Music Playlist Generator

EXECUTIVE SUMMARY

The main point of this program is to demonstrate the use of sorting and searching algorithms in the context of a music playlist generator. The program allows the user to input their playlist, which consists of songs with details such as title, artist, genre, and duration.

The program first sorts the playlist based on the duration of the songs using the insertion sort algorithm. This ensures that the songs are arranged in ascending order of duration, creating a cohesive playlist.

After sorting, the program provides the user with options to search for songs based on different criteria, such as artist, duration, song title, or genre. The program then performs the search operation using linear search and displays the songs that match the user's search criteria, including the index positions of the songs in the playlist.

The purpose of this program is to showcase how sorting and searching algorithms can be applied to organise and retrieve data effectively. It simulates a music playlist generator that allows users to create a personalised playlist and search for songs based on their preferences, providing a practical example of algorithmic concepts in a real-world scenario.

Objectives and Goals

The objective of this program is to create a music playlist generator that allows users to:

- Input their playlist with a maximum of 100 songs
- Sort the playlist
- Search for songs
- Display search results

By achieving these objectives, the program aims to provide users with a personalised music playlist based on their preferences and facilitate easy retrieval of songs based on various search criteria. And the ultimate goal of the program is to provide users with a tool to generate a music playlist tailored to their preferences, ensuring a pleasant listening experience. By incorporating sorting and searching algorithms, the program helps organise and retrieve songs efficiently, allowing users to navigate and explore their playlist with ease2nd June

Project Outline

1. Welcome Message:
Display a welcome message to greet the user and introduce the Music Playlist Generator.
2. Input the Playlist:
a. Prompt the user to enter the number of songs in their playlist.
b. Iterate over the number of songs and prompt the user to enter the details for each song (title, artist, genre, and duration).
3. Sort the Playlist:
a. Implement the insertion sort algorithm to sort the playlist based on the duration of the songs.
b. Display the sorted playlist, including the song details (title, artist, genre, and duration).
4. Search for Songs:
a. Present options to the user for searching songs based on different criteria (artist, duration, song title, or genre).
b. Based on the user's choice, prompt them to enter the search value.
c. Perform the search operation using appropriate algorithms and display the search results, including the index positions and details of the matching songs.
d. If no songs match the search criteria, display a suitable message indicating the absence of matching songs.
5. Program Termination:
End the program or provide an option to restart the process if the user wishes to generate a new playlist.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SONGS 100
struct Song
{
char title[50];
char artist[50];
char genre[50];
int duration;
};2nd June
void insertionSort(struct Song playlist[], int n)
{
int i,j;
struct Song key;
for (i = 1; i < n; i++)
{
key = playlist[i];
j = i - 1;
while (j >= 0 && playlist[j].duration > key.duration)
{
playlist[j + 1] = playlist[j];
j = j - 1;
}
playlist[j + 1] = key;
}
}
void searchByArtist(struct Song playlist[], int n, char artist[])
{
int found = 0;
for (int i = 0; i < n; i++)
{
if (strcmp(playlist[i].artist, artist) == 0)
{
printf("Found at Index Value: %d\n", i);
printf("Title: %s\n", playlist[i].title);
printf("Artist: %s\n", playlist[i].artist);
printf("Genre: %s\n", playlist[i].genre);
printf("Duration: %d seconds\n\n", playlist[i].duration);
found = 1;2nd June
}
}
if (!found)
{
printf("No songs found for artist: %s\n", artist);
}
}
void searchByDuration(struct Song playlist[], int n, int duration)
{
int found = 0;
for (int i = 0; i < n; i++)
{
if (playlist[i].duration == duration)
{
printf("Found at Index Value: %d\n", i);
printf("Title: %s\n", playlist[i].title);
printf("Artist: %s\n", playlist[i].artist);
printf("Genre: %s\n", playlist[i].genre);
printf("Duration: %d seconds\n\n", playlist[i].duration);
found = 1;
}
}
if (!found)
{
```

