# PRÁCTICA 2

FAA – 1º Ingeniería Informática

Análisis experimental de algoritmos de ordenación

Alba Márquez Rodríguez

## 3. Introducción. Algoritmos de Ordenación.

En esta práctica implementaremos un algoritmo que tiene como objetivo el análisis y estudio de la eficiencia de algoritmos. Con el estudio empírico correspondiente de 3 algoritmos de ordenación: burbuja, inserción y selección. Por separado y comprándolos.

Los algoritmos de ordenación consisten en algoritmos que ordenan elementos que están en la memoria principal del ordenador. Se asume que los elementos están en una relación de orden < y almacenados en un contenedor lineal (vector o lista). El objetivo es obtener algoritmos de ordenación lo más eficientes posibles (en tiempo de cálculo y memoria). El tiempo de ejecución viene dado por la función complejidad del número de elementos a ser ordenados (talla de 100 a 1000).

El estudio empírico consiste en el estudio experimental del comportamiento del algoritmo. Midiendo el tiempo en los tres posibles algoritmos: burbuja, inserción y selección. Este variará según la capacidad de cada ordenador y el uso que se esté haciendo en el momento de la memoria.

## 4. Cálculo de los tiempos teóricos:

Aplicando las reglas estudiadas en clase calcularemos el número de operaciones elementales del algoritmo. Lo pondremos como una función de T(n). Siendo n las veces que se ejecutará el bucle. Al depender el tiempo de ejecución de la talla, n=talla

### 4.1. Pseudócódigos y análisis de coste

Este es el pseudocódigo que se nos da en el enunciado del ejercicio, completamos con las operaciones elementales (asignaciones, accesos…). Para todos los algoritmos cogeremos el caso medio ya que es el que nos interesa en esta práctica, para obtenerlo debemos calcular antes el caso mejor y el peor.

Burbuja

```
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
```

Calculamos y tenemos en cuenta que para el mejor caso el si no se ejecuta así que sólo se tendrían en cuenta la condicional mientras que en el caso peor sí. Todos los casos son O(n^2)
El caso medio es:

$$T(n) = \frac{25}{4}n^2 + \frac{3}{4}n - \frac{3}{2}$$

Caso general $T = T_{para}$

$$T_{para} = 2+1+1+\sum_{i=n-1}^{1} T_{ciclopara} = 4 + 7(n-1) + (4+T_{si})\frac{n(n-1)}{2}$$

$$T_{ciclop} = 1+2+1+T_{para2} : \quad 7 + (4+T_{si})i$$

$$T_{para2} = 1+1+1+\sum_{1}^{i} T_{ciclopara2} = 3+(4+T_{si})i$$

$$T_{ciclop} = 1+2+1+T_{si} = 4+T_{si}$$

Caso mejor ($si = 4$)

$$4+7(n-1)+(4+4)\left(\frac{n^2-n}{2}\right)$$

$$4+7n-7+4(n^2-n)$$

$$T(n) = 4n^2+3n-3$$

Caso peor ($si = 13$)

