**Instituto Politécnico Nacional**

*“Escom”*

**Análisis de algoritmos**

**Profesor:**

Edgardo Adrián Franco Martínez

**Integrantes:**

González Barrios Alan Ernesto

Romero Gamarra Joel Mauricio

Zavala Pérez Rene

**Introducción**

El análisis de algoritmos es una parte importante de la Teoría de [complejidad computacional](https://es.wikipedia.org/wiki/Complejidad_computacional) más amplia, que provee estimaciones teóricas para los recursos que necesita cualquier [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) que resuelva un problema computacional dado. Estas estimaciones resultan ser bastante útiles en la búsqueda de algoritmos eficientes.

Por tal razón los códigos de los algoritmos siguientes, se implementaron y se probaron, con tal de hallar cuales eran los algoritmos más eficientes, rápidos y que utilizaran menor espacio en memoria.

**Planteamiento del Problema**

Con base en el archivo de entrada proporcionado que tiene 10,000,000 números diferentes; estos se tienen que ordenar bajo los siguientes métodos de ordenamiento y comparar experimentalmente las complejidades de estos:

* Burbuja (Bubble Sort)
* Burbuja Simple
* Burbuja Optimizada
* Inserción (Insertion Sort)
* Selección (Selection Sort )
* Shell (Shell Sort)
* Ordenamiento con árbol binario de búsqueda (Tree Sort)

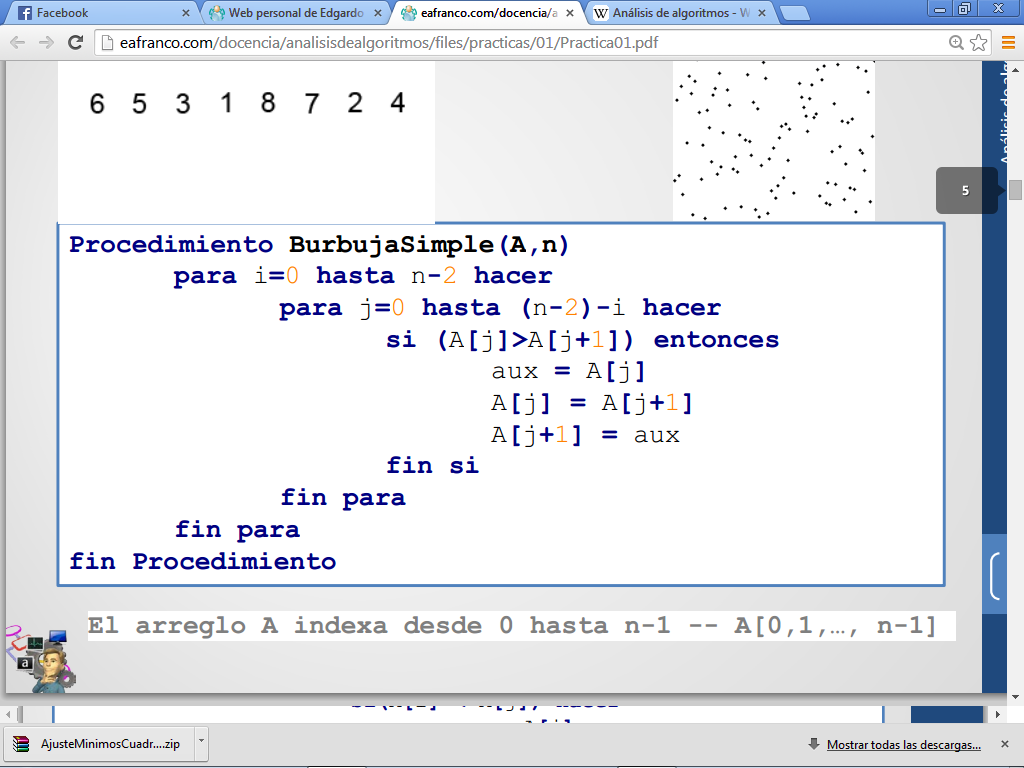
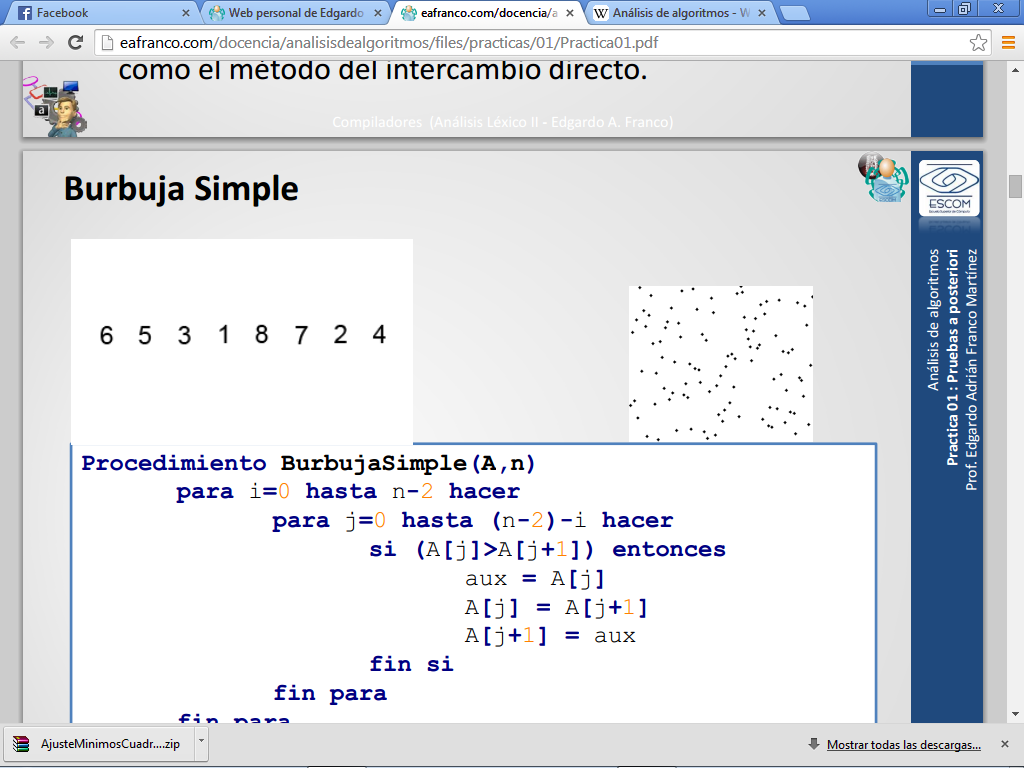
**Algoritmos**

El problema pide leer de un archivo texto que consta con 10,000,000 de números escritos de manera aleatoria, donde cada uno de los algoritmos siguientes, procederá a leer los números de este y ordenarlos, con el fin de observar que algoritmos son más rápidos y eficientes en su funcionamiento, por ello es necesario ver a detalle cómo es que cada uno de estos algoritmos funciona, apoyándonos en pseudocodigo:

**Ordenamiento Burbuja**

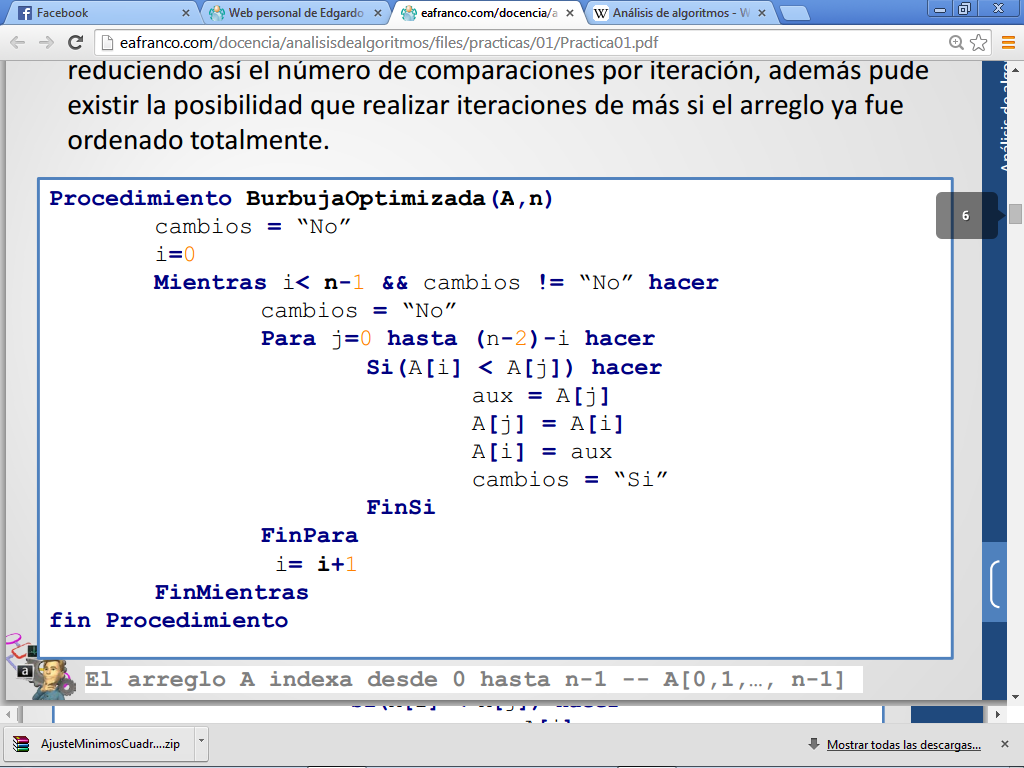
**Burbuja Simple**

Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada



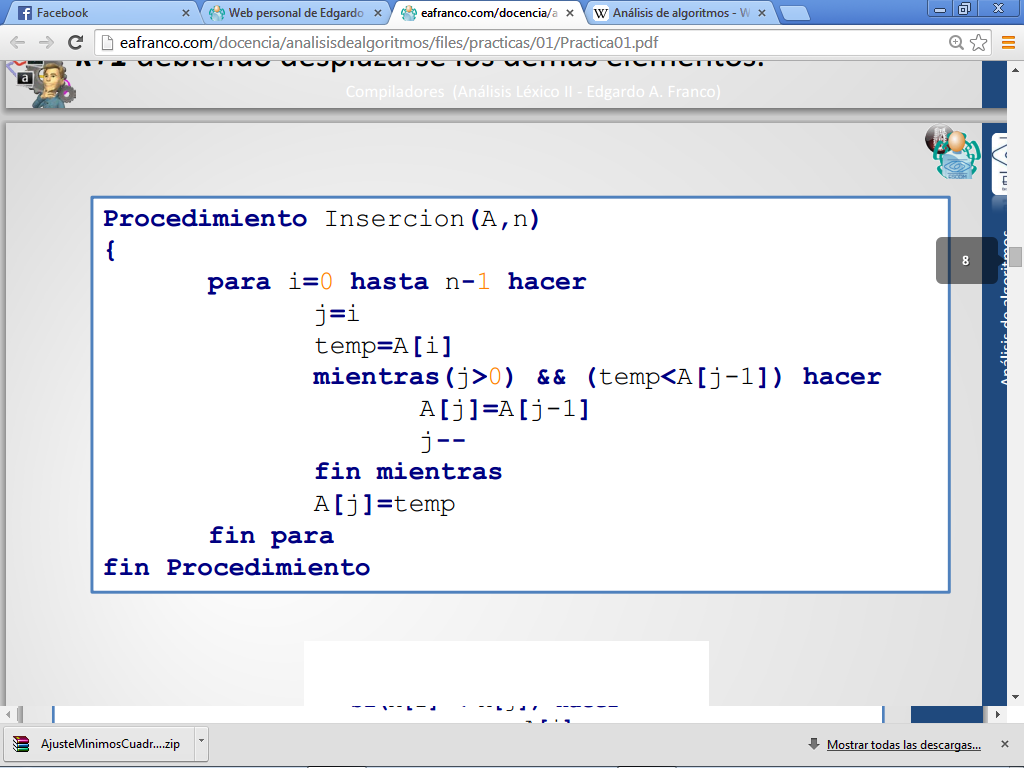
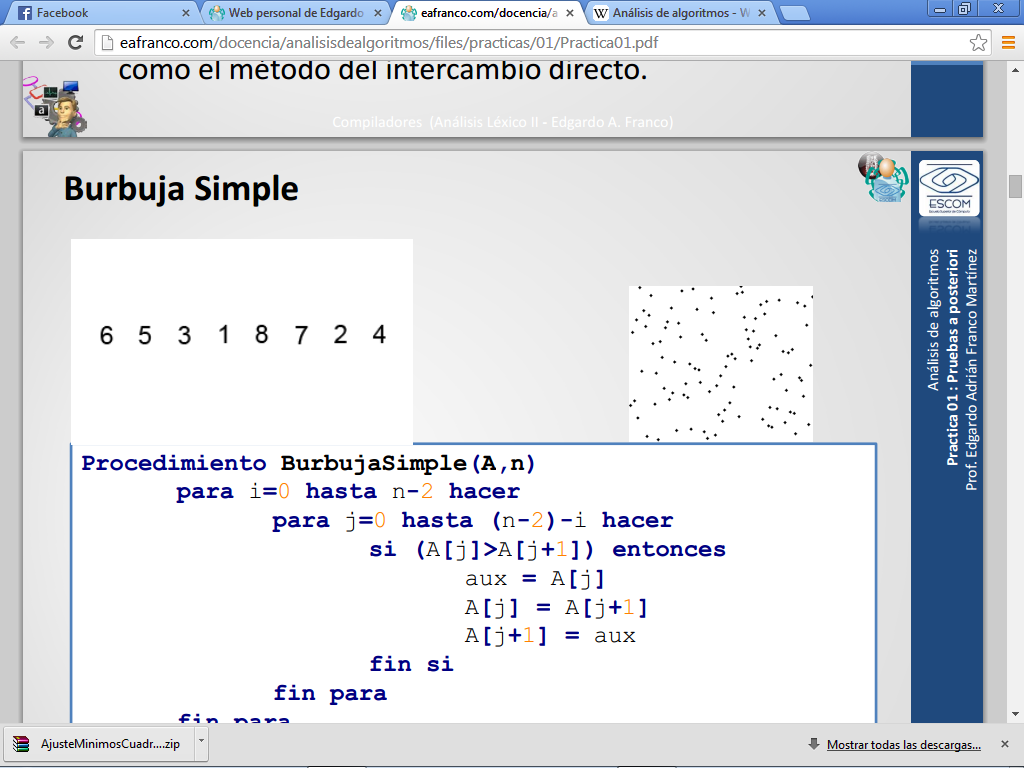
**Burbuja Optimizada**

Sin embargo el burbuja está diseñado para aunque los números ya estén ordenados y en su posición, el algoritmo aun así sigue comparando hasta acabar todas las iteraciones, por ello se optimizo su código con tal evitar los pasos innecesarios.



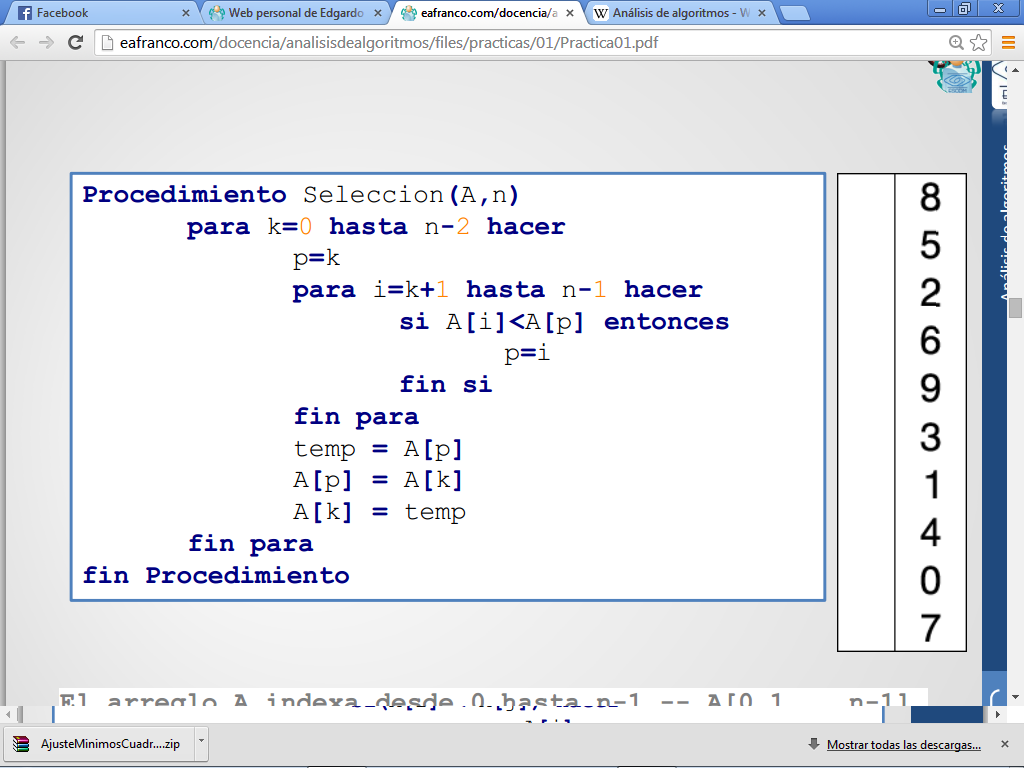
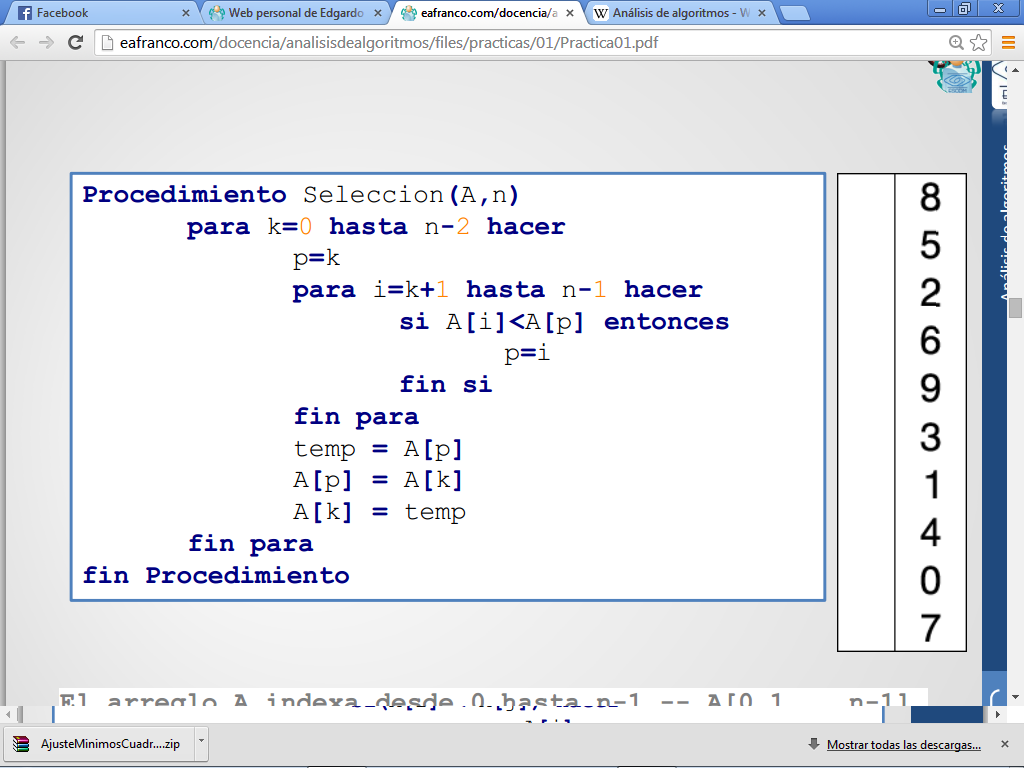
**Ordenamiento por Inserción**

Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay k elementos ordenados de menor a mayor, se toma el elemento k+1 y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se inserta el elemento k+1 debiendo desplazarse los demás elementos.



**Ordenamiento por Selección**

Se basa en buscar el mínimo elemento de la lista e intercambiarlo con el primero, después busca el siguiente mínimo en el resto de la lista y lo intercambia con el segundo, y así sucesivamente.

