Algoritmos y Estructuras de Datos I

Trabajo Práctico N°6	Unidad 6
Modalidad: Semi -Presencial	**Estratégica Didáctica:** Trabajo grupal
Metodología de Desarrollo: Det. docente	**Metodología de Corrección:** Via Classroom.
Carácter de Trabajo: Obligatorio – Con Nota	**Fecha Entrega:** A confirmar por el Docente.

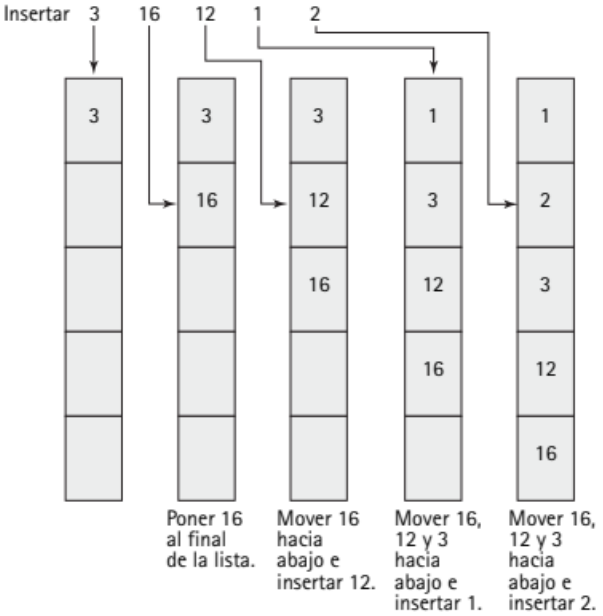
MARCO TEÓRICO

1. ¿Describir el concepto de Algoritmo de ordenación por inserción

A fin de elaborar un algoritmo para insertar un nuevo elemento en la lista, observamos primero que estamos trabajando con una lista no ordenada y que los valores no se tendrán que mantener en algún orden particular. En un ordenamiento por inserción, los valores son insertados uno por uno en una lista que originalmente estuvo vacía. Un ordenamiento por inserción se usa a menudo cuando los datos de introducción se deben ordenar; cada valor se pone en su lugar correcto conforme se está leyendo.

El algoritmo funciona de la siguiente manera:

Se comienza con una lista vacía o con el primer elemento, que se considera ordenado. A continuación, se toma el siguiente elemento y se compara con los elementos de la lista ordenada para encontrar su posición correcta. Se inserta este elemento en la posición adecuada y se repite el proceso hasta que todos los elementos estén ordenados.



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

Los elementos de una lista tiene un orden alfabético o números de forma ascendente o descendente. La ordenación por inserción no parte de una lista vacía, sino que inserta cada elemento en su posición correcta dentro de una lista ya ordenada. En cambio, la ordenación por intercambios no se basa en insertar en una lista vacía, sino en comparar e intercambiar elementos para ordenar la lista.

3. Dar pautas de Claves de ordenación

Las claves de ordenación, son:

- Realizar la función para la comparación sobre un campo concreto.
- Realizar un intercambio a otro campo el ordenamiento
    1. Repasar la lista para buscar el número más pequeño.
    2. Anotar en la hoja una segunda columna.
    3. Tachar el número en la lista original.
    4. Repetir este proceso, siempre buscando el número más pequeño que permanezca en la lista original.
    5. Detenerse cuando todos los números estén tachados.

Es así, como buscamos el valor más pequeño en la lista y lo cambiamos con el componente en la primera posición en la lista. Buscamos el siguiente valor más pequeño en la lista y lo cambiamos con el componente en la segunda posición. Este proceso continúa hasta que todos los componentes estén en el lugar correcto.

