Movie Recommendation System

Submitted by

Sabrina Manahil Waseem (SE-23006)

Muqaddas Mehboob (SE-23008)

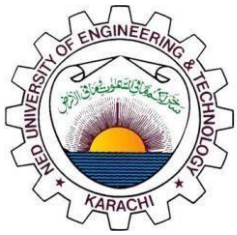
Zunaira Anwar (SE-23014)

Submitted to

Miss Marvi Jokhio

Complex Engineering Problem

November 21, 2024



Department Of Software Engineering

NED University Of Engineering & Technology

Abstract

Our Movie Recommendation System is a console application made in C++ to help users find and explore movies easily. Users can log in/ Sign Up and then search for movies by IMDb rating, genre, release year, or director. The application also allows users to add movies to a favorites list, get random movie suggestions with a "Surprise Me" option, and sort search results. Additionally, users can view detailed descriptions of any movie they select from search results.

Problem Statement:

Users often spend a lot of time searching through long lists of movies, struggling to find content they will enjoy. This project aims to build a system that suggests movies based on the user's preferences, helping them discover new content with minimal effort.

Features/ Operation

Our Movie Recommendation System will have the following features and operations:

- **Signup/Login:** This allows users to sign up and log in to access the application.

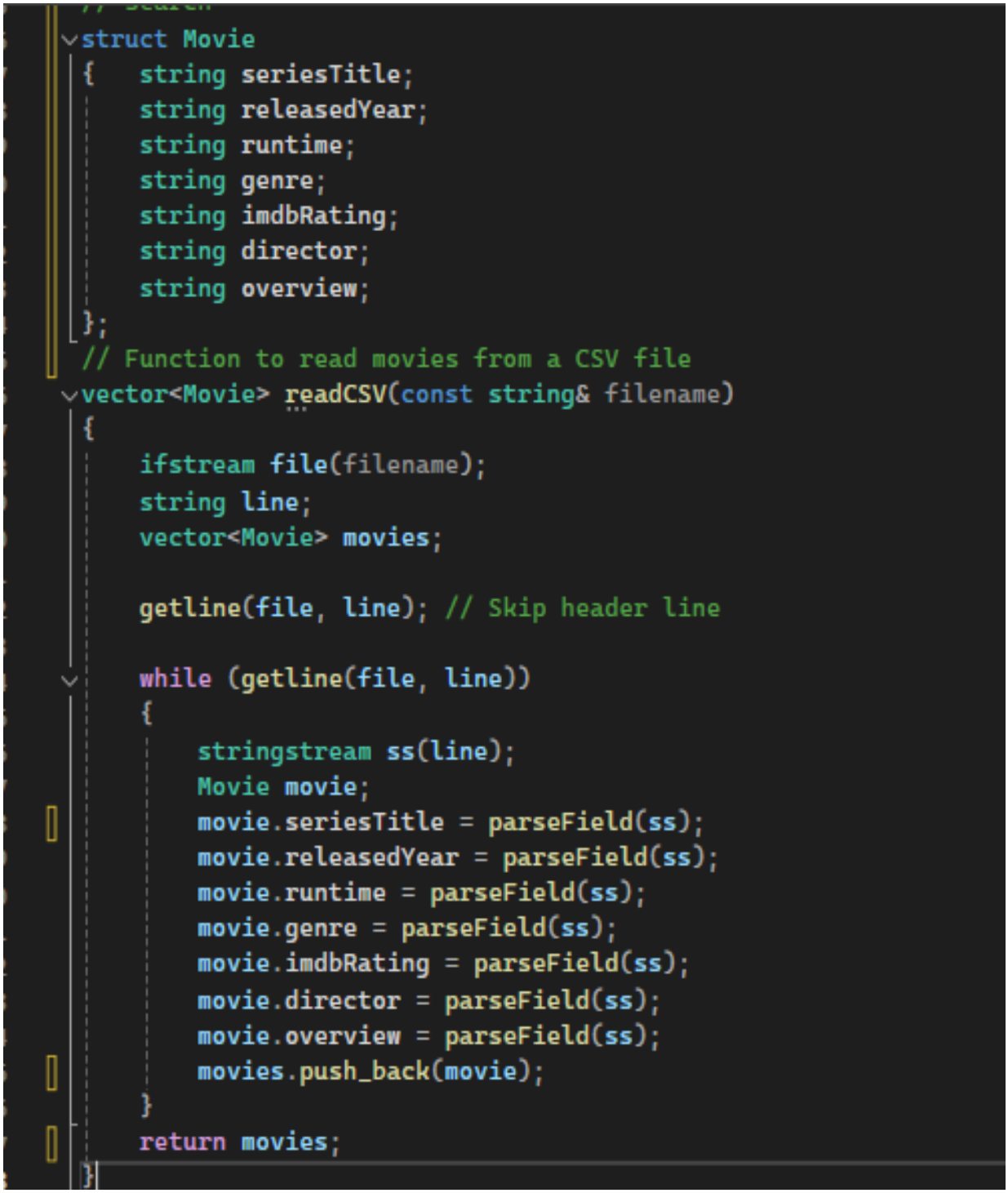
```
// Signup
bool signup(const string& username, const string& password)
{    ifstream infile("users.txt");
    if (!infile)
    {   setTextColor(12); // Red
        cout << "Error: Could not open users file.\n";
        return false;
    }
    string user, pass;
    while (infile >> user >> pass)
    {
        if (user == username)
        {
            setTextColor(12); // Red
            system("cls");
            cout << "\nUser already exists. Try a different username.\n";
            return false;
        }}
    infile.close();
    ofstream outfile("users.txt", ios::app);
    if (!outfile)
    {
        setTextColor(12); // Red
        cout << "Error: Could not open users file to write.\n";
        return false;    }
    outfile << username << " " << password << endl;
    setTextColor(10); // Green
    system("cls");
    cout << "\nSignup Successful!\n";
    return true;
}
```

```cpp
// Login
bool login(const string& username, const string& password)
{
    ifstream infile("users.txt");
    if (!infile)
