



**INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**

ANÁLISIS DE ALGORITMOS

PROFESORA: LUZ MARÍA SÁNCHEZ GARCÍA

INTEGRANTES:

VÁZQUEZ MORENO MARCOS OSWALDO 2016601777

DE LOS SANTOS DÍAZ LUIS ALEJANDRO 2017630451

PRÁCTICA 3 ANÁLISIS DE ALGORITMOS DE ORDENAMIENTO

3CM2

09 DE MARZO DE 2019

Introducción

En esta práctica se estudiará el problema de ordenar un array de elementos sobre los cuales se puede establecer una relación de orden (i.e los operadores $<$, $=$ tienen sentido).

Los algoritmos de este documento serán escritos en C y serán intercambiables entre sí; es decir, todos aceptarán los mismos parámetros: un array A de datos y un entero que representa el tamaño del array.

Si bien en todo este documento se mostrará como ordenar de forma creciente (de menor a mayor), esperamos que el lector no tenga problemas realizar las modificaciones pertinentes (que deberán ser obvias) en los algoritmos para poder ordenar de forma decreciente. El array de entrada A tendrá elementos de tipo Dato los cuales podrán ser cualquier tipo de dato representable (una estructura, un objeto de una clase en particular, un número, un string, etc). Dicho array será usado al estilo C, es decir los índices de los N elementos de A serán 0, 1, 2, ..., N-1. Se supondrá por simplicidad que los datos aceptan el uso de operadores " $<$ " y " $>$ ". Si bien esto no será así en la práctica, no perderemos generalidad. La estrategia para establecer un orden entre los elementos dependerá de la tecnología que usemos para implementar el algoritmo de ordenación. En Appendix A se discute esto con más profundidad y para tres tecnologías de programación distintas: programación estructurada (C), programación orientada a objetos (C++) y programación de objetos (Smalltalk, Self, etc.) También se utilizará el operador de asignación de manera especial en algunos casos. Por ejemplo, en el siguiente segmento de código, en tmp se almacenará una copia del iésimo elemento del array A de entrada. (Gurin, 2004)

Se tomarán en cuenta los siguientes tipos de ordenamiento:

- **Ordenamiento de burbuja:** 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.
- **Ordenamiento de burbuja mejorada:** Este método consta de que los elementos que están detrás del que se está comparando, ya están ordenados, permite realizar un número menor de comparaciones. (Práctica3_AlgoritmosOrdenamiento, 2019)
- **Ordenamiento por selección:** Si el array de datos es A y su tamaño es N, lo que hace el algoritmo, para cada i de $[0..N-2]$ es intercambiar $A[i]$ con el mínimo elemento del subarray $[A[i+1], \dots, A[N]]$.
- **Ordenamiento por inserción:** Si la entrada esta "casi ordenada", el algoritmo se ejecuta mucho más rápidamente. Esta velocidad tiende a un tiempo $O(N)$, peor caso que se cumple cuando la entrada está totalmente ordenada.
- **Ordenamiento de Shell:** A diferencia del algoritmo de ordenación por inserción, este algoritmo intercambia elementos distantes. Es por esto que,

puede deshacer más de una inversión en cada intercambio, hecho del cual nos aprovechamos para ganar velocidad.

- **Ordenamiento de Árbol binario:** Una búsqueda binaria es aquella que realiza aproximaciones realizando comparaciones en una colección de datos previamente ordenadas con el fin de dividir el problema a razón de la mitad hasta encontrar el dato buscado

Planteamiento del problema

Programar en lenguaje C, cada uno de los algoritmos de ordenamiento mencionados.

El programa será capaz de recibir un parámetro “n” que indica el numero de enteros a ordenar a partir de un archivo con máximo 10,000,000 números en desorden.

Medir el tiempo que tarda cada algoritmo de ordenar el archivo completo y compare los tiempos de ejecución de cada uno.

Realizar un análisis temporal para cada algoritmo ordenando.

Graficar el comportamiento temporal de cada algoritmo.

Graficar una comparativa de los 5 algoritmos de ordenamiento.

Realizar una aproximación polinomial del comportamiento temporal, de cada uno de los algoritmos probados.

Determinar con base en las aproximaciones obtenidas cuál será el tiempo real de cada algoritmo.

Diseño de la solución

A continuación, se muestran los diagramas de flujo de nuestra propuesta de solución para los algoritmos de ordenamiento.

Primeramente, se muestra en el diagrama 1.1 el main.c de nuestra propuesta de solución.

A continuación, se muestra en el diagrama 1.2 la implementación de nuestro archivo Práctica3.c

