Universidad Autónoma de Aguascalientes

Estructuras Computacionales

INGENIERÍA EN COMPUTACIÓN INTELIGENTE 2° A

Proyecto 4 Ordenación

parcial II

Nombre de los alumnos:

Ángel David Ortiz Quiroz ID: 261481

Ximena Rivera Delgadillo ID: 261261

Erick Iván Ramírez Reyes ID: 260806

Diego Emanuel Saucedo Ortega ID: 261230

José Luis Sandoval Pérez ID: 261731

Carlos Daniel Torres Macias ID: 244543

Profesor: Miguel Ángel Meza de Luna

Fecha de entrega: 03 de junio del 2022

Método de la Burbuja

```cpp
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
using namespace std;


void bubble(int A[], int N)
 {
    for (int i = 0; i < N; i++)
   {
        for (int j = 0; j < N; j++)
      {
            if (A[i] < A[j])
           {
                A[i] += A[j];
                A[j] = A[i] - A[j];
                A[i] = A[i] - A[j];
            }
        }//fin for j
    }//fin for i
}

int main()
{
    int N;   int A[100];

    cout << "Ingrese el numero de elementos: "; cin >> N;
    for (int i = 0; i < N; i++)
    {
        cout << "Elemento " << i << ": "; cin >> A[i];
    }//fin for i

    bubble(A, N);

    cout << "Arreglo ordenado:\n";

    for (int i = 0; i < N; i++)
   {
        cout << "\t" << A[i];
    }//fin for i

    cout << "\n .:PROGRAMA REALIZADO POR :. ";
    cout << "\nAngel David Ortiz Quiroz ID:261481";
    cout << "\nErick Ivan Ramirez Reyes ID:260806";
    cout << "\nXimena Rivera Delgadillo ID:261261";
    cout << "\nJose Luis Sandoval Perez ID:261731";
    cout << "\nDiego Emanuel Saucedo Ortega ID:261230";
    cout << "\nCarlos Daniel Torres Macias ID : 244543";
    cout << "\n";
    return 0;
}
```

Método Quicksort

```
#include <iostream>
using namespace std;

void ordBurbuja(int a[], int n)
 {
    bool interruptor = true;
    int pasada, j;
    for (pasada = 0; pasada < n - 1 && interruptor; pasada++)
   {
        interruptor = false;
        for (j = 0; j < n - pasada - 1; j++)
        {

            if (a[j] > a[j + 1])
            {
                interruptor = true;
                swap(a[j], a[j + 1]);
            }

        }

    }

}


void quicksort(double a[], int primero, int ultimo)
{
    int i, j, central;
    double pivote;

    central = (primero + ultimo) / 2;
    pivote = a[central];

    i = primero;
    j = ultimo;

    do {


        while (a[i] < pivote) i++;

        while (a[j] > pivote) j--;

        if (i <= j)
       {
            swap(a[i], a[j]);
            i++;
            j--;

        }


    } while (i <= j);
```

```
    if (primero < j)
    {
        quicksort(a, primero, j);
    }

    if (i < ultimo) {
        quicksort(a, i, ultimo);
    }
}


int main()
{
    int a1[7] = { 10,5,30,20,15,40,90 };

    double a2[10] = {30,2,15,8,10,11,9,17,21,4 };

    int n1 = 7, n2 = 10;

    ordBurbuja(a1, n1);

    cout << "Metodo de la Burbuja" << endl;

    for (int i = 0; i < n1; i++)
    {

        cout << a1[i] << ", ";

    }
    cout << endl;
    cout << "Metodo QuickSort" << endl;

    quicksort(a2, 0, 9);

    for (int i = 0; i < n2; i++)
    {

        cout << a2[i] << ", ";

    }

    cout << endl;

    system("pause");
    cout << "\n .:PROGRAMA REALIZADO POR :. ";
    cout << "\nAngel David Ortiz Quiroz ID:261481";
    cout << "\nErick Ivan Ramirez Reyes ID:260806";
    cout << "\nXimena Rivera Delgadillo ID:261261";
    cout << "\nJose Luis Sandoval Perez ID:261731";
    cout << "\nDiego Emanuel Saucedo Ortega ID:261230";
    cout << "\nCarlos Daniel Torres Macias ID : 244543";
    cout << "\n";


    return 0;
}
```

