Análisis de algoritmos de búsqueda

FAA

Por: Ismael Da Palma Fernández

ÍNDICE

3. Introducción. Algoritmos de búsqueda

El objetivo de esta práctica es realizar el estudio teórico de tres algoritmos de búsqueda, y posteriormente implementarlos en lenguaje de C++ y comparar sus eficiencias.

Al igual que en las prácticas anteriores, vamos a usar el Visual Studio 2019. Y adicionalmente el Gnuplot para realizar el muestreo de las gráficas a partir de los resultados obtenidos.

4. Cálculo del tiempo teórico

Se nos ha propuesto el pseudocódigo de tres algoritmos de búsqueda con el objetivo de implementarlos en la práctica, junto con los de ordenación, vistos en la práctica anterior. Todos estos algoritmos reciben como parámetros un vector de enteros(v[]), el tamaño del array(size) y la clave que debe de buscar(key). Estos algoritmos devolverán la posición en la que se encuentra la clave o -1 si no se ha encontrado.

4.1. Pseudocódigo y análisis de coste

➢ Busqueda Secuencial Iterativa (Busqueda Lineal):

```
int AlgoritmosBusqueda::busquedaSecuencialIt(int v[], int size,int key)
{
  //** ESCRIBIR PARA COMPLETAR LA PRACTICA **//
   int i = 0;
   while (v[i] != key && i <= size)
       i++;
   if (v[i] == key)
       return i;
   else
       return -1;
}
```

Tiene un funcionamiento muy sencillo. Realiza una una lectura secuencial de los elementos hasta encontrar la clave.

Si la clave se encuentra en el vector, la búsqueda acabará antes de llegar al final y devolverá la posición i-ésima donde ha encontrado el elemento.

Si no encuentra la clave, recorrerá todo el vector sin resultado alguno y devolverá un valor imposible (en este caso el -1).

Es el más sencillo y el más lento de los tres algoritmos de búsqueda (esto ya lo comprobaremos con las graficas de coste).

EFICIENCIA TEÓRICA:

- **Caso mejor:** La clave se encuentra en el primer elemento del vector, independientemente del tamaño, va a tardar siempre lo mismo. Por lo que su orden es constante O(1).

- **Caso peor:** La clave no se encuentra en el vector, por lo que recorrerá el vector entero (de n elementos). Por lo que su orden es de O(n).

- **Caso medio:** La clave se encuentra en zonas aleatorias del vector (en posiciones intermedias). Seguirá siendo de O(n) al igual que el caso peor, aunque su conteo de operaciones será menor.

ANÁLISIS POR CONTEO DE OPERACIONES:

$\rightarrow Caso\ Mejor\colon\ T(n) = 5 \qquad T(n) \in O(1)$

$\rightarrow Caso\ Peor\colon\ T(n) = 7 + 6n \qquad T(n) \in O(n)$

$\rightarrow Caso\ Medio\colon\ T(n) = 7 + 3n \qquad T(n) \in O(n)$

➢ Busqueda Binaria Iterativa:

```
int AlgoritmosBusqueda::busquedaBinariaIt(int v[], int size,int key)
{
    //** ESCRIBIR PARA COMPLETAR LA PRACTICA **//
    bool encontrado = false;
    int mitad, primero=0, ultimo=size;

    while (primero <= ultimo && !encontrado)
    {
        mitad = (primero + ultimo) / 2;
        if (key == v[mitad])
            encontrado = true;
        else
        {
            if (key < v[mitad])
                ultimo = mitad - 1;
            else
                if (key > v[mitad])
                    primero = mitad + 1;
        }
    }
    if (encontrado)
        return mitad; //Se encuentra
    else
        return -1; //No se encuentra el elemento
}
```

Para la búsqueda binaria es necesario que el array esté ordenado, por ello he hecho uso del algorimo más eficiente visto en la práctica anterior (Inserción).

Una vez ordenado el array, el algorimo va dividiendo el vector para obtener la clave a través del elemento mitad.

Si la clave está en la mitad hemos acabado y devolvemos la posición. Si no lo encuentra puede darse dos casos:

-Si la clave es menor que el elemento mitad, se cogerá el primer subconjunto.
-Si la clave es más grande que el elemento mitad, se seguirá buscando por el segundo subconjunto.

Es más eficiente que la busqueda secuencial ya que realiza menos comparaciones, aunque para poder usarlo correctamente es necesario que el vector esté ordenado.