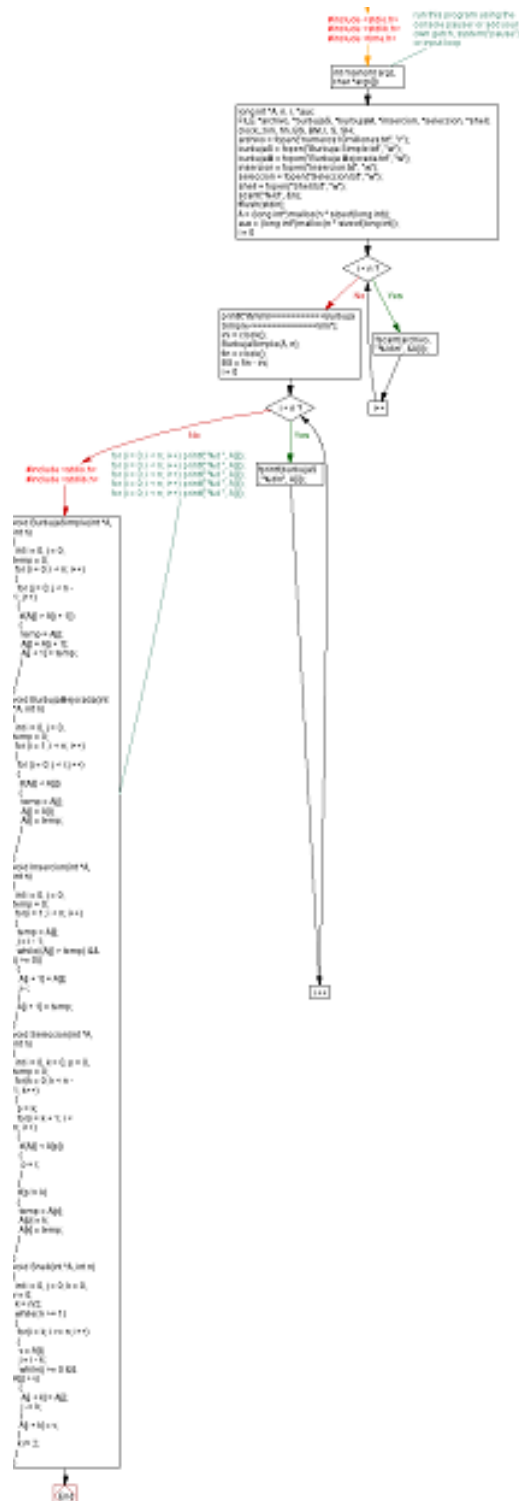


Diagrama 1.2 Práctica3.c

Implementación de la solución

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. /*
5.  PRACTICA 3
6.  ANALISIS DE ALGORITMOS DE ORDENAMIENTO
7.  . ALGORITMOS
8.  PROFRA LUZ MARIA
9.  LUIS ALEJANDRO DE LOS SANTOS
10. MARCOS OSWALDO VAZQUEZ MORENO
11. ESCUELA SUPERIOR DE COMPUTO
12.
13. */
14.
15. int main(int argc, char *argv[]) {
16.     long int *A, n, i, *aux;
17.     FILE *archivo, *burbujaS, *burbujaM, *insercion, *seleccion, *shell;
18.     clock_t ini, fin, BS, BM, I, S, SH;
19.     archivo = fopen("numeros10millones.txt", "r");
20.     burbujaS = fopen("Burbuja Simple.txt", "w");
21.     burbujaM = fopen("Burbuja Mejorada.txt", "w");
22.     insercion = fopen("Insercion.txt", "w");
23.     seleccion = fopen("Seleccion.txt", "w");
24.     shell = fopen("Shell.txt", "w");
25.     scanf("%ld", &n);
26.     fflush(stdin);
27.     A = (long int*)malloc(n * sizeof(long int));
28.     aux = (long int*)malloc(n * sizeof(long int));
29.     for (i = 0; i < n; i++)
30.     {
31.         fscanf(archivo, "%ld\n", &A[i]);
32.     }
33.     printf("\t\n\n\n=====>Burbuja Simple<=====\\n\\n");
34.     ini = clock();
35.     BurbujaSimple(A, n);
36.     fin = clock();
37.     BS = fin - ini;
38.     for (i = 0; i < n; i++) fprintf(burbujaS, "%d\\n", A[i]);
39.     printf("\t\n\n\n=====>Burbuja Mejorada<=====\\n\\n");
40.     ini = clock();
41.     BurbujaMejorada(A, n);
42.     fin = clock();
43.     BM = fin - ini;
44.     for (i = 0; i < n; i++) fprintf(burbujaM, "%d\\n", A[i]);
45.     printf("\t\n\n\n=====>Insercion<=====\\n\\n");
46.     ini = clock();
47.     Insercion(A, n);
48.     fin = clock();
49.     I = fin - ini;
50.     for (i = 0; i < n; i++) fprintf(insercion, "%d\\n", A[i]);
51.     printf("\t\n\n\n=====>Seleccion<=====\\n\\n");
52.     ini = clock();
53.     Seleccion(A, n);
54.     fin = clock();
55.     S = fin - ini;
56.     for (i = 0; i < n; i++) fprintf(seleccion, "%d\\n", A[i]);
57.     printf("\t\n\n\n=====>Shell<=====\\n\\n");
58.     ini = clock();
59.     Shell(A, n);
```

```

60.     fin = clock();
61.     SH = fin - ini;
62.     for (i = 0; i < n; i++) fprintf(shell, "%d\n", A[i]);
63.     printf("\n\nTiempos de ejecucion\n");
64.     printf("Burbuja Simple: %f\n", (double)BS/CLOCKS_PER_SEC);
65.     printf("Burbuja Mejorada: %f\n", (double)BM/CLOCKS_PER_SEC);
66.     printf("Insercion: %f\n", (double)I/CLOCKS_PER_SEC);
67.     printf("Seleccion: %f\n", (double)S/CLOCKS_PER_SEC);
68.     printf("Shell: %f\n", (double)SH/CLOCKS_PER_SEC);
69.     fclose(archivo);
70.     fclose(burbujaS);
71.     fclose(burbujaM);
72.     fclose(insercion);
73.     fclose(seleccion);
74.     fclose(shell);
75.     return 0;
76. }

```

Práctica3.c

