Análisis de algoritmos

Integrantes:

Macías Castillo Josué

Torres Hernández Eduardo

Vargas Romero Erick Efraín

Prof.: Franco Martínez Edgardo Adrián

Grupo: 3CM2

Práctica 01: Pruebas a posteriori

(Algoritmos de ordenamiento)

Algoritmos de ordenamiento

Bubble sort

Descripción

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

El método de la burbuja es uno de los más simples, es tan fácil como comparar todos los elementos de una lista contra todos, si se cumple que uno es mayor o menor a otro, entonces los intercambia de posición.

Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo.

Código

```
for (i=0; i<N; i++){
      for(j=0; j<N-1; j++){
            if(vector[j]>vector[j+1]){
                  aux= vector[j];
                  vector[j]= vector[j+1];
                  vector[j+1]=aux;
            }
      }
}
```

Bubble sort optimizado

Descripción

Como al final de cada iteración el elemento mayor queda situado en su posición, ya no es necesario volverlo a comparar con ningún otro número, reduciendo así el número de comparaciones por iteración, además pude existir la posibilidad que realizar iteraciones de más si el arreglo ya fue ordenado totalmente

Código

```
while(i<size && flag==true){
      flag= false;
      for (j=0;  j<size-1;  j++){
            if(array[j] > array[j+1]){
                  aux= array[j];
                  array[j]= array[j+1];
                  array[j+1]= aux;
                  flag= true;
            }
      }
      i++;
}
```

Insertion sort

Descripción

Es una manera muy natural de ordenar para un ser humano, y puede usarse fácilmente para ordenar un mazo de cartas numeradas en forma arbitraria.

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.

Código

```
for(k = 0; k < n; k++)
{
    pos = k;
    temp = numeros[k];
    while(pos > 0 && temp < numeros[pos - 1] )
    {
        numeros[pos] = numeros[pos - 1];
        pos--;
    }
    numeros[pos] = temp;
}
```

Análisis temporal

Selection sort

Descripción

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.

Buscar el mínimo elemento entre una posición i y el final de la lista Intercambiar el mínimo con el elemento de la posición i

Código

```
for(k = 0; k < n; k++)
{
    min = k;
    for(j = k + 1; j < n; j++)
    {
        if(numeros[j] < numeros[min])
        {
           min = j;
        }
    }
    aux = numeros[k];
    numeros[k] = numeros[min];
    numeros[min] = aux;
}
```

Shell sort

Descripción

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.

El algoritmo Shell 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. Los pasos múltiples sobre los datos se hacen con tamaños de espacio cada vez más pequeños. El último paso del ordenamiento Shell es un simple ordenamiento por inserción, pero para entonces, ya está garantizado que los datos del vector están casi ordenados.

Shell propone que se haga sobre el arreglo una serie de ordenaciones basadas en la inserción directa, pero dividiendo el arreglo original en varios sub-arreglos tales que cada elemento esté separado k elementos del anterior (a esta separación a menudo se le llama salto o gap) Se debe empezar con k=n/2, siendo n el número de elementos del arreglo, y utilizando siempre la división entera (TRUNC)

Después iremos variando k haciéndolo más pequeño mediante sucesivas divisiones por 2, hasta llegar a k=1.

Código

```
      while(div>1){     //Mientras la division sea mayor a 1 se
cumplira
            div= div/2; //Cada iteracion dividira entre 2
            flag= true; //La bandera indica que si ha habido una
division entre 2
            while(flag==true){      //La bandera indica que hay
ordenamientos por realizar
                  flag= false;      //Inmediatamente se desactiva la
bandera para posteriormente en caso de que hayan reemplazos volver
activarla
                  i=0;
                  while(i+div <N){  //Indicamos que no puede salirse
del tamanio del arreglo
                        if(vector[i]>vector[i+div]){  //Se realiza la
comparacion entre los elementos del arreglo
                              aux= vector[i];   // Se realiza el
almacen del dato que vamos a reemplazar en los siguientes dos pasos
                              vector[i]=vector[i+div];
                              vector[i+div]= aux;
                              flag= true; //Se indica a la bandera
que se ha encontrado una comparacion y se ha reemplazado algun valor
                        }
                        i++;        //Se incrementa el subindice del
arreglo
                  }
            }
      }
```

Tree sort

Descripción

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.

La eficiencia de este algoritmo está dada según la eficiencia en la implementación del árbol binario de búsqueda, lo que puede resultar mejor que otros algoritmos de ordenamiento.

Código

```
void insert(BinaryTree &sub_tree, int data){

    //Comparamos si sub árbol es nulo
    if(!sub_tree){
        //Si es un sub árbol nulo es creado
        sub_tree = newTree(data);
    }
    //Comparamos el valor de la raíz con el nuevo valor recibido
    else if(data >= sub_tree -> value){
        //SI el valor es mayor o igual se añade un nodo  a la
derecha de la raíz
        insert(sub_tree->right, data);
    } else {
        //Si es menor el nodo se añade a la izquierda de la raiz
        insert(sub_tree -> left, data);
    }
}
```

Mediciones

Para esta sección se ha medido cuanto tarda cada uno de los algoritmos en ordenar el archivo completo, además de que es posible comparar el tiempo real vs el tiempo de CPU

Bubble sort

Tabla

Tamaño (N)	Temp Proc	Tiempo Real	E/S
10,000,000	Indefinido	Indefinido	indefinido

Gráfica tiempo real

Debido a que el algoritmo es demasiado ineficiente no fue posible el ordenar los 10,000,000 de números.

Gráfica tiempo de CPU

Debido a que el algoritmo es demasiado ineficiente no fue posible el ordenar los 10,000,000 de números.

Bubble sort optimizado

Tabla

Tamaño (N)	Temp Proc	Tiempo Real	E/S
10,000,000	Indefinido	Indefinido	indefinido

Gráfica tiempo real

Debido a que el algoritmo es demasiado ineficiente no fue posible el ordenar los 10,000,000 de números.

Gráfica tiempo de CPU

Debido a que el algoritmo es demasiado ineficiente no fue posible el ordenar los 10,000,000 de números.

Insertion sort

Tabla

Tamaño (N)	Temp Proc	Tiempo Real	E/S
10,000,000	Indefinido	Indefinido	indefinido

Gráfica tiempo real

Debido a que el algoritmo es demasiado ineficiente no fue posible el ordenar los 10,000,000 de números.

Gráfica tiempo de CPU

Debido a que el algoritmo es demasiado ineficiente no fue posible el ordenar los 10,000,000 de números.

Selection sort

Tabla

Tamaño (N)	Temp Proc	Tiempo Real	E/S
10,000,000	Indefinido	Indefinido	indefinido

Gráfica tiempo real

Debido a que el algoritmo es demasiado ineficiente no fue posible el ordenar los 10,000,000 de números.

Gráfica tiempo de CPU

Debido a que el algoritmo es demasiado ineficiente no fue posible el ordenar los 10,000,000 de números.

Shell sort

Tabla

Tamaño (N)	Temp Proc	Temp Real	Tiempo E/S
10000000	37.841173	37.969924	0.000016

Gráfica tiempo real

Gráfica tiempo de CPU

Tree sort

Tabla

Tamaño (N)	Temp Proc	Tiempo real	Tiempo E/S
10000000	36.360991	36.8515971	0.44309

Gráfica tiempo real

Gráfica tiempo de CPU

Análisis temporal

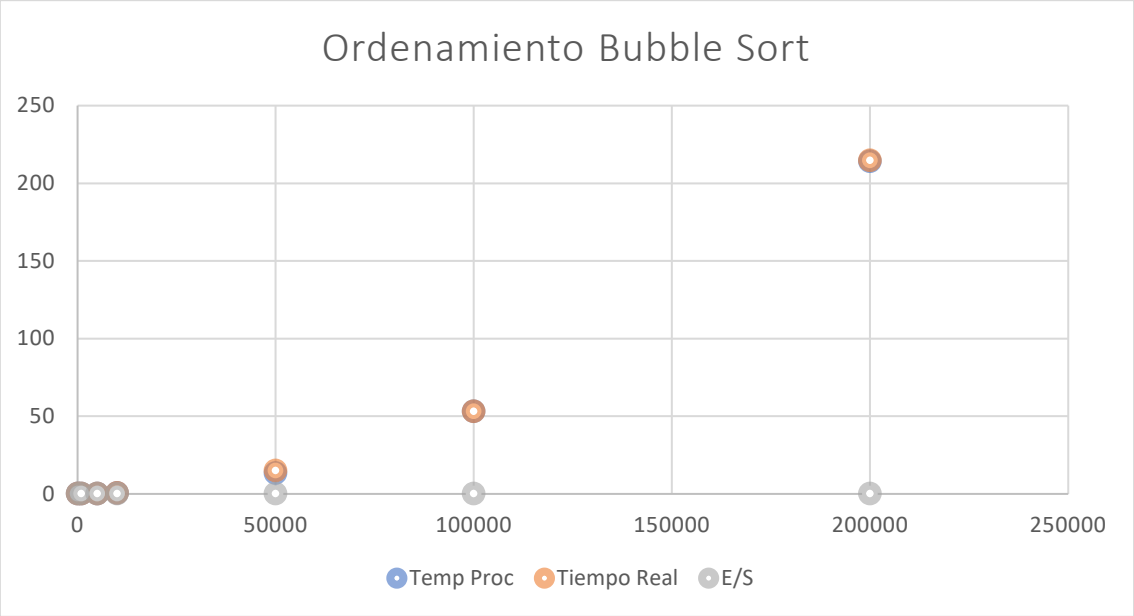
Para esta sección se han creado diferentes gráficas

