ESTRUCTURA DE DATOS

Álvaro Fernández García, grupo A1.

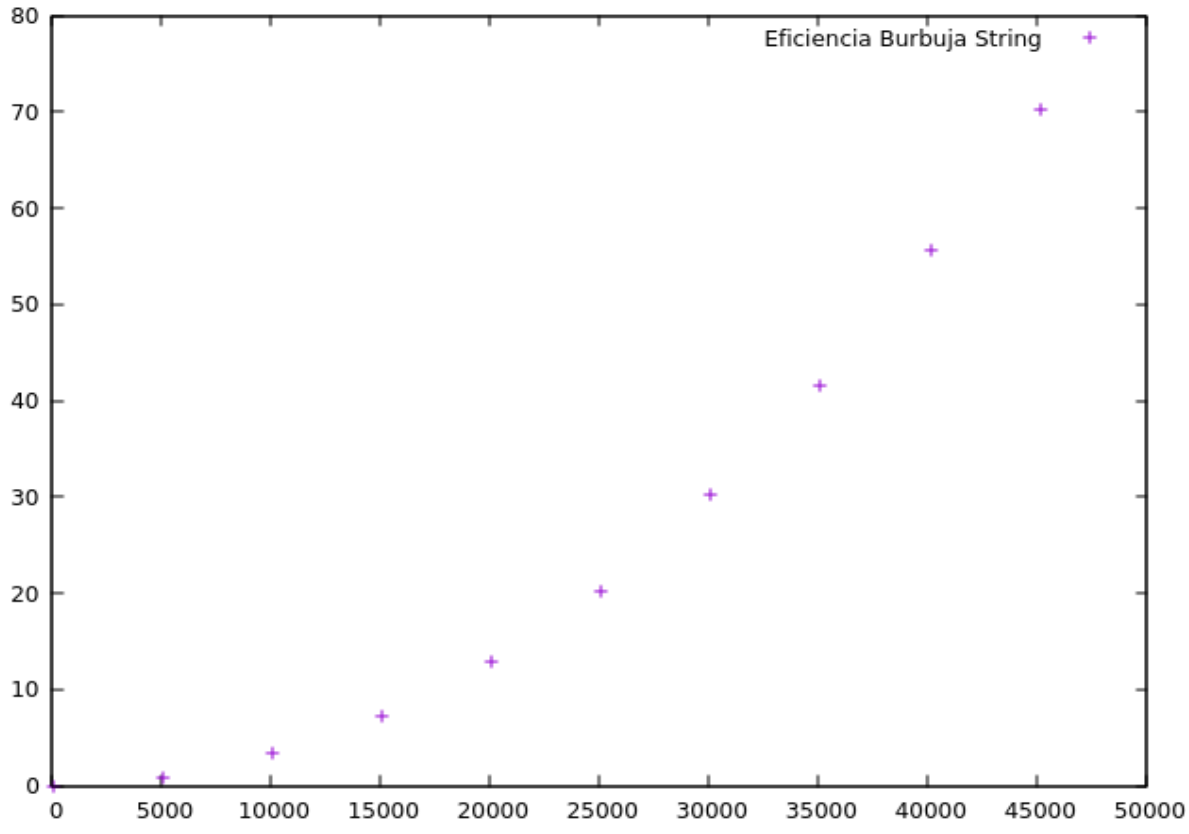
-PRÁCTICA 1: EFICIENCIA DE LOS ALGORITMOS-

Apartado 6.1: Ejemplo 1: Algoritmo de ordenación por burbuja.

Como ya hemos comprobado durante la práctica, el análisis teórico del algoritmo de la burbuja presenta un orden cuadrático, así que pasaremos a realizar el análisis práctico.
Una vez realizado el programa para la ordenación de string mediante el método de la burbuja y haberlo ejecutado para realizar cálculos de 100 a 50.000 elementos incrementando de 5.000 en 5.000 obtenemos la siguiente tabla:

Entrada	Tiempo
100	0.000382778
5100	0.918352
10100	3.3409
15100	7.23221
20100	12.8262
25100	20.2014
30100	30.2469
35100	41.514
40100	55.551
45100	70.2358

A continuación, representamos los datos obtenidos:



Realizaremos la regresión cuadrática con la fórmula:

```
f(x) = a*x*x + b*x + c
```

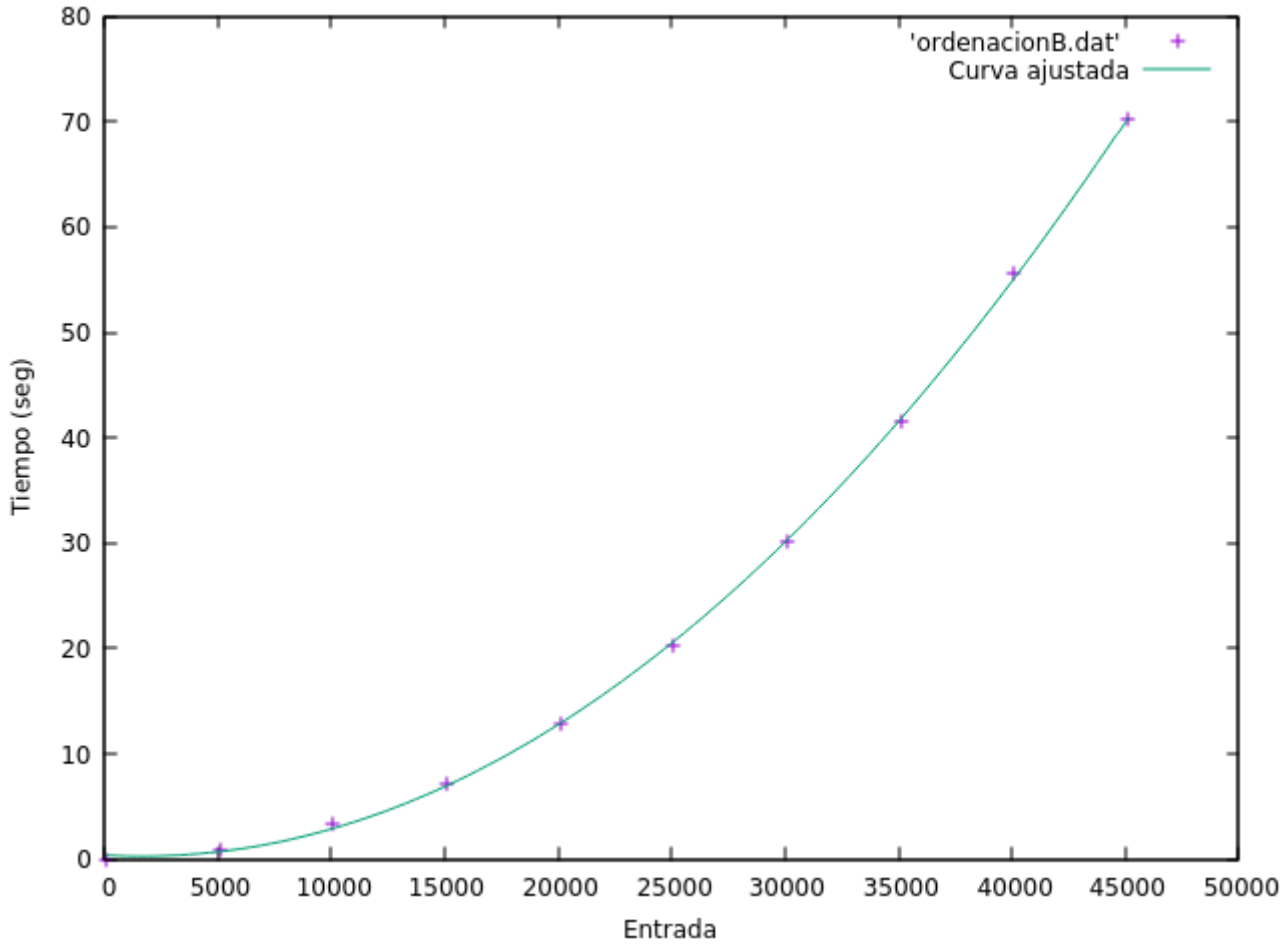
Procederemos a continuación a calcular las constantes ocultas a y b.
En la salida observamos en la sección “Final Set of Parameters” los valores:

```
a = 3.71141e-08
b = -0.000123822
c = 0.393904
```

Por tanto la función quedaría en este caso como:

```
f(x) = 3.71141e-08*x*x - 0.000123822*x + 0.393904
```

La representación quedaría de la forma:



Apartado 6.2

1.) Realizar el análisis de eficiencia cuando consideramos el código en ocurrencias.cpp. En este caso, debemos de modificar el código para asegurarnos que tenemos la misma salida que en el ejemplo del algoritmo burbuja (tamaño y tiempo).

