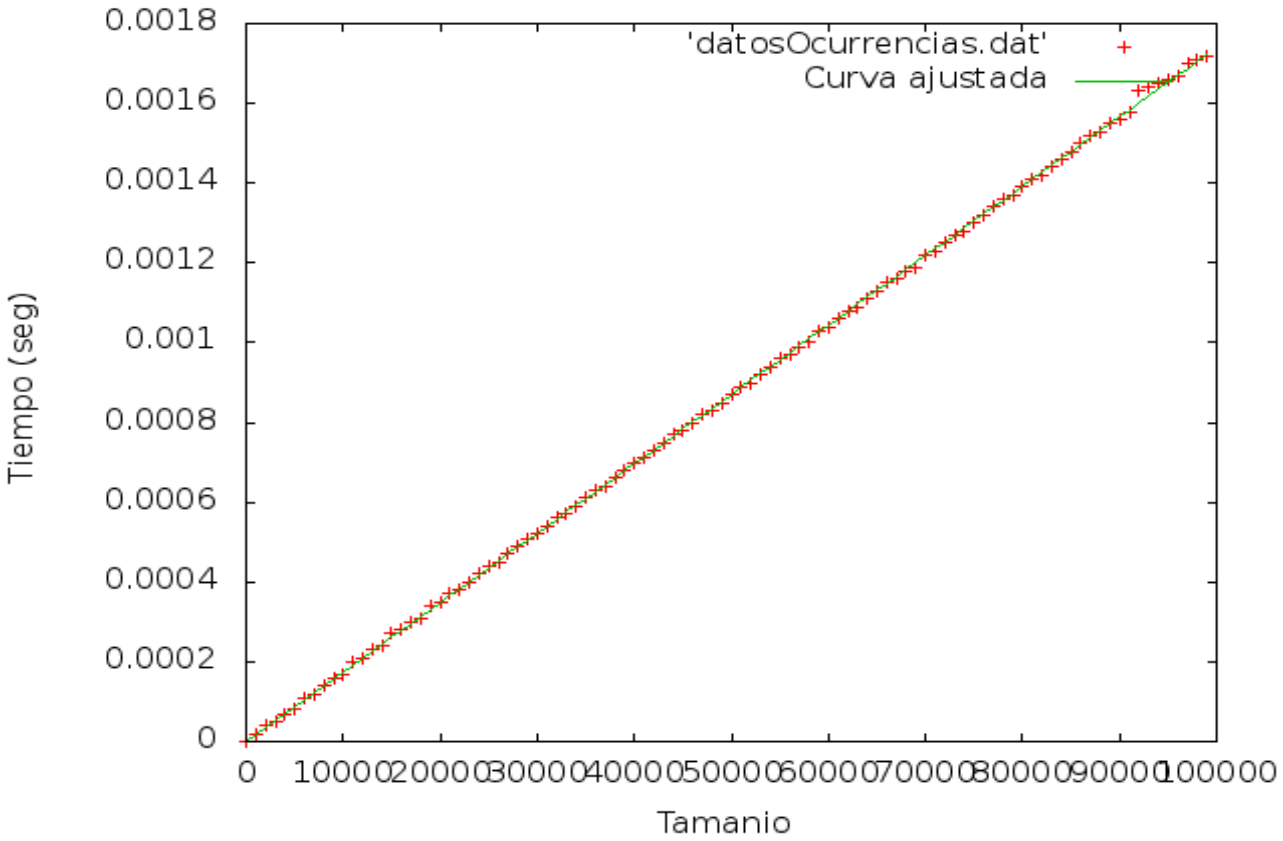
1. Análisis de eficiencia del código “ocurrencias.cpp”

Análisis teórico:

```
int contar_hasta( vector<string> & V, int ini, int fin, string & s) {
        int cuantos = 0;                                  O(1)
        for (int i=ini; i< fin ; i++)                     O(max(1,1,1))
                if (V[i]==s) {                  O(1)
                        cuantos ++;             O(1)      O(1)
                }                                         O(fin - ini)
        return cuantos;                                   O(1)
}
```

O(max(1,fin – ini,1)) = O(fin – ini) ==> ini siempre vale 0 en la llamada en el main() por lo que la eficiencia es de orden O(fin) = O(n)

Análisis empírico:



La curva que se ha ajustado a los puntos de los datos es una recta f(x) = a * x; donde a = 1.74067 $*10^{-08}$.

Con un error en el ajuste del 0.05686%

Por tanto el análisis practico se ajusta a lo descrito por el análisis teórico.