Bubble sort

Tabla

Tamaño (N)	Temp Proc	Tiempo Real	E/S
100	0.000058	5.1022E-05	0
1000	0.004213	0.00423789	0
5000	0.11677	0.1173799	0
10000	0.487121	0.49061298	0
50000	13.212454	15.0038271	0.025718
100000	52.982683	53.144228	0.003284
200000	214.045297	214.644705	0.000013

Gráfica

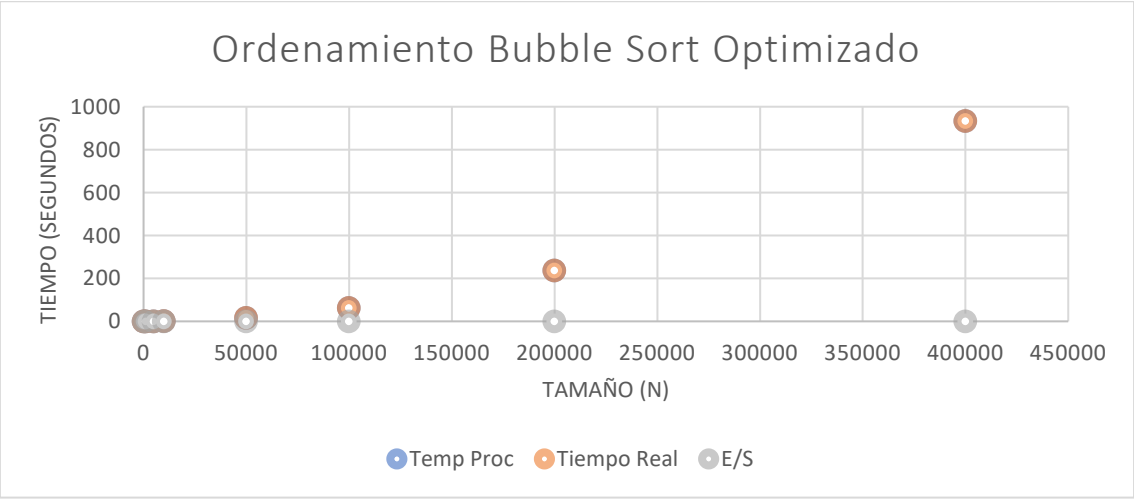


Bubble sort optimizado

Tabla

Tamaño (N)	Temp Proc	Tiempo Real	E/S
100	0.000088	8.1062E-05	0
1000	0.005771	0.00615907	0.000389
5000	0.14997	0.15026402	0
10000	0.560914	0.56199217	0
50000	15.718181	15.745755	0
100000	62.647085	62.6782699	0.010081
200000	237.039776	237.216744	0.050073
400000	932.845821	933.207164	0.090039

Gráfica

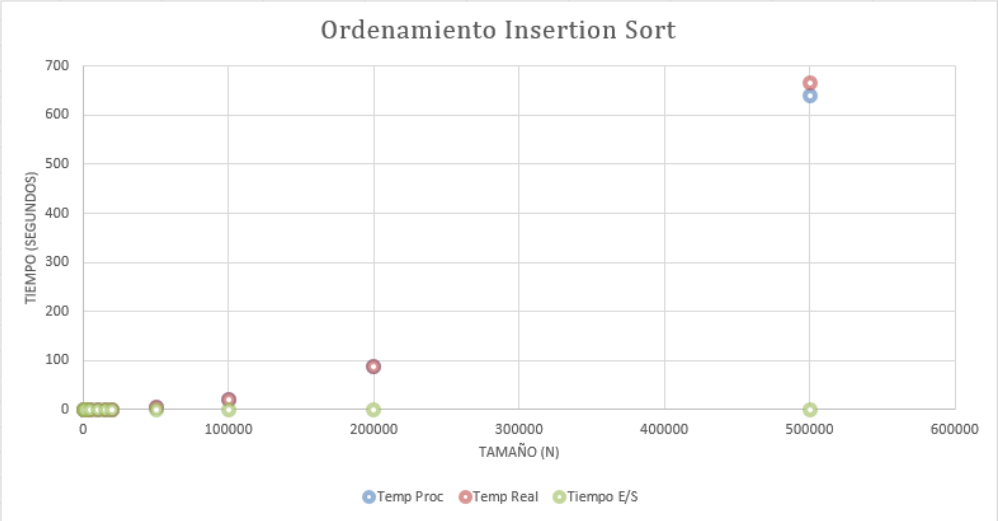


Insertion sort

Tabla

Tamaño (N)	Temp Proc	Tiempo Real	E/S
100	0.0001080000	0.0000998974	0
500	0.0004900000	0.0004830360	0
1000	0.0013240000	0.0052568913	0
5000	0.0207630000	0.0396978855	0
10000	0.0985130000	0.1986839771	0
15000	0.1769760000	0.3418521881	0
20000	0.6058190000	0.6857118607	0
50000	5.4364810000	5.6286108494	0.0039510000
100000	20.4139270000	20.7199790478	0.0021170000
200000	87.5077740000	88.6801528931	0.0100710000
500000	639.3639300000	664.8530659676	0.4229730000

Gráfica

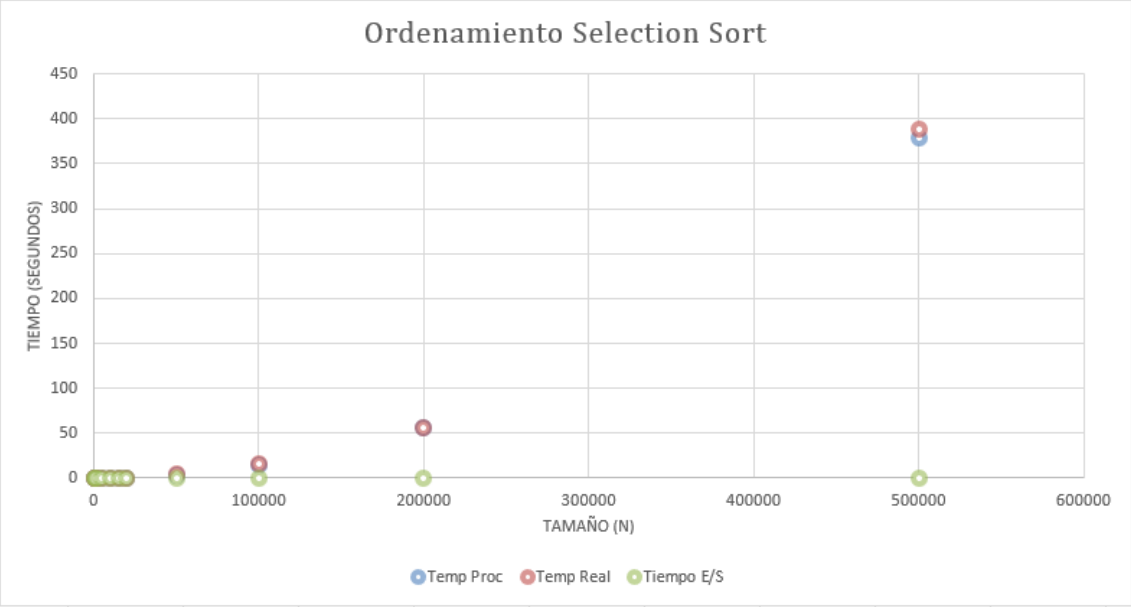


Selection sort

Tabla

Tamaño (N)	Temp Proc	Tiempo Real	E/S
100	0.0001290000	0.0001220703	0
500	0.0007360000	0.0007281303	0
1000	0.0022720000	0.0022649765	0
5000	0.0328480000	0.0642380714	0
10000	0.1194360000	0.2425880432	0
15000	0.2609050000	0.4560501575	0
20000	0.4800550000	0.5315189362	0
50000	3.6687930000	3.9456200600	0
100000	14.8299070000	15.4145851135	0.0206390000
200000	55.3970990000	55.9712560177	0
500000	378.8869640000	388.1334750652	0.0992920000