EFICIENCIA TEÓRICA:

- **Caso mejor:** La clave se encuentra en el centro del vector, en su primera llamada.
- **Caso peor:** La clave no se encuentra en el vector, tiene que realizar divisiones hasta quedarse con un elemento.
- **Caso medio:** La clave se encuentra en el centro del vector, pero no en la primera llamada.

ANÁLISIS POR CONTEO DE OPERACIONES

$\rightarrow Caso\ Mejor$: $T(n) = 11$ $\quad T(n) \in O(1)$

$\rightarrow Caso\ Peor$: $T(n) = 6 + 9\sum_{0}^{n/2} i = 6 + 9\log n$ $\quad T(n) \in O(\log n)$

$\rightarrow Caso\ Medio$: $T(n) = 6 + \frac{9}{2}\log n$ $\quad T(n) \in O(\log n)$

➢ Busqueda por Interpolación Iterativa:

```
int AlgoritmosBusqueda::busquedaInterpolacionIt(int v[], int size,int key)
{
    //** ESCRIBIR PARA COMPLETAR LA PRACTICA **//
    int p, primero, ultimo;
    primero = 0;
    ultimo = size-1;

    while ((v[ultimo] >= key) && (v[primero] < key))
    {
        p = primero + ((ultimo - primero) * (key - v[primero]) / (v[ultimo] - v[primero]));
        if (key > v[p])
            primero = p + 1;
        else if (key < v[p])
            ultimo = p - 1;
        else
            primero = p;
    }

    if (v[primero] == key)
        return primero;
    else
        return -1;
}
```

Es una mejora con respecto a la búsqueda binaria.

Este algoritmo puede ir a diferentes ubicaciones del vector según el valor de la clave que se busca. Por ejemplo, si la clave está más cerca del último elemento del vector, la búsqueda comenzará por el final del vector.

En este algoritmo no se busca el elemento mitad del subconjunto sino el primero de este.

Es el algoritmo de búsqueda más eficiente de los tres, pero ocurre lo mismo que con la binaria (el array debe de estar ordenado para su correcto uso).

EFICIENCIA TEÓRICA: teniendo en cuenta el resultado de “p”.

- **Caso mejor:** La clave se encuentra tras el primer resultado de “p”.
- **Caso peor:** La clave no se encuentra en el array, realiza n cálculos de “p” y sale sin ningún resultado.
- **Caso medio:** La clave se encuentra en el array, pero no se obtiene después del primer cálculo de “p”.

COMPLEJIDAD: debido a la fórmula de "p"

$$Caso\ Mejor:\ T(n) \in O(\log\log n)$$

$$Caso\ Peor:\ T(n) \in O(n)$$

$$Caso\ Medio:\ T(n) \in O(\log\log n)$$

4.2. Conclusiones

Como era de esperar, el coste temporal del algoritmo depende de la talla del problema, como de la instancia. Aunque el algoritmo de Interpolación es más eficiente con vectores grandes y el binario con vectores pequeños.

- ➔ **Suponiendo vectores grandes**, el orden de eficiencia es: Selección << Binaria << Interpolación
- ➔ **Suponiendo vectores pequeños**, el orden de eficiencia es: Selección << Interpolación ≈ Binaria

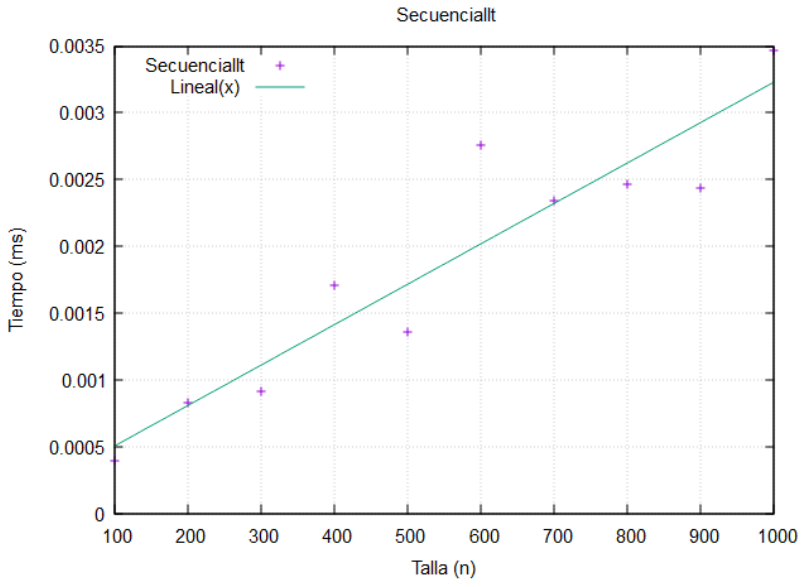
En cuanto a la complejidad espacial, todos ellos son de O(1), la cantidad de memoria consumida por el algoritmo no es proporcional al tamaño del vector (se hace uso de variables locales).

