**Unidad 7-Listas Ordenamiento Recorrido y Busqueda**1. (Describir el concepto de Algoritmo de ordenación por **inserción)**

El algoritmo de ordenación por inserción es un concepto muy básico para cualquier persona, se basa en tomar el primer elemento de una lista, ubicarlo en la primer posición y después ir comparando los siguientes elementos de la lista con el primero para ir dándole un orden(descendente o ascendente). Tomemos por ejemplo una lista de números

[7, 2, 1, 4, 6, 3,5]

Si tomamos esta lista y la ordenamos según los pasos que nos indica el algoritmo en nuestra primer iteración tendríamos esto.  
  
[7, 2, 1, 4, 6, 3,5] -> 7  
  
En nuestra segunda iteracion seria algo asi.  
  
[7, 2, 1, 4, 6, 3,5] -> 2, 7  
  
Y en la siguiente seguiría comparando el siguiente elemento con el anterior hasta no encontrar uno menor y ubicándolo en la primer posición o en caso de ser mayor, ubicándolo en la primer posición asignada hasta ordenar nuestra lista.  
  
[7, 2, 1, 4, 6, 3,5] -> 1, 2, 3, 4, 5, 6, 7.

2. (Qué Diferencia tiene con el Algoritmo de ordenación por inserción con intercambios)

La diferencia con la inserción con intercambios, es que la segunda opción es mejor para usar en listas ya preparadas y que tenemos que agregar o quitar elementos y volver a ordenarlos. Con intercambios podemos elegir desde que elemento de la lista revisar y comparar, en caso de que sea mayor(o menor según como este ordenada la lista) que el siguiente item, se intercambiaran.

3. (Dar pautas de Claves de ordenación)  
  
Las claves de ordenación son guias que usamos como referencia para saber que tomar en cuenta a la hora de ordenar nuestras listas, no todas las listas son iguales ni tienen el mismo tipo de elementos y por eso necesitamos saber que nos sirve mas en cada caso.  
Si tenemos una lista de números enteros es fácil de ordenar, solo comparamos entre ellos y ordenamos de menor a mayo. Pero si tenemos una lista compleja con varios campos como nombre, dni, sueldo, etc, no es tan simple. En el caso de una lista con varios campos tomamos uno como referencia para hacer la comparación y ordenarlos, como puede ser ordenarlos alfabéticamente de la A-Z o por DNI o algún código de identificación individual, es mas conveniente para la maquina ordenar numéricamente que alfabéticamente por que tiene menos valores a tener en cuenta a la hora de hacer la comparación.  
Como estamos haciendo una comparación entre campos dentro de un array tenemos que ubicarnos en ese campo dentro del array para hacer la comparación, algo asi seria en código:  
  
lista[pos - 1].nombre > lista[pos].nombre

4. (Que entiende por Estabilidad de la ordenación)  
  
Que un algoritmo de ordenación sea estable quiere decir que siempre que ordenemos los mismos datos con el mismo algoritmo nos va a devolver la misma lista. Por ejemplo:  
Tenemos 4 palabras de 5 letras.  
peach

straw

apple

spork  
  
Si las ordenamos solo teniendo en cuenta la primer letra un algortimo estable nos devolvería esto siempre que lo ejecutemos.  
  
apple

peach

straw

spork  
  
Straw se ordena antes que spork porque estaba antes cuando se ingresaron los datos. En cambio uno inestable podría intercambiar straw y spork al azar.

5. (Describir la Ordenación por **selección** directa)  
La ordenacion por selección directa toma una lista desordenada y la recorre desde el primer elemento hasta el final buscando el elemento de menor valor, cuando lo encuentra lo intercambia con el primer elemento y pasa al siguiente elemento de menor valor pero que sea mayor que el anterior. Cual es el problema con este método? Que recorre toda la lista en búsqueda de un elemento, toma recorridos largos para buscar un solo elemento y después vuelve a hacer el mismo recorrido, en una lista de 10 números enteros no parece tan malo pero si fuera una lista de miles de datos con números complejos la diferencia de tiempo de ejecución va a ser enorme, además que no es un método estable así que no se puede garantizar que mantenga el mismo orden relativo original.  
  
6. (Cómo funciona el Método de la **burbuja**)  
El método de la burbuja es una variación optimizada de la selección directa. En el de burbuja tomamos al ultimo elemento y lo comparamos con el anterior, es menor? En caso de que si lo intercambiamos y pasamos al siguiente. Y que pasa si se encuentra con otro menor, digamos que estoy llevando un 7 y lo comparo con un 5, en ese caso deja el 7 en esa posición y tomamos al 5 para seguir comparando con el resto de la lista hasta encontrar otro menor o dejarlo en la primer posición.  
Lo bueno de este método es que además de ser estable(mantiene el orden relativo original) es que no hacemos recorridos demás, en el caso de que el resto de la lista este ordenada y solo tengamos un valor mal ubicado, solo va a mover ese numero y cuando lo acomode termina el proceso, sin saltos largos ni bucles innecesarios.  
  
7. (¿Cómo gestionaría Listas ordenadas y estructuras?)

