Práctica 2. Análisis experimental de algoritmos de Ordenación.

Índice

Práctica 2. Análisis experimental de algoritmos de Ordenación.

1. Objetivo de la práctica.

El objetivo principal de la práctica es estudiar empíricamente Algoritmos de Ordenación. Los análisis teóricos están incluidos en los contenidos del tema 5 de la asignatura.

El objetivo específico de la práctica es:

- Estudiar experimentalmente el comportamiento temporal de un algoritmo de ordenación comparándolo, además, con el de otros procedimientos de ordenación. Estudiaremos y compararemos los algoritmos de ordenación **`Burbuja`**, **`Inserción`** y **`Selección`**. Al finalizarlo, se dispondrá de los tiempos utilizados para ordenar un vector usando varios algoritmos, de modo que tendremos criterios objetivos para poder elegir la alternativa más adecuada.

2. Estudio práctico.

2.1. Cálculo del Tiempo Teórico

A partir de la expresión del algoritmo, se aplicarán las reglas conocidas para contar el número de operaciones que realiza un algoritmo. Este valor será expresado como una función de $T(n)$ que dará el número de operaciones requeridas para un caso concreto del problema caracterizado por tener un tamaño ***n***. El análisis lo realizaremos para los casos mejor, peor y medio.

2.2. Cálculo del Tiempo de la Eficiencia Empírica para el caso medio.

Se trata de llevar a cabo un estudio puramente empírico, es decir, estudiar experimentalmente el comportamiento de cada algoritmo para el **caso medio** exclusivamente. Para ello mediremos los recursos empleados (tiempo) para cada tamaño dado de las entradas. Una implementación posiblemente aceptable para medir el caso medio es realizar varias pruebas para un mismo tamaño de las entradas y realizar la media aritmética de las medidas.

En el caso de los algoritmos de ordenación el tamaño viene dado por el número de componentes del vector a ordenar (**n**).

2.1.1. Cálculo del tiempo.

Para obtener el tiempo empírico de una ejecución de un algoritmo lo haremos como se especificó en la práctica 1, por tanto podemos utilizar la clase Mtime proporcionada.

2.1.2. Cómo Mostrar los Datos

Para mostrar la eficiencia empírica haremos uso de los ficheros (tablas) que recojan el tiempo invertido para el caso medio de cada algoritmo (ver código y ejecución).

También utilizaremos la representación gráfica realizada a partir de los datos almacenados en los ficheros.

3. Descripción del problema

Este estudio de costes se va a realizar sobre *vectores* de enteros generados aleatoriamente.

3.1. Definición de la clase base `ConjuntoInt`.

La definición de la clase `ConjuntoInt`, para que se pueda especificar (y pueda variar) el tamaño máximo del vector, es la proporcionada en la Práctica 1, modificada y adaptada para esta práctica.

3.2. La clase AlgoritmosOrdenacion.

La clase **`AlgoritmosOrdenacion`** define los Algoritmos de Ordenación para ordenar un vector de enteros, ***`v`***, de un cierto tamaño **`size`** en orden ascendente. Define la interfaz de los siguientes métodos de Ordenación en vectores:

- Burbuja.
- Inserción.
- Selección.

```
class AlgoritmosOrdenacion{
public:
  AlgoritmosOrdenacion();
  ~AlgoritmosOrdenacion();
/*
 * Función ordenaBurbuja, implementa el método de ordenación Burbuja
 * param v: el array de enteros a ordenar
 * param size: tamaño del array de enteros a ordenar
 */
  void ordenaBurbuja(int v[], int size);
/*
 * Función ordenaInsercion, implementa el método de ordenación por Inserción
 * param v: el array de enteros a ordenar
 * param size: tamaño del array de enteros a ordenar
 */
  void ordenaInsercion(int v[], int size);
/*
 * Función ordenaSeleccion, implementa el método de ordenación por Selección
 * param v: el array de enteros a ordenar
 * param size: tamaño del array de enteros a ordenar
 */
  void ordenaSeleccion(int v[], int size);
};
```

NOTA IMPORTANTE-> La implementación de los métodos hay que realizarlos siguiendo el seudocódigo dado a continuación:

```
procedimiento burbuja(T[1..n])
   para i← n-1 hasta 1 hacer
      para j ← 1 hasta i hacer
            si T[j] > T[j+1] entonces
               auxiliar ← T[j]
               T[j] ← T[j+1]
               T[j+1] ← auxiliar
            fsi
      fpara
   fpara
fprocedimiento
```

```
procedimiento inserción(T[1..n])
   para i← 2 hasta n hacer
      x ← T[i]
      j ← i-1
      mientras j>0 AND x < T[j] hacer
         T[j+1] ← T[j]
         j ← j-1
      fmientras
      T[j+1] ← x
   fpara
fprocedimiento
```

```
procedimiento selección(T[1..n])
   para i← 1 hasta n-1 hacer
      posminimo ← i
      para j ← i +1 hasta n hacer
            si T[j] < T[posminimo] entonces
              posminimo ← j
            fsi
      fpara
      auxiliar ← T[posminimo]
      T[posminimo] ← T[i]
      T[i] ← auxiliar
   fpara
fprocedimiento
```