    {
        setTextColor(12); // Red
        cout << "Error: Could not open users file.\n";
        return false;
    }

    string user, pass;
    while (infile >> user >> pass)
    {
        if (user == username && pass == password)
        {
            setTextColor(10); // Green
            cout << "\nLogin successful! Welcome, " << username << "!\n";
            return true;
        }
    }

    setTextColor(12); // Red
    system("cls");
    cout << "\nInvalid username or password.\n";
    return false;
}
```

```
do
{
    int choice;
    string username, password;

    setTextColor(9); // Blue
    cout << "Please choose an option:\n";
    setTextColor(11); // Light cyan
    cout << "1. Signup\n";
    cout << "2. Login\n";
    cout << "3. Exit\n\n";

    setTextColor(13); // Light purple
    cout << "Enter your choice (1-3): ";
    cin >> choice;

    if (choice == 1 || choice == 2)
    {
        setTextColor(14);
        cout << "\nEnter username: ";
        cin >> username;
        cout << "Enter password: ";
        cin >> password;
    }

    setTextColor(13); // Light purple
    if (choice == 1)
    {
        cout << "\nAttempting to Signup...\n";
        bool sign = signup(username, password);
        if (sign)
        {
            setTextColor(14); // Yellow
            cout << endl;
```

```
            cout << endl;
            cout << "Login to your account " << endl;
            cout << "\nEnter username: ";
            cin >> username;
            cout << "Enter password: ";
            cin >> password;
            cout << "\nAttempting to Login...\n";
            is_successful = login(username, password);
        }
    }
    else if (choice == 2)
    {
        cout << "\nAttempting to Login...\n";
        is_successful = login(username, password);
    }
    else if (choice == 3)
    {
        setTextColor(10); // Green
        system("cls");
        cout << "\nExiting the program. Goodbye!\n";
        exit(0); // Terminates the program and closes the console
        break;
    }
    else
    {
        setTextColor(12); // Red
        system("cls");
        cout << "Invalid choice. Please enter 1 for Signup, 2 for Login, or 3 to Exit.\n";
    }

    if (!is_successful)
    {
        setTextColor(9); // Blue
        cout << "\nPlease try again.\n\n";
    }

} while (!is_successful);
```