```

1. #include <stdio.h>
2. #include <stdlib.h>
3. /*
4.     PRACTICA 3
5.     ANALISIS DE ALGORITMOS DE ORDENAMIENTO
6.     .   ALGORITMOS
7.     PROFRA LUZ MARIA
8.     LUIS ALEJANDRO DE LOS SANTOS
9.     MARCOS OSWALDO VAZQUEZ MORENO
10.    ESCUELA SUPERIOR DE COMPUTO
11.    */
12.
13. void BurbujaSimple(int *A, int n)
14. {
15.     int i = 0, j = 0, temp = 0;
16.     for (i = 0; i < n; i++)
17.     {
18.         for (j = 0; j < n - 1; j++)
19.         {
20.             if(A[j] > A[j + 1])
21.             {
22.                 temp = A[j];
23.                 A[j] = A[j + 1];
24.                 A[j + 1] = temp;
25.             }
26.         }
27.     }
28.     //for (i = 0; i < n; i++) printf("%d ", A[i]);
29. }
30.
31. void BurbujaMejorada(int *A, int n)
32. {
33.     int i = 0, j = 0, temp = 0;
34.     for (i = 1; i < n; i++)
35.     {
36.         for (j = 0; j < i; j++)
37.         {
38.             if(A[i] < A[j])
39.             {
40.                 temp = A[j];
41.                 A[j] = A[i];

```

```

42.         A[i] = temp;
43.     }
44. }
45. }
46. //for (i = 0; i < n; i++) printf("%d ", A[i]);
47. }
48.
49. void Insercion(int *A, int n)
50. {
51.     int i = 0, j = 0, temp = 0;
52.     for(i = 1; i < n; i++)
53.     {
54.         temp = A[i];
55.         j = i - 1;
56.         while((A[j] > temp) && (j >= 0))
57.         {
58.             A[j + 1] = A[j];
59.             j--;
60.         }
61.         A[j + 1] = temp;
62.     }
63.     //for (i = 0; i < n; i++) printf("%d ", A[i]);
64. }
65.
66. void Seleccion(int *A, int n)
67. {
68.     int i = 0, k = 0, p = 0, temp = 0;
69.     for(k = 0; k < n - 1; k++)
70.     {
71.         p = k;
72.         for(i = k + 1; i < n; i++)
73.         {
74.             if(A[i] < A[p])
75.             {
76.                 p = i;
77.             }
78.         }
79.         if(p != k)
80.         {
81.             temp = A[p];
82.             A[p] = k;
83.             A[k] = temp;
84.         }
85.     }
86. }
87. //for (i = 0; i < n; i++) printf("%d ", A[i]);
88. }
89.
90. void Shell(int *A, int n)
91. {
92.     int i = 0, j = 0, k = 0, v = 0;
93.     k = n/2;
94.     while( k >= 1)
95.     {
96.         for(i = k; i <= n; i++)
97.         {
98.             v = A[i];
99.             j = i - k;
100.            while(j >= 0 && A[j] > v)
101.            {
102.                A[j + k] = A[j];
103.                j -= k;

```



```

104.         }
105.         A[j + k] = v;
106.     }
107.     k /= 2;
108. }
109. // for (i = 0; i < n; i++) printf("%d ", A[i]);
110. }
111. int BinarySearch(int Data[], int DataSize, int NumberToSearch) { //===
    BINARY SEARCH =====
112.     int Initial = 0, Final = DataSize; //Vari
    ables that we need
113.
114.     while (Initial <= Final) { //Whil
    e find make sense
115.
116.         int Middle = Initial + ((Final - Initial) / 2); //Find
        a new SearchPosition
117.
118.         if (Data[Middle] == NumberToSearch) //If a
        ll ok!
119.             return Middle; //If w
        e find it!
120.         else if (Data[Middle] > NumberToSearch) //If w
        e need to go to side
121.             Final = Middle - 1; //Find
            the new final position
122.         else //If w
        e need to go to side
123.             Initial = Middle + 1; //Find
            the new initial position
124.     }
125.
126.     return -1;

```

//SEGUNDA SOLUCION

```

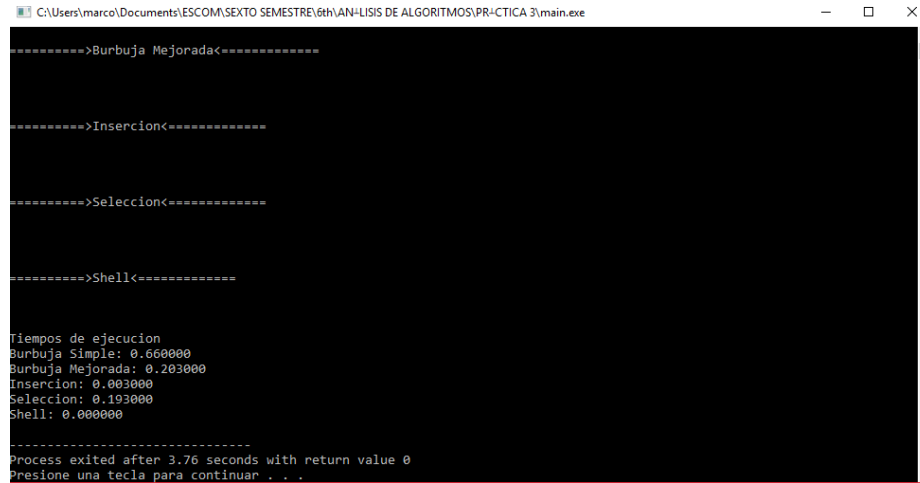
} void ArbolBinario (int *numeros, int n)
{
    uswtime(&utime0, &stime0, &wtime0);
    //Comenzamos a tomar el tiempo del algoritmo
    numeros = InsertarABB(numeros, n);
    uswtime(&utime1, &stime1, &wtime1);
    //Terminamos de tomar el tiempo del algoritmo