3.3. La clase Graficas.

La clase **Graficas** contiene métodos para guardar las gráficas de los resultados, es decir, crea los ficheros por lo lotes (comandos) para generar los ficheros gráficos correspondientes a:

- Caso medio de un algoritmo de ordenación con ajuste a su Orden de complejidad.
- Casos medios de dos algoritmos de Ordenación.

```
/*
 * Clase Graficas, contiene métodos para guardar las gráficas de los resultados,
 * es decir, crea los ficheros por lo lotes (comandos) para generar los ficheros
 * gráficos que corresponda.
 */
#ifndef _GRAFICA
#define _GRAFICA
#include <string>
#include <iostream>
using namespace std;

class Graficas
{
public:
/*
* Configura los parámetros para el fichero de comandos y dibuja la gráfica del caso
* medio de un método de ordenación y su ajuste a la función correspondiente.
* param método: nombre del método de ordenación.
* param método: orden de complejidad del método de ordenación.
*/
void   generarGraficaMEDIO(string metodo,int orden);

/*
* Configura los parámetros para el fichero de comandos
* y dibuja la gráfica de dos métodos de ordenación.
* param metodo1: primer método de ordenación.
* param metodo2: segundo método de ordenación.
 */
void generarGraficaCMP(string metodo1,string metodo2);
};
#endif
```

3.4. Estudio experimental de la eficiencia en el caso medio de un algoritmo

En la práctica anterior (práctica 1) se determinó que el estudio experimental de la eficiencia de un algoritmo conlleva la elaboración de conjuntos de prueba que reflejen condiciones diferentes de funcionamiento del mismo (caso en que el algoritmo se comporta mejor, en que se comporta peor, aleatorio, etc.) realizado para volúmenes de datos, o tallas, crecientes del problema. A medida que se generan los distintos casos de prueba, o bien posteriormente, se somete el, o los algoritmos, a dichos casos, determinándose para cada uno de ellos una medida del esfuerzo computacional necesario para su resolución (dicha medida suele ser el tiempo de ejecución).

En el caso de **estudios del tiempo promedio** será necesario generar, para cada posible talla, diferentes instancias aleatorias, de forma que la medida final sería un promedio de las efectuadas individualmente

Por último, los resultados se tabulan y se presentan adecuadamente en pantalla y se graban en **ficheros de datos** y **ficheros gráficos**.

4. Actividades a desarrollar.

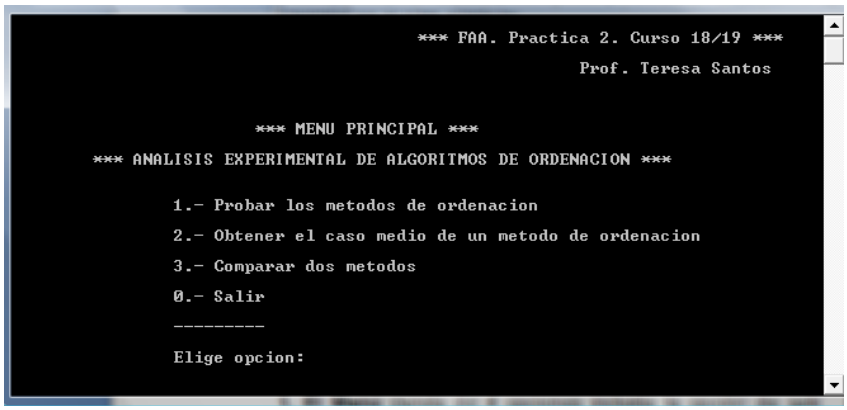
Para el desarrollo de esta práctica, Análisis experimental de los Algoritmos de Ordenación, se detalla y se proponen para su realización las siguientes actividades:

4.1. Programa principal.

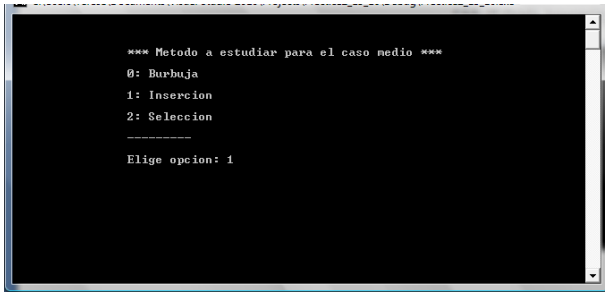
Completar el programa principal incorporando los menús como se especifica a continuación.

