A LOW-LEVEL DESIGN OF THE MUSIC SYSTEM

by

KISHORE N (19BEC1069)

Project report submitted to

Dr. Susan Elias

SCHOOL OF ELECTRONICS ENGINEERING

CSE2003 – Data Structures and Algorithms

in

B.Tech. ELECTRONICS AND COMMUNICATION ENGINEERING

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Vandalur – Kelambakkam Road

Chennai – 600127

APRIL 2022

BONAFIDE CERTIFICATE

Certified that this project report entitled "**A LOW-LEVEL MUSIC DESIGN SYSTEM"** is a bonafide work of **KISHORE N** who carried out the project work under my supervision and guidance for **CSE2003-Data Structures and Algorithms.**

Dr. Susan Elias

Deputy Director, Centre of Advanced data science

School of Electronics Engineering (SENSE),

VIT University, Chennai

Chennai – 600 127.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Susan Elias, Deputy Director, Centre of Advanced data science**, School of Electronics Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Siva Subramanian. A,** Dean of School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Head of the Department **Dr. Vetrivelan. P** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and the wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

TABLE OF CONTENTS

ABSTRACT

This project entitled 'Music Player' aims at developing a website for accessing and playing songs over the internet. Owing to the growing usage of seamless access to entertainment, there must be appropriate improvements in the performance and stability of applications. Time and Space constraints need to be resolved for storing and playing songs seamlessly over the internet. Usage of appropriate data structures and algorithms can make Music players more efficient and reliable.

Chapter -1

INTRODUCTION

1.1 Objective:

The music player uses appropriate data structures for implementing features in optimum time complexity and uses a divide and conquer Closest Pair algorithm for finding similar songs.

1.2 Benefits

The main benefit of the work is to obtain functionalities of the music system in optimum time complexity

1.3 Features

A low system design of the music system was implemented using various data structures Implementing various features such as obtaining top tracks, random shuffling, seeding of songs, or be it adding and removing different albums other than the regular functioning of a music player. The main aim is to implement the functionalities in the most optimized way using appropriate data structures with less time complexity. Data structures used were doubly linked lists for storing and traversal of songs. A priority queue has this property that when you poll out or remove the elements it's removed in the natural order but when you try to use an iterator to traverse and display the contents, it follows a random order.

This spark of an idea was used in random shuffling and natural ordering of songs. Then songs were also ordered based on how frequently they were played, but when you extract information from an API other criteria might be considered as well. To achieve this a combination of Linked Hash Map and priority queue was used where the priority queue was used to sort the songs in descending order frequency and a Linked Hash Map to maintain the order of the elements of an array of song ids passed to it.

This maintains a hash table of the song ID as key and the frequency of the ids as values. Also, the song album inputs were stored in an Array List before all these above operations were performed. The reason for choosing an array list is due to its property of storing as many elements as possible, unlike an array that has a fixed size. Followed by this Closest Pair Algorithm a divide and conquer approach was used to find similar songs using their id

Chapter 2

2.1 Requirements

i. Intellij Java Ide

2.2 Proposed System

1. **First the three java classes are created, the song class, the album class, and the algorithm/main class.**
2. **The song class gets the input of the song and its name through user-defined function via arguments.**
3. **Then overrides the toString class and returns the song, title, and the time duration time.**
4. **We initialize the ArrayList of datatype songs and another ArrayList of Integer class which stores song ids and then a doubly Linked list for the storage and traversal of songs.**
5. **These data structures are just used to add songs and its id into a playlist which is the doubly linked list,**
6. **Music albums are created using two Array Lists. One stores the string input and another its respective id.**
7. **We pass this into a Doubly Linked List which maintains these songs as set of albums.**
8. **Traversal to next song previous song or be it playing same songs could be done.**
9. **Then these stored songs are accessed and passed into a priority queue. Songs from the priority queue follow a random order each time you access it using an iterator. So custom comparator class condition was defined which maintains a score function as the priority attribute of the queue or basically a max heap.**
10. **Also when a song is played it is removed and pushed to the end of the queue as well.**
11. **When you try to access the songs from the priority queue we obtain a random shuffling each time for that we use a random list iterator or seeder.**
12. **If you don't want it to shuffle we can have this in the original order for that instead of using an iterator we use a user-defined pop function kind of a logic that treats it as a stack to maintain the order by which is popped and doesn't get shuffled.**
13. **Then a new Hashmap is initialized of type integer class(song ids).**
14. **To sort songs based on the frequency a function is initialized ad takes input as an array of return type maps(integer, integer).**