Comenzamos realizando el análisis teórico. La parte del algoritmo que nos interesa medir es la siguiente:

```
int contar_hasta( vector<string> & V, int ini, int fin,
string & s) {
  int cuantos = 0;
  for (int i=ini; i< fin ; i++)
    if (V[i]==s) {
      cuantos ++;
    }
  return cuantos;
}
```

El tiempo de Iteración (It) pertenecen al orden constante puesto que tanto el cuerpo del bucle, como la comprobación y el incremento son operaciones elementales. Por otra parte el bucle se repetira fin – ini veces, lo cual dependerá de la entrada seleccionada, llamémoslo n, en definitiva obtenemos:

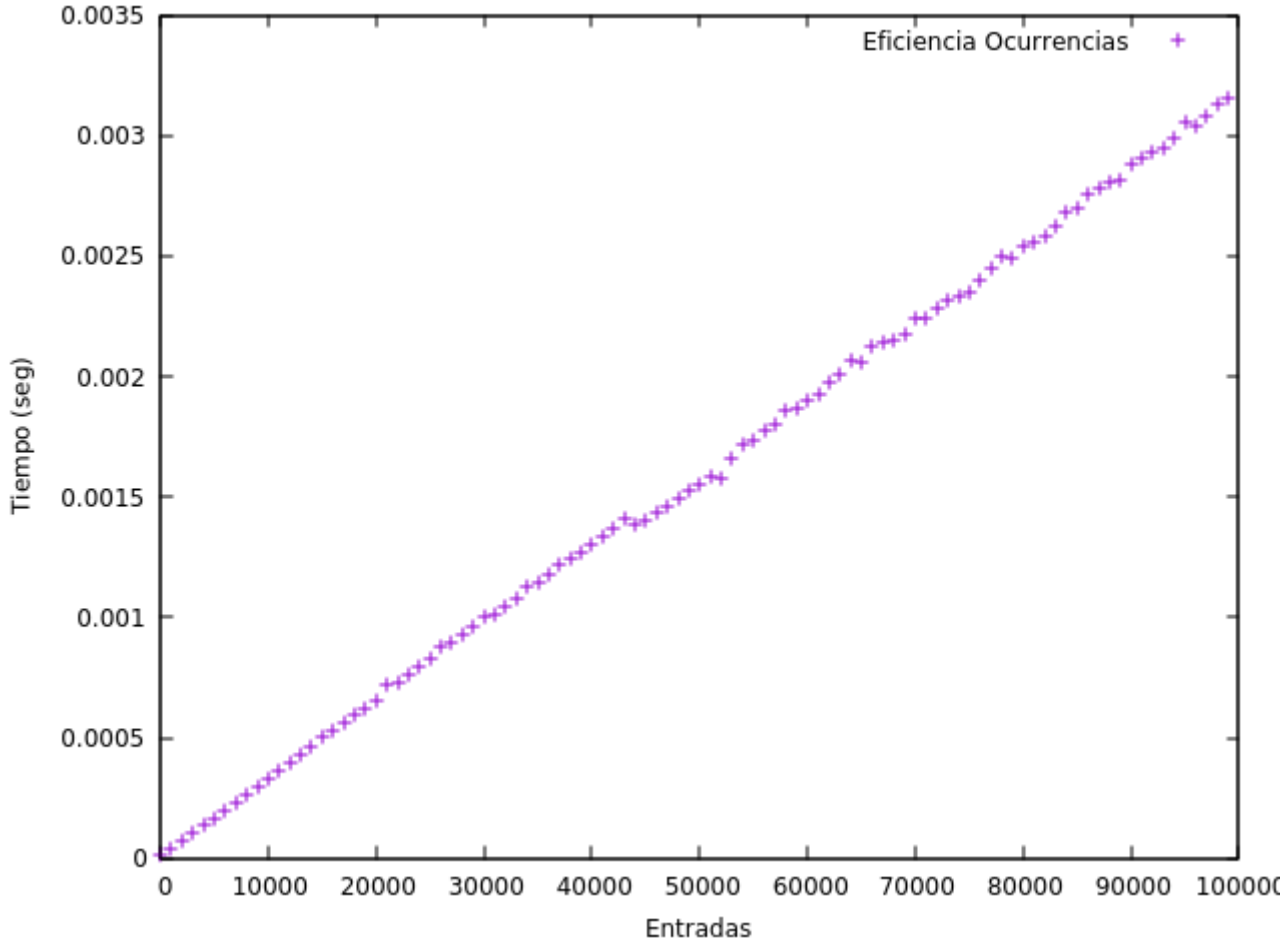
$$\Sigma(i=1 \text{ a } n)\ 1 = n \in O(n)$$

Por tanto vemos que nuestro algoritmo tiene órden n.
Una vez modificado el código, pasaremos al análisis práctico. Aquí aparece una parte de la extensa tabla de tiempos:

Entrada	**Tiempo**
10	1.4384e-05
1010	3.8271e-05
2010	7.3634e-05
3010	0.000103927
4010	0.000135866
5010	0.000167622
6010	0.000199512
7010	0.000232489
8010	0.000265829
9010	0.000299381
10010	0.000332843
11010	0.000365117
…	
90010	0.0028795
91010	0.00290628
92010	0.0029278
93010	0.00294493

```
94010     0.00299064
95010     0.00305464
96010     0.00303944
97010     0.00308121
98010     0.0031307
99010     0.00315678
```

Una vez realizado esto pasamos a representar los datos con GnuPlot, obteniendo la siguiente salida:



Podemos comprobar que sigue un orden lineal, realizaremos el ajuste con la siguiente función:

```
f(x) = a*x+b
```

