Análisis de algoritmos de ordenación.



# Índice

# P0. [Portada](#_top)

# P1. [Índice](#_Índice)

# P2. [Introducción](#_Introducción)

# P2. [Cálculo de tiempo teórico](#_Cálculo_de_tiempo)

## [Código y análisis de coste](#_Código_y_análisis)

## [Burbuja](#_Código_y_análisis) [Selección](#_Código_y_análisis_1) [Inserción](#_Código_y_análisis_2)

[Quicksort](#_Código_y_análisis_3)

## [Tabla y gráficas de coste](#_Tabla_y_graficas)

## [Conclusiones](#_Conclusiones)

# P4. [Cálculo del tiempo experimental](#_Cálculo_del_tiempo)

## [Tablas y gráficas de coste](#_Tablas_y_gráficas)

## [Conclusiones](#_Conclusiones)

# P5. [Comparación de los resultados teórico y experimental](#_Comparación_de_los)

# P5. [Diseño de la aplicación](#_Diseño_de_la)

# P6. [Conclusiones y valoraciones personales de la práctica](#_Conclusiones_y_valoraciones)

# Introducción

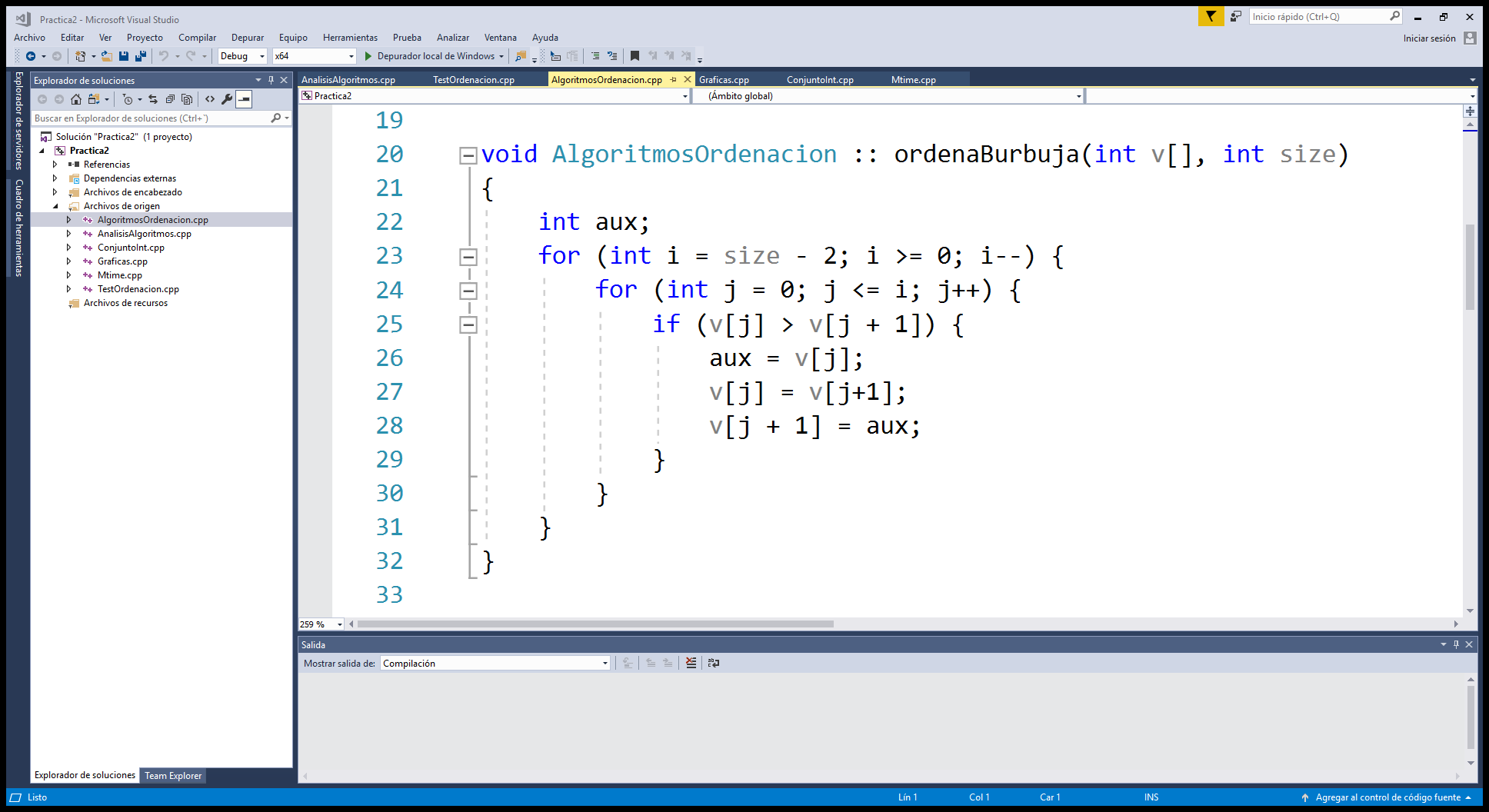
El objetivo de esta práctica es realizar el estudio teórico de múltiples algoritmos de ordenación, posteriormente implementarlos en un lenguaje de programación y comparar su eficiencia empíricamente.

