Instituto Politécnico Nacional

Escuela Superior de Cómputo

Pruebas a Posteriori (Algoritmos de ordenamiento)

Unidad de aprendizaje: Análisis de algoritmos

Grupo: 3CM2

Equipo Atenea:
Pérez García Atziri Loretto Estrada
Galilea América

M. en C.:
Edgardo Adrian Martínez

16 de septiembre de 2018

Índice

Ejercicio 2: Complejidad temporal y análisis de casos

Pérez, Loretto
3CM2

16 de septiembre de 2018

1. Planteamiento del problema

Con base en el archivo de entrada proporcionado que tiene 10,000,000 de números diferentes, ordenarlo bajo los siguientes métodos de ordenamiento y comprara experimentalmente las complejidades de estos.

- Burbuja (Bubble Sort).
  - Burbuja Simple
  - Burbuja Mejorada
- Inserción (Insertion Sort).
- Selección (Selection Sort).
- Shell (Shell Sort).
- Ordenamiento con árbol binario de búsqueda (Binary Search Tree)

2. Actividades

Programar en ANSI C, cada uno de los algoritmos de ordenamiento mencionados.Adaptar el programa para que sea capaz de recibir un parámetro 'n' que indica el número de enteros a ordenar a partir de un archivo con máximo 10,000,000 de números en desorden. Medir el tiempo que tarda cada algoritmo en ordenar el archivo completo (n=10,000,000) y compare los tiempos (Real y de CPU) de cada algoritmo gráficamente en una gráfica de barras (2 gráficas de barras). Nota: Auxiliarse de la librería de C proporcionada para medir tiempo de ejecución bajo Linux. Realizar un análisis temporal para cada algoritmo ordenado:

3.
   - Los primeros 100, 1000, 5000, 10000, 50000, 100000, 200000, 400000, 600000, 800000, 1000000, 2000000, 3000000, 4000000, 5000000, 6000000, 7000000, 8000000, 9000000 y 10000000.
   - Graficar el comportamiento temporal de cada algoritmo.

4. Graficar una comparativa de los 5 algoritmos de ordenamiento (Tiempo real).

5. Realizar una aproximación polinomial del comportamiento temporal (tiempo real), de cada uno de los algoritmos probados según el punto 4.
   Nota: Aproximar cada algoritmo con un polinomio de grado 1, 2, 3, 4 y 8.

6. Mostrar gráficamente la comparativa de las aproximaciones para cada algoritmo (5 graficas) y determinar de manera justificada cuál es la mejor aproximación para cada algoritmo.

7. Determine con base en las aproximaciones obtenidas, cuál será el tiempo real de cada algoritmo para intervalo de 50000000, 100000000, 500000000, 1000000000 y 500000000 de números a ordenar.

3. Implementación de los algoritmos

Cada uno de los programas tienen una función que realiza el algoritmo, mientras lo demás es la captura de datos y la medición del tiempo.

3.1. Burbuja Simple

```
/****************BURBUJA SIMPLE*************************
LORETTO ESTRADA GALILEA AMERICA
PEREZ GARCIA ATZIRI***/
/***********LIBRERIAS*******************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tiempo.h"

/*******************************Funcion Burbuja*****************************/
//Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la longitud
void Burbuja(int *A, int n){
   int aux;//Se declara variable auxiliar para hacer el cambio
   for(int i = 0; i<n-2; i++){//Se realiza n-1 veces el proceso para ordenar todos los numeros
      for(int j=0; j<n-2; j++){
         if(*(A+j) > *(A+j+1)){//Compracion si el primero nÃºmero es mayor al siguiente
            aux = *(A+j);//Se asigna el valor del numero mayor de ambos a aux
            *(A+j) = *(A+j+1);//Se hace el intercambio
            *(A+j+1) = aux;//Se establece el numero mayor en la poscion +1
         }
      }
   }
}
/*******************************Funcion PRINCIPAL*****************************/
//Toma como entrada la longitud del arreglo.
int main(int argc, char const *argv[]){
```

```
27    int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
28    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
       ↪  tiempos
29    int n;
30    //*************************************************************
31    //RecepciÃ³n y decodificaciÃ³n de argumentos
32    //*************************************************************
33
34    //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
35    if (argc!=2)
36    {
37       printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
       ↪  100\n",argv[0]);
38       exit(1);
39    }
40    //Tomar el segundo argumento como tamaÃ±o del algoritmo
41    else
42    {
43       n=atoi(argv[1]);
44    }
45
46    //*************************************************************
47    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
48    //*************************************************************
49    uswtime(&utime0, &stime0, &wtime0);
50    //*************************************************************
51
52    //*************************************************************
53    //Algoritmo
54    //*************************************************************
55    array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
       ↪  numeros a ordenar
56    for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
       ↪  espacio de memoria pedido anteriormente
57    scanf("%d", array+i);
58    }
59    Burbuja(array, n);
60       //*************************************************************
61    //Evaluar los tiempos de ejecuciÃ³n
62    //*************************************************************
63    uswtime(&utime1, &stime1, &wtime1);
64
65    //Calculo del tiempo de ejecucion del programa
66    printf("\n");
67    printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
68    printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
69    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
70    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
       ↪  (wtime1 - wtime0));
71    printf("\n");
```

```
72
73     //Mostrar los tiempos en formato exponecial
74     printf("\n");
75     printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
76     printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
77     printf("sys (Tiempo en acciones de E/S)  %.10e s\n",  stime1 - stime0);
78     printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
       ↪  (wtime1 - wtime0));
79     printf("\n");
80     //*****************************************************************
81
82     //Terminar programa normalmente
83     exit (0);
84
85 }
```

3.2. Burbuja Optimizada

```
1  /*****************BURBUJA OPTIMIZADA*************************
2  LORETTO ESTRADA GALILEA AMÉRICA
3  PEREZ GARCIA ATZIRI***/
4  /************LIBRERIAS*******************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include "tiempo.h"
9
10 /*********************************Funcion BurbujaOpt*****************************/
11 //Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la longitud
   ↪  de este.
12 void BurbujaOpt(int *A, int n){
13    int aux, i, cambios;//Se declaran variables auxiliares para hacer el cambio
14    cambios = 1;
15    i = 0;
16    while(i<n && cambios!=0){//Se realiza n-1 veces el proceso para ordenar todos los
      ↪  numeros
17       cambios= 0;
18       for(int j=0; j<n-2; j++){
19          if(*(A+i) < *(A+j)){//CompraciÃ³n si el primero nÃºmero es mayor al siguiente
20             aux = *(A+i);//Se asigna el valor del numero mayor de ambos a aux
21             *(A+i) = *(A+j);//Se hace el intercambio
22             *(A+j) = aux;//Se establece el numero mayor en la poscion +1
23             cambios = 1;
24          }
25       }
26       i++;
27    }
28 }
29
30 /*********************************Funcion PRINCIPAL*****************************/
```

```
31 //Toma como entrada la longitud del arreglo.
32 int main(int argc, char const *argv[]){
33    int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
34    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
       ↪  tiempos
35    int n;
36    //*****************************************************************
37    //RecepciÃ³n y decodificaciÃ³n de argumentos
38    //*****************************************************************
39
40    //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
41    if (argc!=2)
42    {
43       printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
       ↪  100\n",argv[0]);
44       exit(1);
45    }
46    //Tomar el segundo argumento como tamaÃ±o del algoritmo
47    else
48    {
49       n=atoi(argv[1]);
50    }
51
52    //*****************************************************************
53    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
54    //*****************************************************************
55    uswtime(&utime0, &stime0, &wtime0);
56    //*****************************************************************
57
58    //*****************************************************************
59    //Algoritmo
60    //*****************************************************************
61    array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
       ↪  numeros a ordenar
62    for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
       ↪  espacio de memoria pedido anteriormente
63    scanf("%d", array+i);
64    }
65    BurbujaOpt(array, n);
66       //*****************************************************************
67    //Evaluar los tiempos de ejecuciÃ³n
68    //*****************************************************************
69    uswtime(&utime1, &stime1, &wtime1);
70
71    //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
72    printf("\n");
73    printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
74    printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
75    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
```

```
76    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
      ↪  (wtime1 - wtime0));
77    printf("\n");
78
79    //Mostrar los tiempos en formato exponecial
80    printf("\n");
81    printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
82    printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
83    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
84    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
      ↪  (wtime1 - wtime0));
85    printf("\n");
86    //*****************************************************************
87
88    //Terminar programa normalmente
89    exit (0);
90
91 }
```

3.3. Inserción

```
1  LORETTO ESTRADA GALILEA AMERICA
2  PEREZ GARCIA ATZIRI***/
3  /**********LIBRERIAS*******************/
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7  #include "tiempo.h"
8
9  /*******************************Funcion Insercion*******************************/
10 //Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la
   ↪  longitud
11 void Insercion(int* Arreglo, int n){
12    int auxiliar, j;
13    for(int i=0; i<n ; i++){ // Ordenar los n elementos
14       j = i; // los iteradores i y j se igualan
15       auxiliar = Arreglo[i]; // Auxiliar toma el valor de la posicion i del arreglo
16       while((j>0)&&(auxiliar<Arreglo[j-1])){
17          Arreglo[j] = Arreglo[j-1]; // Se le asigna el  valor de la posicion anterior
18          j--; // Disminuye el iterador j.
19       }
20       Arreglo[j] = auxiliar; // se asigna el valor de la posicion j del arreglo a
         ↪  auxiliar
21    }
22 }
23 /*******************************Funcion PRINCIPAL*******************************/
24 //Toma como entrada la longitud del arreglo.
25 int main(int argc, char const *argv[]){
26    int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
```

```
27    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
       ↪  tiempos
28    int n;
29    //*************************************************************
30    //RecepciÃ³n y decodificaciÃ³n de argumentos
31    //*************************************************************
32
33    //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
34    if (argc!=2)
35    {
36       printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
       ↪  100\n",argv[0]);
37       exit(1);
38    }
39    //Tomar el segundo argumento como tamaÃ±o del algoritmo
40    else
41    {
42       n=atoi(argv[1]);
43    }
44
45    //*************************************************************
46    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
47    //*************************************************************
48    uswtime(&utime0, &stime0, &wtime0);
49    //*************************************************************
50
51    //*************************************************************
52    //Algoritmo
53    //*************************************************************
54    array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
       ↪  numeros a ordenar
55    for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
       ↪  espacio de memoria pedido anteriormente
56    scanf("%d", array+i);
57    }
58    Insercion(array, n);
59       //*************************************************************
60    //Evaluar los tiempos de ejecuciÃ³n
61    //*************************************************************
62    uswtime(&utime1, &stime1, &wtime1);
63
64    //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
65    printf("\n");
66    printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
67    printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
68    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
69    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
       ↪  (wtime1 - wtime0));
70    printf("\n");
71
```

```
72    //Mostrar los tiempos en formato exponecial
73    printf("\n");
74    printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
75    printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
76    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
77    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
      ↪  (wtime1 - wtime0));
78    printf("\n");
79    //*****************************************************************
80
81    //Terminar programa normalmente
82    exit (0);
83
84 }
```

3.4. Selección

```
1  /*****************SELECCION*************************
2  LORETTO ESTRADA GALILEA AMÃ%RICA
3  PEREZ GARCIA ATZIRI***/
4  /***********LIBRERIAS*******************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include "tiempo.h"
9
10 /*******************************Funcion Seleccion*****************************/
11 //Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la
   ↪  longitud
12 void Seleccion(int *A, int n){
13    int min = 0, aux = 0; // se declara la variable usada para determinar el numero
      ↪  minimo, y el auxiliar para realizar el intercambio
14    for(int i = 0; i<n; i++){ //Se realiza este proceso n veces para sacar el minimo n
      ↪  veces y acomodarlo en el lufar correcto
15       min = i;//Se establce la premisa inicial que el primer numero es el mÃnimo del
         ↪  arreglo
16       for (int j = i+1; j < n; ++j)//Se busca el minimo de todo el cjto de numeros. Se
         ↪  empieza desde +1 debido a que ocmo el primer numero es el minimo (segun
         ↪  premisa anterior) no es necesario incluir este numero.
17       {
18          if(*(A+j) < *(A+min))//Si el numero en el apuntador es menor al que se tenia
            ↪  considerado como mÃnimo:
19             min = j;//Se establece el nuevo Minimo
20       }
21       aux = *(A+min);//Se realiza el intercambio sabiendo ya que min va al principio
         ↪  del arreglo.
22       *(A+min) = *(A+i); //intercambian lugares, ahora el que es minimo toma el lugar
         ↪  del que se creia minimo anteriormente.
23       *(A+i) =  aux;//El minimo anterior toma el lugar en el que estaba el minimo
         ↪  nuevo.
```

```
24      }
25      //for(int i = 0; i <n; i++)
26      //   printf("%d\n",*(A+i));
27  }
28  /*******************************Funcion PRINCIPAL*****************************/
29  //Toma como entrada la longitud del arreglo.
30  int main(int argc, char const *argv[]){
31      int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
32      double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
        ↪  tiempos
33      int n;
34      //*************************************************************
35      //RecepciÃ³n y decodificaciÃ³n de argumentos
36      //*************************************************************
37
38      //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
39      if (argc!=2)
40      {
41          printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
            ↪  100\n",argv[0]);
42          exit(1);
43      }
44      //Tomar el segundo argumento como tamaÃ±o del algoritmo
45      else
46      {
47          n=atoi(argv[1]);
48      }
49
50      //*************************************************************
51      //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
52      //*************************************************************
53      uswtime(&utime0, &stime0, &wtime0);
54      //*************************************************************
55
56      //*************************************************************
57      //Algoritmo
58      //*************************************************************
59      array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
        ↪  numeros a ordenar
60      for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
        ↪  espacio de memoria pedido anteriormente
61      scanf("%d", array+i);
62      }
63      Seleccion(array, n);
64          //*************************************************************
65      //Evaluar los tiempos de ejecuciÃ³n
66      //*************************************************************
67      uswtime(&utime1, &stime1, &wtime1);
68
69      //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
```

```
70     printf("\n");
71     printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
72     printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
73     printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
74     printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
       ↪  (wtime1 - wtime0));
75     printf("\n");
76
77     //Mostrar los tiempos en formato exponecial
78     printf("\n");
79     printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
80     printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
81     printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
82     printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
       ↪  (wtime1 - wtime0));
83     printf("\n");
84     //*****************************************************************
85
86     //Terminar programa normalmente
87     exit (0);
88
89 }
```