15. **Then we use a custom descending comparator used for storing value, and frequency tuples in a max heap.**
16. **Then we add these entries into a priority queue.**
17. **Then form a resulting array by polling out songs.**
18. **Behind this functioning to maintain the order in which these functions are happening a LinkedHashmap is used which acts like an ordered dictionary as normal dictionaries also change order each time we do some operation except for the fact that they maintain appropriate or respective key and a value pair.**
19. **Thus they are ordered in a way where the most played is on the front side and the least played is on the rear side.**
20. **So when you visualized it's a layer of a combination of data structures providing their functionalities.**
21. **Then closest pair algorithm is used which uses a divide and conquer approach to find a similar song in linear time.**
22. **Therefore, every functionality mentioned above is done in Linear time complexity except for the storing of songs part as appropriate combinations of data structures and algorithms were used to obtain this.**

Chapter 3

Class Description:

The project is implemented in java and uses three Java classes where the first-class contains or gets the information about the song title and duration, next class takes the inputs of albums and ids in string and numeric id format. The third class contains the complete design functioning part.

Modules description:

Song(title, duration)

- Gets the duration and title of songs

@override toString()

- **Concept of overriding is used to custom return the songs**

Music_album(string, string)

- **Gets the song name and respective artist**

Music_album(string,int)

- **Gets the name and song id**

Lookforsong()

- **Check if a song is present if yes return the song ids and artist names**

Addsong()

- **Unique Songids inputs are taken**

Add to playlist (track number, Linked List)

- **If songid and title match the song is returned inside an album**

Add song (id, priority queue playlist)

- **Adds all the songs to playlist.**

Switch case () contains different functions inside.

- **List iterator checks if songs are present in the list or if present in the front and the rear end of the node and does the traversal.**
- **Print menu () makes it a menu-driven or raw UI-based interface.**

Find closest distance ()

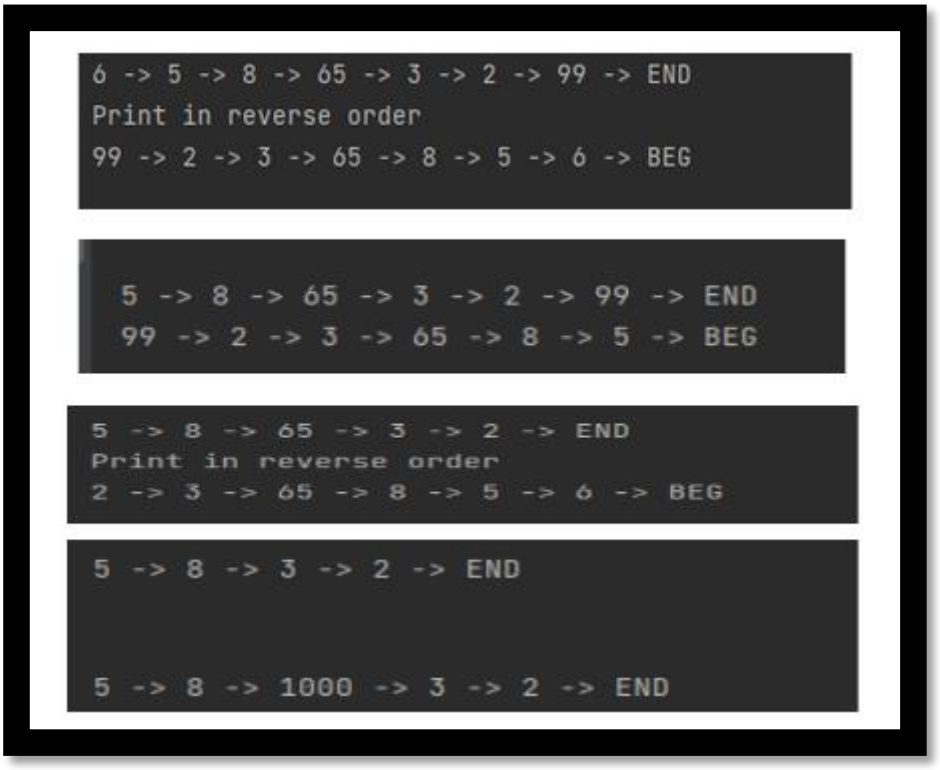
- **Two arrays maintaining song ids of two different albums.**
- **Store each of them in a separate ArrayList.**
- **Sort them based on the x and y axis.**
- **Calls closest recursive function.**

