***Music Playlist Implementation***

**Abstract about project:**

* The project mainly deals with implementation of priority playlist using few appropriate data structures.
* It is user friendly to order the playlist based on some inbuilt or user defined priority.
* **Data Structure used : Priority Heaps( Priority Queues )**
* **Prototypes defined in the program:**
  + **struct node**

**{**

**char str[50];**

**int priority;**

**};**

* There are many functions used in the programs to perform certain task.
* **Insert1 , Insert2 , Sort , Heapify , Delete , Display , Play\_Music , Destroy , Check\_Priority , etc functions are used in the program**.
* **Pros:**
  + The project aims to sort the playlist according to user’s requirement.
  + There are in-built priorities if user wants to use it.
  + The time complexity to insert and delete has reduced to **O(log n)** and hence is **better compared to linked list , arrays ,** etc.
  + The user also can play the song just on a simple call. **(Responsive)**
  + The CSV file has all the songs names. Each time the user creates a playlist an **abstract of file** is created as heap, hence any modification to heap does not modify the CSV file.
  + Deletion of the song is also possible.
* **Use Cases:**
  + The user has permission to use in-built or his own priority.
  + Then he can delete a song or play a song.

**Description about the Data Structure, Logic and Functionality used:**

The project mainly involves prioritizing the music playlist according to the requirement of the user. It is dependent on the user how to use the priorities but we have designed it thinking that the user is wise enough to enter unique priorities and a check is kept if he by chance enters the same priority.

There are many functionalities involved in the program:

1. **Insert1:** This function is used to insert the song into the heap directly along with the inbuilt priorities specified by the user. This takes O(1) time to store the data directly into heap.
2. **Insert2:** Here the user has the option to enter his own priorities in a wise manner as repetitions are not allowed but a check priority functions checks if the user has by chance entered same priority or not. This also takes O(1) time to store the data into the heap.
3. **Sort:** This function is used to sort the heap according to the priorities where 1(one) is given the highest priority and n is given least priority. Heapify function is used to create the priority heaps internally.
4. **Delete:** This helps the user to delete first song from the heap and the other songs are yet available on heap. This takes O(log n) time to delete a song.
5. **Play:** This function is used to play the song on the music player. The C code calls the music player using the system command available under window header.
6. **Display:** This function displays the songs in the sorted order along with its priorities. The user can verify which song he wishes to listen and whether the list is according to his order or not.
7. **Destroy:** This function de-allocates the place allocated the space allocated for heap earlier.

Hence the project involves all these functions which coordinate to make a responsive playlist implementation.

**Conclusion:**

We have learnt that using appropriate data structure to solve a problem reduces both space as well as time complexity. In our case if priority liked list where used then insertion or deletion would take O(n)

Time complexity.

Hence it is important to choose appropriate data structure.

**Cons:**

* As C program is used a **suitable user interface cannot be provided**.
* The music playlist can only be called but its playlist cannot be accessed.
* The **icons cannot be used to stop or forward to the next song** in the playlist.

**Improvements:**

* **User friendly interface** can be developed.
* The deletion logic can be improved using some other concept as same priorities are given to songs on same level.
* Can built a playlist to do the following functions with the help of other languages.