3.5. Shell

```
1   /*********** SHELL ***********
2  LORETTO ESTRADA GALILEA AMERICA
3  PEREZ GARCIA ATZIRI*/
4  /***********LIBRERIAS*******************/
5  #include <stdlib.h>
6  #include <stdio.h>
7  #include <math.h> // Biblioteca para 'TRUNC'
8  #include <string.h>
9  #include "tiempo.h"
10
11 /*******************************Funcion Shell*****************************/
12 //Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la
    ↪  longitud
13 void Shell(int* A, int n){
14    int i,j,k,auxiliar; // Declarando variables auxiliares, iteradores
15    k = (n/2, -1); // Definiendo el tamaÃ±o del paso
16
17    while(k>0){
18       for(i=k;i<n; i++){ // Se lleva a cabo n-1 veces el procedimiento.
19          j = i; // Hacemos que i sea igual a j
20          auxiliar = A[i];
21          while((j>=k)&&(A[j-k]>auxiliar)){ // Hacemos el cambio si es que se cumple la
             ↪  condicion.
22             A[j] = A[j-k]; // Se hace el cambio.
23             j = j-k; // El iterador j cambia su valor con respecto de k.
```

```
24          }
25          A[j] = auxiliar; // El numero mayor se establece en la posicion j.
26       }
27       k=(k/2,-1); // Definiendo un nuevo tamaÃ±o de paso.
28    }
29    //for(int i = 0; i <n; i++)
30    //   printf("%d\n",*(A+i));
31
32 }
33 /*******************************Funcion PRINCIPAL*****************************/
34 //Toma como entrada la longitud del arreglo.
35 int main(int argc, char const *argv[]){
36    int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
37    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
      ↪  tiempos
38    int n;
39    //*************************************************************
40    //RecepciÃ³n y decodificaciÃ³n de argumentos
41    //*************************************************************
42
43    //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
44    if (argc!=2)
45    {
46       printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
         ↪  100\n",argv[0]);
47       exit(1);
48    }
49    //Tomar el segundo argumento como tamaÃ±o del algoritmo
50    else
51    {
52       n=atoi(argv[1]);
53    }
54
55    //*************************************************************
56    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
57    //*************************************************************
58    uswtime(&utime0, &stime0, &wtime0);
59    //*************************************************************
60
61    //*************************************************************
62    //Algoritmo (llamada a la funciÃ³n que contiene el algoritmo)
63    //*************************************************************
64    array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
      ↪  numeros a ordenar
65    for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
      ↪  espacio de memoria pedido anteriormente
66    scanf("%d", array+i);
67    }
68    Shell(array, n);
69    //*************************************************************
```

```
70      //Evaluar los tiempos de ejecuciÃ³n
71      //*****************************************************************
72      uswtime(&utime1, &stime1, &wtime1);
73
74      //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
75      printf("\n");
76      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
77      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
78      printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
79      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
        ↪  (wtime1 - wtime0));
80      printf("\n");
81
82      //Mostrar los tiempos en formato exponecial
83      printf("\n");
84      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
85      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
86      printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
87      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
        ↪  (wtime1 - wtime0));
88      printf("\n");
89      //*****************************************************************
90
91      //Terminar programa normalmente
92      exit (0);
93
94  }
```

3.6. Arbol de búsqueda binaria

3.6.1. Estructura del árbol

```
1   #include "Elem.h"
2   typedef struct Nodo
3   {
4     Elem raiz;
5     struct Nodo *izq;
6     struct Nodo *der;
7   } *Arbin;
8
9   Arbin
10  cons (Elem r, Arbin i, Arbin d)
11  {
12    Arbin t = (Arbin) malloc (sizeof (struct Nodo));
13    t->raiz = r;
14    t->izq = i;
15    t->der = d;
16    return t;
17  }
18
```

```
19 Arbin
20 vacio ()
21 {
22   return NULL;
23 }
24
25 int
26 esvacio (Arbin a)
27 {
28   return a == NULL;
29 }
30
31 Elem
32 raiz (Arbin a)
33 {
34   return a->raiz;
35 }
36
37 Arbin
38 izquierdo (Arbin a)
39 {
40   return a->izq;
41 }
42
43 Arbin
44 derecho (Arbin a)
45 {
46   return a->der;
47 }
48
49 void
50 ImpPreorden (Arbin a)
51 {
52   if (!esvacio (a))
53     {
54       ImpElem (raiz (a));
55       ImpPreorden (izquierdo (a));
56       ImpPreorden (derecho (a));
57     }
58 }
59
60
61 void
62 ImpInorden (Arbin a)
63 {
64   if (!esvacio (a))
65     {
66       ImpInorden (izquierdo (a));
67       ImpElem (raiz (a));
68       ImpInorden (derecho (a));
```

```
69      }
70  }
71
72  void
73  ImpPostorden (Arbin a)
74  {
75    if (!esvacio (a))
76      {
77        ImpPostorden (izquierdo (a));
78        ImpPostorden (derecho (a));
79        ImpElem (raiz (a));
80
81      }
82  }
83
84  int
85  maximo (int a, int b)
86  {
87    if (a > b)
88      return a;
89    else
90      return b;
91  }
92
93  int
94  altura (Arbin a)
95  {
96    if (esvacio (a))
97      return 0;
98    else
99      return 1 + maximo (altura (izquierdo (a)), altura (derecho (a)));
100 }
```

3.6.2. Busqueda Binaria

```
1   #include "ArBin.h"
2   typedef Arbin DicBin;
3
4
5   /*********************************Funcion InsOrd*****************************/
6   //Toma como entrada elemento por elemento que se quiere ingresar al arbol, y el Arbol
     ↪  en cuestiÃ³n.
7   DicBin InsOrd (Elem e, DicBin a)
8   {
9     //Si el arbol es vacio, solo se inserta el elemento en el hijo izquierdo
10    if (esvacio (a))
11      return cons (e, vacio (), vacio ());
12    //Si el Ã¡rbol no es vacio, se evalua si el elemento a ingresar es menor a la raiz
       ↪  del arbol
13    else if (EsMenor (e, raiz (a)))
```

```
14      //Si lo es, se ingresa el nuevo elemnto en el hijo izquierdo
15      return cons (raiz (a), InsOrd (e, izquierdo (a)), derecho (a));
16    else
17      //Si es mayor o igual, se ingresa el elemento en la hoja derecha
18      return cons (raiz (a), izquierdo (a), InsOrd (e, derecho (a)));
19  }
```

3.6.3. Programa final

```
1   /*********** BUSQUEDA BINARIA ***********
2   LORETTO ESTRADA GALILEA AMÃ%RICA
3   PEREZ GARCIA ATZIRI*/
4   /***********LIBRERIAS*******************/
5   #include <stdio.h>
6   #include <stdlib.h>
7   #include "BusBin.h"
8   #include <string.h>
9   #include "tiempo.h"
10
11  /*******************************Funcion PRINCIPAL*****************************/
12  //Toma como entrada la longitud del arreglo.
13  int main(int argc, char const *argv[]){
14    DicBin a = vacio ();
15    Elem *array;
16    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
      ↪  tiempos
17    int n;
18    //*****************************************************************
19    //RecepciÃ³n y decodificaciÃ³n de argumentos
20    //*****************************************************************
21
22    //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
23    if (argc!=2)
24    {
25      printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
        ↪  100\n",argv[0]);
26      exit(1);
27    }
28    //Tomar el segundo argumento como tamaÃ±o del algoritmo
29    else
30    {
31      n=atoi(argv[1]);
32    }
33    //*****************************************************************
34    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
35    //*****************************************************************
36    uswtime(&utime0, &stime0, &wtime0);
37    //*****************************************************************
38
39    //*****************************************************************
```

```
40    //Algoritmo (llamada a la funciÃ³n que contiene el algoritmo)
41    //*****************************************************************
42    array =(int*)(malloc(sizeof(int)*argc)); //se pide espacio en memoria para guardar
       ↪  los numeros a ordenar
43    for(int i=0; i<argc; i++){//Se guardan los elementos pedidos en la entrada en cada
       ↪  espacio de memoria pedido anteriormente
44      scanf("%d", array+i);
45      if (*(array+i))
46      a = InsOrd (*(array+i), a);//Se van guardando en orden dentro del arbol
47      }
48    //*****************************************************************
49      //Evaluar los tiempos de ejecuciÃ³n
50    //*****************************************************************
51    uswtime(&utime1, &stime1, &wtime1);
52
53    //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
54    printf("\n");
55    printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
56    printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
57    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
58    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1
       ↪  - wtime0));
59    printf("\n");
60
61    //Mostrar los tiempos en formato exponecial
62    printf("\n");
63    printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
64    printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
65    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
66    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1
       ↪  - wtime0));
67    printf("\n");
68    //*****************************************************************
69
70    //Terminar programa normalmente
71    exit (0);
72    return 0;
73  }
```

4. Especificaciones técnica

Plataforma	**Experimental**	
Hardware	Procesador	AMD E1 Essential
H Conjunto de instrucciones	Computadora 64 Bits	Lenovo G405 H
H	Reloj	1000Mhz
H	RAM	2GB
Software	Compilador	gcc
S	Sistema Operativo	Ubuntu