- **Search Movies**: Search for movies based on IMDb rating, genre, release year, or director.



```
struct Movie
{   string seriesTitle;
    string releasedYear;
    string runtime;
    string genre;
    string imdbRating;
    string director;
    string overview;
};
// Function to read movies from a CSV file
vector<Movie> readCSV(const string& filename)
{
    ifstream file(filename);
    string line;
    vector<Movie> movies;

    getline(file, line); // Skip header line

    while (getline(file, line))
    {
        stringstream ss(line);
        Movie movie;
        movie.seriesTitle = parseField(ss);
        movie.releasedYear = parseField(ss);
        movie.runtime = parseField(ss);
        movie.genre = parseField(ss);
        movie.imdbRating = parseField(ss);
        movie.director = parseField(ss);
        movie.overview = parseField(ss);
        movies.push_back(movie);
    }
    return movies;
}
```

```cpp
// Function to search and display movie based on user criteria
void search()
{
    string filename = "E:\\movies.csv";
    vector<Movie> movies = readCSV(filename);

    // Set title color (e.g., blue text on black background)
    setTextColor(14); // Blue text
    cout << "Select search option:\n";
    cout << "1. Search by Genre\n";
    cout << "2. Search by IMDb Rating\n";
    cout << "3. Search by Released Year\n";
    cout << "4. Search by Director\n"
        << endl;
    setTextColor(13);
    cout << "Enter choice (1-4): ";

    int choice;
    cin >> choice;
    cin.ignore(); // Clear the input buffer

    string searchValue, criteria;
    cout << endl;
    setTextColor(11); // Light cyan
    switch (choice)
    {
    case 1:
        criteria = "genre";
        cout << "Enter genre to search for (e.g., 'Drama', 'Comedy'): ";
        break;
    case 2:
        criteria = "imdb";
        cout << "Enter IMDb rating to search for (e.g., '8.5'): ";
        break;
    case 3:
```

```cpp
    case 3:
        criteria = "year";
        cout << "Enter released year to search for (e.g., '2020'): ";
        break;
    case 4:
        criteria = "director";
        cout << "Enter director's name to search for: ";
        break;
    default:
        cout << "Invalid choice. Exiting.\n";
        return;
    }

    getline(cin, searchValue);
    vector<Movie> filteredMovies = searchMovies(movies, criteria, searchValue);

    if (!filteredMovies.empty())
    {
        // Display the results with colored table
        displayTable(filteredMovies);
```