Ordenaciones Externas

```
#include<iostream>
#include <conio.h>
#include <string>
#include <windows.h>
#include<fstream> //Libreria para los ficheros
using namespace std;

int main()
{
	Menu_Intercalacion_De_Archivo();
    system("pause");
    cout << "\n .:PROGRAMA REALIZADO POR :. ";
    cout << "\nAngel David Ortiz Quiroz ID:261481";
    cout << "\nErick Ivan Ramirez Reyes ID:260806";
    cout << "\nXimena Rivera Delgadillo ID:261261";
    cout << "\nJose Luis Sandoval Perez ID:261731";
    cout << "\nDiego Emanuel Saucedo Ortega ID:261230";
    cout << "\nCarlos Daniel Torres Macias ID : 244543";
    cout << "\n";

	return 0;
}


class Intercalacion {
private:
	void abrir(fstream* f, string nom, int tip = 1);
	void cerrar(fstream* f);
public:
	void limpiar();
	bool hayDatos(char nom[]);
	void mostrar(char nom[]);
	void insertar(int d, char nom[]);
	void ordenar();
};
//-- Metodos para los ficheros --//
void Intercalacion::abrir(fstream* f, string nom, int tip) {
	if (tip == 1) //LECTURA
		(*f).open(nom, ios::in);//->
  //MODO TEXTO (Acceder a los datos) usaré ">>"
	else if (tip == 2) //ESCRITURA SIN BORRAR
		(*f).open(nom, ios::out | ios::app);   //->
  //MODO TEXTO (Colocar datos y no borrará) usaré "<<"
	else                                       //ESCRITURA y BORRAR
		(*f).open(nom, ios::out); //->
  //MODO TEXTO (Colocar datos) usaré "<<"
}
void Intercalacion::cerrar(fstream* f) {
	(*f).close();
}
void Intercalacion::limpiar() {
	fstream F1, F2, F3;
	abrir(&F1, "F1.txt", 3);
	abrir(&F2, "F2.txt", 3);
```

```
	abrir(&F3, "F3.txt", 3);
	cerrar(&F1);
	cerrar(&F2);
	cerrar(&F3);
}
bool Intercalacion::hayDatos(char nom[]) {
	fstream F;
	abrir(&F, nom, 1);
	int x = -10001;
	F >> x;
	if (x != -10001)
		return true;
	else
		return false;
	cerrar(&F);
}

void Intercalacion::mostrar(char nom[]) {
	fstream F;
	abrir(&F, nom, 1);
	int dato;
	F >> dato;
	while (!F.eof()) {
		cout << dato << " ";
		F >> dato;
	}
	cerrar(&F);
}
void Intercalacion::insertar(int d, char nom[]) {
	fstream F;
	abrir(&F, nom, 2);
	F << d << " ";
	cerrar(&F);
}
//-- Metodo propio de Intercalación de Archivo --//
void Intercalacion::ordenar() {
	fstream F1, F2, F3;
	abrir(&F1, R"(F1.txt)", 1);
	abrir(&F2, R"(F2.txt)", 1);
	abrir(&F3, R"(F3.txt)", 3);

	int  r1, r2;
	F1 >> r1;
	F2 >> r2;
	//-- Escribir en orden cuando los dos tengan datos.
	while (!F1.eof() && !F2.eof())
		if (r1 < r2) {
			F3 << r1 << " ";
			F1 >> r1;
		}
		else {
			F3 << r2 << " ";
			F2 >> r2;
		}
	//-- Escribir lo sobrante.
	if (!F1.eof())
		while (!F1.eof()) {
			F3 << r1 << " ";
```

```
                        F1 >> r1;
                }
        else if (!F2.eof())
                while (!F2.eof()) {
                        F2 << r2 << " ";
                        F2 >> r2;
                }
        cerrar(&F1);
        cerrar(&F2);
        cerrar(&F3);
}
void Menu_Intercalacion_De_Archivo() {
        Intercalacion A;
        char* z[1];
        char* principal[5]= { "MENU METODO DE INTERCALACION DE ARCHIVO : ","INGRESAR
DATOS A UN FICHERO.", "ORDENAR LOS DATOS.","MOSTRAR LOS DATOS DE UN
FICHERO.","ATRAS"};
        char* a[4] = { "MENU METODO DE INTERCALACION DE ARCHIVO
(INGRESAR):","INGRESAR UN NUMERO.","INGRESAR VARIOS NUMEROS.","ATRAS." };
        char* b[4] = { "MENU METODO DE INTERCALACION DE ARCHIVO: ","FICHERO
1.","FICHERO 2.","FICHERO 3." };

        int op1 = 1, op2, op3, N, dato, mayor1, mayor2;
        char nomFic[7];
        mayor1 = mayor2 = -1001;
        A.limpiar();
        do {
                op1 = construirMenu(principal, 5, op1 + 1);
                switch (op1) {
                case 1:
                        op2 = 1;
                        do {
                                op2 = construirMenu(a, 4, op2 + 1);
                                if (op2 != 3) {
                                        op3 = construirMenu(b, 3);
                                        if (op3 == 1)
                                                strcpy(nomFic, "F1.txt");
                                        else
                                                strcpy(nomFic, "F2.txt");
                                }
                                if (op2 == 1 || op2 == 2) { //UNO O VARIOS DATOS.
                                        N = 1;
                                        while (op2 == 2 && (N <= 1 || N >= 1000)) {
                                                z[0] = "Digite cuantos datos desea
ingresar:";
                                                construirMenu(z, 1);
                                                cin >> N;
                                        }
                                        for (int i = 0; i < N; i++) {
                                                do {
                                                        z[0] = "Digite un dato para
colocarlo:";
                                                        construirMenu(z, 1);
                                                        cin >> dato;