5. Gráficas de barras

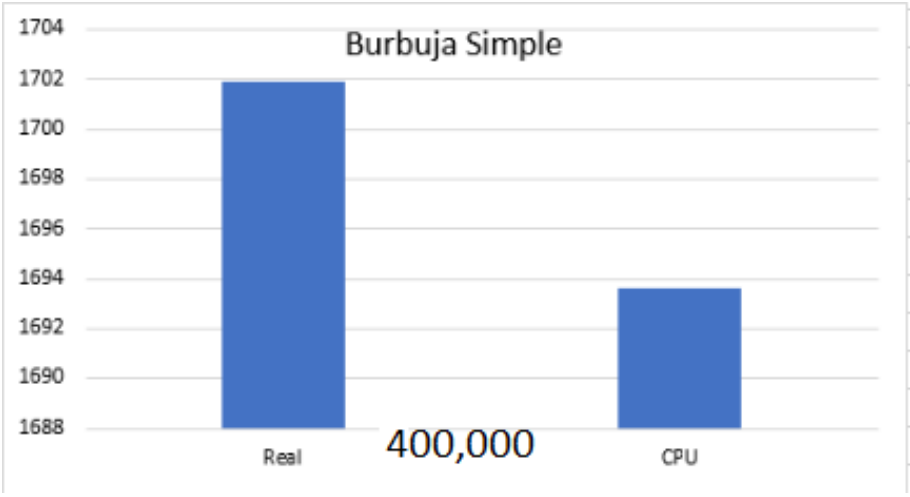

Figura 1: Burbuja Simple

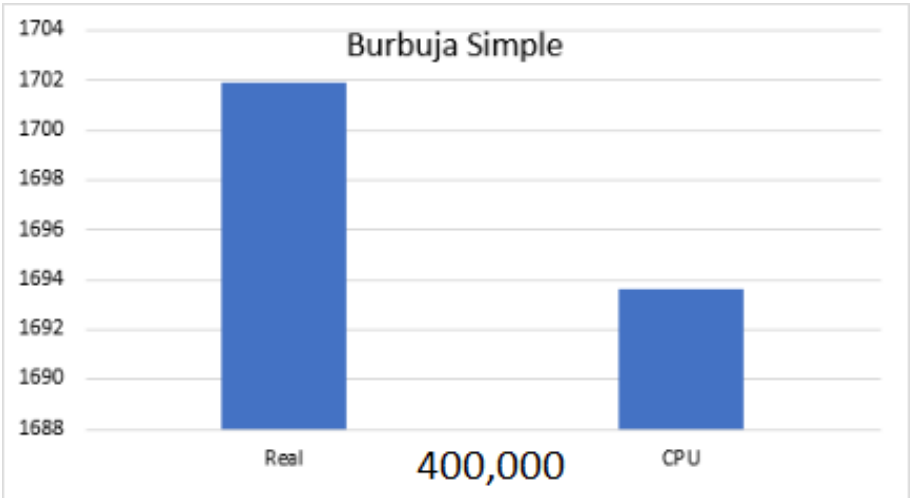

Figura 2: Burbuja Optimizada

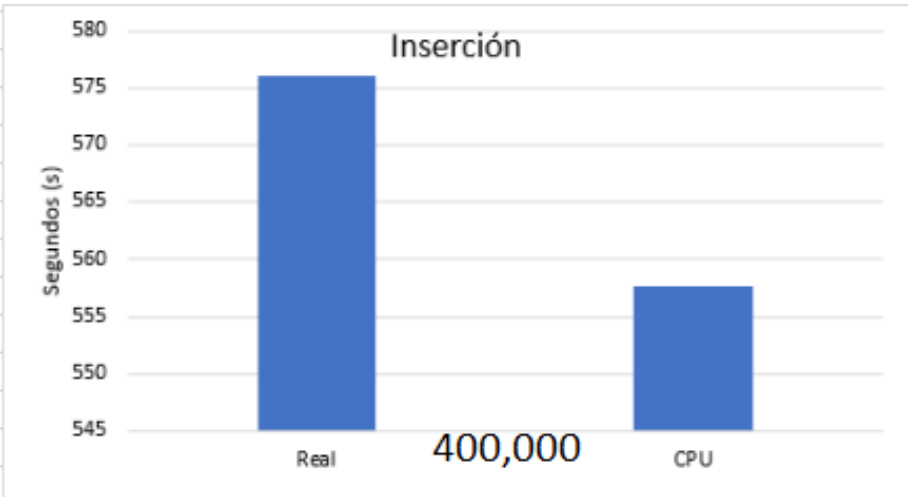

Figura 3: Inserción

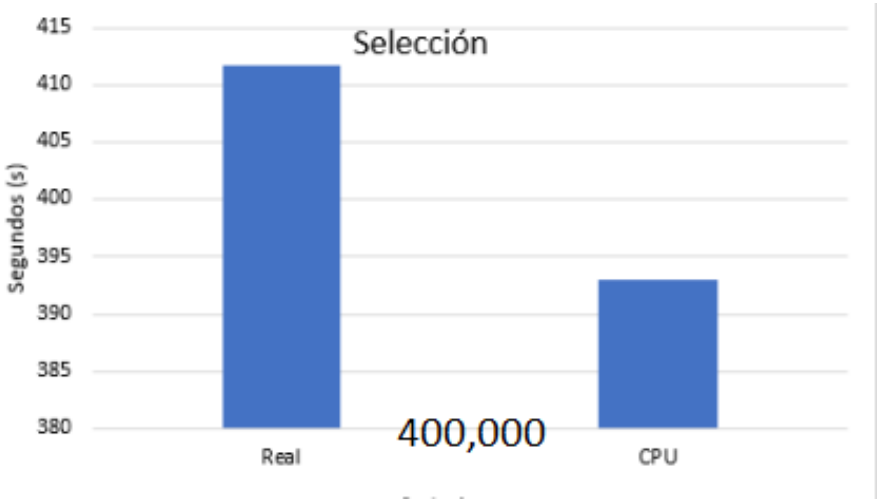

Figura 4: Selección

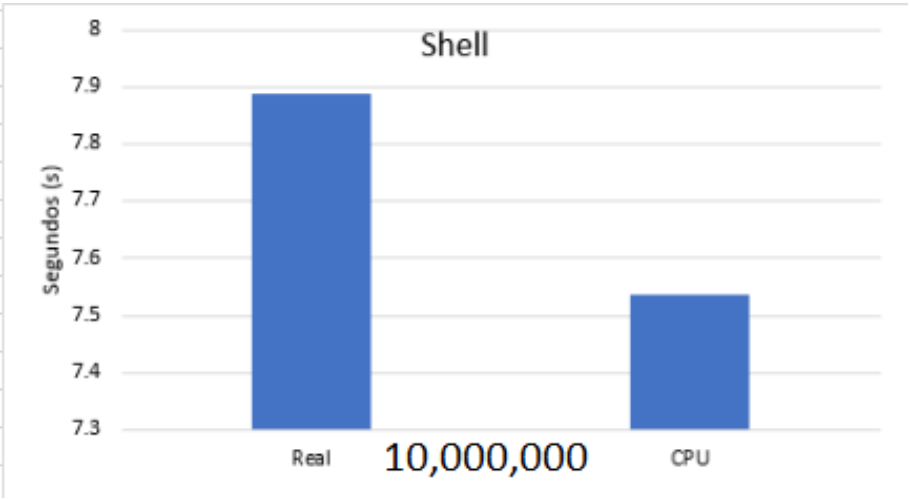

Figura 5: Shell

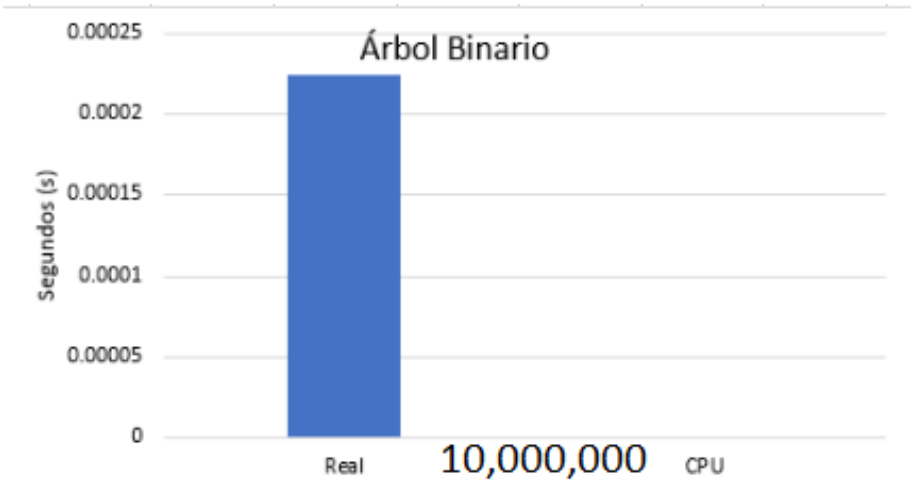

Figura 6: Arbol Binario

6. Comparación de resultados

Algoritmo	**Numeros ordenados**	**Tiempo real (s)**	**Tiempo CPU (s)**	**Tiempo E/S (s)**	**CPU/Wall**
Burbuja Simple	400000	1701.926	1693.64	0.4	99.537 %
Burbuja Optimizada	400000	2675.73	2662.656	1.18	99.553 %
Inserción	400000	576.1012	557.664	0.512	96.88 %
Selección	400000	411.852	392.975	0.103	95.5448 %
Shell	10000000	7.888	7.536	0.268	98.93 %
Burbuja Simple	10000000	0.000225	0.00	0.0	0.0001 %

