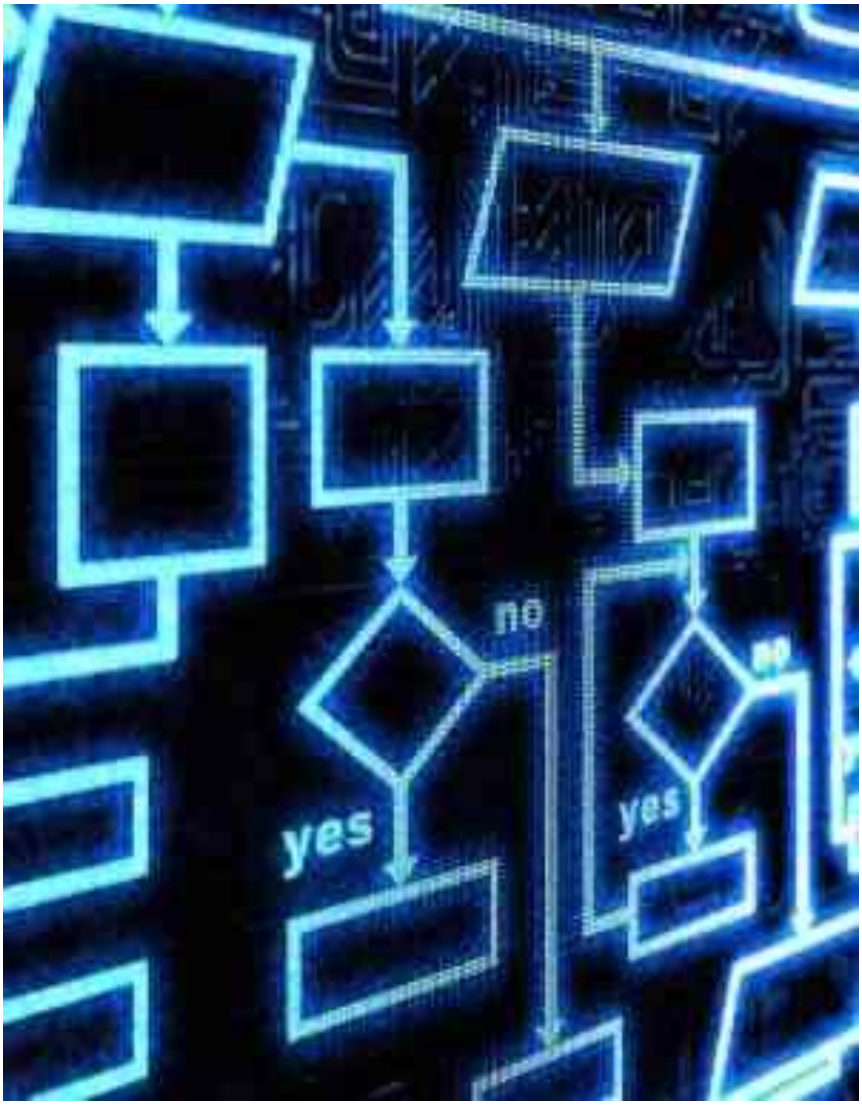


Análisis de algoritmos de ordenación

FAA

Por: Ismael Da Palma Fernández

ÍNDICE

3. Introducción. Algoritmo de ordenación

El objetivo de esta práctica es realizar el estudio teórico de tres algoritmos de ordenación, implementándolos en un lenguaje de programación (C++) y comparar su eficiencia empíricamente.

Al igual que en la práctica anterior, vamos a usar el Visual Studio 2019. Y adicionalmente hemos utilizado Gnuplot para realizar las gráficas de los resultados obtenidos.

4. Cálculo del tiempo teórico

4.1. Pseudocódigo y análisis de coste

Los algoritmos de ordenación que hemos utilizado son: Burbuja, Inserción y Selección.