$$4+7(n-1)+(4+13)\frac{(n^2-n)}{2}$$

$$4+7n-7+\frac{17n^2}{2}-\frac{17n}{2}$$

$$T(n) = \frac{17n^2}{2} - \frac{3}{2}n - 3$$

Caso medio

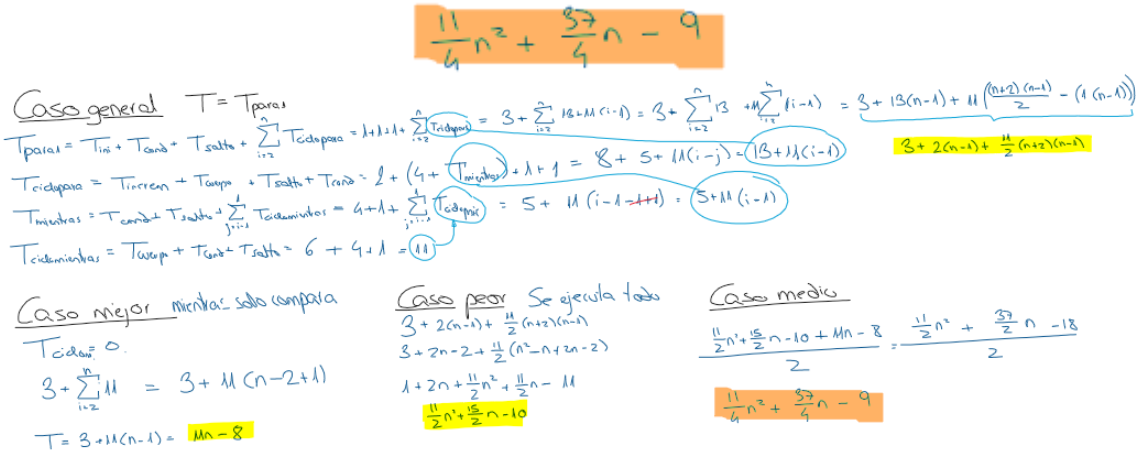
$$T(n) = \left(4n^2 + \frac{17}{2}n^2 + 3n - \frac{3}{2}n - 3 - 3\right)/2$$

$$T(n) = \frac{25}{4}n^2 + \frac{3}{4}n - \frac{3}{2}$$

## Inserción

```
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
```

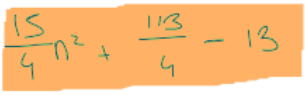
Volvemos a repetir el procedimiento en este algoritmo, en el caso mejor mientras no se ejecutará (sólo las comparaciones) y en el peor se ejecutará un número n de veces, esto hace que en el caso mejor el O(n) y en el peor O(n^2), en consecuencia el caso medio es O(n^2). El caso medio es:

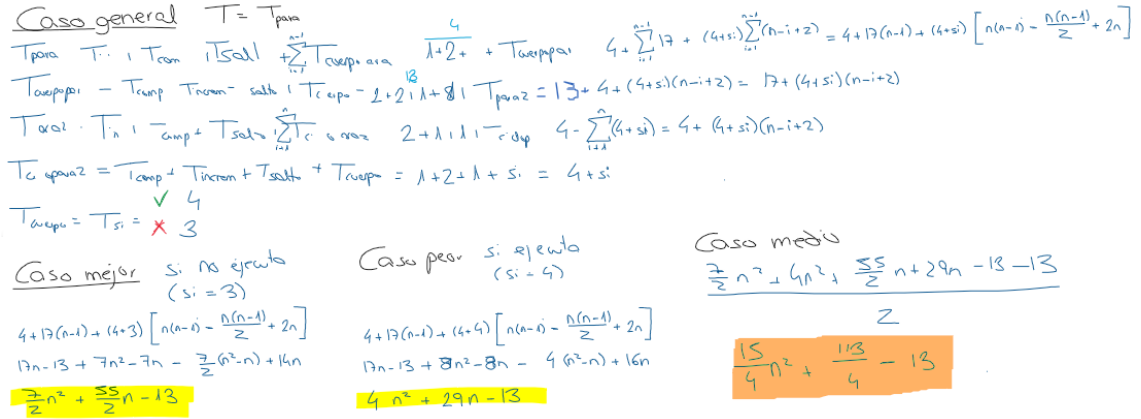



## Selección

```
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
```

En este algoritmo el caso mejor y peor vuelven a depender de una condicional (el si) y seguirá por tanto una estructura parecida a Burbuja. Todos los casos son de O(n^2). El caso medio será:



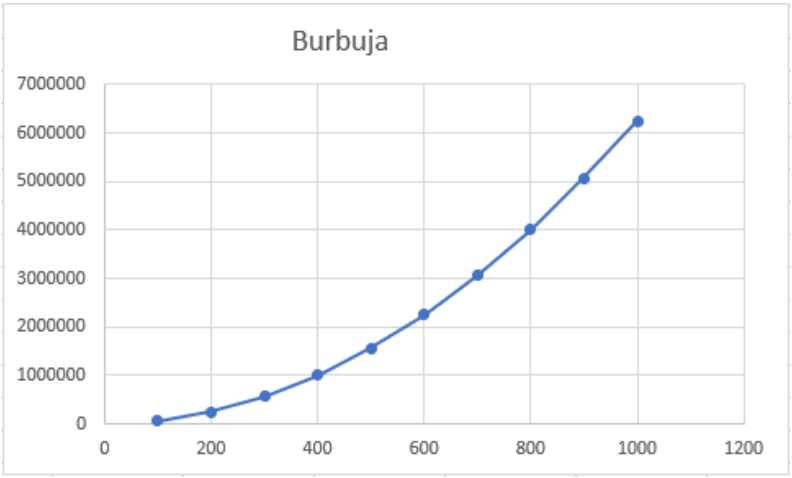



De estos datos podemos comparar las fórmulas y suponer los resultados (cuáles serán más/menos eficiente). Por el orden de complejidad el más eficiente será inserción aunque en el caso medio todos sean del mismo orden. Luego selección y por último burbuja.

## 4.2. Tablas (ficheros) y Gráficas de coste

Burbuja

| Talla | T(n) |
|---|---|
| 100 | 62573,5 |
| 200 | 250148,5 |
| 300 | 562723,5 |
| 400 | 1000298,5 |
| 500 | 1562873,5 |
| 600 | 2250448,5 |
| 700 | 3063023,5 |
| 800 | 4000598,5 |
| 900 | 5063173,5 |
| 1000 | 6250748,5 |

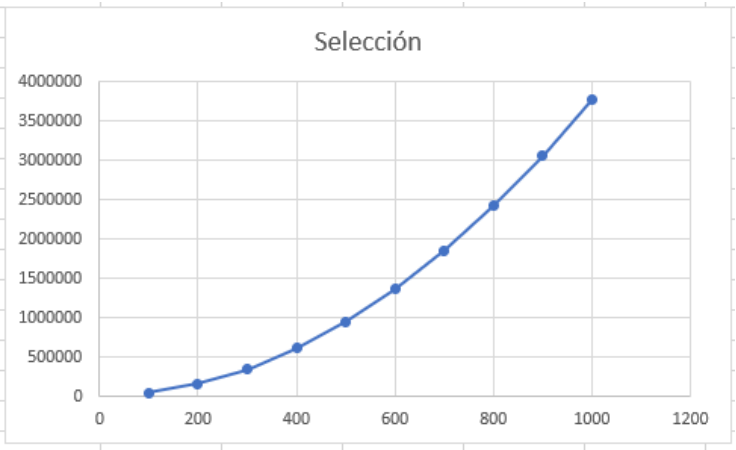



Inserción

| Talla | T(n) |
|---|---|
| 100 | 28416 |
| 200 | 111841 |
| 300 | 250266 |
| 400 | 443691 |
| 500 | 692116 |
| 600 | 995541 |
| 700 | 1353966 |
| 800 | 1767391 |
| 900 | 2235816 |
| 1000 | 2759241 |

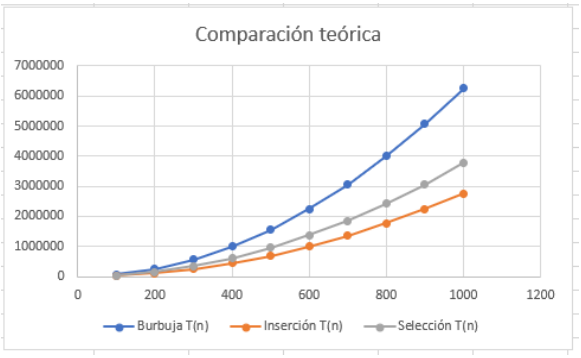

Selección

| Talla | T(n) |
|---|---|
| 100 | 40312 |
| 200 | 155637 |
| 300 | 345962 |
| 400 | 611287 |
| 500 | 951612 |
| 600 | 1366937 |
| 700 | 1857262 |
| 800 | 2422587 |
| 900 | 3062912 |
| 1000 | 3778237 |