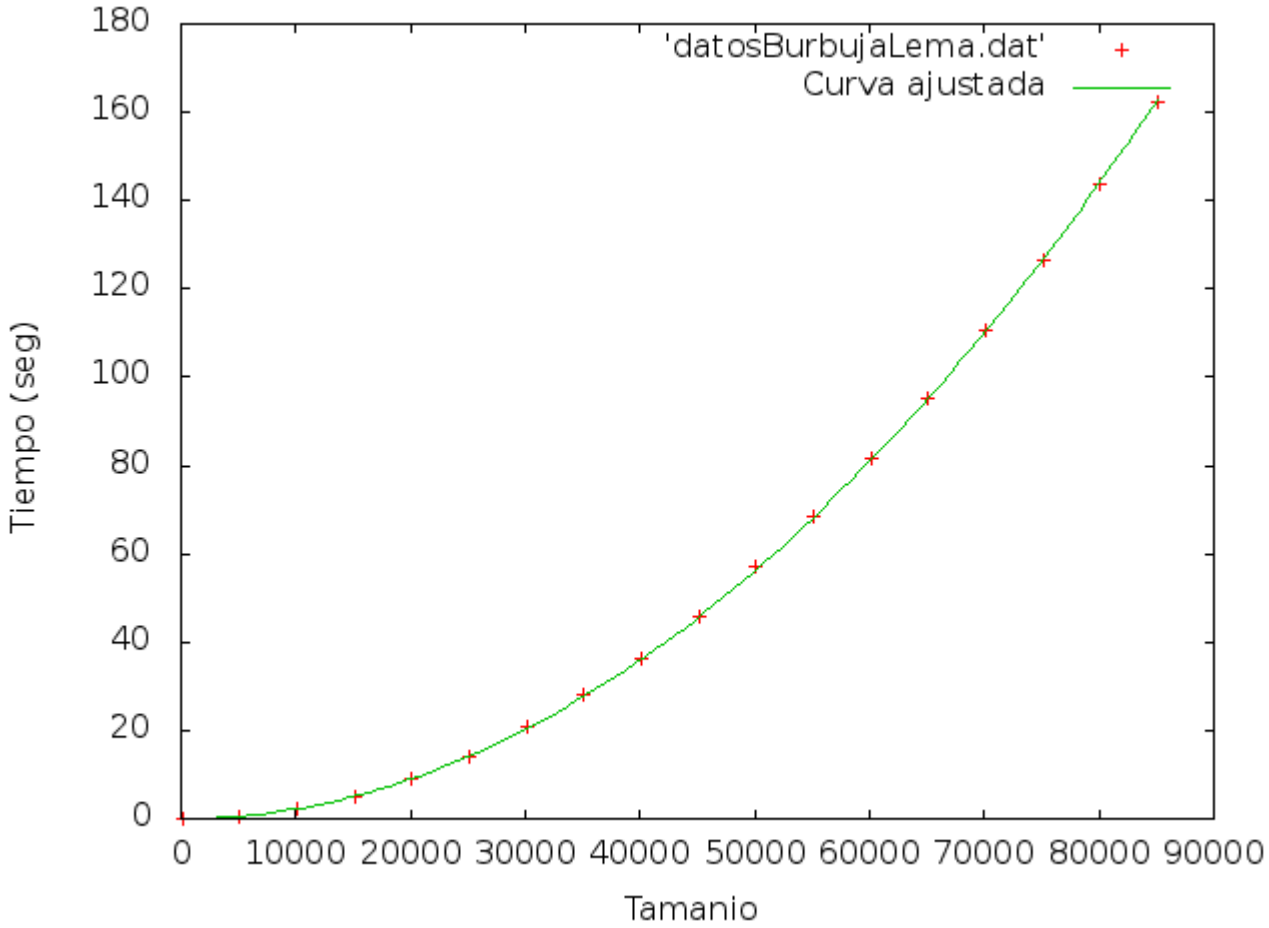
2. Análisis eficiencia de los algoritmos: burbuja, inserción y selección.

