Assistant Professor(Ad),**

**Department of CSE,**

**SRM University - AP.**

**INDEX**

**Introduction**

Project Modules:

**Architecture Diagram**

**Module Description**

Programming Details naming conventions to be used: Field/table details: Course Reference

Algorithm Details:

(i)Sorting

(ii)Searching

(ii) Storing the details in a text file

**Source Code**

power\_rangers\_course\_reference.cpp

**Comparison of Sorting Algorithms**

**Comparison of Searching Algorithms**

**Screen Shots**

**Conclusion**

Introduction

The Course\_Reference program is designed to streamline the management of course-related resources. The application supports basic CRUD (Create, Retrieve, Update, Delete) operations on these records, with functionalities for sorting and searching. The data is stored in a text file for persistence,This project demonstrates effective data management techniques, basic algorithm comparisons, and efficiency evaluation for sorting and searching operations.

Project Modules:

**Objective**

The Course Reference Module is designed to help manage and organize references related to various courses. It enables the storage, retrieval, updating, and deletion of reference records tied to specific course materials, making it easier to manage resources effectively.

**Features**

ID-based Identification: Each reference entry has a unique ID and Course ID to help identify and associate it with a specific course.

Attribute-based Sorting and Searching: Users can sort and search references based on the Course Reference Code, ensuring quick access to relevant materials.

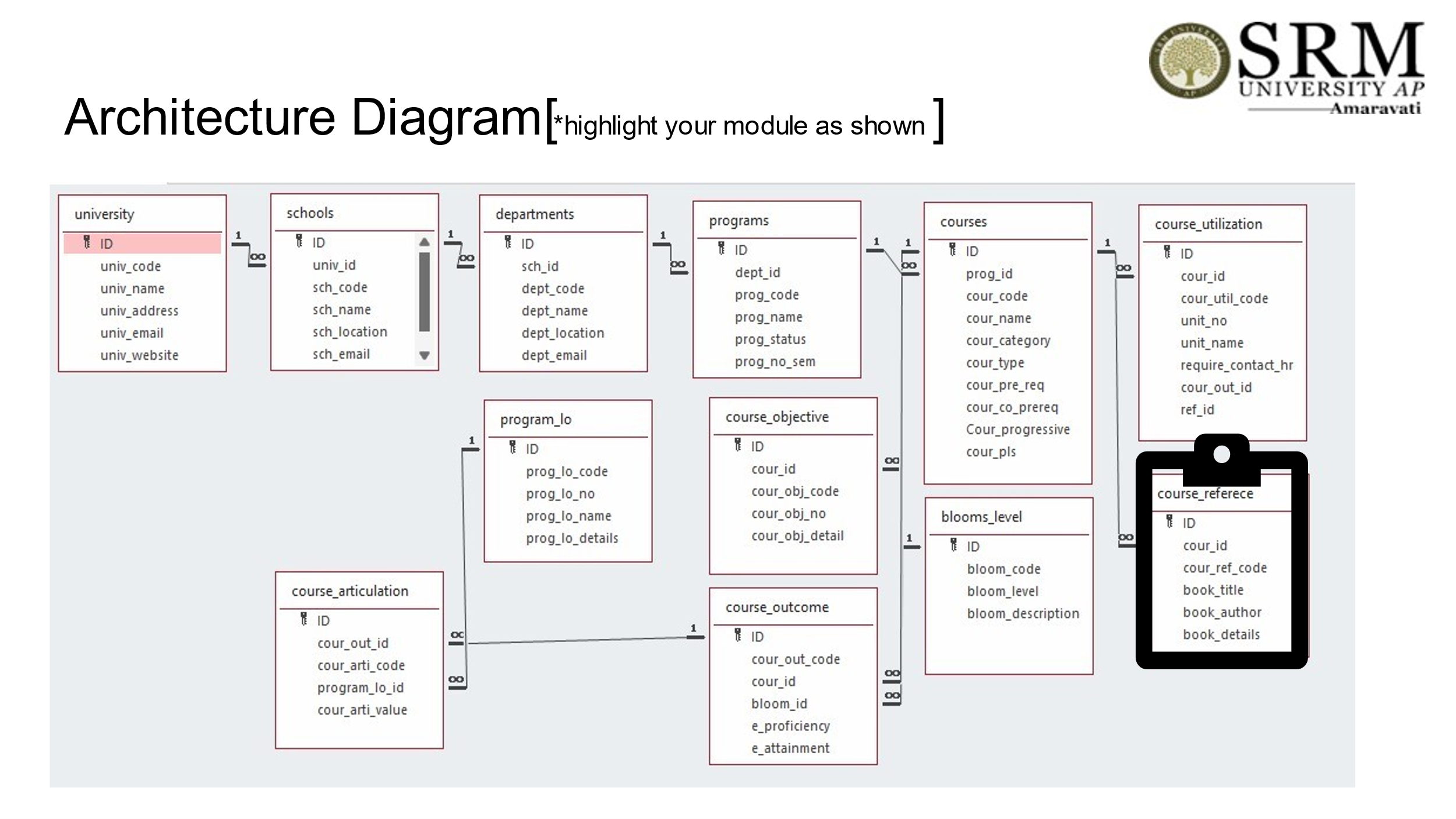
CRUD Operations: The module allows users to create, retrieve, update, and delete course references as needed, enabling comprehensive reference management.

**Implementation Details**

Insertion Sort: Used to sort course references by the Course Reference Code, making it easier to browse through ordered entries.

Linear Search: Used to locate specific course references based on their Course Reference Code, allowing for targeted look-ups.

Architecture Diagram



# Module Description

**Module Name: Course Reference**

The Course Reference Module facilitates CRUD operations on course reference records. Users can:

**Module Description:**

* Add a new course reference record.
* View (retrieve) existing records.
* Update specific course reference details.
* Delete course reference records.
* Data is stored in a file (course\_reference\_setting.txt), with each change (create, update, delete) directly reflected in this file. Sorting and searching functionalities are included, allowing users to organize and locate course references based on attributes like the course reference code.

Programming Details naming conventions to be used:

* **File name:** power\_rangers\_course\_reference.cpp

● **Function/method name**

○ **Create:** power\_rangers\_course\_reference\_create

○ **Update:** power\_rangers\_course\_reference\_update

○ **Retrieve:** power\_rangers\_course\_reference\_ retrive

○ **Delete:** power\_rangers\_course\_reference\_delete

○**Sorting:**power\_rangers\_ course\_reference\_insertion\_sort

○**Searching:**power\_rangers\_ course\_reference\_search

○**Storing:** power\_rangers\_course\_reference\_storing

○**Comparison(both searching and Sorting)**:

■ For Searching- power\_rangers\_course\_reference\_ Compare\_Search\_binary\_search

■ For Sorting- power\_rangers\_course\_reference\_ Compare\_sorting\_bubble\_sort

○ **Time Complexity(both searching and Sorting):**

■For Searching- power\_rangers\_course\_reference\_complexity\_Linear\_Search

■ For Sorting- power\_rangers\_course\_reference\_ compexity\_Selection\_Sort

○ **Algorithm Details(pseudocode or steps)(both searching and Sorting):**

■ For Searching- power\_rangers\_course\_reference\_linear\_search \_details

■ For Sorting- power\_rangers\_course\_reference\_insertion\_sort\_details

● **File name(for storing the details)**

○ File name to be used is:-course\_reference\_setting .txt

Field/table details: Course Reference

|  |  |
| --- | --- |
| **Field Name** | **Data type** |
| id | integer |
| cour\_id | integer |
| cour\_ref\_code | String |
| book\_title | String |
| book\_author | String |
| book\_details | String |

Algorithm Details:

(i)Sorting

Sorting in the Course Reference Module is based on the attribute cour\_ref\_code (Course Reference Code). The module uses Insertion Sort as its primary sorting algorithm and includes comparisons with Selection Sort for further insight:

Primary Sorting Algorithm (Insertion Sort): This algorithm works by taking each element and placing it in its correct position relative to already sorted elements, which makes it efficient for smaller datasets or nearly sorted data. Its simplicity suits the course reference data but has limitations with large datasets due to its O(n²) time complexity.

Comparison Algorithm (Selection Sort): Similar in complexity to Insertion Sort, Selection Sort selects the minimum element in each pass and places it in order. While effective for basic datasets, it’s less efficient than Insertion Sort when applied to partially sorted data.