### 4.3. Conclusiones

| Talla | Burbuja T(n) | nserción T(n | elección T(n) |
|---|---|---|---|
| 100 | 62573,5 | 28416 | 40312 |
| 200 | 250148,5 | 111841 | 155637 |
| 300 | 562723,5 | 250266 | 345962 |
| 400 | 1000298,5 | 443691 | 611287 |
| 500 | 1562873,5 | 692116 | 951612 |
| 600 | 2250448,5 | 995541 | 1366937 |
| 700 | 3063023,5 | 1353966 | 1857262 |
| 800 | 4000598,5 | 1767391 | 2422587 |
| 900 | 5063173,5 | 2235816 | 3062912 |
| 1000 | 6250748,5 | 2759241 | 3778237 |




Los tres algoritmos tienen crecimientos cuadráticos en sus casos medios, que son los que nos interesan, tras los cálculos y comparación de las fórmulas obtenidas intuimos que el más eficiente será inserción, después selección y el menos eficiente al consumir más tiempo: burbuja.

Tras realizar tablas en Excel con las tallas correspondientes y fórmulas y crear gráficas observamos que estábamos en lo cierto, todas tienen crecimiento cuadrático y se cumple el orden de eficiencia supuesto.

## 5. Cálculo del tiempo experimental:

Para el estudio empírico tendremos que medir los recursos empleados (tiempo). Para ello elaboramos pruebas en diferentes condiciones (incremento de la talla del problema) ya que el tiempo de ejecución del algoritmo dependerá del tamaño del problema. Para cada caso la medida computacional será el tiempo de ejecución.

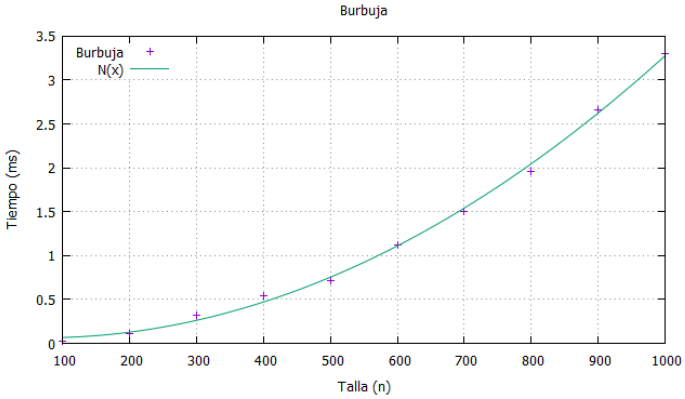
Para el caso “estudio promedio” al ser aleatorio introduciremos un bucle for que tome diferentes valores y luego haga una media. Haremos esto para los tres algoritmos que estudiaremos. Con la constante NUMREPETICIONES. Así la medida final será un promedio de todas las efectuadas para conseguir un resultado lo más empírico y veraz posible. Estos datos se mostraran en la pantalla de la terminal y guardados en el fichero de datos y en el fichero de gráficos. En el fichero gráficos queda guardado por la clase Gráficas que crea un fichero de gnuplot (.gpl) con el código necesario para que se ejecute con la gráfica correspondiente.

Esto queda implementado gracias a la clase Mtime, con QueryPerformanceCounter obtenemos el tiempo inicial y final que se tarda en ejecutar el algoritmo de ordenación correspondiente (dentro de la clase AlgoritmosOrdenación hemos implementado burbuja, inserción y selección), hacemos la diferencia y así obtenemos el tiempo de ejecución del caso correspondiente. Haciendo uso de los métodos de la clase ConjuntoInt podemos obtener vectores de datos aleatorios del tamaño talla (que es lo que vamos incrementando para analizar y estudiar el incremento del tiempo según la talla del problema). Estos vectores son los que ordenará el Algoritmo de Ordenación correspondiente. Hacemos uso de un bucle for para calcular una media y que sea más exacto y real los casos medidos en estudio promedio.

