Centro Universitario de Ciencias Exactas e Ingenierías (CUCEI)

# Actividad de aprendizaje 7:
# Métodos de ordenamiento

Víctor Agustín Díaz Méndez
Ingeniería en Informática

Estructura de Datos I (Sección D12)
Profesor: Dr. Gutierrez Hernandez Alfredo

## Problema

Reutilice el resultado de la actividad 06. Se ha encontrado un problema en el programa: la búsqueda binaria no funciona muy bien, el jefe de programadores se ha dado cuenta que la lista casi nunca está ordenada y por eso falla, por lo que es necesario añadirle una opción para ordenarla.

Haga un programa que realice el ordenamiento de la lista, opcionalmente por nombre de la canción como por nombre del cantante. Para evaluar su funcionamiento la radiodifusora quiere probar con distintos tipos de ordenamiento.

## Requerimientos

- El estilo de programación debe ser Orientado a Objetos.
- Debe ofrecer dos opciones de ordenamiento: burbuja (mejorada), inserción, selección, y selección;
- Los métodos de la clase lista se implementarán como métodos de la clase Lista.

## Resumen:

Para la practica utilice cuatro distintos métodos de ordenamiento: burbuja mejorada, selección, inserción, y shell. Des estos métodos de ordenamiento Shell es mucho mas eficiente que los anteriores. Se nota especialmente en arreglos de gran tamaño.

Burbuja mejorada: Este metodo consiste en verificar que si un valor en la posición “x” es mayor al valor siguiente en el arreglo *(x +1).* Si el primer elemento es mayor que segundo intercambian posiciones en el arreglo. Este proceso se repite hasta recorrer todo el arreglo o se puede evitar repetir las ultimas posiciones cada vez que se recorra el arreglo. Suponiendo que *“l”* es la ultima posición del arreglo se decrementaria en uno cada vez que se recorriera el arreglo.

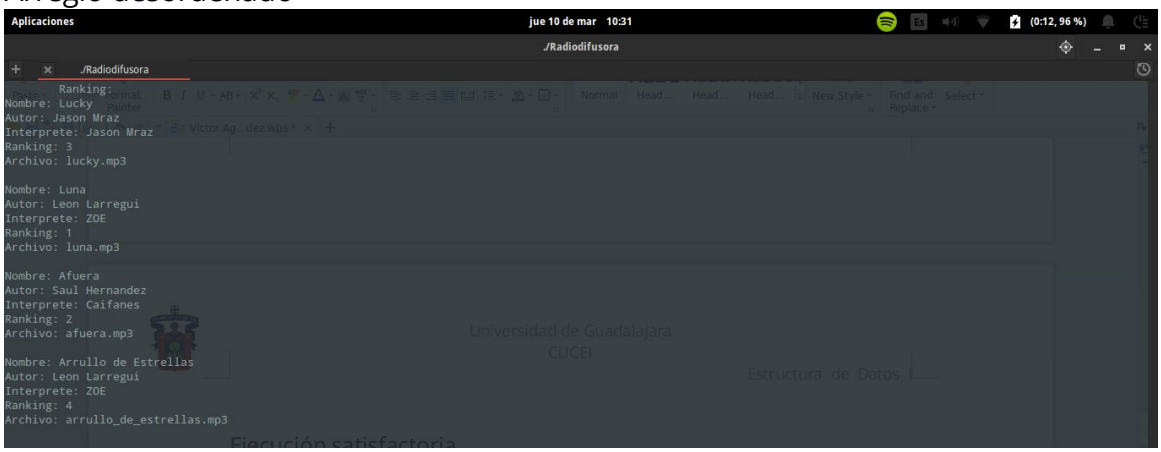
Selección: Este algoritmo consiste en ordenar el arreglo encontrando el valor mas pequeño y guardándolo en la primera posición, suponiendo que mi arreglo este delimitado por *“i”* como comienzo de este y que cada ocasión que se recorra el ciclo de búsqueda aumente, cambiando el comienzo de nuestro arreglo *imaginario.*

Inserción: Este método de ordenamiento consiste en seleccionar un valor en la posición *“j”* que se guardara en un variable auxiliar que recorra el arreglo en retroceso hasta encontrar un valor menor que el suyo o llegar al limite, los elementos que se encuentren entre este y se recorrerán y en la posición final de *“j”* se guardara el elemento. El valor de *“j”* se delimitara por el valor de *“i”* que se incrementara despues de cada recorrido que haga *“i”,* el ciclo que contiene *“i”* se