(ii)Searching

Searching in the Course Reference Module enables users to find specific course references based on fields such as cour\_ref\_code (Course Reference Code). Two algorithms are employed:

Primary Searching Algorithm (Linear Search): This simple, straightforward algorithm checks each record one by one. It is effective for smaller datasets or when data is unsorted, with a time complexity of O(n).

Comparison Algorithm (Binary Search): When course references are sorted by cour\_ref\_code, Binary Search can be applied, providing a more efficient O(log n) time complexity. This algorithm quickly narrows down the search by repeatedly dividing the dataset in half.

(iii) Storing the details in a text file

* Storing the details in the text file once details are entered.
* Delete the detail from the text file once details are deleted.
* Update the text file once details are updated.

Source Code

#include<iostream>

#include<fstream>

#include<vector>

#include<iomanip>

using namespace std;

class Course\_Reference {

public:

int id;

int cour\_id;

string cour\_ref\_code;

string book\_title;

string book\_author;

string book\_details;

};

int course\_reference\_count = 0;

vector<Course\_Reference> course\_references;

const char\* FILE\_NAME = "course\_reference\_setting.txt";

void power\_rangers\_course\_reference\_create();

void power\_rangers\_course\_reference\_update();

void power\_rangers\_course\_reference\_retrive();

void power\_rangers\_course\_reference\_delete();

void power\_rangers\_course\_reference\_storing();

void power\_rangers\_course\_reference\_sort\_by\_ref\_code();

void power\_rangers\_course\_reference\_search\_by\_ref\_code();

void power\_rangers\_course\_reference\_complexity\_search(){

cout<<"Time Complexity analysis:"<<endl;

cout<<"Linear Search: O(n)"<<endl;

cout<<"Binary Search: O(logn)"<<endl;

}

void power\_rangers\_course\_reference\_complexity\_sorting(){

cout<<"Time Complexity analysis:"<<endl;

cout<<"Insertion Sort: O(n)"<<endl;

cout<<"Selection Sort: O(n)"<<endl;

}

void load\_from\_file() {

ifstream file(FILE\_NAME);

if(!file.is\_open()) {

cerr<<"Failed to open file: "<<FILE\_NAME<<endl;

return;

}

Course\_Reference course\_ref;

while(file >> course\_ref.id >> course\_ref.cour\_id) {

file.ignore();

getline(file, course\_ref.cour\_ref\_code);

getline(file, course\_ref.book\_title);

getline(file, course\_ref.book\_author);

getline(file, course\_ref.book\_details);

course\_references.push\_back(course\_ref);

course\_reference\_count++;

}

file.close();

}

void power\_rangers\_course\_reference\_storing() {

ofstream file(FILE\_NAME, ios::out | ios::trunc);

if(!file.is\_open()) {

cerr<<"Failed to open file: "<<FILE\_NAME<<endl;

return;

}

for(int i = 0;i < course\_references.size();i++) {

file << course\_references[i].id << " "

<< course\_references[i].cour\_id << "\n"

<< course\_references[i].cour\_ref\_code << "\n"

<< course\_references[i].book\_title << "\n"

<< course\_references[i].book\_author << "\n"

<< course\_references[i].book\_details << "\n";

}

file.close();

}

void power\_rangers\_course\_reference\_create() {

Course\_Reference c;

cout<<"Enter ID: ";

cin>>c.id;

cout<<"Enter Course ID: ";

cin>>c.cour\_id;

cin.ignore();

cout<<"Enter Course Reference Code: ";

getline(cin,c.cour\_ref\_code);

cout<<"Enter Book Title: ";

getline(cin,c.book\_title);

cout<<"Enter Book Author: ";

getline(cin,c.book\_author);

cout<<"Enter Book Details: ";

getline(cin,c.book\_details);

course\_reference\_count++;

course\_references.push\_back(c);

power\_rangers\_course\_reference\_storing();

cout<<"Course Reference created successfully!"<<endl;

}

void power\_rangers\_course\_reference\_update() {

int id;

cout<<"Enter ID to update: ";

cin>>id;

bool found = false;

for(int i = 0;i < course\_references.size();i++){

if(course\_references[i].id == id){

cout<<"Enter new ID: ";

cin>>course\_references[i].id;

cout<<"Enter new Course ID: ";

cin>>course\_references[i].cour\_id;

cin.ignore();

cout<<"Enter new Course Reference Code: ";

getline(cin,course\_references[i].cour\_ref\_code);

cout<<"Enter new Book Title: ";

getline(cin,course\_references[i].book\_title);

cout<<"Enter new Book Author: ";

getline(cin,course\_references[i].book\_author);

cout<<"Enter new Book Details: ";

getline(cin,course\_references[i].book\_details);

power\_rangers\_course\_reference\_storing();

cout<<"Course Reference updated successfully!"<<endl;

found = true;

break;

}

}

if(!found) cout<<"Course Reference with ID "<<id<<" not found!"<<endl;

}

void power\_rangers\_course\_reference\_retrieve() {

cout << "\n\t\t\t\t\tList of Course References" << endl;

cout << "----------------------------------------------------------------------------------------------------" << endl;

if (course\_references.size() <= 0) {

cout << "No Course References found!" << endl;

return;

}

cout << left << setw(10) << "ID"

<< setw(15) << "Course ID"

<< setw(20) << "Reference Code"

<< setw(25) << "Book Title"

<< setw(20) << "Author"

<< "Details" << endl;

cout << "----------------------------------------------------------------------------------------------------" << endl;

for (int i = 0; i < course\_references.size(); i++) {

cout << left << setw(10) << course\_references[i].id

<< setw(15) << course\_references[i].cour\_id

<< setw(20) << course\_references[i].cour\_ref\_code

<< setw(25) << course\_references[i].book\_title

<< setw(20) << course\_references[i].book\_author

<< course\_references[i].book\_details << endl;

}

}

void power\_rangers\_course\_reference\_delete() {

int id;

cout<<"Enter ID to delete: ";

cin>>id;

bool found = false;

for(int i = 0;i < course\_references.size();i++) {

if(course\_references[i].id == id) {

course\_references.erase(course\_references.begin()+i);

course\_reference\_count--;

found = true;

cout<<"Course Reference deleted successfully!"<<endl;

break;

}

}

if(found) power\_rangers\_course\_reference\_storing();

else cout<<"Course Reference with ID "<<id<<" not found!"<<endl;

}

void power\_rangers\_course\_reference\_linear\_search\_by\_ref\_code() {

string id;

cout<<"Enter Course Reference Code to search: ";

cin.ignore();

getline(cin,id);

for(int i = 0;i < course\_references.size();i++) {

if(course\_references[i].cour\_ref\_code == id) {

cout<<"ID : "<< course\_references[i].id << endl;

cout<<"Course ID : "<< course\_references[i].cour\_id << endl;

cout<<"Course Reference Code : "<< course\_references[i].cour\_ref\_code << endl;

cout<<"Book Title : "<< course\_references[i].book\_title << endl;

cout<<"Book Author : "<< course\_references[i].book\_author << endl;

cout<<"Book Details : "<< course\_references[i].book\_details << endl;

return;

}

}

cout<<"Course Reference with "<<id<<" not found!"<<endl;

}

void power\_rangers\_course\_reference\_insertion\_sort\_by\_ref\_code() {

for(int i = 1;i < course\_references.size();i++) {

Course\_Reference temp = course\_references[i];

int j = i-1;

while(j >= 0 && course\_references[j].cour\_ref\_code > temp.cour\_ref\_code) {

course\_references[j+1] = course\_references[j];

j--;

}

course\_references[j+1] = temp;

}

cout<<"Course References sorted by Course Reference Code."<<endl;

power\_rangers\_course\_reference\_storing();

power\_rangers\_course\_reference\_retrieve();

}

void power\_rangers\_course\_reference\_compare\_sorting\_selection\_sort() {

for (int i = 0; i < course\_references.size() - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < course\_references.size(); j++) {

if (course\_references[j].cour\_ref\_code < course\_references[minIndex].cour\_ref\_code) {

minIndex = j;

}

}

// Swap the found minimum element with the first element

if (minIndex != i) {

swap(course\_references[i], course\_references[minIndex]);