Lo importante de una lista ordenada es que siga ordenada, cuando se ingresa un nuevo item o cuando se cargan de un archivo hay que ordenarlo. La mayoría de las operaciones se ejecutan de la misma forma si la lista esta ordenada o no, pero la eficiencia a la hora de ejecutar nuestro programa va a ser diferente si la lista esta ordenada, en caso de ingresar ítems nuevos a la lista se tienen que ordenar según el criterio de ordenamiento que sigamos y guardar los cambios en la lista para mantener el orden y optimizar la eficiencia.  
  
8. (Como se realiza las Búsquedas en listas ordenadas)

Cuando buscamos un elemento en una lista no ordenada se recorre hasta encontrarlo o hasta el final, si la lista esta ordenada se la recorre hasta encontrarlo o al elemento mayor.  
Al tener la lista ordenada si estamos buscando un elemento la búsqueda termina después de encontrarlo o al valor mayor, si estamos buscando al 17 y llegamos al 18 entonces la búsqueda termina ahí, como esta ordenada sabemos que si del 1 al 18 no encontramos ese valor, entonces no existe.

9. (Que es y cómo se Implementa la Búsqueda binaria)  
  
En la búsqueda binaria iniciamos desde ambos extremos de la lista y vamos reduciendo el rango de búsqueda. Tomamos el inicio y el fin, tomamos la mitad de la lista y nos vamos acercando hasta encontrar el valor.  
  
Índice del elemento en la mitad: mitad = (ini + fin) / 2

Si no se encuentra, ¿dónde seguir buscando?

Buscado < elemento en la mitad: fin = mitad - 1

Buscado > elemento en la mitad: ini = mitad + 1

Si ini > fin, no queda dónde buscar

Si bien este método de búsqueda es mas complejo es bastante eficiente, no sobrecargamos el programa con bucles gigantes y vamos reduciendo la cantidad de valores a procesar a medida que nos acercamos al valor buscado.

**Marco Practico:**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

const int MAX = 100;

typedef struct

{

    string nombre, apellido, carrera, dni;

    int ano;

} Alumno;

typedef Alumno Alumnos[MAX];

typedef struct

{

    Alumnos alumnos;

    int contador;

} ListaDeAlumnos;

int i;

void cargarLista(ListaDeAlumnos &listaDeAlumnos);

void guardarLista(ListaDeAlumnos &listaDeAlumnos);

void agregarAlumno(ListaDeAlumnos &listaDeAlumnos);

void borrarAlumno(ListaDeAlumnos &listaDeAlumnos, string nombre, string apellido);

void mostrarLista(ListaDeAlumnos &listaDeAlumnos);

void ordenarDescendente(ListaDeAlumnos &listaDeAlumnos);

void ordenarAscendente(ListaDeAlumnos &listaDeAlumnos);

bool operator<(Alumno alumnIzq, Alumno alumnDer);

bool operator>(Alumno alumnIzq, Alumno alumnDer);

int main()

{

    ListaDeAlumnos listaDeAlumnos;

    string nombre, apellido;

    int opcion;

    listaDeAlumnos.contador = 0;

    cargarLista(listaDeAlumnos);

    do

    {

        cout << "1 - Agregar un alumno" << endl;

        cout << "2 - Borrar a un alumno" << endl;

        cout << "3 - Mostrar la lista" << endl;

        cout << "4 - Ordenar la lista" << endl;

        cout << "0 - Salir" << endl;

        cin >> opcion;

        switch(opcion)

        {

            case 1:

                agregarAlumno(listaDeAlumnos);

                break;

            case 2:

                cout << "Ingrese el nombre del alumno que desee eliminar: ";

                cin >> nombre;

                cout << "Ingrese el apellido del alumno que desee eliminar: ";

                cin >> apellido;

                borrarAlumno(listaDeAlumnos, nombre, apellido);

                break;

            case 3:

                mostrarLista(listaDeAlumnos);

                break;

            case 4:

                cout << "5 - Por nombres descendente" << endl;

                cout << "6 - Por nombres ascendente" << endl;

                cin >> opcion;

                switch(opcion)

                    case 5:

                    ordenarDescendente(listaDeAlumnos);

                    break;

                    case 6:

                    ordenarAscendente(listaDeAlumnos);

                    break;

                break;

            case 0:

                guardarLista(listaDeAlumnos);

                break;

            default:

                cout << "Opcion no valida" << endl;

                break;

        }

    } while (opcion != 0);

    return 0;

}

void cargarLista(ListaDeAlumnos &listaDeAlumnos)

{

    ifstream file;

    file.open("Alumnos.txt", ios::in);

    if(!file.is\_open())

    {

        cout << "Error de apertura del archivo!" << endl;

    }

    else

    {

        while (!file.eof())

        {

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].nombre;

            file.get();

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].apellido;

            file.get();

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].carrera;

            file.get();

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].dni;

            file >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].ano;

            if((!listaDeAlumnos.alumnos[listaDeAlumnos.contador].nombre.empty())&&(!listaDeAlumnos.alumnos[listaDeAlumnos.contador].apellido.empty())&&(!listaDeAlumnos.alumnos[listaDeAlumnos.contador].carrera.empty())&&(!listaDeAlumnos.alumnos[listaDeAlumnos.contador].dni.empty())&&(listaDeAlumnos.alumnos[listaDeAlumnos.contador].ano != 0))