5. Cálculo del tiempo experimental

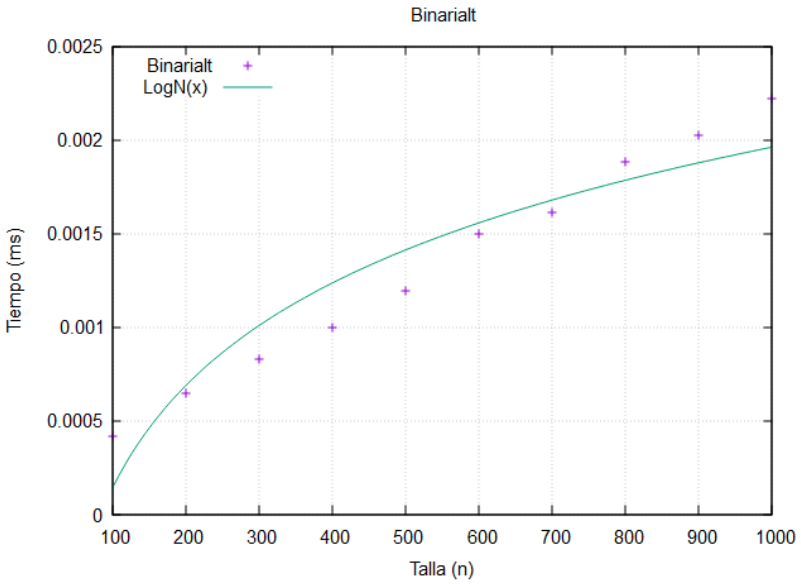
Al igual que en la práctica 2, sólo hemos tenido que calcular el tiempo medio de cada uno de los algoritmos de búsqueda, para ello hemos repetido cada talla de cada algoritmo un número de veces en concreto que viene indicado en el fichero "Constantes.h". En cada repetición se crea una clave al azar a través del método generarKey(), implementado en la clase ConjuntoInt, y se buscara si existe o no esa clave en el array. Una vez se han realizado las repeticiones se procede a hacer la media del tiempo.

5.1. Tablas y gráficas de coste

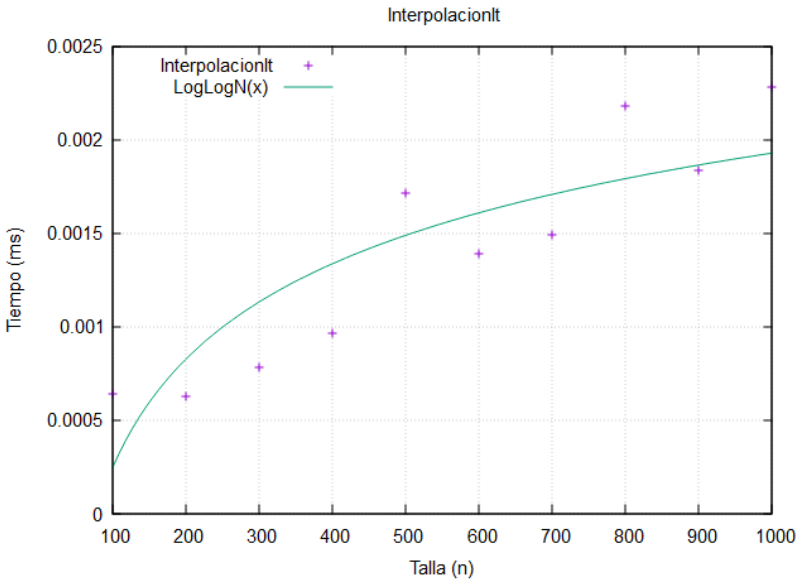
```
C:\Users\ismae\source\repos\Practica_3_FAA\Debug\Practica_

*** Busqueda SecuencialIt ***

Tiempos de ejecucion promedio


Talla           Tiempo <mseg>

100                0.00039
200                0.00083
300                0.00091
400                 0.0017
500                 0.0014
600                 0.0028
700                 0.0023
800                 0.0025
900                 0.0024
1000                0.0035

Datos guardados en el fichero SecuencialIt.dat

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



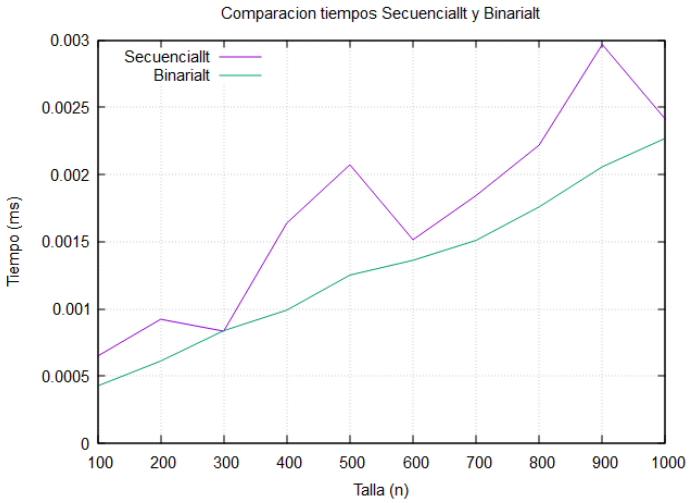
```
C:\Users\ismae\source\repos\Practica_3_FAA\Debug\Practic