➢ Burbuja: código y analisis de coste

```
void AlgoritmosOrdenacion :: ordenaBurbuja(int v[], int size)
{
//** ESCRIBIR PARA COMPLETAR LA PRACTICA **//
   for (int i = size-1; i > 1; i--)
   {
      for (int j = 0; j < i; j++)
      {
         if (v[j] > v[j + 1])
         {
            int aux;
            aux = v[j];
            v[j] = v[j + 1];
            v[j + 1] = aux;
         }
      }
   }
}
```

Este algoritmo realiza sucesivas pasadas, en cada una se recorre hasta el final de la tabla, arrastrando el mayor elemento, así se asegura que el elemento mayor queda al final del recorrido (recorrido que se va acortando a medida que más elementos están ordenados).

A menor densidad, más cerca del inicio se quedan.

De los tres algoritmos, es el más sencillo de implementar.

Eficiencia teórica: $T(n) \in O(n^2)$ *Para todos los casos*

- Caso mejor: Al estar el array ordenado, el número de comparaciones que realiza es de $O(n^2)$, por lo que no hay intercambios.
- Caso medio y peor: el número de comparaciones e intercambios es de $O(n^2)$.

$$\textbf{\textit{Caso peor, mejor y medio}}: C(n) = \sum_{i=1}^{n-1}\sum_{j=1}^{i} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)(n-1+1)}{2} = \frac{n(n-1)}{2}$$

$$\textbf{\textit{Caso peor}}: I(n) = \sum_{i=1}^{n-1}\sum_{j=1}^{i} 1 = \frac{n(n-1)}{2}$$

C(n) = Tiempo de las comparaciones

I(n) = Tiempo de los intercambios

$$\textbf{\textit{Caso mejor}}: I(n) = 0$$

$$\textbf{\textit{Caso medio}}: I(n) = \frac{n(n-1)}{4} \qquad \textbf{\textit{Caso medio}}: T(n) = C(n) + 3 \cdot I(n) = \frac{5}{4}n(n-1)$$

➢ Selección: código y analisis de coste

```
void AlgoritmosOrdenacion :: ordenaSeleccion(int v[], int size)
{
//** ESCRIBIR PARA COMPLETAR LA PRACTICA **//
   for (int i = 0; i < size - 1; i++)
   {
      int posminimo = i;
      for (int j = i + 1; j < size; j++)
          if (v[j] < v[posminimo])
              posminimo = j;
      int aux;
      aux = v[posminimo];
      v[posminimo] = v[i];
      v[i] = aux;
   }
}
```

El algoritmo de selección realiza una búsqueda lineal del elemento menor de la tabla y lo mueve a su posición correspondiente.
De esta forma, el array se va ordenando directamente con los elementos en sus posiciones finales y solo realiza un intercambio por cada pasada.

Eficiencia teórica: $T(n) \in O(n^2)$ *Para todos los casos*

- Caso mejor, medio y peor: En todos los casos el número de comparaciones es de $O(n^2)$ y el número de intercambios es de O(n).

$$\textbf{\textit{Caso peor, mejor y medio}}: C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

$$\textbf{\textit{Caso peor, mejor, medio}}: I(n) = \sum_{i=1}^{n-1} 1 = n - 1$$

C(n) = Tiempo de las comparaciones

I(n) = Tiempo de los intercambios

$$\textbf{\textit{Caso medio}}: T(n) = C(n) + 3 \cdot I(n) = \frac{n(n-1)}{2} + 3(n-1)$$

➢ Inserción: código y analisis de coste

```
void AlgoritmosOrdenacion :: ordenaInsercion(int v[], int size)
{
//** ESCRIBIR PARA COMPLETAR LA PRACTICA **//
   for (int i = 1; i < size; i++)
   {
      int x = v[i];
      int j = i - 1;

      while (j >= 0 && x < v[j])
      {
         v[j + 1] = v[j];
         j--;
      }
      v[j + 1] = x;
   }
}
```

Este algoritmo realiza el proceso inverso a Selección. En lugar de buscar el elemento para colocarlo en su posición, busca la posición que corresponde a cada elemento que encuentra.

Al final de cada pasada, aumenta el tamaño del subarray izquierdo dónde los elementos ya están ordenados. Suponiendo el primer elemento ordenado, comienza desde el segundo a intentar intercambiarlo con el primer elemento mayor que se encuentre recorriendo el vector de izquierda a derecha.