Gráfica

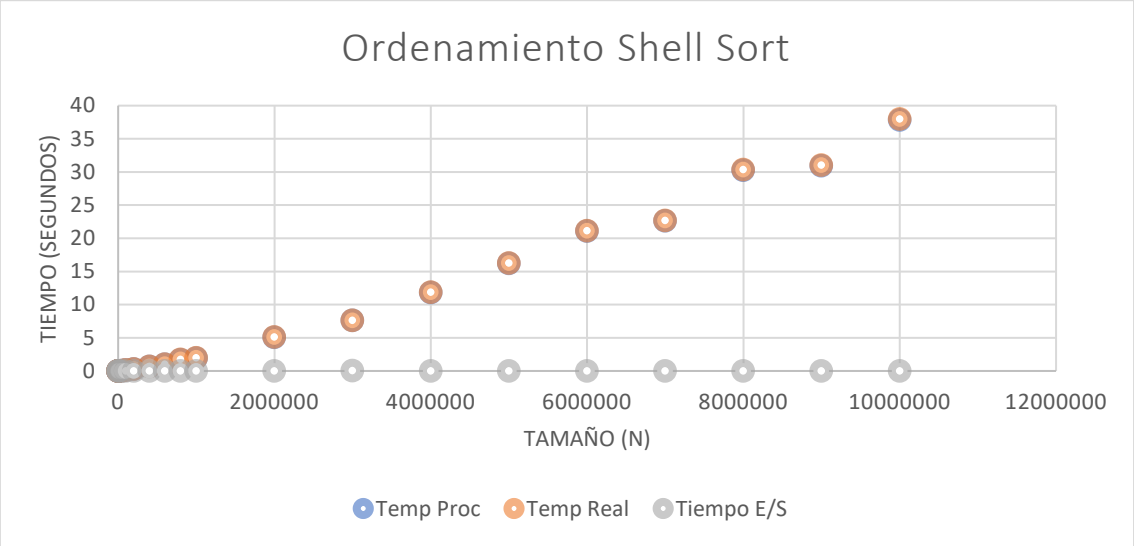


Shell sort

Tabla

Tamaño (N)	Temp Proc	Temp Real	Tiempo E/S
100	0	2.0027E-05	0
1000	0.000318	0.00030994	0.00032
5000	0.002664	0.00267506	0.001267
10000	0.006518	0.00658298	0
50000	0.048304	0.05091095	0
100000	0.127345	0.12753892	0
200000	0.283924	0.28462982	0.003282
400000	0.681707	0.68290997	0.005986
600000	1.016061	1.01894498	0.000151
800000	1.700886	1.71369314	0.003121
1000000	1.948163	1.96243715	0.003388
2000000	5.084782	5.10144901	0.00601
3000000	7.610406	7.63501	0.021515
4000000	11.824409	11.8690839	0.000082
5000000	16.192861	16.2482481	0
6000000	21.080136	21.1563051	0.000263
7000000	22.614992	22.6865108	0.000102
8000000	30.238467	30.335557	0.003391
9000000	30.932332	31.033217	0.000019
10000000	37.841173	37.969924	0.000016

Gráfica

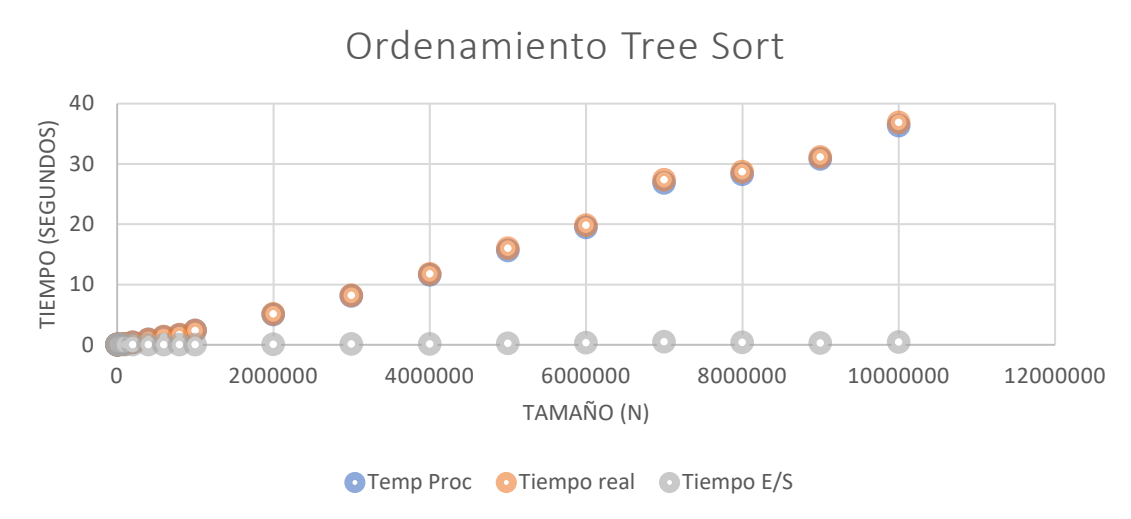


Tree sort

Tabla

Tamaño (N)	Temp Proc	Tiempo real	Tiempo E/S
100	0.000128	0.00012088	0
1000	0.000981	0.00097489	0
5000	0.005671	0.00567102	0
10000	0.011523	0.01151705	0
50000	0.079781	0.08059287	0.000659
100000	0.135554	0.13696003	0.001412
200000	0.407793	0.41948414	0.011205
400000	0.86857	0.89133811	0.02136
600000	1.31591	1.35039902	0.033295
800000	1.638582	1.64695907	0.007694
1000000	2.339014	2.34992003	0.009991
2000000	5.03624	5.10820818	0.070072
3000000	8.078074	8.20745492	0.126725
4000000	11.580127	11.7547228	0.14382
5000000	15.652728	15.985816	0.243307
6000000	19.474537	19.828558	0.329773
7000000	26.833486	27.3749421	0.479802
8000000	28.273371	28.7141092	0.393087
9000000	30.815384	31.1632862	0.316612
10000000	36.360991	36.8515971	0.44309

Gráfica



Comparativa

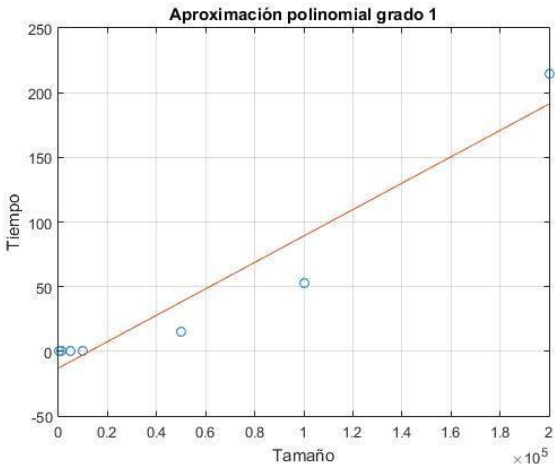
En esta sección se ha graficado una comparativa de todos los algoritmos de ordenamiento que se han implementado. La comparativa se ha realizado sobre el tiempo real

Aproximación polinomial

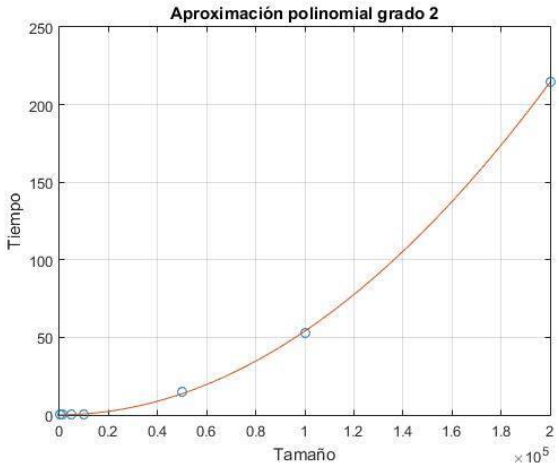
Para esta sección se han realizado aproximaciones polinomiales del comportamiento temporal en tiempo real de los algoritmos que fueron probados en el punto anterior. Cada uno de los algoritmos se han aproximado con polinomios de grado 1, 2, 3, 4 y 8

Bubble sort

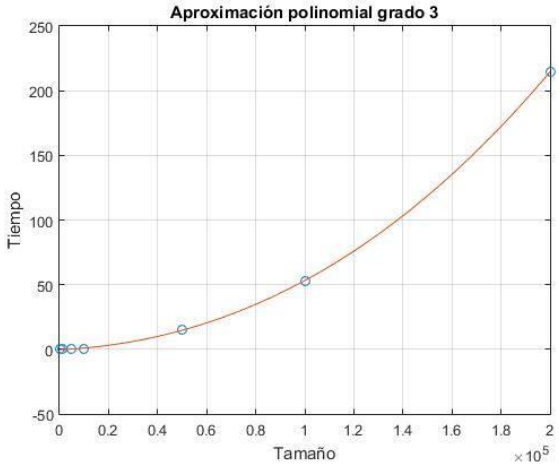
Grado 1:



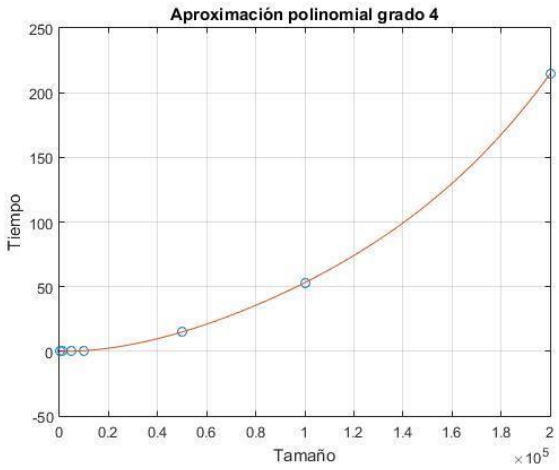
Grado 2:



Grado 3:

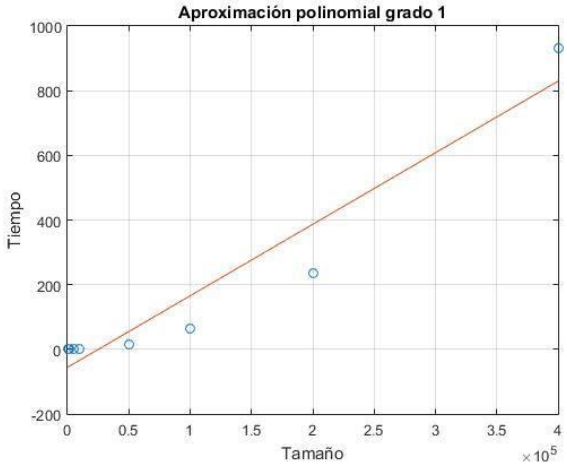


Grado 4:

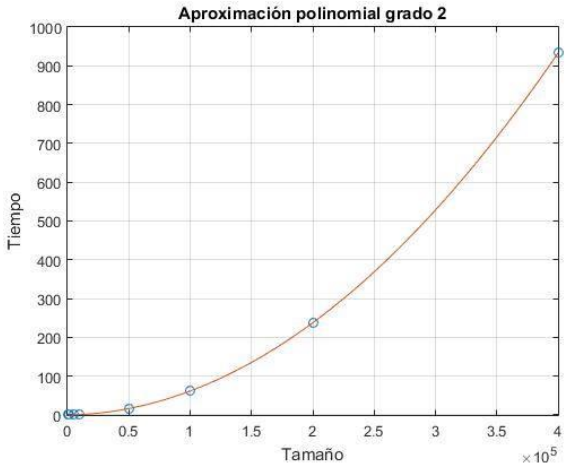


Bubble sort optimizado

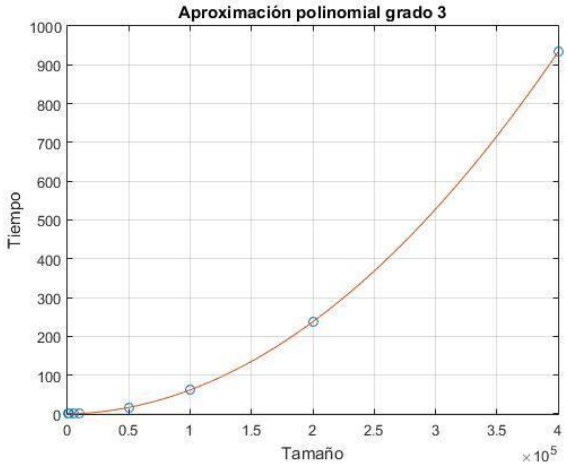
Grado 1:



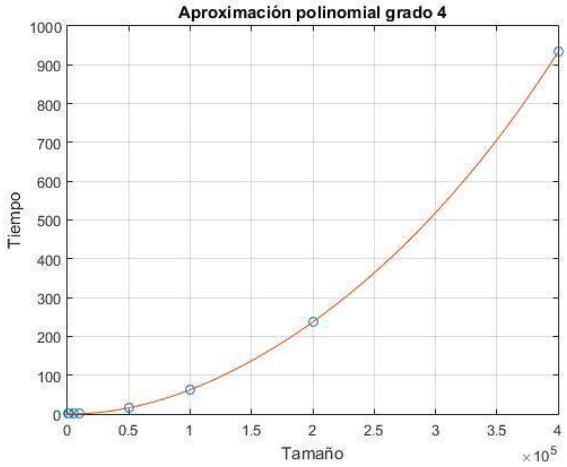
Grado 2:



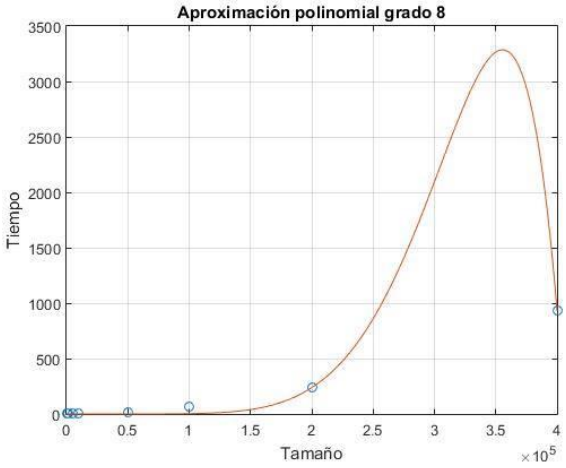
Grado 3:



Grado 4:

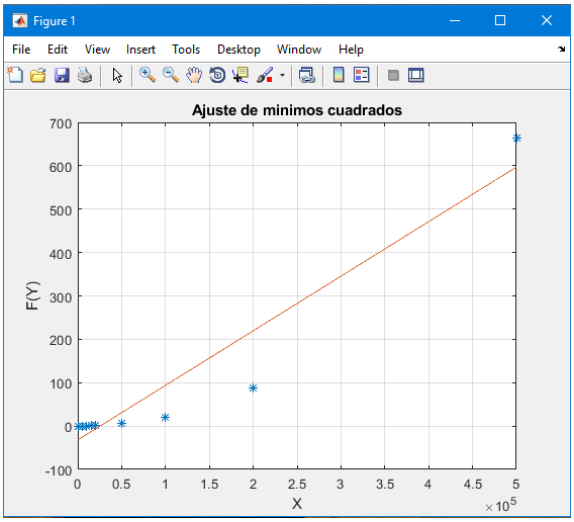


Grado 8:



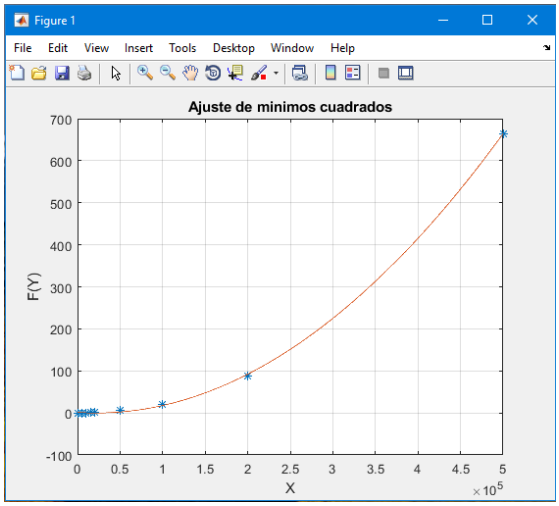
Insertion sort

Grado 1:



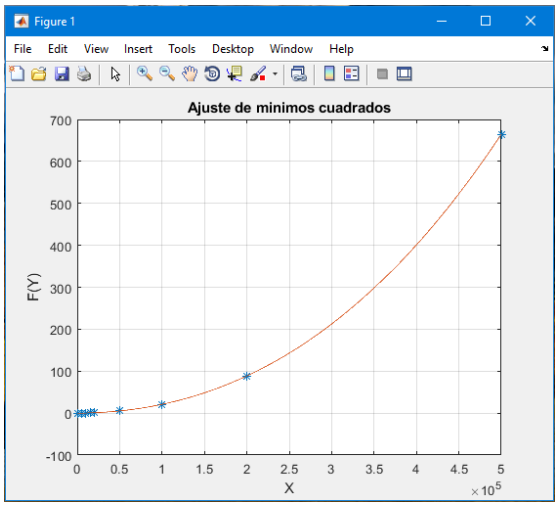
Como se observa en la grafica no conviene utilizar un polinomio de grado 1 ya que no se acerca a los puntos y por lo tanto su aproximación es mala.

Grado 2:



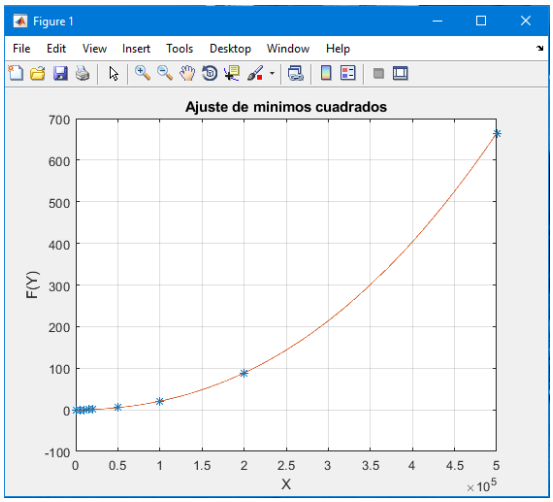
Como se observa en la grafica si conviene utilizar un polinomio de grado 2 ya que se aproxima a los puntos mostrados por lo tanto tiene una buena aproximación.

Grado 3:



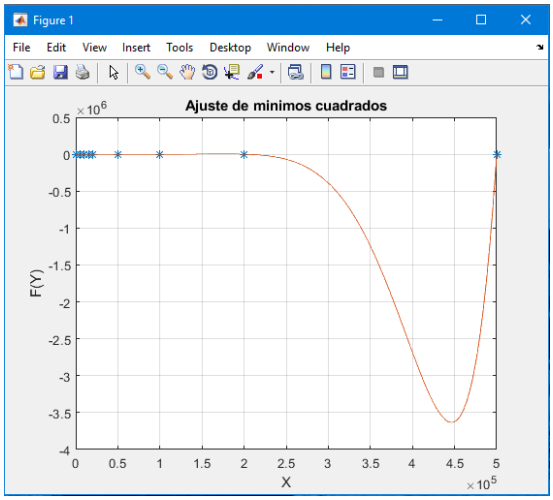
En esta grafica se ve que conviene aun más utilizar un polinomio de grado 3 ya que tiene una mayor precisión a comparación del de grado 2 por lo tanto tiene una buena aproximación.

Grado 4:



En esta grafica el polinomio de grado 4 tiene un parecido con el de grado 3 e igualmente es mas preciso en el acercamiento con los puntos por lo tanto tiene una buena aproximación.

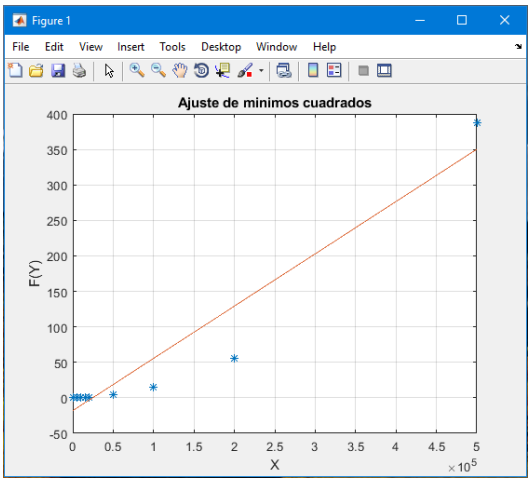
Grado 8:



Cuando el polinomio es de grado 8 se torna de una forma diferente a comparación con las demas, pero aun toca los puntos.