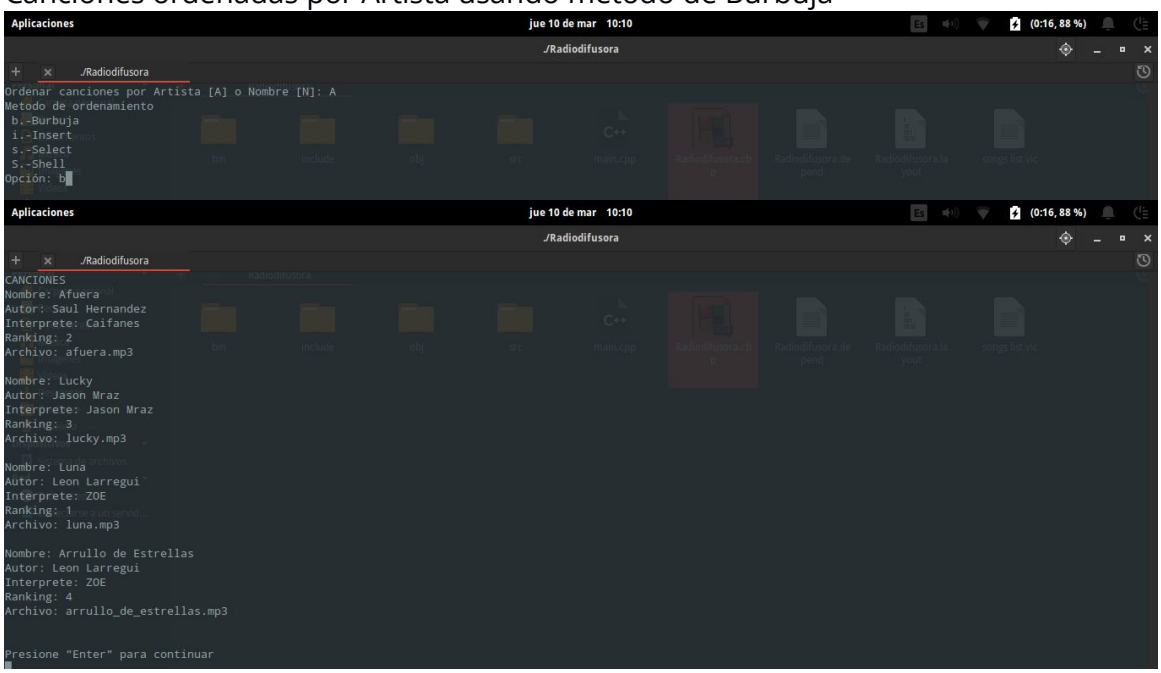
detiene hasta llegar a la ultima posición del arreglo.
Shell: Es le mas eficiente de los metodos de ordenamiento utilizados en el programa, pues es el que realiza el menor numero de comparaciones e intercambios. EL funcionamiento de shell consiste en utilizar un factor que en este caso fue tres cuartos, para calcular el diferencial de las comparaciones.