Burbuja:

Análisis teórico:

Él análisis teórico de este algoritmo ya se realizó en el guión de prácticas por lo que únicamente hay que recordar que era de orden $O(n^2)$

Análisis empírico:

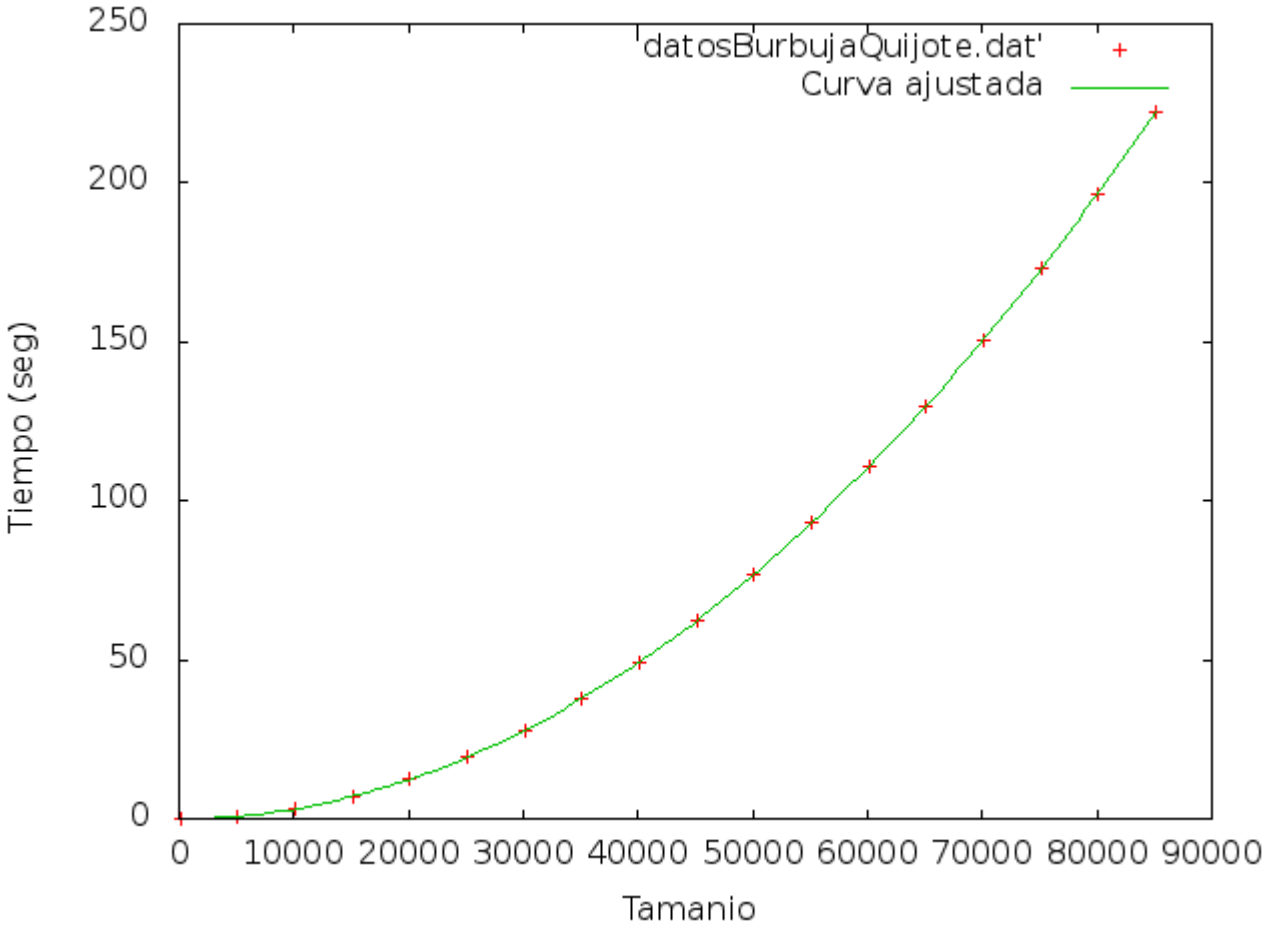


(USANDO AUX = DICCIONARIO)

La curva que se ha ajustado a los puntos de los datos es una recta $f(x) = a * x^2$; donde $a = 2.24687 * 10^{-08}$.

Con un error en el ajuste del 0.08397%

Por tanto el análisis practico se ajusta a lo descrito por el análisis teórico.



(USANDO AUX = QUIJOTE)

La curva que se ha ajustado a los puntos de los datos es una recta $f(x) = a * x^2$; donde a =3.06641 $*10^{-08}$.

Con un error en el ajuste del 0.0197%

Por tanto el análisis practico se ajusta a lo descrito por el análisis teórico.