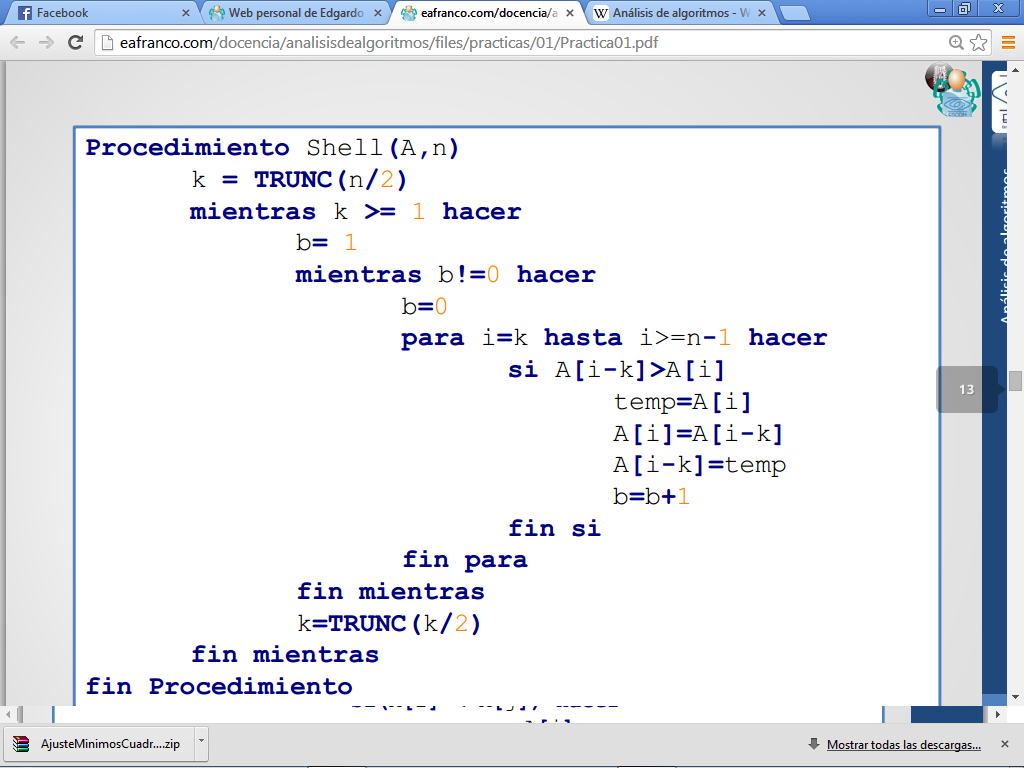
 

**Ordenamiento Shell**

El Shell es una generalización del ordenamiento por inserción, teniendo en cuenta dos observaciones:

1. El ordenamiento por inserción es eficiente si la entrada está "casi ordenada".
2. El ordenamiento por inserción es ineficiente, en general, porque mueve los valores sólo una posición cada vez.

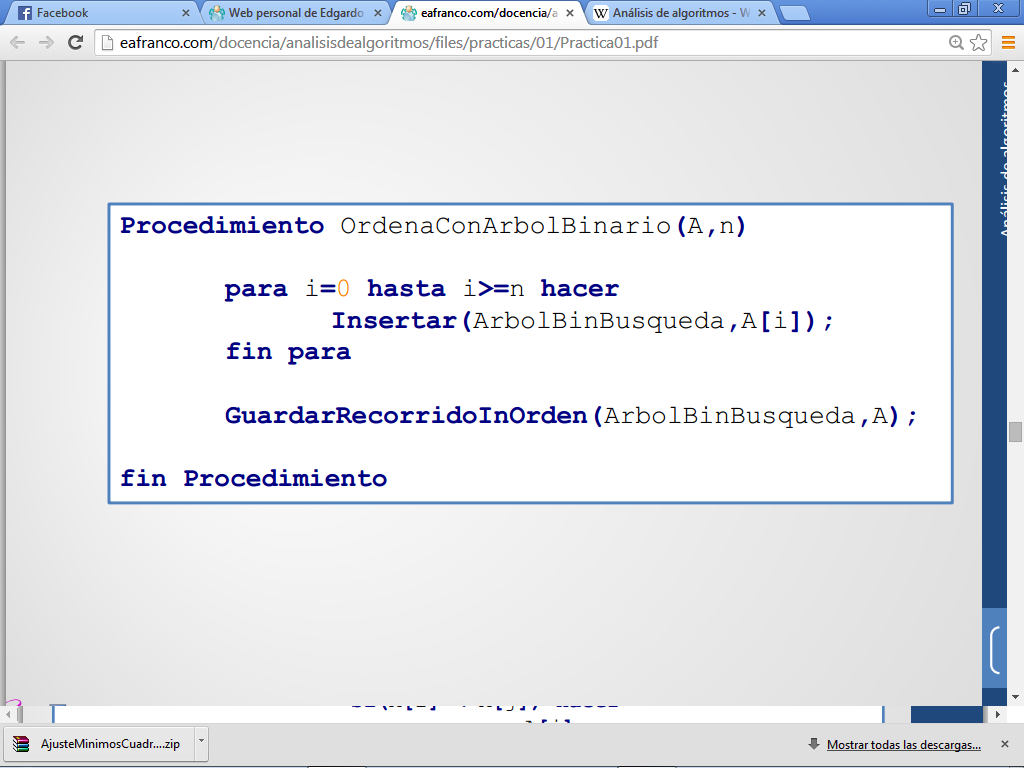
Mejora el ordenamiento por inserción comparando elementos separados por un espacio de varias posiciones. Esto permite que un elemento haga "pasos más grandes" hacia su posición esperada



**Ordenamiento por Árbol de Búsqueda**

El ordenamiento con la ayuda de un árbol binario de búsqueda es muy simple debido a que solo requiere de dos pasos simples.

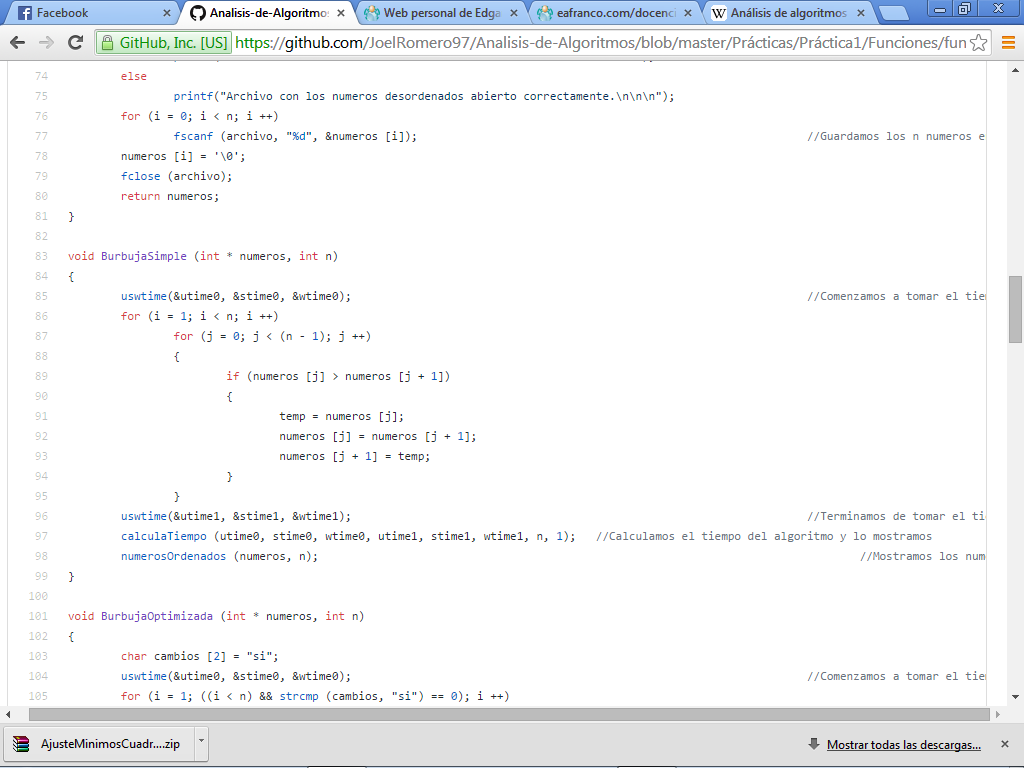
1. Insertar cada uno de los números del vector a ordenar en el árbol binario de búsqueda.
2. Remplazar el vector en desorden por el vector resultante de un recorrido InOrden del Árbol Binario, el cual entregara los números ordenados.



**Implementación**

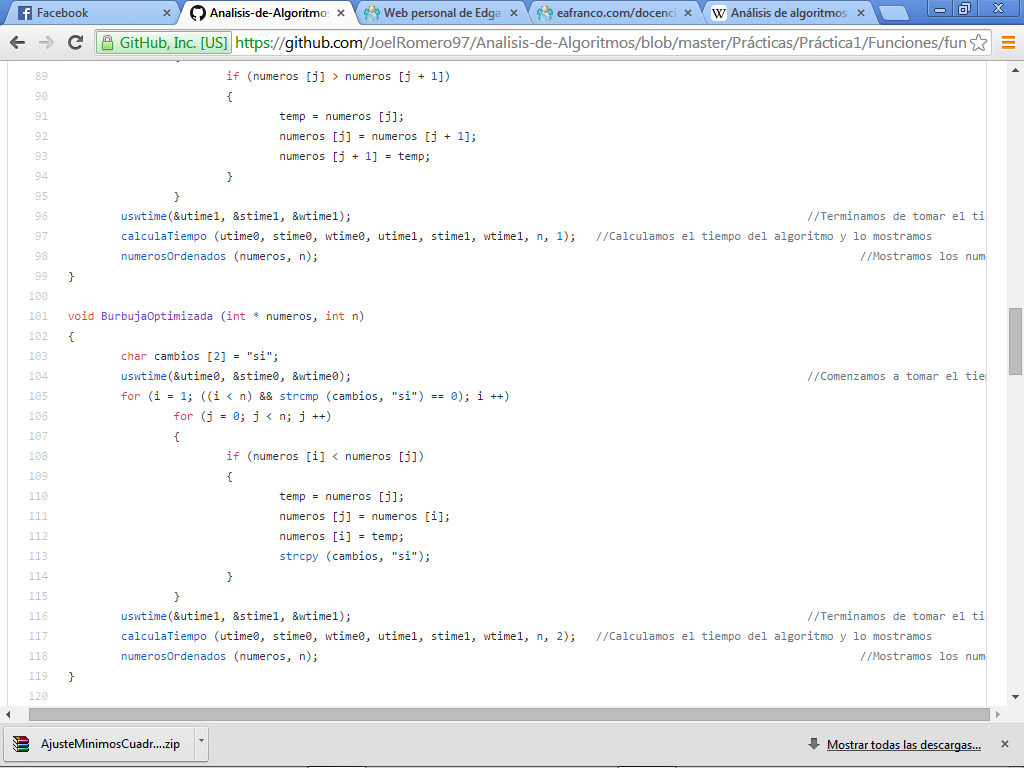
**Burbuja simple**

El burbuja simple mediante los for va desplazándose a lo largo del arreglo mediante el incremento de las variables i y j y manipulándolas como índices del arreglo, a través de una condicional if este verifica si un numero está mal acomodado y lo guarda en un auxiliar y los intercambia de posición con el siguiente del arreglo, sin embargo esto lo hace hasta cumplir la condición de los for, esto pese a que ya estén ordenados



