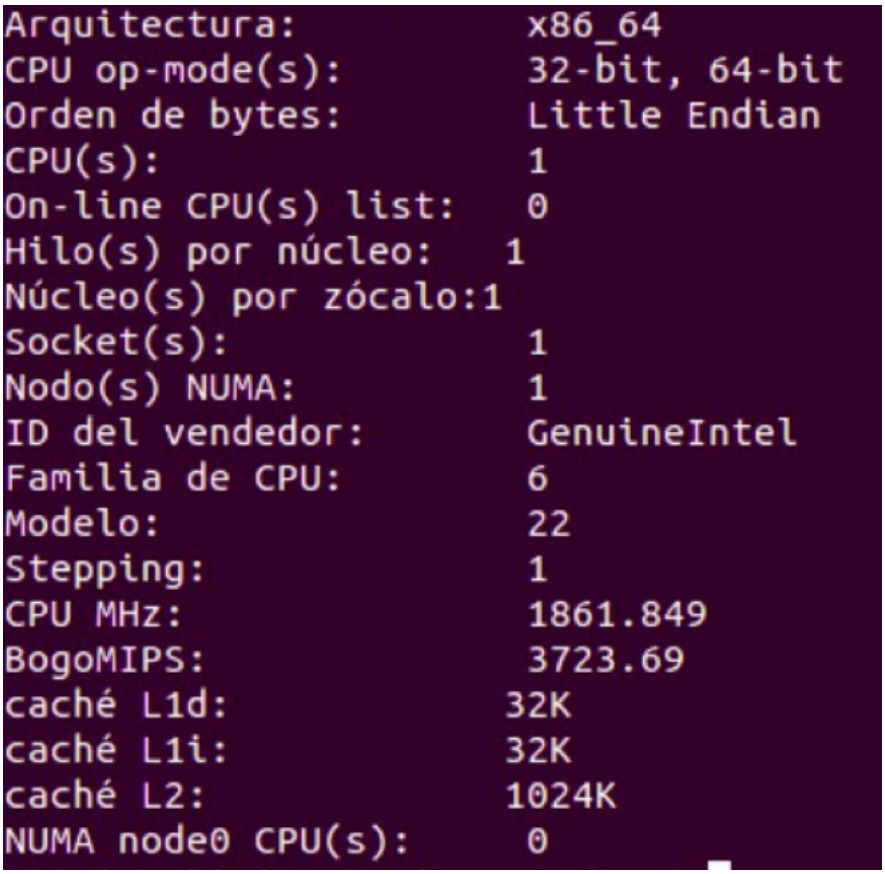


Ejercicio 1: Ordenación de la burbuja .

El siguiente código realiza la ordenación mediante el algoritmo de la burbuja:

```
void ordenar(int *v, int n) {
    for (int i=0; i<n-1; i++)
        for (int j=0; j<n-i-1; j++)
            if (v[j]>v[j+1]) {
                int aux = v[j];
                v[j] = v[j+1];
                v[j+1] = aux;
            }
}
```

Calcule la eficiencia teórica de este algoritmo. A continuación replique el experimento que se ha hecho antes (búsqueda lineal) con este nuevo código. Debe:

- **Crear un fichero ordenacion.cpp con el programa completo para realizar una ejecución del algoritmo.**
- **Crear un script ejecuciones_ordenacion.csh en C-Shell que permite ejecutar varias veces el programa anterior y generar un fichero con los datos obtenidos.**
- **Usar gnuplot para dibujar los datos obtenidos en el apartado previo.**

Los datos deben contener tiempos de ejecución para tamaños del vector 100, 600, 1100, ..., 30000.

Pruebe a dibujar superpuestas la función con la eficiencia teórica y la empírica. ¿Qué sucede?

Eficiencia teorica:

```
void ordenar(int *v, int n) {
    for (int i=0; i<n-1; i++)          ←T(n)= Σ_{i=0}^{n-2} n-i-1 = Σ_{i=0}^{n-2} n - Σ_{i=0}^{n-2} i - Σ_{i=0}^{n-2} 1 =
                                           = n(n-1)- n((n-1)/2)-(n-1) = n²-n -(n²/2)+(n/2)-n+1=
                                           = (n²-3n+2)/2 ∈ O(n²)

        for (int j=0; j<n-i-1; j++)    ←Σ_{j=0}^{n-i-2} 1= n-i-1

            if (v[j]>v[j+1]) {         ←El if y lo que hay dentro vale O(1)
                int aux = v[j];
                v[j] = v[j+1];
                v[j+1] = aux;
            }
}
```

Puesto que $T(n)=(n^2-3n+2)/2 \in O(n^2)$ podemos afirmar que el orden de eficiencia del algoritmo de ordenación por burbuja es $O(n^2)$.

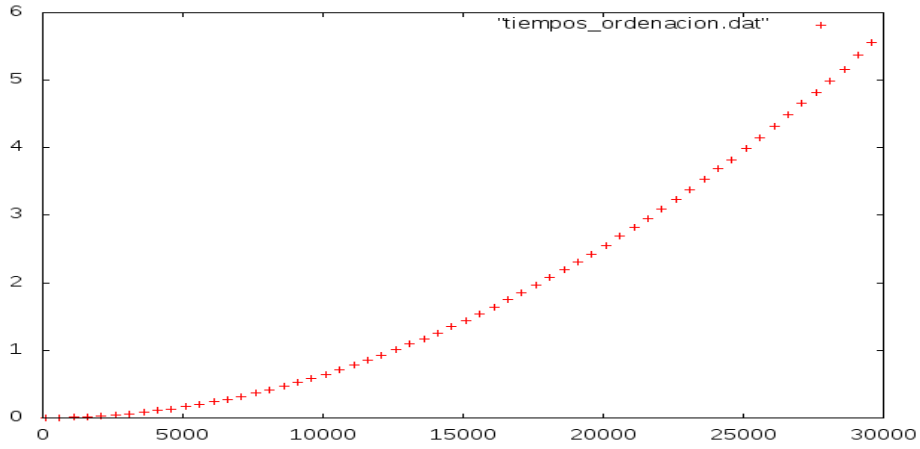
Eficiencia empírica:

Para medir el tiempo de ejecución del algoritmo de ordenación por burbuja, generamos un vector ordenado de mayor a menor , provocando de esta forma que se dé el peor caso posible. El programa tiene un argumentos que se le suministra en la línea de órdenes, el tamaño del vector.