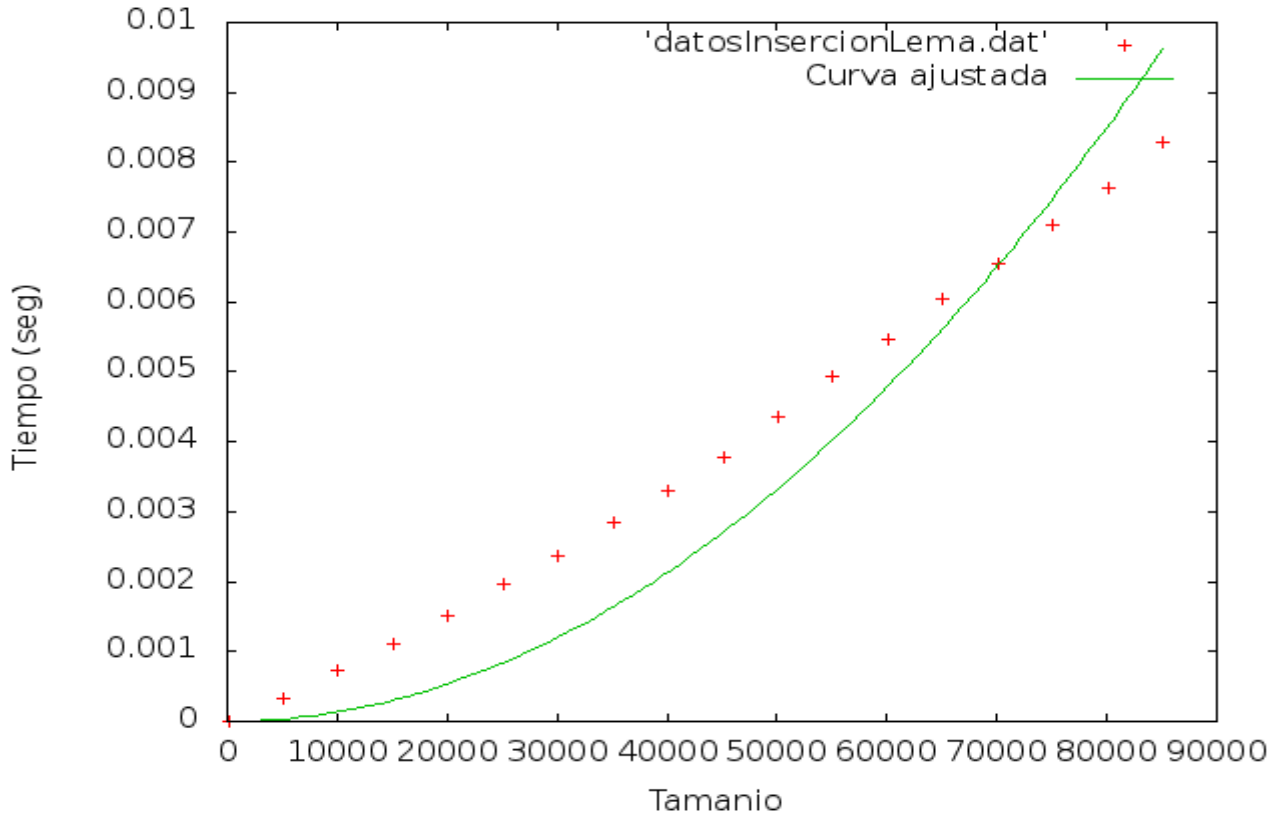
Inserción:

Análisis teórico:

```
void insertSort(vector<string> & v, int inicial, int final) {
        int i, j;                                                          O(1)
        string aux;                                                        O(1)

        for (int i=inicial; i<final; i++) {                                O(max(1,1,1))
        aux=v[i];                                                          O(1)
                for (j=i-1; j>=0 && v[j]>aux; j--){                        O(max(1,1,1))
                        v[j+1] = v[j];                                     O(1)   O(max(1,1))
                        v[j] = aux;                                        O(1)
        }                                                                  O(n)
        }                                                                  O(max(1,n))=O(n)
}                                                                          O(n^2)
                                                                           O(n^2)
```

Análisis empírico:

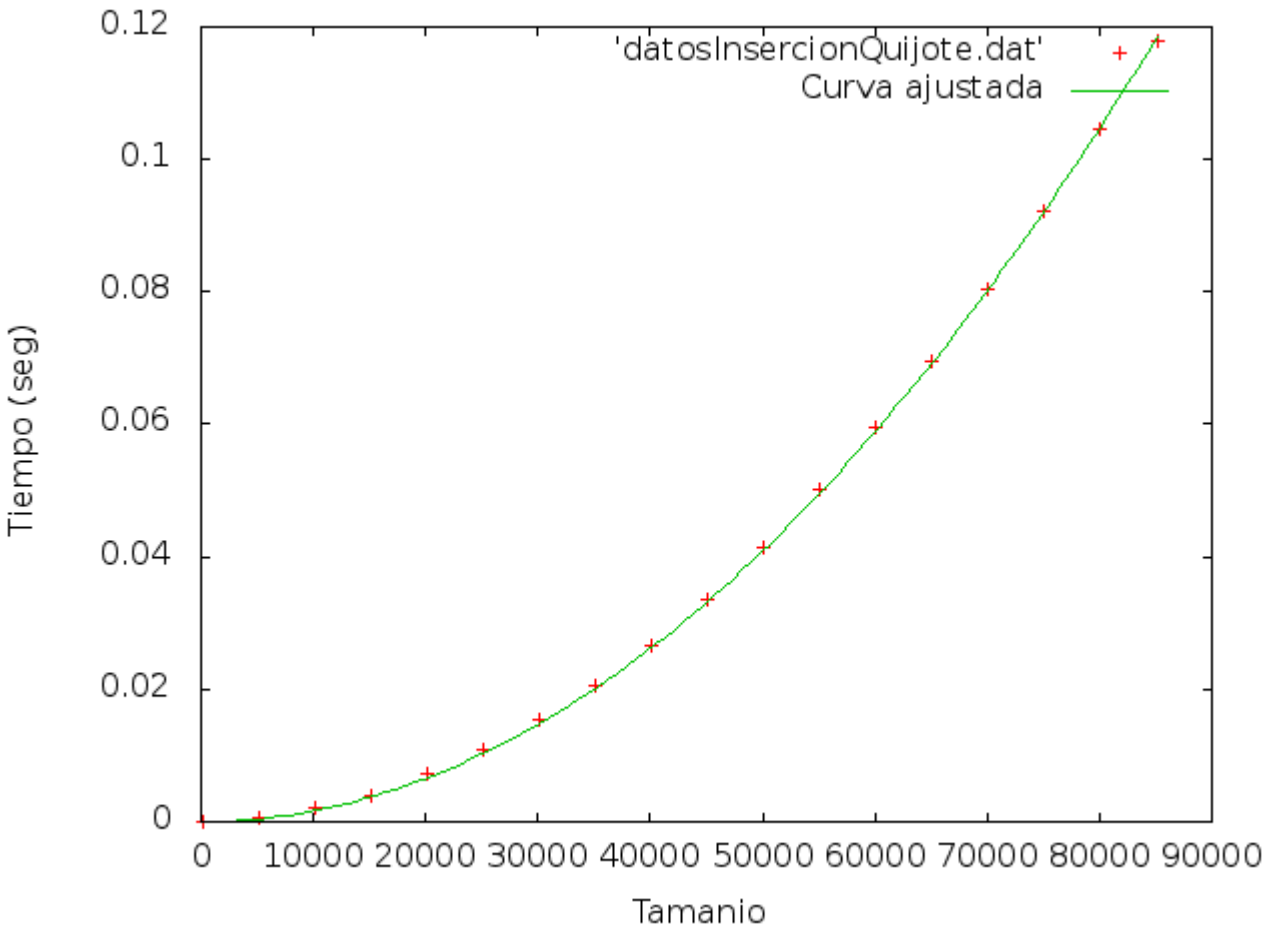


(USANDO AUX = DICCIONARIO)

La curva que se ha ajustado a los puntos de los datos es una recta $f(x) = a * x^2$; donde $a = 1.32788 * 10^{-12}$.

Con un error en el ajuste del 4.73%

El hecho de que los resultados no se ajusten demasiado bien a una curva cuadrática y si lo hagan mejor a una recta se debe a que el vector (Diccionario) está "casi totalmente" ordenado. Por este motivo la gran mayoría de las veces el método de ordenación por inserción no entra en el segundo bucle (el anidado en el primero) por lo que el tiempo de ejecución para este caso concreto se ajusta mejor a una función rectilínea.



(USANDO AUX = QUIJOTE)

La curva que se ha ajustado a los puntos de los datos es una recta $f(x) = a * x^2$; donde $a = 1.63602 * 10^{-11}$.