# Ejecución satisfactoria

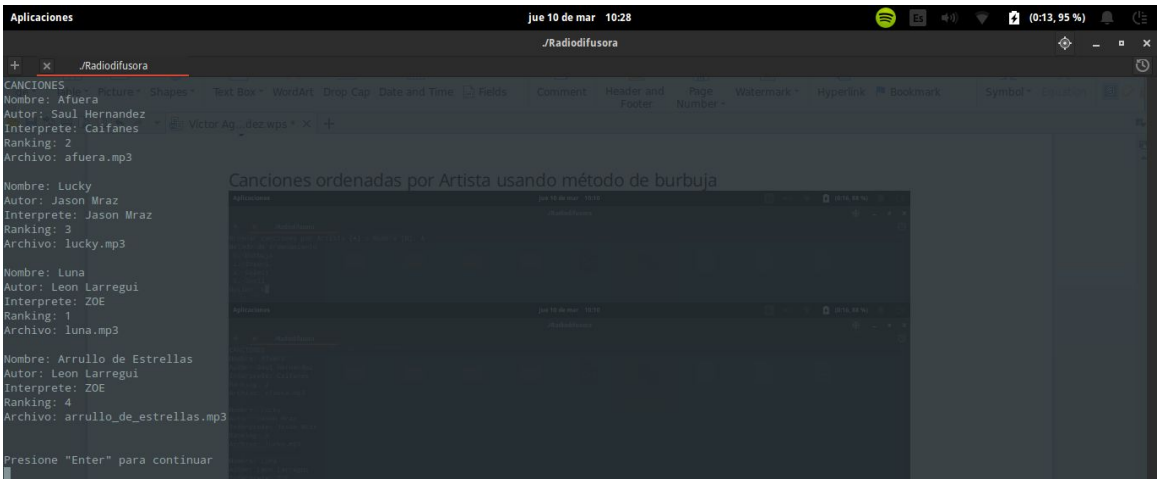
## Arreglo desordenado



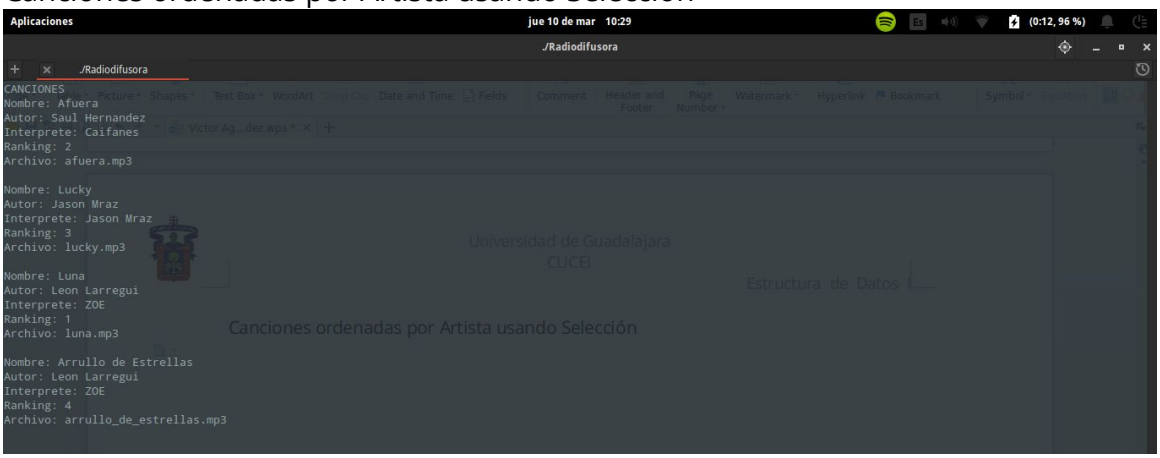

## Canciones ordenadas por Artista usando método de Burbuja



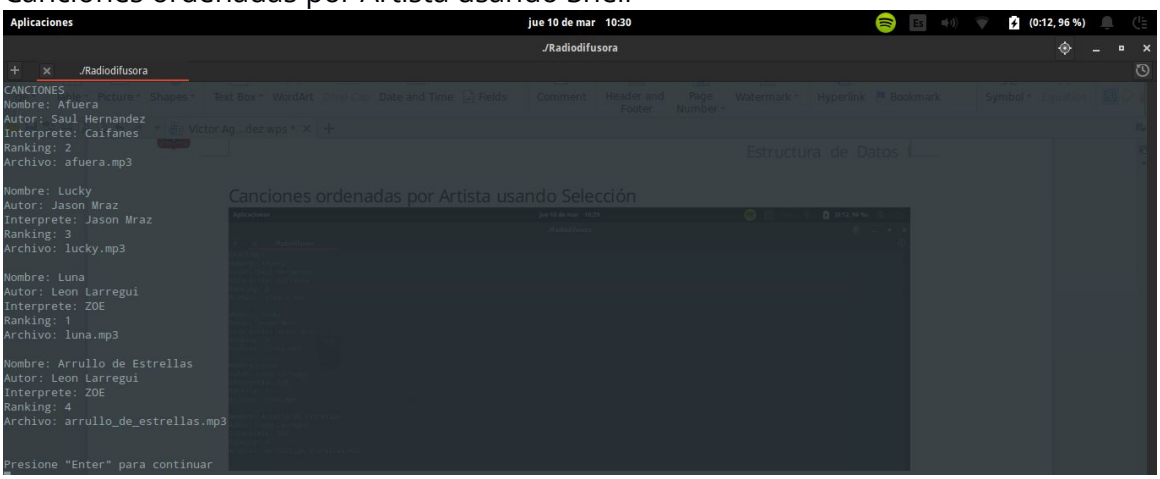

## Canciones ordenadas por Artista usando Inserción


Aplicaciones
jue 10 de mar 10:28
(0:13, 95 %)
./Radiodifusora
./Radiodifusora
CANCIONES
Nombre: Afuera
Autor: Saul Hernandez
Interprete: Caifanes
Ranking: 2
Archivo: afuera.mp3
Nombre: Lucky
Autor: Jason Mraz
Interprete: Jason Mraz
Ranking: 3
Archivo: lucky.mp3
Nombre: Luna
Autor: Leon Larregui
Interprete: ZOE
Ranking: 1
Archivo: luna.mp3
Nombre: Arrullo de Estrellas
Autor: Leon Larregui
Interprete: ZOE
Ranking: 4
Archivo: arrullo_de_estrellas.mp3
Presione "Enter" para continuar
Canciones ordenadas por Artista usando método de burbuja