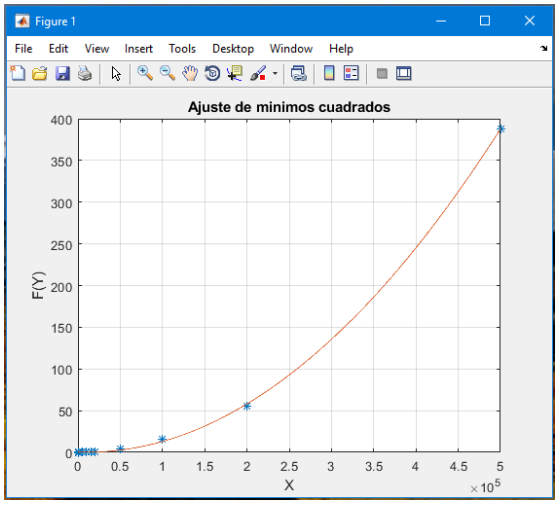
Selection sort

Grado 1:



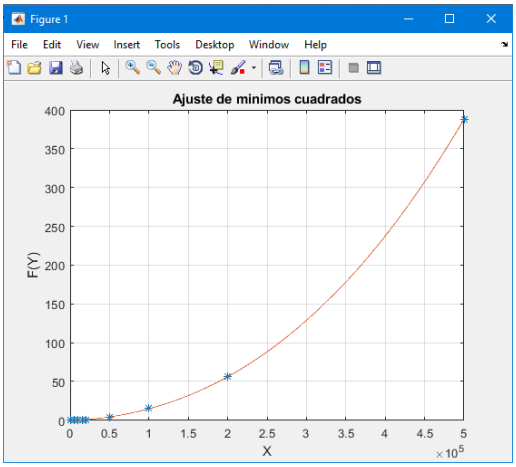
El polinomio de grado 1 no es buena elección para este algoritmo ya que su aproximación es muy mala.

Grado 2:



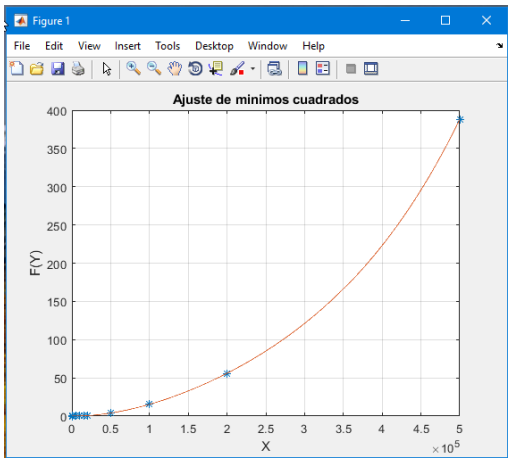
Si se utiliza un polinomio de grado 2 la aproximación es buena pero no es muy precisa.

Grado 3:



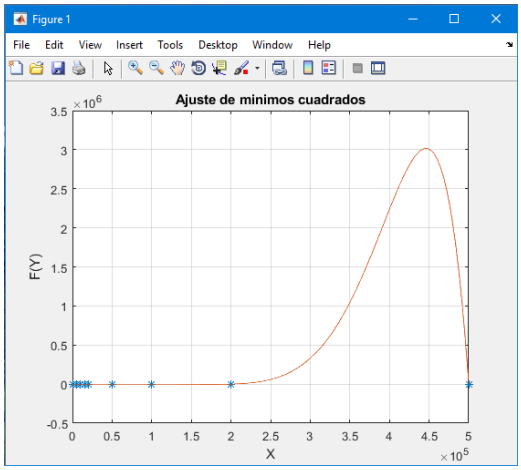
Si se utiliza un algoritmo de grado 3 la aproximación es buena a comparación del grado 2 pero aún le falta precisión.

Grado 4:



Para un polinomio de grado 4 la aproximación es buena y mas precisa que un polinomio de grado 2 y 3.

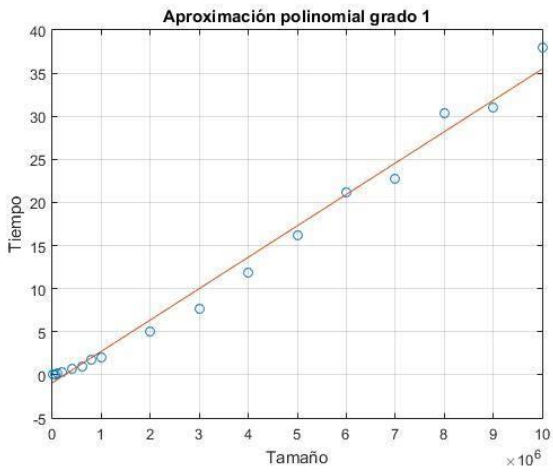
Grado 8:



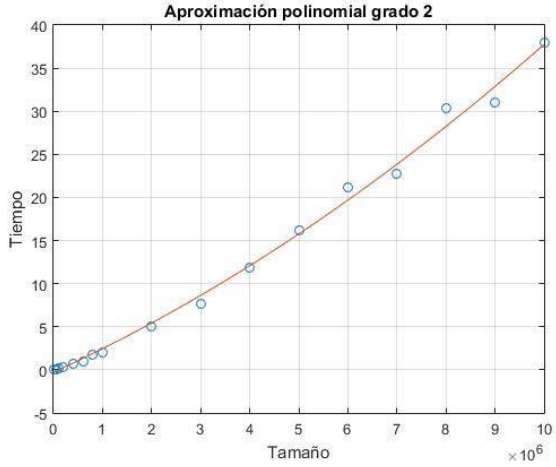
Para un polinomio de grado 8 la aproximación con los puntos es buena, pero es muy diferente a los polinomios anteriores.

Shell sort

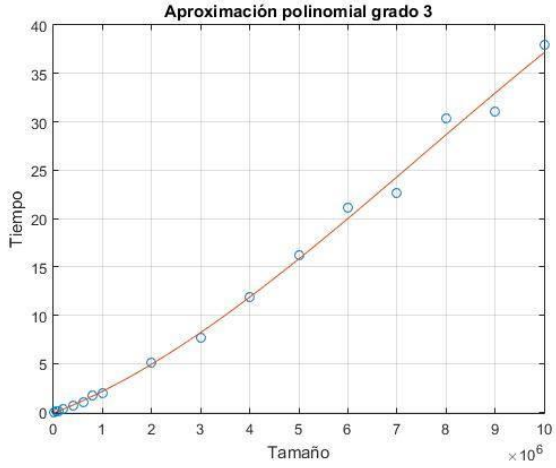
Grado 1:



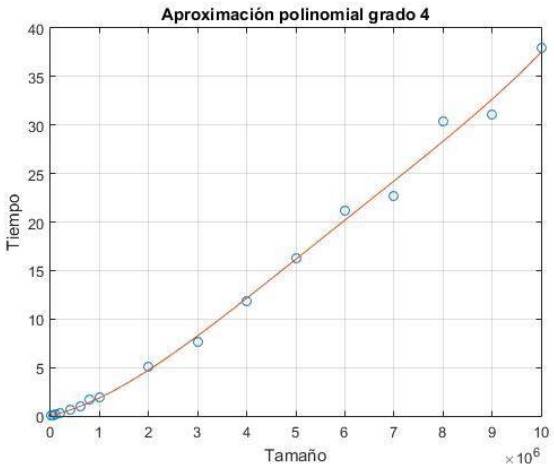
Grado 2:



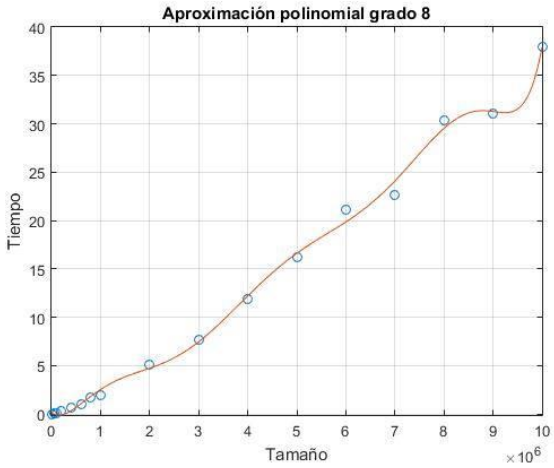
Grado 3:



Grado 4:

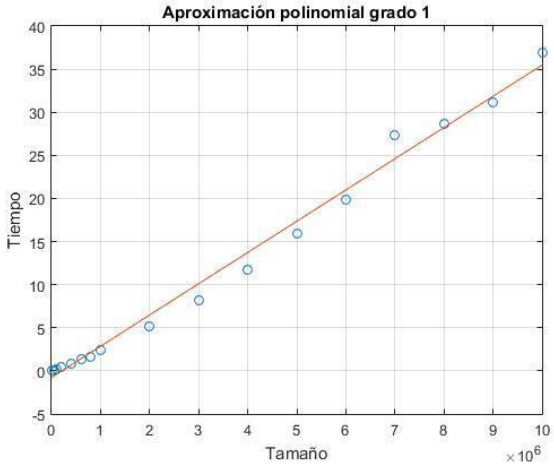


Grado 8:

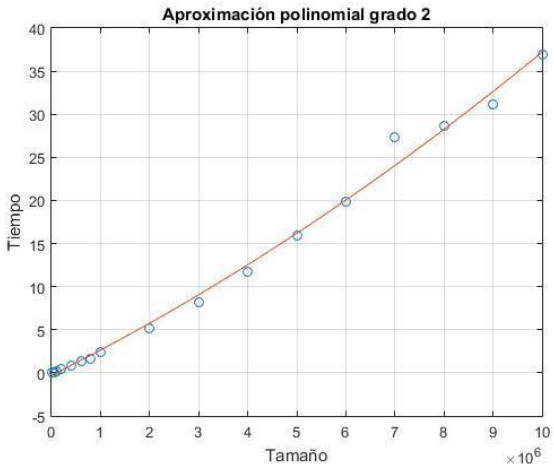


Tree sort

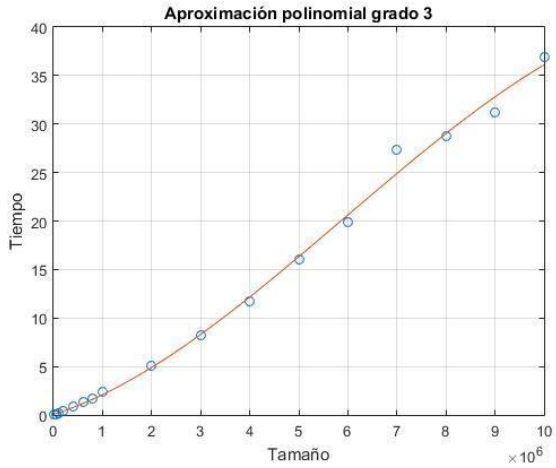
Grado 1:



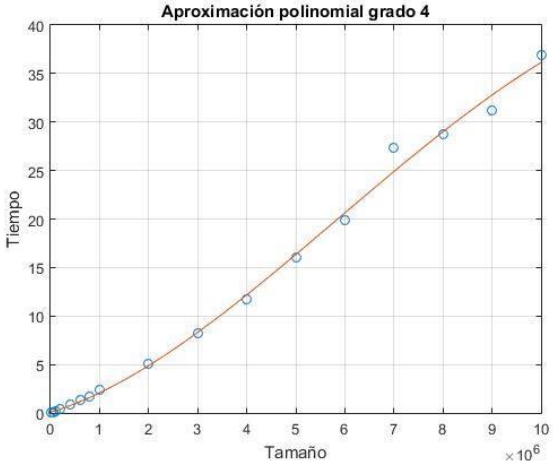
Grado 2:



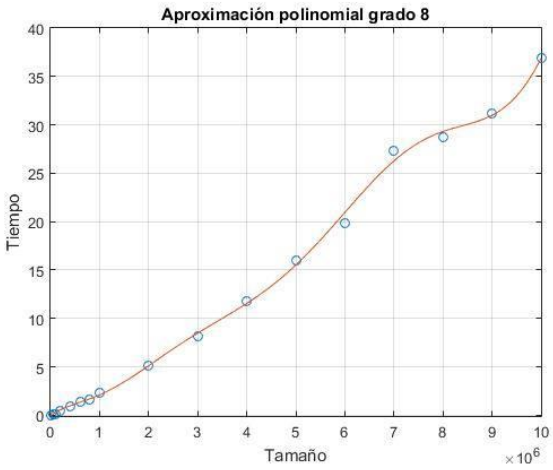
Grado 3:



Grado 4:



Grado 8:



Preguntas

1. **¿Cuál de los 5 algoritmos es más fácil de implementar?**
   Es bastante evidente que el algoritmo de Bubble sort es el más sencillo ya que es bastante fácil de deducir.

2. **¿Cuál de los 5 algoritmos es el más difícil de implementar?**
   Quizá Tree sort, más que nada porque se deben retomar temas como apuntadores para crear nuevamente el árbol binario.

3. **¿Cuál algoritmo tiene menor complejidad temporal?**
   Tree sort

4. **¿Cuál algoritmo tiene mayor complejidad temporal?**

   Bubble sort

5. **¿Cuál algoritmo tiene menor complejidad espacial? ¿Por qué?**
   La tienen 2 algoritmos que son burbuja e inserción, ya que ambos solo utilizan un arreglo donde se encuentran los datos y una variable auxiliar para que estos mismos se desplacen en el arreglo.

6. **¿Cuál algoritmo tiene mayor complejidad espacial? ¿Por qué?**
   El algoritmo de árbol, ya que por cada dato insertado en el se debe crear un nodo, el cual es un struct con 1 entero y 2 apuntadores a nodos

7. **¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué?**
   Si, porque cada algoritmo controla su informacion de distintas maneras por lo tanto los resultados fueron muy variados, pero se logró comprobar la eficiencia de los algoritmos.

8. **¿Sus resultados experimentales difieren mucho de los del resto de los equipos? ¿A qué se debe?**
   Si, se debe a los distintos recursos que tienen las maquinas, entre mayor procesamiento tenga mas rapido se ejecuta el algoritmo.

9. **¿Existió un entorno controlado para realizar las pruebas experimentales? ¿Cuál fue**?
   Si, ya que se finalizaron todos los procesos externos y solo quedaron activos los procesos de sistema.

10. **¿Qué recomendaciones darían a nuevos equipos para realizar esta práctica?**
    Primero que nada, se recomienda el previo conocimiento de apuntadores ya que se utilizan para el algoritmo de árbol, además de tener conocimientos de estructuras.

Anexos

Código bubble sort

```
#include <iostream>
#include "tiempo.h"

using namespace std;

int main (int argc, char *argv[]){

      //-------------------------------------------------Variables---
-------------------------------------------------//

     int N= atoi(argv[1]);
      int k=0; //Dato detectado por la entrada estandar
      int i=0, j=0; //Contadores
      int aux; //Auxiliar para almacenar el numero
      int *vector = new int [N]; //Vector contenedor de los numeros a
ordenar
    double utime0, stime0, wtime0,utime1, stime1, wtime1;
//Variables para medición de tiempos

      //-------------------------------------------------------------
-------------------------------------------------//


    //--------------------------- Se ingresan los valores al arreglo
desde el archivo de texto---------------------------//

      while(cin>>k && i<N){ //Se leera un numero hasta que se
encuentre el fin del archivo
            vector[i]= k;
            i++;
      }

      //-------------------------------------------------------------
-------------------------------------------------//

    //-------------------------------------------------Algoritmo
principal-----------------------------------------//

      uswtime(&utime0, &stime0, &wtime0); // Se inicia el conteo para
contar el algoritmo

      for (i=0; i<N; i++){
            for(j=0; j<N-1; j++){
                  if(vector[j]>vector[j+1]){
                        aux= vector[j];
                        vector[j]= vector[j+1];
                        vector[j+1]=aux;
                  }
            }
      }

      uswtime(&utime1, &stime1, &wtime1); // Termina de contarse el
tiempo de procesamiento, de E/S y el uso de CPU

      /* NOTA: El problema con este algoritmo de ordenacion es que
aunque el arreglo no se encuentre tan desordenado
    el for menos anidado se ejecutara n veces lo que hace perder
eficiencia, para verificar si el arreglo sigue
```

```
    desordenado se utiliza una bandera y asi optimizar el algoritmo,
por esta razon se utiliza Bubble Sort Optimizado */

      //-------------------------------------------------------------
-------------------------------------------------//

      //-------------------------------Imprimiendo el resultado
dentro del archivo de texto-----------------------------//

      /*for (i=0; i<N; i++){
            cout << vector[i] << endl;
      }*/

      //-------------------------------------------------------------
-------------------------------------------------//


    //-----------------------------------Cálculo del tiempo de
ejecución del programa--------------------------------//

    cout<<N <<", "<< (wtime1 - wtime0) <<endl;

      /*printf("\n");
      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",
utime1 - utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",
utime1 - utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");*/

      //-------------------------------------------------------------
-------------------------------------------------//

    return 0;
}
```