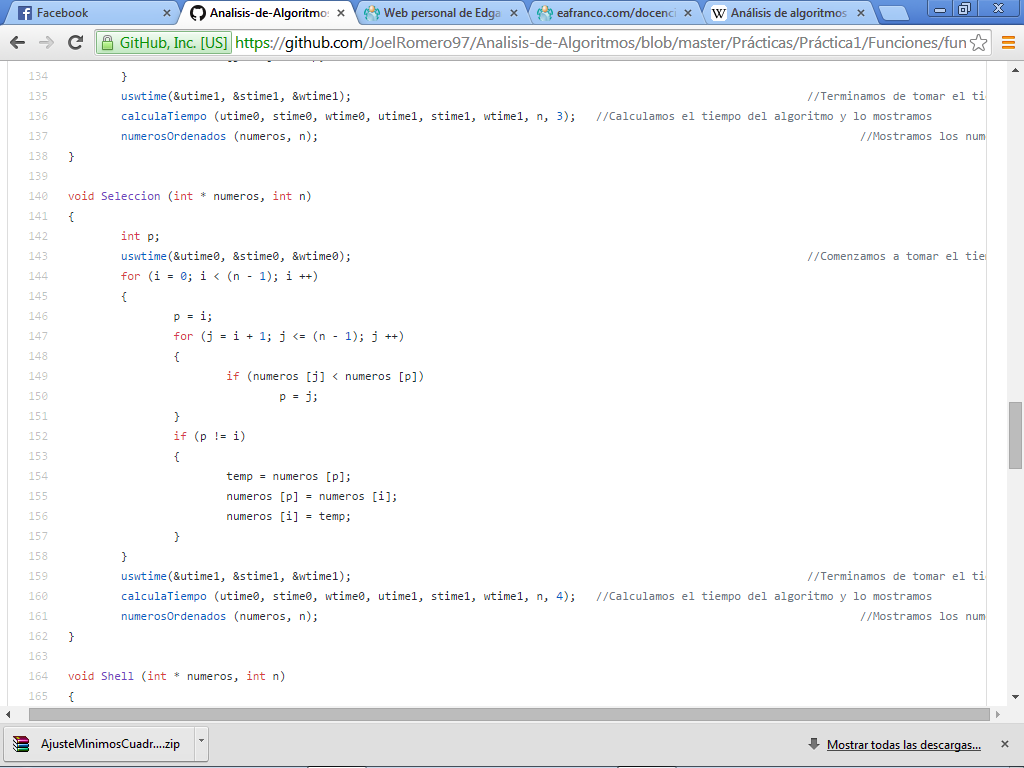
**Burbuja Optimizada**

En la burbuja optimizada de igual forma que el simple recorre el arreglo mediante los for incrementado los índices del arreglo en cuanto encuentra un número mayor lo guarda en auxiliar y lo intercambia con el sig., sin embargo este cuenta con una bandera que le permite saber si los números ya están ordenados para evitar seguir en el ciclo.



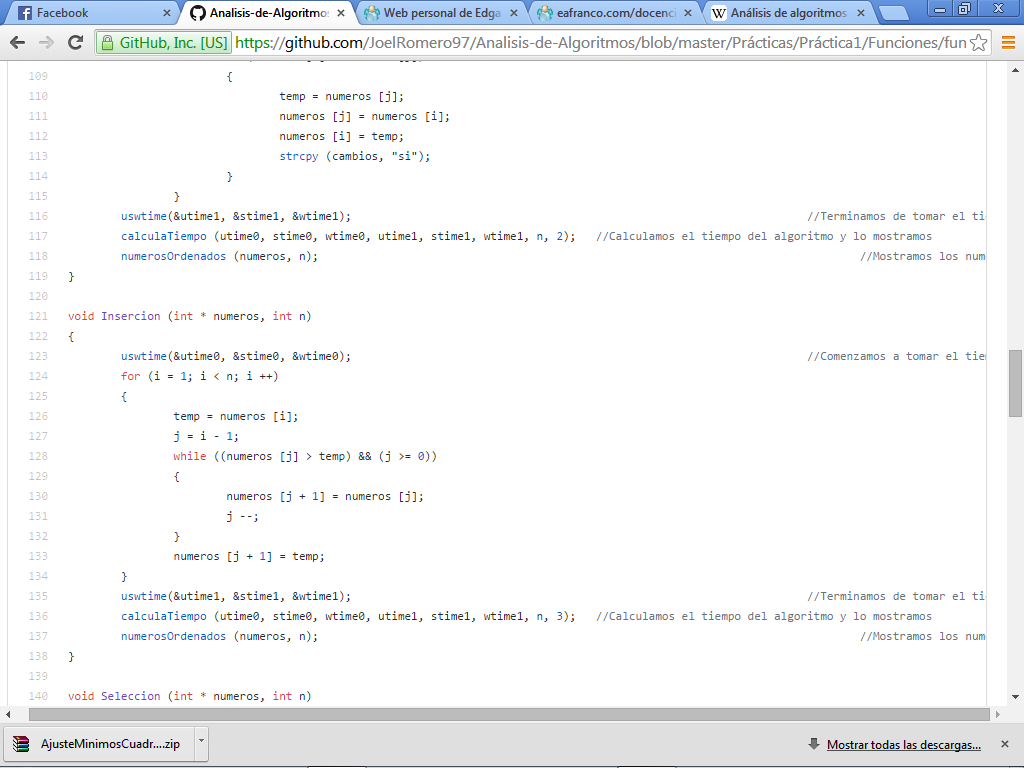
**Selección**

Una razón de porque este código es tan largo a la hora de implementarse es debido a su funcionamiento, este algoritmo recorre todo el arreglo hasta encontrar el número mínimo de la lista y sucesivamente haciendo lo mismo sin embargo si se encuentra en su peor caso puede llevarse mucho tiempo debido que cuenta con muchas operaciones básicas en hacerlo todo.



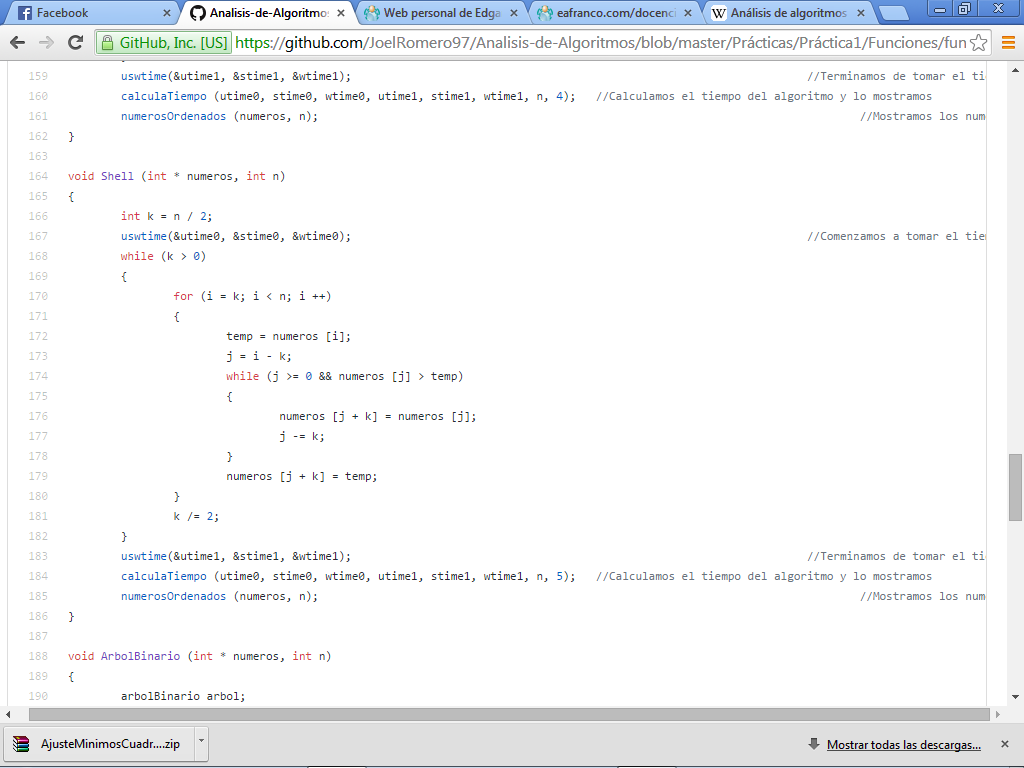
**Inserción**

Este es uno de los algoritmo que tardan más al igual que el de selección solo que en sus peores casos este algoritmo es peor dado que aparte de que recorrería toda la lista este a diferencia del de selección que de una manera más sistemática puede regresarse completamente, el de inserción tiene que recorrer todo el arreglo para buscando la posición idónea del valor desordenado



**Shell**

Este algoritmo toma las ventajas y parece optimizar el comportamiento del algoritmo por insercion, dado que este mejora el ordenamiento por inserción comparando elementos separados por un espacio de varias posiciones. Esto permite que un elemento haga pasos más grandes, haciendo el tiempo de ejecucion del algoritmo el mas rapido de todos.