}

}

cout << "Course References sorted by Course Reference Code using Selection Sort." << endl;

power\_rangers\_course\_reference\_storing();

power\_rangers\_course\_reference\_retrieve();

}

void power\_rangers\_course\_reference\_compare\_search\_binary\_search() {

// power\_rangers\_course\_reference\_insertion\_sort\_by\_ref\_code();

for (int i = 0; i < course\_references.size() - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < course\_references.size(); j++) {

if (course\_references[j].cour\_ref\_code < course\_references[minIndex].cour\_ref\_code) {

minIndex = j;

}

}

// Swap the found minimum element with the first element

if (minIndex != i) {

swap(course\_references[i], course\_references[minIndex]);

}

}

string search\_code;

cout << "Enter Course Reference Code to search: ";

cin.ignore();

getline(cin, search\_code);

int left = 0;

int right = course\_references.size() - 1;

bool found = false;

while (left <= right) {

int mid = left + (right - left) / 2;

if (course\_references[mid].cour\_ref\_code == search\_code) {

// Display the found course reference

cout << "ID : " << course\_references[mid].id << endl;

cout << "Course ID : " << course\_references[mid].cour\_id << endl;

cout << "Course Reference Code : " << course\_references[mid].cour\_ref\_code << endl;

cout << "Book Title : " << course\_references[mid].book\_title << endl;

cout << "Book Author : " << course\_references[mid].book\_author << endl;

cout << "Book Details : " << course\_references[mid].book\_details << endl;

found = true;

break;

}

else if (course\_references[mid].cour\_ref\_code < search\_code) {

left = mid + 1;

}

else {

right = mid - 1;

}

}

if (!found) {

cout << "Course Reference with code " << search\_code << " not found!" << endl;

}

}

void power\_rangers\_course\_reference\_insertion\_sort\_details(){

cout<<"for i = 1 to n-1"<<endl;

cout<<"key = array[i]"<<endl;

cout<<"j = i - 1"<<endl;

cout<<"while j >= 0 and array[j] > key"<<endl;

cout<<"array[j + 1] = array[j]"<<endl;

cout<<"j = j - 1"<<endl;

cout<<"array[j + 1] = key"<<endl;

}

void power\_rangers\_course\_reference\_linear\_search\_details(){

cout<<"function linearSearch(array, target):"<<endl;

cout<<"for i = 0 to array.length - 1"<<endl;

cout<<"if array[i] == target"<<endl;

cout<<"return i"<<endl;

cout<<"return -1 " ;

}

int main() {

load\_from\_file();

int choice;

while(1) {

cout<<"\n1)Create Course Reference\n2)Update Course Reference\n3)Retrieve Course References\n4)Delete Course Reference\n5)Linear Search by Course Reference Code\n6)Insertion Sort by Course Reference Code\n7)Display Complexity for searching\n8)Selection Sort by Course Reference Code\n9)Binary Search by Course Reference Code\n10)Display Complexity for sorting\n11)pseudocode for linear search\n12)pseudocode for Insertion sort\n13)Exit \n";

cout<<"Enter your choice : ";

cin>>choice;

switch(choice) {

case 1:

power\_rangers\_course\_reference\_create();

break;

case 2:

power\_rangers\_course\_reference\_update();

break;

case 3:

power\_rangers\_course\_reference\_retrieve();

break;

case 4:

power\_rangers\_course\_reference\_delete();

break;

case 5:

power\_rangers\_course\_reference\_linear\_search\_by\_ref\_code();

break;

case 6:

power\_rangers\_course\_reference\_insertion\_sort\_by\_ref\_code();

break;

case 7:

power\_rangers\_course\_reference\_complexity\_search();

break;

case 8:

power\_rangers\_course\_reference\_compare\_sorting\_selection\_sort();

break;

case 9:

power\_rangers\_course\_reference\_compare\_search\_binary\_search();

break;

case 10:

power\_rangers\_course\_reference\_complexity\_sorting();

break;

case 11:

power\_rangers\_course\_reference\_linear\_search\_details();

break;

case 12:

power\_rangers\_course\_reference\_insertion\_sort\_details();

break;

case 13:

exit(0);

default:

cout<<"Invalid Choice\n";

}

}

return 0;

}

# Comparison of Sorting Algorithms

Mechanism:

Insertion Sort : It starts with 1st element as sorted part. Then in takes each element from the unsorted part and inserts it into the correct position in sorted part, repeating this until the total array is sorted.

Selection Sort : It finds the minimum element from the unsorted part of the array and swap it with the 1st unsorted element. It moves the smallest element to its correct position in the sorted part. It is repeated until the total array is sorted.

Efficiency:

Time Complexity : Worst case - O(n^2) for both insertion and selection sort.

Insertion sort : Best case - O(n) when array is sorted.

Selection sort : It has consistent time complexity of O(n^2), regardless of initial order of elements.

Sapce Complexity : Both has space complexity of O(1).

For a dataset like course\_references, Insertion sort is better when updates are frequent and the data is partially sorted, as it efficiently places new elements in the correct position.

However, if the goal is a one time sort with memory optimization, Selection sort can be a good choice, as it uses less extra memory but has a higher time complexity O(n^2) for large datasets.

# Comparison of Searching Algorithms

Linear search : It searches each element upto end of the array, until the target is found.

Binary search : It divides the array in half to find the location of target. It eliminates half of the remaining elements in each step.

Time Complexity:

Linear search : Best case - O(1)

Worst case - O(n)

Binary search : Best case - O(1), when target is in middle

Worst case - O(logn)

Requirements:

Linear Search : No requirements. It works on both sorted and unsorted.

Binary Search : Data should be sorted. If not Binary search cannot be applied.

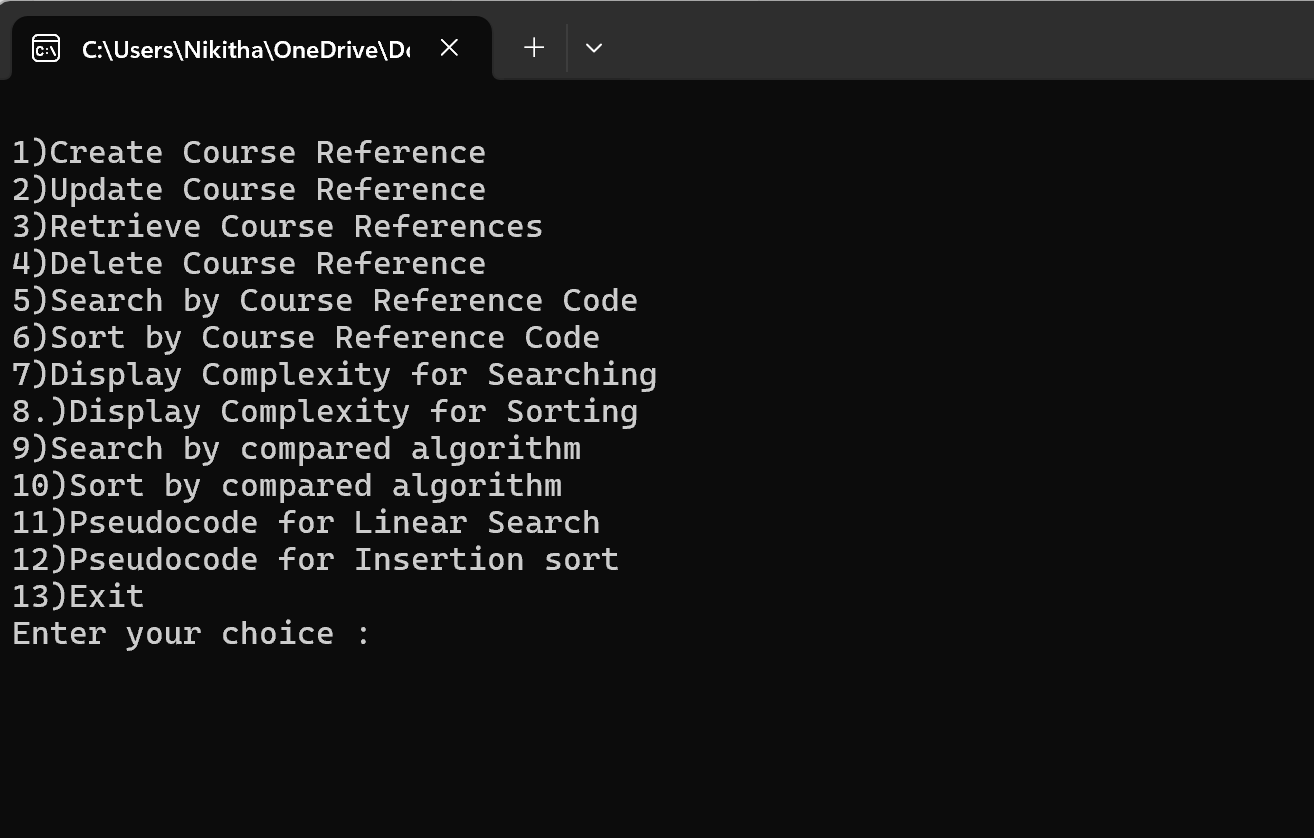
Which to Choose?