            {

                listaDeAlumnos.contador++;

            }

        }

    }

    file.close();

}

void guardarLista(ListaDeAlumnos &listaDeAlumnos)

{

    ofstream file;

    file.open("Alumnos.txt", ios::in);

    if(file.is\_open())

    {

        for(i = 0; i < listaDeAlumnos.contador; i++)

        {

            file << listaDeAlumnos.alumnos[i].nombre<< " " << listaDeAlumnos.alumnos[i].apellido<< " " << listaDeAlumnos.alumnos[i].carrera<< " " << listaDeAlumnos.alumnos[i].dni<< " " << listaDeAlumnos.alumnos[i].ano<<endl;

        }

    }

    file.close();

}

void mostrarLista(ListaDeAlumnos &listaDeAlumnos)

{

    string nombre, apellido, carrera, dni;

    int ano;

    cout << "Listado de estudiantes: " << endl;

    for(i = 0; i < listaDeAlumnos.contador; i++)

    {

        nombre = listaDeAlumnos.alumnos[i].nombre;

        apellido = listaDeAlumnos.alumnos[i].apellido;

        carrera = listaDeAlumnos.alumnos[i].carrera;

        dni = listaDeAlumnos.alumnos[i].dni;

        ano = listaDeAlumnos.alumnos[i].ano;

        cout << "Alumno: " << nombre << " " << apellido << ". DNI: " << dni << " De la carrera: " << carrera << " año: " << ano << endl;

        cout << endl;

    }

}

void agregarAlumno(ListaDeAlumnos &listaDeAlumnos)

{

    if(listaDeAlumnos.contador < MAX)

    {

        cout << "Ingrese el nombre del alumno: ";

        cin.ignore();

        getline(cin, listaDeAlumnos.alumnos[listaDeAlumnos.contador].nombre);

        cout << "Ingrese el apellido del alumno: ";

        getline(cin, listaDeAlumnos.alumnos[listaDeAlumnos.contador].apellido);

        cout << "Ingrese la carrera del alumno: ";

        getline(cin, listaDeAlumnos.alumnos[listaDeAlumnos.contador].carrera);

        cout << "Ingrese el numero de DNI del alumno: ";

        getline(cin, listaDeAlumnos.alumnos[listaDeAlumnos.contador].dni);

        cout << "Ingrese en que ano de la carrera esta el alumno: ";

        cin >> listaDeAlumnos.alumnos[listaDeAlumnos.contador].ano;

        listaDeAlumnos.contador = (listaDeAlumnos.contador + 1);

    }

    else

    {

        cout << "Maximo de alumnos en la lista alcanzado (100!)" << endl;

    }

}

void borrarAlumno(ListaDeAlumnos &listaDeAlumnos, string nombre, string apellido)

{

    int pos = 0;

    for(i = 0; i < listaDeAlumnos.contador; i++)

    {

        if((listaDeAlumnos.alumnos[i].nombre == nombre)&&(listaDeAlumnos.alumnos[i].apellido == apellido))

        {

            pos = i;

            for(i = pos; i < listaDeAlumnos.contador -1; i++)

            {

                listaDeAlumnos.alumnos[i] = listaDeAlumnos.alumnos[i+1];

            }

            listaDeAlumnos.contador = (listaDeAlumnos.contador -1);

            cout << "Alumno eliminado correctamente!" << endl;

        }

    }

}

void ordenarDescendente(ListaDeAlumnos &listaDeAlumnos)

{

    for(int i = 1; i < listaDeAlumnos.contador; i++)

    {

        int pos = i;

        while((pos > 0) && (listaDeAlumnos.alumnos[pos -1] < listaDeAlumnos.alumnos[pos]))

        {

            Alumno tmp;

            tmp = listaDeAlumnos.alumnos[pos];

            listaDeAlumnos.alumnos[pos] = listaDeAlumnos.alumnos[pos -1];

            listaDeAlumnos.alumnos[pos -1] = tmp;

            pos--;

        }

    }

}

void ordenarAscendente(ListaDeAlumnos &listaDeAlumnos)

{

    for(int i = 1; i < listaDeAlumnos.contador; i++)

    {

        int pos = i;

        while((pos > 0) && (listaDeAlumnos.alumnos[pos -1] > listaDeAlumnos.alumnos[pos]))

        {

            Alumno tmp;

            tmp = listaDeAlumnos.alumnos[pos];

            listaDeAlumnos.alumnos[pos] = listaDeAlumnos.alumnos[pos -1];

            listaDeAlumnos.alumnos[pos -1] = tmp;

            pos--;

        }

    }

}

bool operator<(Alumno alumnIzq, Alumno alumnDer)

{

    return(alumnIzq.nombre < alumnDer.nombre);

}

bool operator>(Alumno alumnIzq, Alumno alumnDer)

{

    return(alumnIzq.nombre > alumnDer.nombre);

}