## 5.1. Tablas (ficheros) y Gráficas de coste

Burbuja

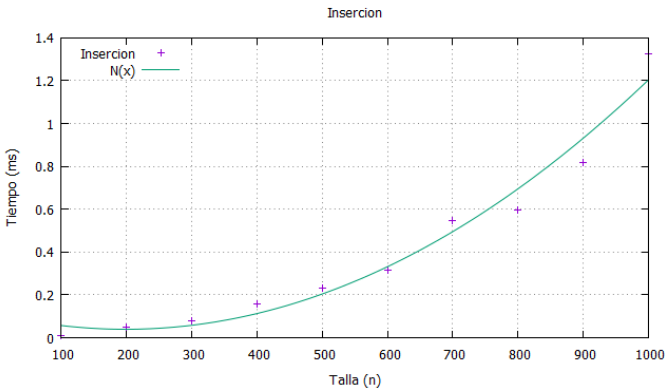
```
*** Ordenacion por Burbuja ***

Tiempos de ejecucion promedio

        Talla           Tiempo (mseg)

        100                     0.03
        200                     0.11
        300                     0.32
        400                     0.54
        500                     0.72
        600                      1.1
        700                      1.5
        800                        2
        900                      2.7
        1000                     3.3

Datos guardados en el fichero Burbuja.dat

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




En este algoritmo obtenemos una gráfica creciente de forma cuadrática (parte de una parábola). Esto es lo esperado ya que a mayor tamaño de problema mayor tiempo de ejecución. Hay puntos que no coinciden con la curva. Esto se debe a que al tratarse de una medida empírica y en tiempo real, el ordenador puede tardar más o menos en el proceso (por programas que tengamos abiertos por ejemplo). Observaremos que esto pasa en el resto de algoritmos.

Inserción

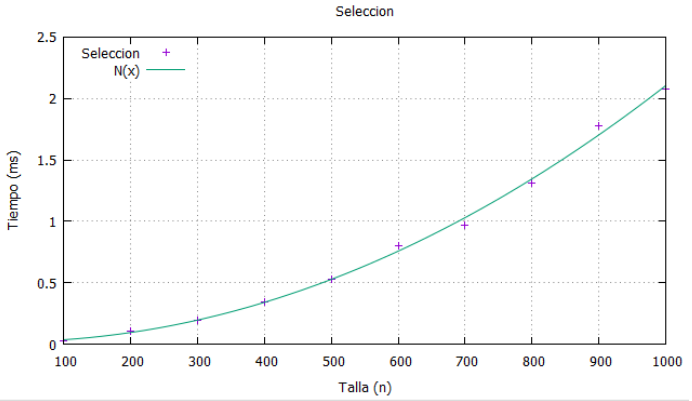
```
*** Ordenacion por Insercion ***

Tiempos de ejecucion promedio

        Talla           Tiempo (mseg)

        100                    0.011
        200                     0.05
        300                     0.08
        400                     0.16
        500                     0.23
        600                     0.31
        700                     0.55
        800                      0.6
        900                     0.82
        1000                     1.3

Datos guardados en el fichero Insercion.dat

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




Como en el caso anterior obtenemos una curva creciente cuadrática (una parte de una parábola). A medida que la talla de la tabla es mayor. Hay puntos que no coinciden con la curva, como ya hemos dicho, al tratarse de un estudio empírico los resultados no serán exactos por otros programas y condiciones que tenga el ordenador en el momento de ejecución de cada talla, a pesar de haber hecho un bucle for para que el valor sea lo más aproximado posible.