- **View Movie Details**: Display detailed descriptions of selected movies from the search results.

```
void viewMovieDescription(const vector<Movie>& movies)
{
    setTextColor(11);
    cout << "\nDo you want to view the description of any movie from the search results? (yes/no): ";
    string userResponse;
    setTextColor(13);
    cout <<endl<<endl<< "Enter choice : ";
    setTextColor(13);
    cin >> userResponse;
    setTextColor(11);
    if (toLowerCase(trim(userResponse)) == "yes")
    {
        cout <<endl<< "Enter the title of the movie: ";
        cin.ignore();
        string movieTitle;
        getline(cin, movieTitle);
        // Linear search for the movie by title
        bool found = false;
        for (const auto& movie : movies)
        {
            if (toLowerCase(trim(movie.seriesTitle)) == toLowerCase(trim(movieTitle)))
            {
                setTextColor(10); // Green
                cout << "\nOverview: " << (!movie.overview.empty() ? movie.overview : "No description available.") << "\n";
                found = true;
                break;
            }
        }
        if (!found)
        { cout << "\nMovie not found in the dataset. Please ensure the title is correct.\n"; }
    } else { }
}
```

```
        // Ask if the user wants to view the description of a movie
        viewMovieDescription(filteredMovies);
        setTextColor(14);
        cout << endl
            << "Do you want to go to Option Menu page? " << endl
            << " 1. Yes " << endl
            << " 2. No" << endl
            << " 3. Exit" << endl;
        int ans; cout << endl;
        setTextColor(13);
        cout << endl << "Enter your choice: (1-3) ";
         cin >> ans;
        if (ans == 1)
        {  display();
        }
        else if (ans == 2)
        {   setTextColor(10); // green
            cout << endl
                << "Have a nice day. Enjoy your movie :) ";
        }
        else if (ans == 3)
        {   system("cls");
            setTextColor(10); // green
            cout << "\nExiting the program. Goodbye!\n";
        }
        else
        {    system("cls");
            setTextColor(12); // Red text for error message
            cout << "Invalid choice. Please enter 1 for yes, 2 for no, or 3 to Exit.\n";
        }
    }
    else
    {
        setTextColor(12); // Red text for error message
        cout << "\nNo movies found for the selected criteria.\n";
        setTextColor(7); // Reset to default color
```

- **Add to Favorites**: Save favorite movies to remember later.

```
// circular queue for add to favorite feature
class CircularQueue
{
private:
    string* queue;
    int front;
    int rear;
    int capacity;
    int count;

public:
    // Constructor
    CircularQueue(int size)
    {   capacity = size;
        queue = new string[capacity];
        front = -1;
        rear = -1;
        count = 0;
    }
    // Destructor
    ~CircularQueue(){  delete[] queue;   }
    // Add movie to favorites
    void addToFavorite(const string& movieName)
    {  if (count == capacity) { increaseCapacity(); }
        if (count == 0){ front = 0;}
        rear = (rear + 1) % capacity;
        queue[rear] = movieName;
        count++;
        setTextColor(3);
        cout<<endl << "Movie \"" << movieName << "\" added to favorites.\n";
    }
```

- **Manage and View Favorites list**: Display user movies they added to the favorites list.

```
// Remove movie from favorites
void removeFromFavorite()
{
    if (count == 0)
    {   setTextColor(3);
        cout << endl << "Favorites list is empty. Nothing to remove.\n";
        return;
    }
    cout << endl << "Enter the name of the movie to remove: ";
    string movieName;
    getline(cin, movieName); // Get the movie name from the user

    bool found = false;
    int newRear = front; // Temporary variable to calculate the new rear
    for (int i = 0; i < count; i++)
    {
        int index = (front + i) % capacity;
        if (queue[index] == movieName)
        {  found = true;
            cout << endl << "Movie \"" << movieName << "\" removed from favorites.\n";
            continue; // Skip adding this element to the new queue
        }
        queue[newRear] = queue[index];
        newRear = (newRear + 1) % capacity;
    }
    if (found)
    {  rear = (newRear - 1 + capacity) % capacity; // Update rear
        count--;                                    // Decrease count after removal
    }
    else
    {  cout << endl << "Movie \"" << movieName << "\" not found in favorites.\n"; }
}
```

```
    // Display all movies in favorites
    void displayFavorites() const
    {
        if (count == 0)
        {
            setTextColor(12);
            cout <<endl<< "Your favorites list is empty.\n";
            return;
        }
        setTextColor(10);
        cout<<endl << endl << "Your Favorites List:\n";
        for (int i = 0; i < count; i++)
        {
            cout << "- " << queue[(front + i) % capacity] << endl;
        }
    }
};
```

```
void handleFavorites()
{    CircularQueue favorites(20); // CircularQueue object
    string movieName;
    int choice = 1;
    while (choice == 1 ||choice == 2 ||choice == 3 ||choice == 4 )
    { // Loop for displaying the menu repeatedly
        setTextColor(11); // Yellow
        cout << "\n--- Favorites Options ---\n";
        cout << endl;
        setTextColor(14);
        cout << "1. Add movie to favorites\n";
        cout << "2. Remove movie from favorites\n";
        cout << "3. Display favorites list\n";
        cout << "4. Go back to menu\n";
        setTextColor(13); // Light purple
        cout<<endl << "Enter your choice: (1-4) ";
        cin >> choice;
        cin.ignore(); // To ignore the newline character after choice input
        switch (choice)
        {   case 1:
            setTextColor(3);
            cout <<endl<< "Enter the name of the movie to add: ";
            getline(cin, movieName);
            favorites.addToFavorite(movieName); break;
        case 2:
            favorites.removeFromFavorite(); break;
        case 3:
            setTextColor(3);
            favorites.displayFavorites(); break;
        case 4:
            system("cls");
            display(); break;
        default:
            setTextColor(12); // Red
            cout << "Invalid choice. Please try again.\n";
        }
    }
```