Para esta ocasión se ha escogido el lenguaje C++ y el entorno de desarrollo integrado Visual Studio 2017. Adicionalmente hemos utilizado GNUplot para graficar los resultados.

# Cálculo de tiempo teórico

Fueron propuestos en pseudocódigo e implementados por el alumno, en cuatro funciones. Todas ellas reciben un vector de enteros y el número de elementos que contiene, cuando terminan el vector queda ordenado en orden creciente.

## Código y análisis de coste del algoritmo de la burbuja

Este algoritmo puede ser descrito mediante un símil con burbujas de aire subiendo en el agua. En este caso la densidad es el número a ordenar, a menor es su densidad más cerca del inicio de la tabla quedan.

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.

De los tres, es el algoritmo más sencillo.

Eficiencia teórica:

* Caso mejor: Al no comprobar si el array se encuentra ordenado, el número de comparaciones que realiza es de orden n2, no hay intercambios.
* Caso medio y peor: El número de comparaciones y intercambios es de orden n2.

## Código y análisis de coste del algoritmo de selección

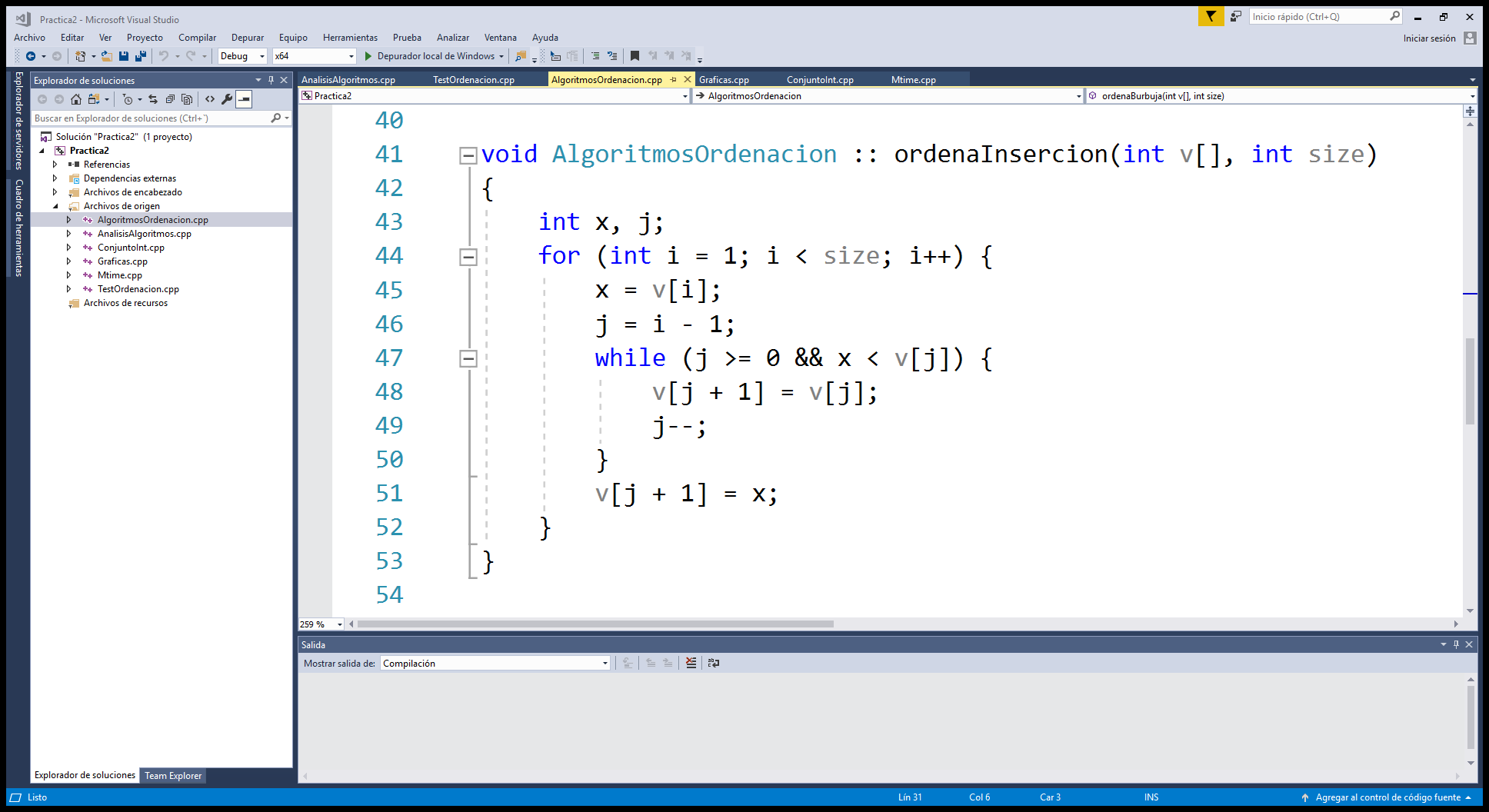
El algoritmo de selección busca el elemento menor de la tabla y lo mueve a la posición correspondiente, que es hallada mediante una búsqueda lineal.  
De esta forma, el array se va ordenando directamente con los elementos en sus posiciones finales y solo realiza un intercambio por cada pasada.