Con un error en el ajuste del 0.1709%

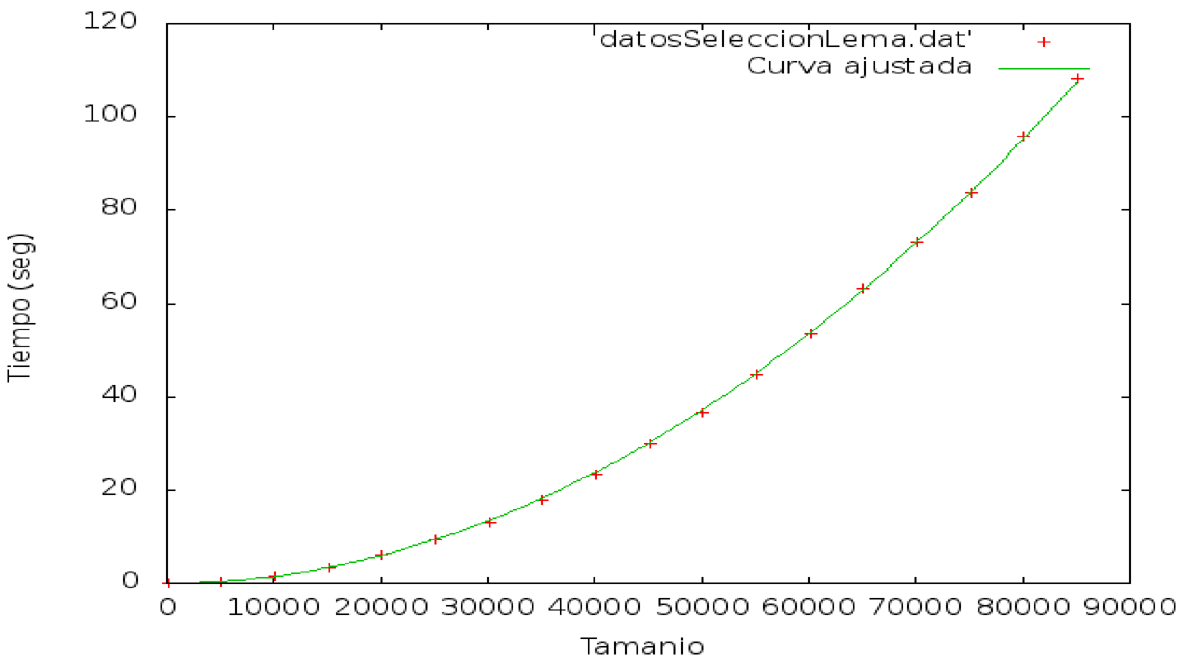
Por tanto el análisis practico se ajusta a lo descrito por el análisis teórico.

Selección:

Análisis teórico:

```
void ordena_seleccion(vector<string> & v, int inicio, int final) {
        string aux;                                              O(1)
        for (int i=inicio; i < final; i++) {            O(1)
                int min = i;                            O(1)
                for (int c=i+1; c < final; c++){
                        if (v[min] > v[c])                      O(n)
                                min = c;        O(1)
                }                                              O(n)     O(n²)     O(n²)
                aux=v[i];                               O(1)
                v[i] = v[min];                          O(1)
                v[min] = aux;                           O(1)
        }
}
```

Análisis empírico:

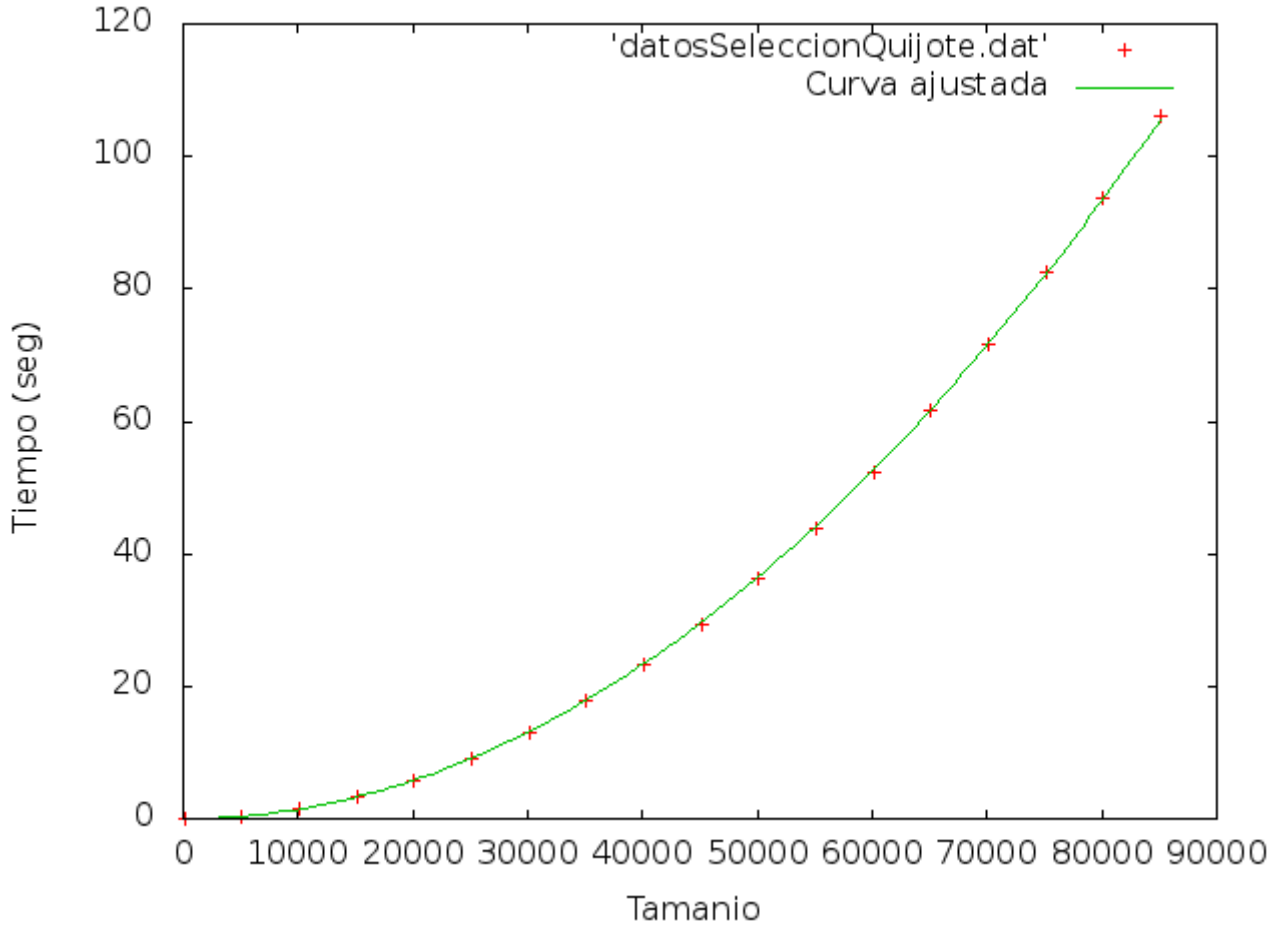


(USANDO AUX = DICCIONARIO)

La curva que se ha ajustado a los puntos de los datos es una recta $f(x) = a * x^2$; donde $a = 1.48677 * 10^{-08}$.

Con un error en el ajuste del 0.1762%

Por tanto el análisis practico se ajusta a lo descrito por el análisis teórico.



(USANDO AUX = QUIJOTE)

La curva que se ha ajustado a los puntos de los datos es una recta $f(x) = a * x^2$; donde $a = 1.4584 * 10^{-08}$.