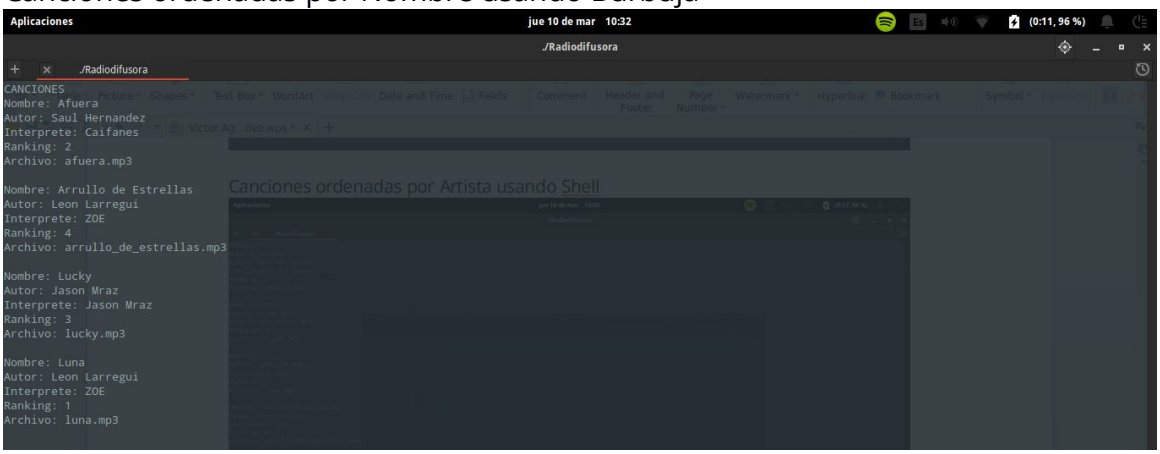
## Canciones ordenadas por Artista usando Selección



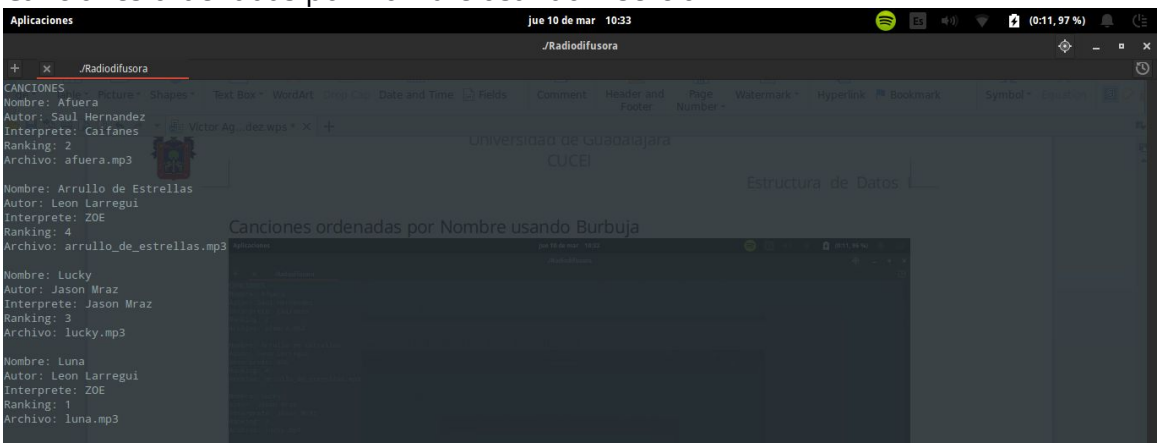

## Canciones ordenadas por Artista usando Shell



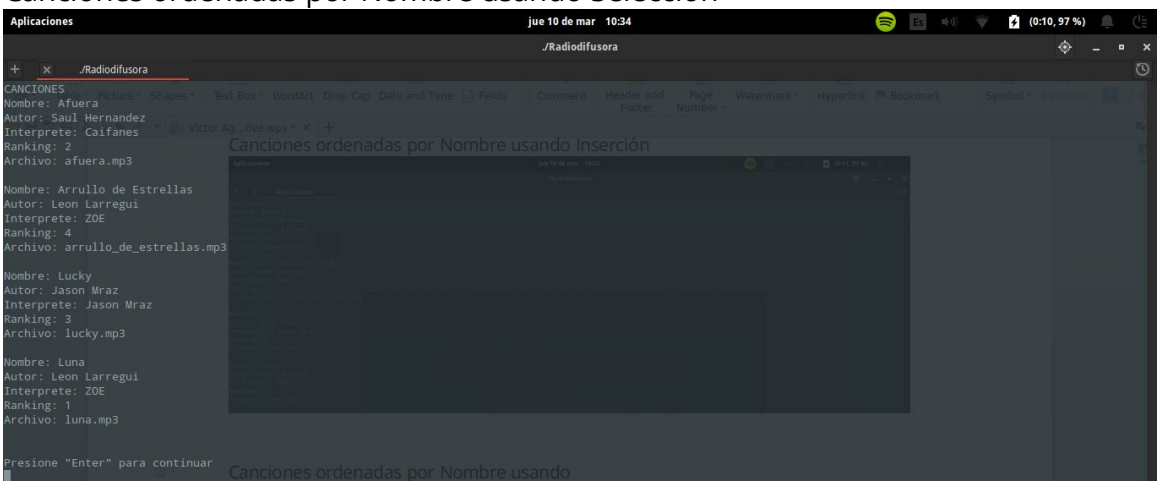
## Canciones ordenadas por Nombre usando Burbuja