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.

Eficiencia teórica:

* Caso mejor, medio y peor: El número de comparaciones es de orden n2 y el de intercambios es de orden n, para todos los casos.

## Código y análisis de coste del algoritmo de inserción

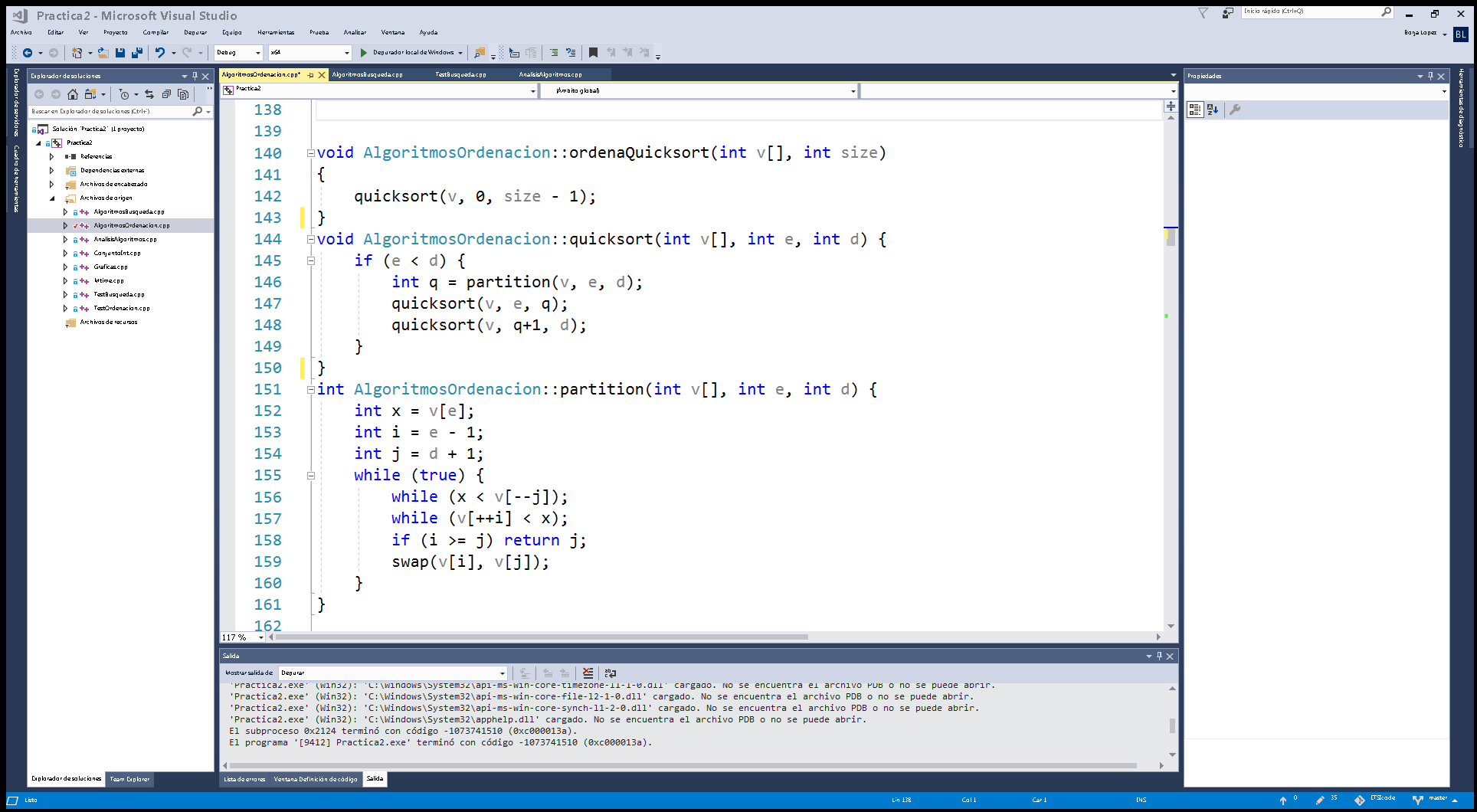
Este algoritmo realiza el proceso inverso a Selección. En lugar de buscar el elemento que encaja en una posición, busca la posición que corresponde a cada elemento que se encuentra.