    //Ordenamos y mostramos los numeros ordenados por el algoritmo
    calculaTiempo (utime0, stime0, wtime0, utime1, stime1, wtime1, n, 6);
    //Calculamos el tiempo del algoritmo y lo mostramos
    numerosOrdenados (numeros, n);
}

```

Funcionamiento

Es importante mencionar que mostramos las salidas en archivos, ya que es de la misma manera que lo estamos recibiendo y es importante para nosotros llevar un orden, sin embargo, cuando lo realiza hacemos una impresión en consola como se muestra en la imagen 2.1 (anexando capturas de pantalla de los archivos .txt generados)



```
=====>Burbuja Mejorada<=====

=====>Insercion<=====

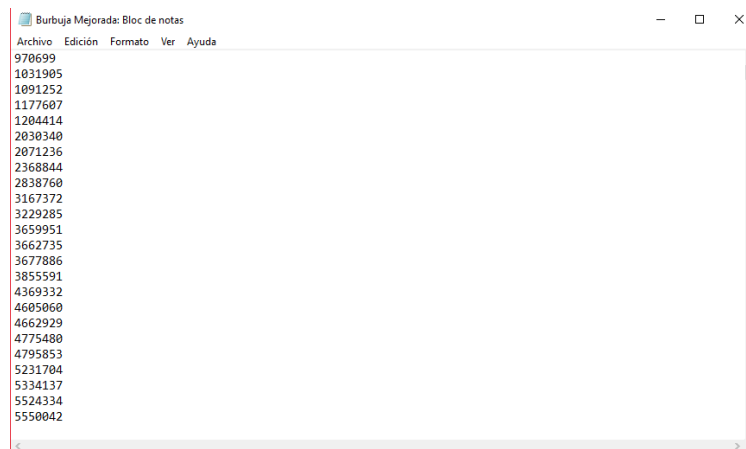
=====>Seleccion<=====

=====>Shell<=====

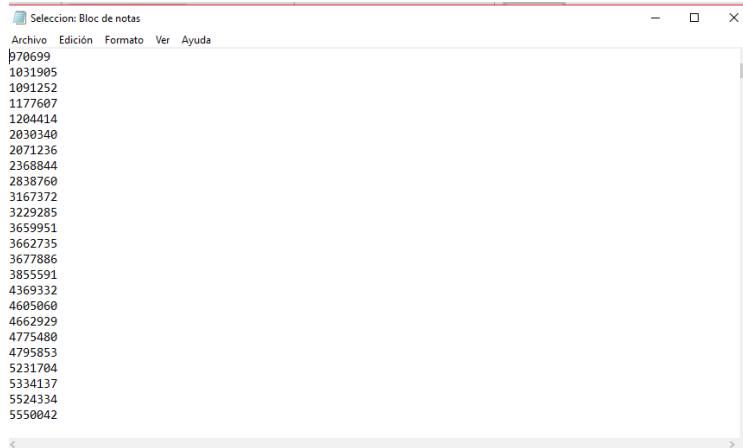
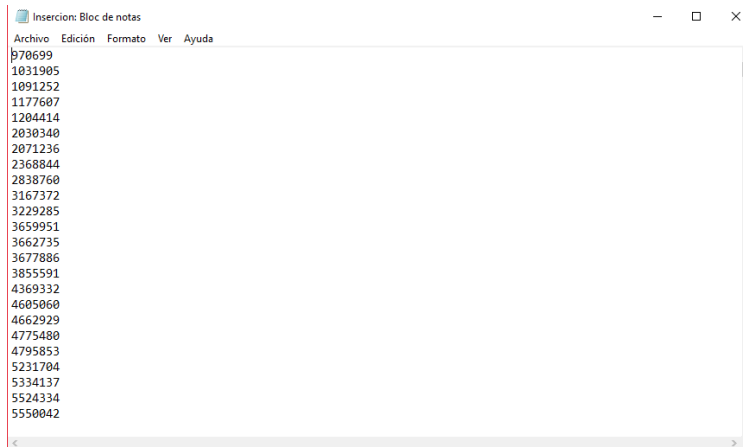
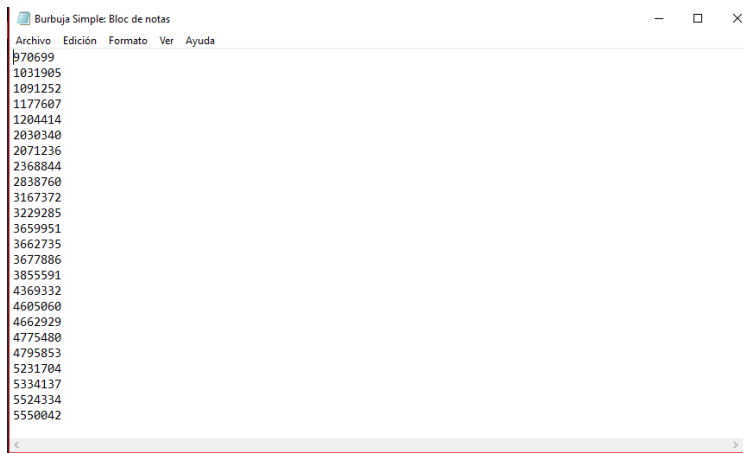
Tiempos de ejecucion
Burbuja Simple: 0.660000
Burbuja Mejorada: 0.203000
Insercion: 0.003000
Seleccion: 0.193000
Shell: 0.000000