Eficiencia teórica: $T(n) \in O(n)$ *si el vector se encuentra ordenado*
$T(n) \in O(n^2)$ *en cualquier otro caso*

- Caso mejor: Si el vector ya está ordenado, le basta con una pasada para determinar que se encuentra ordenado. Su complejidad es de O(n).

- Caso medio y peor: El número de comparaciones es de $O(n^2)$ y el de intercambios es de O(n).

$$\textbf{\textit{Caso peor, mejor, medio}}: I(n) = \sum_{i=1}^{n-1} 1 = n - 1$$

$$\textbf{\textit{Caso peor}}: C(n) = \sum_{i=2}^{n-1} \sum_{j=1}^{i-1} 1 = \sum_{i=2}^{n} i - 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

$$\textbf{\textit{Caso mejor}}: C(n) = \sum_{i=2}^{n} 1 = n - 1$$

C(n) = Tiempo de las comparaciones

I(n) = Tiempo de los intercambios

$$\textbf{\textit{Caso medio}}: C(n) = \frac{\frac{n(n-1)}{2} + n - 1}{2} = \frac{(n-1)(n+2)}{4}$$

$$\textbf{\textit{Caso medio}}: T(n) = C(n) + 3 \cdot I(n) = (n-1)\frac{n+14}{4}$$

4.2. Conclusiones

Como se esperaba, el coste temporal del algoritmo depende de la talla del problema y tanto de la instancia. Los tres algoritmos lentos que estudiamos son de orden cuadrático, no obstante, algunos realizan menos instrucciones que otros.

El algoritmo de Selección resulta más eficiente que Burbuja puesto que solo realiza un intercambio por cada pasada. Igualmente, no detecta si el vector está ordenado. Por tanto, hace pasadas innecesarias.

Por orden de eficiencia: Burbuja << Selección << Inserción

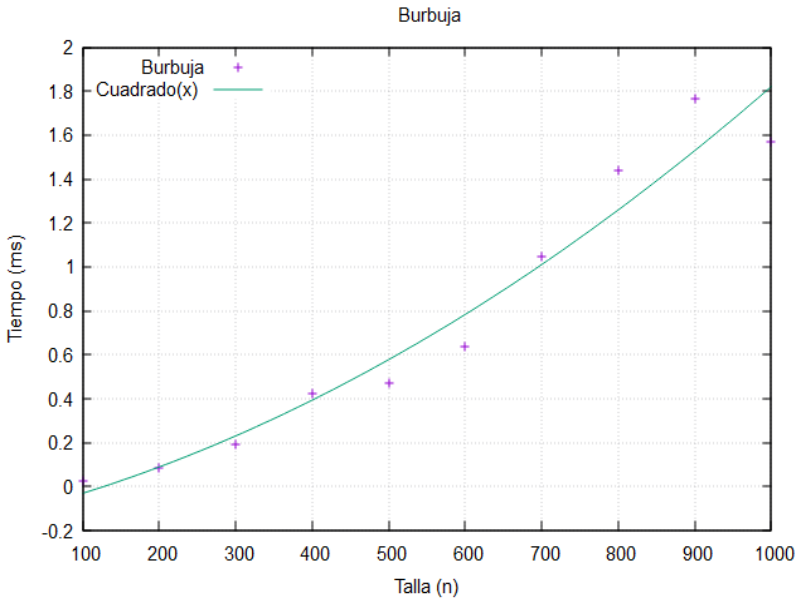
En cuanto a la complejidad espacial, todos los algoritmos son de $O(1)$, la cantidad de memoria consumida por el algoritmo no es proporcional al número de elementos a ordenar. En general, todos utilizan una variable auxiliar para los intercambios.