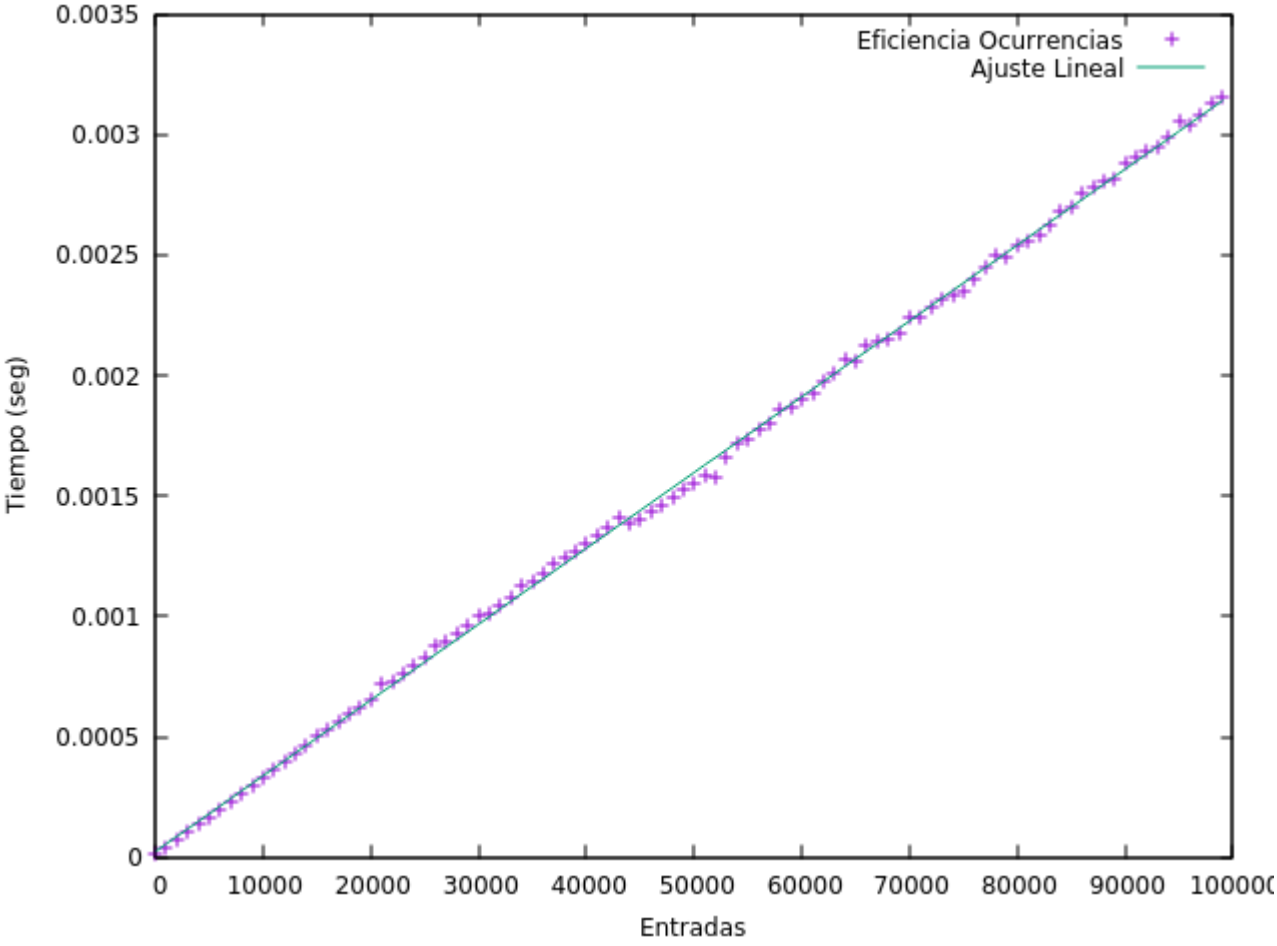
Procederemos a continuación a calcular las constantes ocultas a y b.
En la salida observamos en la sección “Final Set of Parameters” los valores:

```
a = 3.14823e-08
b = 2.28014e-05
```

Por tanto, podemos decir que nuestro algoritmo responde a la función:

```
T(n) = 3.14823e-08 * n +  2.28014e-05
```

Se adjunta el resultado de GnuPlot tras la regresión:



El procedimiento para realizar el análisis de la eficiencia de un algoritmo es siempre el mismo, por tanto en ejercicios posteriores se proporcionará una información más reducida.

2.) Análisis de eficiencia del código en frecuencias.cpp, considerando como entradas el texto del fichero quijote.txt.

Versión 1:

Análisis teórico:

Tiempo de iteración: $\Sigma(i=1 \text{ a } n)\ 1 = n \in O(n)$ //Función "contar hasta".

$\Sigma(i=1 \text{ a } n)\ n = [n(n+1)]/2 \in O(n^2)$

Tabla:

Entrada	**Tiempo**
10	5.5223e-05
110	0.000452139
210	0.00152731
310	0.003353
410	0.00581742
510	0.00897057
610	0.0128007
710	0.0173157
810	0.0224186
910	0.0282514
1010	0.0347281

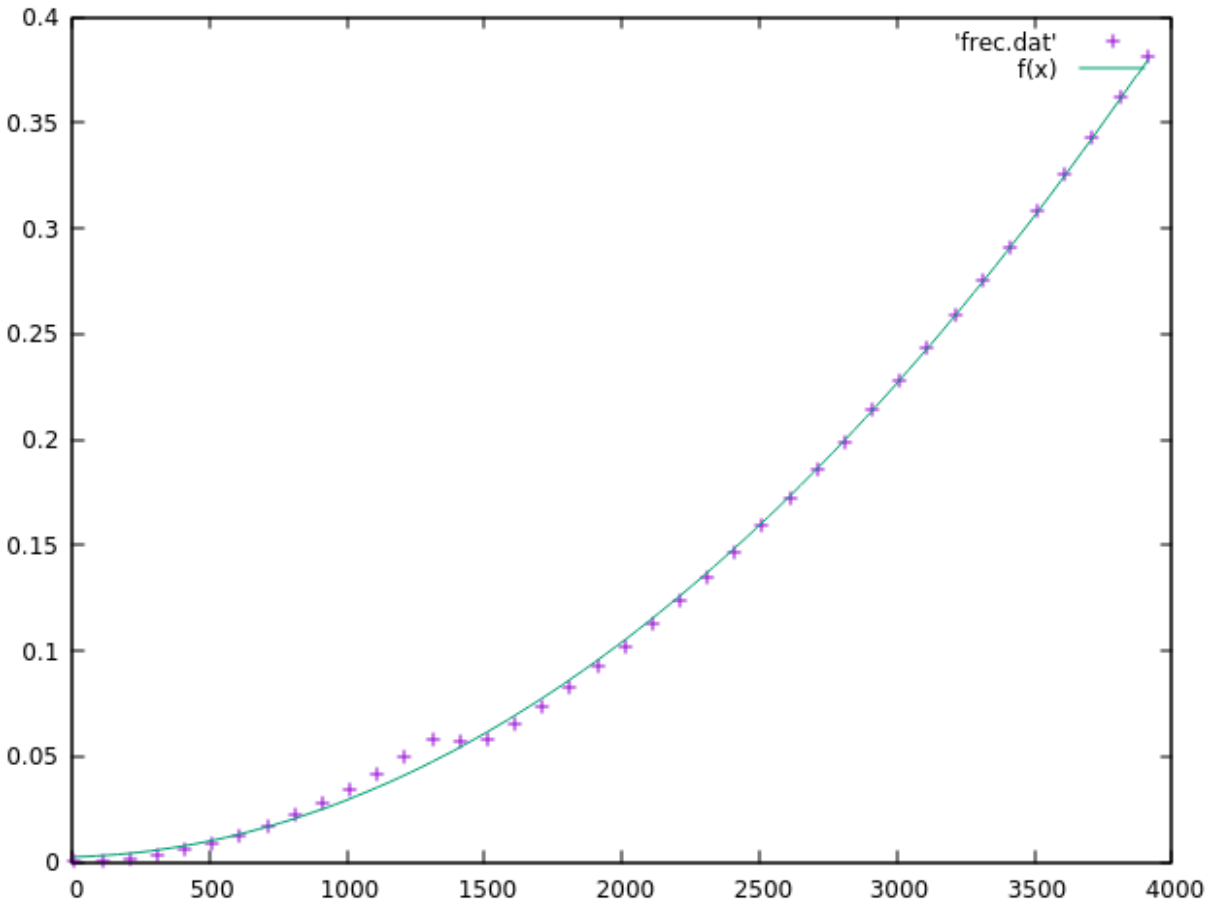
…

Fórmula empelada para el ajuste:

```
f(x) = a*x*x + b*x +c
```

Fórmula tras el ajuste:

```
f(x) = 2.39045e-08*x*x + 3.0518e-06*x + 0.00238689
```



Versión 2:

Análisis teórico:

Tiempo de iteración: $\Sigma(i=1 \text{ a } n)\ 1 = n \in O(n)$ //Función “buscar”.