- He creado el fichero **ordenacion.cpp** para realizar la ejecución del algoritmo.
- He creado el script ejecuciones_ordenacion.csh para ejecutar varias veces el programa anterior para tamaños del vector 100, 600, 1100, ..., 30000 y generar un fichero con los datos obtenidos.

-He usado gnuplot para dibujar los datos obtenidos en el apartado previo, resultando:

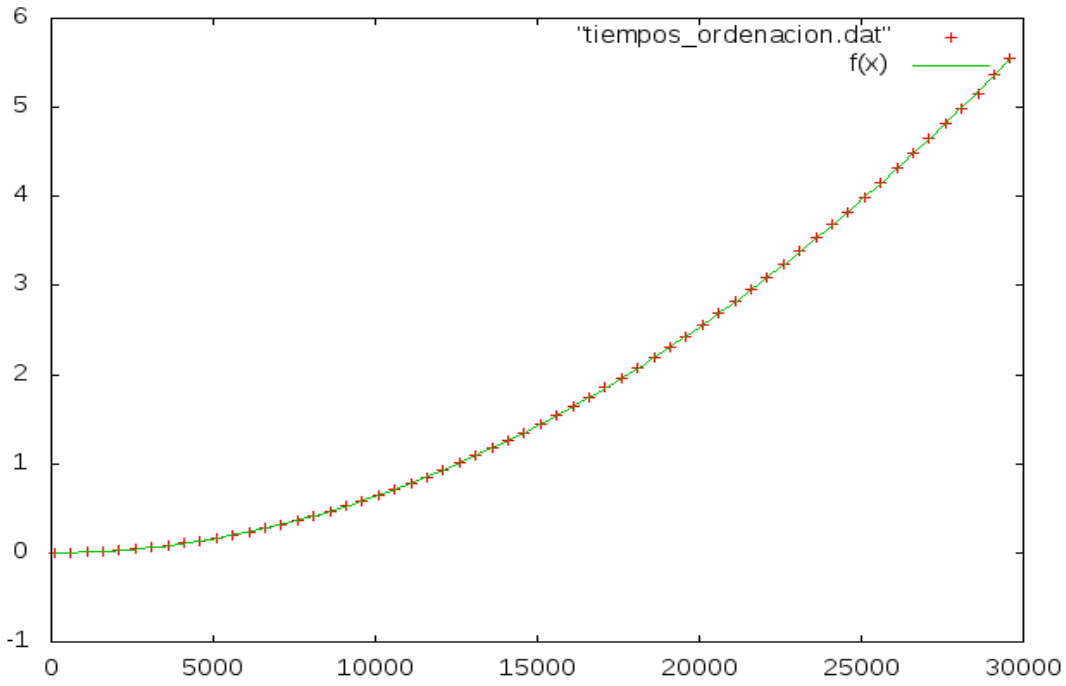
gnuplot> plot "tiempos_ordenacion.dat"



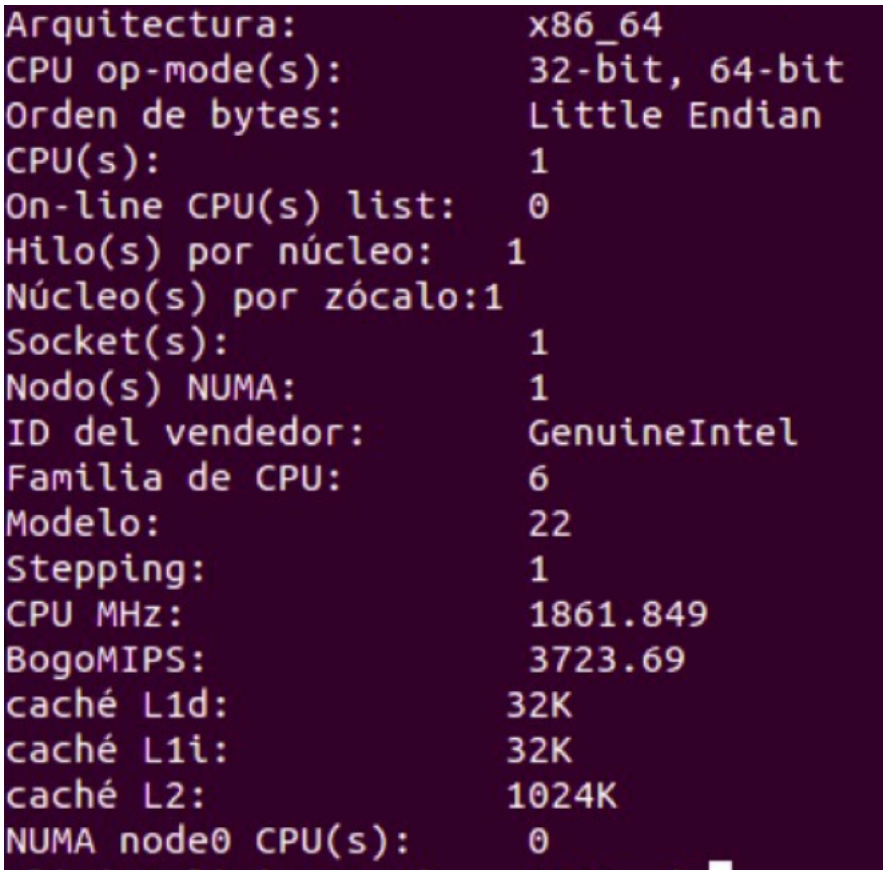
Dibujar superpuestas la función con la eficiencia teórica y la empírica:

Para el calculo del ajuste teorico con el empirico primero he declarado la funcion f(x) como "f(x)=a*x**2+b*x+c" ya que para el caso peor es $0(n^2)$.
Una vez que he declarado la funcion he pasado a ajustar con la orden "fit" para calcula la a, b y c. Despues he calculado el ajuste dibujando las dos graficas.
Pasos:

1) gnuplot> f(x)=a*x2+b*x+c**
2) fit f(x) "tiempos_ordenacion.dat" via a,b,c
3) gnuplot> plot "tiempos_ordenacion.dat" , f(x)



La grafica de la eficiencia teorica se ajusta con la de eficiencia empírica.



Ejercicio 2: Ajuste en la ordenación de la burbuja

Replique el experimento de ajuste por regresión a los resultados obtenidos en el ejercicio 1 que calculaba la eficiencia del algoritmo de ordenación de la burbuja. Para ello considere que f(x) es de la forma ax2+bx+c

Para ello en gnuplot escribimos:

1) gnuplot> f(x)=a*x2+b*x+c**

2) gnuplot> fit f(x) "tiempos_ordenacion.dat" via a,b,c

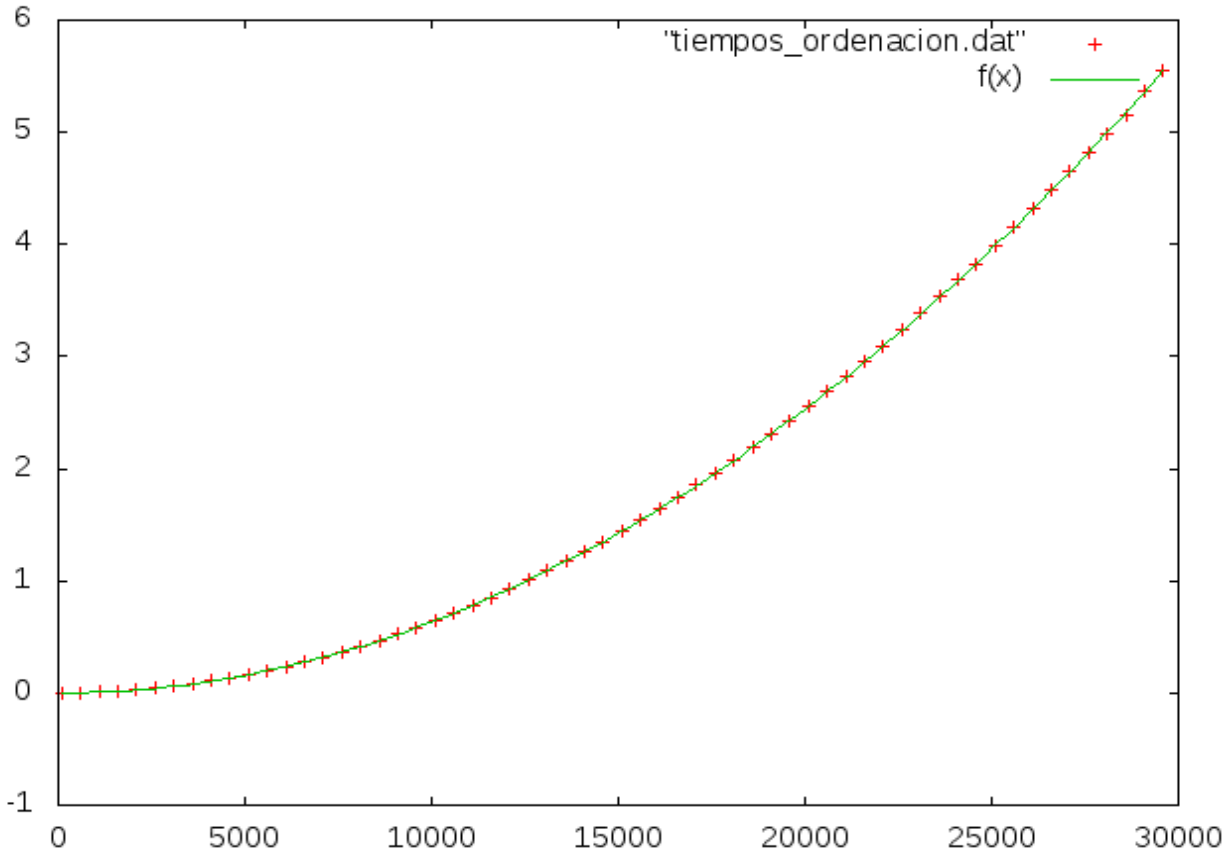
Con lo que obtendremos los valores a, b y c que producen un mejor ajuste entre la curva teórica y la empírica:

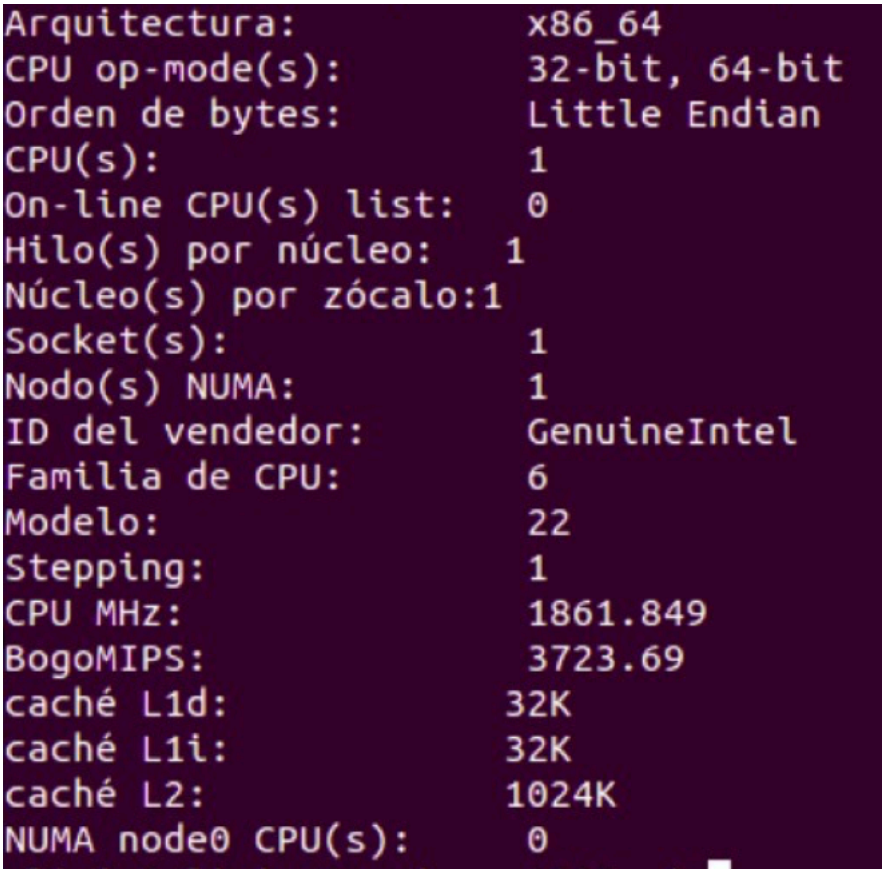
a = 6.32373e-09
b = 1.93992e-07
c = -0.000366902

```
After 12 iterations the fit converged.
final sum of squares of residuals : 0.00153502
rel. change during last iteration : -2.68648e-10

degrees of freedom    (FIT_NDF)                        : 57
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00518943
variance of residuals (reduced chisquare) = WSSR/ndf   : 2.69302e-05

Final set of parameters            Asymptotic Standard Error
=======================            ==========================

a               = 6.32373e-09      +/- 9.994e-12    (0.158%)
b               = 1.93992e-07      +/- 3.067e-07    (158.1%)
c               = -0.000366902     +/- 0.001971     (537.2%)


correlation matrix of the fit parameters:

               a      b      c
a               1.000
b              -0.968  1.000
c               0.738 -0.861  1.000
gnuplot> plot "tiempos_ordenacion.dat" , f(x)
```

3) gnuplot> plot "tiempos_ordenacion.dat" , f(x)

Para dibujar las funciones (teorica y empírica)superpuestas, resultando la siguiente gráfica:


"tiempos_ordenacion.dat"
f(x)
6
5
4
3
2
1
0
-1
0
5000
10000
15000
20000
25000
30000



Ejercicio 3: Problemas de precisión

Junto con este guión se le ha suministrado un fichero ejercicio_desc.cpp. En él se ha implementado un algoritmo. Se pide que:

- **Explique qué hace este algoritmo.**
- **Calcule su eficiencia teórica.**
- **Calcule su eficiencia empírica.**

Si visualiza la eficiencia empírica debería notar algo anormal. Explíquelo y proponga una solución. Compruebe que su solución es correcta. Una vez resuelto el problema realice la regresión para ajustar la curva teórica a la empírica.

¿Qué hace este algoritmo?:

1) Se ha creado un vector te tamaño "tam" con elementos aleatorios entre 0 y "tam". Luego no está ordenado.
2) Desde en main se llama a la función con los siguientes parametros:
   operacion(v,tam,tam+1,0,tam-1);
3)El algoritmo es:

```
int operacion(int *v, int n, int x, int inf, int sup) { //x es el elemento a buscar
    int med;
    bool enc=false;
    while ((inf<sup) && (!enc)) {
        med = (inf+sup)/2; //punto medio
        if (v[med]==x)          ←Este if no se va a cumplir nunca, porque x=tam+1 y
                                  el vector solo contiene numeros del 0 hasta "tam"
            enc = true;
        else if (v[med] < x)
            inf = med+1;
        else
            sup = med-1;
    }
    if (enc)
        return med;
    else
        return -1;
}
```

Este algoritmo es el algoritmo de busqueda binaria, el cual consiste en que el elemento que estamos buscando se compara con el elemento que ocupa la mitad del vector, si coinciden se termina la búsqueda, si no, se determina la mitad del vector en la que puede estar el elemento y se repite el proceso. Para ello el vector debe estar ordenado, cosa que no esta porque esta creado con elementos aleatorios entre 0 y tam.

Eficiencia Teórica:

$$T(n) = 1+2+3+\sum_{inf=0}^{sup-1}(3+2+2+3)+1+1 = 8+\sum_{inf=0}^{sup-1}10 = 8+10sup \approx 8+10\,n \in \mathbf{O(n)}$$