If our dataset is small or unsorted, linear search is a good choice.

If our dataset is large and sorted, binary search is highly recommended for better performance.

# Screen Shots

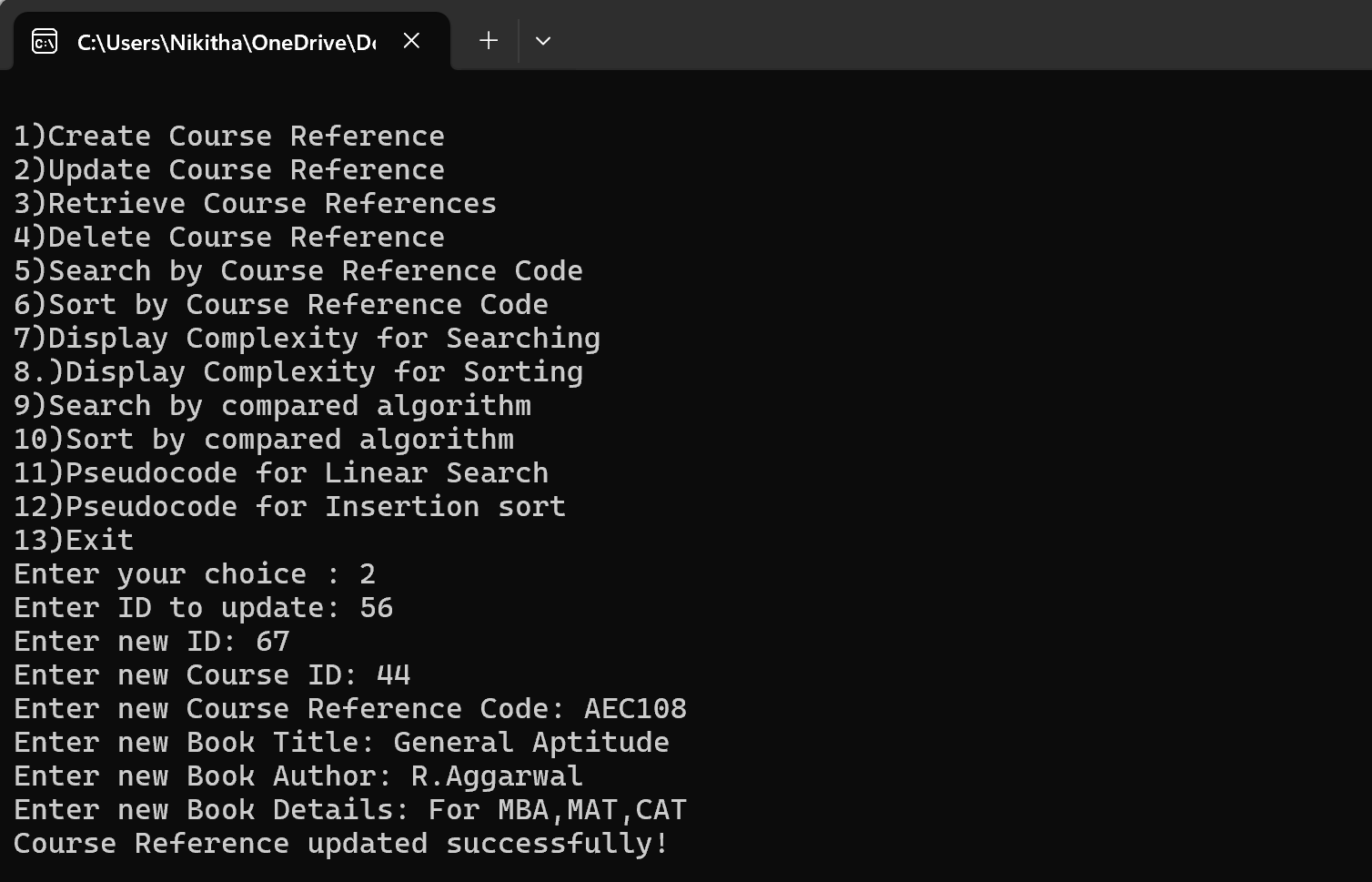
* **Menu**

****

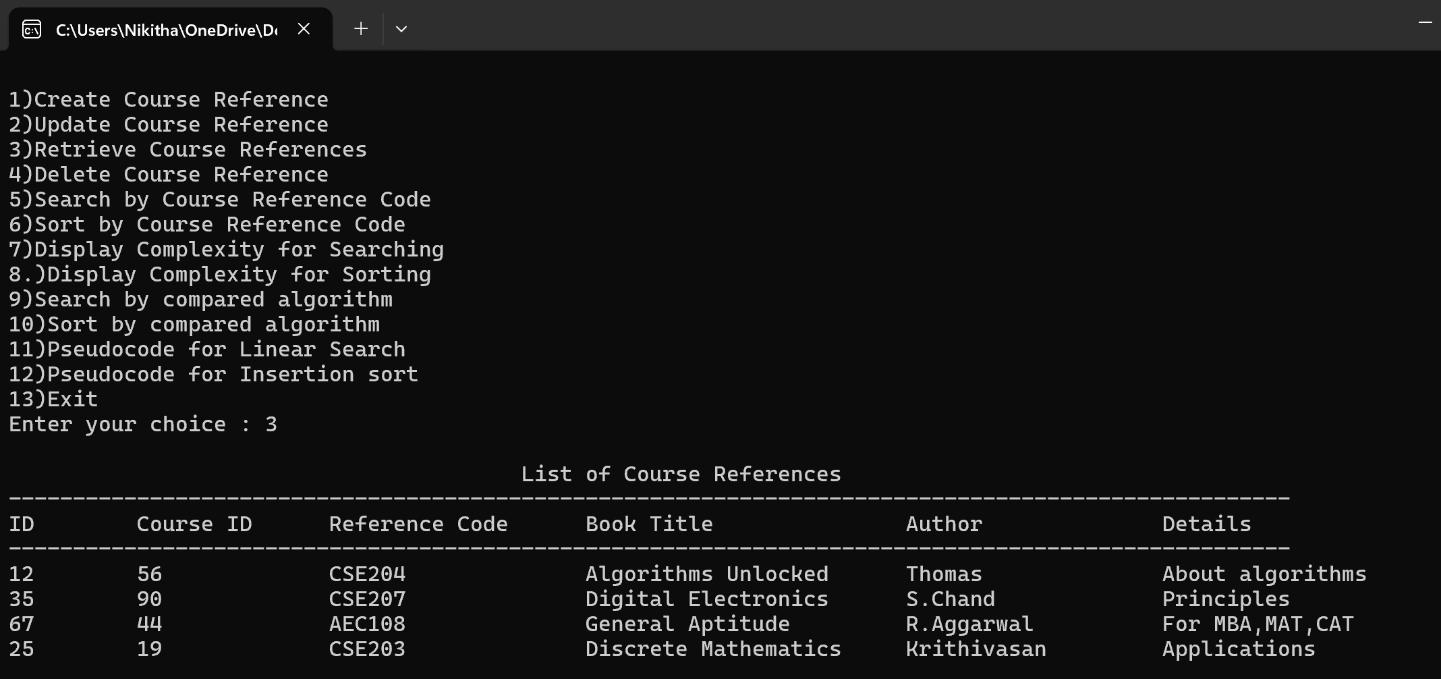