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 desde la izquierda.

Eficiencia teórica:

* Caso mejor: Si el vector ya está ordenado, encuentra que la posición para insertar el elemento es en la que ya se encuentra. Le basta con una pasada para determinar que se encuentra ordenado. Complejidad de orden n.
* Caso medio y peor: El número de comparaciones es de orden n2 y el de intercambios es de orden n.

## Código y análisis de coste del algoritmo Quicksort

Quicksort es un algoritmo de ordenación recursivo. Comienza por seleccionar un pivote, todos los elementos anteriores son menores y los elementos siguientes son mayores que él. De esta forma, la ordenación independiente de los subconjuntos divididos por el pivote, provoca la ordenación del array en su totalidad. Por este motivo, pese a seguir el modelo Divide y Vencerás, no necesita combinar los resultados.

Esto lo hace generalmente mejor que otros algoritmos de su mismo orden como Mergesort o HeapSort.

Una vez ha dividido el conjunto que le pasan, manda a dividir los dos subconjuntos por separado.  
No se utiliza memoria auxiliar, sino que todas las llamadas utilizan la misma zona de memoria acotada con diferentes índices.

Eficiencia teórica:

* Caso mejor: Por el Teorema Maestro, 2 divisiones (a=2), se dividen entre 2 (b=2), la división tiene orden O(n) , es decir n1 (p=1). Estamos en el caso a = bp (2 = 21), t(n) es del orden O(np\*logn)
* Caso medio: La división no se produce exactamente por la mitad, el tiempo está entre el mejor y el peor.
* Caso peor: El pivote elegido es siempre el primer elemento del array, luego se realizan n2 divisiones.

## Tabla y graficas de coste

|  |  |  |  |
| --- | --- | --- | --- |
| Talla | Tiempo (ms) | | |
| **Burbuja** | **Selección** | **Intersección** |
| 10 | 270 | 172 | 75 |
| 20 | 1140 | 647 | 250 |
| 30 | 2610 | 1422 | 525 |
| 40 | 4680 | 2497 | 900 |
| 50 | 7350 | 3872 | 1375 |
| 60 | 10620 | 5547 | 1950 |
| 70 | 14490 | 7522 | 2625 |
| 80 | 18960 | 9797 | 3400 |
| 90 | 24030 | 12372 | 4275 |
| 100 | 29700 | 15247 | 5250 |

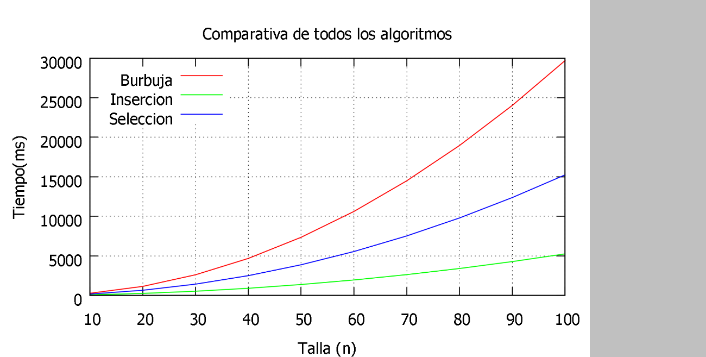
Para obtener el caso medio se tomó la suma de las comparaciones y los intercambios mínimos y los máximos divididos entre dos. Un intercambio requiere tres veces más operaciones elementales que una comparación.