El método ***main*** proporciona las opciones del menú para verificar que los métodos de ordenación implementados funcionan correctamente, calcular su coste en el caso medio y estudiar cual de los métodos es el más eficiente.

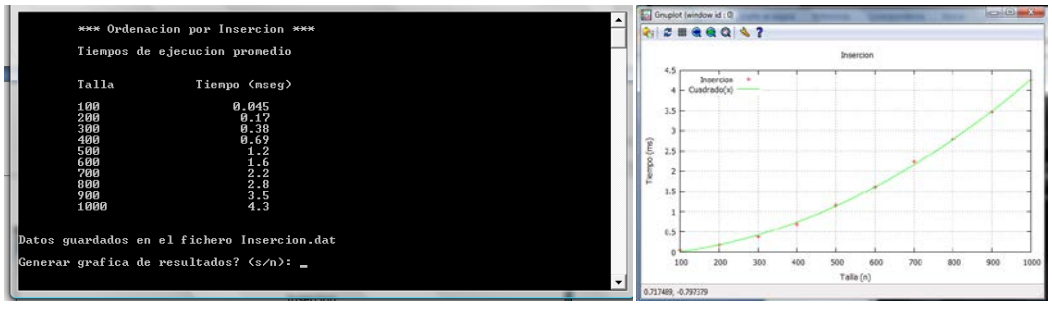
El programa principal está compuesto por una cabecera con el **nombre del alumno/a** (donde pone Prof. poned Alumno/a) y un menú cuya estructura y descripción es la siguiente:

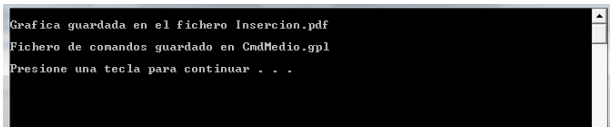


1. **El Menú** consta de 4 opciones incluida la opción de salir. El programa no finalizará hasta que la opción salir sea seleccionada.
   - **Opción 1**: Esta opción **comprueba** el funcionamiento de todos los métodos de Ordenación implementados en la clase `AlgoritmosOrdenacion.`
   - **Opción 2**: Esta opción abrirá un submenú para efectuar el cálculo de los tiempos en el **caso medio** para uno de los métodos de Ordenación y realizar la **gráfica** comparativa con su función de orden O.
   - **Opción 3**: Esta opción abrirá un submenú para efectuar el cálculo de los tiempos en el **caso medio para dos de los métodos** de ordenación (Búsqueda) y realizar la **gráfica** comparativa de ambos.
   - **Opción 0**: Esta opción hace que el programa vuelva la menú principal.
3. Los **submenús** correspondientes son los siguientes:
   - Submenú de la **Opción 2:**

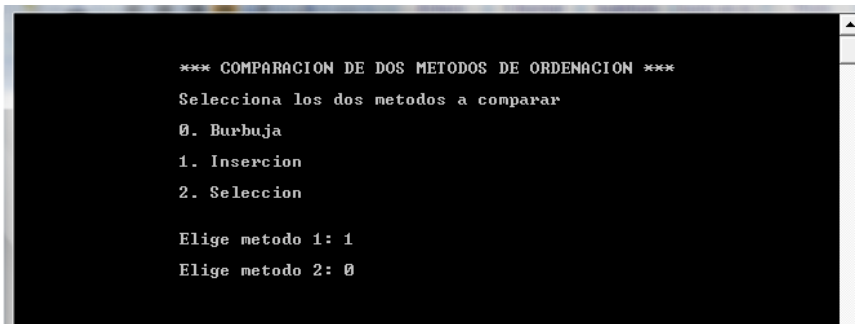


   - Como resultado de una opción 1 (Inserción). por ejemplo, se mostrará la información siguiente:

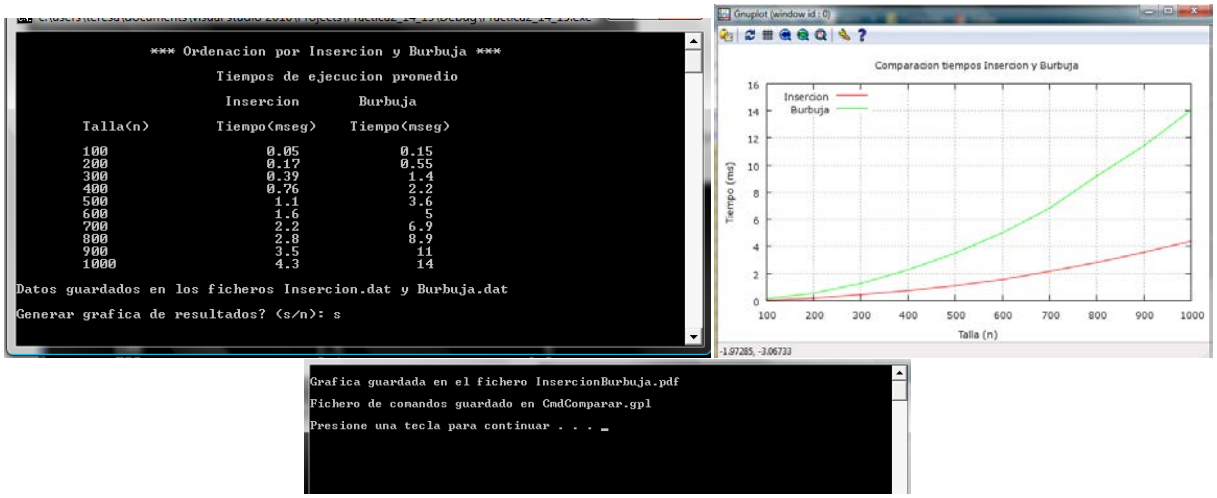




- Submenú de la **Opción 3**



- Como resultado de una opción, 1 (Inserción) y 0 (Burbuja) por ejemplo, se mostrará la información siguiente:



4.2. Implementación de los algoritmos.

En la clase **`AlgoritmosOrdenacion`** tenéis que implementar los métodos de ordenación según los seudocódigos proporcionados.

4.3. Comprobación de los métodos de ordenación.

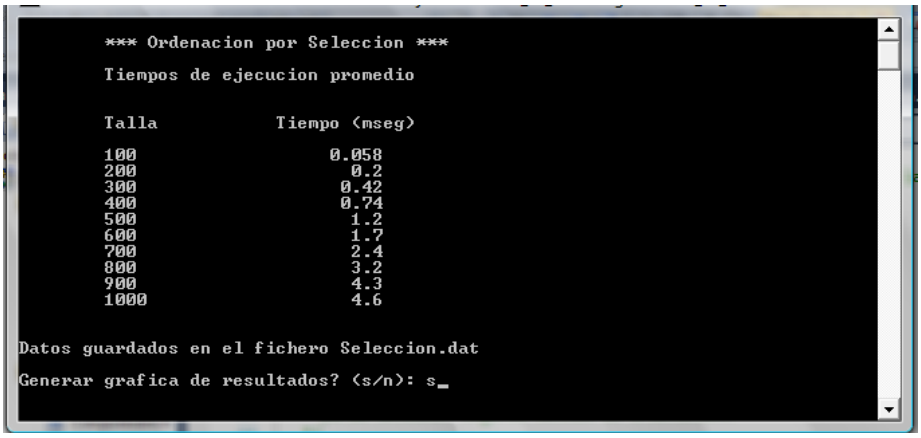
En la web de la asignatura se encuentra la clase **`TestOrdenacion`** que se habrá de incluir dentro del proyecto *Practica2*. En esta clase está implementado un método, `comprobarMetodosOrdenacion()`, que se encarga de comprobar si los algoritmos de ordenación funcionan correctamente. Tenéis que ver cómo funciona y comprobarlos.

4.4. Cálculo del caso medio para los métodos de ordenación

La clase `TestOrdenacion`, también calcula el coste para el caso medio de los métodos de ordenación y muestra el resultado por pantalla del método elegido de forma tabulada, permite guardar los mismos en un fichero de datos y realizar la gráfica de los resultados y guardarla en un fichero junto con el ajuste de la correspondiente función de orden O prevista de forma teórica. Tenéis que implementar esta tarea en el método **`casoMedio`** el cual recibe el nº del método de ordenación a estudiar como parámetro. La implementación debe seguir los siguientes pasos:

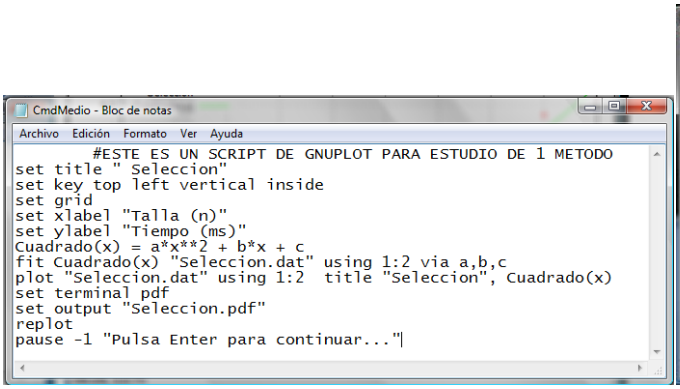
1. Almacenar y mostrar por pantalla el tiempo de ejecución en promedio de dicho método para vectores de enteros generados aleatoriamente, y con tallas sucesivamente crecientes: 100, 200, …, 1000.,
2. La clase `TestOrdenacion` dispone de un método que es de gran utilidad para esta tarea:
   - **`ordenarArrayDeInt`**: ordena un *array* de enteros para una talla determinada utilizando el método de ordenación indicado, y devuelve el tiempo empleado en el proceso. Para cada una de estas tallas se ejecutará el algoritmo de ordenación un cierto número de veces definido por