```
printf("No songs found for duration: %d seconds\n", duration);
}
}
void searchByTitle(struct Song playlist[], int n, char title[])
{2nd June
int found = 0;
for (int i = 0; i < n; i++)
{
if (strcmp(playlist[i].title, title) == 0)
{
printf("Found at Index Value: %d\n", i);
printf("Title: %s\n", playlist[i].title);
printf("Artist: %s\n", playlist[i].artist);
printf("Genre: %s\n", playlist[i].genre);
printf("Duration: %d seconds\n\n", playlist[i].duration);
found = 1;
}
}
if (!found)
{
printf("No songs found with title: %s\n", title);
}
}
void searchByGenre(struct Song playlist[], int n, char genre[])
{
int found = 0;
for (int i = 0; i < n; i++)
{
if (strcmp(playlist[i].genre, genre) == 0)
{
printf("Found at Index Value: %d\n", i);
printf("Title: %s\n", playlist[i].title);
printf("Artist: %s\n", playlist[i].artist);
printf("Genre: %s\n", playlist[i].genre);
printf("Duration: %d seconds\n\n", playlist[i].duration);2nd June
found = 1;
}
}
if (!found)
{
printf("No songs found for genre: %s\n", genre);
}
}
int main()
{
struct Song playlist[MAX_SONGS];
int numSongs;
printf("Welcome to the Music Playlist Generator!\n\n");
printf("Enter the number of songs in your playlist (maximum %d): ",
MAX_SONGS);
scanf("%d", &numSongs);
printf("\nEnter the details for each song:\n");
for (int i = 0; i < numSongs; i++)
{
printf("\nSong %d:\n", i + 1);
printf("Title: ");
scanf("%s", playlist[i].title);
printf("Artist: ");
scanf("%s", playlist[i].artist);
printf("Genre: ");2nd June
scanf("%s", playlist[i].genre);
printf("Duration (in seconds): ");
scanf("%d", &playlist[i].duration);
}
insertionSort(playlist, numSongs);
```

```
printf("\nSorted Playlist (by duration):\n");
for (int i = 0; i < numSongs; i++)
{
printf("Found at Index Value: %d\n", i);
printf("Title: %s\n", playlist[i].title);
printf("Artist: %s\n", playlist[i].artist);
printf("Genre: %s\n", playlist[i].genre);
printf("Duration: %d seconds\n\n", playlist[i].duration);
}
int option;
char searchValue[50];
printf("Search for songs by:\n");
printf("1. Artist\n");
printf("2. Duration\n");
printf("3. Song Title\n");
printf("4. Genre\n");
printf("Enter your option: ");
scanf("%d", &option);
printf("Enter the search value: ");
scanf("%s", searchValue);2nd June
switch (option)
{
case 1:
searchByArtist(playlist, numSongs, searchValue);
break;
case 2:
searchByDuration(playlist, numSongs, atoi(searchValue));
break;
case 3:
searchByTitle(playlist, numSongs, searchValue);
break;
case 4:
searchByGenre(playlist, numSongs, searchValue);
break;
default:
printf("Invalid option.\n");
break;
}
return 0;
}2nd June
```

Output2nd June

Time Complexity

The searching and sorting techniques used for this program are:

Linear Search:

- The program performs a linear search to find songs that match the user's search criteria. It iterates through the playlist array and compares the desired field (artist, duration, title, or genre) with the search value.
- In this program this has been used for searchbyArtist, searchbyDuration, searchbyTitle and searchbyGenre.
- **Time Complexity**

BEST CASE: O[1]

AVERAGE CASE: O[N]

WORST CASE: O[N]2nd June

Insertion Sort:

- The program sorts the playlist based on the duration of songs using the insertion sort algorithm. This algorithm compares each element with the elements before it and inserts it at the correct position in the sorted subarray.
- In this program it's been used for sorting the playlist with respect to their Duration.

- **Time Complexity**

BEST CASE: O[N]
AVERAGE CASE: O[N^2]
WORST CASE: O[N^2]

Conclusion

In conclusion, the Music Playlist Generator project demonstrates the use of sorting and searching techniques to create a personalized music playlist based on user preferences. The program allows the user to input song details such as title, artist, genre, and duration. The playlist is then sorted based on the duration of songs using the insertion sort algorithm. The user can search for songs in the playlist using various criteria such as artist, duration, title, or genre. The project highlights the importance of sorting algorithms in organizing data and the significance of searching algorithms in retrieving specific information. By incorporating these techniques, the program enables users to create cohesive and enjoyable playlists tailored to their preferences. The overall objective of the program is to provide a user-friendly interface for generating personalized music playlists and facilitating efficient searching based on user-defined criteria.