5. Cálculo del tiempo experimental

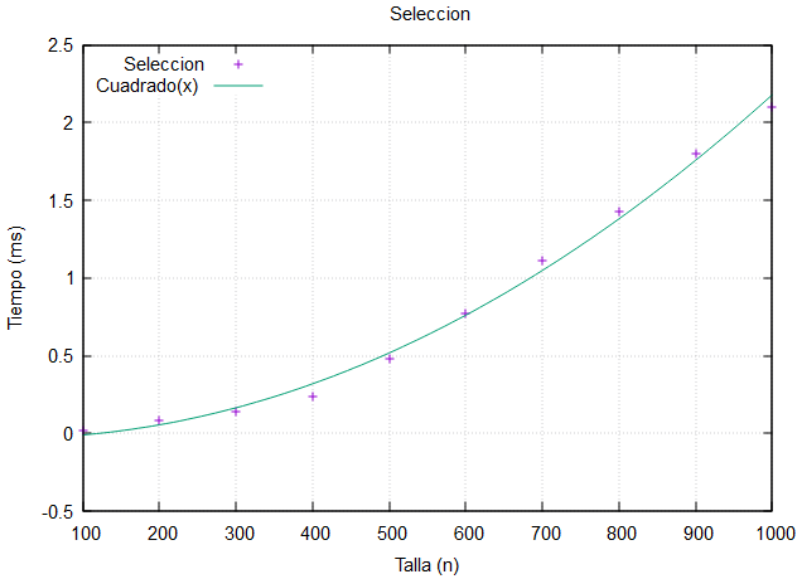
Para esta práctica sólo hemos tenido que calcular el tiempo medio de cada uno de los algoritmos de ordenación, para ello hemos repetido cada talla de cada algoritmo un número de veces en concreto que viene indicado en el fichero “Constantes.h”.

5.1. Tablas y gráficas de coste

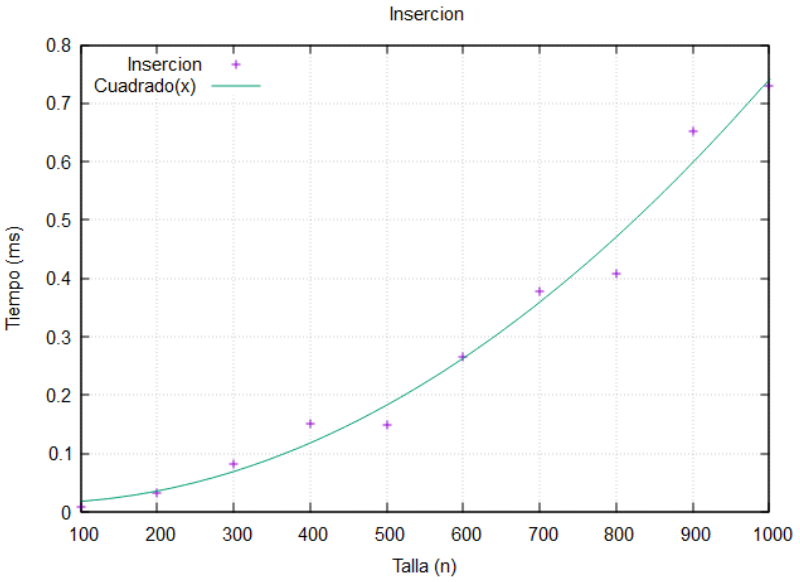
```
C:\Users\ismae\source\repos\Practica_2_FAA\Debug\Practica_2_FAA.exe

                *** Ordenacion por Burbuja ***

                Tiempos de ejecucion promedio


                Talla           Tiempo <mseg>

                100                     0.028
                200                     0.087
                300                      0.19
                400                      0.42
                500                      0.47
                600                      0.64
                700                         1
                800                       1.4
                900                       1.8
                1000                      1.6

Datos guardados en el fichero Burbuja.dat

Generar grafica de resultados? (s/n): s
```




```
C:\Users\ismae\source\repos\Practica_2_FAA\Debug\Practica_2_FAA.exe

                *** Ordenacion por Seleccion ***

                Tiempos de ejecucion promedio


                Talla           Tiempo <mseg>

                100                     0.017
                200                     0.085
                300                      0.14
                400                      0.24
                500                      0.48
                600                      0.77
                700                       1.1
                800                       1.4
                900                       1.8
                1000                      2.1

Datos guardados en el fichero Seleccion.dat

Generar grafica de resultados? (s/n):
```



```
C:\Users\ismae\source\repos\Practica_2_FAA\Debug\Practica_2_FAA.exe

              *** Ordenacion por Insercion ***

              Tiempos de ejecucion promedio


              Talla           Tiempo <mseg>

              100                0.0093
              200                 0.032
              300                 0.082
              400                  0.15
              500                  0.15
              600                  0.27
              700                  0.38
              800                  0.41
              900                  0.65
              1000                 0.73

Datos guardados en el fichero Insercion.dat

Generar grafica de resultados? (s/n):
```



