EEL ACTIVITY NO.5

SARVESH MANDE   B25CE1205   CE3   C

Research:

Searching is a foundational operation in computer science and software development, enabling efficient data retrieval from collections of records. In real-world applications — such as banking systems, healthcare databases, e-commerce platforms, and enterprise resource planning (ERP) tools — searching allows users to locate specific records quickly among large datasets.

The two most widely used searching techniques are **Linear Search** and **Binary Search**. Linear search, which checks each element sequentially, is simple to implement and works on unsorted data. It is commonly used in small datasets or when data cannot be pre-sorted due to frequent updates. Its time complexity is O(n), making it suitable for systems with limited scalability requirements.

Binary search, by contrast, requires data to be sorted beforehand and operates by repeatedly dividing the search interval in half. With a time complexity of O(log n), it is significantly faster than linear search for large datasets and is the standard in systems requiring high performance, such as database indexes, file systems, and API endpoints handling thousands of queries per second.

Beyond algorithmic methods, modern applications often use **indexed data structures** (e.g., hash tables, B-trees) and **database engines** (e.g., SQL with WHERE clauses) to optimize search performance. These systems automatically maintain indexes on frequently queried fields (e.g., user ID, product code), allowing near-instant lookups regardless of dataset size.

In user-facing applications, search functionality is typically designed around **user intent** — supporting partial matches, case-insensitive queries, and fuzzy matching (e.g., “john” finding “Johnathan”). This enhances usability and mirrors natural human behavior.

Ultimately, the choice of search method depends on data size, frequency of updates, performance requirements, and system architecture. While linear search remains prevalent in educational contexts and small-scale programs, industry-grade systems overwhelmingly rely on indexed and optimized search mechanisms to ensure responsiveness and scalability.

- https://www.geeksforgeeks.org/linear-search/
- https://www.geeksforgeeks.org/binary-search/
- https://en.wikipedia.org/wiki/Search_algorithm
- https://www.programiz.com/dsa/linear-search
- https://www.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search

Analyze:

The Cinema Booking System is a simple console-based program written in C language. The main purpose of this project is to manage movie ticket bookings in an organized way. In real life, similar systems are used in multiplexes, theaters, and online ticket booking platforms to store customer details, track reservations, and manage show timings without relying on manual registers or paperwork.

The program uses a structure called Cinema to store all booking-related information together. This structure contains fields like customer name, movie name, show time, number of seats, ticket price, and show date. Using a structure makes it easier to group related data as a single unit instead of creating separate variables for each detail. This approach is similar to how databases store records where each booking is like a row and each field is like a column.

The system works through a menu-driven interface that keeps running until the user chooses to exit. It offers four main options — adding new bookings, displaying all bookings, sorting records, and searching for a specific booking. When the user selects add booking, the program takes input and stores it in an array of structures. The display option prints all saved records one by one so the user can view complete booking details easily.

For sorting, the program uses Bubble Sort algorithm which arranges bookings from lowest to highest ticket price. This helps users see cheaper and expensive bookings at a glance. Although bubble sort is not the fastest method, it is easy to understand and works well for small amounts of data. For searching, the program uses Linear Search which goes through each record and compares movie names using the strcmp function. If a match is found it displays the details, otherwise it shows a not found message.

Overall, this project covers important programming concepts like structures, arrays, loops, switch-case, sorting, and searching. It is a practical example of how basic programming knowledge can be used to build useful applications that solve real-world problems like managing cinema bookings in a simple and efficient way.

Ideation

The main objective and idea is to create an intuitive booking interface:

- **Add Bookings**: Allow users to input details like customer name, movie name, show time, number of seats, cost, and date.

- **Display All Bookings**: Provide a list of all reservations for viewing.

- **Sort Bookings**: Implement a basic sorting algorithm to order bookings by price.

- **Search Bookings**: Enable searching through the booking by sorting them with the help of the values

Algorithm:

1.Start

2.Initialize Variables

.

3.Input User Details

4.Input Items (Case 1 - Add)

5.Display Bookings (Case 2)

6.Sort Bookings (Case 3)

.Exit (Case 5)