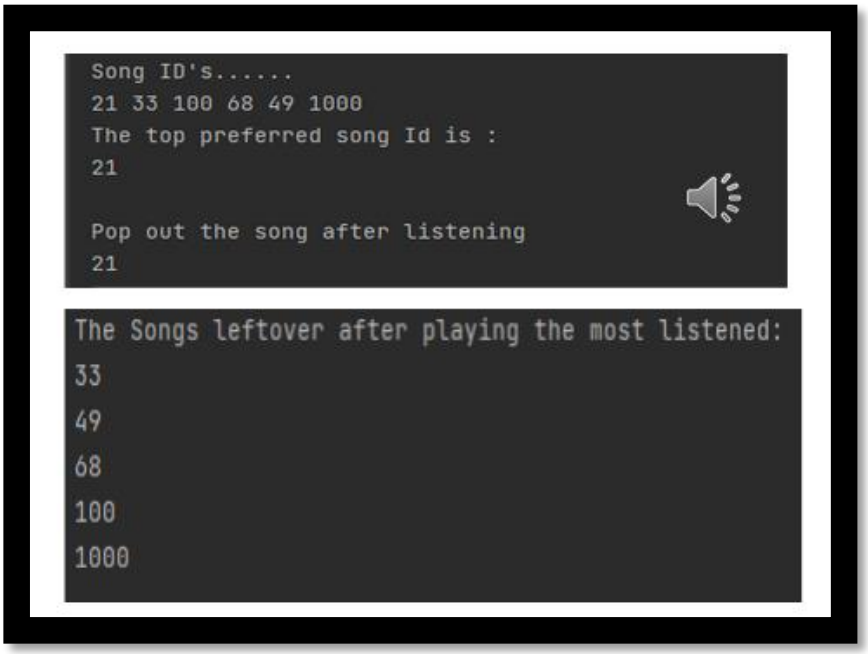
Closest recursive ()