- **Surprise Me**: Get random movie recommendations.

```
// class surpriseMovie for the surprise me feature objects
class SurpriseMovie {
private:
    string name;
    string director;

public:
    // Constructor
    SurpriseMovie(string movieName, string movieDirector)
        : name(movieName), director(movieDirector) {}

    // Getters
    string getName() const { return name; }
    string getDirector() const { return director; }
};
```

```
// Function to surprise the user with 20 random movies
void surpriseMe() {
    ifstream file("E:\\movies.csv");
    if (!file.is_open()) {
        cerr << "Error opening file!" << endl;
        return;
    }

    vector<SurpriseMovie> movies;
    string line;

    getline(file, line);

    while (getline(file, line)) {
        vector<string> fields = parseCSVLine(line);

        if (fields.size() >= 7) {
            string name = fields[0];
            string director = fields[5];
            movies.emplace_back(name, director);
        }
    }
    file.close();
    srand(time(0));
    vector<SurpriseMovie> randomMovies;
    while (randomMovies.size() < 20 && !movies.empty()) {
        int index = rand() % movies.size();
        randomMovies.push_back(movies[index]);
        movies.erase(movies.begin() + index);
    }
    Stack movieStack;
```

```
// class stack to implement surprise me
class Stack {
private:
    vector<SurpriseMovie> stackArray;
    int top;

public:
    Stack() : top(-1) {}

    // Push a movie onto the stack
    void push(const SurpriseMovie& movie) {
        stackArray.push_back(movie);
        top++;
    }

    // Pop a movie from the stack
    SurpriseMovie pop() {
        if (top == -1) {
            throw out_of_range("No movies to recommend");
        }
        SurpriseMovie movie = stackArray[top];
        stackArray.pop_back();
        top--;
        return movie;
    }

    // Check if the stack is empty
    bool isEmpty() const {
        return top == -1;
    }
};
```

```
}
Stack movieStack;

for (const auto& movie : randomMovies) {
    movieStack.push(movie);
}
```