                                                        if (op3 == 1 && dato > mayor1)
                                                                mayor1 = dato;
                                                        else if (op3 == 2 && dato > mayor2)
                                                                mayor2 = dato;
```

```
                                        else {
                                              z[0] = "El dato debe ser
mayor.";
                                              construirMenu(z, 1);
                                              system("pause>>NULL");
                                              dato = -10001;
                                        }
                                  } while (dato < -10000 || dato>10000);
                                  A.insertar(dato, nomFic);
                                  z[0] = "Se ha colocado el dato.";
                                  construirMenu(z, 1);
                                  system("pause>>NULL");
                            }
                      }
                } while (op2 != 3);
                break;
          case 2:
                if (A.hayDatos("F1.txt") && A.hayDatos("F2.txt")) {//Si hay
datos.
                      z[0] = "Se ha ordenado los ficheros y se ha guardado en el
fichero 3.";
                      A.ordenar();
                }

                else if (!A.hayDatos("F1.txt"))
                      z[0] = "El fichero 1 no se ha ingresado al menos un
dato.";
                else
                      z[0] = "El fichero 2 no se ha ingresado al menos un
dato.";
                construirMenu(z, 1);
                system("pause>>NULL");
                break;
          case 3:
                op3 = construirMenu(b, 4);
                if (op3 == 1)
                      strcpy(nomFic, "F1.txt");
                else if (op3 == 2)
                      strcpy(nomFic, "F2.txt");
                else
                      strcpy(nomFic, "F3.txt");
                if (A.hayDatos(nomFic)) {//Si hay datos.
                      z[0] = "Los datos son: ";
                      construirMenu(z, 1);
                      A.mostrar(nomFic);
                }
                else {
                      z[0] = "No hay datos en el fichero.";
                      construirMenu(z, 1);
                }
                system("pause>>NULL");
                break;
          }
    } while (op1 != 4);
}
void limpiar() {
    system("cls");
    fflush(stdin);
```

```
	cin.clear();
}
void gotoxy(int x, int y) {
	COORD Cursor_Pos = { x,y };
	SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), Cursor_Pos);
}
void texcolor(int x){
	SetConsoleTextAttribute(GetStdHandle (STD_OUTPUT_HANDLE),x);
}
int construirMenu(char* opciones[], int can, int seleccion = 2, int X = 15, int Y =
3) {
	bool ban = true;
	int i, lim, tecla, opc = seleccion;
	limpiar();
	do {

		//-- Buscar la frase más larga --//
		for (lim = i = 0; i < can && ban; i++) {
			if (strlen(opciones[i]) > lim)
				lim = strlen(opciones[i]);
		}
		for (i = Y; i <= (can + Y + 1) && ban; i++) {
			gotoxy(X, i);

			//-- Esquinas y lineas verticales del lado izquierdo. --//
			if (i == Y)                                cout << "É";
			else if (i != can + Y + 1)      cout << "º";
			else                                                cout << "È";

			//-- Lineas horizontales y las opciones.--//
			if (i == Y || i == can + Y + 1) {
				for (int j = 0; j < lim + 6; j++) cout << "Í";
			}
			else {

				//-- Cambio de Color --//
				if (opc == i - Y)
					texcolor(4);

				//-- Las letras --//
				if (i - (Y + 1) == 0)
					gotoxy(X + 1 + (lim + 1 - strlen(opciones[i - (Y +
1)])) / 2, i); // Centrado
				else
					gotoxy(X + 1, i);
				cout << "   " << opciones[i - (Y + 1)];
				texcolor(7);
			}

			//-- Esquinas y lineas verticales del lado derecho. --//
			gotoxy(X + lim + 6 + 1, i);
			if (i == Y)                                cout << "»";
			else if (i != can + Y + 1)      cout << "º";
			else                                                cout << "¼";
		}

		//-- Funciones de reconocer las teclas --//
		if (can == 1) {
```

```
                        gotoxy(X, i);
                        return 0;
                }

                else
                        tecla = getch();
                if (tecla == 72) {
                        if (opc != 2)//-- Flecha Arriba del teclado
                                opc--;
                        else
                                opc = can;
                        ban = true;
                }
                else if (tecla == 80) {//-- Flecha Abajo del teclado
                        if (opc != can)
                                opc++;
                        else
                                opc = 2;
                        ban = true;
                }
                else
                        ban = false;
        } while (tecla != 13);
        return opc - 1;
}
```