Selección

```
*** Ordenacion por Seleccion ***

Tiempos de ejecucion promedio

        Talla           Tiempo (mseg)

        100                 0.026
        200                   0.1
        300                  0.19
        400                  0.35
        500                  0.53
        600                   0.8
        700                  0.97
        800                   1.3
        900                   1.8
        1000                  2.1

Datos guardados en el fichero Seleccion.dat

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



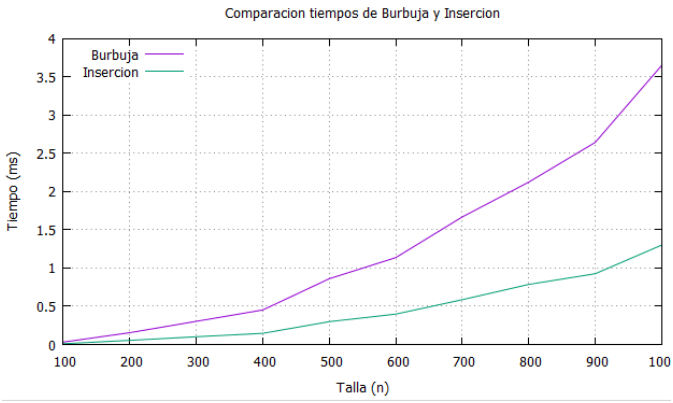

En este algoritmo, como en los otros casos, nos encontramos una curva cuadrática, parte de una parábola. También obtenemos puntos que no coinciden con la curva, esto se debe a lo explicado en otros casos

## 5.2. Conclusiones

Para sacar conclusiones compararemos los datos y gráficas que comparan 2 algoritmos. Esto está implementado en el método comparar. Dos algoritmos, seleccionados por el usuario, ordenan el mismo vector variando la talla.

Burbuja-Inserción

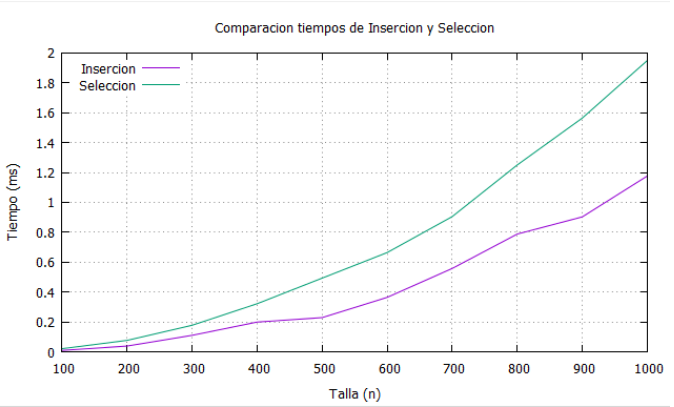
| Talla | Tiempo (mseg) | |
|---|---|---|
| | Burbuja | Insercion |
| 100 | 0.032 | 0.0096 |
| 200 | 0.15 | 0.054 |
| 300 | 0.3 | 0.1 |
| 400 | 0.45 | 0.15 |
| 500 | 0.86 | 0.3 |
| 600 | 1.1 | 0.4 |
| 700 | 1.7 | 0.59 |
| 800 | 2.1 | 0.79 |
| 900 | 2.6 | 0.93 |
| 1000 | 3.7 | 1.3 |




Entre estos dos métodos, podemos observar rápidamente que el más eficiente es el de inserción. Tanto para vectores de una talla pequeña como para mayores tiene un tiempo de ejecución menor que el de burbuja además de tener un crecimiento más lento.

Inserción-selección

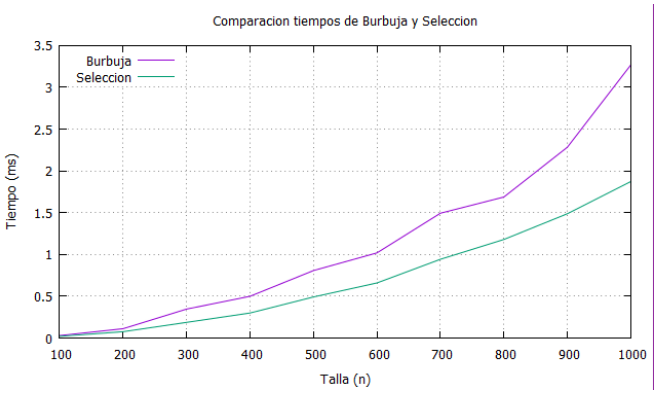
| Talla | Tiempo (mseg) | |
|---|---|---|
| | Insercion | Seleccion |
| 100 | 0.0091 | 0.021 |
| 200 | 0.038 | 0.076 |
| 300 | 0.11 | 0.18 |
| 400 | 0.2 | 0.32 |
| 500 | 0.23 | 0.49 |
| 600 | 0.36 | 0.66 |
| 700 | 0.56 | 0.9 |
| 800 | 0.79 | 1.3 |
| 900 | 0.9 | 1.6 |
| 1000 | 1.2 | 1.9 |




Entre estos dos métodos el más eficiente es el de inserción. Para vectores pequeños y grandes.

Burbuja – selección

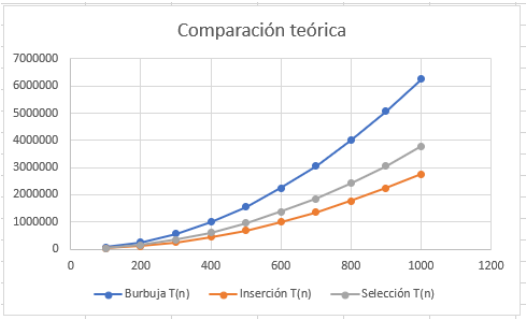
| Talla | Tiempo (mseg) | |
|---|---|---|
| | Burbuja | Seleccion |
| 100 | 0.03 | 0.02 |
| 200 | 0.11 | 0.075 |
| 300 | 0.35 | 0.19 |
| 400 | 0.5 | 0.3 |
| 500 | 0.81 | 0.49 |
| 600 | 1 | 0.66 |
| 700 | 1.5 | 0.94 |
| 800 | 1.7 | 1.2 |
| 900 | 2.3 | 1.5 |
| 1000 | 3.3 | 1.9 |

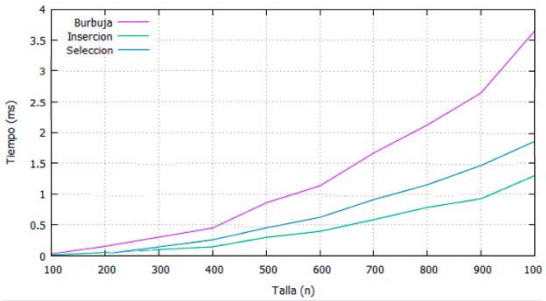



Entre estos dos métodos el más eficiente es selección al ser el que menos tiempo de ejecución tiene.