Código bubble sort optimizado

```
#include <iostream>
#include "tiempo.h"

using namespace std;

//#define N 10000000 //Constante que determina la longitud del
arreglo

int main (int argc, char *argv[]){
```

```
    //-----------------------------------------------Variables-------
----------------------------------------//

    int N= atoi(argv[1]), size = 0;
    size = N;
      int *array= new int [N];
      int k=0; //k es el dato entrante que viene del archivo de texto
      int i=0, j=0; // Contadores
      bool flag= true; //Esta bandera indicara si el arreglo aun se
encuentra deordenado
      int aux=0; //Variable auxiliar que nos servira para guardar
temporalmente los valores del arreglo
      double utime0, stime0, wtime0,utime1, stime1, wtime1;
//Variables para medición de tiempos

      //-----------------------------------------------------------------
----------------------------------------//


      //----------------------- Se ingresan los valores al arreglo
desde el archivo de texto-----------------------//

      while(cin>>k && N--){ //Mientras cin encuentre valores en el
archivo de texto
            array[i]= k;
            i++;
      }

    //-----------------------------------------------------------------
----------------------------------------//

    //-----------------------------------------Algoritmo
principal---------------------------------------//

      i=0;

    uswtime(&utime0, &stime0, &wtime0); // Se inicia el conteo para
contar el algoritmo

      while(i<size && flag==true){
            flag= false;
            for (j=0;  j<size-1;  j++){
                  if(array[j] > array[j+1]){
                        aux= array[j];
                        array[j]= array[j+1];
                        array[j+1]= aux;
                        flag= true;
                  }
            }
            i++;
      }

      uswtime(&utime1, &stime1, &wtime1); // Termina de contarse el
tiempo de procesamiento, de E/S y el uso de CPU

      /* NOTA: Se emplea el mismo algoritmo que Bubble Sort  a
diferencia de la bandera, esta bandera nos ayuda
      para evitar que siga realizando comparaciones cuando el
arreglo ya se encuentre totalmente ordenado,
```

```
      a diferencia de Bubble Sort que sigue realizando  n veces
comparaciones aunque el arreglo  ya se encuentre
     totalmente ordenado. Bubble Sort optimizado nos ahorrara muchas
operaciones y asi ahorrar una cantidad
     significativa de tiempo y recursos */

      //-----------------------------------------------------------------
-----------------------------------------//


      //----------------------------Imprimiendo el resultado dentro
del archivo de texto-------------------------//
      /**
      for(i=0; i<size; i++){
            cout << array[i] << endl;
      }**/

    //-----------------------------------------------------------------
-----------------------------------------//


     //-------------------------------Cálculo del tiempo de
ejecución del programa----------------------------//

     cout<<"------------------------------------ Para "<<size<<"
numeros -------------------------------------"<<endl;
      printf("\n");
      printf("real  (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user  (Tiempo de procesamiento en CPU) %.10f s\n",
utime1 - utime0);
      printf("sys  (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real  (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user  (Tiempo de procesamiento en CPU) %.10e s\n",
utime1 - utime0);
      printf("sys  (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");

      //-----------------------------------------------------------------
-----------------------------------------//

      return 0;
}
```

Código insertion sort

```
//*****************************************************************
**********
//Josue Macias Castillo, Eduardo Torres Hernandez, Erick Efrain
Vargas Romero
//Curso: Análisis de algoritmos
//(C) Septiembre 2018
```

```
//ESCOM-IPN
//Algoritmo de ordenamiento por insercion
//Compilación: "g++ Insercion.cpp -o insert"
//Ejecución: ./insert (Linux y MAC OS), insert.exe (Windows)
//*****************************************************************
**********
//*****************************************************************
**********
//LIBRERIAS
//*****************************************************************
**********
#include<stdio.h>
#include "tiempo.h"
#include<iostream>
using namespace std;
//*****************************************************************
**********
//DEFINICIONES
//*****************************************************************
**********
//#define N 10000000
//*****************************************************************
**********
//PROGRAMA PRINCIPAL
//*****************************************************************
**********
int main(int argc, char* argv[])
{

//*****************************************************************

      //Variables del main
      //*************************************************************
*****
      double utime0, stime0, wtime0,utime1, stime1, wtime1;
//Variables para medición de tiempos
      int n;      //n determina el tamaño del algorito dado por
argumento al ejecutar
      int i; //Variables para loops

      //*************************************************************
*****
      //Recepción y decodificación de argumentos
      //*************************************************************
*****

      //Si no se introducen exactamente 2 argumentos (Cadena de
ejecución y cadena=n)
      if (argc!=2)
      {
            printf("\nIndique el tamanio del algoritmo - Ejemplo:
[user@equipo]$ %s 100\n",argv[0]);
            exit(1);
      }
      //Tomar el segundo argumento como tamaño del algoritmo
      else
      {
            n=atoi(argv[1]);
      }
```

```
      //*****************************************************************
*****
      //Iniciar el conteo del tiempo para las evaluaciones de
rendimiento
      //*****************************************************************
*****
      uswtime(&utime0, &stime0, &wtime0);
      //*****************************************************************
*****


//*****************************************************************
******
    //DECLARACION DE VARIABLES

//*****************************************************************
******
    int k = 0;
    int pos, temp,m;
    int j = 0;
    int *numeros = new int [n];

//*****************************************************************
******
    //CICLO PARA RECEPCION DE DATOS

//*****************************************************************
******
    while(cin >> m && n--)
    {
        numeros[k] = m;
        k++;
    }

//*****************************************************************
******
    //ALGORITMO DE ORDENAMIENTO

//*****************************************************************
******
    for(k = 0; k < n; k++)
    {
        pos = k;
        temp = numeros[k];
        while(pos > 0 && temp < numeros[pos - 1] )
        {
            numeros[pos] = numeros[pos - 1];
            pos--;
        }
        numeros[pos] = temp;
    }

//*****************************************************************
*******
    //IMPRESION DE NUMEROS ORDENADOS

//*****************************************************************
*******
    /*for(j = 0; j < N; j++)
        cout << numeros[j] << " ";*/
```

```
//*****************************************************************

      //Evaluar los tiempos de ejecución
      //*************************************************************
*****
      uswtime(&utime1, &stime1, &wtime1);

    cout<<"Para "<<n<<" números"<<endl;
      //Cálculo del tiempo de ejecución del programa
      printf("\n");
      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",
utime1 - utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",
utime1 - utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");
      //*************************************************************
*****

    //Terminar programa normalmente
    return 0;
}
```

Código selection sort

```
//*****************************************************************
**********
//Josue Macias Castillo, Eduardo Torres Hernandez, Erick Efrain
Vargas Romero
//Curso: Análisis de algoritmos
//(C) Septiembre 2018
//ESCOM-IPN
//Algoritmo de ordenamiento por seleccion
//Compilación: "g++ Seleccion.cpp -o select"
//Ejecución: ./select (Linux y MAC OS), select.exe (Windows)
//*****************************************************************
*********
//*****************************************************************
*********
//LIBRERIAS
//*****************************************************************
*********
#include<stdio.h>
#include "tiempo.h"
#include<iostream>
using namespace std;
```

```
//*****************************************************************
**********
//DEFINICIONES
//*****************************************************************
**********
//#define N 10000000
//*****************************************************************
**********
//PROGRAMA PRINCIPAL
//*****************************************************************
**********
int main(int argc, char *argv[])
{

//*****************************************************************

      //Variables del main
      //*************************************************************
*****
      double utime0, stime0, wtime0,utime1, stime1, wtime1;
//Variables para medición de tiempos
      int n;      //n determina el tamaño del algorito dado por
argumento al ejecutar
      int i; //Variables para loops

      //*************************************************************
*****
      //Recepción y decodificación de argumentos
      //*************************************************************
*****

      //Si no se introducen exactamente 2 argumentos (Cadena de
ejecución y cadena=n)
      if (argc!=2)
      {
            printf("\nIndique el tamanio del algoritmo - Ejemplo:
[user@equipo]$ %s 100\n",argv[0]);
            exit(1);
      }
      //Tomar el segundo argumento como tamaño del algoritmo
      else
      {
            n=atoi(argv[1]);
      }

      //*************************************************************
*****
      //Iniciar el conteo del tiempo para las evaluaciones de
rendimiento
      //*************************************************************
*****
      uswtime(&utime0, &stime0, &wtime0);
      //*************************************************************
*****


//*****************************************************************
******
    //DECLARACION DE VARIABLES
```

```
//*****************************************************************
******
    int k = 0, j = 0;
    int aux,min,m;
    int *numeros = new int [n];

//*****************************************************************
******
    //CICLO PARA RECEPCION DE DATOS

//*****************************************************************
******
    while(cin >> m && n--)
    {
        numeros[k] = m;
        k++;
    }

//*****************************************************************
******
    //ALGORITMO DE ORDENAMIENTO

//*****************************************************************
******
    for(k = 0; k < 5; k++)
    {
        min = k;
        for(j = k + 1; j < 5; j++)
        {
            if(numeros[j] < numeros[min])
            {
               min = j;
            }
        }
        aux = numeros[k];
        numeros[k] = numeros[min];
        numeros[min] = aux;
    }

//*****************************************************************
*******
    //IMPRESION DE NUMEROS ORDENADOS

//*****************************************************************
*******
   /* for(k = 0; k < 5; k++)
        cout << numeros[k] << " ";*/


//*****************************************************************

      //Evaluar los tiempos de ejecución
      //*************************************************************
*****
      uswtime(&utime1, &stime1, &wtime1);
      cout<<"Para "<<n<<" numeros"<<endl;
      //Cálculo del tiempo de ejecución del programa
      printf("\n");
      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
```

```
      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",
utime1 - utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",
utime1 - utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");
      //************************************************************
*****

    //Terminar programa normalmente
    return 0;
}
```

