LAB ASSIGNMENT

Data Structure and Algorithm Design

Submitted By:

Chandan Kumar (2022SP93032, 2YA)
Harshwardhan Chougule (2022SP93078, 2YA)
Rushabh Porwal (2022SP93034, 2YA)
Afsar Hussain (2022SP93088, 2YA)
Prerna Sharma (2022SP93066, 2YA)

Submitted To:

Prof. Vadivellan M

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Problem Statement:

Implement a **Song Playlist ADT** using doubly linked list. Each node in the list should contain the song title and the duration of the song.

Following operations need to be implemented using the concept of doubly linked list.

1. Add a song to the playlist – adding from beginning or end of the DLL.
2. Delete a song from the playlist – delete a song based on a position in the linked list which is input through a prompt and ensure that the input position is valid.
3. Find a song by name – prompt for the song name. Exact name matching is fine. Partial match is not required.
4. Next track / Previous track – prompt for a starting position and then traverse forward and backward based on the user inputs. Ensure relevant message is displayed when the beginning and end of the playlist is reached.
5. Sort playlist by the song title – use one of the standard sorting algorithms.
6. Display playlist – display all the songs in the playlist.

Solution Provided:

Language Used: Java

Version: JDK11

Code Structure:

1. **Node Class**

   Contains basic node definition and getters & setters' methods as follows:

   - Node Definition – with variables song, duration, next node and previous node.
   - setSong() – setter method to set the song name.
   - setDuration() – setter method to set the duration of the song.
   - getSong() – getter method to get the song name.
   - getDuration() – getter method to get the song duration.

2. **LinkedList Class:**

   This class contains all the methods to perform different playlist operations. The list of methods with their details and complexity analysis is as follows:

Method	Details
addSong(String t, String dur)	**Description:** The function adds a song to the playlist at the end. First it checks whether the list is empty, then add the new song in the head node and mail Prev of head and tail to null. If the list is not empty, then add the song to the end of the list. **Time Complexity:** The time complexity of the add operation in the code is O (N), where n is the number of nodes in the list. The reason for this is due to searching for duplicate songs each time. Every time we add a new song it searches for a duplicate song in the playlist. **Space Complexity:** The space complexity of the add operation in the code is O (1), which means it requires constant space. The reason for this constant space complexity is that the amount of additional memory used by the code does not depend on the size of the input or the number of songs in the list. When adding a new song to the end of the list, a new **Node** object is created, and the necessary references are updated. However, this operation only requires a fixed amount of memory to store the new node object, regardless of the size of the list.
findSong(String t)	Description: The function finds a song from the playlist based on the given name and returns the position of the song in the playlist. First it checks whether the list is empty if yes, returns. Otherwise, it traverses through the playlist and searches for the song based on the name and if it's present, it returns the position. If the end of the playlist, i.e if it finds the tail node and doesn't find the song, then it returns a message indicating song is not present. Time Complexity: The time complexity of finding a node value in a linked list linearly is O(n), where n is the number of nodes in the list. This is because the algorithm must traverse the entire list to find the node with the desired value. Space complexity : The space complexity of finding a node value in a linked list is O(1). This is because the only extra space required is a pointer to the current node. The reason for this constant space complexity is that the amount of additional memory used by the

	code does not depend on the size of the input or the number of nodes in the list.
deleteAtPos(int pos)	**Description:** The function deletes a song from the playlist based on a given position. First it checks whether the list is empty if yes, returns. Otherwise, navigate to the position given from where we must delete the song and change the previous and next of the node. **Time Complexity:** The time complexity of the delete operation in the code is O(n), where n is the number of songs in the playlist. The reason for this time complexity is that the code needs to traverse the list to find the node at the specified position. In the worst-case scenario, the position may be at the end of the list, requiring traversal through all n nodes. **Space Complexity:** The space complexity of the delete operation in the provided code list is O (1), which means it requires constant space. The reason for this constant space complexity is that the amount of additional memory used by the code does not depend on the size of the input or the number of nodes in the list.
getSongAtPos(int pos)	**Description:** The function retrieves songs from the playlist based on a given position. It allows the user to navigate through the playlist by selecting 'N' for the next track, 'P' for the previous track, or 'E' to exit. The function iterates through the playlist until it reaches the desired position or the end of the playlist. It handles various scenarios such as an empty playlist, an invalid position, and provides user feedback for navigating through the playlist. It returns the song information of the current position in the playlist. **Time Complexity:** The time complexity of the code is O(n), where n is the number of nodes in the playlist. The method iterates through the playlist to reach the desired position or the end of the playlist, and the user interaction loop can also potentially traverse the entire playlist. **Space Complexity:** The space complexity of the code is O(n), where n is the number of nodes in the playlist. This is due to the memory required to store the

	playlist nodes. The auxiliary variables user in the code has a constant space requirement.
sortList()	**Description:** The function sort the data by using Quick Sort. We can recursively sort the linked list using pointers to first and last nodes of a doubly linked list. The partition function for a linked list is also similar to partition for arrays. Instead of returning index of the pivot element, it returns a pointer to the pivot element. **Time Complexity:** Time complexity of this function is same as time complexity of QuickSort() for arrays. It takes O(n^2) time in the worst case and O(nLogn) in average and best cases. The worst case occurs when the linked list is already sorted **Space Complexity:** The space complexity of the code is O(n), where n is the number of nodes in the playlist. O(n) The extra space is due to the function call stack.
display()	**Description:** This method prints the songs in the playlist along with their position, name, and duration. It iterates through the playlist, starting from the head, and prints the song information for each node until it reaches the end of the playlist. It also checks if the playlist is empty or not and displays the appropriate message accordingly. **Time Complexity:** The time complexity of the code is O(n), where n is the number of nodes in the playlist. This is because the code iterates through each node in the playlist once to display its information. The time taken to display the song is directly proportional to the number of nodes in the playlist. **Space Complexity:** The space complexity of the code is O (1) or constant. This is because the memory usage does not depend on the size of the playlist. The code only requires a constant amount of additional memory to store variables like “current”, “position”, and temporary values during

	execution. The memory usage remains constant regardless of the size of the playlist.