Así obtenemos que ordenados por eficiencia. El más eficiente es inserción, luego selección y el menos eficiente es burbuja.

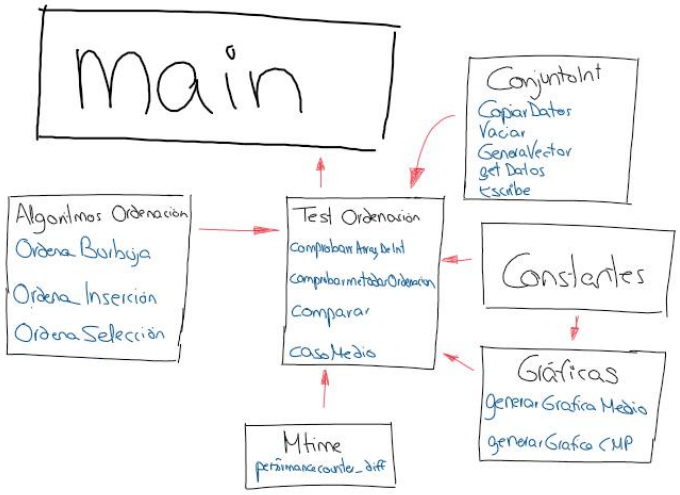
## 6. Comparación de los resultados teórico y experimental.




En este caso las escalas no son importantes ya que son aproximadamente proporcionales. Aún así podemos observar que la escala de la teórica es mucho más grande que la empírica. Esto se debe a que la teórica está medida en OE mientras la empírica en ms. Por otro lado tenemos los picos, la gráfica teórica es mucho más suave al ser datos que siguen una fórmula cuadrática mientras que en la empírica nos encontramos una gráfica con picos. Teniendo en cuenta que esto se debe al tiempo de ejecución de cada ordenador. Los resultados son coherentes y coinciden, burbuja es el menos eficiente, inserción el más.

## 7. Diseño de la aplicación.




El programa se compone de 5 clases: ConjuntoInt, Mtime, TestOrdenación, Graficas y AlgoritmosOrdenación además de un fichero de constantes.

En el fichero constantes, como su nombre indica, de definen unas constantes para recurrir a ellas y simplificar el programa.

La clase ConjuntoInt nos permite crear vectores de tamaños variables. Se genera el vector de forma aleatoria para el tamaño dado.

La clase Mtime nos permite obtener el valor del contador y la frecuencia. Que nos permite obtener el tiempo para medir la eficiencia del algoritmo de forma empírica.

La clase AlgoritmosOrdenación contiene los algoritmos de ordenación que hemos estudiado. Burbuja, inserción y selección. Esto se ha implementado a partir de los pseudocódigos, que se han cambiado a lenguaje de c++.

La clase TestOrdenación es la que contiene las funciones principales del programa que se llaman en el main. Tiene un constructor que inicia con los nombres de los 3 posibles algoritmos (burbuja, inserción y selección).

El método comprobar métodos de ordenación se generar un vector y es ordenado por el método seleccionado. Si ejecutamos el programa nosotros podremos introducir el tamaño del vector. Se genera con valores aleatorios. Así podemos observar el funcionamiento de cada algoritmo y si ordena correctamente independientemente de la talla introducida