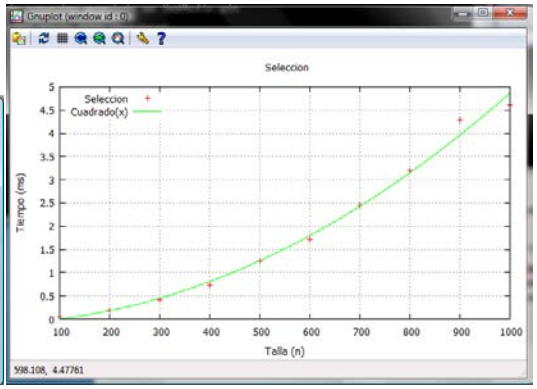
una constante, `NUMREPETICIONES=10` (por ej.). De esta forma se podrá obtener el coste medio como la media aritmética de las medidas tomadas. La salida del programa deberá ser tabulada, con un formato legible, similar al que se muestra a continuación:



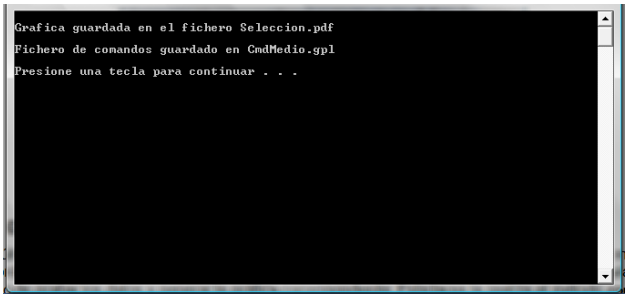
3. Se mostrará el nombre del fichero en el que se han grabado los datos, que tiene que ser el mismo nombre que el algoritmo al que corresponda.
4. Finalmente mostrará una opción para dibujar y grabar la gráfica resultante según apartado siguiente:
5. La clase Graficas contiene el método **`generarGraficaMEDIO`** utilizado para crear el fichero de comandos (.gpl) para generar el fichero (.pdf) que contiene la gráfica correspondiente a los datos del fichero creado en el apartado anterior (.dat). Los contenidos de los ficheros se muestran a continuación

   5.1. Fichero por lotes **"CmdMedio.gpl"** y su correspondiente fichero gráfico "**Selección.pdf**":





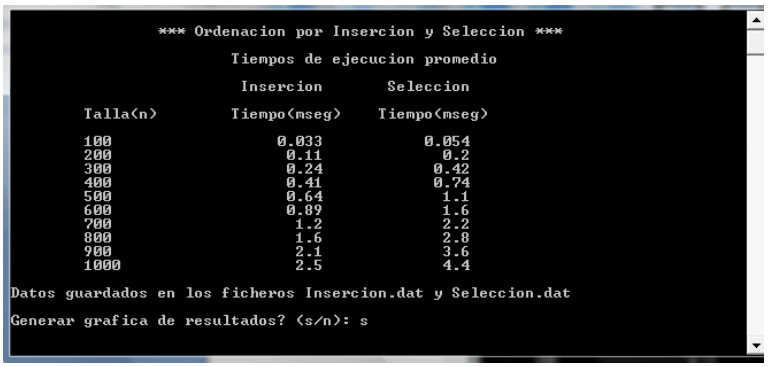
   5.2. Informe visualizado en pantalla:



4.5. Comparación de los métodos de ordenación

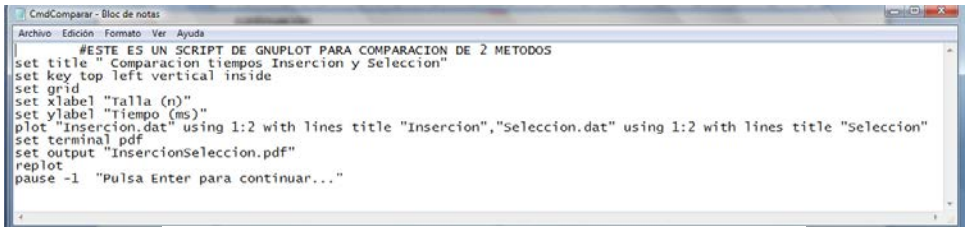
Por último, la clase `TestOrdenacion`, también se encarga de comparar el coste de dos de los métodos de ordenación que recibe como parámetros, mostrando el resultado por pantalla de forma tabulada así como de grabar los datos y generar la gráfica correspondiente. Esta tarea la realiza el método **`comparar`** de la clase `TestOrdenacion` que tenéis también por tanto que implementar, el cual:

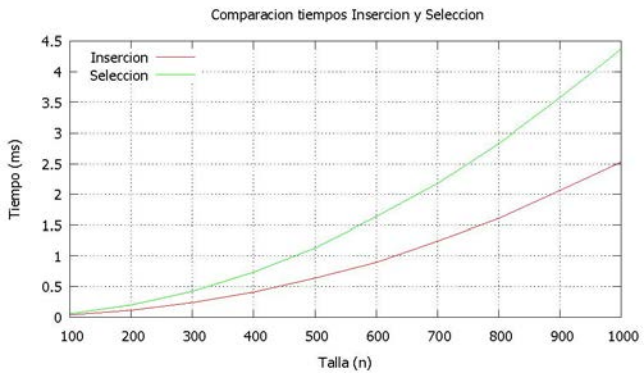
1. Almacena y muestra por pantalla el tiempo de ejecución en promedio de dichos métodos para *vectores* de enteros generados aleatoriamente, y con tallas sucesivamente crecientes: 100, 200, …, 1000., El método `ordenarArrayDeInt` de la clase `TestOrdenacion` también es de utilidad para esta tarea. Al igual que en el caso anterior la salida del programa deberá ser tabulada, con un formato legible, similar al que se muestra a continuación:



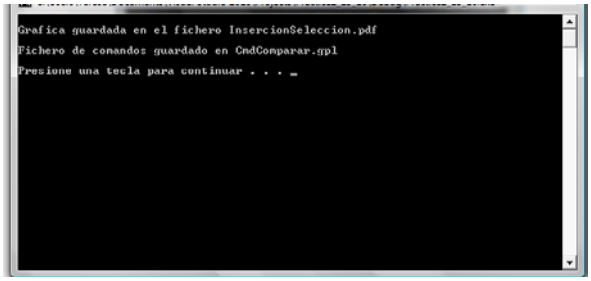
2. Se mostrará el nombre del fichero en el que se han grabado los datos, que tiene que ser la concatenación de los nombres de los algoritmos comparados.
3. Finalmente mostrará una opción para dibujar y grabar la gráfica resultante según apartado siguiente:
4. La clase Graficas contiene el método **generarGraficaMEDIO** utilizado para crear el fichero de comandos (.gpl) para generar el fichero (.pdf) que contiene la gráfica correspondiente a los datos del fichero creado en el apartado anterior (.dat). Los contenidos de los ficheros se muestran a continuación:

   4.1. Fichero por lotes **"CmdComparar.gpl"** y su correspondiente fichero gráfico "**Selección.pdf":**





   4.2. Informe visualizado en pantalla:



5. Representación e interpretación de los resultados gráficos.

Habitualmente, la interpretación de los resultados obtenidos mediante un proceso de temporización similar al realizado se facilita representando los resultados de forma gráfica. Para ello, se sitúan en abscisas valores crecientes de la talla, mientras que en ordenadas se representan los tiempos de ejecución. Además, para poder prever tiempos de ejecución fuera del rango de tallas medido, es conveniente ajustar los resultados mediante una función matemática de forma que, sustituyendo en la misma los valores de talla deseados, sea posible deducir el comportamiento temporal requerido.

Podemos realizar la representación gráfica de los resultados mediante el uso de hojas de cálculo (*OpenOffice*, por ejemplo), o de programas más especializados de interpretación y presentación gráfica como *gnuplot*, que será el utilizado en esta práctica (un resumen del mismo está en el Anexo 1. Representación gráfica de la práctica_1).

En esta práctica obtenemos **dos** tipos de resultados gráficos que además se grabarán en fichero en disco:

1. Gráfica correspondiente al **tiempo promedio** de un método de ordenación con **ajuste** a la función correspondiente al orden O según estudio teórico.
2. Gráfica **comparativa** de los tiempos promedios de dos métodos de ordenación.

- Para el **primer caso**, por ejemplo, para obtener la gráfica correspondiente al método de Inserción con gnuplot se pueden seguir los siguientes pasos:

1. Guardar los resultados de las mediciones (opción 2 del submenú) en un fichero llamado, por ejemplo, Insercion.dat.

2. Crear el fichero de comandos (por ejemplo “CmdMedio.gpl”).

3. Escribir las líneas de comandos necesarios en el fichero anterior (.gpl) para que la apariencia (título, ejes, etc) de la gráfica sea la mostrada en los ejemplos.