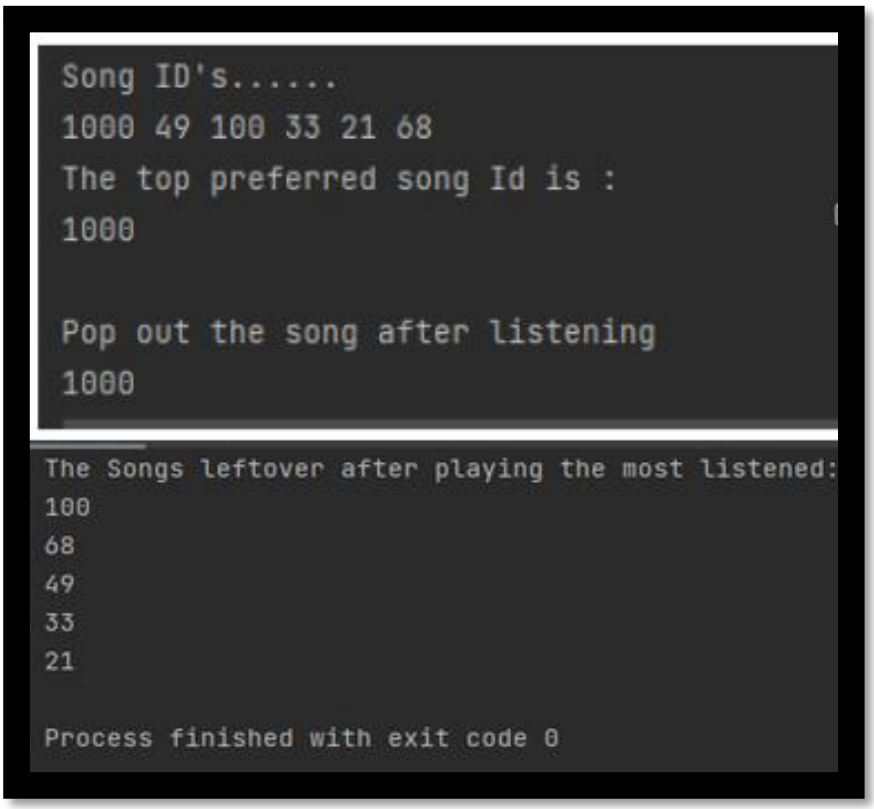
- **Find the midpoint of the cloud of points.**
- **Split them into two different arrays again and find the minimum distance using the Euclidean approach for three different cases.**
- **The First case is two songs from the left.**
- **Second case two songs on right.**
- **Third case two one from the left and the other from the right.**
- **Finally call Bruteforce when only two or three points are left.**
- **Recursively divide and conquer the solution and return the ids of similar songs. This is very similar to the cosine similarity instead angles are used and very much similar to convex hull as well.**

Chapter-4

Results




Song ID's.....
21 33 100 68 49 1000
The top preferred song Id is :
21
Pop out the song after listening
21
The Songs leftover after playing the most listened:
33
49
68
100
1000


Song ID's.....
1000 49 100 33 21 68
The top preferred song Id is :
1000
Pop out the song after listening
1000
The Songs leftover after playing the most listened:
100
68
49
33
21
Process finished with exit code 0

```
Now playing Song{title='Arabic kadaloram', duration=4.5}

Music Player

Choose any options below :

0 - to Exit Playlist
1 - to play next song
2 - to play previous song
3 - to replay the current song
4 - list of all songs
5 - print all available options
6 - delete current song
```

```
Choose any options below :

0 - to Exit Playlist
1 - to play next song
2 - to play previous song
3 - to replay the current song
4 - list of all songs
5 - print all available options
6 - delete current song
1
Now playing Song{title='Kana lane', duration=3.5}
1
Now playing Song{title='Nila kaigiradae', duration=5.0}
```

```
Choose any options below :

0 - to Exit Playlist
1 - to play next song
2 - to play previous song
3 - to replay the current song
4 - list of all songs
5 - print all available options
6 - delete current song
1
Now playing Song{title='Kana lane', duration=3.5}
1
Now playing Song{title='Nila kaigiradae', duration=5.0}
2
Now playing Song{title='Kana lane', duration=3.5}
```

```
4

Song{title='Arabic kadaloram', duration=4.5}
Song{title='Kana lane', duration=3.5}
Song{title='Nila kaigiradae', duration=5.0}
Song{title='Rap god', duration=4.5}
Song{title='Lose yourself', duration=4.5}
Song{title='Ore Kana', duration=6.0}
Song{title='venilave venilave', duration=4.0}
Song{title='Ale Ale', duration=5.7}
Song{title='8 mile', duration=3.52}
Song{title='Not Afraid', duration=3.5}
Song{title='venom', duration=3.56}
Song{title='Without me', duration=3.58}
```

```
5
Music Player

Choose any options below :

0 - to Exit Playlist
1 - to play next song
2 - to play previous song
3 - to replay the current song
4 - list of all songs
5 - print all available options
6 - delete current song
```

```
 Passing the respective SOng IDs into priority Queue


Random Shuffling of Playlist
1 3 4 13 10 5 60 30 13 14 12 13

Getting the songs in the original order....
Song0:1
Song1:3
Song2:4
Song3:5
Song4:10
Song5:12
Song6:13
Song7:13
Song8:13
Song9:14
Song10:30
Song11:60
```