7. Comportamiento temporal de cada algoritmo y aproximacion polinomial

7.1. Burbuja Simple

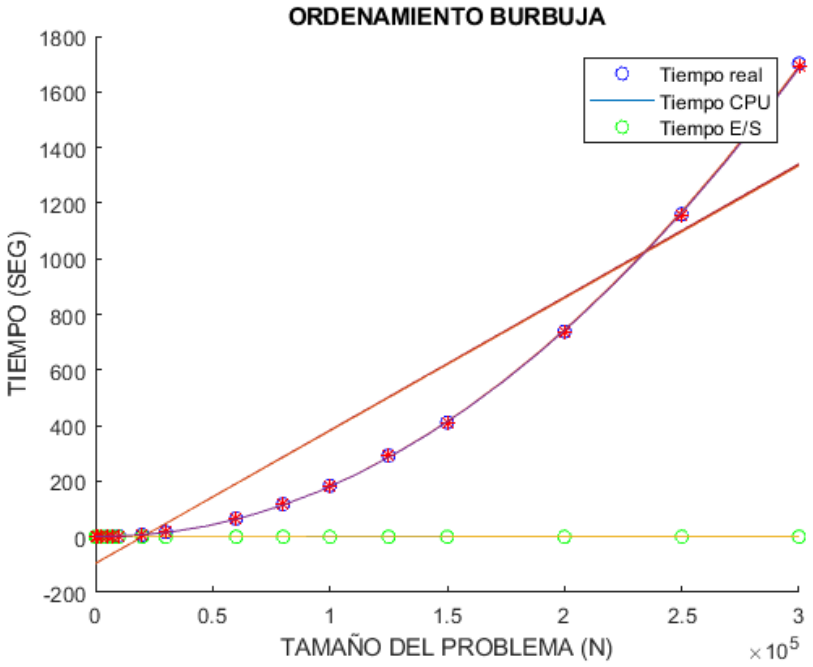

Figura 7: Burbuja Simple Polinomio Grado 1

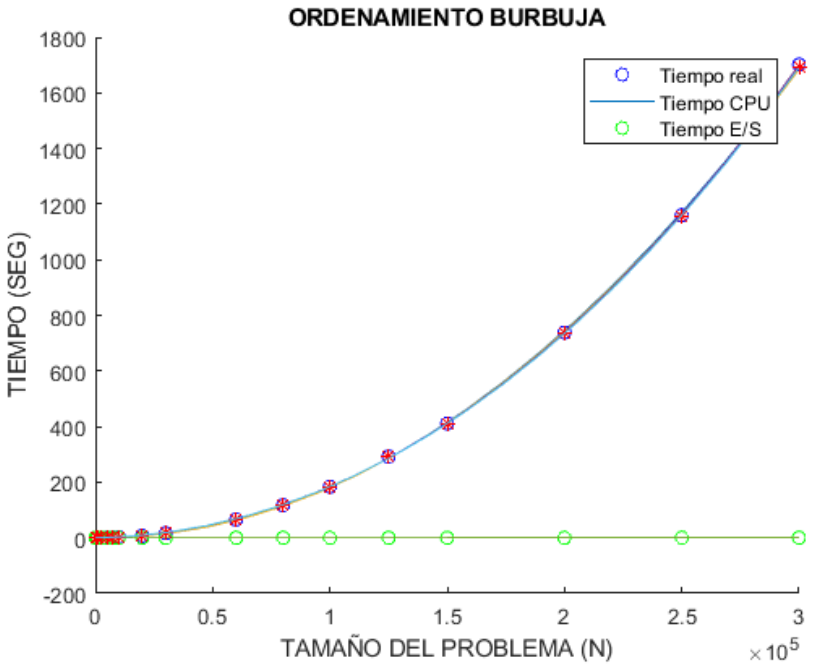

Figura 8: Burbuja Simple Polinomio Grado 2

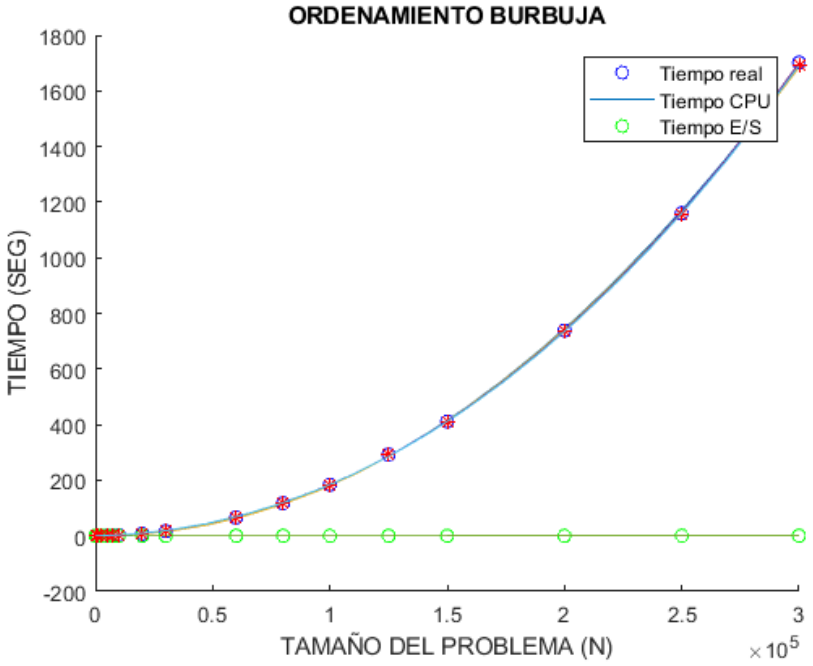

Figura 9: Burbuja Simple Polinomio Grado 3

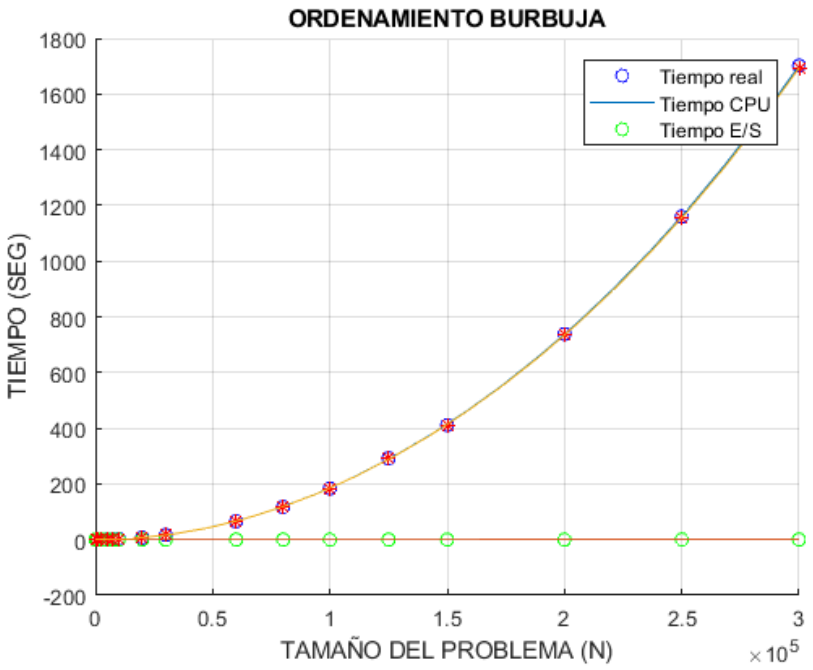

Figura 10: Burbuja Simple Polinomio Grado 4

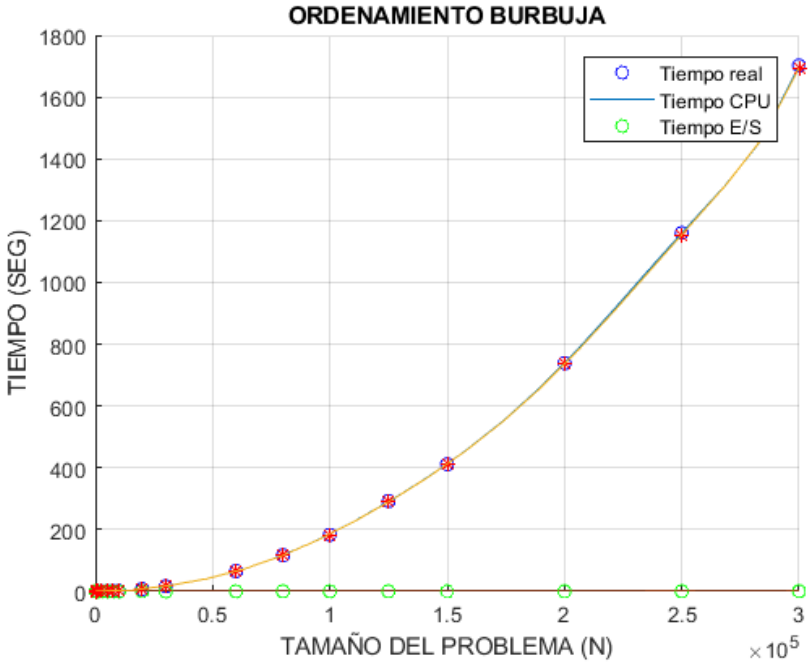

Figura 11: Burbuja Simple Polinomio Grado 8

7.2. Burbuja Optimizada

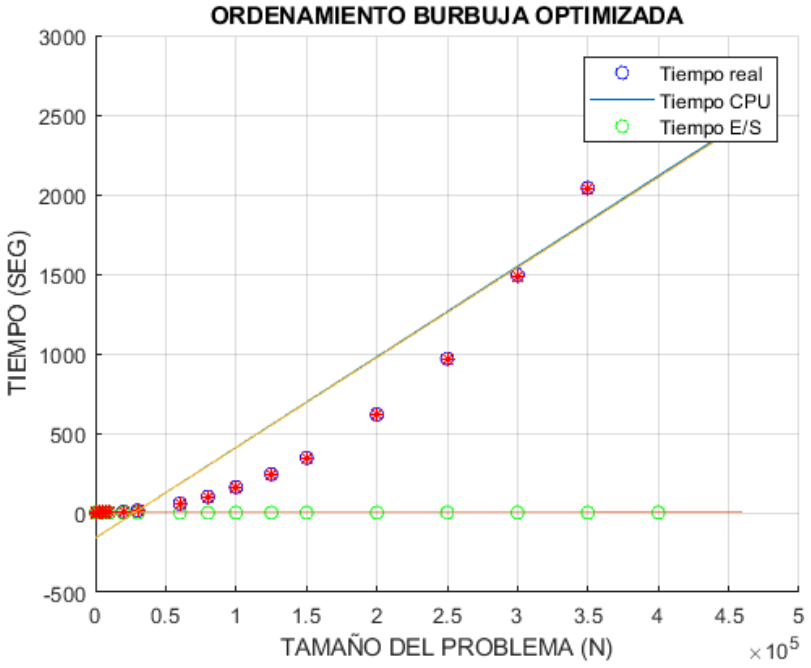

Figura 12: Burbuja Optimizada Polinomio Grado 1

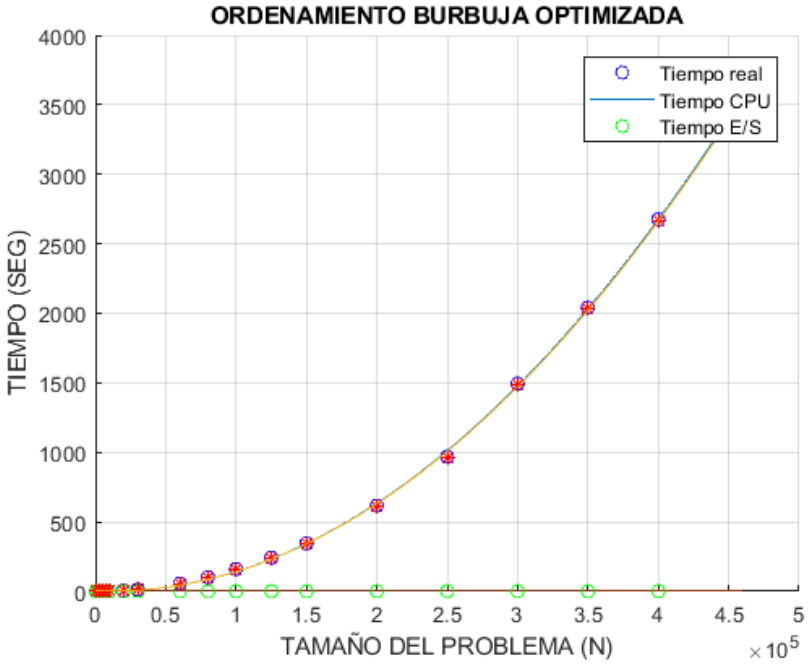

Figura 13: Burbuja Optimizada Polinomio Grado 2

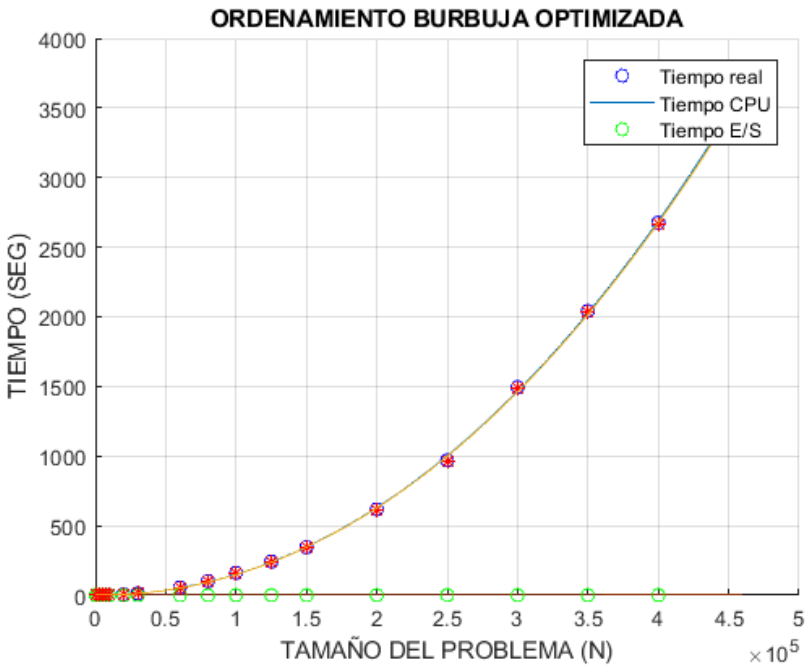

Figura 14: Burbuja Optimizada Polinomio Grado 3

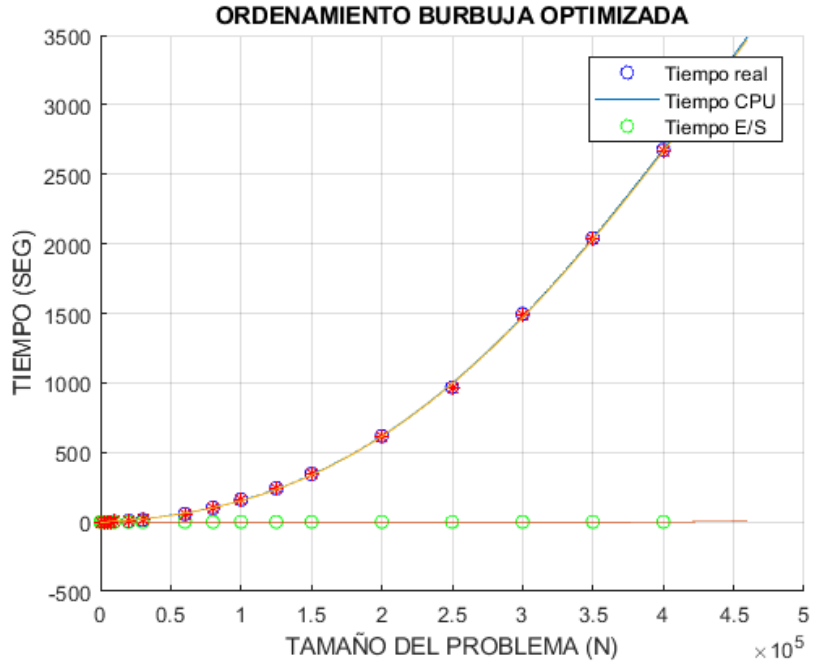

Figura 15: Burbuja Optimizada Polinomio Grado 4

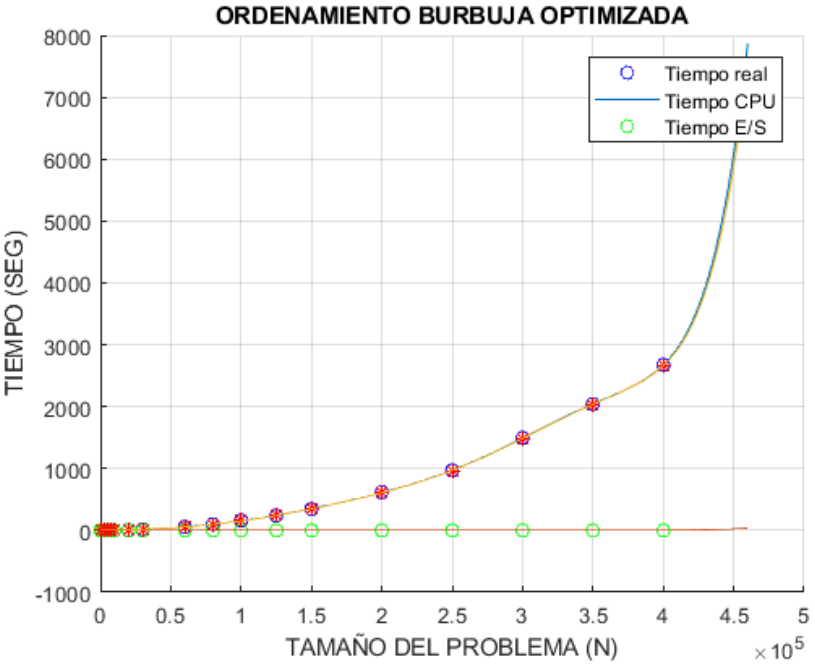

Figura 16: Burbuja Optimizada Polinomio Grado 8

7.3. Inserción

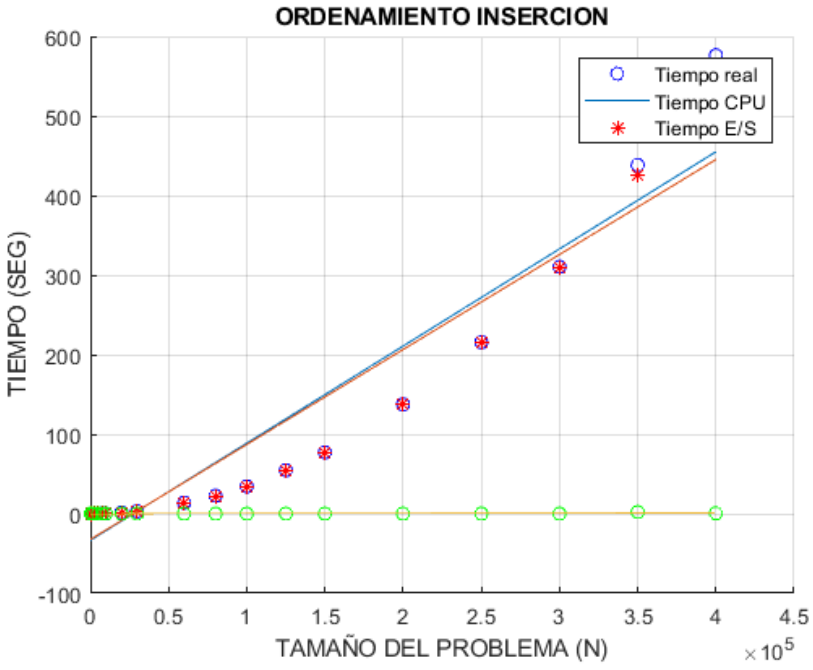

Figura 17: Inserción Polinomio Grado 1

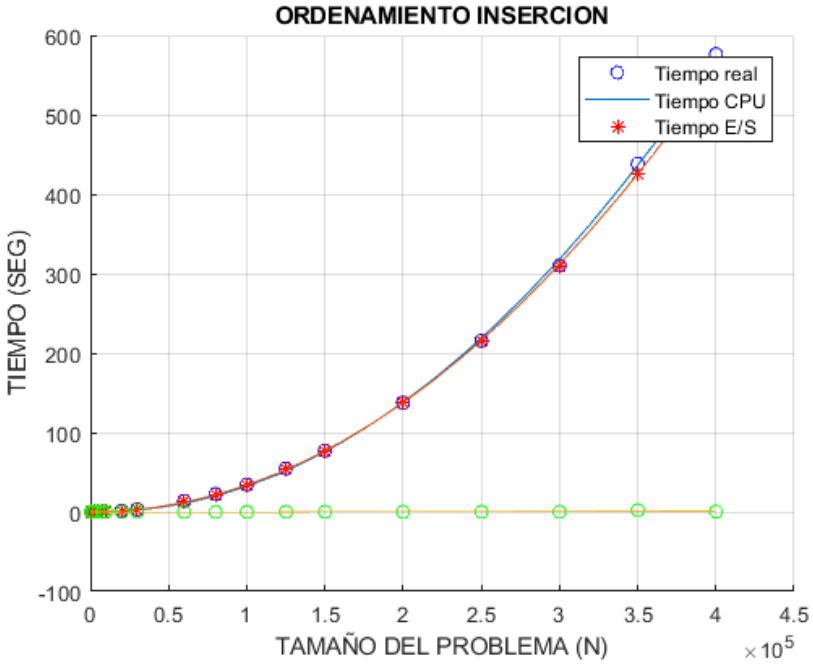