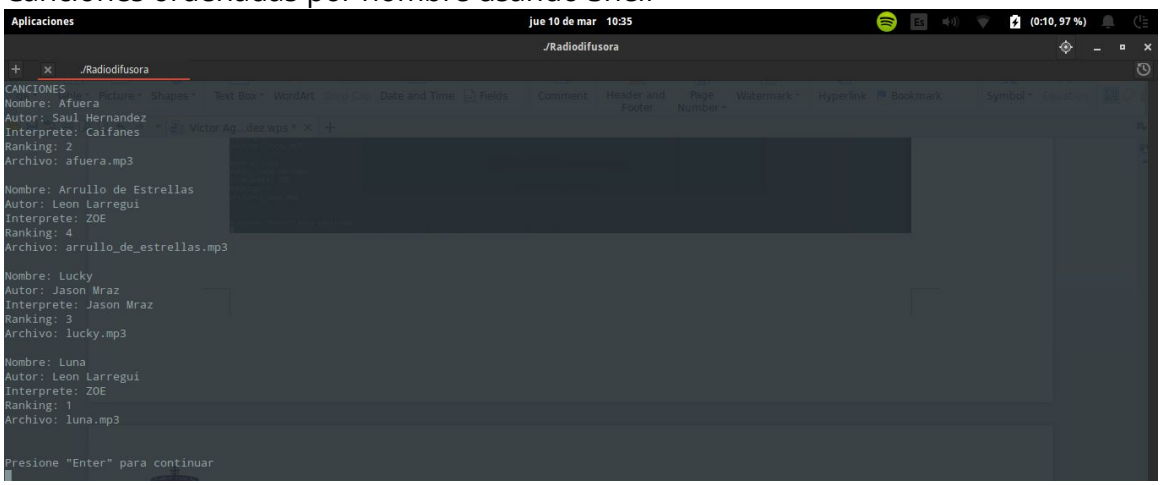

## Canciones ordenadas por Nombre usando Inserción




## Canciones ordenadas por Nombre usando Selección



Canciones ordenadas por nombre usando Shell



# Código fuente

## Main.cpp

```
#include "menu.h"

int main() {
	Menu menu;
	menu.showMenu();
	return 0;
}
```

## SongsList.h

```
#ifndef SONGSLIST_H
#define SONGSLIST_H

#include <exception>

#include "song.h"

#define ARRAY_SIZE 1024

class SongsListException : public std::exception {
private:
	std::string msg;
public:
	explicit SongsListException(const char* message) : msg(message) {}
	explicit SongsListException(const std::string& message) :msg(message) {}
	virtual ~SongsListException() throw () {}
	virtual const char* what() const throw () {
		return msg.c_str();
	}
};

class SongsList {
private:
	Song data[ARRAY_SIZE];
	int last;
	bool isSort;
	bool isValidPos(int);
	void intercambia(Song&, Song&);

	//Sort data methods. "s" is the sort way, artist or name
	void sortDataBubble(char s);
	void sortDataShell(char s);
	void sortDataInsert(char s);
```

```
        void sortDataSelect(char s);

public:
        void initialize();

        bool isEmpty();
        bool isFull();

        void insertData(int, const Song&);
        void deleteData(int);

        int getFirstPos();
        int getLastPos();
        int getPrevPos(int);
        int getNextPos(int);

        int findData(const string&);
        int findDataBinary(const string&);
        int findDataLineal(const string&);
        int findArtist(const string&);

        Song retrieve(int);

        void sortData(const char& s, const char& op);

        void printData();
        void showSong(const int& i);

        void deleteAll();

        void writeToDisk(const string);

        void readFromDisk(string);
};

#endif // SONGSLIST_H
```