Prioritizing songs based on the most played songs

[13, 13, 13, 4, 10, 1, 30, 60, 3, 14, 5, 12]

```
    int []x = {12,13,40,5,14,3};
    int []y = {3,30,50,1,10,4};

    //(5,3) are similar pair of songs ,(1,4) are similar pair of songs ->closest pair
    System.out.println("Closest distance between similar songs : "+ findclosestdistance(x,y,x.length));
    System.out.println();
    System.out.println("Therefore similar songs are  with ID's (3,5) and (1,4)....");
}
```

```
Closest distance between similar songs : 3.605551275463989

Therefore similar songs are  with ID's (3,5) and (1,4)....

Process finished with exit code 0
```

\

Chapter 5

APPENDIX

Source code:

SONG CLASS

```
package com.kishy.j_comp;

public class Song {
    private String title;
    private double duration;

    public Song(String title, double duration) {
        this.title = title;
        this.duration = duration;
    }

    public String getTitle() {
        return title;
    }

    public double getDuration() {
        return duration;
    }

    @Override
    public String toString() {
        return "Song{" +
                "title='" + title + '\'' +
                ", duration=" + duration +
                '}';
    }
}
```

Music albums

```
package com.kishy.j_comp;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.PriorityQueue;

public class Music_album {
    private String name;
    private String artist;
```

```
    private int ID;
    private ArrayList<Song> songs;
    private ArrayList<Integer> songIds;

    public Music_album(String name, String artist) {
        this.name = name;
        this.artist = artist;
        this.songs = new ArrayList<Song>();
    }

    public Music_album(String name, int ID) {
        this.ID = ID;
        this.name = name;
        this.songIds = new ArrayList<Integer>();

    }

    public Song LookforSong(String title) {

        for (Song checkedSong : songs) {
            if (checkedSong.getTitle().equals(title)) return checkedSong;
        }
        return null;
    }

    public boolean addSong(String title, double duration) {
        if (LookforSong(title) == null) {
            songs.add(new Song(title, duration));

            return true;
        } else {
            return false;
        }
    }


    public void addSong(int songID) {

        songIds.add(songID);


    }

    public boolean addToPlayList(int trackNumber, LinkedList<Song>
PlayList) {
        int index = trackNumber - 1;
        if (index > 0 && index <= this.songs.size()) {
            PlayList.add(this.songs.get(index));
            return true;
        }
//          System.out.println("this album does not have song with
trackNumber "+trackNumber);
        return false;
    }

    public boolean addToPlayList(String title, LinkedList<Song> PlayList) {
        for (Song checkedSong : this.songs) {
            if (checkedSong.getTitle().equals(title)) {
                PlayList.add(checkedSong);
                return true;
            }
```

```
        }
//        System.out.println(title + "there is no such song in album");
        return false;
    }


    public void addToPlayList(int ID, PriorityQueue<Integer> PlayList) {
        PlayList.addAll(this.songIds);

    }
}
package com.kishy.j_comp;

import java.util.*;

import static com.kishy.j_comp.close.findclosestdistance;
import static com.kishy.j_comp.close.sortSongsByFrequency;

public class Main {

    private static ArrayList<Music_album> musicalbums = new ArrayList<>();
    private static ArrayList<Music_album> musical_ids = new ArrayList<>();


    public static void main(String[] args) {

        Music_album musicalbum = new Music_album("Album1","ARR");

        musicalbum.addSong("Arabic kadaloram",4.5);
        musicalbum.addSong("Kana lane",3.5);
        musicalbum.addSong("Nila kaigiradae",5.0);
        musicalbum.addSong("Ore Kana",6.0);
        musicalbum.addSong("Ale Ale",5.7);
        musicalbum.addSong("venilave venilave",4.0);
        musicalbums.add(musicalbum);

        musicalbum = new Music_album("Album2","Eminem");

        musicalbum.addSong("Rap god",4.5);
        musicalbum.addSong("Not Afraid",3.5);
        musicalbum.addSong("Lose yourself",4.5);
        musicalbum.addSong("8 mile",3.52);
        musicalbum.addSong("venom",3.56);
        musicalbum.addSong("Without me",3.58);

        musicalbums.add(musicalbum);