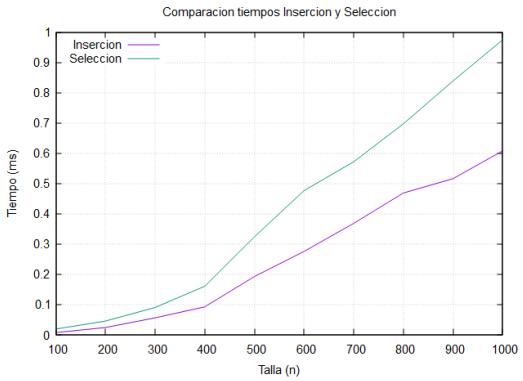
5.2. Conclusiones

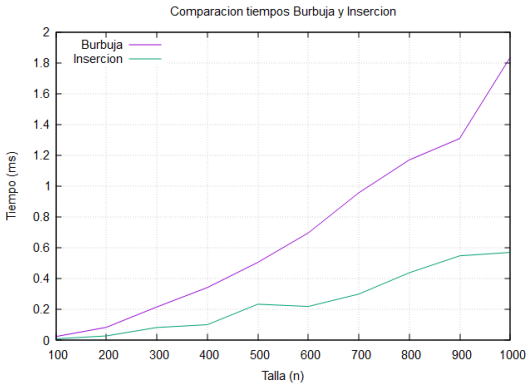
Los resultados individuales muestran un grado tolerable de dispersión, sin elementos anómalos. Cabe apreciar una mejor aproximación que en los algoritmos de complejidad lineal, vistos en la práctica 1.

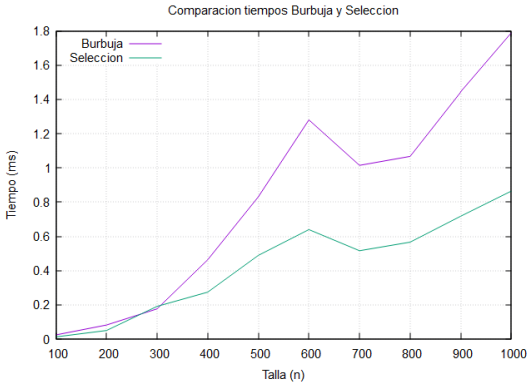
6. Comparación de los resultados teórico y experimental

Los resultados experimentales reflejan la misma relación entre los algoritmos.

El modelo de coste temporal empírico analizado describe correctamente al algoritmo, puesto que las fórmulas se ajustan a los resultados obtenidos.







Como podemos ver en las gráficas el algoritmo de inserción es el más rápido de los tres, le sigue después el algoritmo de Selección y el más lento de todos es el de Burbuja.

Como ocurría en la practica anterior, pueden ocurrir pequeños casos anómalos, como puede verse en la grafica de "Burbuja y Selección", que por un instante el algoritmo de Burbuja es más rápido que el de Selección siendo el Burbuja el más lento de los tres algoritmos de ordenación.

7. Diseño de la aplicación 

El método principal con la interfaz de menús y submenús está en el **"Principal.cpp"**. Desde allí se llama a los métodos de "TestOrdenacion" (que realiza los casos medio de cada algoritmo o de dos de ellos). Además, "TestOrdenacion" usa la clase "ConjuntoInt" para la ejecución de los algoritmos a estudiar, "Mtime" para medir el tiempo transcurrido de cada uno y la clase "Graficas" para la generación de los ficheros de gnuplot (fichero con terminación .gpl). El Principal también hace uso de la clase "Constantes.h".