Figura 18: Inserción Polinomio Grado 2

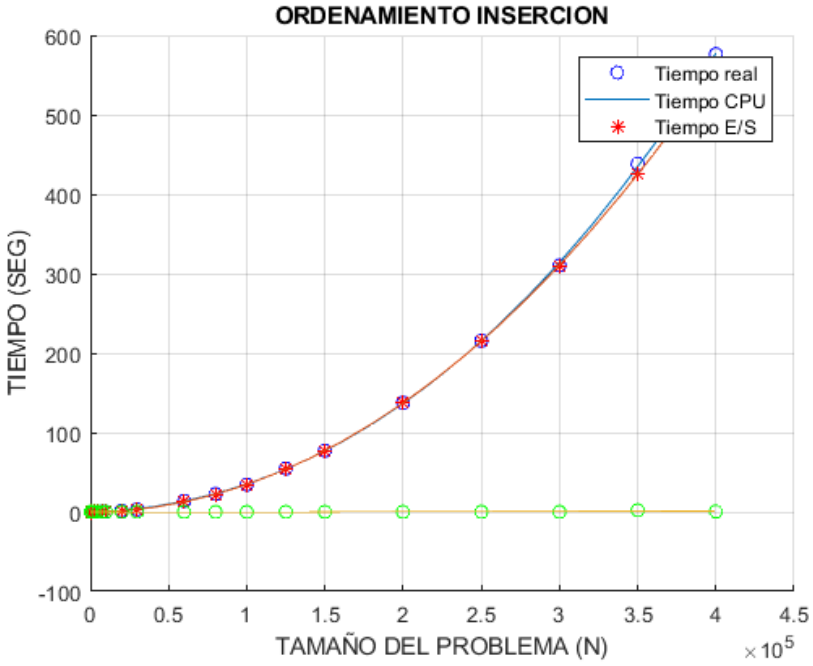

Figura 19: Inserción Polinomio Grado 3

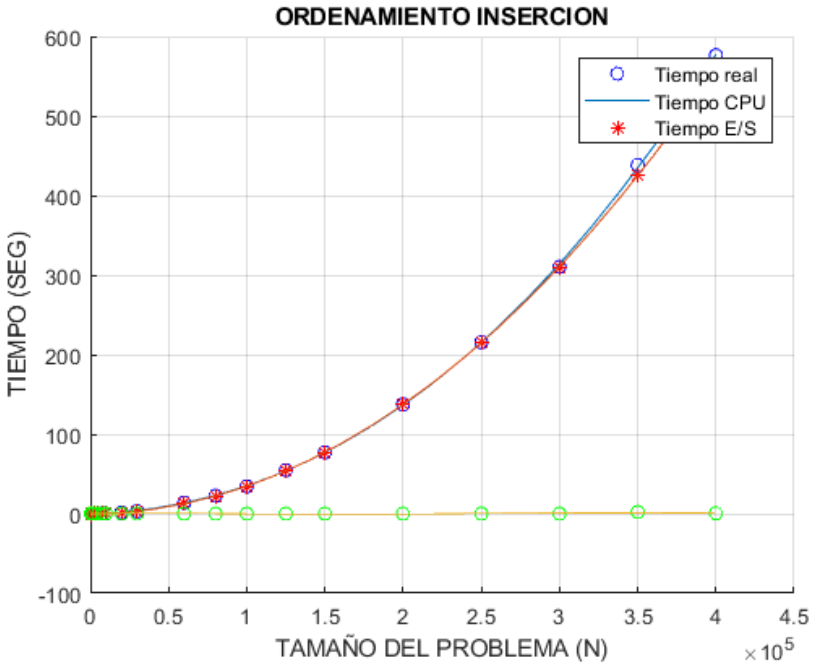

Figura 20: Inserción Polinomio Grado 4

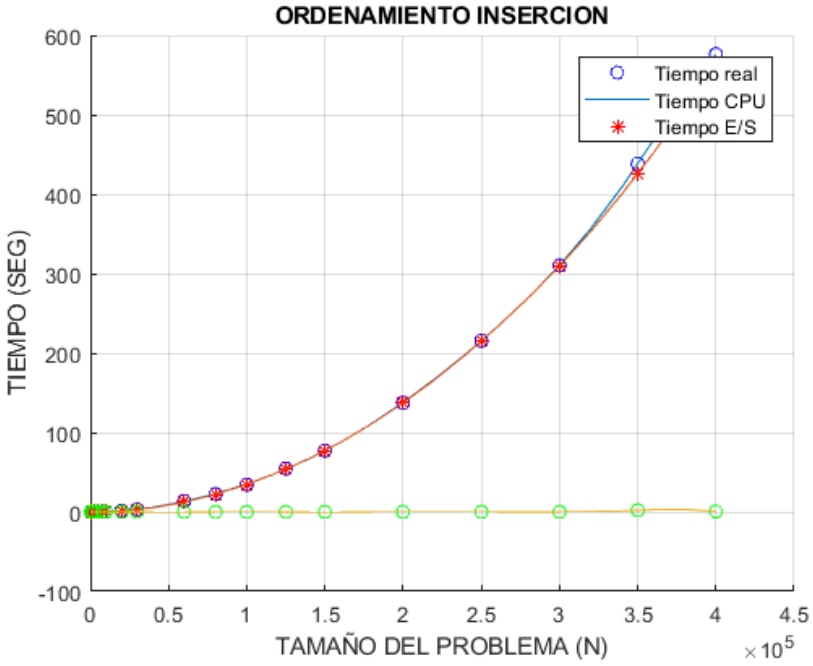

Figura 21: Inserción Polinomio Grado 8

7.4. Selección

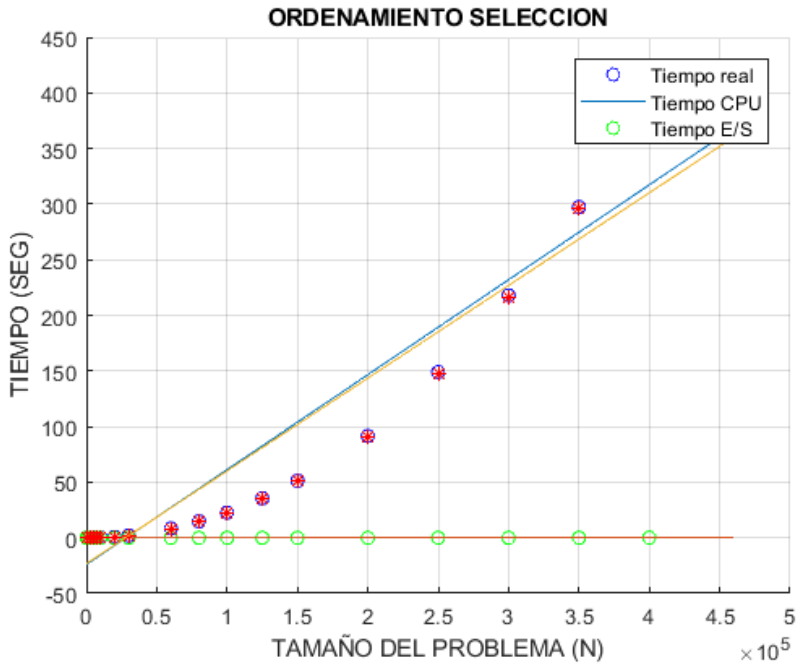

Figura 22: Selección Polinomio Grado 1

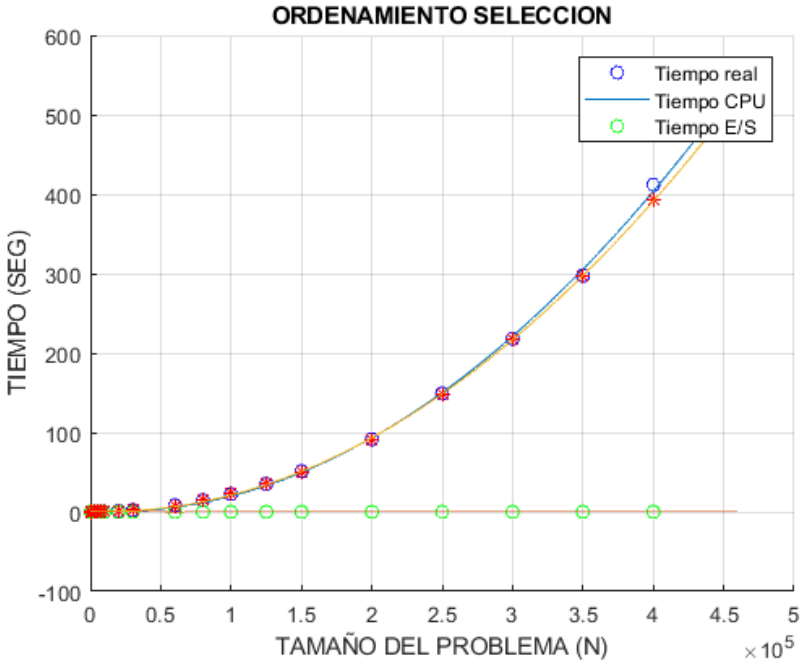

Figura 23: Selección Polinomio Grado 2

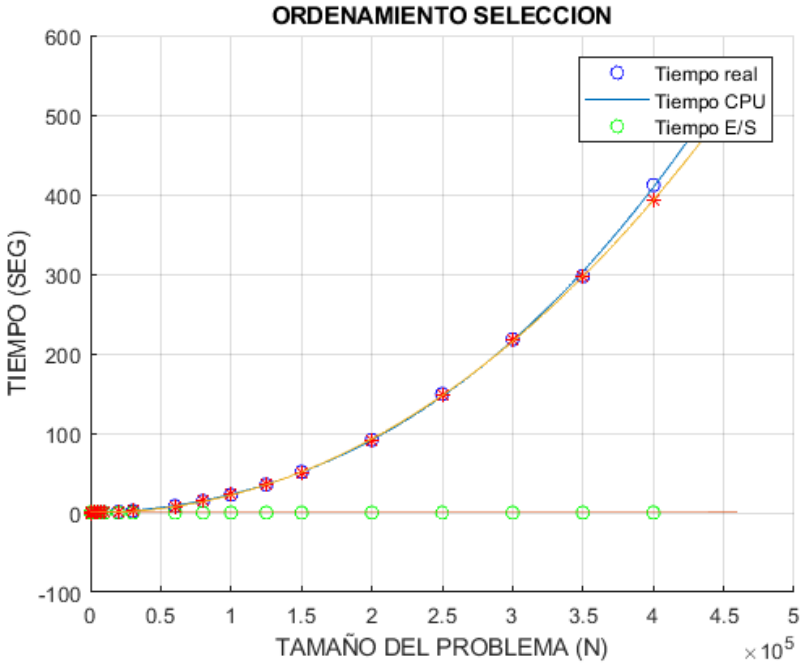

Figura 24: Selección Polinomio Grado 3

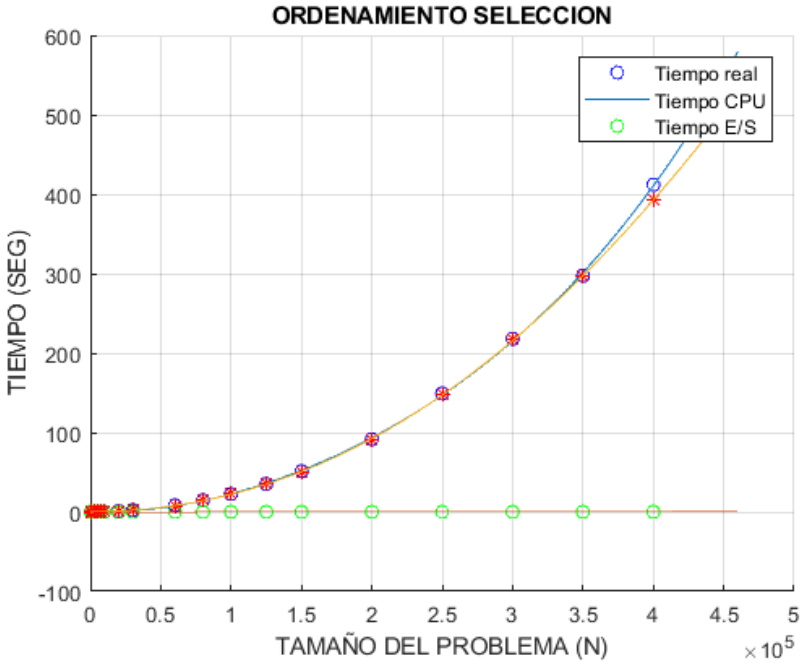

Figura 25: Selección Polinomio Grado 4

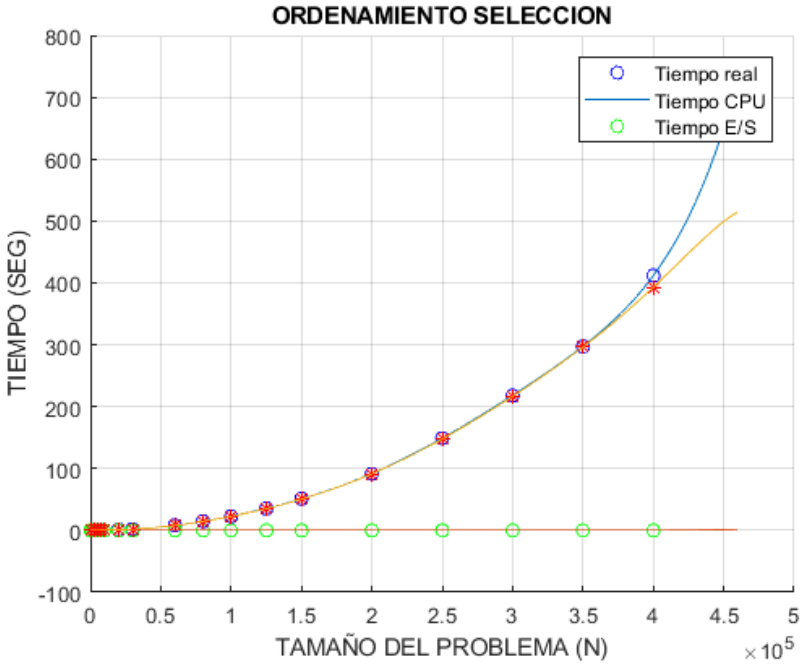

Figura 26: Selección Polinomio Grado 8

7.5. Shell

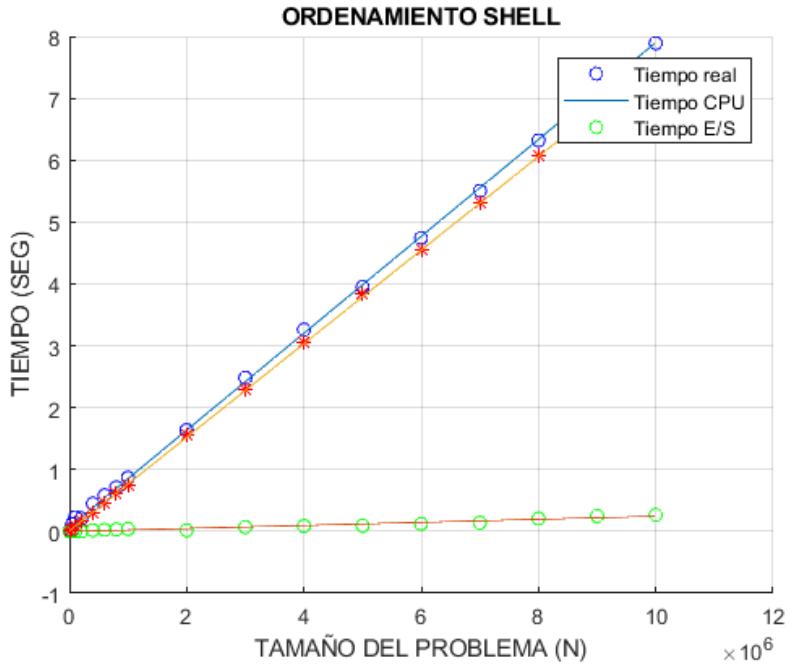

Figura 27: Shell Polinomio Grado 1

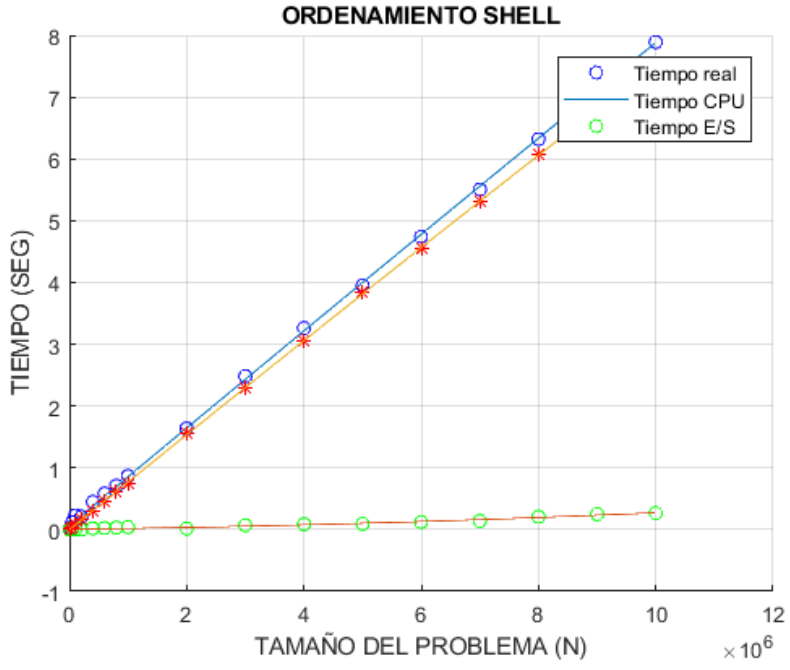

Figura 28: Shell Polinomio Grado 2

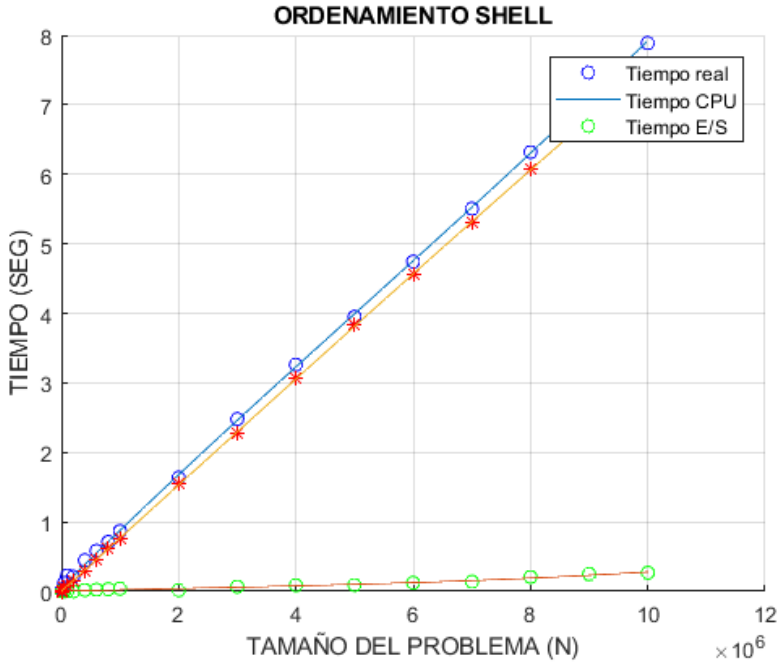

Figura 29: Shell Polinomio Grado 3

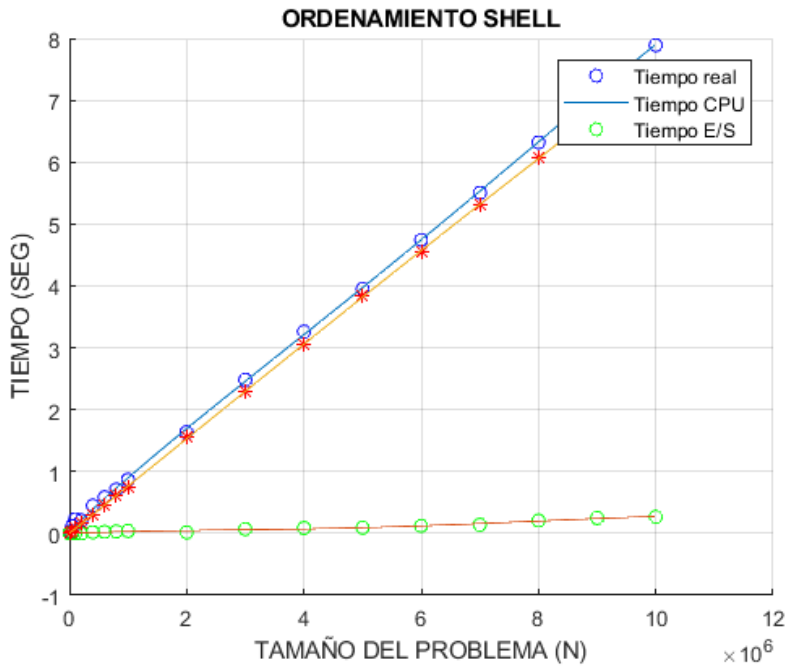

Figura 30: Shell Polinomio Grado 4

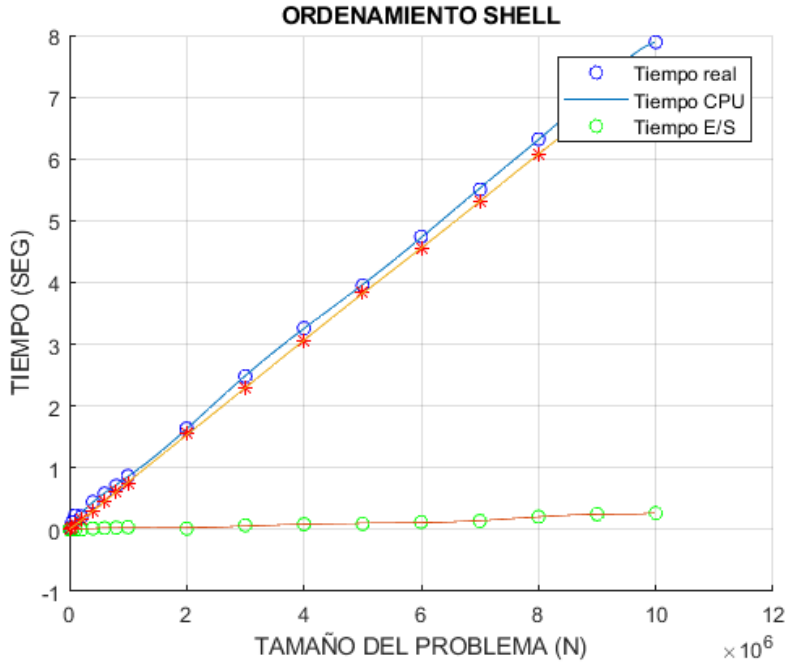

Figura 31: Shell Polinomio Grado 8

7.6. Arbol Binario

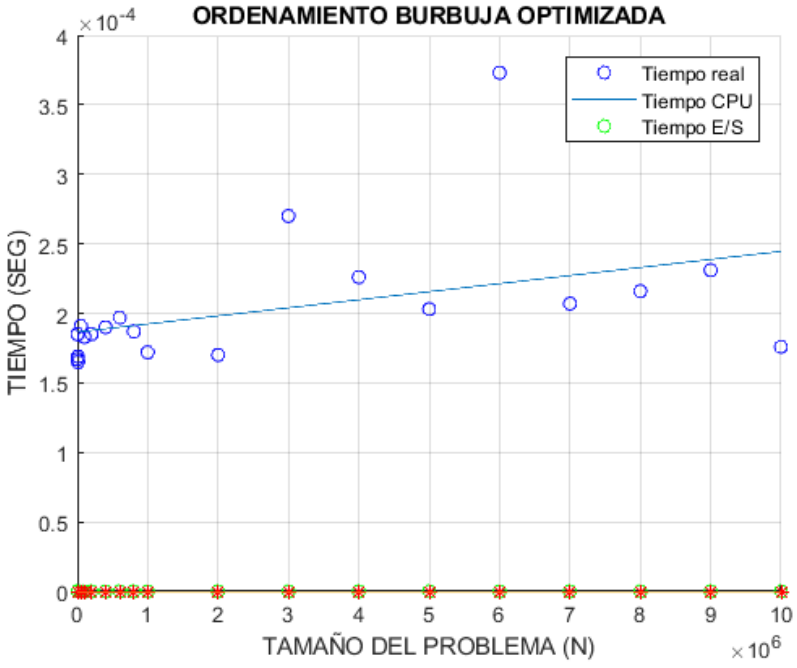