**Árbol de Búsqueda**

El ordenamiento del árbol de búsqueda es uno de los más sencillos dado que este utiliza la recursividad, y por su misma forma de comportamiento el tiempo ejecución es menor



**Actividades y Pruebas**

Las pruebas de los algoritmos **Burbuja Simple**, **Inserción** y **Shell**, se realizaron en una computadora con las siguientes especificaciones:

- Notebook HP 15-ay009la (ENERGY STAR)

- Procesador Intel Core i5 de 6ta Generación a 2.8 GHz

- 8 GB de SDRAM DDR4-2133 (1 x 8 GB)

- Disco Duro-SATA de 1 TB y 5400 rpm

Las pruebas de los algoritmos **Burbuja Optimizada**, **Selección** y **Árbol Binario**, se realizaron en una computadora con las siguientes especificaciones:

- Laptop Dell Vostro 3550

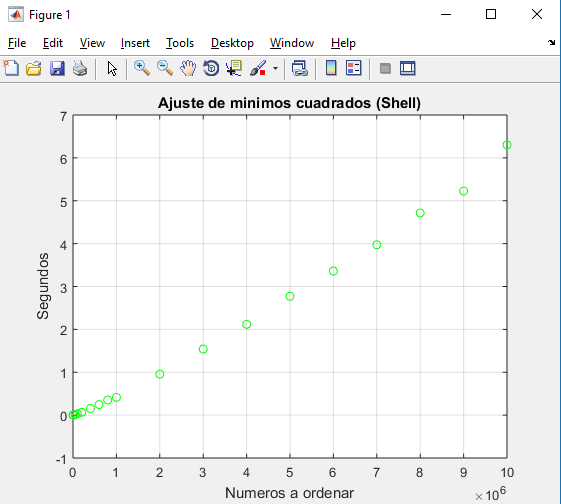
- Procesador Intel Core i7 a 2.70 GHz

- 4GB RAM

- Disco Duro 750Gb

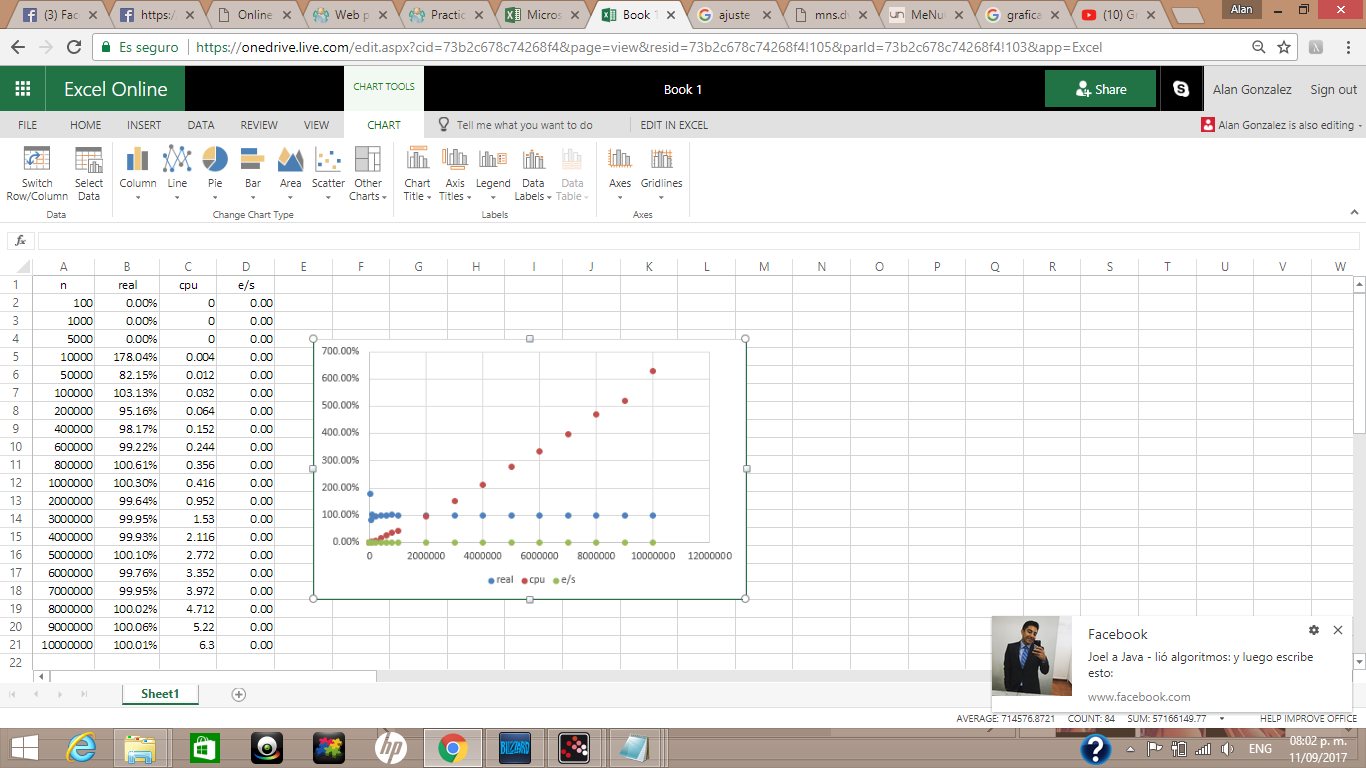
(Las pruebas con los datos medidos y los tiempos están en el archivo AnalizadorTiempos.txt)