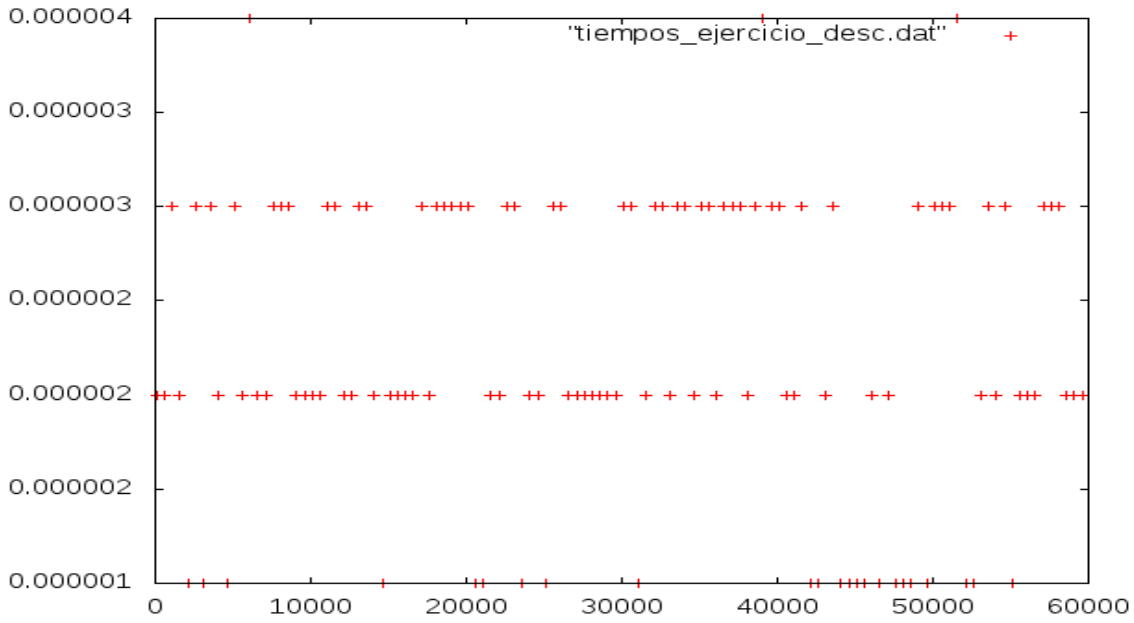
Eficiencia Empírica:

1) Creamos el ejecutable con : **g++ ejercicio_desc.cpp -o ejercicio_desc**

2) Creo el script **ejecuciones_ejercicio_desc.csh** para ejecutar varias veces el programa anterior para tamaños del vector 100, 600, 1000, ..., 60000 y generar un fichero con los datos obtenidos.

3)Para dibujar los datos obtenidos en el apartado anterior uso gnuplot:

gnuplot> plot "tiempos_ejercicio_desc.dat"



El resultado de la eficiencia empírica es una gráfica con varios valores repetidos en el eje de ordenadas. Para un mismo tamaño resultan tiempos diferentes.

Mi solución:

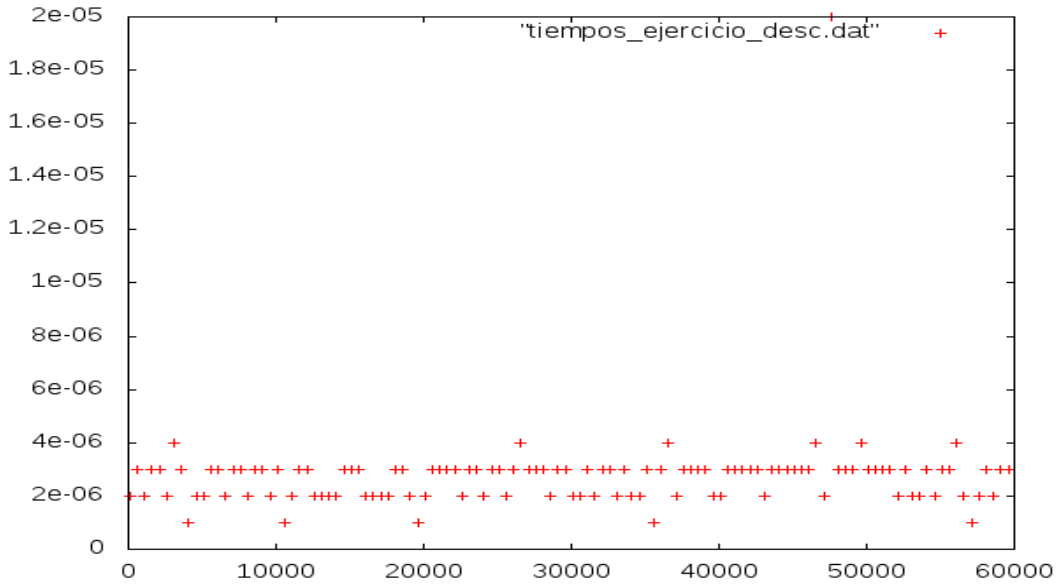
1) Crear un vector te tamaño "tam" con elementos ordenados de menor a mayor.

2) Desde en main llamar a la función con los siguientes parametros:
operacion(v,tam+1,0,tam-1);

3)El algoritmo es:

```
int operacion(int *v, int x, int inf, int sup) { //x es el elemento a buscar
    int med;
    bool enc=false;
    while ((inf<=sup) && (!enc)) {
        med = (inf+sup)/2; //punto medio
        if (v[med]==x)
            enc = true;
        else if (v[med] < x)
            inf = med+1;
        else
            sup = med-1;
    }
    if (enc)
        return med;
    else
        return -1;
```

Ahora la eficiencia empirica queda de la siguiente forma:



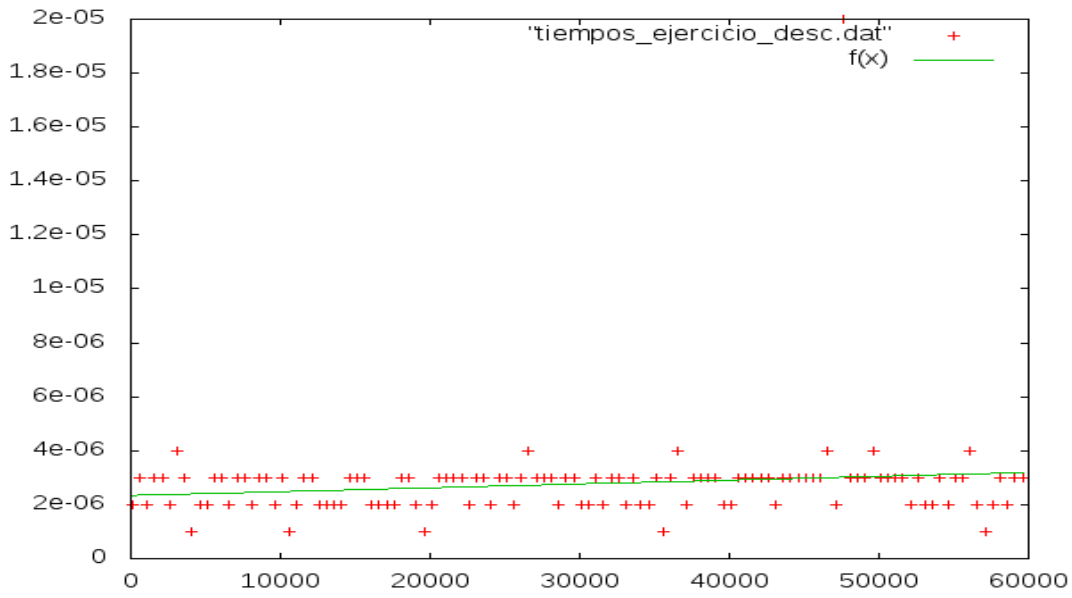
Regresión para ajustar la curva teórica a la empírica:

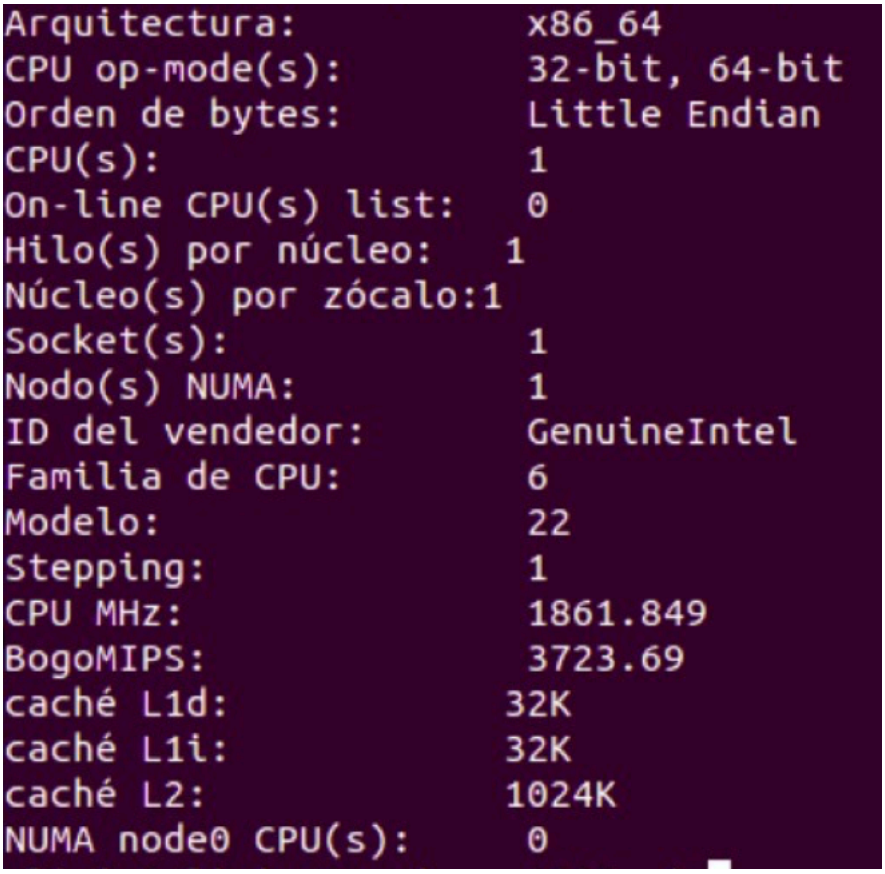
1) gnuplot> f(x)=a*x+b

2) gnuplot> fit f(x) "tiempos_ejercicio_desc.dat" via a,b

a = 1.42371e-11 +/- 8.975e-12 (63.04%)
b = 2.34169e-06 +/- 3.097e-07 (13.23%)

3)gnuplot>plot "tiempos_ejercicio_desc.dat" , f(x)





Ejercicio 4: Mejor y peor caso

Retome el ejercicio de ordenación mediante el algoritmo de la burbuja. Debe modificar el código que genera los datos de entrada para situarnos en dos escenarios diferentes:

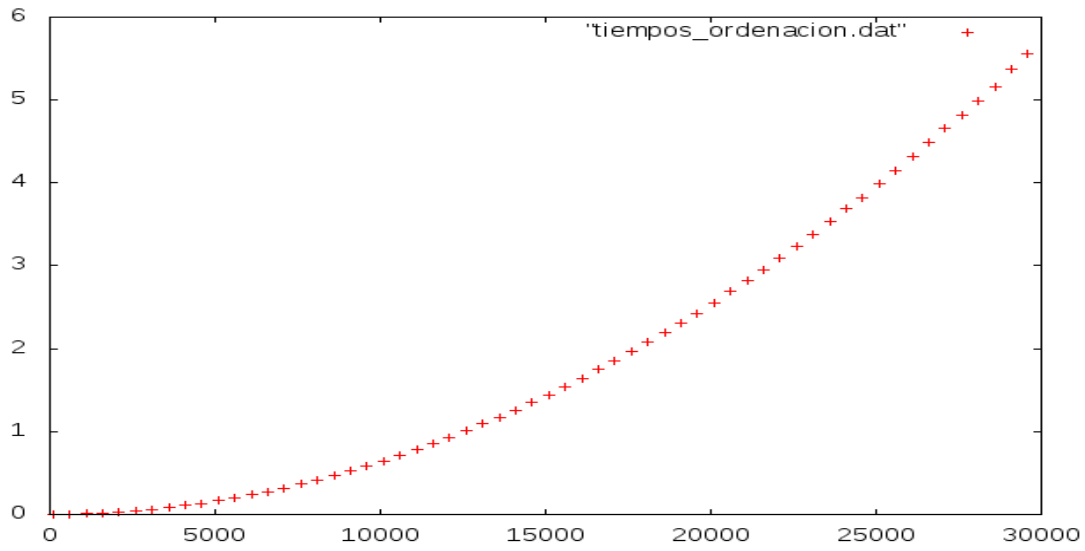
- **El mejor caso posible. Para este algoritmo, si la entrada es un vector que ya está ordenado el tiempo de cómputo es menor ya que no tiene que intercambiar ningún elemento.**