# **To create a Course Reference**

# 

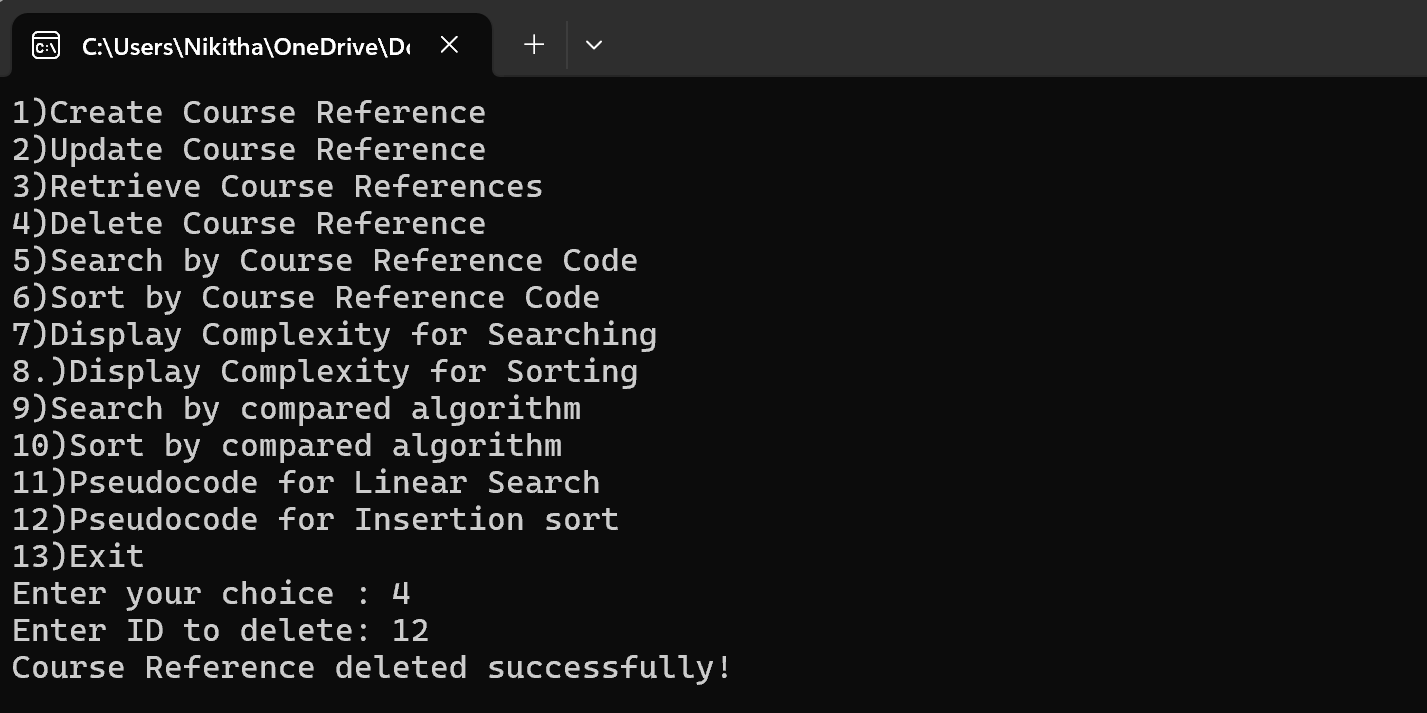
**To update a Course Reference**



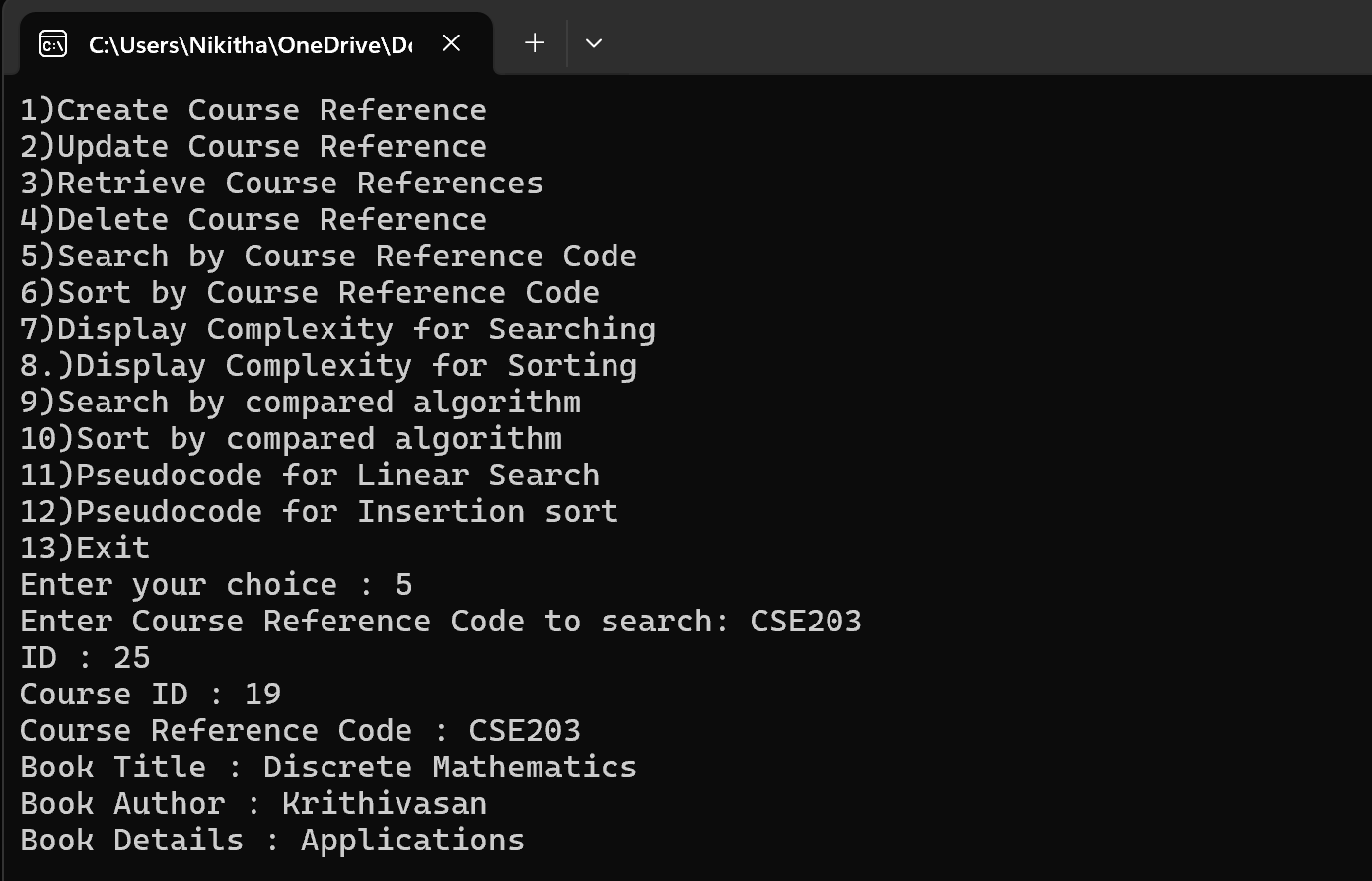
**To retrieve all Course References**



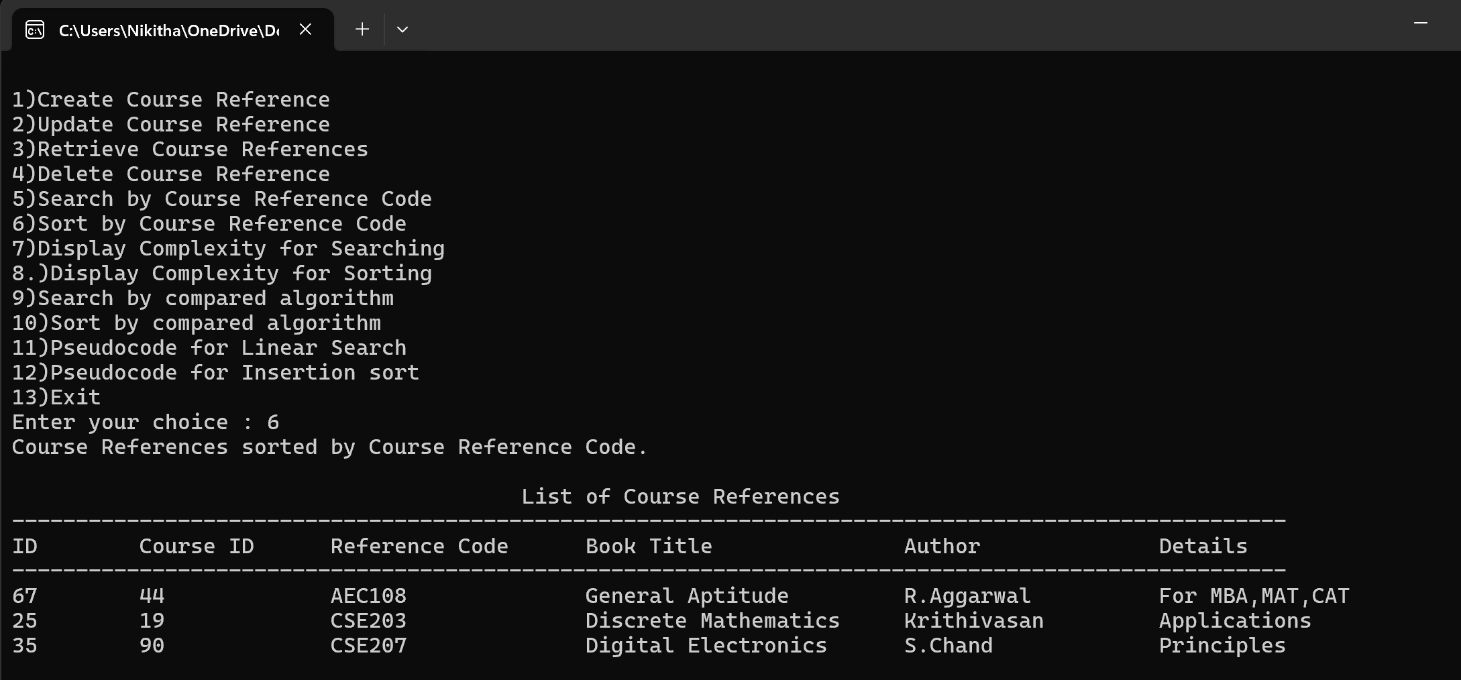
