***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:

a) • Crear un fichero **ordenacion.cpp** con el programa completo para realizar una ejecución del algoritmo.

b) • 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.

c) • 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. (30000 tardaría mucho, así que mejor lo hago hasta 15000)
Pruebe a dibujar superpuestas la función con la eficiencia teórica y la empírica. ¿Qué sucede?

1.

a) O(n^2) - (usar apuntes de clase de teoría)

b)
```
#include <iostream>
#include <ctime> // Recursos para medir tiempos
#include <cstdlib> // Para generación de números pseudoaleatorios
using namespace std;


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];
```

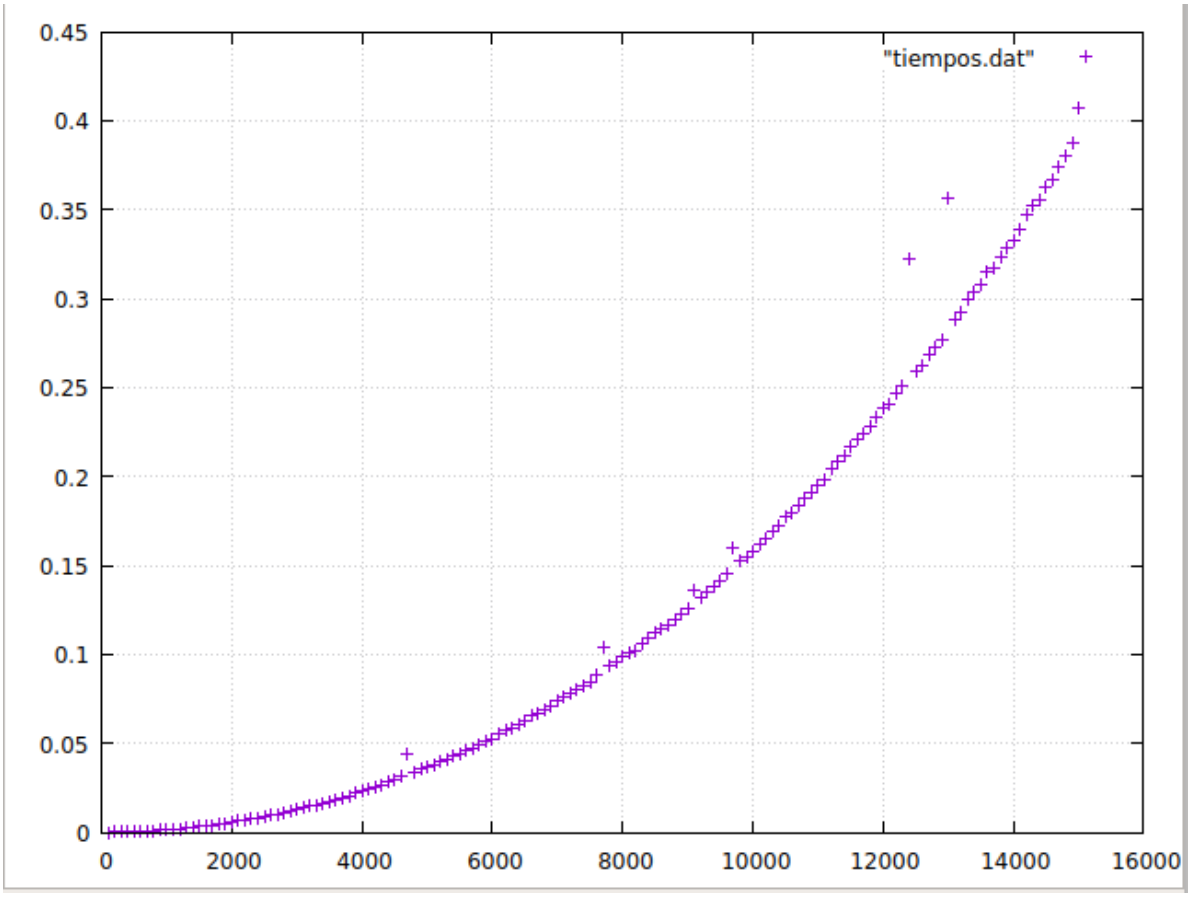
```
                    v[j+1] = aux;
                }
    }


    void sintaxis() {
            cerr << "Sintaxis:" << endl;
            cerr << " TAM: Tamaño del vector (>0)" << endl;
            cerr << " VMAX: Valor máximo (>0)" << endl;
            cerr << "Genera un vector de TAM números aleatorios en [0,VMAX[" <<
    endl;
            exit(EXIT_FAILURE);
    }

    int main(int argc, char * argv[]) {
     if (argc!=3) // Lectura de parámetros
            sintaxis();
     int tam=atoi(argv[1]); // Tamaño del vector
     int vmax=atoi(argv[2]); // Valor máximo
     if (tam<=0 || vmax<=0)
            sintaxis();

     // Generación del vector aleatorio
     int *v=new int[tam]; // Reserva de memoria
            srand(time(0)); // Inicialización generador números pseudoaleatorios
     for (int i=0; i<tam; i++) // Recorrer vector
            v[i] = rand() % vmax; // Generar aleatorio [0,vmax[

     clock_t tini; // Anotamos el tiempo de inicio
     tini=clock();


     ordenar(v,tam);

     clock_t tfin; // Anotamos el tiempo de finalización
     tfin=clock();
     // Mostramos resultados (Tamaño del vector y tiempo de ejecución en seg.)
     cout << tam << "\t" << (tfin-tini)/(double)CLOCKS_PER_SEC << endl;

     delete [] v; // Liberamos memoria dinámica
    }
```