- **El peor caso posible. Si la entrada es un vector ordenado en orden inverso estaremos en la peor situación posible ya que en cada iteración del bucle interno hay que hacer un intercambio.**

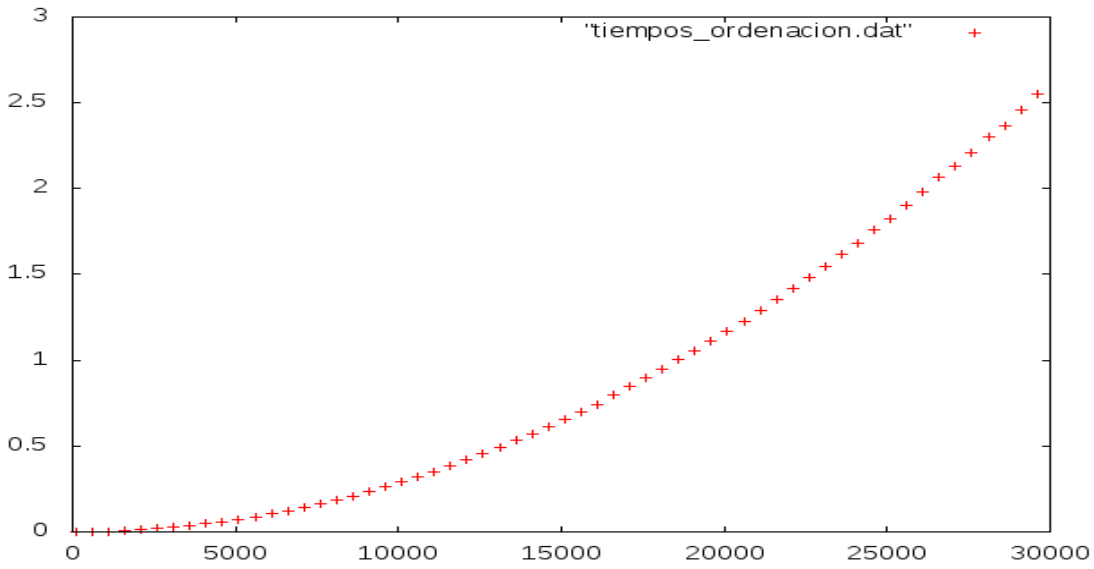
Calcule la eficiencia empírica en ambos escenarios y compárela con el resultado del ejercicio 1.

Peor de los casos: el ejercicio1 lo modifique, puesto que el ejemplo de la busqueda lineal venia implementado para el peor para el peor de los casos (y creí que tambian había que implementarlo para el peor de los casos). Luego este apartado y el ejercicio1 es igual.

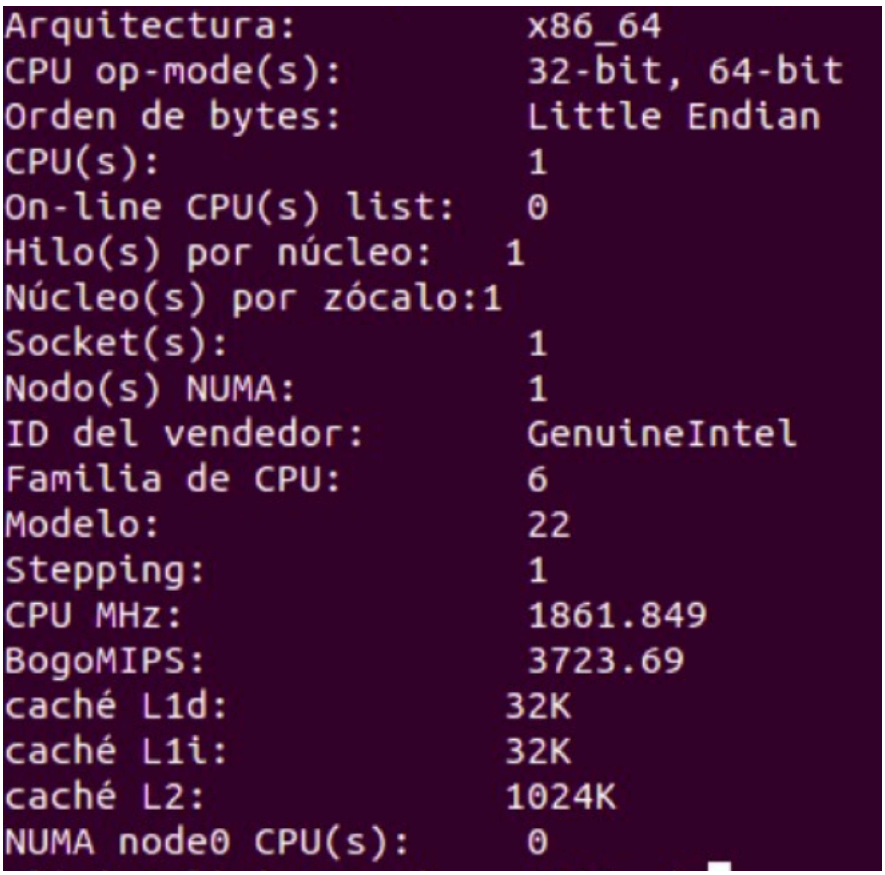
Creo el vector ordenado de mayor a menor.