                *** Busqueda BinariaIt ***

                Tiempos de ejecucion promedio


                Talla           Tiempo <mseg>

                100                0.00042
                200                0.00065
                300                0.00083
                400                 0.001
                500                0.0012
                600                0.0015
                700                0.0016
                800                0.0019
                900                 0.002
                1000               0.0022

Datos guardados en el fichero BinariaIt.dat

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



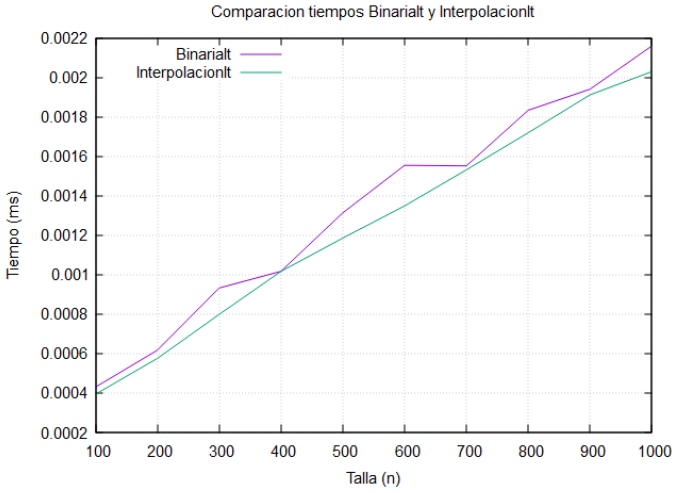

```
C:\Users\ismae\source\repos\Practica_3_FAA\Debug\Practica_3_

                *** Busqueda InterpolacionIt ***

                Tiempos de ejecucion promedio


                Talla           Tiempo <mseg>

                100                0.00064
                200                0.00063
                300                0.00078
                400                0.00097
                500                 0.0017
                600                 0.0014
                700                 0.0015
                800                 0.0022
                900                 0.0018
                1000                0.0023

Datos guardados en el fichero InterpolacionIt.dat

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



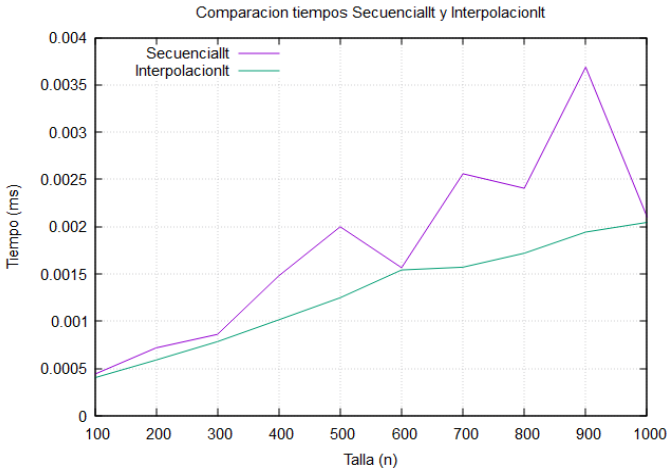
Ahora compararemos los algoritmos entre si para ver cual es el más eficiente.



Podemos observar que la búsqueda Secuencial es mucho más costosa que la Binaria. En algunos casos en concreto llegan a coincidir o a ser ligeramente iguales.



Aquí podemos comprobar que la búsqueda Binaria y de Interpolación tienen un comportamiento similar, podemos ver que la Interpolación es ligeramente más eficiente que la Binaria. Esto concuerda con lo visto en la teoría.



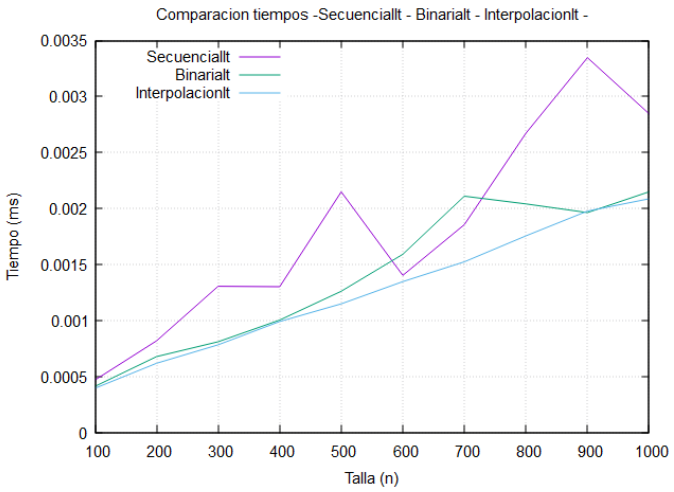
Se puede observar como la búsqueda por Interpolación es más eficiente en tallas altas, lo contrario ocurre con el de Secuencial.
En tallas bajas se puede ver como la Interpolación es casi igual al de Selección.

5.2. Conclusiones

De todas las comparaciones vistas en las gráficas anteriores concluimos que la Interpolación es el algoritmo de búsqueda más eficiente de los tres, aunque llega a ser ligeramente más eficiente que el Binario y ligeramente igual de eficiente que el Secuencial en tallas pequeñas.