Código ejemplo de clave de ordenamiento:

```
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
using namespace std;
const int N = 15; // Tamaño máximo del array
// Definición de una estructura para almacenar los datos
typedef struct {
   int codigo;
   string nombre;
   double sueldo;
} tDato;
typedef tDato tArray[N]; // Definición del tipo tArray que es un array de tDato de tamaño N
// Definición de una estructura para manejar una lista de datos
typedef struct {
   tArray datos;
   int cont; // Contador de elementos en la lista
} tLista;
// Función para mostrar la lista
void mostrar(tLista lista) {
   for (int i = 0; i < lista.cont; i++) {
      cout << setw(10) << lista.datos[i].codigo
          << setw(20) << lista.datos[i].nombre
          << setw(12) << fixed << setprecision(2) << lista.datos[i].sueldo << endl;
   }       }
// Sobrecarga del operador > para comparar dos tDato por el campo nombre
bool operator>(tDato opIzq, tDato opDer) {
   return (opIzq.nombre > opDer.nombre);
}
int main() {
   tLista lista; // Declaración de una lista de tipo tLista
   ifstream archivo; // Declaración de un objeto ifstream para manejar archivos
   lista.cont = 0; // Inicialización del contador de la lista a 0
   archivo.open("datos.txt"); // Apertura del archivo "datos.txt"
   if (!archivo.is_open()) { // Verificación de si el archivo se abrió correctamente
      cout << "Error de apertura del archivo!" << endl; // Mensaje de error en caso de fallo
   } else {
      tDato dato;
      archivo >> dato.codigo; // Lectura del primer dato
      // Lectura de los datos del archivo mientras haya espacio en el array y el código no sea -1
      while ((lista.cont < N) && (dato.codigo != -1)) {
```

```
        archivo >> dato.nombre >> dato.sueldo; // Lectura del nombre y sueldo
        lista.datos[lista.cont] = dato; // Almacenamiento del dato en la lista
        lista.cont++; // Incremento del contador de la lista
        archivo >> dato.codigo; // Lectura del siguiente código
    }
    cout << "Antes de ordenar:" << endl;
    mostrar(lista); // Mostrar la lista antes de ordenar
    // Ordenamiento por inserción
    for (int i = 1; i < lista.cont; i++) { // Desde el segundo elemento hasta el último
        int pos = i;
        while ((pos > 0) && (lista.datos[pos-1] > lista.datos[pos])) {
            tDato tmp;
            tmp = lista.datos[pos]; // Intercambio de elementos
            lista.datos[pos] = lista.datos[pos - 1];
            lista.datos[pos - 1] = tmp;
            pos--;
        }   }
    cout << "Después de ordenar:" << endl;
    mostrar(lista); // Mostrar la lista después de ordenar
}
return 0; // Fin del programa}
```

4. Que entiende por Estabilidad de la ordenación

La estabilidad en la ordenación es importante cuando queremos mantener el orden de los elementos que tienen el mismo valor. Si dos elementos son iguales, su posición relativa en la lista no cambiará después de ordenar. La estabilidad asegura que el orden de los elementos con el mismo valor se mantiene igual que en la lista original.

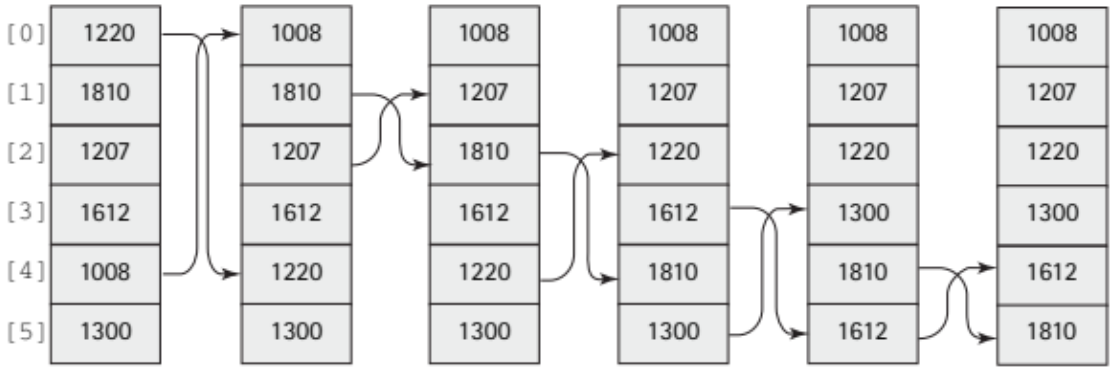
5. Describir la Ordenación por selección directa

La selección directa encuentra el valor más pequeño en la lista y lo intercambia con el elemento en la primera posición. Luego encuentra el siguiente valor más pequeño y lo intercambia con el elemento en la segunda posición. Continúa este proceso hasta que todos los elementos estén ordenados.
Un ejemplo del proceso sería:
Inicio del proceso: Comienza con el primer elemento, hasta el penúltimo elemento de la lista.

Buscar el menor: Para cada posición, busca el elemento más pequeño en la sublista desde un elemento, hasta el último elemento.

Intercambiar: Intercambia el elemento en la posición i con el elemento más pequeño encontrado, si no son el mismo.

Repetir: Repite este proceso para cada posición hasta que todos los elementos estén en el lugar correcto.



6. Cómo funciona el Método de la burbuja

El método de la burbuja es un algoritmo de ordenamiento en el que los elementos menores van ascendiendo hasta alcanzar su posición correcta. El bucle externo recorre el array, mientras que el bucle interno compara los elementos adyacentes para verificar si están en el orden correcto. Si no lo están, se realiza el intercambio correspondiente. Este proceso continúa hasta que se completa la ordenación. Si en una iteración del bucle externo no se realizan intercambios, significa que la lista está ordenada y no es necesario seguir.

7. ¿Cómo gestionaría Listas Listas ordenadas y estructuras?