**To delete a Course Reference**

****

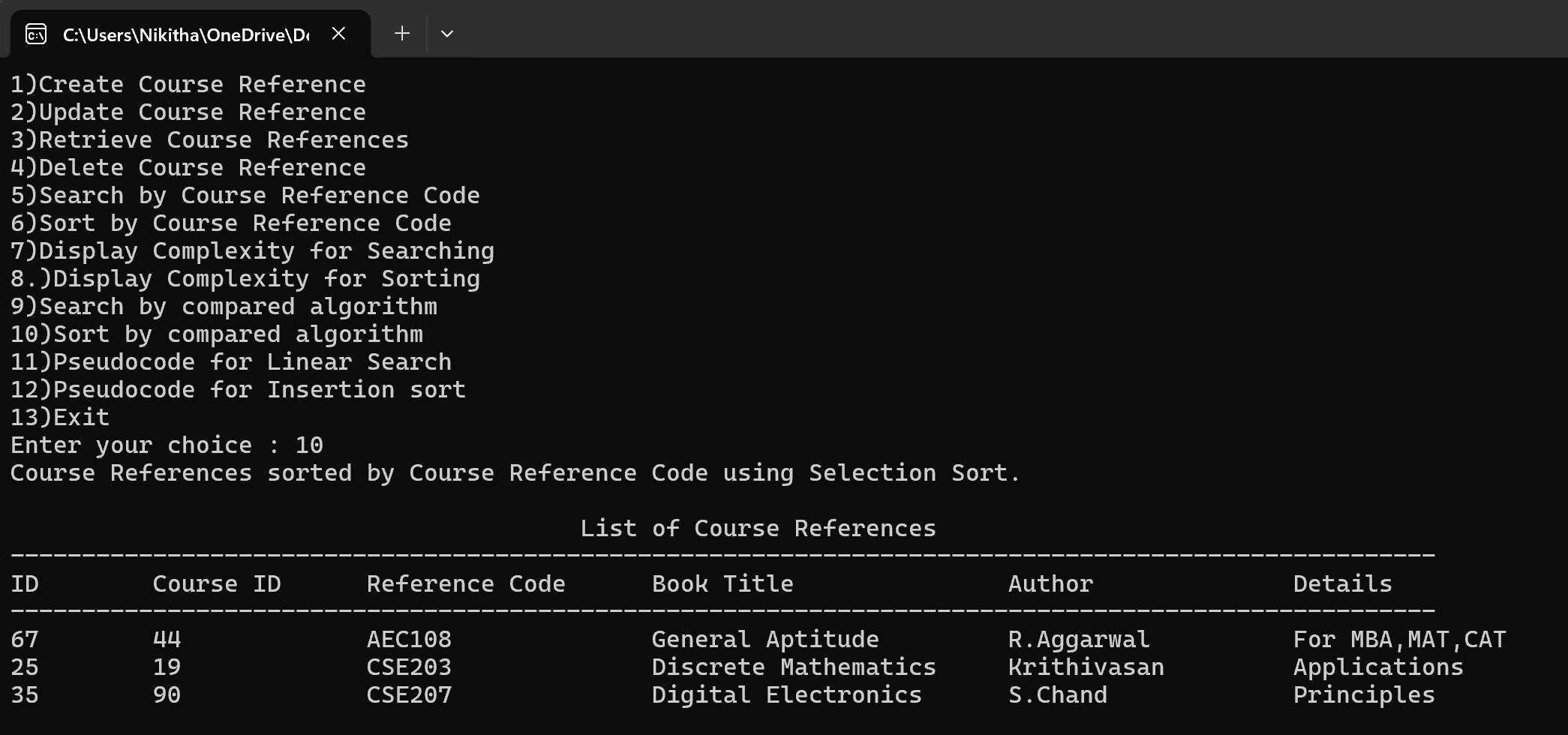
**To search a Course Reference (used Linear Search)**

****

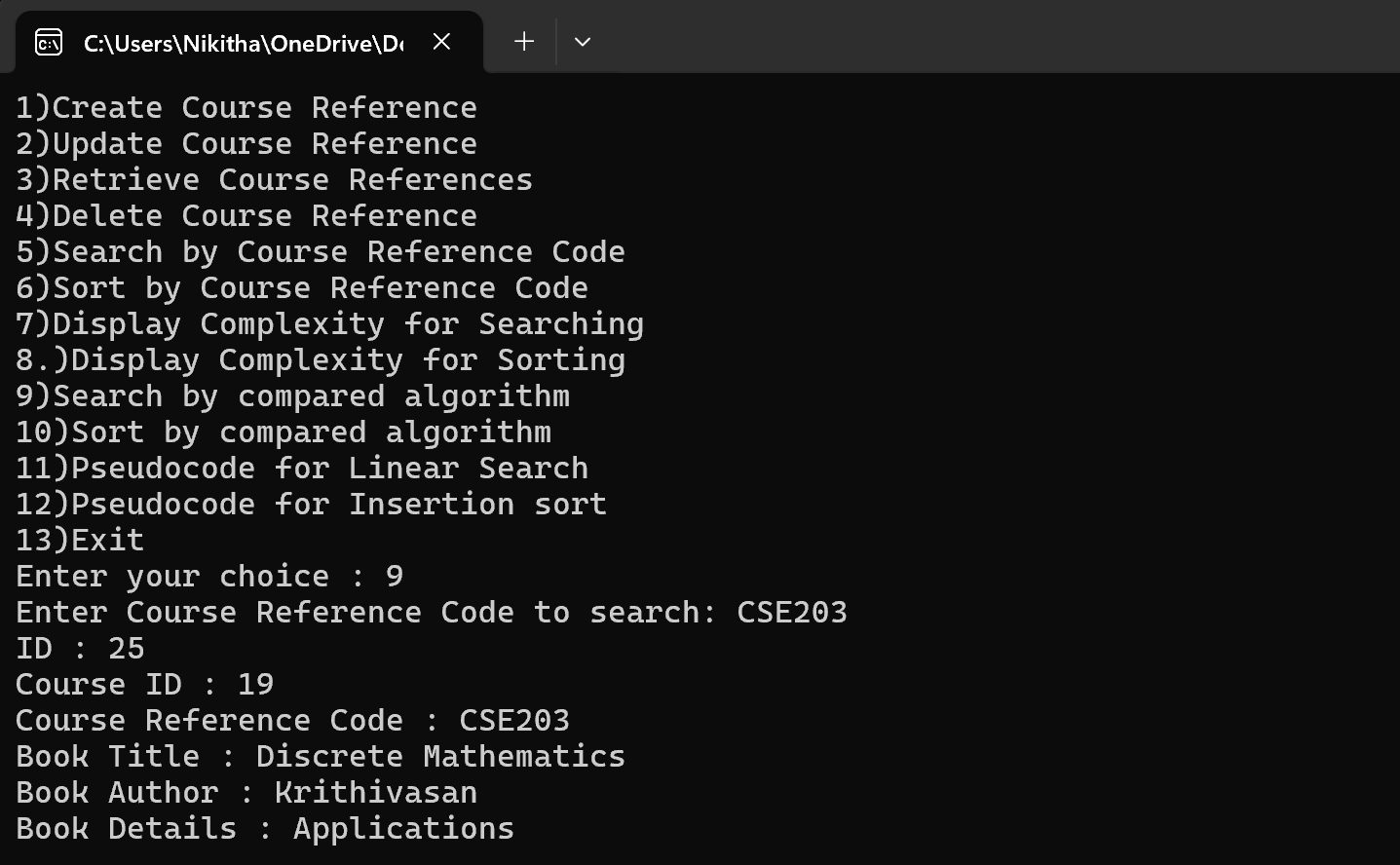
**To sort all Course References and display them (used Insertion Sort)**