Análisis Comparativo

1. Ordenamiento de burbuja

Este tipo de ordenamiento es un método en C es un algoritmo para ordenar arreglos; no es el más rápido, pero es uno que sirve para introducir los conceptos de ordenamiento de arreglos en C.

El uso de este ordenamiento es sencillo simplemente se recorre el arreglo en un ciclo for, y dentro de ese ciclo, se hace otro ciclo; es decir, tenemos dos ciclos.

En el segundo ciclo (que va desde 0 hasta la longitud del arreglo menos el paso del primer ciclo) comparamos el elemento actual con el siguiente, y si el actual es mayor, intercambiamos los valores.

Esto se repite y al final el arreglo estará ordenado.

El algoritmo es sencillo; hay que recorrer todo el arreglo y si encontramos que el elemento actual (arreglo[x]) es menor al elemento siguiente (arreglo[x+1]) entonces los intercambiamos.

Es importante hacer este recorrido hasta la longitud menos 1 para que cuando lleguemos al penúltimo elemento y hagamos un x+1 el índice no esté fuera de los límites del arreglo.

Con esto habremos ordenado solo una parte del arreglo; hay que hacer todo este recorrido de nuevo, específicamente N veces en donde N es la longitud del arreglo; al terminar, el arreglo estará ordenado.

2. Método Quicksort

Este método es uno de los más eficientes y veloces de los métodos de ordenación interna. Es también conocido con el nombre del método rápido y de ordenamiento por partición.

Quicksort es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a n log n.

Este método fue creado por el científico británico Charles Antony Richard Hoare, también conocido como Tony Hoare en 1960, su algoritmo Quicksort es el algoritmo de ordenamiento más ampliamente utilizado en el mundo.

La idea central de este algoritmo consiste en lo siguiente:

Se toma un elemento x de una posición cualquiera del arreglo.

Se trata de ubicar a x en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentran a su izquierda sean menores o iguales a x y todos los elementos que se encuentren a su derecha sean mayores o iguales a x.

Se repiten los pasos anteriores, pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta de x en el arreglo.

Este tipo de ordenamientos también puede aplicarse combinado con el uso de punteros o apuntadores para el manejo de las diferentes estructuras de datos que existen.