$\Sigma(i=1 \text{ a } n)\ n = [n(n+1)]/2 \in O(n^2)$

Tabla:

Entrada	Tiempo
0	7.73e-07
100	0.00021291
200	0.000485841
300	0.000961788
400	0.00154114
500	0.00223875
600	0.00290533
700	0.00366778
800	0.00444224
900	0.00525847
1000	0.0060885
1100	0.00697327

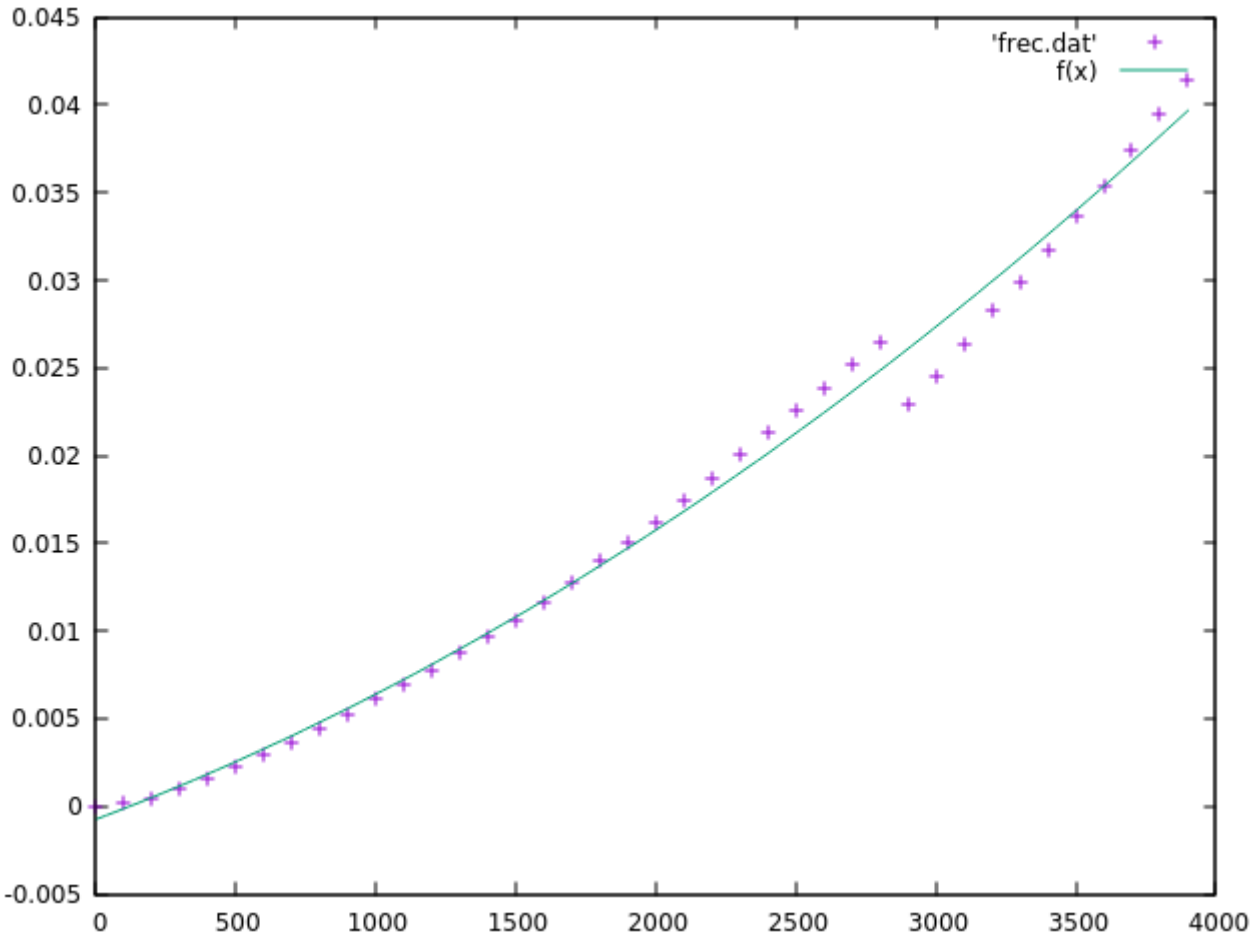
...

Fórmula empelada para el ajuste:

```
f(x) = a*x*x + b*x +c
```

Fórmula tras el ajuste:

```
f(x) = 1.11059e-09*x*x + 6.03447e-06*x - 0.000766335
```



Versión 3:

Análisis teórico:

Tiempo de iteración: O(n) // A pesar de tener un órden logarítmico, en un bloque de intrucciones siempre predomina el peor de los casos.

$$\Sigma(i=1 \text{ a } n)\ n = [n(n+1)]/2 \in O(n^2)$$

Tabla:

Entrada	**Tiempo**
0	8.41e-07
100	0.000290659
200	0.000478274
300	0.000715178
400	0.000978848
500	0.00129143
600	0.00156098
700	0.00185834
800	0.00216002
900	0.00245682
1000	0.00279184
1100	0.00318259

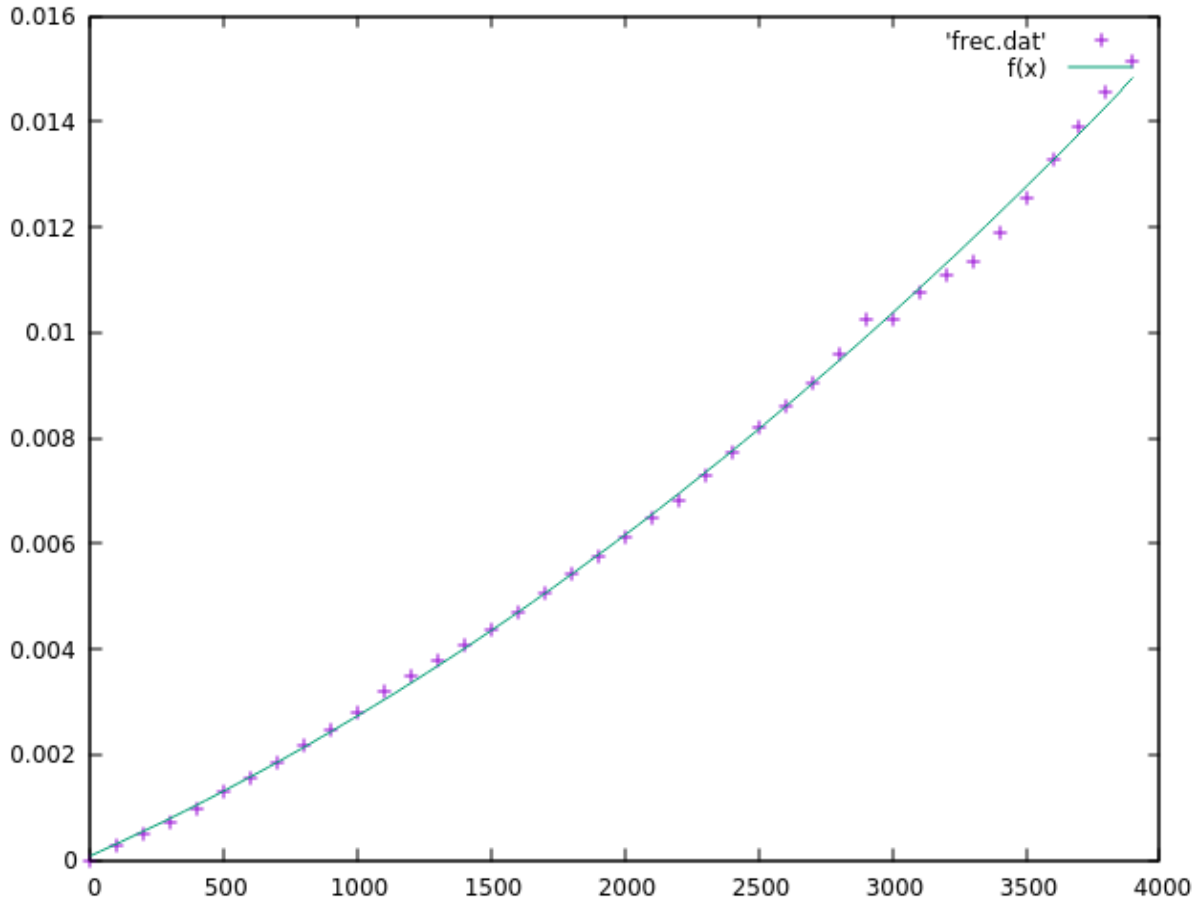
...