Esto se puede ver de forma más clara con la cuarta opción del menú de búsqueda (comparar todos los algoritmos).

```
C:\Users\ismae\source\repos\Practica_3_FAA\Debug\Practica_3_FAA.exe

                    *** COMPARACION DE TODOS LOS METODOS DE BUSQUEDA ***

                                Tiempos de ejecucion promedio

                          SecuencialIt          BinariaIt             InterpolacionIt

      Talla<n>          Tiempo <mseg>         Tiempo <mseg>          Tiempo <mseg>

      100                0.00047                0.00042               0.0004
      200                0.00082                0.00068               0.00062
      300                 0.0013                0.00081               0.00078
      400                 0.0013                0.001                 0.00099
      500                 0.0021                0.0013                0.0012
      600                 0.0014                0.0016                0.0013
      700                 0.0019                0.0021                0.0015
      800                 0.0027                0.002                 0.0018
      900                 0.0033                0.002                 0.002
      1000                0.0028                0.0021                0.0021

Datos guardados en los ficheros SecuencialItBinariaItInterpolacionIt.dat

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



6. Comparación de los resultados teórico y experimental

Los resultados experimentales reflejan la misma relación entre los algoritmos. Siendo la búsqueda Secuencial la más lenta, Binaria e Interpolación bastante parecidos.

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

Un análisis por conteo de operaciones elementales no ofrece los mejores resultados. Ya que es importante notar que, aunque la búsqueda Secuencial es la que más comparaciones realiza, también es la que menos instrucciones por comparación tiene. Mientras que la de Interpolación realiza menos comparaciones, también necesita más operaciones.

7. Diseño de la aplicación

El código y la estructura de esta practica es la misma que la de la práctica 2, pero en este caso se ha realizado una pequeña mejora en el **"Principal.cpp"**. Se ha creado una nueva clase llamada "Menus" que contiene todos los menús y submenús que se usan en la práctica, así como las opciones a elegir, dejando en el main solo los métodos a los que llama cada case del switch (sin ningún cout o cin). Esto se ha realizado para una mejor comprensión del código y evitar perderse.