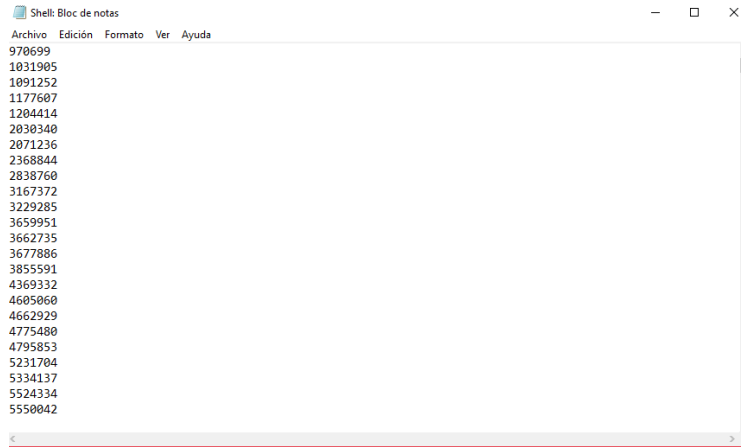
Process exited after 3.76 seconds with return value 0
Presione una tecla para continuar . . .
```

Imagen 2.1



```
Burbuja Mejorada: Bloc de notas
Archivo Edición Formato Ver Ayuda
970699
1031905
1091252
1177607
1204414
2030340
2071236
2368844
2838760
3167372
3229285
3659951
3662735
3677886
3855591
4369332
4605060
4662929
4775480
4795853
5231704
5334137
5524334
5550042
```





Con $n = 10,000$ mostrando exclusivamente la comparación de tiempos de ejecución de cada uno en la imagen 2.2

```
Tiempos de ejecucion
Burbuja Simple: 0.660000
Burbuja Mejorada: 0.203000
Insercion: 0.003000
Seleccion: 0.193000
Shell: 0.000000