****

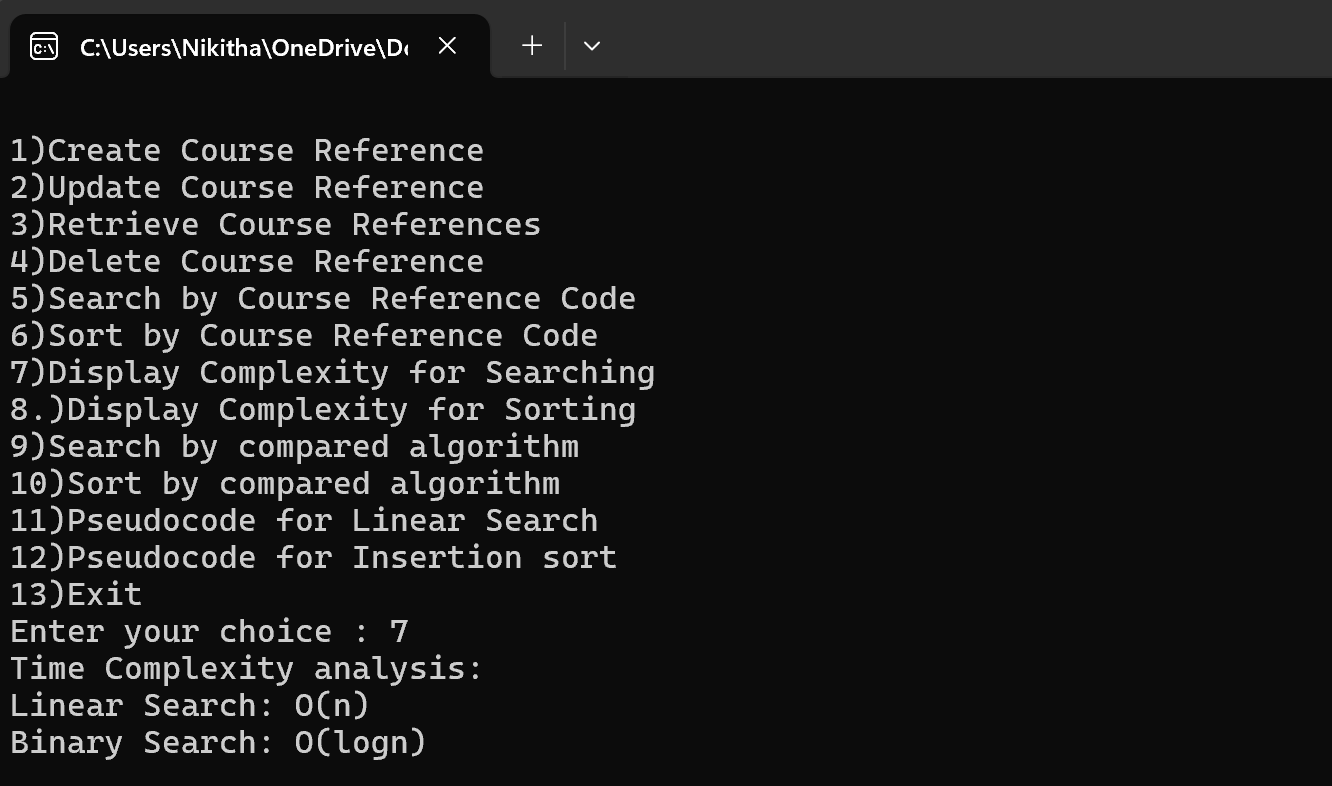
**To sort all Course References using compared algorithm and display them (used Selection Sort)**

****

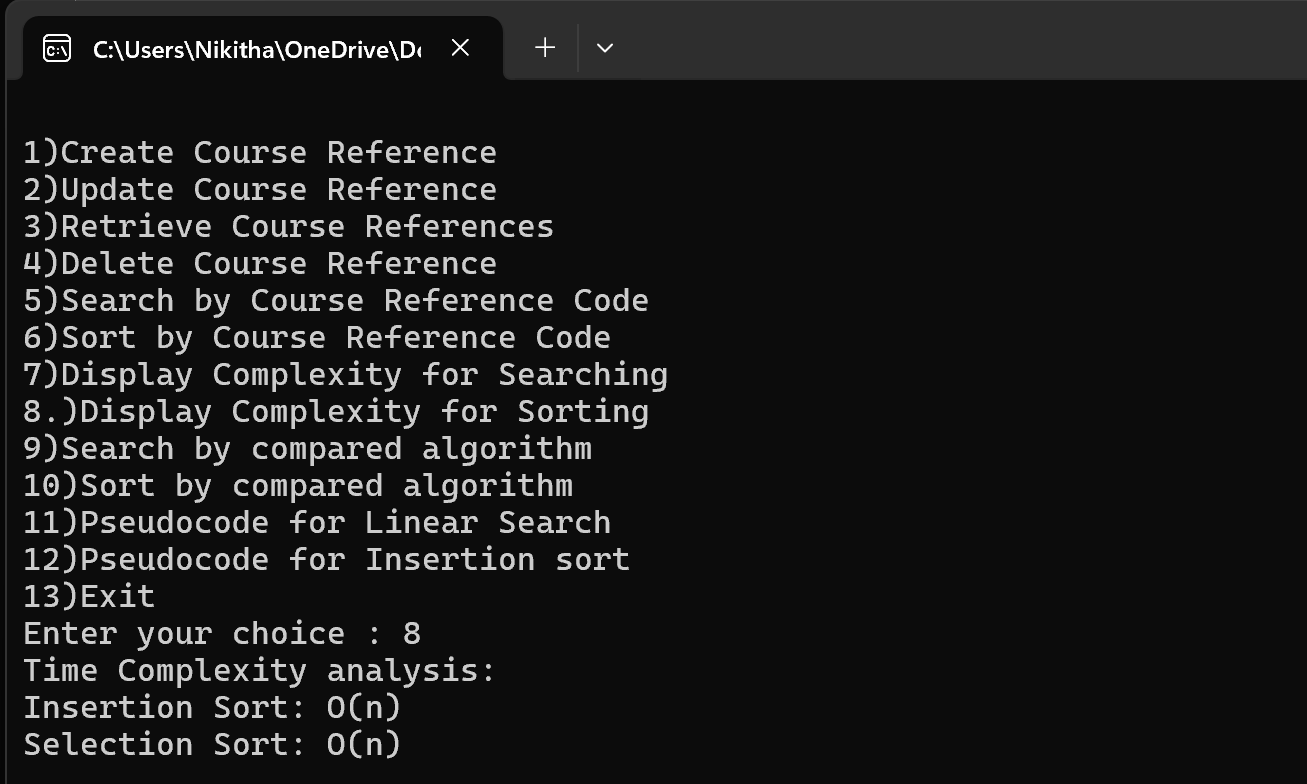
**To search a Course Reference using compared algorithm (used Binary Search)**