Las listas ordenadas son similares a las listas sin orden, pero con la diferencia de que sus elementos siguen un orden específico. Al insertar un nuevo elemento en una lista ordenada, este debe ser colocado en la posición correspondiente según el criterio de ordenamiento del código. Esto garantiza que la lista se mantenga ordenada en todo momento.

Para hacer la búsqueda más eficiente, se puede comprobar la carga de los datos desde un archivo de texto. Si los elementos en el archivo de texto se guardaron en orden, al leerlos se mantendrá esa organización. Sin embargo, si los elementos están desordenados en el archivo, se deberá realizar una inserción ordenada al cargar los datos en la lista.

8. Como se realiza las Búsquedas en listas ordenadas

La búsqueda en lista ordenada recorre hasta encontrar el mayor elemento, mientras que al final el valor sea menos que el buscado

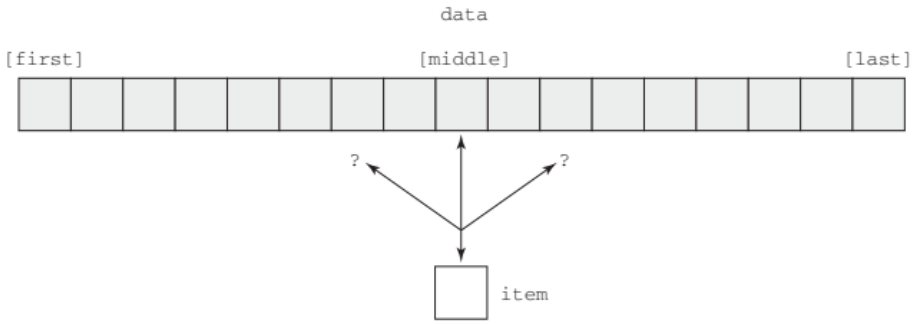
```
//Se define la lista como un array de enteros de tamaño 10 utilizando typedef
const int N = 10;
typedef int tLista[N];
tLista lista;

int buscado;
cout << "Valor a buscar: ";
cin >> buscado;
//recorre la lista hasta encontrar un elemento mayor o igual al valor buscado
// O bien, llegar al final de la lista.
int i = 0;
while ((i < N) && (lista[i] < buscado)) {
   i++;
}
```

```
//Se determina si el valor ha sido:
// encontrado
//no ha sido encontrado
//o si se ha encontrado un valor mayor que el buscado.
if (i == N) {
   // Al final: no se ha encontrado
   cout << "No encontrado!" << endl;
} else if (lista[i] == buscado) {
   // Encontrado!
   cout << "Encontrado en posición " << i + 1 << endl;
} else {
   // Hemos encontrado uno mayor
   cout << "No encontrado!" << endl;
}
```

9. Qué es y cómo se Implementa la Búsqueda binaria

La búsqueda binaria es un algoritmo eficiente para encontrar un elemento en una lista ordenada. Es más rápida que la búsqueda lineal, especialmente para listas grandes. Se basa en el principio de la aproximación sucesiva, dividiendo repetidamente la lista en mitades hasta encontrar el elemento buscado o determinar que no está presente.
La implementación de la Búsqueda Binaria se inicia con una comparación inicial, donde se compara el elemento buscado con el elemento en el medio de la lista.
Luego, si el elemento buscado es igual al elemento medio, se ha encontrado el elemento. Si el elemento buscado es menor que el elemento medio, se continúa la búsqueda en la mitad izquierda de la lista. Por otro lado, si el elemento buscado es mayor que el elemento medio, se continúa la búsqueda en la mitad derecha de la lista.
Este proceso se repite, redefiniendo la lista como la mitad correspondiente, hasta encontrar el elemento o determinar que no está presente en la lista.



MARCO PRÁCTICO

Desarrollar un Programa que:

a. La Directora nos Encarga llevar un registro de los Alumnos por carrera y año . Para ello deberemos armar un Programa que guarde los datos de los Alumnos en Alumnos.txt.
Adicionalmente deberá realizar las siguientes Tareas:

1. Mostrar Listado de los Alumnos x pantalla
2. Mostrar un Alumno Determinado (buscar y mostrar x plantalla)
3. Insertar un Alumno
4. Eliminar un Alumno
5. Buscar un alumno
6. Que permita Ordenar de Forma Ascendente y Descendente los Alumnos y Mostrarlos x pantalla
7. Opcional - Que permita elegir el campo de Ordenamiento.

Algunas Consideraciones:

1. Usar Archivo para persistir y recuperar
2. Usar Estructuras, contador y arrays (max 100 alumnos)
3. Usar sobrecarga de Operadores

bool operator>(tRegistro opIzq, tRegistro opDer);
bool operator<(tRegistro opIzq, tRegistro opDer);

4. Modularizar en funciones la implementación
5. Mantener el Main lo mas pequeño posible.

Lic. Oemig José Luis.