4. Ajustar los resultados a un polinomio de segundo grado para poder predecir el comportamiento del método para tallas más grandes. Para ello se crea una nueva función con tres parámetros (a, b y c) Cuadrado(x) = a*x**2 + b*x + c

6. Ajustar con el comando fit de gnuplot el valor de dichos parámetros: **fit** Cuadrado(x) “Insercion.dat” using 1:2 via a,b,c

7. Mostrar la gráfica resultante (Figura 1), con los tiempos obtenidos y con la función de ajuste: **plot** “Insercion.dat” using 1:2, Cuadrado(x)

8. Guardar la gráfica en disco:

```
set terminal pdf
set output" Insercion.pdf."
replot
```

7. Grabar el fichero de comandos.
8. Finalmente, ejecutar gnuplot desde el programa (system (“CmdMedio.gpl”)), y se obtiene:

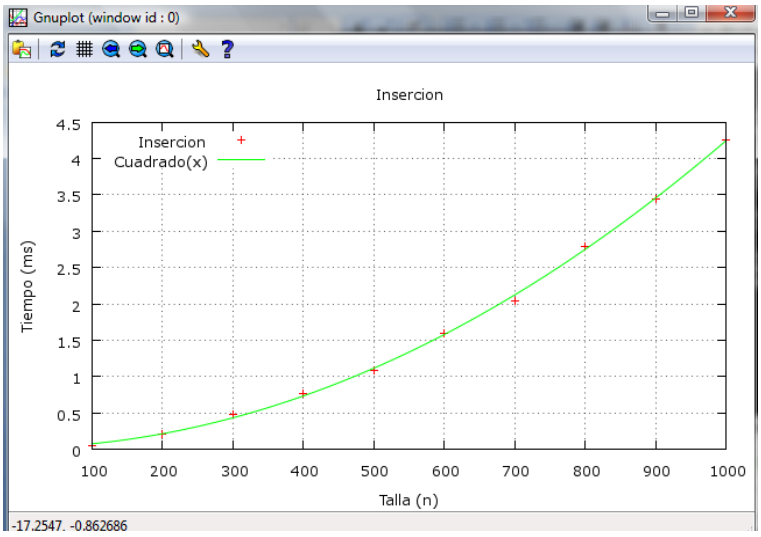


Figura 1: Inserción, tiempo promedio. Ajuste a una parábola.

Así se puede predecir el comportamiento temporal de cualquiera de los dos algoritmos mediante el uso de las funciones de ajuste. Calcular, por ejemplo, cuánto tiempo emplearían ambos algoritmos en ordenar una array de 1e+010 elementos.

- Para el **segundo** caso, por ejemplo, para obtener la **gráfica comparativa** correspondiente a los métodos de *Insercion* y *Burbuja* en el *gnuplot* se pueden seguir los siguientes pasos:

1. Guardar los resultados (opción 3 del menú) en los dos ficheros llamados, por ejemplo, *Insercion.dat* y *Burbuja.dat*

2. Crear el fichero de comandos (por ejemplo “CmdComparar.gpl”)

3. Escribir las líneas de comandos necesarios en el fichero anterior (.gpl) para que la apariencia (título, ejes, etc) de la gráfica sea la mostrada en los ejemplos.

4. Mostrar la gráfica resultante (Figura 2), con los tiempos obtenidos para ambos métodos:

plot "Insercion.dat" using 1:2 with lines,"Burbuja.dat " using 1:2 with lines

5. Guardar la gráfica en disco:

set terminal *pdf*

set output "*InsercionBurbuja.pdf*."

replot

6. Grabar el fichero de comandos

7. Ejecutar gnuplot desde el programa (system (“CmdComparar.gpl”)).

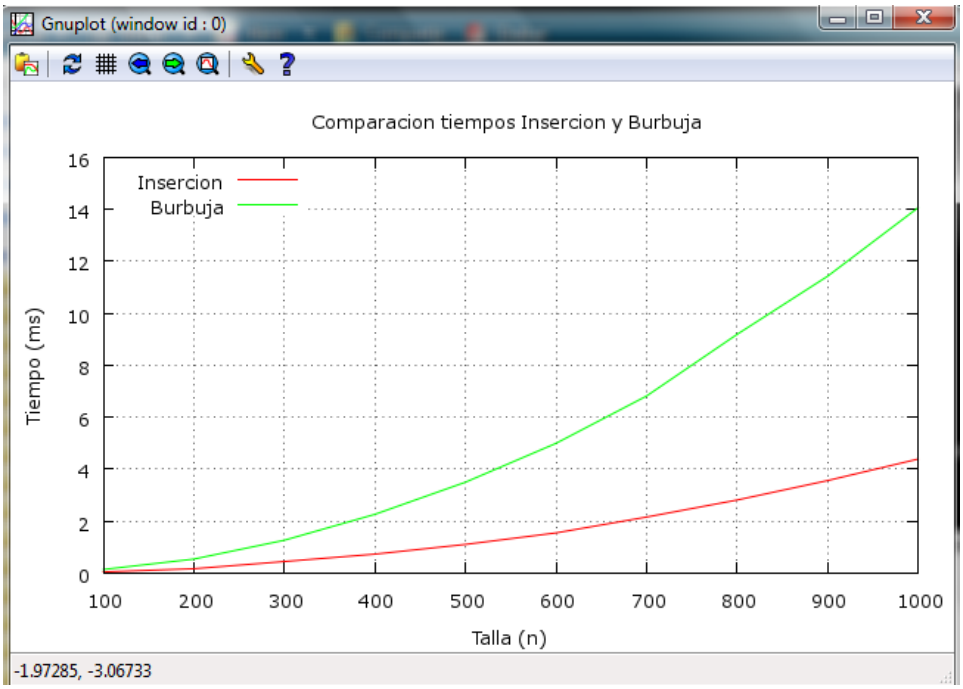


Figura 2: Inserción y Burbuja, tiempos promedios.