El main está compuesto de un menú principal que nos permite acceder a dos submenús: si seleccionamos obtener caso promedio y comparar dos métodos. Con la primera opción de probar los métodos de ordenación metemos talla y se ejecuta cada método de ordenación. Con obtener el caso medio debemos seleccionar el método del que queremos obtenerlo y se ejecuta mostrando tiempos y la posibilidad de generar y mostrar la gráfica. Al comparar dos métodos seleccionamos los métodos a comparar y se muestran los tiempos de ejecución de cada método para comparar la ordenación del mismo vector, nos da la posibilidad de mostrar y generar gráfica. Además, las gráficas se guardan en un archivo de pdf. Para salir del programa simplemente tendremos que seleccionar salir en el menú principal.

## 8. Conclusiones y valoraciones personales de la práctica.

Esta práctica nos ha permitido comprobar, comparar y aprender como funcionan los algoritmos de ordenación y las diferencias en la eficiencia. Mostrando la importancia de realizar un estudio de la eficiencia de un algoritmo antes de implementarlo para conseguirlo con los mínimos recursos posibles. En la práctica anterior sólo se tuvo que implementar el main e implementar los métodos empíricos para TestAlgoritmos. En esta práctica se ha reducido la parte realizada y hemos tenido que elaborar más parte del programa. Al querer comprobar la eficiencia por tiempo de ejecución nos hemos podido basar en lo hecho en la anterior práctica, ahorrando así tiempo de programación. La diferencia y dificultad en la elaboración del programa quizás haya sido elaborar los ficheros gnuplot desde c++. El lenguaje de gnuplot no lo conocemos y dificulta encontrar errores y solucionarlos. Pero al ser parecida a la práctica anterior esto se ha podido realizar de forma más sencilla. Otra dificultad ha sido tener que generar otro vector para que el estudio fuera realmente empírico durante la comparación. Algunos alumnos no caímos en este dato pero finalmente se aclaró en clase y se realizó.

Otra dificultad ha sido calcular los datos teóricos, ha sido un proceso muy largo ya que se ha tenido que calcular el caso medio de tres algoritmos diferentes aunque parecidos. Esto ha reforzado saber calcular el tiempo de un algoritmo en función de su talla y en sus tres casos.

No se ha explicado y no sabemos como se debe elaborar el diseño gráfico de una aplicación por lo que esto también ha sido una dificultad añadida sin saber con certeza como se debe hacer ni si así es su forma correcta.

Ha sido una práctica interesante y que se basa en la práctica anterior. Esto facilita su elaboración, ahorrando, como ya he indicado, tiempo de programación. Además de seguir usando archivos y ficheros que ayuda a la asimilación de la teoría dada en otras asignaturas. También nos hace ser conscientes de la necesidad del estudio de algoritmos antes de su implementación.