SongsList.cpp

```
#include "songsList.h"

#include <iostream>
#include <fstream>
#include <cstdlib>
```

```
using namespace std;

bool SongsList::isValidPos(int p) {
	return p >= 0 and p <=last;
}

void SongsList::initialize() {
	last = -1;
	isSort = false;
}

bool SongsList::isEmpty() {
	return last==-1;
}

bool SongsList::isFull() {
	return last==ARRAY_SIZE-1;
}

void SongsList::insertData(int p, const Song& song) {
	if(isFull()) {
		throw SongsListException("The list is full, trying to insert");
	}
	if(p != -1 and !isValidPos(p)) {
		throw SongsListException("Invalid position, trying to insert");
	}
	for(int i = last; i > p; i--) {
		data[i + 1] = data[i];
	}
	data[p + 1] = song;
	last++;
}

void SongsList::deleteData(int p) {
	if(isEmpty()) {
		throw SongsListException("DATA UNDERRUN. The list is empty, trying to delete");
	}
	if(!isValidPos(p)) {
		throw SongsListException("INVALID POSITION, trying ot delete");
	}
	for(int i = p; i < last; i++) {
		data[i] = data[i + 1];
```

```
        }
        last--;
}

int SongsList::getFirstPos() {
        if(isEmpty()) {
                return -1;
        }
        return 0;
}

int SongsList::getLastPos() {
        return last;
}

int SongsList::getPrevPos(int p) {
        if(isEmpty() or !isValidPos(p) or p == 0) {
                return -1;
        }
        return p-1;
}

int SongsList::getNextPos(int p) {
        if(isEmpty() or !isValidPos(p) or p == last - 1) {
                return -1;
        }
        return p-1;
}

int SongsList::findData(const string& name) {
        if(isSort) {
                return findDataBinary(name);
        } else {
                return findDataLineal(name);
        }
}

int SongsList::findDataBinary(const string& name) {
        int i = 0,
                j = last;
        int medio;
        while (i <= j) {
                medio = (i + j) / 2;
```

```
            if(data[medio].name == name) {
                return medio;
            }
            if(name < data[medio].name) {
                j = medio - 1;
            } else {
                i = medio + 1;
            }
        }
        return -1;
}

int SongsList::findDataLineal(const string& name) {
        for(int i = 0; i <= last; i++) {
            if(data[i].name == name) {
                return i;
            }
        }
        return -1;
}

int SongsList::findArtist(const string& player) {
        for(int i = 0; i <= last; i++) {
            if(data[i].player == player) {
                return i;
            }
        }
        return -1;
}


Song SongsList::retrieve(int p) {
        if(isEmpty()) {
            throw SongsListException("DATA UNDERRUN, list empty, trying to get data");
        }
        if(!isValidPos(p)) {
            throw SongsListException("INVALID POSITION, trying to get data");
        }
        return data[p];
}

void SongsList::sortData(const char& s, const char& op) {
```

```
        switch (op) {
        case 'b':
                sortDataBubble(s);
                break;
        case 'i':
                sortDataInsert(s);
                break;
        case 's':
                sortDataSelect(s);
                break;
        default :
        case 'S':
                sortDataShell(s);
                break;
        }
        isSort = true;
}

void SongsList::sortDataBubble(char s) {
        int i = last,
                j;
        bool seguir;
        bool esMayor;
        do {
                seguir = false;
                j = 0;
                while ( j < i) {

                        if (s == 'N') {
                                esMayor = data[j].name > data[j + 1].name;
                        } else {
                                esMayor = data[j].player > data[j + 1].player;
                        }

                        if (esMayor) {