**Ajuste de mínimos cuadrados**

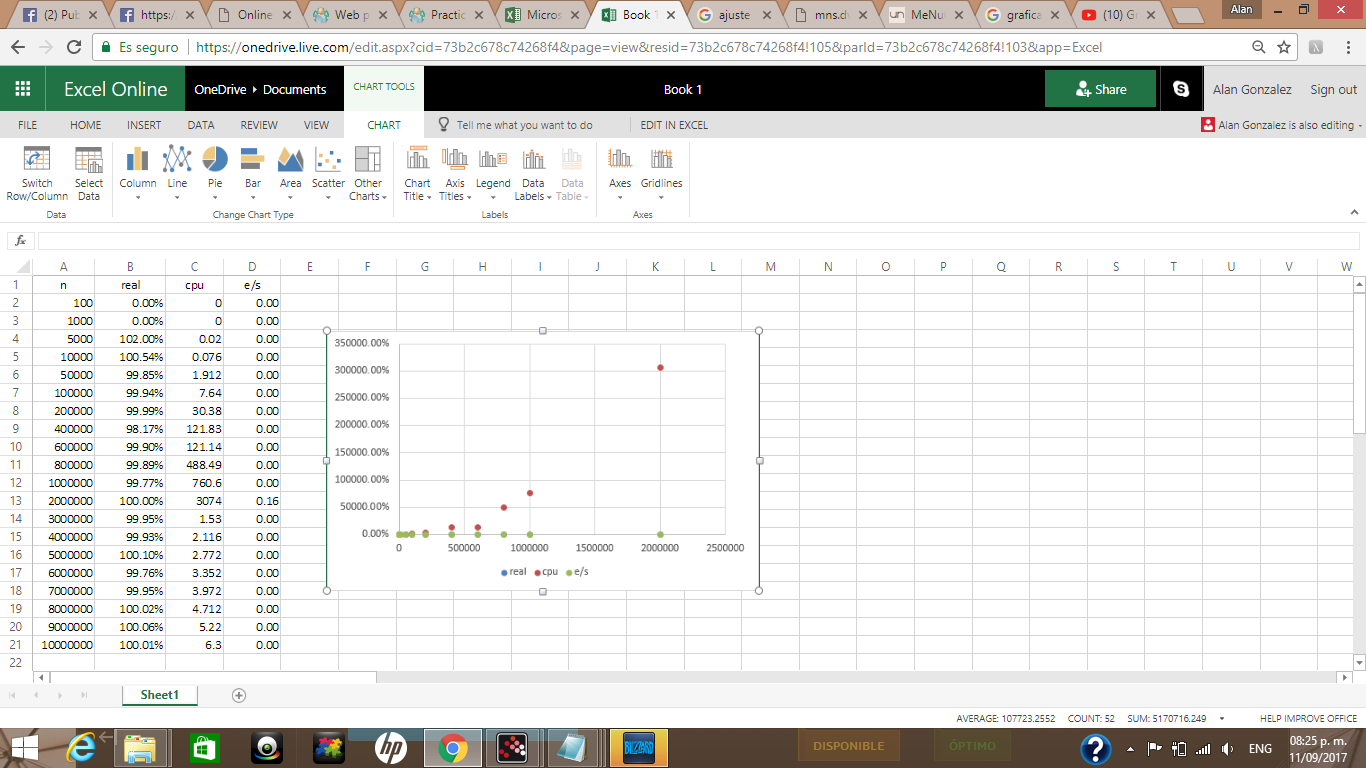


**Graficas de tiempos con muestras**

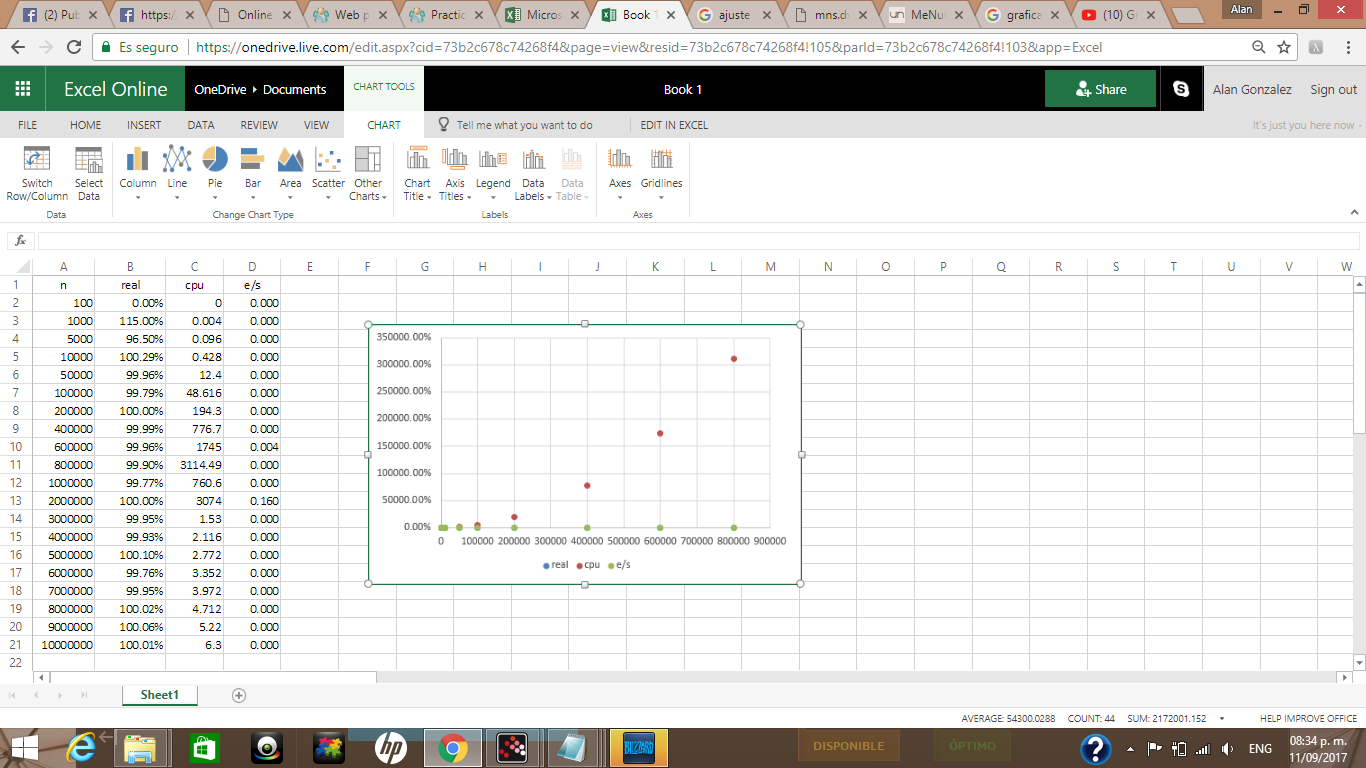
**Selección**



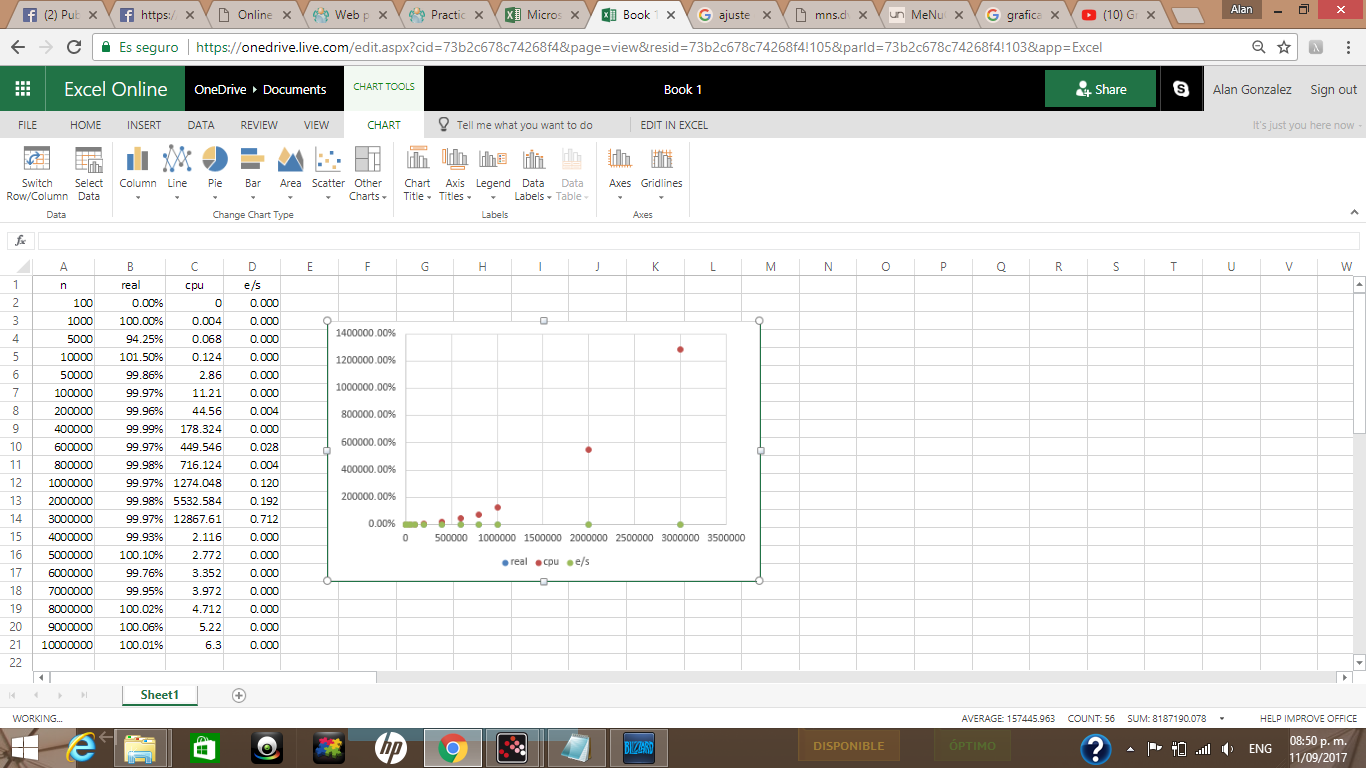
**Burbuja Optimizada**



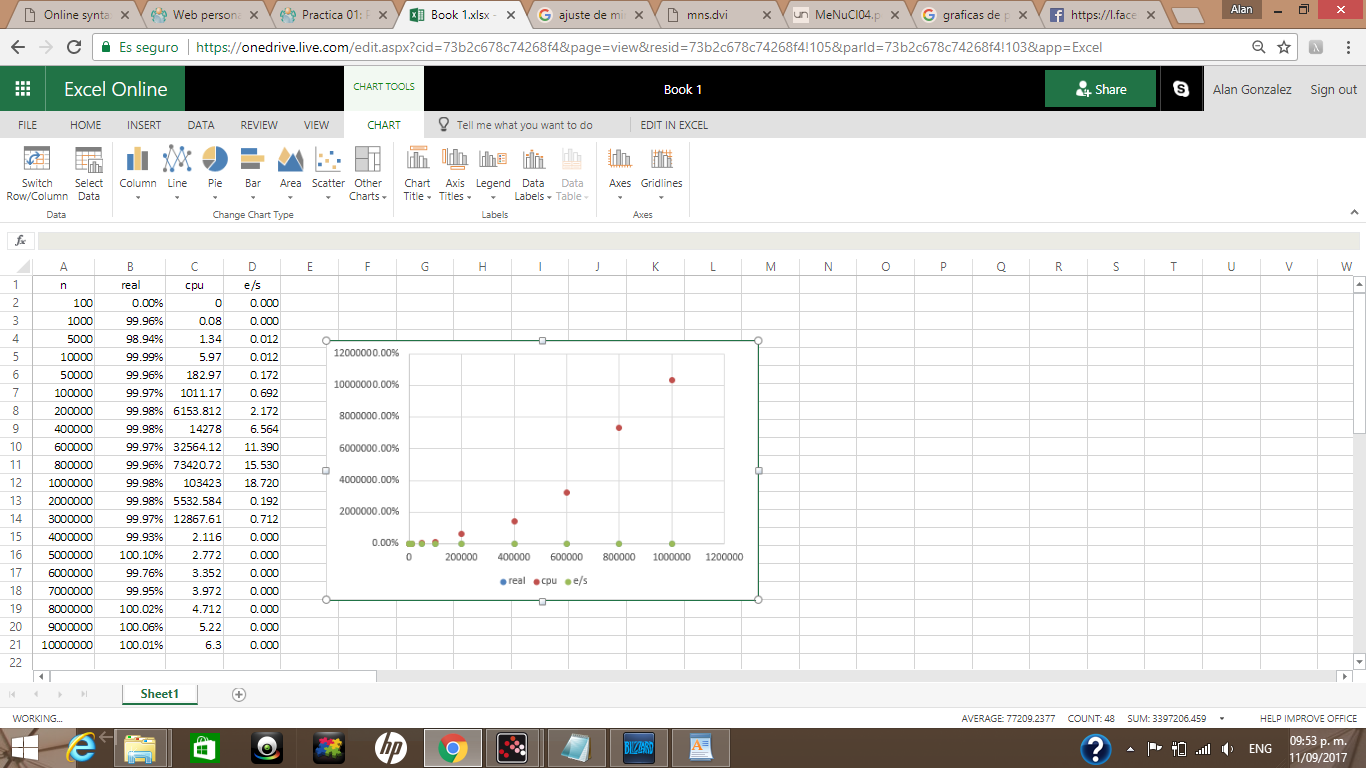
**Shell**



**Insercion**



**Árbol de Búsqueda**



**Cuestionario**

¿Cuál de los 5 algoritmos es más fácil de implementar?

Shell

¿Cuál de los 5 algoritmos es el más difícil de implementar?

Por tardanza Inserción, en nuestro caso por la programación del árbol, el árbol

¿Cuál algoritmo tiene menor complejidad temporal?

Shell

¿Cuál algoritmo tiene mayor complejidad temporal?

Inserción

¿Cuál algoritmo tiene menor complejidad espacial?

¿Por qué?

Árbol, como tal sus operaciones básicas son 2 y las variables que ocupa i, n y un arreglo de A [n+1]

¿Cuál algoritmo tiene mayor complejidad espacial?

¿Por qué?

De acuerdo al analsis que hicimos Burbuja optimizada selección y shell, son los más complejos espacialmente, utilizan un arreglo de A[n] donde eso equivale a n y sus otras variables, por ejemplo selección: k, n , i, y temp dando una complejidad espacial de n+5

¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué?

Si, los algoritmos que tenían más operaciones básicas eran los más complejos por ejemplo, inserción este algoritmo para cantidades grandes en un peor caso tendría que estar siempre regresándose y recorriendo todo el arreglo, desde el análisis del algoritmo ya mostraba complejidad y fue demostrada cuando se hizo la prueba para cantidades grandes.

¿Sus resultados experimentales difieren mucho de los del resto de los equipos? ¿A qué se debe?

Puede ser a la especificación que hicieron a la hora del análisis temporal que hubiesen tomado ciertas condiciones distintas, o de igual manera la forma en que se implementó el código pudo haber sido de una manera más eficiente, por ejemplo, nosotros tuvimos problemas con el árbol.

¿Existió un entorno controlado para realizar las pruebas experimentales? ¿Cuál fue?

Realmente no, una vez que quedaron desarrollados los algoritmos procedimos a probarlos inmediatamente en una sola computadora para poder tener una referencia comparativa entre los algoritmos

¿Qué recomendaciones darían a nuevos equipos para realizar esta práctica?

Pues realmente fue una práctica sencilla, dado que solo fue implementar el código que ya se nos fue otorgado en seudocódigo, pero una recomendación que les daría seria tener un mejor análisis de complejidad a los algoritmos para poder definir de más exacta el comportamiento de los algoritmos y con más tiempo realizar el reporte.

**Errores Detectados**

Donde nos surgió un error, fue en el ordenamiento mediante el árbol de búsqueda, para los 10,000,000 de números al ordenarlos debería haberse tardado alrededor de 15 segundos más o menos, sin embargo, nuestro algoritmo se tarda más que esa cifra, entre el equipo completo se estuvo checando el algoritmo de rodamiento por inorden y no se encontró falla, donde probablemente es donde este fallando seria en la creación de los nodos del árbol.

**Posibles Mejoras**

Como tal los algoritmos de ordenamiento ya tienen bastante funcionalidad y son bastante rápidos y eficientes por ejemplo el Shell siendo el más rápido de todos tardándose alrededor de 4 segundos, en el caso de la burbuja optimizada como su nombre lo dice este ya ha sido mejorado, donde podríamos incluir mejoras serian en el caso del algoritmo de ordenamiento por árbol de búsqueda, dado que ahí fue donde nos surgió el único error que tuvimos.