Dentro de que los tres algoritmos son lentos (orden n2); Inserción resulta ser más eficiente, seguido de Selección y finalmente Burbuja.

Quicksort no pudo ser comparado en esta tabla.

## Conclusiones

Como se esperaba, el coste temporal del algoritmo depende de la talla del problema y tanto de la instancia. Los tres algoritmos lentos estudiando son de orden polimétrico cuadrado, no obstante, algunos realizan menos instrucciones que otros. Quicksort en el caso medio crece más lentamente

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

En cuanto a complejidad espacial, todos los algoritmos son , la cantidad de memoria consumida por el algoritmo no es proporcional al numero de elementos a ordenar.  
En general, todos utilizan una variable temporal para los intercambios.

# Cálculo del tiempo experimental

Para cada medición se generan nuevos conjuntos de datos aleatorios, pero solo se mide el tiempo de búsqueda. Este proceso se repite múltiples veces y se toma la media.

## Tablas y gráficas de coste

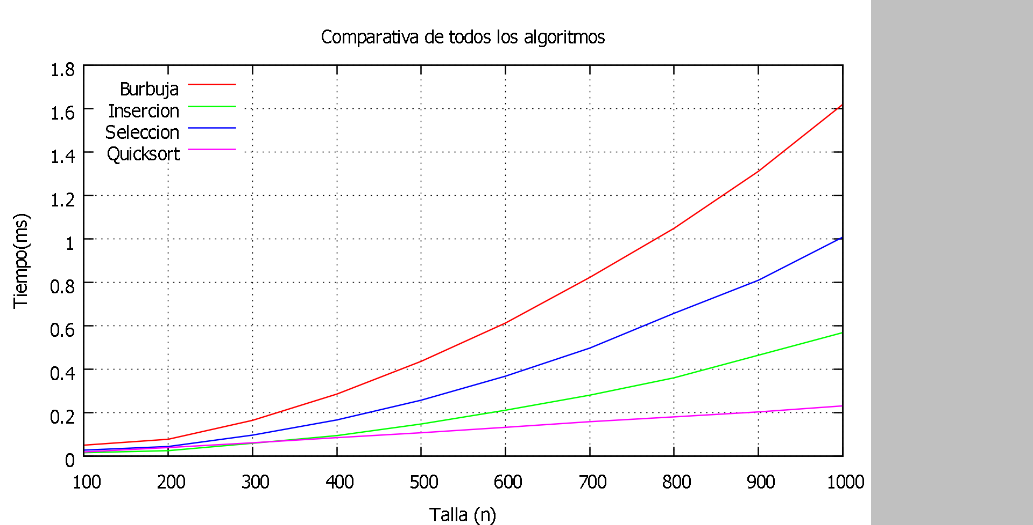
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Talla | Tiempo (ms) | | |  |
| **Burbuja** | **Selección** | **Intersección** | **Quicksort** |
| 100 | 0.0240 | 0.0140 | 0.0074 | 0. 0190 |
| 200 | 0.0880 | 0.0471 | 0.0290 | 0. 0393 |
| 300 | 0.1607 | 0.0916 | 0.0546 | 0. 0610 |
| 400 | 0.2646 | 0.1530 | 0.0931 | 0. 0849 |
| 500 | 0.3930 | 0.2459 | 0.1514 | 0. 1075 |
| 600 | 0.5663 | 0.3350 | 0.1977 | 0. 1323 |
| 700 | 0.7546 | 0.4617 | 0.2842 | 0. 1584 |
| 800 | 0.9703 | 0.5953 | 0.3677 | 0. 1806 |
| 900 | 1.2093 | 0.7388 | 0.4438 | 0. 2033 |
| 1000 | 1.4974 | 0.9245 | 0.5368 | 0. 2311 |