Figura 32: Shell Polinomio Grado 1

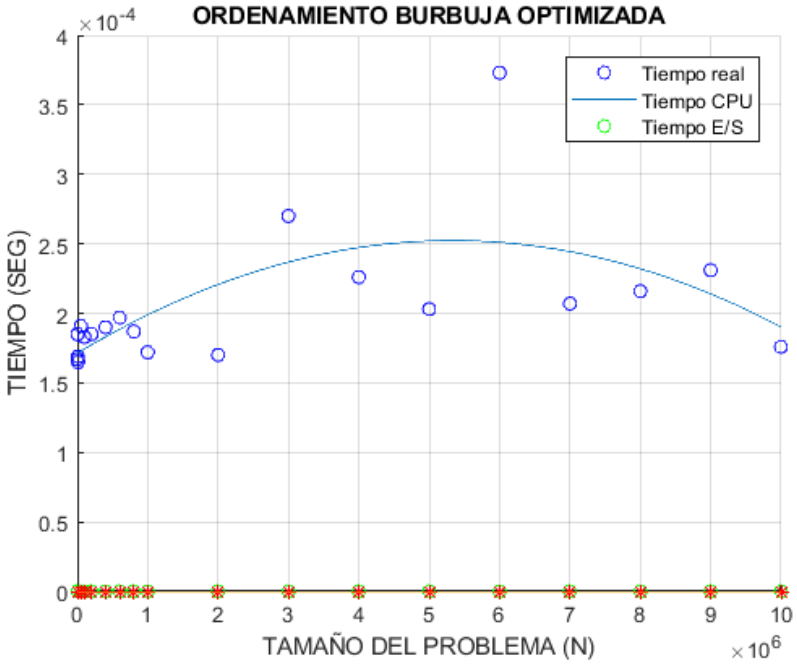

Figura 33: Shell Polinomio Grado 2

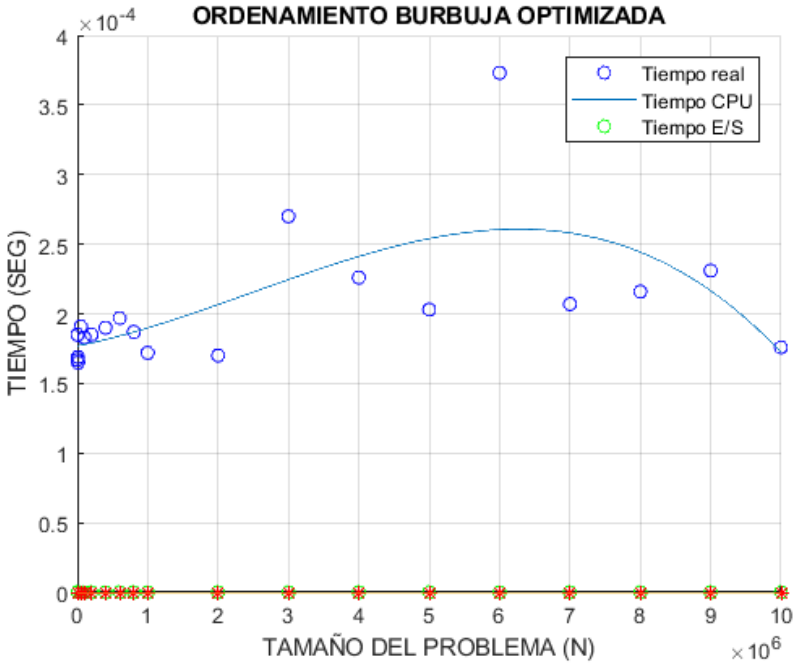

Figura 34: Shell Polinomio Grado 3

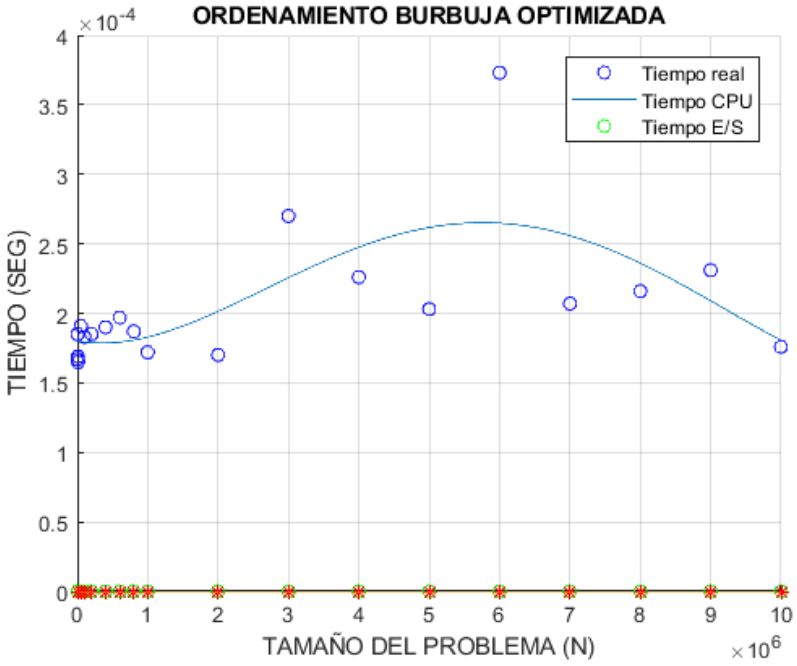

Figura 35: Shell Polinomio Grado 4

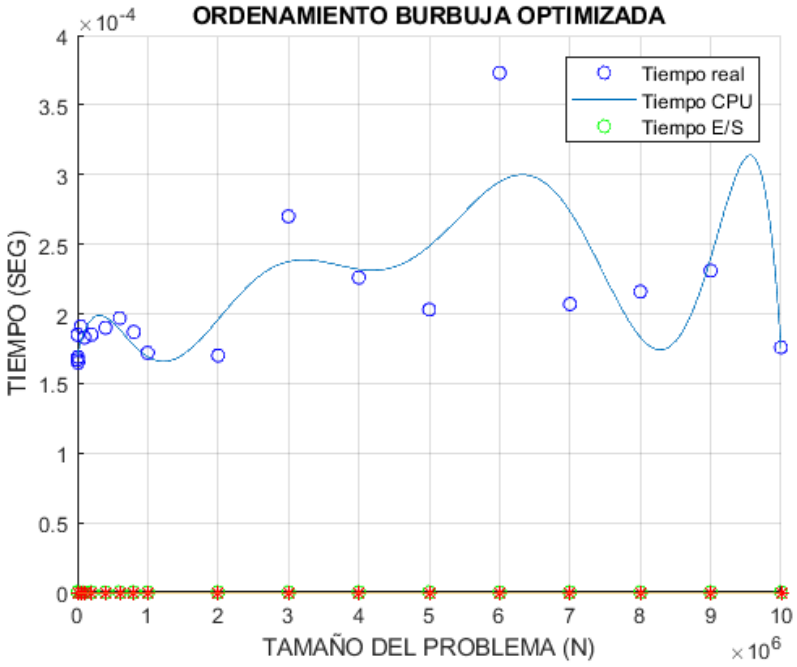

Figura 36: Shell Polinomio Grado 8

8. Comparación en tiempo real

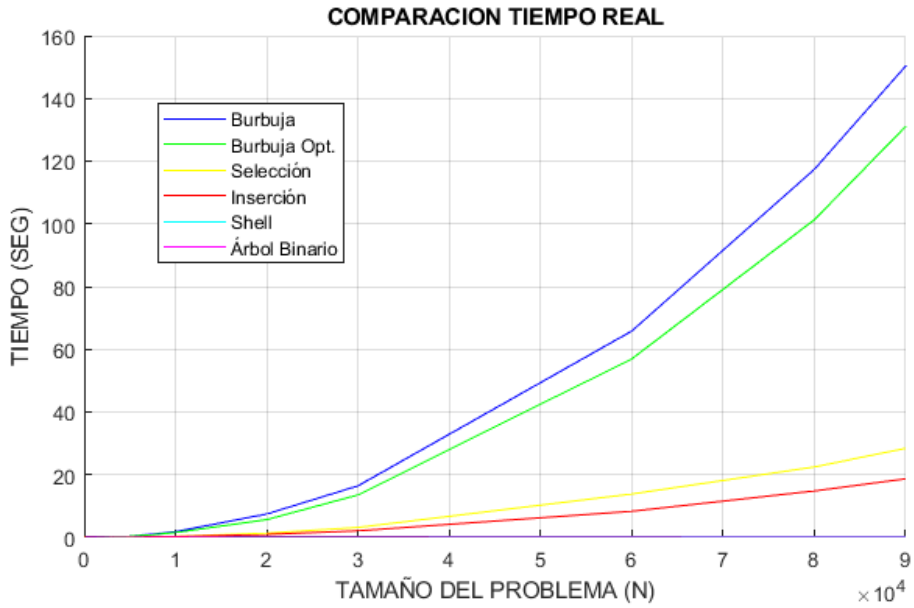

Figura 37: Tiempo Real 1

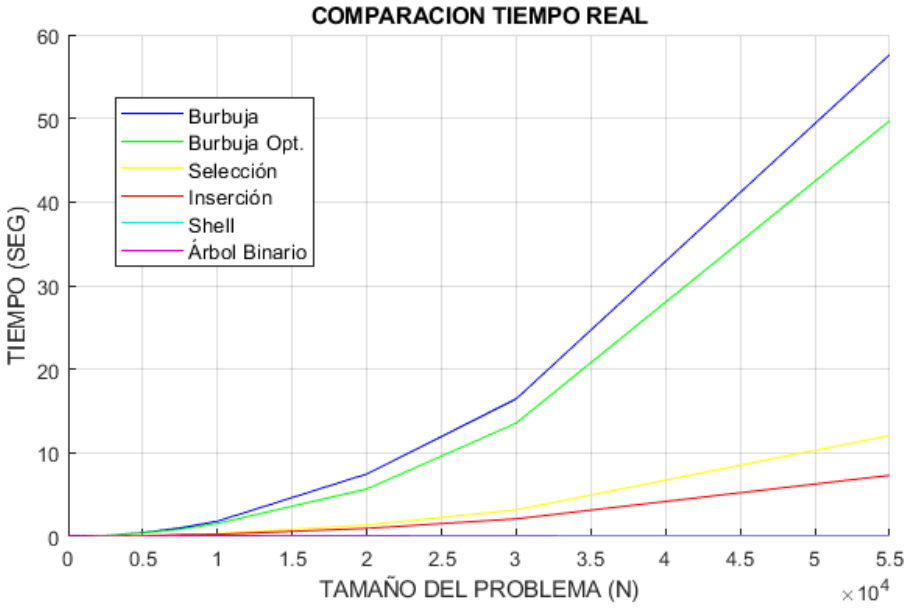

Figura 38: Tiempo Real 2

9. Preguntas

1. ¿Cuál de los 5 algoritmos es más fácil de implementar?
   Burbuja e Inserción.

2. ¿Cuál de los 5 algoritmos es el más difícil de implementar?
   Shell debido a que desconociamos el funcionamiento de este algoritmo

3. ¿Cuál algoritmo tiene menor complejidad temporal?
   Ordenamiento por Árbol Binario.

4. ¿Cuál algoritmo tiene mayor complejidad temporal?
   Burbuja Simple

5. ¿Cuál algoritmo tiene menor complejidad espacial? ¿Por qué?
   Burbuja, Inserción y ArboolBinario. Porque necesitan menos espacio en memoria para funcionar (A parte del tamaño del problema, usan pocas variables auxiliares).