**Conclusiones**

**Alan Ernesto González Barrios**

Esta práctica me sirvió de introducción formal a algo que previamente ya conocía sin embargo nunca le había prestado una atención como tal, en clases me he dado cuenta de lo verdaderamente fundamental que es el análisis de algoritmos dado que estos optimizados o implementados de una correcta y eficiente pueden permitirnos pruebas matemáticas por ejemplo en un abrir y cerrar de ojos, abriendo paso a la innovación.

**Romero Gamarra Joel Mauricio**

La práctica fue un poco sencilla, ya que únicamente era pasar los algoritmos a código en C para su ejecución, sin embargo, un problema que se me presentó personalmente fue al momento de tratar de hacer el recorrido iterativo inorden para el árbol de búsqueda binaria, fue bastante complicado. Pero, los demás algoritmos, quedaron correctamente, haciendo uso de la modularidad y creación de nuestra librería (Funciones.h) para no tener todo en un mismo archivo, además, la creación de los scripts para ejecutar los algoritmos fue igual interesante ya que no pedimos el valor de n como se hace típicamente con un scanf, si no por parámetro al momento de la ejecución. En general, sin duda el mejor algoritmo de ordenamiento es Shell, ya que es bastante rápido y ordenó los 10,000,000 de números en un promedio de entre 4 y 6 segundos. Es interesante notar la complejidad de los demás algoritmos, porque todos los demás eran demasiado lentos al no estar optimizados, llegando a tardarse tiempos de hasta 4 horas en ordenar menos de la mitad de los números.

**Zavala Pérez Rene**

En esta práctica aprendí que existen muchos algoritmos de búsqueda, pero algunos son más eficientes que otros, esto se vio reflejado en los tiempos de espera de cada algoritmo, ya que en algunas ocasiones los algoritmos más sencillos tendían a no soportar tantos datos o llegaban a ser demasiado lentos, cada algoritmo cumple su función, pero en la práctica observé que para cada necesidad se pueden utilizar los distintos algoritmos vistos en la practica

**ANEXO CÓDIGOS**

**Ordenamiento.c**

#include <stdio.h>

#include "Funciones/funciones.c"

**int** main(**int** argc, **char** **const** \*argv[])

{

**int** n, algoritmo;

**if** (argc < 3)

printf("Error, faltan argumentos.\n");

**else**

{

n = atoi (argv [1]);

algoritmo = atoi (argv [2]);

}

seleccionarAlgoritmo (algoritmo, n);

**return** 0;

}

**Funciones.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "../Arbol/Arbol.c"

#include "../MedirTiempo/tiempo.c"

#include "funciones.h"

**int** i, j, temp; //Variables globales para manejar los ciclos y temporal

**double** utime0, stime0, wtime0,utime1, stime1, wtime1;

**void** seleccionarAlgoritmo (**int** algoritmo, **int** n)

{

**int** \* numeros = (**int** \*) malloc (**sizeof** (**int**) \* n); //Creamos un arreglo din�mico de tama�o n para los numeros

numeros = leerArchivo (numeros, n); //Obtenemos los n numeros en un arreglo dinamico

**switch** (algoritmo)

{

**case** 1:

BurbujaSimple (numeros, n); //Ordenamos los numeros con el primer algoritmo

**break**;

**case** 2:

BurbujaOptimizada (numeros, n); //Ordenamos los numeros con el segundo algoritmo

**break**;

**case** 3:

Insercion (numeros, n); //Ordenamos los numeros con el tercer algoritmo

**break**;

**case** 4:

Seleccion (numeros, n); //Ordenamos los numeros con el cuarto algoritmo

**break**;

**case** 5:

Shell (numeros, n); //Ordenamos los numeros con el quinto algoritmo

**break**;

**case** 6:

ArbolBinario (numeros, n); //Ordenamos los numeros con el sexto algoritmo

**break**;

**default**:

printf("Opci�n invalida.\n\n");

exit (0);

}

}

**int** \* leerArchivo (**int** \* numeros, **int** n)

{

FILE \* archivo;

archivo = fopen ("numeros.txt", "r"); //Abrimos el archivo de numeros para lectura

**if** (archivo == NULL)

{

printf("Error al abrir el archivo con los numeros desordenados .\n\n\n");

exit (0);

}

**for** (i = 0; i < n; i ++)

fscanf (archivo, "%d", &numeros [i]); //Guardamos los n numeros en un arreglo dinamico

numeros [i] = '\0';

fclose (archivo);

**return** numeros;

}

**void** BurbujaSimple (**int** \* numeros, **int** n)

{

uswtime(&utime0, &stime0, &wtime0); //Comenzamos a tomar el tiempo del algoritmo

**for** (i = 1; i < n; i ++)

**for** (j = 0; j < (n - 1); j ++)

{

**if** (numeros [j] > numeros [j + 1])

{

temp = numeros [j];

numeros [j] = numeros [j + 1];

numeros [j + 1] = temp;

}

}

uswtime(&utime1, &stime1, &wtime1); //Terminamos de tomar el tiempo del algoritmo

calculaTiempo (utime0, stime0, wtime0, utime1, stime1, wtime1, n, 1); //Calculamos el tiempo del algoritmo y lo mostramos

numerosOrdenados (numeros, n); //Mostramos los numeros ordenados por el algoritmo

}

**void** BurbujaOptimizada (**int** \* numeros, **int** n)

{

**char** cambios [2] = "si";

uswtime(&utime0, &stime0, &wtime0); //Comenzamos a tomar el tiempo del algoritmo

**for** (i = 1; ((i < n) && strcmp (cambios, "si") == 0); i ++)

**for** (j = 0; j < n; j ++)

{

**if** (numeros [i] < numeros [j])

{

temp = numeros [j];

numeros [j] = numeros [i];

numeros [i] = temp;

strcpy (cambios, "si");

}

}

uswtime(&utime1, &stime1, &wtime1); //Terminamos de tomar el tiempo del algoritmo

calculaTiempo (utime0, stime0, wtime0, utime1, stime1, wtime1, n, 2); //Calculamos el tiempo del algoritmo y lo mostramos

numerosOrdenados (numeros, n); //Mostramos los numeros ordenados por el algoritmo

}

**void** Insercion (**int** \* numeros, **int** n)

{

uswtime(&utime0, &stime0, &wtime0); //Comenzamos a tomar el tiempo del algoritmo

**for** (i = 1; i < n; i ++)

{

temp = numeros [i];

j = i - 1;

**while** ((numeros [j] > temp) && (j >= 0))

{

numeros [j + 1] = numeros [j];

j --;

}

numeros [j + 1] = temp;

}

uswtime(&utime1, &stime1, &wtime1); //Terminamos de tomar el tiempo del algoritmo

calculaTiempo (utime0, stime0, wtime0, utime1, stime1, wtime1, n, 3); //Calculamos el tiempo del algoritmo y lo mostramos

numerosOrdenados (numeros, n); //Mostramos los numeros ordenados por el algoritmo

}

**void** Seleccion (**int** \* numeros, **int** n)

{

**int** p;

uswtime(&utime0, &stime0, &wtime0); //Comenzamos a tomar el tiempo del algoritmo

**for** (i = 0; i < (n - 1); i ++)

{

p = i;

**for** (j = i + 1; j <= (n - 1); j ++)

{

**if** (numeros [j] < numeros [p])

p = j;

}

**if** (p != i)

{

temp = numeros [p];

numeros [p] = numeros [i];

numeros [i] = temp;

}

}

uswtime(&utime1, &stime1, &wtime1); //Terminamos de tomar el tiempo del algoritmo

calculaTiempo (utime0, stime0, wtime0, utime1, stime1, wtime1, n, 4); //Calculamos el tiempo del algoritmo y lo mostramos

numerosOrdenados (numeros, n); //Mostramos los numeros ordenados por el algoritmo

}

**void** Shell (**int** \* numeros, **int** n)

{

**int** k = n / 2;

uswtime(&utime0, &stime0, &wtime0); //Comenzamos a tomar el tiempo del algoritmo

**while** (k > 0)

{

**for** (i = k; i < n; i ++)

{

temp = numeros [i];

j = i - k;

**while** (j >= 0 && numeros [j] > temp)

{

numeros [j + k] = numeros [j];

j -= k;

}

numeros [j + k] = temp;

}

k /= 2;

}

uswtime(&utime1, &stime1, &wtime1); //Terminamos de tomar el tiempo del algoritmo

calculaTiempo (utime0, stime0, wtime0, utime1, stime1, wtime1, n, 5); //Calculamos el tiempo del algoritmo y lo mostramos

numerosOrdenados (numeros, n); //Mostramos los numeros ordenados por el algoritmo

}

**void** ArbolBinario (**int** \* numeros, **int** n)

{

arbolBinario arbol;

elemento e;

posicion pos;

InicializarArbol (&arbol);

printf("\n\n\nInsertando numeros en el arbol binario...\n\n");

**for** (i = 0; i < n; i ++)

{

e.numero = numeros [i]; //Guardamos cada numero en el elemento 'numero' del nodo

InsertarABB (&arbol, e); //Insertamos los numeros en el arbol

}

free (numeros);

printf("\n\nNumeros ordenados...\n\n");

pos = Raiz (&arbol);

uswtime(&utime0, &stime0, &wtime0); //Comenzamos a tomar el tiempo del algoritmo

Inorden (&arbol, pos); //Ordenamos y mostramos los numeros ordenados por el algoritmo

uswtime(&utime1, &stime1, &wtime1); //Terminamos de tomar el tiempo del algoritmo

calculaTiempo (utime0, stime0, wtime0, utime1, stime1, wtime1, n, 6); //Calculamos el tiempo del algoritmo y lo mostramos

}

**void** calculaTiempo (**double** utime0, **double** stime0, **double** wtime0, **double** utime1, **double** stime1, **double** wtime1, **int** n, **int** algoritmo)

{

**char** \* algorithm = (**char** \*) malloc (**sizeof** (**char**));

**switch** (algoritmo)

{

**case** 1:

strcpy (algorithm, "Simple Bubble Sort");

**break**;

**case** 2:

strcpy (algorithm, "Optimized Bubble Sort");

**break**;

**case** 3:

strcpy (algorithm, "Insertion Sort");

**break**;

**case** 4:

strcpy (algorithm, "Selection Sort");

**break**;

**case** 5:

strcpy (algorithm, "Shell Sort");

**break**;

**case** 6:

strcpy (algorithm, "Binary Tree Sort");