-----
Process exited after 3.76 seconds with return value 0
Presione una tecla para continuar . . .
```

Imagen 2.2 Comparación de tiempos.

El peor caso ocurre cuando A está ordenado descendientemente. Tenemos dos ciclos for anidados, el externo se repite siempre $n-1$ veces y el interno se repite a lo más $n-1$ veces también. Dentro del for interno hay 4 operaciones constantes. Por lo tanto, la complejidad de los algoritmos es $O(n^2)$.

Plataforma experimental

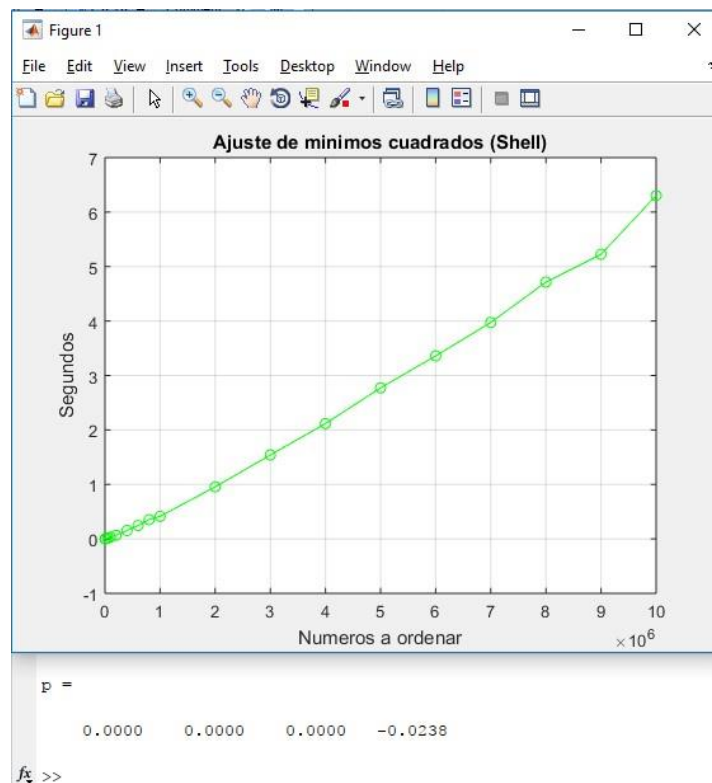
La ejecución de los algoritmos anteriores se llevó a cabo en una computadora personal que se describe en la siguiente imagen.

Especificaciones del dispositivo	
HP Laptop 15-bs0xx	
Nombre del dispositivo	LAPTOP-13V7QO38
Procesador	Intel(R) Core(TM) i3-6006U CPU @ 2.00GHz 2.00 GHz
RAM instalada	8.00 GB
Id. del dispositivo	ACEA8FAF-1F85-49D4-BC5D-A7059ECE254D
Id. del producto	00327-30000-00000-AAOEM
Tipo de sistema	Sistema operativo de 64 bits, procesador x64
Lápiz y entrada táctil	La entrada táctil o manuscrita no está disponible para esta pantalla

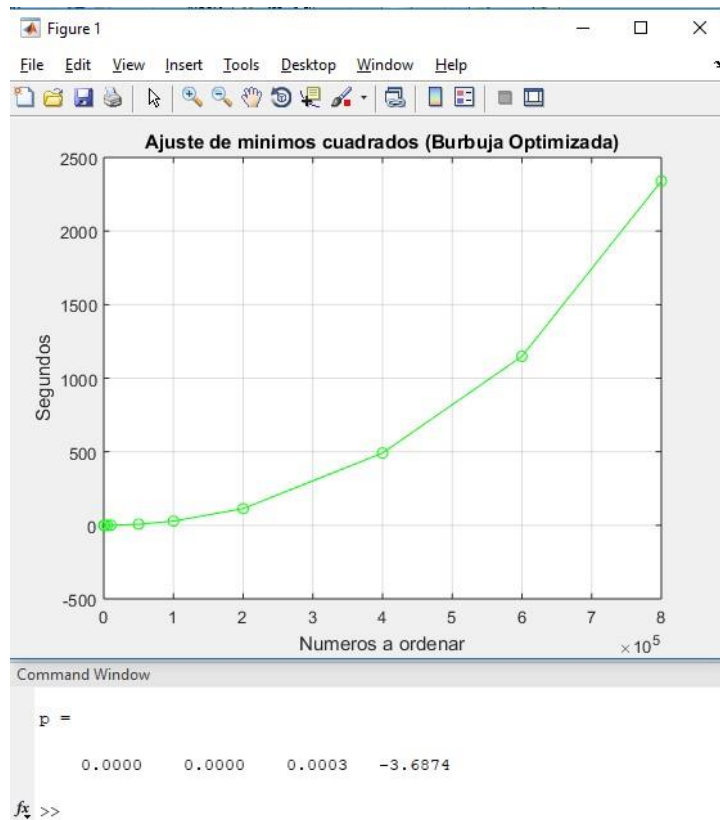
Imagen 3.1 Plataforma Experimental

El compilador utilizado fue gcc integrado dentro del IDE DevC en un sistema operativo de 64 bits Windows 10.

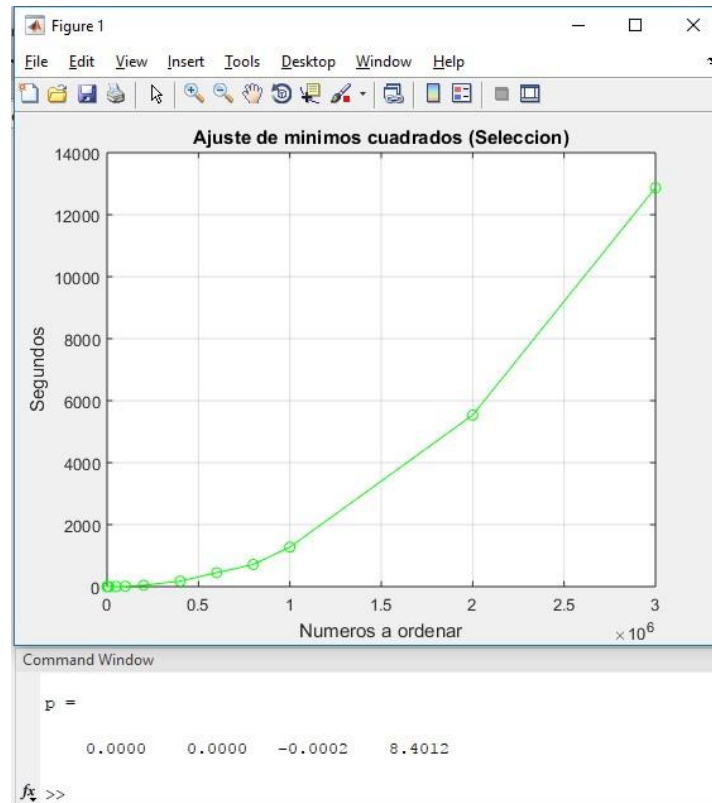
Gráficas de funciones



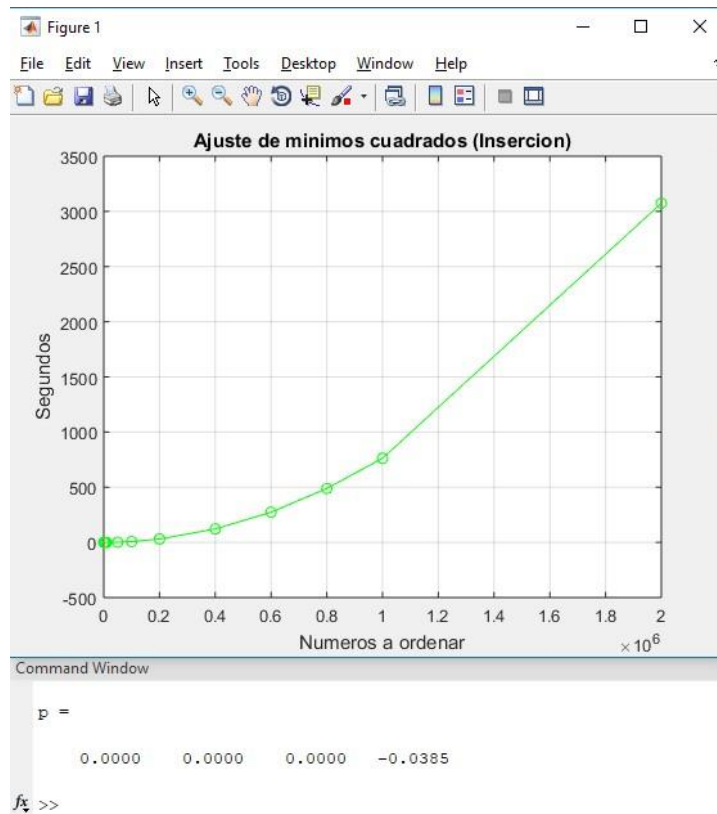