## 

## 

## 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.

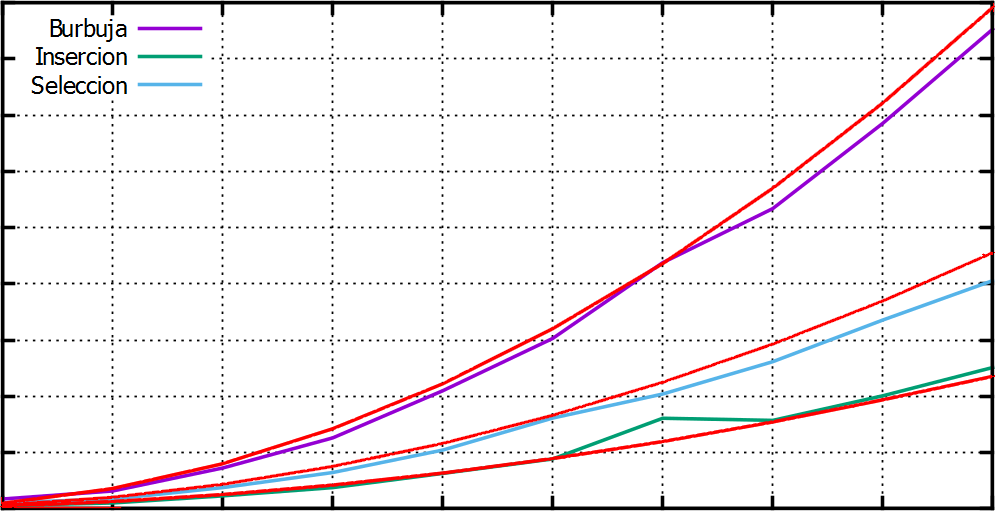
Puede apreciarse la rápida tendencia a crecer de las funciones cuadráticas. Seguir aumentando la talla, requeriría un tiempo de ejecución mucho mayor; frente al bajo crecimiento de Quicksort.

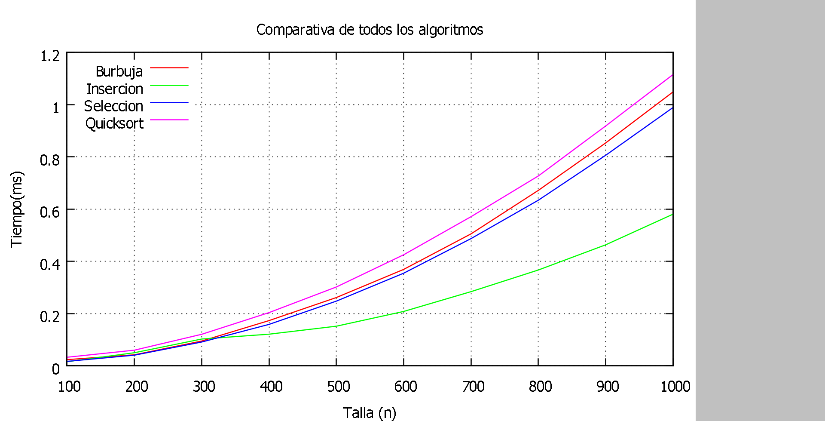
Pese a esto, se ha observado que Quicksort resulta más lento que Inserción y Selección en tallas pequeñas y que su eficiencia se vuelve presente a partir de la mitad de la prueba, donde toma clara ventaja.

# Comparación de los resultados teórico y experimental

Los resultados experimentales reflejan la misma relación entre los algoritmos. Siendo Inserción el método más rápido, burbuja el más lento.

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

En este gráfico puede visualizarse la relación entre los resultados experimentales y el cálculo teórico (líneas rojas). Para generar las curvas se calculó el numero de comparaciones e intercambios para el caso medio de cada algoritmo. Se ponderó tres veces más a los intercambios, ya que son operaciones más pesadas.



En el caso de Quicksort la gráfica generada corresponde con un crecimiento n\*log(n), excepto en el caso peor, donde debe ser aproximado mediante n2.

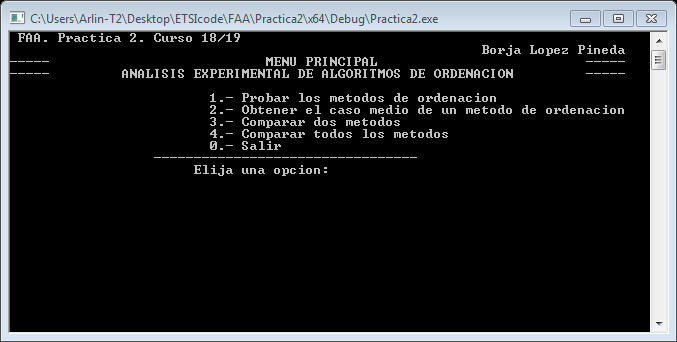
En comparación Quicksort resultó ser el más lento de los cuatro algoritmos para el caso peor, ganando por clara ventaja Inserción.