3. **SongPlaylist Class:**

This class contains the main function. It allows the users to create a playlist, add songs to it, delete songs, search for songs, retrieve songs position, sort the playlist by song title, and display the playlist. The class users a LinkedList data structure to manage the playlist. Users interact with the application through a menu-driven console interface.

Output:

1. **Adding New Song:**

```
Test Playlist - Playlist Operations :
1. Add a song to the playlist
2. Delete a song from the playlist
3. Find a song by name
4. Get Song by Position
5. Sort playlist by song title
6. Display playlist
0. Exit
Enter Menu Option :
1
*****Add New Song*****
Enter Song Title :
Shape of You
Enter Song Duration :
3:52
Added the song 'Shape of You' to the playlist successfully :)
Go Back to Menu (1) or Exit (0) :
```

2. **Display the Playlist:**

```
Test Playlist - Playlist Operations :
1. Add a song to the playlist
2. Delete a song from the playlist
3. Find a song by name
4. Get Song by Position
5. Sort playlist by song title
6. Display playlist
0. Exit
Enter Menu Option :
6
*****Displaying The Complete Playlist*****
Playlist Songs:
1. Shape of You  <3:52>
2. Kiki Do You Love Me  <3:39>
3. Titanium  <4:07>
4. Shotgun  <3:21>
5. Believer  <3:23>
Go Back to Menu (1) or Exit (0) :
```

3. **Delete Song from the Playlist:**

```
Test Playlist - Playlist Operations :
1. Add a song to the playlist
2. Delete a song from the playlist
3. Find a song by name
4. Get Song by Position
5. Sort playlist by song title
6. Display playlist
0. Exit
Enter Menu Option :
2
*****Delete A Song Based on Position*****
Enter Postion of Song to be Deleted :
4
Song Deleted from the playlist successfully :)
Go Back to Menu (1) or Exit (0) :
```

```
Test Playlist - Playlist Operations :
1. Add a song to the playlist
2. Delete a song from the playlist
3. Find a song by name
4. Get Song by Position
5. Sort playlist by song title
6. Display playlist
0. Exit
Enter Menu Option :
6
*****Displaying The Complete Playlist*****
Playlist Songs:
1. Shape of You  <3:52>
2. Kiki Do You Love Me  <3:39>
3. Titanium  <4:07>
4. Believer  <3:23>
Go Back to Menu (1) or Exit (0) :
```

4. **Search Song By Name:**

```
Test Playlist - Playlist Operations :
1. Add a song to the playlist
2. Delete a song from the playlist
3. Find a song by name
4. Get Song by Position
5. Sort playlist by song title
6. Display playlist
0. Exit
Enter Menu Option :
3
*****Find A Song from Playlist*****
Enter Name of Song to be Searched :
titanium
Position of Your Song is: 3
Current Song is  3. Titanium  < 4:07 >
Enter 'N' for next track, 'P' for previous track, or 'E' to exit:
```

5. **Sort Playlist By Song Name:**

```
Test Playlist - Playlist Operations :
1. Add a song to the playlist
2. Delete a song from the playlist
3. Find a song by name
4. Get Song by Position
5. Sort playlist by song title
6. Display playlist
0. Exit
Enter Menu Option :
5
*****Sorting The Playlist Based on Song Name*****
Playlist Sorted Successfully :)
 Check Display Playlist Option to check the Sorted Playlist Order.
Go Back to Menu (1) or Exit (0) :
```

```
Test Playlist - Playlist Operations :
1. Add a song to the playlist
2. Delete a song from the playlist
3. Find a song by name
4. Get Song by Position
5. Sort playlist by song title
6. Display playlist
0. Exit
Enter Menu Option :
6
*****Displaying The Complete Playlist*****
Playlist Songs:
1. Believer  <3:23>
2. Kiki Do You Love Me  <3:39>
3. Shape of You  <3:52>
4. Titanium  <4:07>
Go Back to Menu (1) or Exit (0) :
```

6. **Get Song Track at a Position in the Playlist:**

```
Test Playlist - Playlist Operations :
1. Add a song to the playlist
2. Delete a song from the playlist
3. Find a song by name
4. Get Song by Position
5. Sort playlist by song title
6. Display playlist
0. Exit
Enter Menu Option :
4
*****Get Song By Playlist*****
Enter Song Postion :
2
Current Song is  2. Kiki Do You Love Me  < 3:39 >
Enter 'N' for next track, 'P' for previous track, or 'E' to exit: N
Current Song is  3. Shape of You  < 3:52 >
Enter 'N' for next track, 'P' for previous track, or 'E' to exit: P
Current Song is  2. Kiki Do You Love Me  < 3:39 >
Enter 'N' for next track, 'P' for previous track, or 'E' to exit: E
Go Back to Menu (1) or Exit (0) :
```

Code :

https://github.com/rushabhporwal29/DLLSongPlaylist.git