**break**;

}

printf("Time with %s algorithm with n = %d\n", algorithm, n);

printf("\n\n");

printf("Tiempo total: %.10f s\n", wtime1 - wtime0);

printf("Tiempo de procesamiento en CPU: %.10f s\n", utime1 - utime0);

printf("Tiempo en acci�nes de E/S: %.10f s\n", stime1 - stime0);

printf("CPU/Wall: %.10f %% \n",100.0 \* (utime1 - utime0 + stime1 - stime0) / (wtime1 - wtime0));

printf("\n\n");

//Mostrar los tiempos en formato exponecial

printf("\n\n");

printf("Tiempo total: %.10e s\n", wtime1 - wtime0);

printf("Tiempo de procesamiento en CPU: %.10e s\n", utime1 - utime0);

printf("Tiempo en acci�nes de E/S: %.10e s\n", stime1 - stime0);

printf("CPU/Wall: %.10f %% \n",100.0 \* (utime1 - utime0 + stime1 - stime0) / (wtime1 - wtime0));

printf("\n\n");

}

**void** numerosOrdenados (**int** \* numeros, **int** n)

{

FILE \* archivoOrdenado;

archivoOrdenado = fopen ("numerosOrdenados.txt", "w");

**if** (archivoOrdenado == NULL)

{

printf("Error al crear el archivo con los %d numeros ordenados \n", n);

exit (0);

}

**for** (i = 0; i < n; i ++)

fprintf(archivoOrdenado, "%d\n", numeros [i]); //Imprimimos el arreglo de numeros

free (numeros);

fclose (archivoOrdenado);

}

**Funciones.h**

**void** seleccionarAlgoritmo (**int** algoritmo, **int** n);

**int** \* leerArchivo (**int** \* numeros, **int** n);

**void** BurbujaSimple (**int** \* numeros, **int** n);

**void** BurbujaOptimizada (**int** \* numeros, **int** n);

**void** Insercion (**int** \* numeros, **int** n);

**void** Seleccion (**int** \* numeros, **int** n);

**void** Shell (**int** \* numeros, **int** n);

**void** ArbolBinario (**int** \* numeros, **int** n);

**void** calculaTiempo (**double** utime0, **double** stime0, **double** wtime0, **double** utime1, **double** stime1, **double** wtime1, **int** n, **int** algoritmo);

**void** numerosOrdenados (**int** \* numeros, **int** n);

**Árbol.c**

/\*TAD Arbol binario

ESTRUCTURAS DE DATOS 1CM7

ABRIL 2017

\*/

#include <stdlib.h>

#include "Arbol.h"

**int** \* InsertarABB (**int** \* numeros, **int** n)

{

arbolBinario \*miRaiz; //Se declara la Ra�z de arbolBinario

miRaiz = (arbolBinario \*) malloc (**sizeof**(arbolBinario)); //Se reserva espacio de memoria para la variable

miRaiz = NULL;

**for** (**int** i = 0; i < n; ++i)

Insertar(&miRaiz,numeros[i]);

numeros = Inorden(miRaiz,numeros);

}

**void** Insertar(arbolBinario \*\*Raiz,**int** DatoRecibido)

{

arbolBinario \*\*Auxiliar = NULL; //Se crea un nodo auxiliar

Auxiliar = Raiz; //Se iguala el auxiliar numeros la raiz

**if** (\*Raiz == NULL) // si no hay dato se crea un nodo nuevo

\*Raiz = InicializarArbol(DatoRecibido);

**else**

{

**while**(\*Auxiliar != NULL) //Hasta que se encuentra el nodo vacio

{

**if** (DatoRecibido < (\*Auxiliar)->dato) // Si el numero a ingresar es menor en Auxiliar

Auxiliar = &((\*Auxiliar)->izq); // Se va al nodo izquierdo

**else** **if** (DatoRecibido > (\*Auxiliar)->dato) //Si el numero a ingresar es mayor en Auxiliar

Auxiliar = &((\*Auxiliar)->der); //Se va al nodo derecho

}

\*Auxiliar = InicializarArbol (DatoRecibido);// Ya se llego a la posicion adecuada, agrega el nuevo nodo

}

}

/\*

recibe<-�rbol(A), posici�n (P);

Inorden (A,P)

Efecto: Recibe un �rbol binario A y una posici�n P, realiza el recorrido izquierdo -> raiz -> derecho a partir de la posicion P.

Requerimientos: El �rbol binario A es no vac�o y la posici�n P es una posici�n valida.

\*/

**int** \* Inorden (arbolBinario \* Raiz, **int** \*numeros)

{

**int** i = 0;

arbolBinario \*Auxiliar,\*Recorrido;

Auxiliar = Raiz;

**while** (Auxiliar !=NULL)

{

**if** (Auxiliar->izq == NULL)

{

numeros[i] = Auxiliar->dato;

Auxiliar = Auxiliar->der;

i++;

}

**else**

{

Recorrido = Auxiliar->izq;

**while**(Recorrido->der !=NULL && Recorrido->der != Auxiliar)

Recorrido = Recorrido->der;

**if**(Recorrido->der == NULL)

{

Recorrido->der = Auxiliar;

Auxiliar = Auxiliar->izq;

}

**else**

{

Recorrido->der = NULL;

numeros[i] = Auxiliar->dato;

Auxiliar = Auxiliar->der;

i++;

}

}

}

**return** numeros;

}

arbolBinario \* InicializarArbol (**int** numDato)

{

arbolBinario \*A = malloc(**sizeof**(nodo)); //Se reserva memoria paara el arbol

A->dato= numDato;

A->izq=NULL; //Sin nodo izquierdo

A->der=NULL; //Sin nodo derecho

**return** A;

}

**Árbol.h**

#define TRUE 1

#define FALSE 0

**typedef** **struct** nodo

{

**struct** nodo \*izq, \*der;

**int** dato;

}nodo;

**typedef** nodo arbolBinario;

**int** \* InsertarABB(**int** \*numeros, **int** n);

/\*

recibe<-�rbol(A), elemento (E);

InsertarABB (A, E);

Efecto: Recibe un �rbol binario A y un elemento E, compara si E es menor o mayor a la raiz, si E es menor lo inserta a la izquierda, y si es

mayor inserta a E a la derecha.

Requerimientos: El �rbol binario A es no vac�o y el elemento E tiene un numero a ser comparado.

\*/

**void** Insertar(arbolBinario \*\*Raiz,**int** DatoRecibido);

arbolBinario \* InicializarArbol (**int** numDato);

/\*

recibe<-�rbol(A);

Initialize (A)

Efecto: Recibe un �rbol binario A y lo inicializa para su trabajo normal.

\*/

**int** \* Inorden (arbolBinario \*Raiz, **int** \*numeros);

/\*

recibe<-�rbol(A), posici�n (P);

Inorden (A,P)

Efecto: Recibe un �rbol binario A y una posici�n P, realiza el recorrido izquierdo -> raiz -> derecho a partir de la posicion P.

Requerimientos: El �rbol binario A es no vac�o y la posici�n P es una posici�n valida.

\*/

**ANEXO SCRIPTS**

**BurbujaSimple.sh**

#!/bin/bash

**echo** Ejecutando Algoritmo Burbuja Simple

#Burbuja simple

**cd** /home/joel/Escritorio

gcc Ordenamiento**.**c -o Ordenamiento

**.**/Ordenamiento 100 1 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000 1 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 5000 1 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 10000 1 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 50000 1 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 100000 1 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 200000 1 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 400000 1 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 600000 1 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 800000 1 >> AnalizadorTiempos**.**txt

**cd** /home/joel/Escritorio/Scripts

chmod 777 BurbujaOptimizada**.**sh

**.**/BurbujaOptimizada.sh

**BurbujaOptimizada.sh**

#!/bin/bash

**echo** Ejecutando Algoritmo Burbuja Optimizada

#Burbuja Optimizada

**cd** /home/joel/Escritorio

gcc Ordenamiento**.**c -o Ordenamiento

**.**/Ordenamiento 100 2 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000 2 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 5000 2 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 10000 2 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 50000 2 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 100000 2 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 200000 2 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 400000 2 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 600000 2 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 800000 2 >> AnalizadorTiempos**.**txt

**cd** /home/joel/Escritorio/Scripts

chmod 777 Insercion**.**sh

**.**/Insercion.sh

**Insercion.sh**

#!/bin/bash

**echo** Ejecutando Algoritmo Insercion

#Insercion

**cd** /home/joel/Escritorio

gcc Ordenamiento**.**c -o Ordenamiento

**.**/Ordenamiento 100 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 5000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 10000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 50000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 100000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 200000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 400000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 600000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 800000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000000 3 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 2000000 3 >> AnalizadorTiempos**.**txt

**cd** /home/joel/Escritorio/Scripts

chmod 777 Seleccion**.**sh

**.**/Seleccion.sh

**Selección.sh**

#!/bin/bash

**echo** Ejecutando Algoritmo Seleccion

#Seleccion

**cd** /home/joel/Escritorio

gcc Ordenamiento**.**c -o Ordenamiento

**.**/Ordenamiento 100 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 5000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 10000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 50000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 100000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 200000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 400000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 600000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 800000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 2000000 4 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 3000000 4 >> AnalizadorTiempos**.**txt

**cd** /home/joel/Escritorio/Scripts

chmod 777 Shell**.**sh

**.**/Shell.sh

**Shell.sh**

#!/bin/bash

**echo** Ejecutando Algoritmo Shell

#Shell

**cd** /home/joel/Escritorio

gcc Ordenamiento**.**c -o Ordenamiento

**.**/Ordenamiento 100 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 5000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 10000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 50000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 100000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 200000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 400000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 600000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 800000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 2000000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 3000000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 4000000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 5000000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 6000000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 7000000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 8000000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 9000000 5 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 10000000 5 >> AnalizadorTiempos**.**txt

**cd** /home/joel/Escritorio/Scripts

chmod 777 ArbolBusqueda**.**sh

**.**/ArbolBusqueda.sh

**ArbolBusqueda.sh**

#!/bin/bash

**echo** Ejecutando Algoritmo Arbol de Busqueda

#Arbol Busqueda

**cd** /home/joel/Escritorio

gcc Ordenamiento**.**c -o Ordenamiento

**.**/Ordenamiento 100 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 5000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 10000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 50000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 100000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 200000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 400000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 600000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 800000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 1000000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 2000000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 3000000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 4000000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 5000000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 6000000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 7000000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 8000000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 9000000 6 >> AnalizadorTiempos**.**txt

**.**/Ordenamiento 10000000 6 >> AnalizadorTiempos**.**txt

**ANEXO FOTO EQUIPO**

**ANEXO BIBLIOGRAFÍA**

http://eafranco.com/docencia/analisisdealgoritmos/files/practicas/01/Practica01.pdf