c) El script utilizado es: (instalar previamente c-shell)

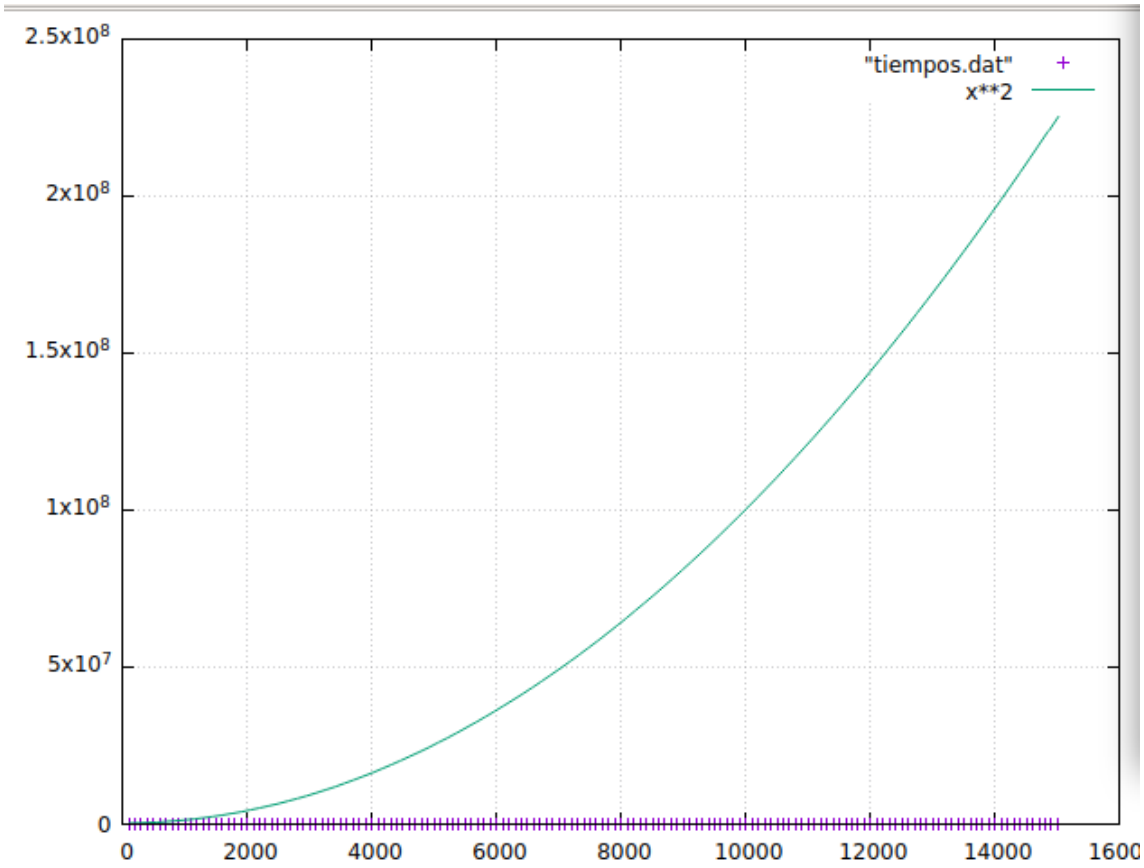
```
#!/bin/csh
@ inicio = 100
@ fin = 15000
@ incremento = 100
@ i = $inicio

echo > tiempos.dat
while ( $i <= $fin )
        echo Ejecución tam = $i
        echo `./ej1 $i 10000` >> tiempos.dat
        @ i += $incremento
End
```