                                intercambia(data[j], data[j + 1]);
                                seguir = true;
                        }

                        j++;
                }

                i--;
```

```
    } while (seguir);
}

void SongsList::sortDataShell(char s) {
    float fact = 3.0/4.0;
    int dif = last * fact;
    int i;

    while (dif > 0) {
        i = 0;

        while (i <= (last - dif)) {

            if((s == 'N') ? (data[i].name > data[i + dif].name)
                    : (data[i].player > data[i + dif].player)) {
                intercambia(data[i], data[i + dif]);
            }

            i++;
        }
        dif = dif * fact;
    }
}

void SongsList::sortDataInsert(char s) {
    int i = 1,
        j;
    Song aux;
    while (i <= last) {
        aux = data[i];

        j = i;

        while (j > 0 and ((s == 'N')
                            ? (aux.name < data[j - 1].name)
                            : (aux.player < data[j - 1].player))) {

            data[j] = data[j-1];
            j--;
        }
        if(i != j) {
            data[j] = aux;
        }
```

```
            i++;
        }

}

void SongsList::sortDataSelect(char s) {
        int i = 0,
            j;
        int smaller;
        bool isSmaller;

        while (i < last) {
            smaller = i;
            j = i + 1;

            while (j <= last) {

                if (s == 'N') {
                    isSmaller = data[j].name < data[smaller].name;
                } else {
                    isSmaller = data[j].player < data[smaller].player;
                }

                if (isSmaller) {
                    smaller = j;
                }

                j++;
            }

            if (smaller != i) {
                intercambia(data[i], data[smaller]);
            }

            i++;
        }
}


void SongsList::intercambia(Song& a, Song& b) {
        Song c = a;
        a = b;
        b = c;
```

```
}

void SongsList::printData() {
	int i;
	for (i = 0; i <= last; i++) {
		showSong(i);
	}
	if(i==0) {
		cout << "No hay canciones en el ranking" << endl;
	}
	cout << endl;
}

void SongsList::showSong(const int& i) {
	cout << "Nombre: " << data[i].name << endl
		 << "Autor: " << data[i].author << endl
		 << "Interprete: " << data[i].player << endl
		 << "Ranking: " << data[i].ranking << endl
		 << "Archivo: " << data[i].file<< endl << endl;
}

void SongsList::deleteAll() {
	last=-1;
}

void SongsList::writeToDisk(const string nombreArchivo) {
	ofstream miArchivo;
	miArchivo.open(nombreArchivo.c_str(), ios_base::out);
	if(miArchivo.is_open()) {
		for(int i = 0; i <= last; i++) {
			miArchivo << data[i].name << endl
					  << data[i].author << endl
					  << data[i].player << endl
					  << data[i].ranking << endl
					  << data[i].file << endl;
		}
		miArchivo.close();
	} else {
		throw SongsListException("Error de escritura al disco");
	}
}

void SongsList::readFromDisk(string nombreArchivo) {
```

```
	ifstream miArchivo;
	miArchivo.open(nombreArchivo.c_str(), ios_base::in);
	string miString;
	if (!miArchivo.is_open()) {
		throw SongsListException("Error de lectura de disco");
	}

	deleteAll();

	try {
		Song s;
		while(getline(miArchivo, miString)) {
			s.name = miString;
			if(getline(miArchivo, miString)) {
				s.author = miString;
				if(getline(miArchivo, miString)) {
					s.player = miString;
					if(getline(miArchivo, miString)) {
						s.ranking = atoi(miString.c_str());
						if(getline(miArchivo, miString)) {
							s.file = miString;
						}
					}
				}
			}
			insertData(getLastPos(), s);
		}
		miArchivo.close();
	} catch (SongsListException ex) {
		miArchivo.close();
		string error = "Hubo problemas al insertar leyendo desde el disco.\nEl error reportado es.";
		error += ex.what();
		throw SongsListException(error);
	}
}
```