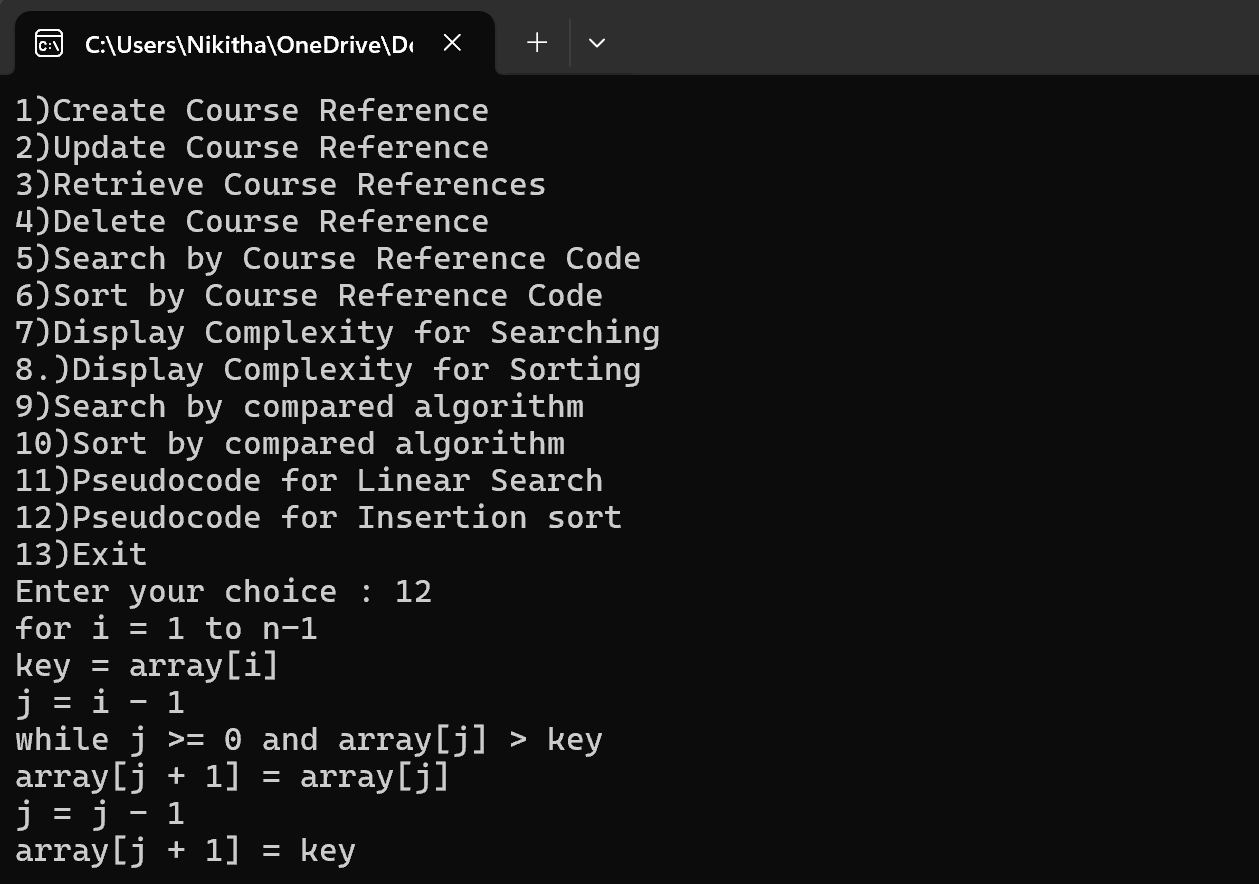
****

**To display Search Time Complexities**



**To display Sort Time Complexities**

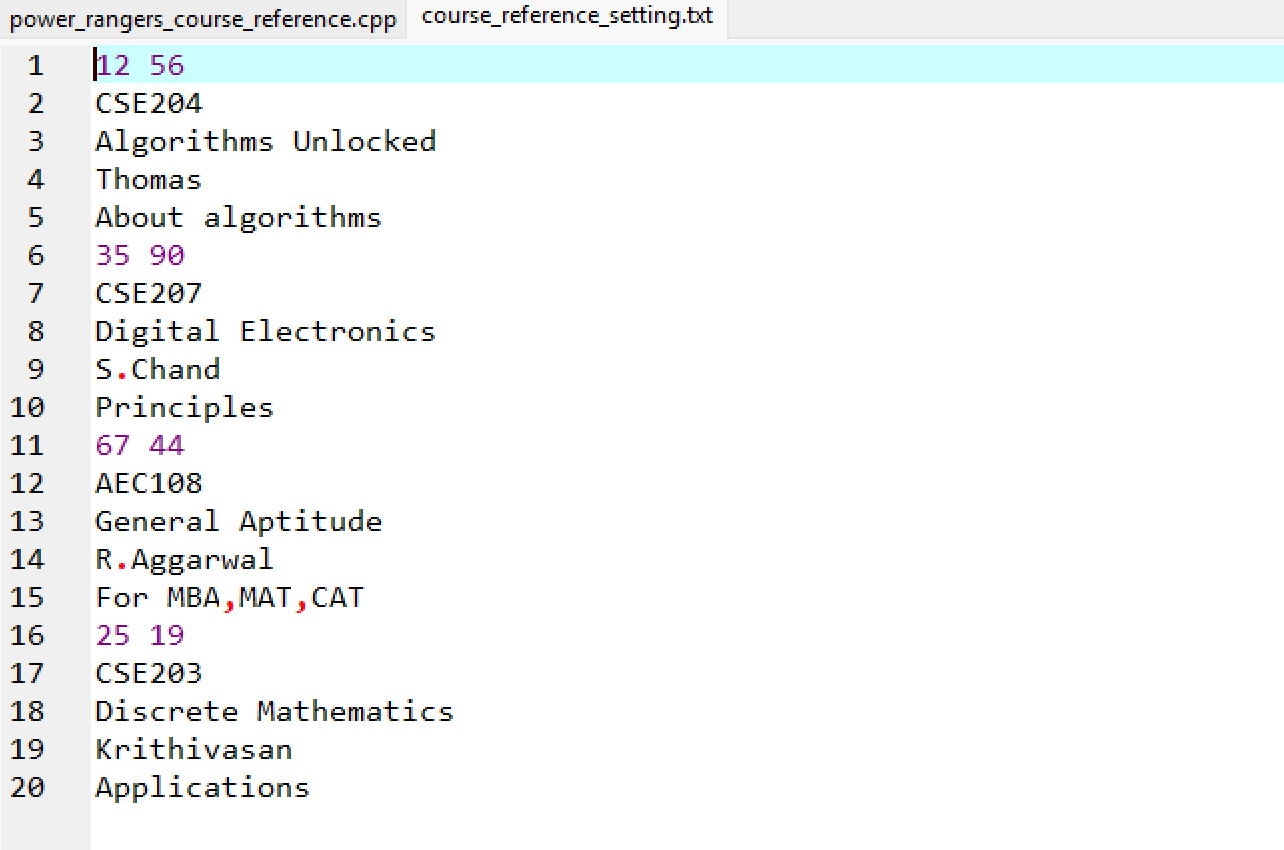
****

**To display pseudocode details for Sorting Algorithm**

**To display pseudocode details for Searching Algorithm**

# 

**Details of module being stored in "course\_reference\_setting.txt" file and updated dynamically based on CRUD operations.**



Conclusion :

The Course Reference module provides an effective solution for managing course references, with functionalities for CRUD operations and efficient data retrieval through sorting and searching. By comparing sorting and searching algorithms, this project highlights the importance of selecting the right algorithm for performance optimization. The module’s modular structure, along with file-based data persistence, ensures a reliable and maintainable system for managing course references. This approach demonstrates how algorithmic choices can significantly impact system efficiency, making it a scalable and robust solution for course reference management.