Usamos gnuplot (instalar y entrar al menú con gnuplot en terminal)

d) plot “tiempos.dat”



e) (ojo, el cuadrado es **2) plot “tiempos.dat” , x**2



***Ejercicio** 2: Ajuste en la ordenación de la burbuja*

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

A: 1.99151e-09
B: -4.22412e-06
C: 0.00572174

Con gnuplot y

f(x) = a*x + b

fit f(x) “tiempos.dat” via a, b conseguiremos el valor de los coeficientes, y luego podemos ver las gráficas con plot "tiempos7.dat", f(x)

***Ejercicio** 4: Mejor y peor caso*

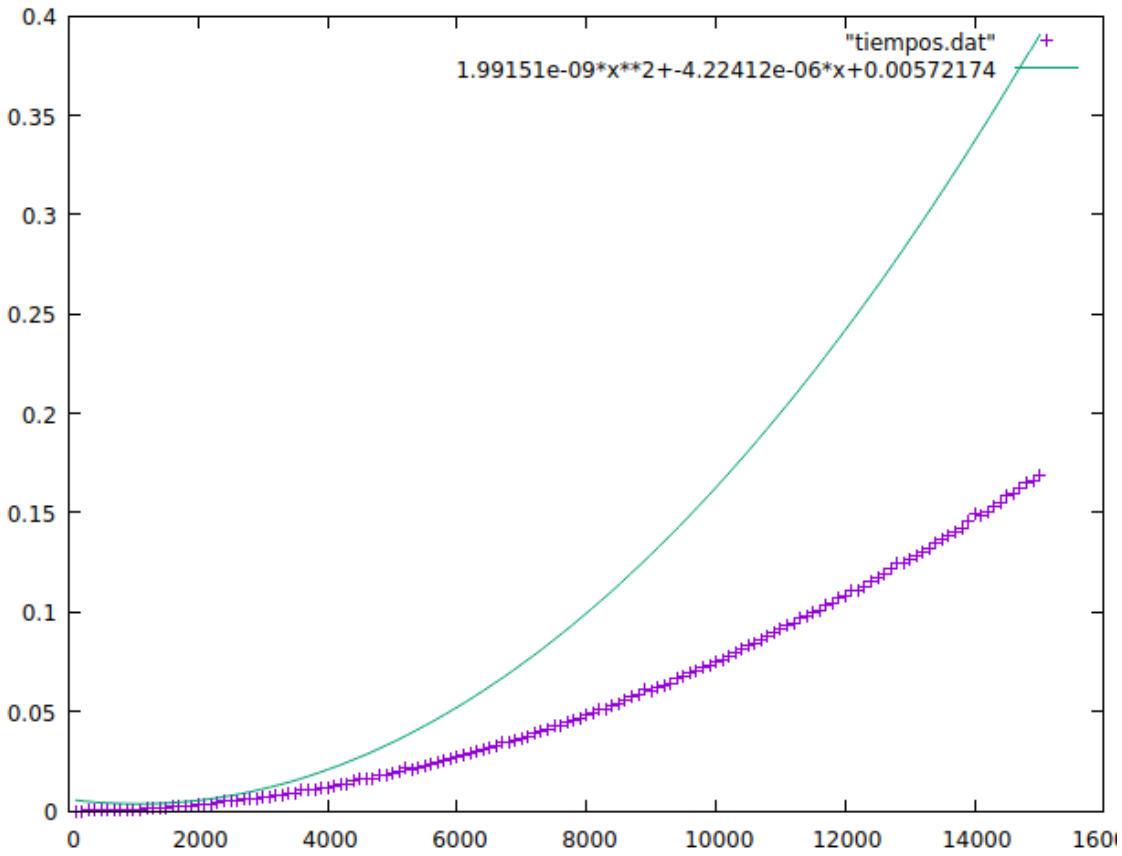
Retome el ejercicio de ordenación mediante el algoritmo de la burbuja. Debe modificar el código que genera los datos de entrada para situarnos en dos escenarios diferentes:

a) • 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. (en vez de generar random y asignar en cada interacción del bucle, generamos valores en orden creciente v[i] = i)