3. Ordenaciones Externas. Mezcla de archivos.

Los algoritmos de ordenación externa son necesarios cuando los datos que se quiere ordenar no caben en la memoria principal (RAM) de la computadora y por tal motivo se encuentran almacenados en un dispositivo secundario externo (el disco duro, cinta, memoria usb, etc.). La mayoría de estos algoritmos utilizan la técnica de divide y vencerás y la intercalación de archivos, para aplicar el ordenamiento.

Los algoritmos de ordenación externa más comunes son dos:

- Intercalación directa
- Mezcla natural

Intercalación Directa

La intercalación o mezcla directas es un algoritmo de ordenación externa, que permite organizar los elementos de un archivo, de forma ascendente o descendente.

La idea central de este algoritmo consiste en realizar de forma sucesiva una partición y una fusión que produce secuencias ordenadas de longitud cada vez mayor. En la primera pasada la partición es de longitud 1 y la fusión produce secuencias ordenadas de longitud 2. En la segunda pasada la partición es de longitud 2 y la fusión produce secuencias ordenadas de longitud 4. Este proceso se repite hasta que la longitud de la partición sea menor o igual al número de elementos del archivo original.

Mezcla Natural

La mezcla natural o mezcla equilibrada es un algoritmo de ordenación externa, que se encarga de organizar los elementos de un archivo de forma ascendente o descendente.

La idea central de este algoritmo consiste en realizar particiones tomando secuencias ordenadas de máxima longitud en lugar de secuencias ordenadas de tamaño fijo previamente determinadas, como la intercalación directa. Posteriormente se realiza la fusión de esas secuencias ordenadas, alternándolas entre los dos archivos auxiliares. Repitiendo este proceso, se logra que el archivo quede completamente ordenado. Para aplicar este algoritmo, se necesitarán

cuatro archivos. El archivo original y tres archivos auxiliares. De estos cuatro archivos, dos serán considerados de entrada y dos de salida, alternativamente en cada paso del algoritmo. El proceso termina cuando al finalizar un paso, el segundo archivo de salida quede vacío y el primero queda completamente ordenado.

Algoritmos de Ordenamiento	Quicksort	Burbuja	Shellsort	Heapsort	Inserción	Selección
Breve Descripción	Utiliza un pivote y ordena los elementos según él.	Se recorre el arreglo intercambiando los elementos adyacentes que estén desordenados. Se recorre el arreglo tantas veces hasta que ya no haya cambios que realizar.	Asigna una distancia y ordena entre ellos.	Almacena los elementos en un montículo y luego extrae el nodo que queda como raíz, La cima siempre contiene el menor elemento.	Toma uno por uno los elementos y avanza hacia su posición con respecto a los anteriormente ordenados hasta recorrer todo el arreglo.	Consiste en encontrar el menor de todos los elementos del arreglo e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenar todo el arreglo.
Característica Principal	División por pivote.	Prácticamente lo que hace es tomar el elemento mayor y lo va recorriendo de posición en posición hasta ponerlo en su lugar.	Compara e intercambia	Utiliza un árbol binario para estructurar el proceso de ordenamiento.	Se puede llegar a demorar mucho	Selecciona el menor elemento de la secuencia no ordenada y lo intercambia.
Ventajas	✓No requiere memoria adicional. ✓Rápida ejecución.	✓ Fácil implementación. ✓ No requiere memoria adicional.	✓ Eficiente para conjuntos de elementos medianos menores a 1000.	✓ Su desempeño promedio es tan bueno como el método Quickshort.	✓ Fácil implementación. ✓ Requerimientos mínimos de memoria.	✓ Fácil implementación. ✓ No requiere memoria adicional. ✓ Rendimiento constante: poca diferencia entre el peor y el mejor caso.
Desventajas	✓Trabaja con recursividad. ✓Implementación complicada.	✓ Realiza numerosas comparaciones. ✓ Lento.	✓ Complejo al momento de analizar.	✓ Implementación complicada.	✓ Lento. ✓ Realiza numerosas comparaciones.	✓ Lento. ✓ Realiza numerosas comparaciones.
Operaciones Máximas	$\Omega(n^2)$ en peor de los casos y $\Omega(n \log n)$ en el	$\Omega(n^2)$	$\Omega(n \log^2 n)$		$\Omega(n^2/4)$	$\Omega(n^2)$
Big O	O(n2)	O(n2)	O(n2)	O(n log n)	O(n2)	O(n2)