Gráfica 3.1



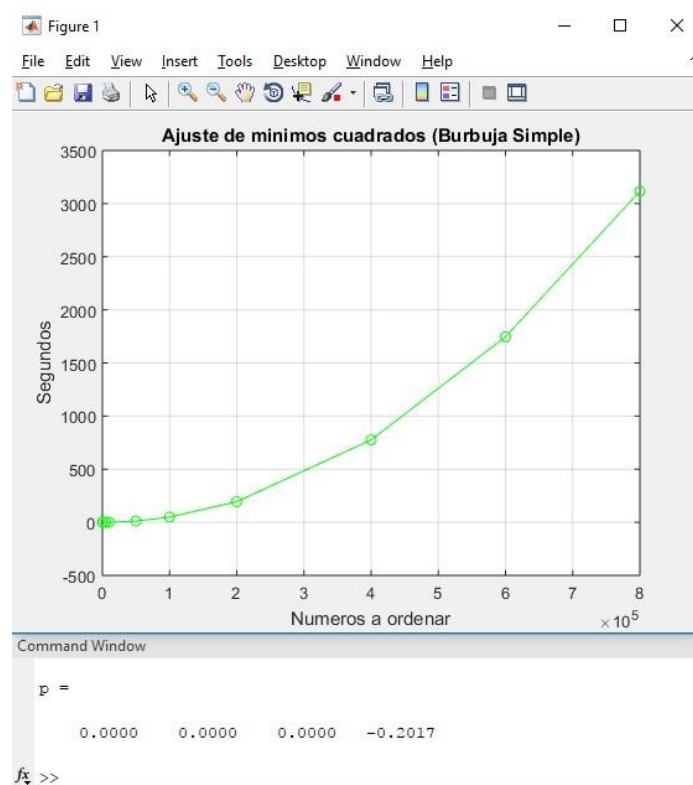
Gráfica 3.2



Gráfica 3.3



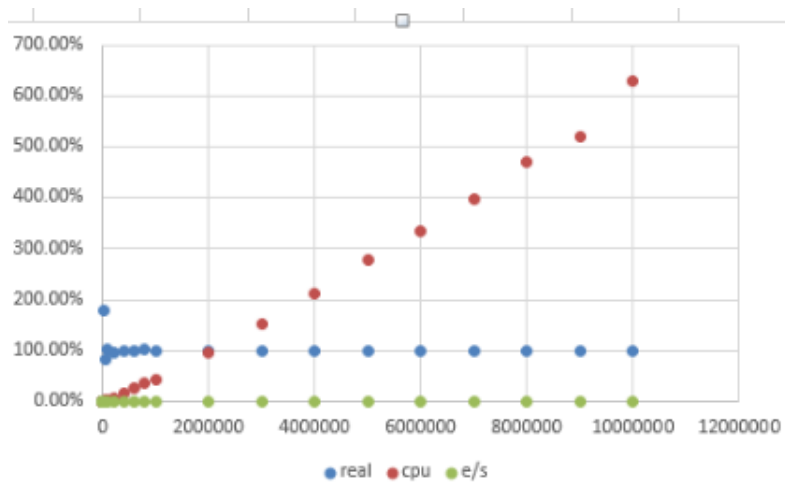
Gráfica 3.4



Gráfica 3.5

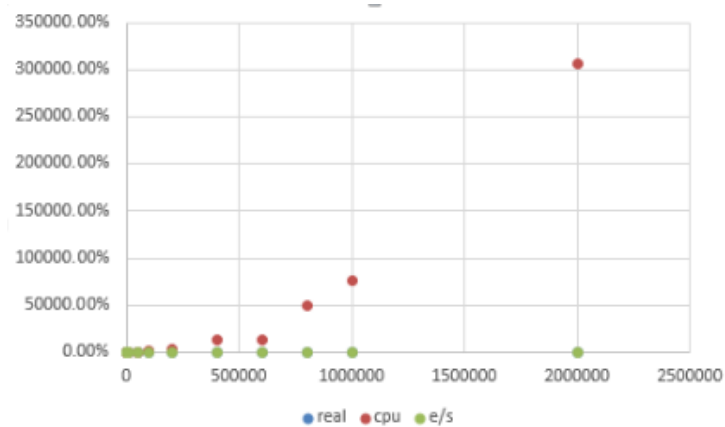
Graficas de tiempos con muestras

Selección



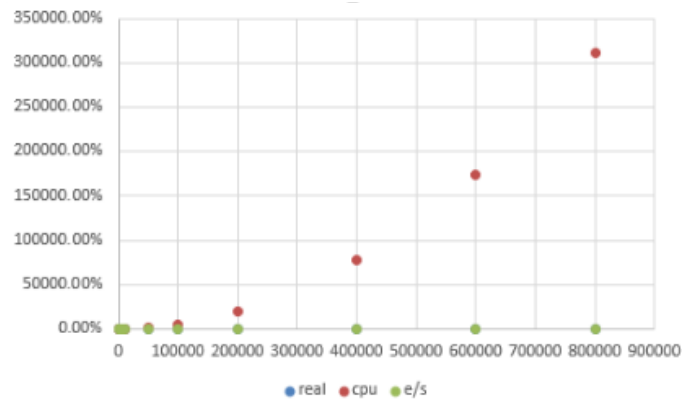
Gráfica 3.6

Burbuja Optimizada



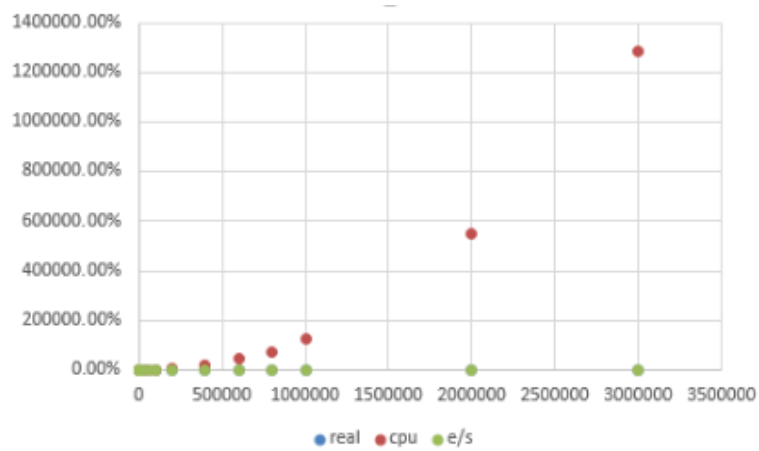
Gráfica 3.7

Shell



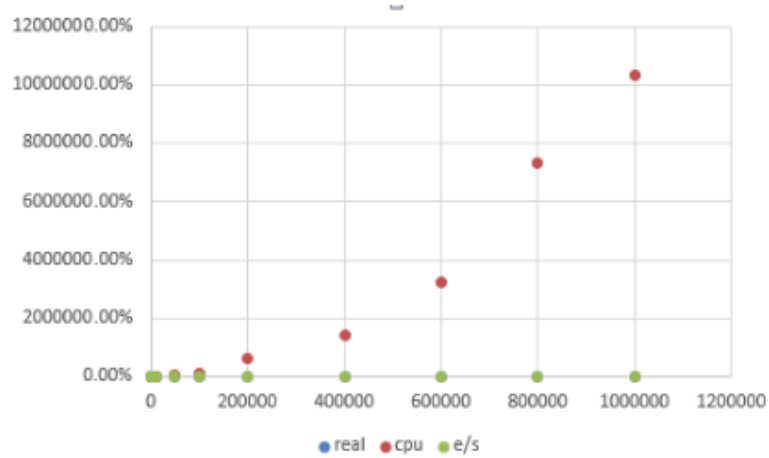
Gráfica 3.8

Insercion



Gráfica 3.9

Árbol de Búsqueda



Gráfica 3.10

Conclusiones

En conclusión, nos dimos cuenta de lo importante que son los algoritmos de ordenamiento, ya que no es la primera vez que utilizamos uno ni mucho menos la última, recordamos nuestro curso de arquitectura de computadoras en donde un examen venía con el algoritmo de ordenamiento de burbuja (bubble sort) a nivel ensamblador, mismo que costaba un grado más de complejidad puesto que se tenían que hacer llamados a subrutinas; tomemos en cuenta también que en anteriores cursos vimos algoritmos como por ejemplo árboles binarios de búsqueda, que sin duda es un tipo de estructura de dato muy complicado de implementar y sobre todo el más complicado para nosotros desarrollar **sin recursividad**, dicho lo anterior nos dimos cuenta de que no fue lo único difícil para nosotros ya que nos quebramos la cabeza intentando lograr que el árbol binario de búsqueda lograra su cometido, sin embargo, no lo logramos. No logramos que nuestro árbol binario de búsqueda nos funcionara adecuadamente.