Fórmula empelada para el ajuste:

```
f(x) = a*x*x + b*x +c
```

Fórmula tras el ajuste:

```
f(x) = 3.88216e-10*x*x + 2.26596e-06*x + 7.39347e-05
```



Versión 4:

Análisis teórico:

El peor de los casos es que, en el segundo bucle, todas las palabras sean distintas, es decir, k = n, por tanto el orden de este algoritmo es O(n*log(n))

Tabla:

Entrada	Tiempo
0	2.842e-06
100	0.000259096
200	0.000371551
300	0.000530343
400	0.000729523
500	0.000875849
600	0.0010088
700	0.00117675
800	0.00135118
900	0.00149474
1000	0.00165219
1100	0.00183135

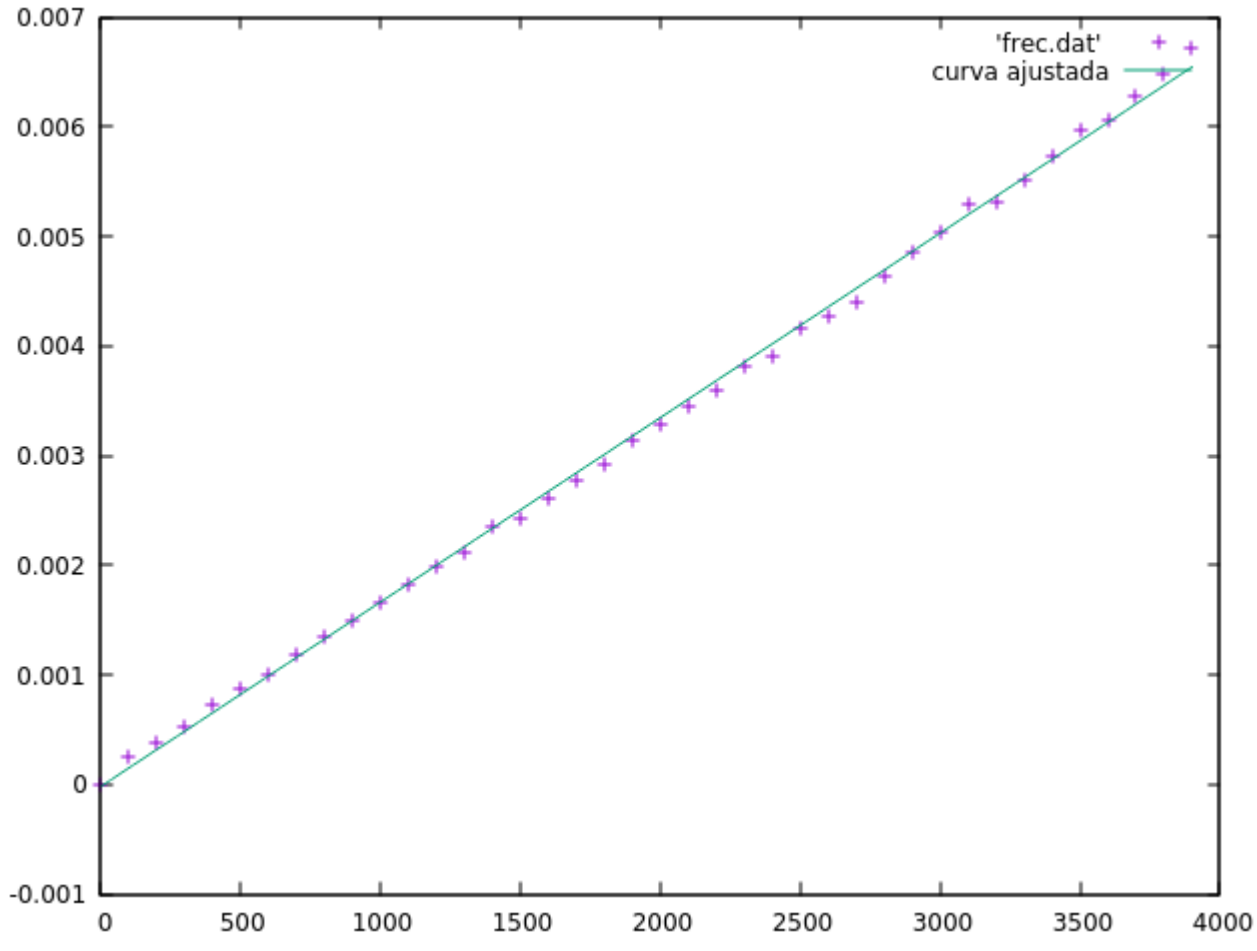
...

Fórmula empelada para el ajuste:

```
f(x) = a*x +b //A pesar del orden utilizaremos esta
                fórmula para el ajuste
```

Fórmula tras el ajuste:

```
f(x) = 1.6841e-06*x - 2.44699e-05
```



3.) Realizar el análisis de eficiencia teórico y práctico con los algoritmos de ordenación que se conocen (burbuja, inserción y selección). El alumno deberá implementar los distintos algoritmos de ordenación de forma que se permitan obtener los tiempos para distintos tamaños de entrada.

BURBUJA

Análisis teórico:

Tiempo de iteración: $\Sigma(i=1 \text{ a } n)\ 1 = n \in O(n)$

$\Sigma(i=1 \text{ a } n)\ n = [n(n+1)]/2 \in O(n^2)$

Algoritmo ejecutado con "quijote.txt":

Tabla:

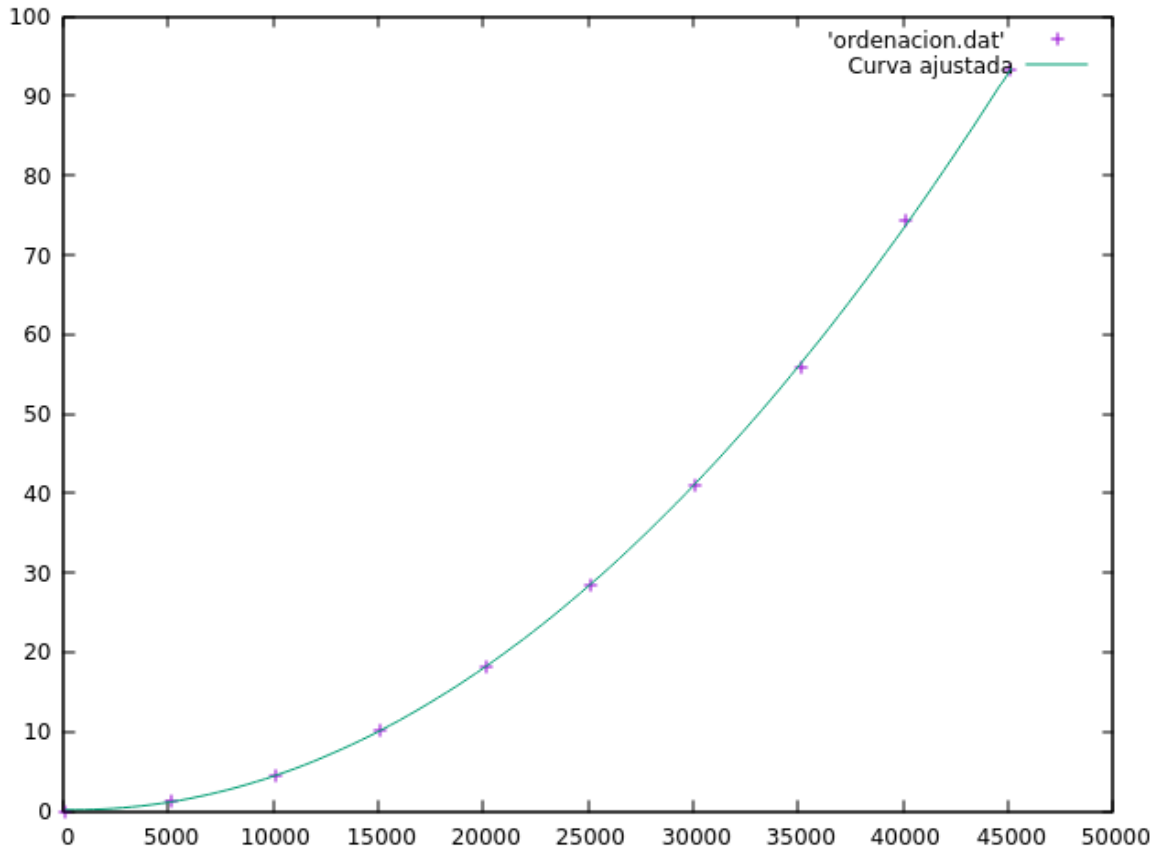
Entrada	Tiempo
100	0.000734004
5100	1.28067
10100	4.57461
15100	10.1673
20100	18.2391
25100	28.4343
30100	41.0376
35100	55.9487
40100	74.2389
45100	93.193

