Project Title**: Music Player with Data Structures**

Problem Statement:

Develop a music player application that utilizes various data structures to efficiently manage and manipulate a large collection of music files. The application should allow users to search for songs, create playlists, and manage their recently played songs.

Data Structure Concepts:

1. **Linked List:** A linked list is a linear data structure that consists of a sequence of nodes, where each node contains a data element and a reference (or pointer) to the next node in the sequence. In this project, a linked list is used to implement playlists. Each node in the linked list represents a song in the playlist. The linked list allows for efficient insertion and deletion of songs from the playlist.

Functionalities:

- Insert Song: The `insertSong()` function takes a song object as input and inserts it into the playlist. The function traverses the linked list to find the appropriate position to insert the song.

- Delete Song: The `deleteSong()` function takes a song object as input and deletes it from the playlist. The function traverses the linked list to find the song to delete and removes the corresponding node.

2. **Stack:** A stack is a LIFO (Last In, First Out) data structure, meaning the last element added to the stack is the first one to be removed. In this project, a stack is used to manage the recently played songs. The stack allows for efficient pushing and popping of songs onto and off the stack.

Functionalities:

- Push Song: The `pushSong()` function takes a song object as input and adds it to the top of the stack. The function represents the act of adding a recently played song to the list.

- Pop Song: The `popSong()` function removes the top element from the stack and returns it. The function represents the act of removing the most recently played song from the list.

3. **Hash Table:** A hash table is a data structure that maps keys to values. It is used to efficiently store and retrieve data. In this project, a hash table is used to store the music library. The song name is used as the key, and the song object itself is used as the value.

   Functionalities:

   - Search Song: The `searchSong()` function takes a song name as input and searches for the corresponding song in the hash table. The hash table allows for constant-time lookup of songs.
   - Insert Song: The `insertSong()` function takes a song object as input and inserts it into the hash table. The hash function maps the song name to a hash value, which is used to determine the location of the song in the hash table.
   - Delete Song: The `deleteSong()` function takes a song name as input and deletes the corresponding song from the hash table. The hash function maps the song name to a hash value, which is used to locate the song in the hash table.

4. Tree: A tree is a hierarchical data structure consisting of nodes connected by edges. In this project, a tree can be used to represent the hierarchical organization of music genres and artists. The tree allows for efficient searching and browsing of music by genre and artist.

   Functionalities:

   - Browse Music: Users can browse music by genre or artist by traversing the corresponding tree structure. The tree allows for efficient navigation through the hierarchy of genres and artists.
   - Search Music: Users can search for music by genre, artist, or song title. The tree can be used to efficiently search for songs based on the specified criteria.

5. **Hashing:** Hashing is a technique for mapping keys to values. In this project, hashing is used to efficiently store and retrieve data from the hash table. The hash function takes a key as input and generates a hash value, which is used to determine the location of the key in the hash table.

   Functionalities:

   - Song Lookup: The hash function is used to efficiently locate songs in the hash table based on their names.

- Collision Resolution: When two different songs have the same hash value, collision resolution techniques are used to handle the conflict.

Efficiency Analysis:

1. **Space Complexity:**

- Hash Table: The space complexity of a hash table is O(n), where n is the number of songs in the music library. This is because the hash table needs to store all the songs in memory.

- Playlist: The space complexity of a playlist is O(n), where n is the number of songs in the playlist. This is because the linked list needs to store all the songs in the playlist in memory.

- Stack: The space complexity of a stack is O(n), where n is the number of recently played songs. This is because the stack needs to store all the recently played songs in memory.

2. **Time Complexity:**

- Search: The time complexity of searching for a song using the hash table is O(1), where 1 is a constant value. This is because the hash table allows for constant-time lookup of songs.

- Insert: The time complexity of inserting a song into the hash table is O(1), where 1 is a constant value. This is because the hash table allows for constant-time insertion of songs.

- Delete: The time complexity of deleting a song from the hash table is O(1), where 1 is a constant value. This is because the hash table allows for constant-time deletion of songs.

- Playlist Insertion: The time complexity of inserting a song into a playlist is O(1), where 1 is a constant value. This is because the linked list allows for constant-time insertion of songs at the end of the list.

- Playlist Deletion: The time complexity of deleting a song from a playlist is O(n), where n is the number of songs in the playlist. This is because the linked list needs to traverse the entire list to find the song to delete.

- Stack Push: The time complexity of pushing a song onto the stack is O(1), where 1 is a constant value. This is because the stack allows for constant-time pushing of songs onto the stack.

- Stack Pop: The time complexity of popping a song from the stack is O(1), where 1 is a constant value. This is because the stack allows for constant-time popping of songs from the stack.

Conclusion:

The use of appropriate data structures in the music player application significantly improves the efficiency of various operations, including searching, inserting, and deleting songs. The hash table provides constant-time lookups for songs, while the linked list allows for efficient insertion and deletion of songs from playlists. The stack efficiently manages the recently played songs. As a result, the application offers a responsive and user-friendly experience for managing and playing music