        LinkedList<Song> playList_1 = new LinkedList<>();
        Map<Integer, Integer> map = new HashMap<>();
        PriorityQueue<Integer> playList_2 = new PriorityQueue<>();

        musicalbums.get(0).addToPlayList("Arabic kadaloram",playList_1);
        musicalbums.get(0).addToPlayList("Kana lane",playList_1);
        musicalbums.get(0).addToPlayList("Nila kaigiradae",playList_1);
        musicalbums.get(1).addToPlayList("Rap god",playList_1);
        musicalbums.get(1).addToPlayList("Lose yourself",playList_1);
        musicalbums.get(0).addToPlayList("Ore Kana",playList_1);
        musicalbums.get(0).addToPlayList("venilave venilave",playList_1);
        musicalbums.get(0).addToPlayList("Ale Ale",playList_1);
```

```
        musicalbums.get(1).addToPlayList("8 mile",playList_1);
        musicalbums.get(1).addToPlayList("Not Afraid",playList_1);
        musicalbums.get(1).addToPlayList("venom",playList_1);
        musicalbums.get(1).addToPlayList("Without me",playList_1);

        /////  Respective song ids below

        play(playList_1);

        System.out.println();
        System.out.println("\n Passing the respective SOng IDs into
priority Queue");
        System.out.println();

        playList_2.add(12);
        playList_2.add(13);
        playList_2.add(13);
        playList_2.add(5);
        playList_2.add(14);
        playList_2.add(3);
        playList_2.add(60);
        playList_2.add(30);
        playList_2.add(13);
        playList_2.add(1);
        playList_2.add(10);
        playList_2.add(4);

        System.out.println();
        System.out.println("Random Shuffling of Playlist");

        for (int songid : playList_2) {


            System.out.print(songid + " ");

        }

        System.out.println();
        System.out.println();
        System.out.println("Getting the songs in the original order....");

        for (int i =0 ;i<12;i++){

            System.out.println("Song" + i + ":" + playList_2.poll());
        }

        System.out.println();
        System.out.println();
        System.out.println("Prioritizing songs based on the most played
songs");
        System.out.println();

        int[] arr =  {12,13,13,5,14,3,60,30,13,1,10,4};

        int []freqSorted = sortSongsByFrequency(arr);

        System.out.println(Arrays.toString(freqSorted));
        System.out.println();

        int []x = {12,13,40,5,14,3};
        int []y = {3,30,50,1,10,4};
```

```
        //(5,3) are similar pair of songs ,(1,4) are similar pair of songs
->closest pair
        System.out.println("Closest distance between similar songs : "+
findclosestdistance(x,y,x.length));
        System.out.println();
        System.out.println("Therefore similar songs are  with ID's (3,5)
and (1,4)....");
    }

    private static void play(LinkedList<Song> playList){

        Scanner sc = new Scanner(System.in);
        boolean quit = false;
        boolean forward = true;
        ListIterator<Song> listIterator = playList.listIterator();

        if(playList.size() == 0){
            System.out.println("This playlist have no song");
        }else {
            System.out.println();
            System.out.println("Now playing " +
listIterator.next().toString());
            System.out.println();
            printMenu();
        }

        while(!quit){
            int action = sc.nextInt();
            sc.nextLine();

            switch (action){

                case 0:
                    System.out.println("Playlist complete");
                    quit = true;
                    break;

                case 1:
                    if(!forward){
                        if(listIterator.hasNext()){
                            listIterator.next();
                        }
                        forward = true;
                    }
                    if(listIterator.hasNext()){
                        System.out.println("Now playing
"+listIterator.next().toString());
                    }else {
                        System.out.println("no song availble, reached to
the end of the list");
                        forward = false;
                    }
                    break;
                case 2:
                    if(forward){
                        if (listIterator.hasPrevious()){
                            listIterator.previous();
                        }