6. ¿Cuál algoritmo tiene mayor complejidad espacial? ¿Por qué?
   Burbuja Optimizada y Selección ya que necesitan 2 variables más para su funcionamiento.

7. ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por que?
   Debido a que se desconocía el ordenamiento Shell, fue inesperado el poco tiempo para ordenar los 10 millones de números.Exceptuando este algoritmo, el tiempo de ejecución de los demás algoritmos fueron esperados. No obstante, otro factor fueron los pocos recursos con los que cuenta la máquina donde se ejecutaron las implementaciones.

8. ¿Sus resultados experimentales difieren mucho de los del resto de los equipos? ¿A que se debe?
   Si, debido a la máquina utilizada, pero finalmente tuvo el mismo comportamiento, en diferente escala.

9. ¿Existió un entorno controlado para realizar las pruebas experimentales? ¿Cuál fue?
   No, las implementaciones fueron ejecutadas con el compilador gcc desde la consola de Linux.

10. ¿Qué recomendaciones darían a nuevos equipos para realizar esta practica?
    Ser pacientes mientras se ejecutan las pruebas de cada implementación, tener instalado el software graficador e implementar los algoritmos tal como el pseudocódigo lo indica.

10. Anexos

10.1. Codigo Fuente

10.2. Burbuja Simple

```
1  /****************BURBUJA SIMPLE*************************
2  LORETTO ESTRADA GALILEA AMERICA
3  PEREZ GARCIA ATZIRI***/
4  /***********LIBRERIAS*******************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include "tiempo.h"
9
10 /*******************************Funcion Burbuja*****************************/
```

```
11 //Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la
    ↪  longitud
12 void Burbuja(int *A, int n){
13    int aux;//Se declara variable auxiliar para hacer el cambio
14    for(int i = 0; i<n-2; i++){//Se realiza n-1 veces el proceso para ordenar todos los
       ↪  numeros
15       for(int j=0; j<n-2; j++){
16          if(*(A+j) > *(A+j+1)){//Compracion si el primero nÃºmero es mayor al
             ↪  siguiente
17             aux = *(A+j);//Se asigna el valor del numero mayor de ambos a aux
18             *(A+j) = *(A+j+1);//Se hace el intercambio
19             *(A+j+1) = aux;//Se establece el numero mayor en la poscion +1
20          }
21       }
22    }
23 }
24 /*******************************Funcion PRINCIPAL*****************************/
25 //Toma como entrada la longitud del arreglo.
26 int main(int argc, char const *argv[]){
27    int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
28    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
       ↪  tiempos
29    int n;
30    //*****************************************************************
31    //RecepciÃ³n y decodificaciÃ³n de argumentos
32    //*****************************************************************
33
34    //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
35    if (argc!=2)
36    {
37       printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
          ↪  100\n",argv[0]);
38       exit(1);
39    }
40    //Tomar el segundo argumento como tamaÃ±o del algoritmo
41    else
42    {
43       n=atoi(argv[1]);
44    }
45
46    //*****************************************************************
47    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
48    //*****************************************************************
49    uswtime(&utime0, &stime0, &wtime0);
50    //*****************************************************************
51
52    //*****************************************************************
53    //Algoritmo
54    //*****************************************************************
```

```
55      array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
        ↪  numeros a ordenar
56      for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
        ↪  espacio de memoria pedido anteriormente
57      scanf("%d", array+i);
58      }
59      Burbuja(array, n);
60         //*****************************************************************
61      //Evaluar los tiempos de ejecuciÃ³n
62      //*****************************************************************
63      uswtime(&utime1, &stime1, &wtime1);
64
65      //Calculo del tiempo de ejecucion del programa
66      printf("\n");
67      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
68      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
69      printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
70      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
        ↪  (wtime1 - wtime0));
71      printf("\n");
72
73      //Mostrar los tiempos en formato exponecial
74      printf("\n");
75      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
76      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
77      printf("sys (Tiempo en acciones de E/S)  %.10e s\n",  stime1 - stime0);
78      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
        ↪  (wtime1 - wtime0));
79      printf("\n");
80      //*****************************************************************
81
82      //Terminar programa normalmente
83      exit (0);
84
85  }
```

10.3. Burbuja Optimizada

```
1   /****************BURBUJA OPTIMIZADA*************************
2   LORETTO ESTRADA GALILEA AMÉRICA
3   PEREZ GARCIA ATZIRI***/
4   /***********LIBRERIAS*******************/
5   #include <stdio.h>
6   #include <stdlib.h>
7   #include <string.h>
8   #include "tiempo.h"
9
10  /*********************************Funcion BurbujaOpt*****************************/
11  //Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la longitud
    ↪  de este.
```

```
12 void BurbujaOpt(int *A, int n){
13    int aux, i, cambios;//Se declaran variables auxiliares para hacer el cambio
14    cambios = 1;
15    i = 0;
16    while(i<n && cambios!=0){//Se realiza n-1 veces el proceso para ordenar todos los
       ↪  numeros
17       cambios= 0;
18       for(int j=0; j<n-2; j++){
19          if(*(A+i) < *(A+j)){//CompraciÃ³n si el primero nÃºmero es mayor al siguiente
20             aux = *(A+i);//Se asigna el valor del numero mayor de ambos a aux
21             *(A+i) = *(A+j);//Se hace el intercambio
22             *(A+j) = aux;//Se establece el numero mayor en la poscion +1
23             cambios = 1;
24          }
25       }
26       i++;
27    }
28 }
29
30 /*******************************Funcion PRINCIPAL*****************************/
31 //Toma como entrada la longitud del arreglo.
32 int main(int argc, char const *argv[]){
33    int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
34    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
       ↪  tiempos
35    int n;
36    //*****************************************************************
37    //RecepciÃ³n y decodificaciÃ³n de argumentos
38    //*****************************************************************
39
40    //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
41    if (argc!=2)
42    {
43       printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
          ↪  100\n",argv[0]);
44       exit(1);
45    }
46    //Tomar el segundo argumento como tamaÃ±o del algoritmo
47    else
48    {
49       n=atoi(argv[1]);
50    }
51
52    //*****************************************************************
53    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
54    //*****************************************************************
55    uswtime(&utime0, &stime0, &wtime0);
56    //*****************************************************************
57
58    //*****************************************************************
```

```
59      //Algoritmo
60      //*****************************************************************
61      array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
         ↪  numeros a ordenar
62      for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
         ↪  espacio de memoria pedido anteriormente
63      scanf("%d", array+i);
64      }
65      BurbujaOpt(array, n);
66         //*****************************************************************
67      //Evaluar los tiempos de ejecuciÃ³n
68      //*****************************************************************
69      uswtime(&utime1, &stime1, &wtime1);
70
71      //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
72      printf("\n");
73      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
74      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
75      printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
76      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
         ↪  (wtime1 - wtime0));
77      printf("\n");
78
79      //Mostrar los tiempos en formato exponecial
80      printf("\n");
81      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
82      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
83      printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
84      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
         ↪  (wtime1 - wtime0));
85      printf("\n");
86      //*****************************************************************
87
88      //Terminar programa normalmente
89      exit (0);
90
91  }
```

10.4. Inserción

```
1   LORETTO ESTRADA GALILEA AMERICA
2   PEREZ GARCIA ATZIRI***/
3   /**********LIBRERIAS*******************/
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <string.h>
7   #include "tiempo.h"
8
9   /*******************************Funcion Insercion*******************************/
```

```
10 //Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la
    ↪ longitud
11 void Insercion(int* Arreglo, int n){
12    int auxiliar, j;
13    for(int i=0; i<n ; i++){ // Ordenar los n elementos
14       j = i; // los iteradores i y j se igualan
15       auxiliar = Arreglo[i]; // Auxiliar toma el valor de la posicion i del arreglo
16       while((j>0)&&(auxiliar<Arreglo[j-1])){
17          Arreglo[j] = Arreglo[j-1]; // Se le asigna el  valor de la posicion anterior
18          j--; // Disminuye el iterador j.
19       }
20       Arreglo[j] = auxiliar; // se asigna el valor de la posicion j del arreglo a
          ↪ auxiliar
21    }
22 }
23 /*******************************Funcion PRINCIPAL*****************************/
24 //Toma como entrada la longitud del arreglo.
25 int main(int argc, char const *argv[]){
26    int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
27    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
       ↪ tiempos
28    int n;
29    //*****************************************************************
30    //RecepciÃ³n y decodificaciÃ³n de argumentos
31    //*****************************************************************
32
33    //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
34    if (argc!=2)
35    {
36       printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
          ↪ 100\n",argv[0]);
37       exit(1);
38    }
39    //Tomar el segundo argumento como tamaÃ±o del algoritmo
40    else
41    {
42       n=atoi(argv[1]);
43    }
44
45    //*****************************************************************
46    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
47    //*****************************************************************
48    uswtime(&utime0, &stime0, &wtime0);
49    //*****************************************************************
50
51    //*****************************************************************
52    //Algoritmo
53    //*****************************************************************
54    array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
       ↪ numeros a ordenar
```

```
55      for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
         ↪  espacio de memoria pedido anteriormente
56      scanf("%d", array+i);
57      }
58      Insercion(array, n);
59         //*****************************************************************
60      //Evaluar los tiempos de ejecuciÃ³n
61      //*****************************************************************
62      uswtime(&utime1, &stime1, &wtime1);
63
64      //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
65      printf("\n");
66      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
67      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
68      printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
69      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
         ↪  (wtime1 - wtime0));
70      printf("\n");
71
72      //Mostrar los tiempos en formato exponecial
73      printf("\n");
74      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
75      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
76      printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
77      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
         ↪  (wtime1 - wtime0));
78      printf("\n");
79      //*****************************************************************
80
81      //Terminar programa normalmente
82      exit (0);
83
84  }
```

10.5. Selección

```
1   /****************SELECCION**************************
2   LORETTO ESTRADA GALILEA AMÃ%RICA
3   PEREZ GARCIA ATZIRI***/
4   /**********LIBRERIAS*******************/
5   #include <stdio.h>
6   #include <stdlib.h>
7   #include <string.h>
8   #include "tiempo.h"
9
10  /*********************************Funcion Seleccion*****************************/
11  //Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la
     ↪  longitud
12  void Seleccion(int *A, int n){
```

```
13      int min = 0, aux = 0; // se declara la variable usada para determinar el numero
        ↪  minimo, y el auxiliar para realizar el intercambio
14      for(int i = 0; i<n; i++){ //Se realiza este proceso n veces para sacar el minimo n
        ↪  veces y acomodarlo en el lufar correcto
15          min = i;//Se establce la premisa inicial que el primer numero es el mÃnimo del
            ↪  arreglo
16          for (int j = i+1; j < n; ++j)//Se busca el minimo de todo el cjto de numeros. Se
            ↪  empieza desde +1 debido a que ocmo el primer numero es el minimo (segun
            ↪  premisa anterior) no es necesario incluir este numero.
17          {
18              if(*(A+j) < *(A+min))//Si el numero en el apuntador es menor al que se tenia
                ↪  considerado como mÃnimo:
19                  min = j;//Se establece el nuevo Minimo
20          }
21          aux = *(A+min);//Se realiza el intercambio sabiendo ya que min va al principio
            ↪  del arreglo.
22          *(A+min) = *(A+i); //intercambian lugares, ahora el que es minimo toma el lugar
            ↪  del que se creia minimo anteriormente.
23          *(A+i) =  aux;//El minimo anterior toma el lugar en el que estaba el minimo
            ↪  nuevo.
24      }
25      //for(int i = 0; i <n; i++)
26      //   printf("%d\n",*(A+i));
27  }
28  /*********************************Funcion PRINCIPAL*****************************/
29  //Toma como entrada la longitud del arreglo.
30  int main(int argc, char const *argv[]){
31      int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
32      double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
        ↪  tiempos
33      int n;
34      //*****************************************************************
35      //RecepciÃ³n y decodificaciÃ³n de argumentos
36      //*****************************************************************
37
38      //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
39      if (argc!=2)
40      {
41          printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
            ↪  100\n",argv[0]);
42          exit(1);
43      }
44      //Tomar el segundo argumento como tamaÃ±o del algoritmo
45      else
46      {
47          n=atoi(argv[1]);
48      }
49
50      //*****************************************************************
51      //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
```

```
52      //*****************************************************************
53      uswtime(&utime0, &stime0, &wtime0);
54      //*****************************************************************
55
56      //*****************************************************************
57      //Algoritmo
58      //*****************************************************************
59      array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
        ↪  numeros a ordenar
60      for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
        ↪  espacio de memoria pedido anteriormente
61      scanf("%d", array+i);
62      }
63      Seleccion(array, n);
64         //*****************************************************************
65      //Evaluar los tiempos de ejecuciÃ³n
66      //*****************************************************************
67      uswtime(&utime1, &stime1, &wtime1);
68
69      //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
70      printf("\n");
71      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
72      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
73      printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
74      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
        ↪  (wtime1 - wtime0));
75      printf("\n");
76
77      //Mostrar los tiempos en formato exponecial
78      printf("\n");
79      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
80      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
81      printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
82      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
        ↪  (wtime1 - wtime0));
83      printf("\n");
84      //*****************************************************************
85
86      //Terminar programa normalmente
87      exit (0);
88
89  }
```

10.6. Shell

```
1   /*********** SHELL ***********
2  LORETTO ESTRADA GALILEA AMERICA
3  PEREZ GARCIA ATZIRI*/
4  /***********LIBRERIAS*******************/
5  #include <stdlib.h>
```

```
6  #include <stdio.h>
7  #include <math.h> // Biblioteca para 'TRUNC'
8  #include <string.h>
9  #include "tiempo.h"
10
11 /*******************************Funcion Shell*****************************/
12 //Toma como entrada el arreglo donde es encuentran los nÃºmeros a ordenar y la
    ↪  longitud
13 void Shell(int* A, int n){
14    int i,j,k,auxiliar; // Declarando variables auxiliares, iteradores
15    k = (n/2, -1); // Definiendo el tamaÃ±o del paso
16
17    while(k>0){
18       for(i=k;i<n; i++){ // Se lleva a cabo n-1 veces el procedimiento.
19          j = i; // Hacemos que i sea igual a j
20          auxiliar = A[i];
21          while((j>=k)&&(A[j-k]>auxiliar)){ // Hacemos el cambio si es que se cumple la
             ↪  condicion.
22             A[j] = A[j-k]; // Se hace el cambio.
23             j = j-k; // El iterador j cambia su valor con respecto de k.
24          }
25          A[j] = auxiliar; // El numero mayor se establece en la posicion j.
26       }
27       k=(k/2,-1); // Definiendo un nuevo tamaÃ±o de paso.
28    }
29    //for(int i = 0; i <n; i++)
30    //   printf("%d\n",*(A+i));
31
32 }
33 /*******************************Funcion PRINCIPAL*************************/
34 //Toma como entrada la longitud del arreglo.
35 int main(int argc, char const *argv[]){
36    int *array;//Espacio en memoria donde se guardan los nÃºmeros a ordenar
37    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
       ↪  tiempos
38    int n;
39    //*****************************************************************
40    //RecepciÃ³n y decodificaciÃ³n de argumentos
41    //*****************************************************************
42
43    //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
44    if (argc!=2)
45    {
46       printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
          ↪  100\n",argv[0]);
47       exit(1);
48    }
49    //Tomar el segundo argumento como tamaÃ±o del algoritmo
50    else
51    {
```

```
52        n=atoi(argv[1]);
53    }
54
55    //*****************************************************************
56    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
57    //*****************************************************************
58    uswtime(&utime0, &stime0, &wtime0);
59    //*****************************************************************
60
61    //*****************************************************************
62    //Algoritmo (llamada a la funciÃ³n que contiene el algoritmo)
63    //*****************************************************************
64    array =(int*)(malloc(sizeof(int)*n)); //se pide espacio en memoria para guardar los
       ↪  numeros a ordenar
65    for(int i=0; i<n; i++){//Se guardan los elementos pedidos en la entrada en cada
       ↪  espacio de memoria pedido anteriormente
66    scanf("%d", array+i);
67    }
68    Shell(array, n);
69    //*****************************************************************
70    //Evaluar los tiempos de ejecuciÃ³n
71    //*****************************************************************
72    uswtime(&utime1, &stime1, &wtime1);
73
74    //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
75    printf("\n");
76    printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
77    printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
78    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
79    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
       ↪  (wtime1 - wtime0));
80    printf("\n");
81
82    //Mostrar los tiempos en formato exponecial
83    printf("\n");
84    printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
85    printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
86    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
87    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) /
       ↪  (wtime1 - wtime0));
88    printf("\n");
89    //*****************************************************************
90
91    //Terminar programa normalmente
92    exit (0);
93
94 }
```