Por otro lado, nos quedamos con un buen sabor de boca sabiendo que logramos sin duda el objetivo de la práctica, pues nos dimos cuenta de la importancia de los algoritmos de búsqueda incluso en un archivo de 10 millones de números mismo que agregamos a la ejecución desde un archivo .txt.

Por último, queremos mencionar que es extremadamente tardado el tiempo de ordenamiento con 5 millones de números por lo que el de 10 millones lo dejamos alrededor de 3 horas y decidimos detenerlo porque además de casi consumir la memoria completa de mi PC no estaba terminando pronto y eso nos imposibilitaba demasiado el poder seguir ejecutando los demás algoritmos y a su vez haciendo pruebas.

Bibliografía

Bibliografía

- Gimeno, J., & González, J. (s.f.). *ocw.udl.cat*. Recuperado el 2017, de <http://ocw.udl.cat/enginyeria-i-arquitectura/programacio-2/continguts-1/2-recursividad.pdf>
- Gurin, S. (Septiembre de 2004). *Algoritmos de ordenación*. Obtenido de Algoritmos de ordenación: http://es.tldp.org/Tutoriales/doc-programacion-algoritmos-ordenacion/alg_orden.pdf
- Práctica3_AlgoritmosOrdenamiento. (09 de Marzo de 2019). *Classroom*. Obtenido de Classroom: <https://classroom.google.com/u/0/c/NzEzNTUxMDU1NVpa/a/NzYwMTgwNzg2NFpa/details>