                        forward = false;
                    }
```

```
                    if(listIterator.hasPrevious()){
                        System.out.println("Now playing
"+listIterator.previous().toString());
                    }else {
                        System.out.println("we are the first song");
                        forward = false;
                    }
                    break;

                case 3:
                    if(forward){
                        if(listIterator.hasPrevious()){
                            System.out.println("Now playing
"+listIterator.previous().toString());
                            forward = false;
                        }else {
                            System.out.println("we are at the start of the
list");
                        }
                    }else {
                        if(listIterator.hasNext()){
                            System.out.println("now playing
"+listIterator.next().toString());
                            forward = true;
                        }else {
                            System.out.println("we have reached to the end
of list");
                        }
                    }
                    break;

                case 4:
                    printList(playList);
                    break;
                case 5:
                    printMenu();
                    break;
                case 6:
                    if (playList.size() >0){
                        listIterator.remove();
                        if(listIterator.hasNext()){
                            System.out.println("now playing
"+listIterator.next().toString());
                        }
                        else {
                            if(listIterator.hasPrevious())
                                System.out.println("now playing
"+listIterator.previous().toString());
                        }
                    }
                    break;


            }
        }


    }
```

```
    private static void printMenu(){
        System.out.println("Music Player \n");
        System.out.println("Choose any options below :");
        System.out.println();
        System.out.println("0 - to Exit Playlist\n"+
                "1 - to play next song\n"+
                "2 - to play previous song\n"+
                "3 - to replay the current song\n"+
                "4 - list of all songs \n"+
                "5 - print all available options\n"+
                "6 - delete current song");

    }

    private static void printList(LinkedList<Song> playList){
        Iterator<Song> iterator = playList.iterator();
        System.out.println("");

        while (iterator.hasNext()){
            System.out.println(iterator.next());
        }

        System.out.println("");
    }

}


class close{
    public static double distance(points p1, points p2) {

        return Math.sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) *
(p1.y - p2.y));
    }

    public static double calculatecase3(ArrayList<points> arr, int size,
double minDistance) {

        double minValue = minDistance;

        for (int i = 0; i < size; i++) {

            for (int j = i + 1; j < Math.min(size, i + 1 + 7); j++) {

                minValue = Math.min(minValue, distance(arr.get(i),
arr.get(j)));
            }
        }
        return minValue;
    }


    public static double Brute_force(ArrayList<points> arrx, int count) {

        double result = Integer.MAX_VALUE;

        for (int i = 0; i < count; i++) {

            for (int j = i + 1; j < count; j++) {

                result = Math.min(result, distance(arrx.get(i),
```

```
arrx.get(j)));
            }

        }
        return result;
    }


    public static double closestRecursive(ArrayList<points> arrX,
ArrayList<points> arrY, int count) {

        //base condition

        if (count <= 3) {
            return Brute_force(arrX, arrX.size());     //we'll use
bruteforce method
        }

        //finding the midpoint

        int mid = count / 2;
        points midpoint = arrX.get(mid);

        //now create two seperate list which contains elements on left and
elements on right

        ArrayList<points> newArrx1 = new ArrayList<>();
        ArrayList<points> newArrx2 = new ArrayList<>();

        for (int i = 0; i < mid; i++) {

            newArrx1.add(arrX.get(i));

        }

        for (int i = mid; i < arrX.size(); i++) {