Fórmula empelada para el ajuste:

```
f(x) = a*x*x + b*x + c
```

Fórmula tras el ajuste:

```
f(x) = 4.68742e-08*x*x - 4.70613e-05*x + 0.165783
```



Algoritmo ejecutado con “lema.txt”:

Tabla:

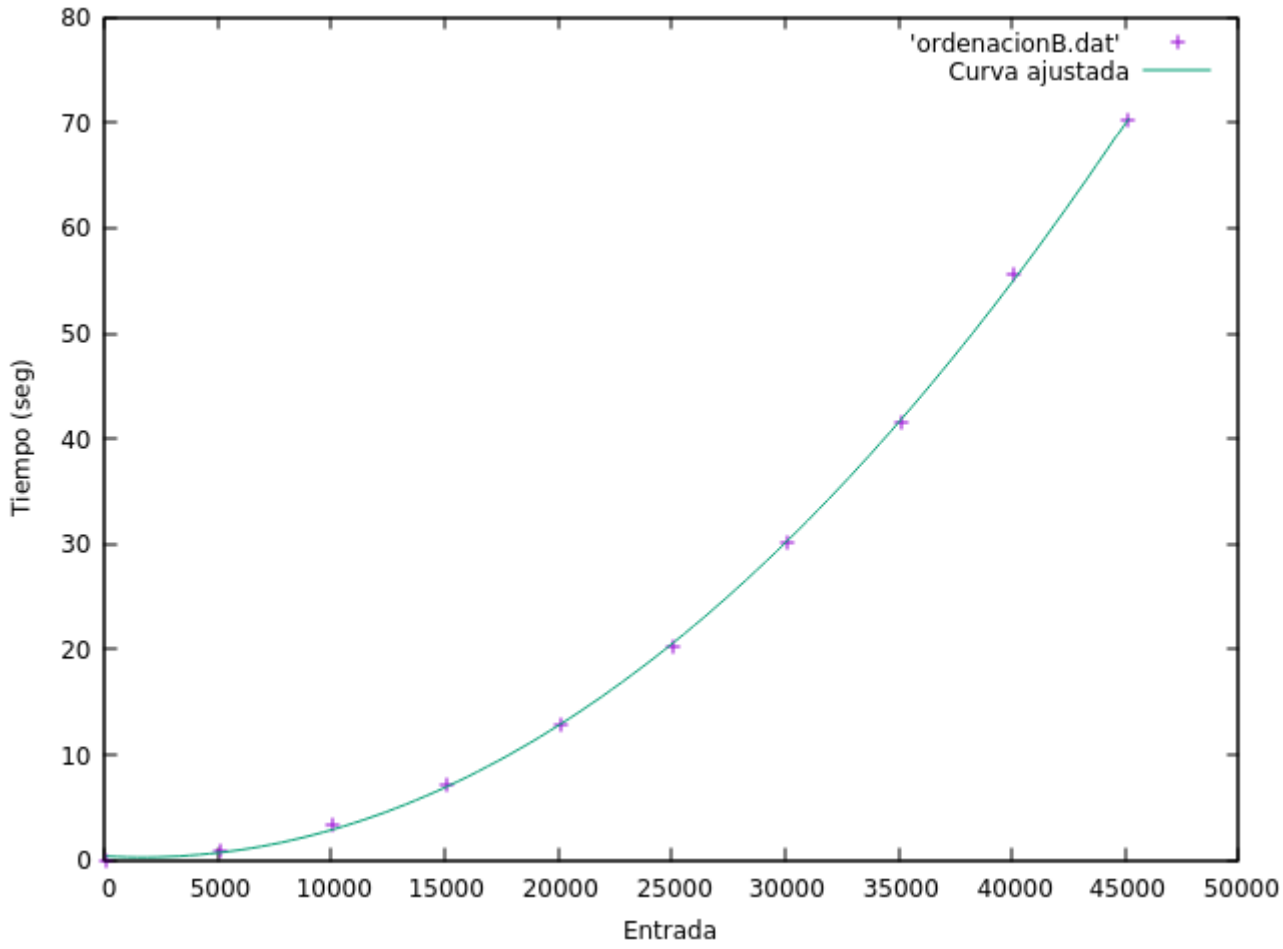
Entrada	Tiempo
100	0.000382778
5100	0.918352
10100	3.3409
15100	7.23221
20100	12.8262
25100	20.2014
30100	30.2469
35100	41.514
40100	55.551
45100	70.2358

Fórmula empelada para el ajuste:

```
f(x) = a*x*x + b*x + c
```

Fórmula tras el ajuste:

```
f(x) = 3.71141e-08*x*x - 0.000123822*x + 0.393904
```



INSERCIÓN

Análisis teórico:

Tiempo de iteración = $\Sigma(i=1 \text{ a } n)\ 1 = n \in O(n)$

$\Sigma(i=1 \text{ a } n)\ n = [n(n+1)]/2 \in O(n^2)$

Algoritmo ejecutado con "quijote.txt":

Tabla:

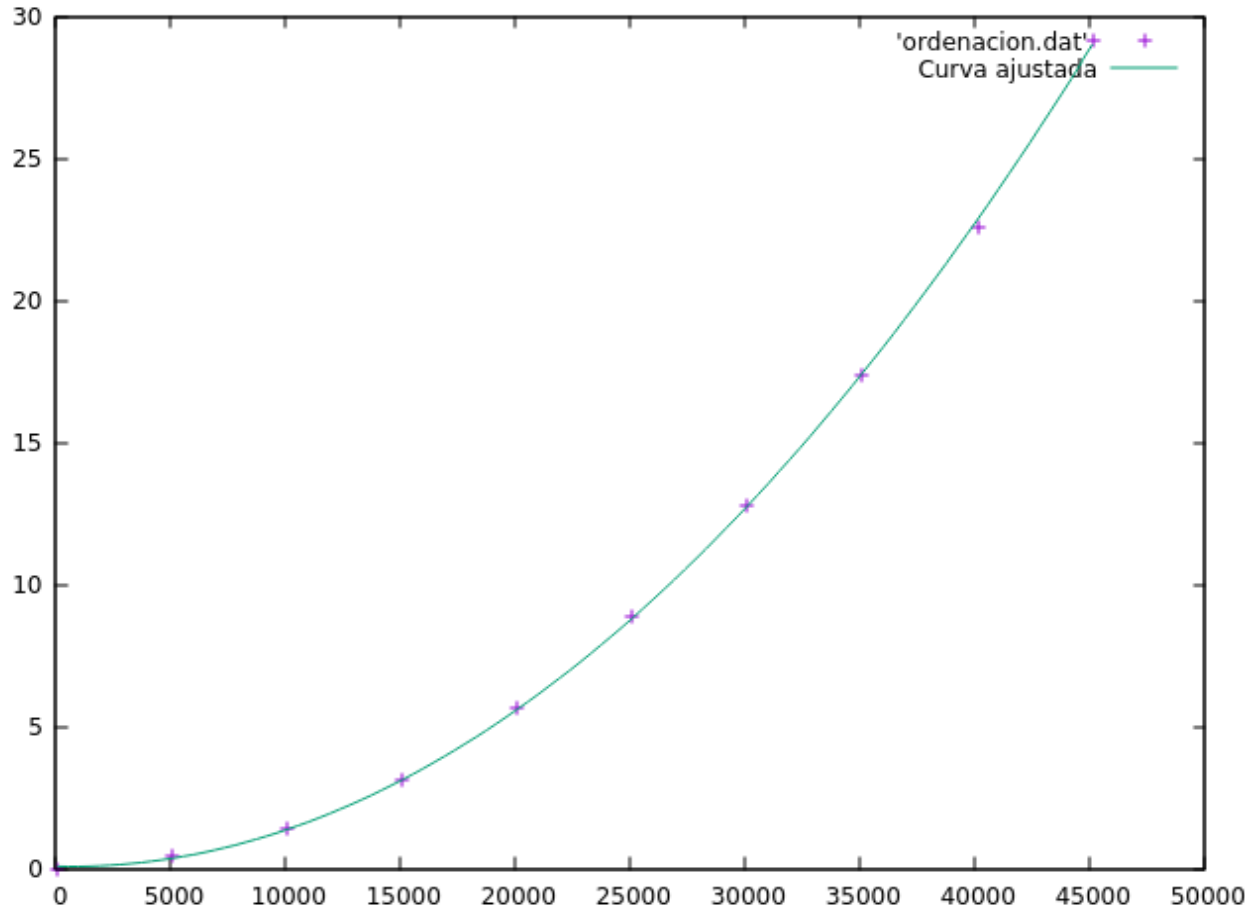
Entrada	Tiempo
100	0.000351161
5100	0.414505
10100	1.39933
15100	3.11562
20100	5.65803
25100	8.86172
30100	12.7845
35100	17.3882
40100	22.5956
45100	29.1873