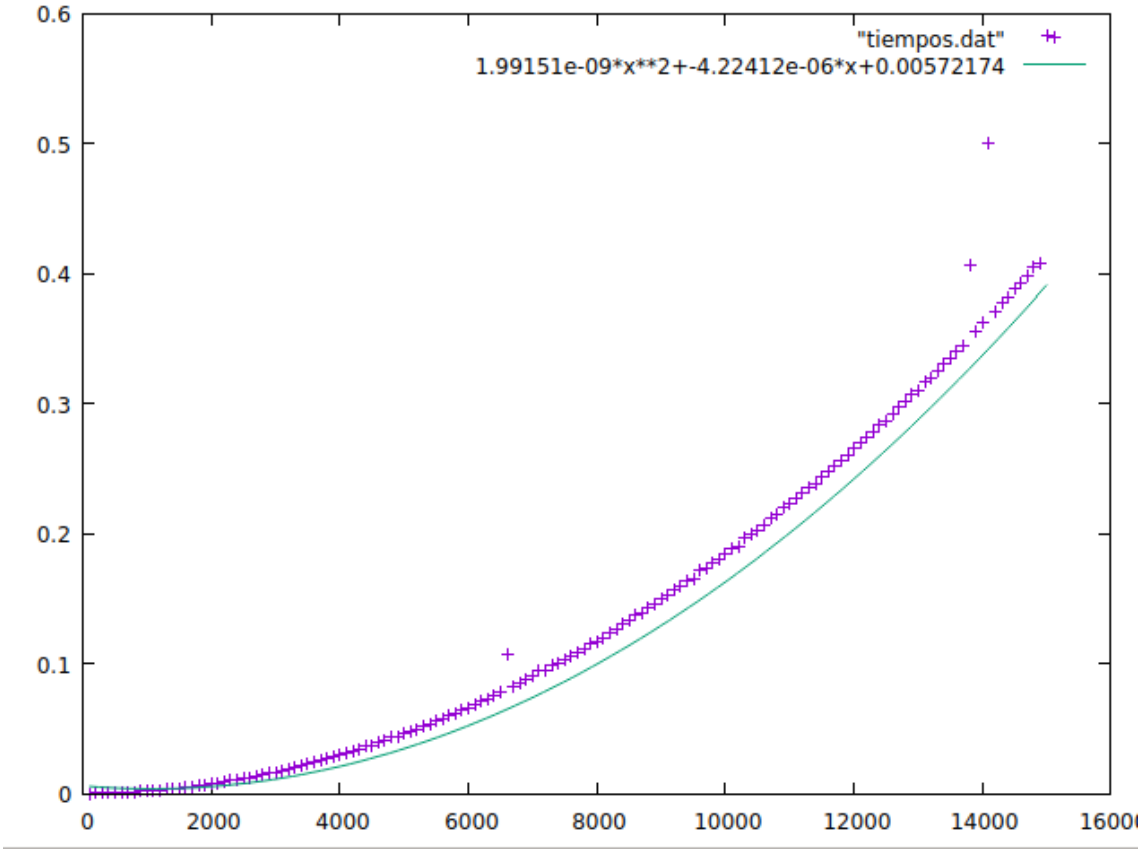
b) • El peor caso posible. Si la entrada es un vector ordenado en orden inverso estaremos en la peor situación posible ya que en cada iteración del bucle interno hay que hacer un intercambio. (v[i] = tam – i)

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

a)



b)


"tiempos.dat"
1.99151e-09*x**2+-4.22412e-06*x+0.00572174
0.6
0.5
0.4
0.3
0.2
0.1
0
0
2000
4000
6000
8000
10000
12000
14000