## Song.h

```
#ifndef SONG_H
#define SONG_H

#include <string>
```

```
using namespace std;

struct Song {
public:
	int ranking;
	string name;
	string author;
	string player;
	string file;
};

#endif // SONG_H
```

Menu.h

```
#ifndef MENU_H
#define MENU_H

#include "songsList.h"

class Menu {
public:
	void showMenu();
private:
	SongsList songsList;
	void addSong();
	void deleteSong();
	void showSongs();
	void showSong(const int& index);

	void ifFinded(int index);
	void searchNameSong();
	void searchArtistSong();
};

#endif // MENU_H
```

Menu.cpp

```
#include "menu.h"
#include <iostream>
#include <cstdlib>
#include "song.h"

#ifdef WIN32
```

```
#define CLEAR "cls"
#else
#define CLEAR "clear"
#endif

using namespace std;

void Menu::showMenu() {
	int option;
	songsList.initialize();
	do {

		system(CLEAR);
		cout << "\tRanking: " << endl;
		songsList.printData();

		cout << "\tRANKING MUSICAL" << endl
			<< "Menu:" << endl
			<< " 1.-Añadir cancion" << endl
			<< " 2.-Eliminar cancion" << endl
			<< " 3.-Mostrar ranking" << endl
			<< " 4.-Guardar en disco" << endl
			<< " 5.-Leer de disco" << endl
			<< " 6.-Buscar cancion por nombre " << endl
			<< " 7.-Buscar cancion por artista" << endl
			<< " 0.-Salir" << endl
			<< "Opcion: ";
		cin >> option;
		cin.ignore();

		switch(option) {
		case 1:
			addSong();
			break;
		case 2:
			deleteSong();
			break;
		case 3:
			showSongs();
			break;
		case 4:
			try {
				songsList.writeToDisk("songs list.vic");
```

```
                cout << "Guardado" << endl;
                cin.get();
            } catch(SongsListException ex) {
                cout << "No se pudo guardar" << endl;
            }
            break;
        case 5:
            try {
                songsList.readFromDisk("songs list.vic");
            } catch(SongsListException ex) {
                cout << "No se pudo leer el archivo" << endl;
            }
            break;
        case 6:
            searchNameSong();
            break;
        case 7:
            searchArtistSong();
        case 0:
            break;
        default:
            system(CLEAR);
            cout << "ERROR, opcion invalida" << endl;
            break;
        }
    } while(option!=0);
}

void Menu::addSong() {
    bool answerError;
    Song song;

    system(CLEAR);

    songsList.printData();
    cout << "\tAñadir cancion" << endl;

    do {
        cout << "Nombre: ";
        getline(cin, song.name);

        if(song.name != "") {
            answerError=false;
```

```
        } else {
            cout << "ERROR, ingresa el nombre de la canción" << endl;
            answerError=true;
        }
    } while(answerError);

    do {
        cout << "Autor: ";
        getline(cin, song.author);

        if(song.author != "") {
            answerError=false;
        } else {
            cout << "ERROR, ingresa el nombre del autor" << endl;
            answerError=true;
        }
    } while(answerError);

    do {
        cout << "Interprete: ";
        getline(cin, song.player);

        if(song.player != "") {
            answerError=false;
        } else {
            cout << "ERROR, ingresa el nombre del interprete" << endl;
            answerError=true;
        }
    } while(answerError);

    do {
        cout << "Puesto en el ranking: " << endl;
        cin >> song.ranking;
        cin.ignore();
        if(song.ranking > 0 and song.ranking <= 50) {
            answerError=false;
        } else {
            cout << "ERROR, el ranking no es valido (1-50)" << endl;
            answerError=true;
        }
    } while(answerError);

    do {
```

```
            cout << "Nombre de archivo: ";
            getline(cin, song.file);

            if(song.player != "") {
                answerError=false;
            } else {
                cout << "ERROR, ingresa el nombre del archivo" << endl;
                answerError=true;
            }
        } while(answerError);

        try {
            songsList.insertData(songsList.getLastPos(), song);
        } catch(SongsListException ex) {
            cout << "ERROR, no se pudo guardar" << endl;
        }

        system(CLEAR);
        cout << "La cancion se guardo exitosamente" << endl << endl;
}

void Menu::deleteSong() {
        string name;

        system(CLEAR);
        songsList.printData();

        cout << "Nombre de la cancion a eliminar" << endl;
        getline(cin, name);

        system(CLEAR);
        try {
            songsList.deleteData(songsList.findData(name));
            cout << "Cancion eliminada" << endl;
        } catch(SongsListException ex) {
            cout << "ERROR, no se pudo eliminar la cancion" << endl;
        }
        cout << endl << endl;
}

void Menu::showSongs() {
        system(CLEAR);
        char s; //Sort option
```

```
    char w; //Sort way

    cout << "Ordenar canciones por Artista [A] o Nombre [N]: ";
    cin >> s;
    cin.ignore();

    if (s == 'A' or s == 'N') {
        cout << "Metodo de ordenamiento" << endl
             << " b.-Burbuja" << endl
             << " i.-Insert" << endl
             << " s.-Select" << endl
             << " S.-Shell" << endl
             << "Opción: ";
        cin >> w;
        cin.ignore();

        if (w == 'b' or w == 'i' or w == 's' or w == 'S') {
            songsList.sortData(s, w);
        }
    }
    system(CLEAR);
    cout << "CANCIONES" << endl;
    songsList.printData();

    cout << "Presione \"Enter\" para continuar" << endl;
    cin.get();
}

void Menu::ifFinded(int index) {

    if(index >= 0) {
        cout << "\tCancion encontrada" << endl;
        showSong(index);
    } else {
        cout << "\tNo se encontro" << endl;
    }
    cin.get();
}

void Menu::searchNameSong() {
    string name;

    system(CLEAR);
```

```
	cout << "\tBusqueda por nombre" << endl
		<< "Nombre de la cancion: ";
	getline(cin, name);
	cout << endl;

	ifFinded(songsList.findData(name));
}

void Menu::searchArtistSong() {
	string name;

	system(CLEAR);
	cout << "\tBusqueda por artista" << endl
		<< "Nombre del artista: ";
	getline(cin, name);
	cout << endl;

	ifFinded(songsList.findArtist(name));
}

void Menu::showSong(const int& index) {
	Song s;
	s = songsList.retrieve(index);

	cout << "Nombre: " << s.name << endl
		<< "Autor: " << s.author << endl
		<< "Interprete: " << s.player << endl
		<< "Ranking: " << s.ranking << endl
		<< "Archivo: " << s.file << endl << endl;

}
```