Así, mediante el análisis de la gráfica, se puede decidir la elección del método de ordenación adecuado.

6. Entrega de la práctica

La fecha de **entrega** es el domingo **5 de mayo**.

La entrega de la práctica se realizará por Moodle en la tarea puesta al efecto y tendrá el siguiente formato:

Crear y subir la carpeta ***Apellido1Apellido2.rar*** la cual debe contener:

- Una subcarpeta con los fuentes del programa (.cpp y .h)
- Una subcarpeta con el ejecutable y los datos.
- La memoria, de **12 páginas como máximo**, en formato pdf. con el esquema especificado a continuación.

Esquema para la memoria.

Índice

1. Portada (todos los autores)

2. Índice

3. Introducción. Algoritmos de ordenación.

4. Cálculo de los tiempos teóricos:

4.1. Pseudócódigos y análisis de coste

4.2. Tablas(ficheros) y Gráficas de coste

4.3. Conclusiones

5. Cálculo del tiempo experimental:

5.1. Tablas(ficheros) y Gráficas de coste

5.2. Conclusiones

6. Comparación de los resultados teórico y experimental.

7. Diseño de la aplicación.

(Mostrar un esquema gráfico global de la estructura de tipos de datos existentes. Detallar la descomposición modular del programa, qué módulos existen, cuál es la responsabilidad de cada uno y la relación de uso. Documentar cualquier otra decisión de diseño que pueda resultar de interés. Funcionamiento y explicación de los métodos implementados en la práctica.)

8. Conclusiones y valoraciones personales de la práctica.

Consideraciones para la memoria.

En las gráficas de coste habrá que tener en cuenta los siguientes puntos:

- Representación gráfica de los tiempos de ejecución obtenidos en la implementación del algoritmo y comparación con el coste teórico esperado analizado.
- Las gráficas deben tener el formato adecuado: leyenda de la gráfica, nombre de los ejes, etc.
- En el caso del tiempo promedio para un método la gráfica de tiempos debe ir acompañada de la curva teórica la cual, se obtendrá del análisis teórico realizado y con la ayuda de la función 'fit' de gnuplot
- Las gráficas deben ir explicadas: qué se observa en la gráfica, si coincide con lo esperado en teoría y de no ser así, por qué, es decir, un contraste teórico/experimental.

7. Anexo. Resumen de los pasos a seguir para realizar la práctica 2.

En la carpeta FicherosPractica2.rar tenéis las siguientes clases que tendréis que incorporar a vuestro proyecto ***ApellidoPractica2***:

- La clase ConjuntoInt, ficheros ConjuntoInt.h y ConjuntoInt.cpp
- La clase AlgoritmosOrdenacion, ficheros AlgoritmosOrdenacion.h y AlgoritmosOrdenacion.cpp
- La clase TestOrdenacion, ficheros TestOrdenacion.h y TestOrdenacion.cpp
- La clase Graficas, ficheros Graficas.h y Graficas.cpp
- La clase **Mtime**, ficheros **Mtime.h** y **Mtime.cpp.** Fichero donde está la función de medir el tiempo. Podéis ver un ejemplo de utilización en el método ordenarArrayDeInt de TestOrdenacion.
- El main es el fichero AnalisisAlgoritmos.cpp

1. Completar el programa principal incorporando los menús como se ha especificado anteriormente.
2. Completar los métodos `OrdenaBurbuja` `OrdenaInsercion` y `OrdenaSeleccion` de la clase `AlgoritmosOrdenacion` el cual lo tenéis que adaptar del seudocódigo especificado en esta memoria.
3. Completar los métodos **`casoMedio`** y **`comparar`** de la clase **TestOrdenacion.**
4. Completar los métodos **`generarGraficaMEDIO`** y **`generarGraficaCMP`** de la clase **Graficas.**
5. Una vez completado el código, crear en el directorio ***Apellido1Apellido2*** tres carpetas:
   - **fuentes**: con los fuentes .h y .cpp
   - **datos**: con el ejecutable y los datos
   - **memoria**: con el contenido de la memoria especificada.
6. Subir ***Apellido1Apellido2.rar***