"ConjuntoInt" tiene la misma funcionalidad que la práctica 1, permite crear arrays de enteros aleatorios y, posteriormente. Sin embargo en esta práctica no tenemos el método "generarKey()" ya que no vamos a buscar un elemento en el array, sino que vamos a ordenar dicho array, para ello vamos a hacer uso de la clase "AlgoritmosOrdenacion".

"AlgoritmosOrdenacion" contine los algoritmos de ordenación (Burbuja, Selección e Inserción) para ordenar un vector de enteros en orden descendente.

"Mtime" permite determinar el tiempo de procesador que tarda en ejecutarse el algoritmo para el caso empírico, puede haber variaciones de tiempo debido a la dependencia de carga de trabajo del ordenador.

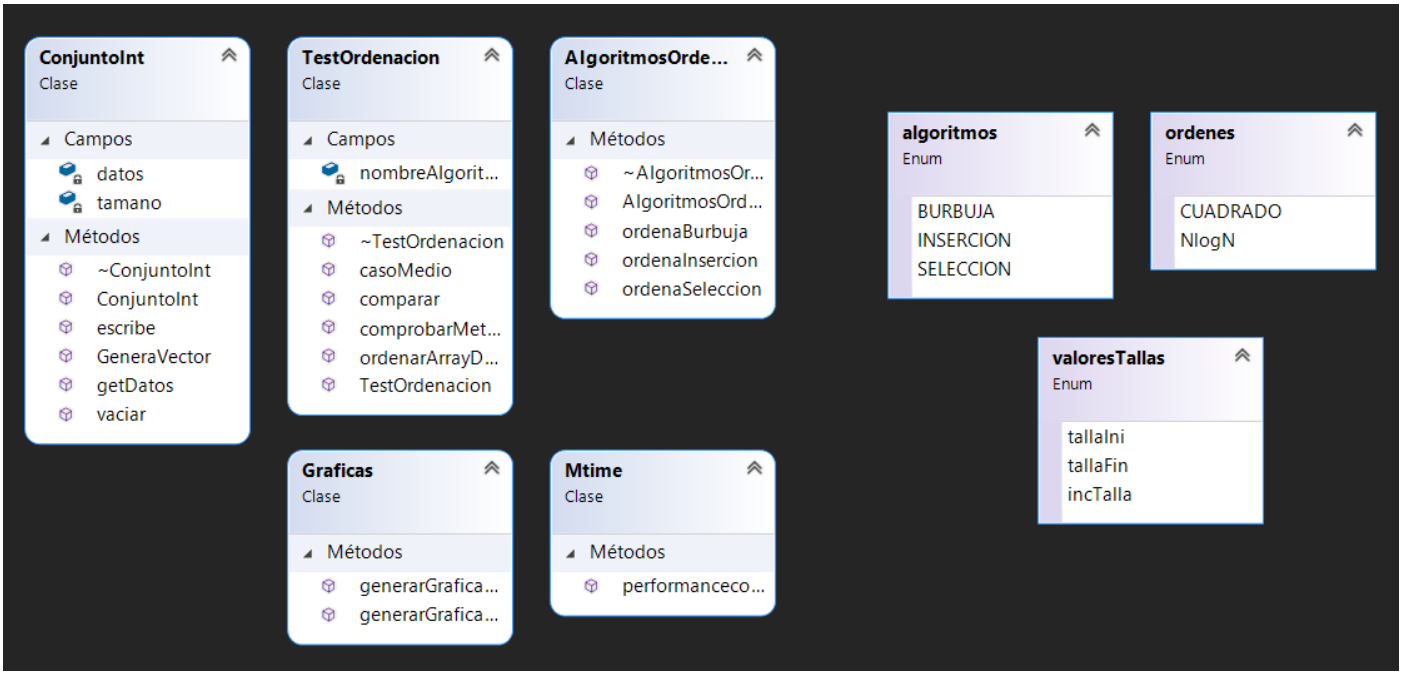
"Constantes.h" contiene una serie de constantes útiles para el desarrollo de la práctica.