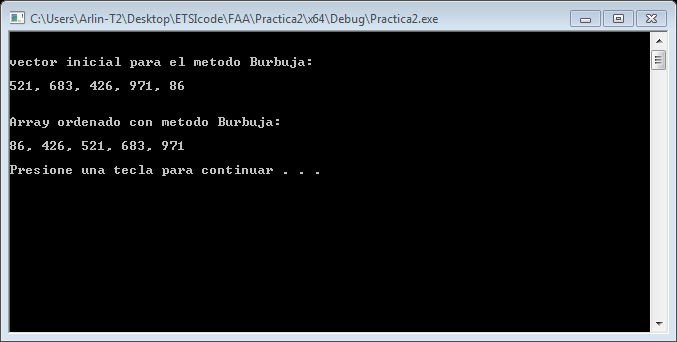
# Diseño de la aplicación

La metodología de programación escogida para la implementación fue diseño modular con orientación a objetos.

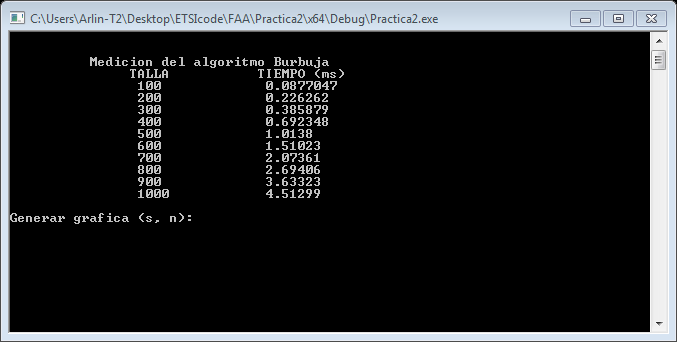
El código soporta agregar más algoritmos, incluso si tienen complejidad temporal distinta a n2.

El método principal junto a la interfaz de menús, está en AnalisisAlgoritmos. Desde allí se llama a los métodos de la clase TestOrdenacion. TestOrdenacion, a su vez, se apoya en la clase ConjuntoInt para la generación de las estancias aleatorias, Mtime para medir el tiempo transcurrido, AlgoritmosOrdenacion para ejecutar los algoritmos, y para generar los archivos de GNUplot y visualizar los datos. Los distintos archivos usan los valores definidos en Constantes.

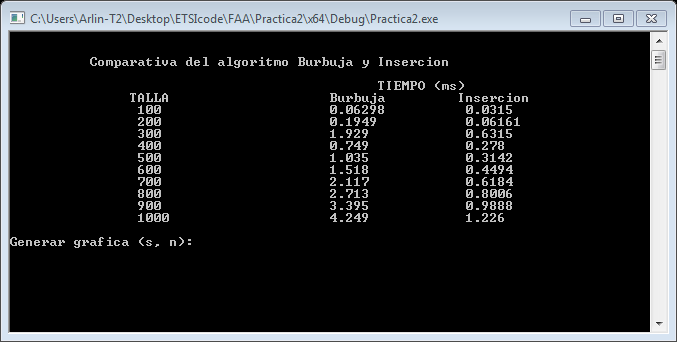
Menú principal, permite acceder a los submenús.



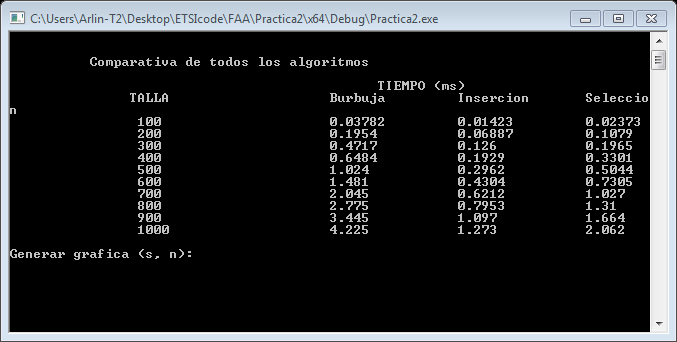
Prueba de los métodos de ordenación. Genera vectores aleatorios del tamaño especificado y los ordena con los distintos algoritmos implementados. Muestra por pantalla el algoritmo antes y después de ser ordenado.