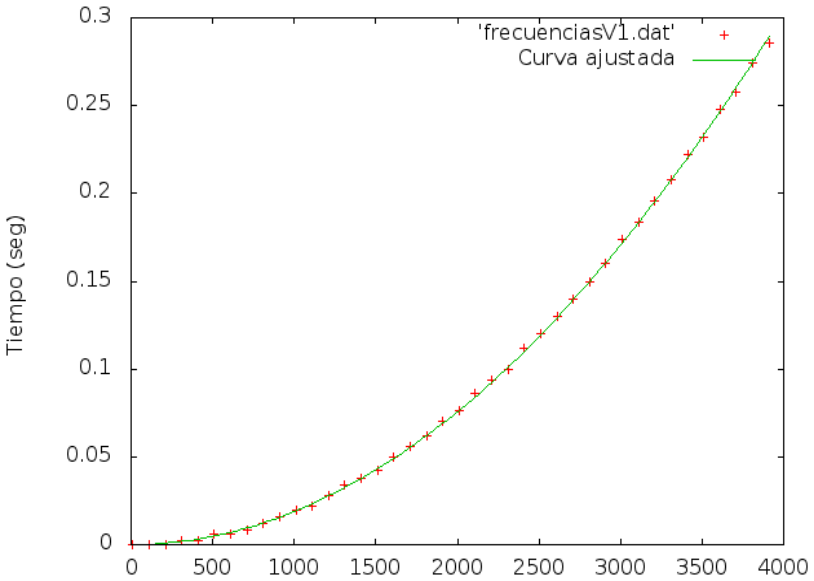
Con un error en el ajuste del 0.09947%

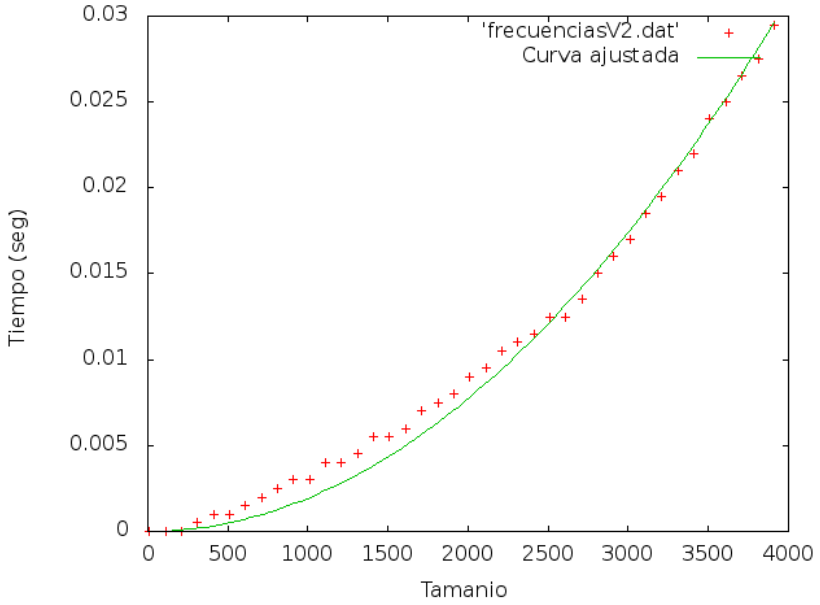
Por tanto el análisis practico se ajusta a lo descrito por el análisis teórico.

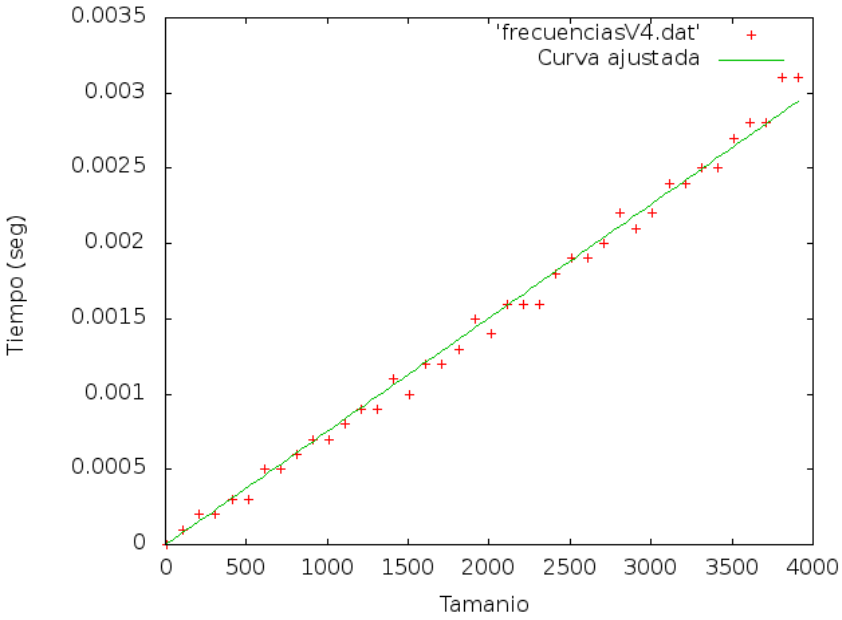
CONCLUSIÓN DEL APARTADO 2

Aún siendo los 3 algoritmos analizados en este apartado del orden $O(n^2)$, se aprecia claramente que el más eficiente de los 3 es el de ordenación por inserción siendo del orden de 1000 a 10000 veces más eficiente con respecto a los otros 2, además se aprecia claramente que su tiempo de ejecución es menor cuanto más favorable sea el caso en cuestión, cosa que no se ve tan acentuada en los otros 2 algoritmos. Lo más destacable del método de ordenación por selección es que su constante oculta (a) varía muy poco de una caso a otro por lo que para 2 casos totalmente distintos y un tamaño de entrada igual para ambos se puede esperar un tiempo de ejecución similar. Para concluir diremos que el menos eficiente de los 3 algoritmos es el de ordenación por burbuja por tener una constante oculta mayor para ambos casos que la de los otros 2 algoritmos.

3. Análisis comparativo de eficiencia para los algoritmos del fichero frecuencia.cpp







FrecuenciasV1	FrecuenciasV2	FrecuenciasV4
$f(x) = a * x^2$	$f(x) = a * x^2$	$f(x) = a * x$
$a = 1.89402 * 10^{-8}$	$a = 1.93651 * 10^{-9}$	$a = 7.53735 * 10^{-7}$
error = 0.1517%	error = 0.9849%	error = 0.6932%

Nota: no he podido realizar la comparativa de la eficiencia del algoritmo frecuenciasV3 debido a que el código de dicho algoritmo era erróneo y al ejecutarlo daba una violación de segmento.