10.7. Arbol de búsqueda binaria

10.7.1. Estructura del árbol

```
1  #include "Elem.h"
2  typedef struct Nodo
3  {
4    Elem raiz;
5    struct Nodo *izq;
6    struct Nodo *der;
7  } *Arbin;
8
9  Arbin
10 cons (Elem r, Arbin i, Arbin d)
11 {
12   Arbin t = (Arbin) malloc (sizeof (struct Nodo));
13   t->raiz = r;
14   t->izq = i;
15   t->der = d;
16   return t;
17 }
18
19 Arbin
20 vacio ()
21 {
22   return NULL;
23 }
24
25 int
26 esvacio (Arbin a)
27 {
28   return a == NULL;
29 }
30
31 Elem
32 raiz (Arbin a)
33 {
34   return a->raiz;
35 }
36
37 Arbin
38 izquierdo (Arbin a)
39 {
40   return a->izq;
41 }
42
43 Arbin
44 derecho (Arbin a)
45 {
46   return a->der;
```

```
47 }
48
49 void
50 ImpPreorden (Arbin a)
51 {
52   if (!esvacio (a))
53     {
54       ImpElem (raiz (a));
55       ImpPreorden (izquierdo (a));
56       ImpPreorden (derecho (a));
57     }
58 }
59
60
61 void
62 ImpInorden (Arbin a)
63 {
64   if (!esvacio (a))
65     {
66       ImpInorden (izquierdo (a));
67       ImpElem (raiz (a));
68       ImpInorden (derecho (a));
69     }
70 }
71
72 void
73 ImpPostorden (Arbin a)
74 {
75   if (!esvacio (a))
76     {
77       ImpPostorden (izquierdo (a));
78       ImpPostorden (derecho (a));
79       ImpElem (raiz (a));
80
81     }
82 }
83
84 int
85 maximo (int a, int b)
86 {
87   if (a > b)
88     return a;
89   else
90     return b;
91 }
92
93 int
94 altura (Arbin a)
95 {
96   if (esvacio (a))
```

```
97      return 0;
98    else
99      return 1 + maximo (altura (izquierdo (a)), altura (derecho (a)));
100 }
```

10.7.2. Busqueda Binaria

```
1  #include "ArBin.h"
2  typedef Arbin DicBin;
3
4
5  /*********************************Funcion InsOrd*****************************/
6  //Toma como entrada elemento por elemento que se quiere ingresar al arbol, y el Arbol
    ↪  en cuestiÃ³n.
7  DicBin InsOrd (Elem e, DicBin a)
8  {
9    //Si el arbol es vacio, solo se inserta el elemento en el hijo izquierdo
10   if (esvacio (a))
11     return cons (e, vacio (), vacio ());
12   //Si el Ã¡rbol no es vacio, se evalua si el elemento a ingresar es menor a la raiz
      ↪  del arbol
13   else if (EsMenor (e, raiz (a)))
14     //Si lo es, se ingresa el nuevo elemnto en el hijo izquierdo
15     return cons (raiz (a), InsOrd (e, izquierdo (a)), derecho (a));
16   else
17     //Si es mayor o igual, se ingresa el elemento en la hoja derecha
18     return cons (raiz (a), izquierdo (a), InsOrd (e, derecho (a)));
19 }
```

10.7.3. Programa final

```
1  /*********** BUSQUEDA BINARIA ***********
2  LORETTO ESTRADA GALILEA AMÃ¼RICA
3  PEREZ GARCIA ATZIRI*/
4  /***********LIBRERIAS*******************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include "BusBin.h"
8  #include <string.h>
9  #include "tiempo.h"
10
11 /*********************************Funcion PRINCIPAL*****************************/
12 //Toma como entrada la longitud del arreglo.
13 int main(int argc, char const *argv[]){
14   DicBin a = vacio ();
15   Elem *array;
16   double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para mediciÃ³n de
      ↪  tiempos
17   int n;
18   //*****************************************************************
```

```
19  //RecepciÃ³n y decodificaciÃ³n de argumentos
20  //*****************************************************************
21
22  //Si no se introducen exactamente 2 argumentos (Cadena de ejecuciÃ³n y cadena=n)
23  if (argc!=2)
24  {
25    printf("\nIndique el tamanio del algoritmo - Ejemplo: [user@equipo]$ %s
      ↪  100\n",argv[0]);
26    exit(1);
27  }
28  //Tomar el segundo argumento como tamaÃ±o del algoritmo
29  else
30  {
31    n=atoi(argv[1]);
32  }
33  //*****************************************************************
34  //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
35  //*****************************************************************
36  uswtime(&utime0, &stime0, &wtime0);
37  //*****************************************************************
38
39  //*****************************************************************
40  //Algoritmo (llamada a la funciÃ³n que contiene el algoritmo)
41  //*****************************************************************
42  array =(int*)(malloc(sizeof(int)*argc)); //se pide espacio en memoria para guardar
     ↪  los numeros a ordenar
43  for(int i=0; i<argc; i++){//Se guardan los elementos pedidos en la entrada en cada
     ↪  espacio de memoria pedido anteriormente
44    scanf("%d", array+i);
45    if (*(array+i))
46    a = InsOrd (*(array+i), a);//Se van guardando en orden dentro del arbol
47    }
48  //*****************************************************************
49    //Evaluar los tiempos de ejecuciÃ³n
50  //*****************************************************************
51  uswtime(&utime1, &stime1, &wtime1);
52
53  //CÃ¡lculo del tiempo de ejecuciÃ³n del programa
54  printf("\n");
55  printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
56  printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1 - utime0);
57  printf("sys (Tiempo en acciÃ³nes de E/S)  %.10f s\n",  stime1 - stime0);
58  printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1
     ↪  - wtime0));
59  printf("\n");
60
61  //Mostrar los tiempos en formato exponecial
62  printf("\n");
63  printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
64  printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1 - utime0);
```

```
65    printf("sys (Tiempo en acciÃ³nes de E/S)  %.10e s\n",  stime1 - stime0);
66    printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1
      ↪  - wtime0));
67    printf("\n");
68    //*****************************************************************
69
70    //Terminar programa normalmente
71    exit (0);
72    return 0;
73  }
```

10.8. Scripts

10.8.1. Burbuja Simple

```
#!/bin/bash
gcc burbuja.c tiempo.c -o bubble
./bubble 50 < numeros10millones.txt > bubble.txt
./bubble 100 < numeros10millones.txt >> bubble.txt
./bubble 500 < numeros10millones.txt >> bubble.txt
./bubble 1000 < numeros10millones.txt >> bubble.txt
./bubble 2500 < numeros10millones.txt >> bubble.txt
./bubble 5000 < numeros10millones.txt >> bubble.txt
./bubble 7500 < numeros10millones.txt >> bubble.txt
./bubble 10000 < numeros10millones.txt >> bubble.txt
./bubble 20000 < numeros10millones.txt >> bubble.txt
./bubble 30000 < numeros10millones.txt >> bubble.txt
./bubble 60000 < numeros10millones.txt >> bubble.txt
./bubble 80000 < numeros10millones.txt >> bubble.txt
./bubble 100000 < numeros10millones.txt >> bubble.txt
./bubble 125000 < numeros10millones.txt >> bubble.txt
./bubble 150000 < numeros10millones.txt >> bubble.txt
./bubble 200000 < numeros10millones.txt >> bubble.txt
./bubble 250000 < numeros10millones.txt >> bubble.txt
./bubble 300000 < numeros10millones.txt >> bubble.txt
./bubble 350000 < numeros10millones.txt >> bubble.txt
./bubble 400000 < numeros10millones.txt >> bubble.txt
```

10.8.2. Burbuja Optimizada

```
#!/bin/bash
gcc burbujaopt.c tiempo.c -o bubbleplus
./bubbleplus 50 < numeros10millones.txt > bubbleplus.txt
./bubbleplus 100 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 500 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 1000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 2500 < numeros10millones.txt >> bubbleplus.txt
```

```
./bubbleplus 5000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 7500 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 10000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 20000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 30000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 60000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 80000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 100000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 125000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 150000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 200000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 250000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 300000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 350000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 400000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 1000000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 5000000 < numeros10millones.txt >> bubbleplus.txt
./bubbleplus 8000000 < numeros10millones.txt >> bubbleplus.txt
```

10.8.3. Inserción

```
#!/bin/bash
gcc insercion.c tiempo.c -o inser
./inser 50 < numeros10millones.txt > inser.txt
./inser 100 < numeros10millones.txt >> inser.txt
./inser 500 < numeros10millones.txt >> inser.txt
./inser 1000 < numeros10millones.txt >> inser.txt
./inser 2500 < numeros10millones.txt >> inser.txt
./inser 5000 < numeros10millones.txt >> inser.txt
./inser 7500 < numeros10millones.txt >> inser.txt
./inser 10000 < numeros10millones.txt >> inser.txt
./inser 20000 < numeros10millones.txt >> inser.txt
./inser 30000 < numeros10millones.txt >> inser.txt
./inser 60000 < numeros10millones.txt >> inser.txt
./inser 80000 < numeros10millones.txt >> inser.txt
./inser 100000 < numeros10millones.txt >> inser.txt
./inser 125000 < numeros10millones.txt >> inser.txt
./inser 150000 < numeros10millones.txt >> inser.txt
./inser 200000 < numeros10millones.txt >> inser.txt
./inser 250000 < numeros10millones.txt >> inser.txt
./inser 300000 < numeros10millones.txt >> inser.txt
./inser 350000 < numeros10millones.txt >> inser.txt
./inser 400000 < numeros10millones.txt >> inser.txt
```

10.8.4. Selección

```
#!/bin/bash
gcc seleccion.c tiempo.c -o selec
./selec 50 < numeros10millones.txt > selec.txt
./selec 100 < numeros10millones.txt >> selec.txt
./selec 500 < numeros10millones.txt >> selec.txt
./selec 1000 < numeros10millones.txt >> selec.txt
./selec 2500 < numeros10millones.txt >> selec.txt
./selec 5000 < numeros10millones.txt >> selec.txt
./selec 7500 < numeros10millones.txt >> selec.txt
./selec 10000 < numeros10millones.txt >> selec.txt
./selec 20000 < numeros10millones.txt >> selec.txt
./selec 30000 < numeros10millones.txt >> selec.txt
./selec 60000 < numeros10millones.txt >> selec.txt
./selec 80000 < numeros10millones.txt >> selec.txt
./selec 100000 < numeros10millones.txt >> selec.txt
./selec 125000 < numeros10millones.txt >> selec.txt
./selec 150000 < numeros10millones.txt >> selec.txt
./selec 200000 < numeros10millones.txt >> selec.txt
./selec 250000 < numeros10millones.txt >> selec.txt
./selec 300000 < numeros10millones.txt >> selec.txt
./selec 350000 < numeros10millones.txt >> selec.txt
./selec 400000 < numeros10millones.txt >> selec.txt
```

10.8.5. Shell

```
#!/bin/bash
gcc shell.c tiempo.c -o concha
./concha 100 < numeros10millones.txt > concha.txt
./concha 1000 < numeros10millones.txt >> concha.txt
./concha 5000 < numeros10millones.txt >> concha.txt
./concha 10000 < numeros10millones.txt >> concha.txt
./concha 50000 < numeros10millones.txt >> concha.txt
./concha 100000 < numeros10millones.txt >> concha.txt
./concha 200000 < numeros10millones.txt >> concha.txt
./concha 400000 < numeros10millones.txt >> concha.txt
./concha 600000 < numeros10millones.txt >> concha.txt
./concha 800000 < numeros10millones.txt >> concha.txt
./concha 1000000 < numeros10millones.txt >> concha.txt
./concha 2000000 < numeros10millones.txt >> concha.txt
./concha 3000000 < numeros10millones.txt >> concha.txt
./concha 4000000 < numeros10millones.txt >> concha.txt
./concha 5000000 < numeros10millones.txt >> concha.txt
./concha 6000000 < numeros10millones.txt >> concha.txt
./concha 7000000 < numeros10millones.txt >> concha.txt
```

```
./concha 8000000 < numeros10millones.txt >> concha.txt
./concha 9000000 < numeros10millones.txt >> concha.txt
./concha 10000000 < numeros10millones.txt >> concha.txt
```

10.8.6. Arbol Binario

```
#!/bin/bash
gcc treeBin.c tiempo.c -o arbol
./arbol 100 < numeros10millones.txt > arbol.txt
./arbol 1000 < numeros10millones.txt >> arbol.txt
./arbol 5000 < numeros10millones.txt >> arbol.txt
./arbol 10000 < numeros10millones.txt >> arbol.txt
./arbol 50000 < numeros10millones.txt >> arbol.txt
./arbol 100000 < numeros10millones.txt >> arbol.txt
./arbol 200000 < numeros10millones.txt >> arbol.txt
./arbol 400000 < numeros10millones.txt >> arbol.txt
./arbol 600000 < numeros10millones.txt >> arbol.txt
./arbol 800000 < numeros10millones.txt >> arbol.txt
./arbol 1000000 < numeros10millones.txt >> arbol.txt
./arbol 2000000 < numeros10millones.txt >> arbol.txt
./arbol 3000000 < numeros10millones.txt >> arbol.txt
./arbol 4000000 < numeros10millones.txt >> arbol.txt
./arbol 5000000 < numeros10millones.txt >> arbol.txt
./arbol 6000000 < numeros10millones.txt >> arbol.txt
./arbol 7000000 < numeros10millones.txt >> arbol.txt
./arbol 8000000 < numeros10millones.txt >> arbol.txt
./arbol 9000000 < numeros10millones.txt >> arbol.txt
./arbol 10000000 < numeros10millones.txt >> arbol.txt
```

11. Referencias

http://www.eafranco.com/docencia/analisisdealgoritmos/files/practicas/01/Practica01.pdf Web Personal de Edgardo Adrián