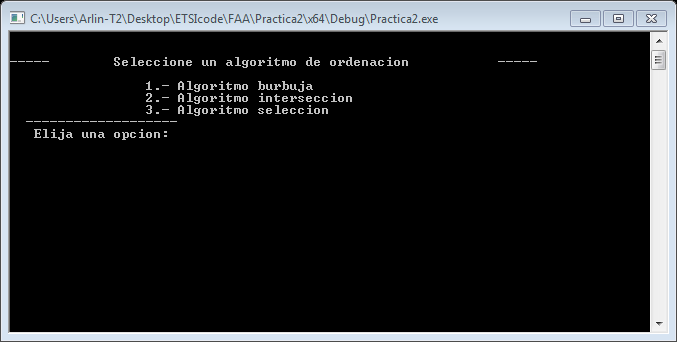
Caso medio de un algoritmo.  
Realiza ordenaciones para instancias de múltiples tallas. Cada operación se repite un numero de veces y se toma la media. Guarda los resultados en un archivo y genera una gráfica si el usuario lo desea.



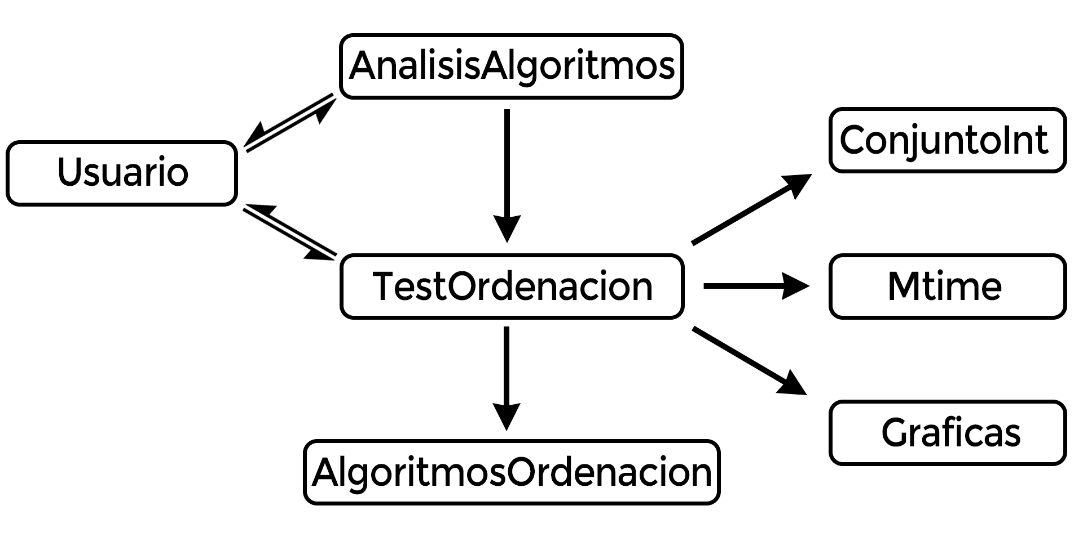
Comparación de dos métodos.  
Mide el coste de dos algoritmos al mismo tiempo. Muestra los resultados, los guarda en un archivo y genera una gráfica doble.



Comparar todos los métodos.  
Mide al mismo tiempo todos los algoritmos. Igualmente muestra los resultados, los guarda en un archivo y genera una gráfica con el número de algoritmos implementados.



Submenú de selección de algoritmos.



# Conclusiones y valoraciones personales de la práctica

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 y 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. La forma exponencial de la curva empieza a distinguirse antes cuantos más hilos tiene la máquina, en un portátil estándar si la talla es baja los resultados parecen no ceñirse a la formula teórica. Mientras que en una estación de trabajo puede una forma perfecta en la curva desde el inicio.

Nota: El programa que se adjunta incluye, no solo los algoritmos aquí estudiados, otros algoritmos de ordenación y búsqueda fueron implementados, como actividad completaría. La funcionalidad requerida se mantiene.

## Material consultado

[Visualización Burbuja](http://www.programming-algorithms.net/article/39344/Bubble-sort), [visualización Selección](http://www.programming-algorithms.net/article/39430/Selection-sort), [visualización Inserción](http://www.programming-algorithms.net/article/39459/Insertion-sort), [visualización QuickSort](http://www.programming-algorithms.net/article/39552/Quicksort)