Build:

```
#include <stdio.h>
#include <stdlib.h>

struct Cinema {
    char customer_name[20];
    char movie_name[20];
    char show_time[15];
    int seats;
    int ticket_price;
    int date;    // DDMMYYYY format
} booking[100], temp;

int n = 0;

int main() {
    int i, j, choice, flag = 0;

    while (1) {
        printf("\nmenu\n1.ADD RECORD\n2.DISPLAY RECORD\n3.SORT\n4.EXIT\nENTER YOUR
CHOICE CODE: ");
```

```
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("\nHow many cinema bookings do you want to add?: ");
                scanf("%d", &n);
                for (i = 0; i < n; i++) {
                    printf("\n--- ENTER DETAILS FOR BOOKING %d ---\n", i + 1);

                    printf("ENTER CUSTOMER NAME: ");
                    scanf("%s", booking[i].customer_name);

                    printf("ENTER MOVIE NAME: ");
                    scanf("%s", booking[i].movie_name);

                    printf("ENTER SHOW TIME: ");
                    scanf("%s", booking[i].show_time);

                    printf("ENTER NUMBER OF SEATS: ");
                    scanf("%d", &booking[i].seats);

                    printf("ENTER TOTAL TICKET PRICE: ");
                    scanf("%d", &booking[i].ticket_price);

                    printf("ENTER SHOW DATE (DDMMYYYY): ");
                    scanf("%d", &booking[i].date);
                }
                break;

            case 2:
                if (n == 0) {
                    printf("\nNo records to display. Add some first.\n");
                    break;
                }
                for (i = 0; i < n; i++) {

printf("\n==========+==========+==========+==========+==========+==========");
                    printf("\nCUSTOMER NAME: %s", booking[i].customer_name);
                    printf("\nMOVIE NAME: %s", booking[i].movie_name);
                    printf("\nSHOW TIME: %s", booking[i].show_time);
                    printf("\nNUMBER OF SEATS: %d", booking[i].seats);
                    printf("\nTOTAL TICKET PRICE: %d", booking[i].ticket_price);
                    printf("\nSHOW DATE (DDMMYYYY): %d", booking[i].date);
                }
            break;

          case 3:
            if (n == 0) {
               printf("\nNo records to sort. Add some first.\n");
               break;
            }
            for (i = 0; i < n - 1; i++) {
               for (j = 0; j < n - i - 1; j++) {
                  if (booking[j].ticket_price > booking[j + 1].ticket_price) {
                     temp = booking[j];
                     booking[j] = booking[j + 1];
                     booking[j + 1] = temp;
                  }
               }
            }
            printf("\nBOOKINGS ARE SORTED BY PRICE AND WILL BE DISPLAYED ON NEXT
DISPLAY OPTION.....");
            break;

          case 4:
            printf("\nThank you for using the Cinema Booking System. Goodbye!\n");
            exit(0);
            break;

          default:
            printf("\nInvalid choice! Please enter a valid option (1-4).\n");
       }
    }

    return 0;
}
```

Output

```
menu
1.ADD RECORD
2.DISPLAY RECORD
3.SORT
4.EXIT
ENTER YOUR CHOICE CODE: 1
```

```
How many cinema bookings do you want to add?: 2

--- ENTER DETAILS FOR BOOKING 1 ---
ENTER CUSTOMER NAME: sarvesh
ENTER MOVIE NAME: kantara
ENTER SHOW TIME: 2.00
ENTER NUMBER OF SEATS: 2
ENTER TOTAL TICKET PRICE: 400
ENTER SHOW DATE (DDMMYYYY): 11122025

--- ENTER DETAILS FOR BOOKING 2 ---
ENTER CUSTOMER NAME: om
ENTER MOVIE NAME: singham
ENTER SHOW TIME: 5.00
ENTER NUMBER OF SEATS: 5
ENTER TOTAL TICKET PRICE: 1000
ENTER SHOW DATE (DDMMYYYY): 12122025

menu
1.ADD RECORD
2.DISPLAY RECORD
3.SORT
4.EXIT
ENTER YOUR CHOICE CODE: 3

BOOKINGS ARE SORTED BY PRICE AND WILL BE DISPLAYED ON NEXT DISPLAY
OPTION.....
menu
1.ADD RECORD
2.DISPLAY RECORD
3.SORT
4.EXIT
ENTER YOUR CHOICE CODE: 2

==========+==========+==========+==========+==========+==========
CUSTOMER NAME: sarvesh
MOVIE NAME: kantara
SHOW TIME: 2.00
NUMBER OF SEATS: 2
TOTAL TICKET PRICE: 400
```

```
SHOW DATE (DDMMYYYY): 11122025
==========+==========+==========+==========+==========+==========
CUSTOMER NAME: om
MOVIE NAME: singham
SHOW TIME: 5.00
NUMBER OF SEATS: 5
TOTAL TICKET PRICE: 1000
SHOW DATE (DDMMYYYY): 12122025
menu
1.ADD RECORD
2.DISPLAY RECORD
3.SORT
4.EXIT
ENTER YOUR CHOICE CODE:
```

Implementation

https://github.com/sarvesh00719/cinema-database-manager-with-the-sorting-and-options.git