Fórmula empelada para el ajuste:

```
f(x) = a*x*x + b*x + c
```

Fórmula tras el ajuste:

```
f(x) = 1.46332e-08*x*x - 1.887e-05*x + 0.0748152
```



Algoritmo ejecutado con “lema.txt”:

Tabla:

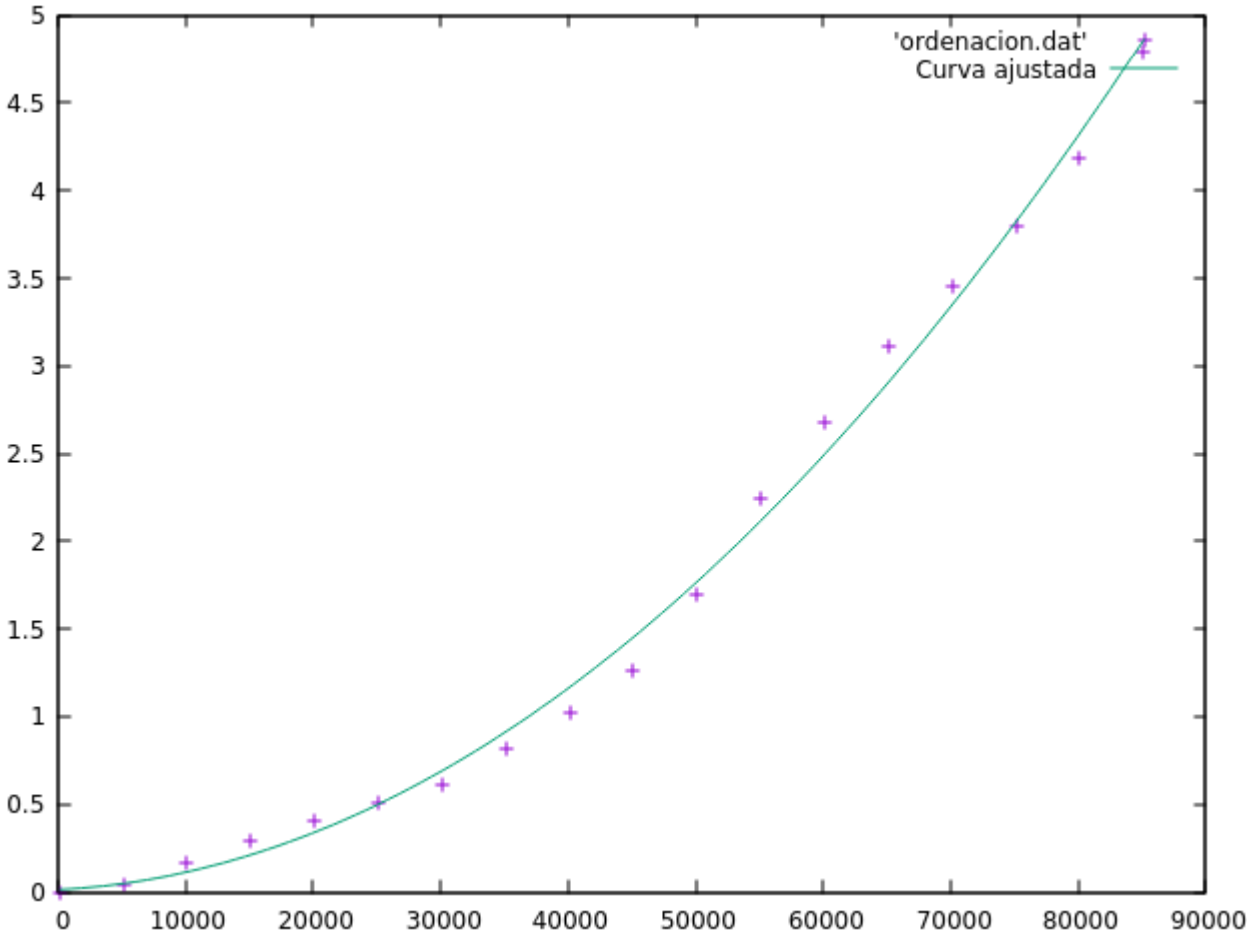
Entrada	Tiempo
100	8.3641e-05
5100	0.0355632
10100	0.162987
15100	0.288362
20100	0.410542
25100	0.507335
30100	0.617126
35100	0.811537
40100	1.0249
45100	1.26734
…	
65100	3.11273
70100	3.45355
75100	3.79752
80100	4.18522
85100	4.79024

Fórmula empelada para el ajuste:

```
f(x) = a*x*x + b*x + c
```

Fórmula tras el ajuste:

```
f(x) = 6.26393e-10*x*x + 3.65035e-06*x + 0.01293
```



SELECCIÓN

Análisis Teórico:

Tiempo de iteración = O(n-i)

$\Sigma(i=0 \text{ a } n-2)\ n+(n-1)+(n-2) = \Sigma(i=2 \text{ a } n)i = [(n+2)(n-1)]/2 \in O(n^2)$

Algoritmo ejecutado con "quijote.txt":

Tabla:

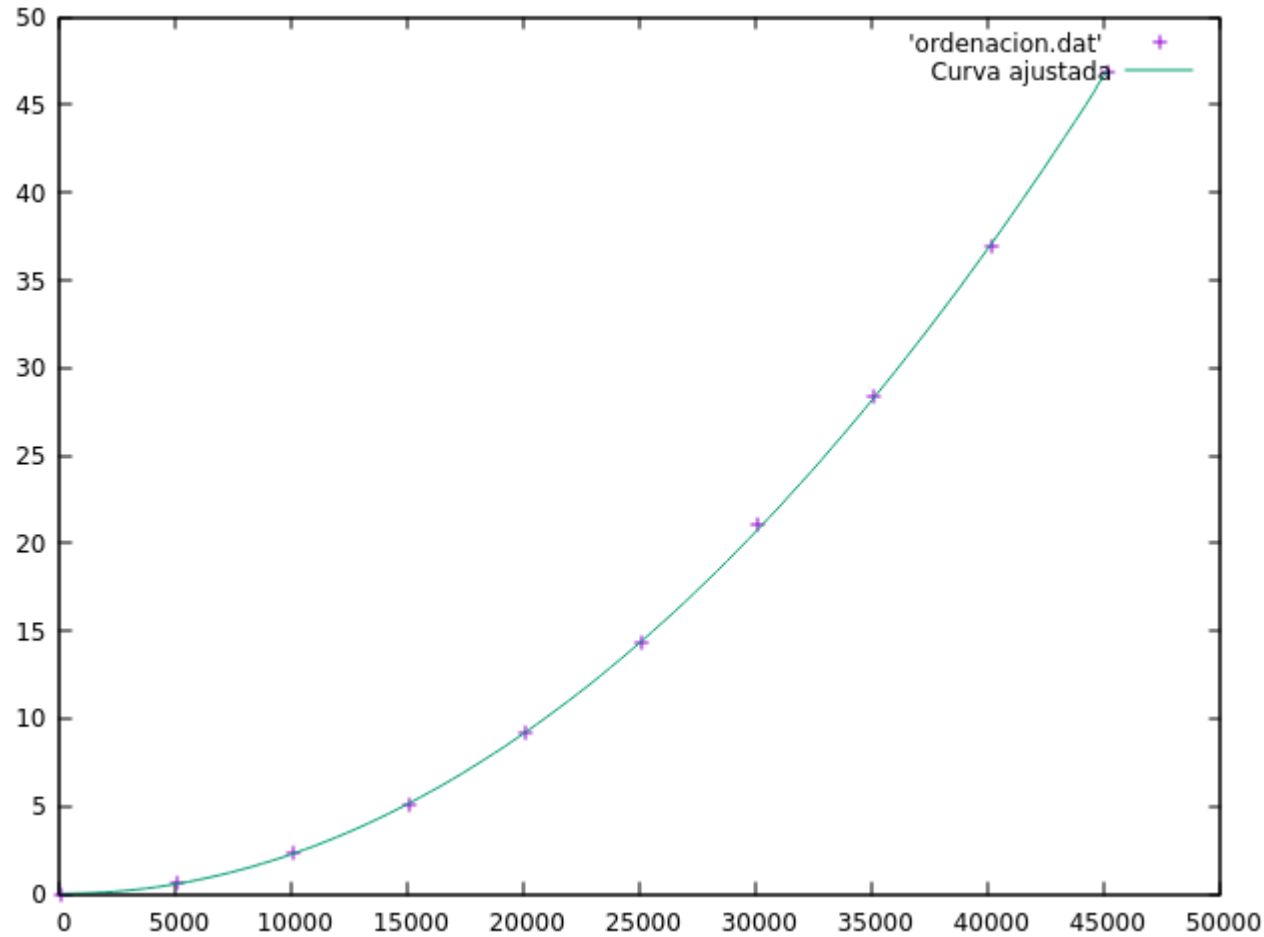
Entrada	Tiempo
100	0.000340785
5100	0.643014
10100	2.30031
15100	5.13945
20100	9.24083
25100	14.2816
30100	21.0323
35100	28.4039
40100	36.9105
45100	46.8902

Fórmula empelada para el ajuste:

```
f(x) = a*x*x + b*x + c
```

Fórmula tras el ajuste:

```
f(x) = 2.32032e-08*x*x - 7.42881e-06*x + 0.0152121
```



Algoritmo ejecutado con “lema.txt”:

Tabla:

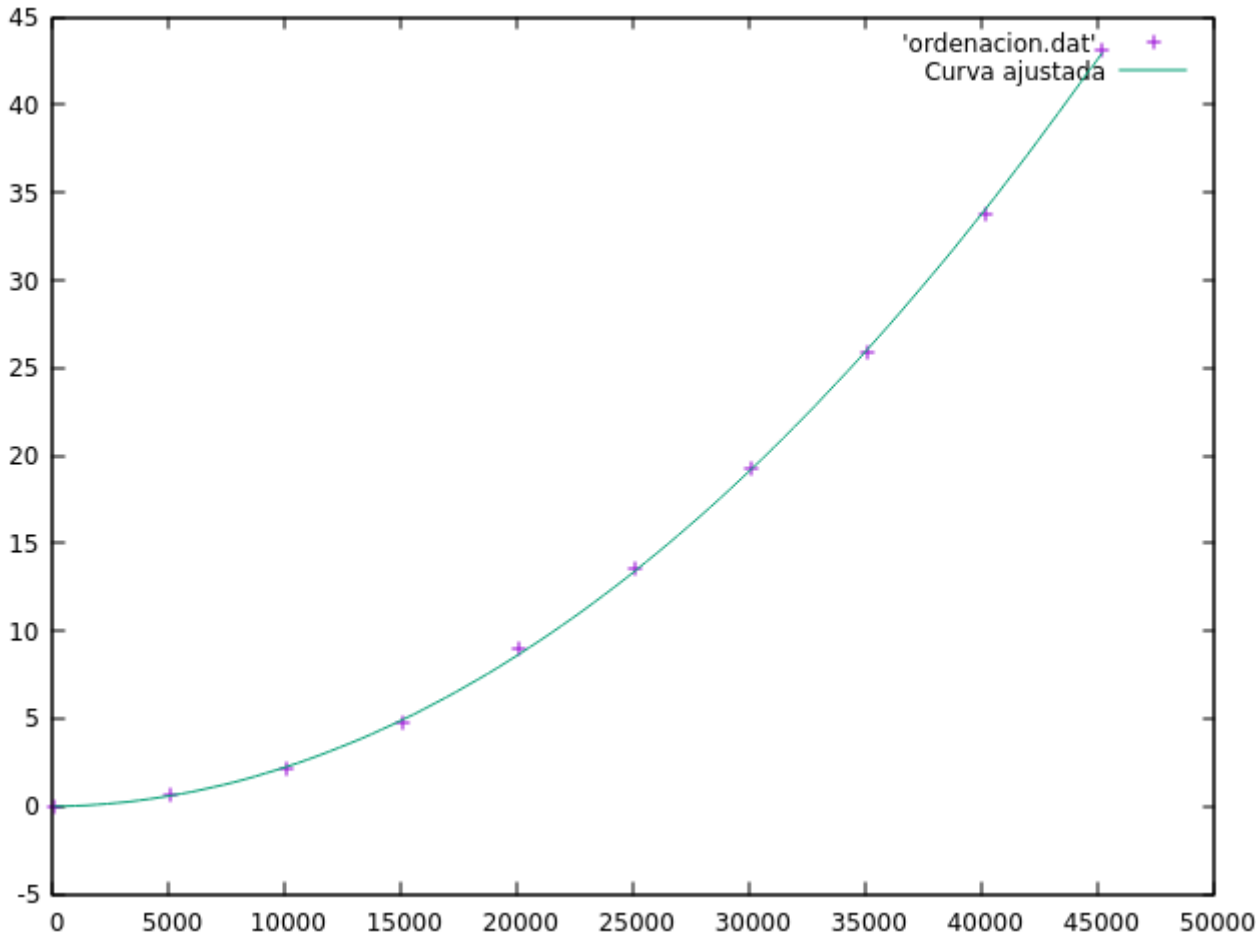
Entrada	Tiempo
100	0.000300488
5100	0.642908
10100	2.1155
15100	4.79928
20100	8.96057
25100	13.6122
30100	19.2539
35100	25.9325
40100	33.7633
45100	43.1156

Fórmula empelada para el ajuste:

```
f(x) = a*x*x + b*x + c
```

Fórmula tras el ajuste:

```
f(x) = 2.0759e-08*x*x + 1.56109e-05*x - 0.0175882
```



Podemos ver que, aunque el análisis teórico nos revele que todos los algoritmos presentan un orden cuadrático, no todos son igual de eficientes. Para ello el análisis práctico nos proporciona una mayor información. Como se observa en los tiempos, el algoritmo de ordenación que presenta una mayor eficiencia es el de inserción, que presenta unos tiempos de ejecución bastánte menores que el resto (sobre todo al ordenar “lema.txt”; a continuación le sigue el algoritmo de selección, y por último el algoritmo de la burbuja, que sería el más ineficiente de todos. Esto nos demuestra la importancia que tiene el análisis híbrido a la hora de estudiar la eficiencia de un algoritmo.