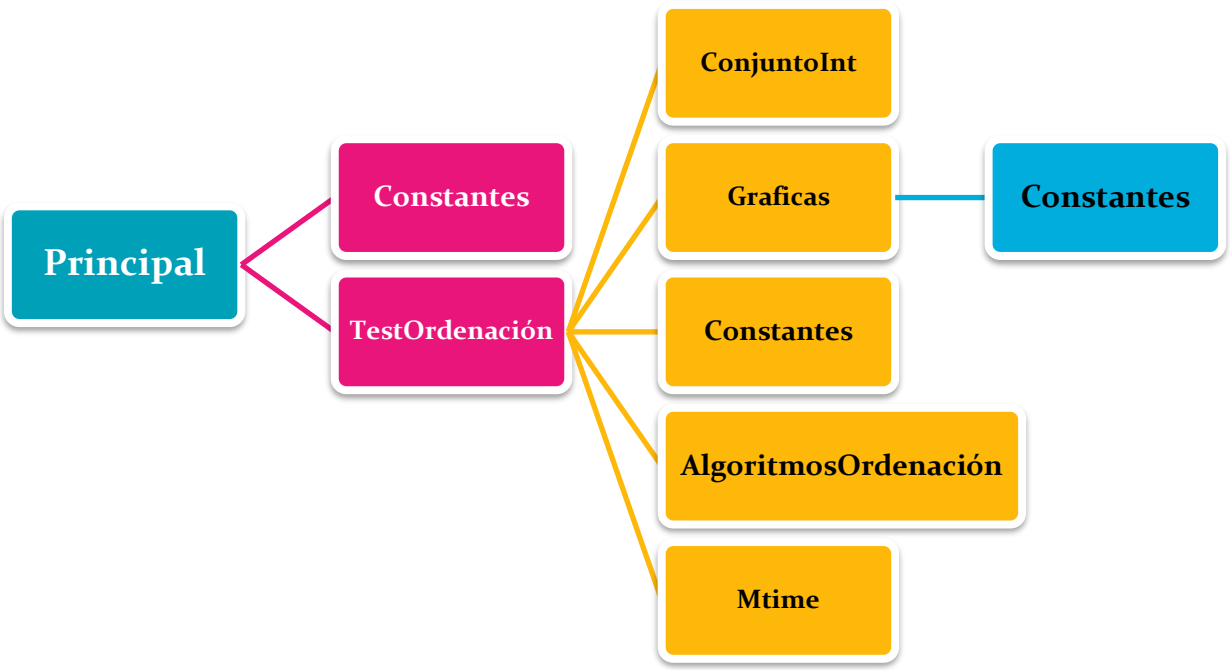
"Graficas" contiene métodos para guardar las gráficas de los resultados, es decir, crea los ficheros por lotes (comandos) para generar los ficheros gráficos que corresponda.

"TestOrdenacion" tiene los siguientes métodos principales:

- **ordenarArrayDeInt:** Ordena un array de enteros según el método indicado.
- **comprobarMetodosOrdenacion:** Permite comprobar si los métodos de ordenación funcionan correctamente. Este método llama a ordenarArrayDeInt.
- **casoMedio:** Calcula el caso medio de un método de ordenación, también genera un fichero donde se guardan los datos obtenidos y permite generar su gráfica al respecto.
- **comparar:** Compara dos métodos de ordenación, guarda o sobrescribe los datos obtenidos en los ficheros de los métodos correspondientes y permite generar su gráfica al respecto.

Contenido y relaciones de las clases de la práctica 2:





8. Conclusiones y valoraciones personales de la práctica

En esta práctica hemos aprendido el manejo, la implementación y el uso que tienen los algoritmos de ordenación y cómo se comportan y diferencian entre ellos. Y aunque hay una gran cantidad de algoritmos distintos, y para cada uno numerosas modificaciones y optimizaciones. No existe ningún algoritmo que sea superior al resto en todos los aspectos.

La ejecución en diversas máquinas reveló que el tiempo de ejecución puede variar drásticamente en función de la configuración de hardware. El tiempo medio es hasta cuatro veces inferior en una estación de trabajo frente a un portátil estándar, por tener mayor velocidad de reloj. En un portátil, si la talla es baja los resultados parecen no ceñirse a la formula teórica. Mientras que en una estación de trabajo puede darse una forma perfecta de la curva desde el inicio.