- **Sort Results**: Sort the search results by IMDb rating, alphabetically, released year and run time.

```cpp
// Sorting functions
bool compareByTitle(const Movie& a, const Movie& b)
{return a.seriesTitle < b.seriesTitle;}

bool compareByYear(const Movie& a, const Movie& b)
{    return a.releasedYear < b.releasedYear; }

bool compareByImdbRating(const Movie& a, const Movie& b)
{    return a.imdbRating < b.imdbRating; }

bool compareByRuntime(const Movie& a, const Movie& b)
{     return a.runtime < b.runtime; }
```

```cpp
// Merge Sort Implementation
void merge(vector<Movie>& movies, int left, int mid, int right, bool (*compare)(const Movie&, const Movie&)) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    vector<Movie> leftMovies(n1), rightMovies(n2);

    for (int i = 0; i < n1; ++i)
        leftMovies[i] = movies[left + i];
    for (int i = 0; i < n2; ++i)
        rightMovies[i] = movies[mid + 1 + i];

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        if (compare(leftMovies[i], rightMovies[j])) {
            movies[k] = leftMovies[i];
            ++i;
        }
        else {
            movies[k] = rightMovies[j];
            ++j;
        } ++k;
    }
    while (i < n1) {
        movies[k] = leftMovies[i];
        ++i;  ++k;
    }
    while (j < n2) {
        movies[k] = rightMovies[j];
        ++j;  ++k;
    }
}
```

```cpp
void mergeSort(vector<Movie>& movies, int left, int right, bool (*compare)(const Movie&, const Movie&)) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(movies, left, mid, compare);
        mergeSort(movies, mid + 1, right, compare);

        merge(movies, left, mid, right, compare);
    }
}

// Wrapper function for merge sort
void sortMovies(vector<Movie>& movies, bool (*compare)(const Movie&, const Movie&)) {
    mergeSort(movies, 0, movies.size() - 1, compare);
}
```

```cpp
        << endl
        << "Do you want to sort the results?" << endl
        << "1. Yes " << endl
        << "2. No" << endl
        << "3. Exit" << endl;
    int sortResponse;
    setTextColor(13);
    cout<<endl << "Enter choice (1-3): ";
    cin >> sortResponse;
    if (sortResponse == 1)
    {
        setTextColor(13);
        cout << endl<<endl;
        setTextColor(3);
        // Ask for sorting criteria after displaying results
        cout << "Select sorting criteria:\n";
        cout << "1. Alphabetically by Title\n";
        cout << "2. By Released Year\n";
        cout << "3. By IMDb Rating\n";
        cout << "4. By Runtime\n"
            << endl;
        setTextColor(13);
        cout<<endl << "Enter choice (1-4): ";

        int sortChoice;
        cin >> sortChoice;

        if (sortChoice == 1) {
            sortMovies(filteredMovies, compareByTitle);
            displayTable(filteredMovies);
        }
        else if (sortChoice == 2) {
            sortMovies(filteredMovies, compareByYear);
            displayTable(filteredMovies);
        }
```

```
        }
        else if (sortChoice == 3) {
            sortMovies(filteredMovies, compareByImdbRating);
            displayTable(filteredMovies);
        }
        else if (sortChoice == 4) {
            sortMovies(filteredMovies, compareByRuntime);
            displayTable(filteredMovies);
        }
        else
        {
            setTextColor(12); // Red text for error message
            cout << "Invalid choice. Please enter 1 , 2 , 3 or 4.\n";
        }
    }
    else if (sortResponse == 2)
    {
        setTextColor(10); // green
        cout << endl
            << "Have a nice day. Enjoy your movie :) " << endl;
    }

    else if (sortResponse == 3)
    {
        system("cls");
        setTextColor(10); // green
        cout << "\nExiting the program. Goodbye!\n";
    }
    else
    {
        system("cls");
        setTextColor(12); // Red text for error message
        cout << "Invalid choice. Please enter 1 for yes, 2 for no, or 3 to Exit.\n";

    }
```

Data Structures & Algorithms

- **Stack:** Store random movie suggestions, allowing a "last in, first out" retrieval of randomly recommended movies.

- **Circular Queue:** Implements a "View Favorites" feature, allowing users to add or remove movies. Favorite movies are displayed in the order they were added, following a first-in, first-out (FIFO) approach. However, the "remove from favorites" function does not follow the traditional FIFO behavior. Instead, it linearly searches for the specified movie in the queue, removes it upon finding, and shifts the remaining movies accordingly.

- **Linear Search Algorithm:** Retrieves movie data from the dataset by checking each movie sequentially. This helps in better and more efficient retrieval of movies along with their details in this system. Linear Search is also used in circular queue’s class remove from favorite function for searching.

- **Merge sort Algorithm:** Sorts all the data of the table based on the selected criteria after searching.