Como la mayoría de las clases ya están explicadas en la práctica 2, procederé a explicar sólo el contenido de lo nuevo que se ha añadido a la práctica 3:

"AlgoritmosBusqueda" contiene los algoritmos vistos en esta memoria para buscar un elemento en un vector de enteros. Dos de ellos necesitan que el vector esté ordenado para poder funcionar de forma correcta, para ello hacemos uso de uno de los algoritmos de la clase **"AlgoritmosOrdenación"** visto en la practica 2.

"Menus" contiene los menús y submenús de los que va a hacer uso el **"Principal.cpp"** durante su ejecución.

"TestBusqueda" es la clase más importante de esta gráfica, ya que es la que nos permite comprobar el correcto funcionamiento de los algoritmos de búsqueda y el cálculo medio de uno, dos y todos los algoritmos de búsqueda.
Tiene los siguientes métodos principales:

- **BuscarArrayDeInt:** busca la clave en el vector que se pasa por parámetro a través del algoritmo que se le indica en la llamada.

- **ComprobarMetodosBusqueda:** Comprueba que todos los algoritmos de búsqueda funcionan correctamente recibiendo por teclado la clave a buscar para cada algoritmo. Este método llama a BuscarArrayDeInt.

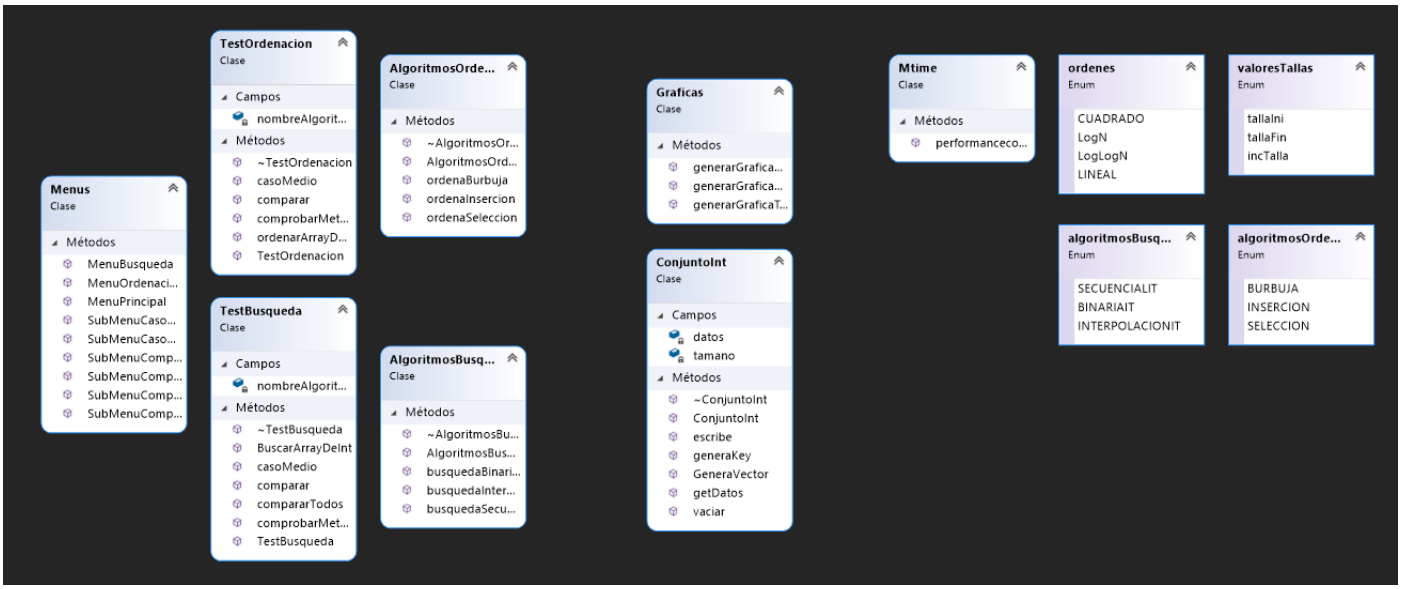
- **casoMedio:** Calcula el caso medio del algoritmo de búsqueda pasado por parámetro, los datos obtenidos se muestran por pantalla y se guardan en un fichero con el nombre del algoritmo y permite generar una gráfica conforme a esos resultados.