Código Shell sort

```
#include <iostream>
#include "tiempo.h"

using namespace std;

//#define N 11//Constante que determina la longitud del arreglo//

int main (int argc, char *argv[]){

    //-------------------------------------------------------
Variables-----------------------------------------------------------
//

    int N = atoi(argv[1]), size = 0;
      size = N;
      int k; //Dato detectado por la entrada estandar
      int i=0, j=0; //Contadores
      int div= N; //Este numero funciona para realizar las
divisiones;
      int aux; //Esta variable nos ayudara a guardar temporalmente
      bool flag; //Esta bandera nos dira si hubo mas cambios
      int *vector = new int [N]; //Vector contenedor de los numeros a
ordenar
      double utime0, stime0, wtime0,utime1, stime1, wtime1;
//Variables para medición de tiempos

      //-------------------------------------------------------------
-------------------------------------------------------------//

      //---------------------------------- Se ingresan los valores
al arreglo desde el archivo de texto-----------------------------//

      while(cin>>k && i<size){ //Se leera un numero hasta que se
encuentre el fin del archivo
```

```
            vector[i]= k;
            i++;
      }

      //-------------------------------------------------------------
------------------------------------------------------------//

      //-----------------------------------------------------
Algoritmo principal-----------------------------------------------
--//

      uswtime(&utime0, &stime0, &wtime0); // Se inicia el conteo para
contar el algoritmo

      while(div>1){     //Mientras la division sea mayor a 1 se
cumplira
            div= div/2; //Cada iteracion dividira entre 2
            flag= true; //La bandera indica que si ha habido una
division entre 2
            while(flag==true){      //La bandera indica que hay
ordenamientos por realizar
                  flag= false;      //Inmediatamente se desactiva la
bandera para posteriormente en caso de que hayan reemplazos volver
activarla
                  i=0;
                  while(i+div <N){  //Indicamos que no puede salirse
del tamanio del arreglo
                        if(vector[i]>vector[i+div]){  //Se realiza la
comparacion entre los elementos del arreglo
                              aux= vector[i];   // Se realiza el
almacen del dato que vamos a reemplazar en los siguientes dos pasos
                              vector[i]=vector[i+div];
                              vector[i+div]= aux;
                              flag= true; //Se indica a la bandera
que se ha encontrado una comparacion y se ha reemplazado algun valor
                        }
                        i++;        //Se incrementa el subindice del
arreglo
                  }
            }
      }

      uswtime(&utime1, &stime1, &wtime1); // Termina de contarse el
tiempo de procesamiento, de E/S y el uso de CPU

      //-------------------------------------------------------------
------------------------------------------------------------//


      //--------------------------------------Imprimiendo el
resultado dentro del archivo de texto-------------------------------
----//

      /*for(j=0;  j<N;  j++){
            cout << vector[j] <<endl;
      }*/

    //-------------------------------------------------------------
------------------------------------------------------------//
```

```
      //-----------------------------------------Cálculo del
tiempo de ejecución del programa-------------------------------------
//

    cout<<N <<", "<< (utime1 - utime0) <<endl;

      printf("\n");
      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",
utime1 - utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",
utime1 - utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");*/
      //-------------------------------------------------------------
-------------------------------------------------------------//

      return 0;
}
```

Código tree sort

TAD ÁRBOL BINARIO

```
/***
 * AUTORES:
 *                                Macías Castillo Josué
 *                                Torres Hernández Eduardo
 *                                Vargas Romero Erick Efraín
 * VERSION: 1.0
 *
 * DESCRIPCIÓN: Árbol binario
 * Esta estructura de datos es bastante simple de visualizar ya que
se tiene un nodo y al ir
 * añadiendo elementos se crean nuevos nodos los cuales van ligados
al anterior es decir
 * el nodo padre tiene nodos hijos. Para ese caso en particular por
ser un árbol binario cada
 * nodo padre a lo más podrá tener dos hijos.
 *
 * OBSERVACIONES: Esta estructura de datos es dinámica es decir
utilizará
 * la memoria que sea necesaria para tener un funcionamiento
adecuado.
 * */

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

```
//Estructura de nuestro elemento nódo
struct Node{
    /**
     * Variables
     * */
    //Esta variable contiene el dato que será almacenado en el nodo
   int value;

   //Aquí tenemos las referencias a los nodos hijo izquierdo y
derecho
   struct Node *left, *right;
};

//Renombramos los apuntadores a nodo como BinaryTree para mantener
un buen órden
typedef struct Node *BinaryTree;

/**
 * newTree: Esta función tiene como finalidad crear un nuevo
 * sub árbol los cuales bien sabemos que son en realidad
 * apuntadores a Nodo.
 * Recibe: int
 * Retorna: BinaryTree (*Nodo)
 * */
BinaryTree newTree(int data){
    //Creamos un nuevo nodo
    BinaryTree newBinaryTree = new Node();
    //Asignamos el valor al nodo
    newBinaryTree -> value = data;
    //Retornamos el nodo
    return newBinaryTree;
}

/**
 * insert: Esta función tiene como finalidad añadir un
 * sub árbol a el árbol que es pasado por referencia a la función
 * Recibe: BinaryTree e int
 * Devuelve: Nada
 * */
void insert(BinaryTree &sub_tree, int data){

    //Comparamos si sub árbol es nulo
    if(!sub_tree){
        //Si es un sub árbol nulo es creado
        sub_tree = newTree(data);
    }
    //Comparamos el valor de la raíz con el nuevo valor recibido
    else if(data >= sub_tree -> value){
        //SI el valor es mayor o igual se añade un nodo  a la
derecha de la raíz
        insert(sub_tree->right, data);
    } else {
        //Si es menor el nodo se añade a la izquierda de la raiz
        insert(sub_tree -> left, data);
    }
}

/**
 * inorder: Esta función tiene como finalidad hacer un recorrido en
```

```
 * profundidad del árbol, en este caso el recorrido se realiza en
in-orden
 * Es decir:
 * Primero se recorre el sub árbol izquierdo
 * Después se recorre la raíz
 * Finalmente se recorre el sub árbol derecho
 * */
void inorder(BinaryTree &sub_tree){
    if(sub_tree){
        inorder(sub_tree -> left);
        inorder(sub_tree->right);
    }
}

/**
 * postorder: Esta función tiene como finalidad hacer un recorrido
en
 * profundidad del árbol, en este caso el recorrido se realiza en
post-orden
 * Es decir:
 * Primero se recorre el sub árbol izquierdo
 * Después se recorre el sub árbol derecho
 * Finalmente  se recorre la raíz
 * */
void postorder(BinaryTree &sub_tree){
    if(sub_tree){
        inorder(sub_tree -> left);
        inorder(sub_tree->right);
        cout<<sub_tree->value<<endl;
    }
}

/**
 * preorder: Esta función tiene como finalidad hacer un recorrido en
 * profundidad del árbol, en este caso el recorrido se realiza en
pre-orden
 * Es decir:
 * Primero se recorre la raíz
 * Después se recorre el sub árbol izquierdo
 * Finalmente se recorre el sub árbol derecho
 * */
void preorder(BinaryTree &sub_tree){
    if(sub_tree){
        cout<<sub_tree->value<<endl;
        inorder(sub_tree -> left);
        inorder(sub_tree->right);
    }
}
```

PROGRAMA PRINCIPAL

```
/***
 * AUTORES:
 *                              Macías Castillo Josué
 *                              Torres Hernández Eduardo
 *                              Vargas Romero Erick Efraín
 * VERSION: 1.0
 *
 * DESCRIPCIÓN: tree sort
 * este es un algoritmo de ordenamiento que hace uso de una
estructura de
```

```
 * datos. La estructuda de datos empleada es el árbol binario.
 * El algoritmo tree sort es un algoritmo que a diferencia del resto
implementados
 * es increíblemente eficiente, permitiendo realizar el ordenamiento
de
 * los números de la entrada en un tiempo bastante breve.
 *
 * OBSERVACIONES: La estructura de datos que se ha empleado o bien
 * el código de esta, se encuentra en el archivo binary_tree.cpp
 * */

//LIbrerías
#include <iostream>
//En esta librería se encuentra el TAD arbol binario
#include "binary_tree.cpp"
#include "tiempo.h"

using namespace std;

int main(int argc, char* argv[]){

    /**       VARIABLES       **/
    //Variables para modificacuón de tiempos
    double utime0, stime0, wtime0,utime1, stime1, wtime1;
    //Esta variable contiene el árbol binario que será utilizado
para
    //el uso de tree sort
    BinaryTree binaryTree = NULL;
    //Esta variable almacenará el número que será leído por entrada
    //estándar
    int n;
    //Esta variable contiene el número de números que serán leídos
    //como máximo
    int N = 0, size = 0;

      //Si no se introducen exactamente 2 argumentos (Cadena de
ejecución y cadena=n)
      if (argc!=2) {
            printf("\nIndique el tamanio del algoritmo - Ejemplo:
[user@equipo]$ %s 100\n",argv[0]);
            exit(1);
      }
      //Tomar el segundo argumento como tamaño del algoritmo
      else{
            N=atoi(argv[1]);
      }

      size = N;
      //*************************************************************
*****
      //Iniciar el conteo del tiempo para las evaluaciones de
rendimiento
      //*************************************************************
*****
      uswtime(&utime0, &stime0, &wtime0);


//*****************************************************************

      //Algoritmo
```

```
      //*****************************************************************
*****
        while(cin>>n && N--){
                insert(binaryTree, n);
        }
    inorder(binaryTree);


//*****************************************************************

      //*****************************************************************
*****
      //Evaluar los tiempos de ejecución
      //*****************************************************************
*****
      uswtime(&utime1, &stime1, &wtime1);
      cout<<"-------------------------- Para "<<size<<" Números -----
------------------------------"<<endl;
      //Cálculo del tiempo de ejecución del programa
      printf("\n");
      printf("real  (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user  (Tiempo de procesamiento en CPU) %.10f s\n",
utime1 - utime0);
      printf("sys  (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real  (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user  (Tiempo de procesamiento en CPU) %.10e s\n",
utime1 - utime0);
      printf("sys  (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
      printf("\n");
      //*****************************************************************
*****
}
```

Compilación

Para realizar la compilación se ha decidido crear algunos scripts (archivos con extensión .sh) La forma de ejecutar estos archivos es desde la terminal de Linux con el comando ./script.sh

La ejecución de estos archivos hará que cada algoritmo sea ejecutado varias veces según la cantidad de números que se desea que lea del archivo.

Bibliografía

[1] E. A. F. Martínez, «Web Personal de Edgardo Adrian Franco Martinez,» 2010. [En línea]. Available: http://www.eafranco.com/. [Último acceso: 10 septiembre 2018].