Mejor de los casos: Creo el vector ordenado de menor a mayor.



Comparación: queda una grafica muy similar, pero se puede apreciar que en la esquina derecha superior para unos mismos tamaños, el tiempo es menor en el mejor caso.



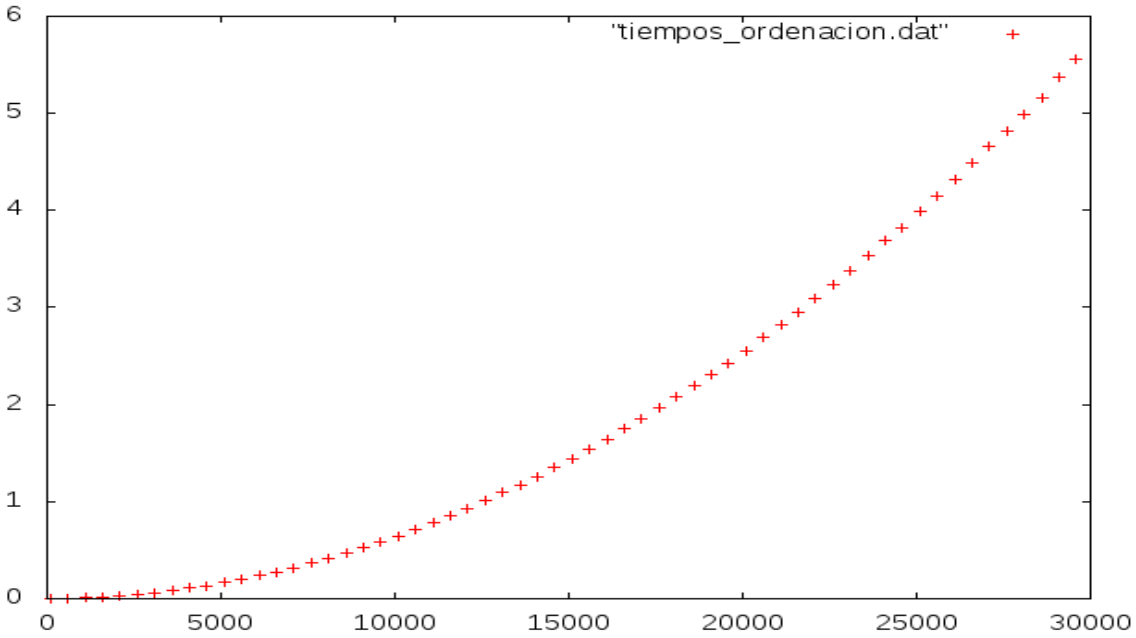
Ejercicio 6: Influencia del proceso de compilación

Retome el ejercicio de ordenación mediante el algoritmo de la burbuja. Ahora replique dicho ejercicio pero previamente deberá compilar el programa indicándole al compilador que optimice el código. Esto se consigue así:

g++ -O3 ordenacion.cpp -o ordenacion_optimizado

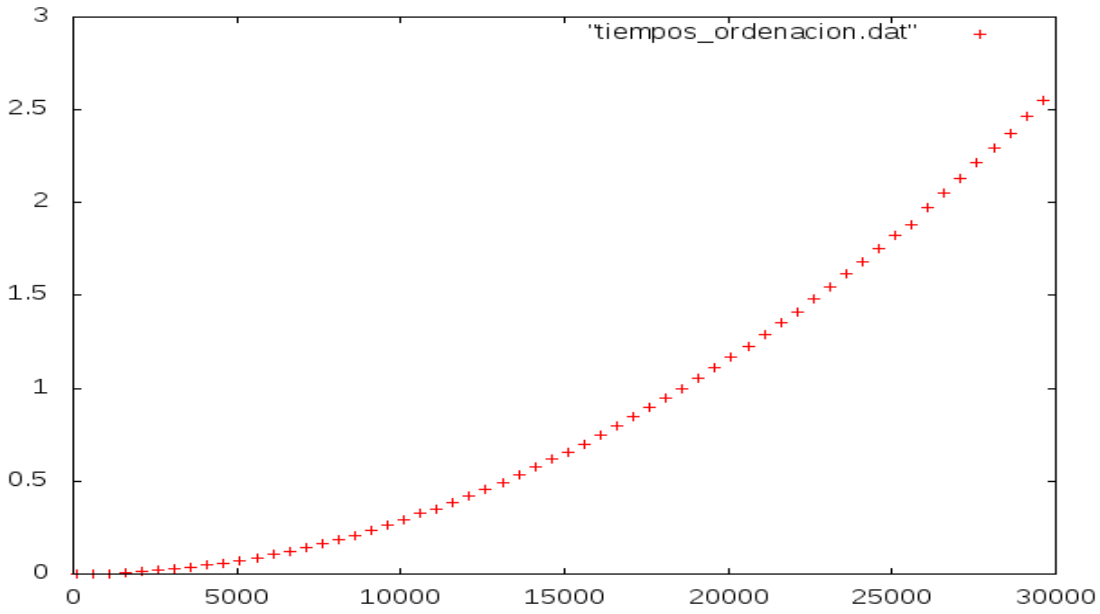
Compare las curvas de eficiencia empírica para ver cómo mejora esto la eficiencia del programa.

Eficiencia empírica codigo ordenacion sin optimizar:



Eficiencia empírica codigo ordenacion optimizado:

Tras compilar con: **g++ -O3 ordenacion.cpp -o ordenacion_optimizado**
y modificar en **ejecuciones_ordenacion.csh** el ejecutable por **ordenacion_optimizado** la grafica es la siguiente:



Comparación entre las curvas de eficiencia empírica para ver cómo mejora esto la eficiencia del programa:

Se puede observar gran diferencia en los tiempos, ya que en el programa optimizado el tiempo es la mitad comparado con el programa sin optimizar.