- **comparar:** Compara dos métodos de búsqueda pasados por parámetro, muestra los datos por pantalla y los guarda en un fichero. También da la posibilidad de generar una gráfica que se ajusta a esos datos.

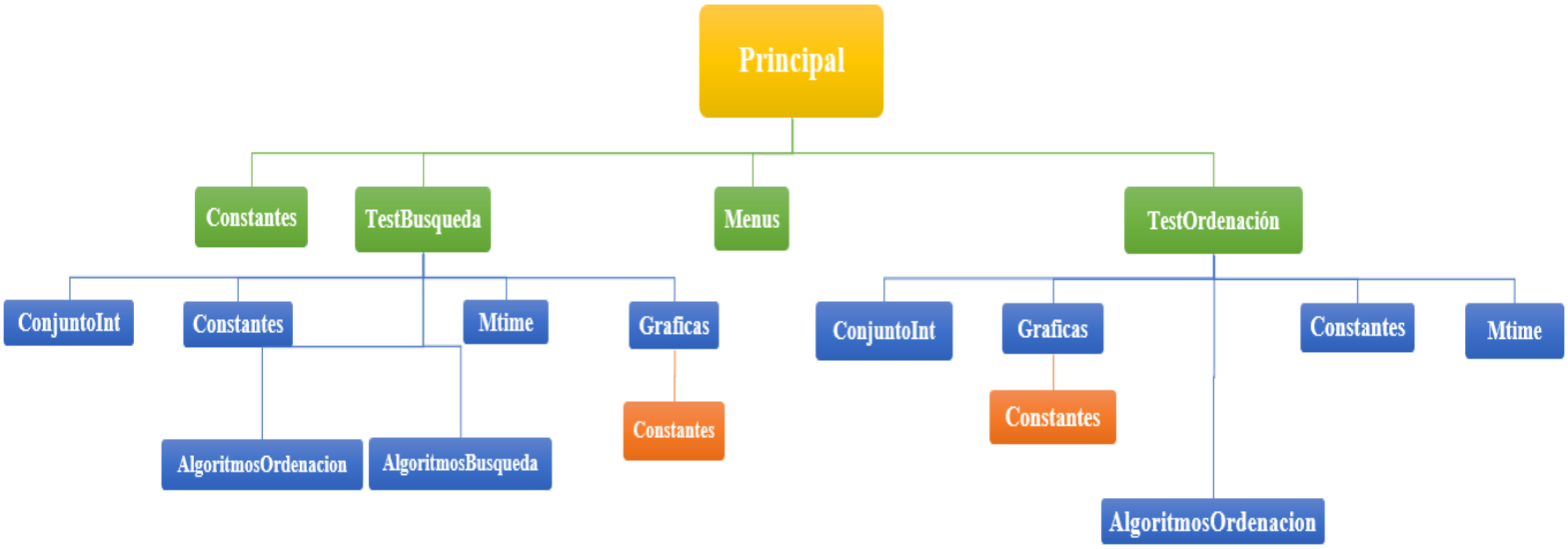
- **compararTodos:** hace lo mismo que los dos métodos anteriores pero con todos los algoritmos de búsqueda.

  Los métodos de casoMedio, comprar y compararTodos también hacen uso de la clase AlgoritmosOrdenación para ordenar el array en cada pasada antes de empezar a contar el tiempo de ejecución del algoritmo.

Diagrama de clases de la práctica 3 con el contenido de cada una de ellas:



Grafica que muestra la relación existente entre las clases de la práctica:



8. Conclusiones y valoraciones personales de la práctica

La búsqueda es una de las operaciones de tratamiento de vectores más habituales.

Una búsqueda en un vector no ordenado tendría una complejidad O(n), por esa razón es preferible implementar algoritmos de búsqueda en vectores ordenados.

Hemos estudiado 3 algoritmos de búsqueda:

- ➔ Búsqueda secuencial
- ➔ Búsqueda binaria
- ➔ Búsqueda por interpolación

El primer algoritmo es el más simple de los tres, recorre completamente el vector y no precisa que el vector esté ordenado. Su complejidad es O(n).

El algoritmo de búsqueda binaria divide en cada iteración el vector en dos, realizando la búsqueda en una sola de las mitades. Su complejidad es O(log n).

La búsqueda por interpolación es una modificación del algoritmo de búsqueda binaria, con una complejidad O(loglogn).