            newArrx2.add(arrX.get(i));

        }

        //now call the recursive function

        double distanceleft = closestRecursive(newArrx1, arrY, mid);
        double distanceright = closestRecursive(newArrx2, arrY, count -
mid);

        double minDistance = Math.min(distanceleft, distanceright);


        //now for case 3 one point form each side of mid

        ArrayList<points> arr_case3 = new ArrayList<>();

        for (int i = 0; i < count; i++) {

            if (Math.abs(arrY.get(i).x - midpoint.x) < minDistance) {

                arr_case3.add(arrY.get(i));

            }
```

```
        }
        return Math.min(minDistance, calculatecase3(arr_case3,
arr_case3.size(), minDistance));

    }


    public static double findclosestdistance(int[] x, int[] y, int count) {

        ArrayList<points> arrX = new ArrayList<>();
        ArrayList<points> arrY = new ArrayList<>();

        for (int i = 0; i < x.length; i++) {

            points p = new points(x[i], y[i]);

            arrX.add(p);
            arrY.add(p);

        }


        Collections.sort(arrX, (a, b) -> a.x - b.x);//sort wrt to x-axis
        Collections.sort(arrY, (a, b) -> a.y - b.y);//sort wrt to y-axis


        return closestRecursive(arrX, arrY, arrX.size());
    }


    static int[] sortSongsByFrequency(int[] arr) {

        //calculate frequencies of songs
        Map<Integer, Integer> freqMap = SongFrequencyMap(arr);


        //Comparator for storing value, frequency tuples in priority queue
(max heap)

        Comparator< Map.Entry<Integer,Integer> > descFreqComparator = (e1,
e2) -> e2.getValue() - e1.getValue();

        PriorityQueue< Map.Entry<Integer, Integer> > pq = new
PriorityQueue<>(descFreqComparator);

        //Add map entries to max heap(priority queue)
        for(Map.Entry<Integer, Integer> entry: freqMap.entrySet()) {
            pq.add(entry);
        }

        //Form the result array by deleting elements from priority queue
        int []result = new int[arr.length];
        int i = 0;
        while( !pq.isEmpty() ) {
            Map.Entry<Integer, Integer> entry = pq.poll();
            for(int j = 0; j < entry.getValue(); j++) {
                result[i++] = entry.getKey();
            }
        }
        return result;
    }
```

```
    private static Map<Integer, Integer> SongFrequencyMap(int []arr) {
        //LinkedHashMap to preserve the original order of elements in arr
        Map<Integer, Integer> freqMap = new LinkedHashMap<>();
        int i;
        for(i = 0; i < arr.length; i++) {
            if(freqMap.containsKey(arr[i])) {
                freqMap.put(arr[i], freqMap.get(arr[i])+1);
            } else {
                freqMap.put(arr[i], 1);
            }
        }
        return freqMap;
    }
}
class points
    {
        int x;
        int y;

        points(int x, int y){

            this.x = x;
            this.y = y;
        }

    }
```

Chapter-6

CONCLUSION

The objective of the work was to design a music system with optimum time complexity is achieved and desired results were obtained.

FUTURE WORK

- **One interesting approach is to design the way the system is approached for example we used some basic cosine similarity and Euclidean approaches and closest pair algorithm to structure the similar songs instead, the similarity of every pair of songs and model a traveling salesman kind of a problem which is solved to obtain a playlist (track ordering)**

- **Next the similarities could be determined by what input is taken, in our work only the song ids were used instead similarities could be used by doing audio signal analysis of music and web-based artist profile comparison and using deep learning for making recommendations along with text similarity and sentiment analysis of search results.**

- **In this process of obtaining this multiple combination of datastructes and hybrid algorithms to obtain seamless performance.**

References

[1]https://www.ijrte.org/wpcontent/uploads/papers/v8i2S11/B10270982S19

[2] https://research.ijcaonline.org/volume100/number9/pxc3898163

[3] https://www.researchgate.net/publication/293684941_A_Java-based_msic_player_for_MP3_Ogg_WAV

Biodata

Name: Kishore N

Mobile number: 9444451707

E-mail: Kishore.n2019@vitstudent.ac.in

Permanent address: Plot G22A Ramagiri Nagar 1st street Velachery Chennai 600042