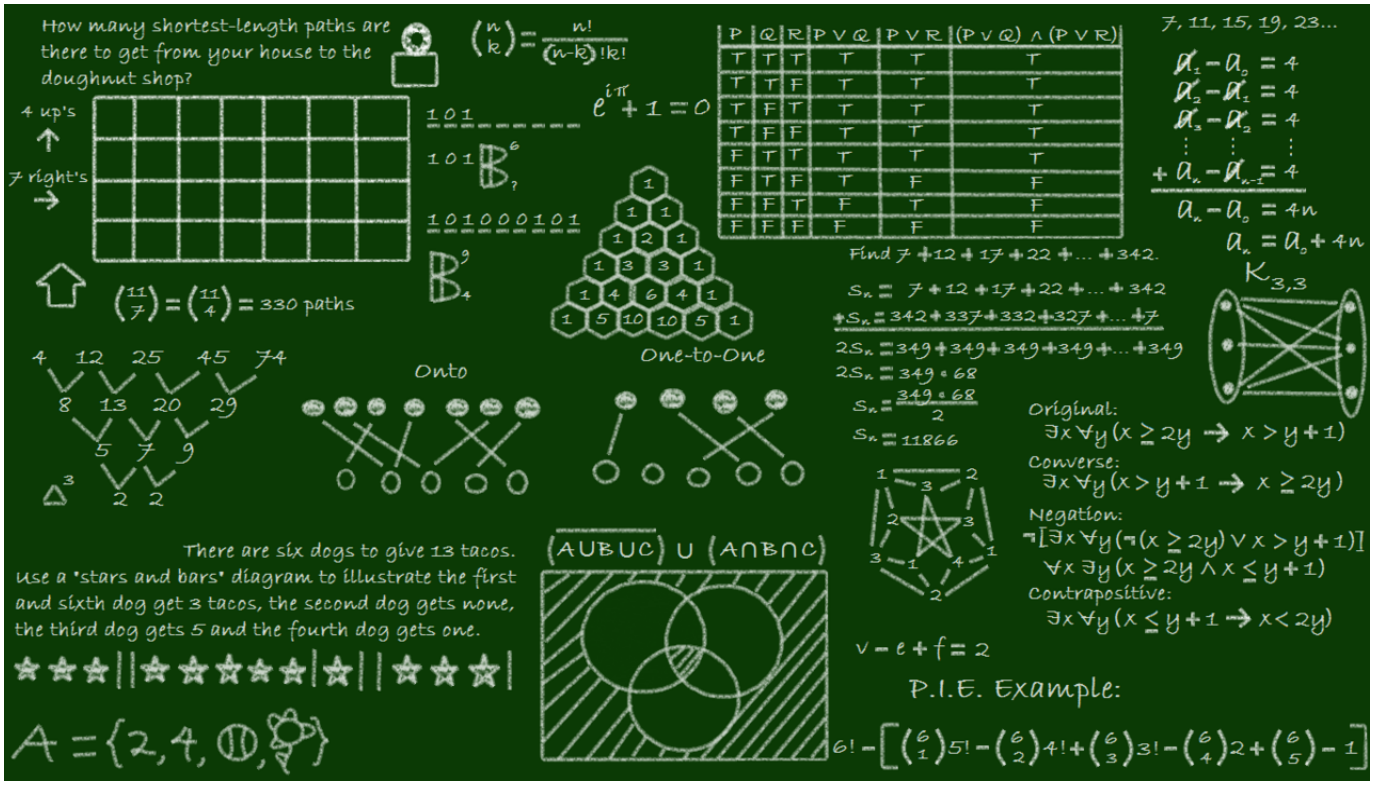
# Memoria de la práctica




Antonio Pablo García Sanabria y José María González Abad

Práctica 3

Análisis experimental de algoritmos de Ordenación

Fundamentos de Análisis de Algoritmos

Universidad de Huelva

09/05/2021

# Índice

# 1. Introducción. Algoritmo de Búsqueda

En esta práctica se realiza el estudio experimental y comparación de los algoritmos de búsqueda **Secuencial**, **Binaria** e **Interpolación**.

# 2. Cálculo de los tiempos teóricos

A partir de la expresión de cada algoritmo, habrá que obtener la expresión de la función complejidad temporal (*T(n)*) correspondiente al algoritmo. El análisis lo realizaremos para los casos mejor, peor y medio.

## 2.1. Pseudocódigo y análisis de coste

A continuación, se exponen los pseudocódigos de los diferentes algoritmos, así como un análisis de coste, de los casos Mejor, Peor y Medio, para cada uno de los algoritmos.

### 2.1.1. Algoritmo Secuencial iterativo

```
Algoritmo BusquedaSecuencialIt(V[1...n], size, clave)
      comienza
            i ← 1
            mientras (V[i] ≠ clave AND i ≤ size) hacer
                  i ← i + 1
            fmientras

            si (V[i] == clave) hacer
                  devolver i
            sino
                  devolver -1
            fsi
      termina
```

Para cada caso aplicaremos diferentes cálculos de tiempo:

Para el caso mejor: $T(n) = 7$

Para el caso medio: $T(n) = 7 + 3*n$

Para el caso peor: $T(n) = 7 + 6*n$

### 2.1.2. Algoritmo Binario Iterativo

```
Algoritmo BusquedaBinariaIt(V[1...n], size, clave)
      comienza
            encontrado ← falso
            mitad : entero
            primero ← 1
            último ← size
```

```
          mientras (primero ≤ último AND !encontrado) hacer
               mitad ← ((primero + último) / 2)
               si clave == V[mitad] hacer
                    encontrado ← cierto
               sino
                    si clave < V[mitad] hacer
                         último ← mitad - 1
                    sino
                         si clave > V[mitad] hacer
                              primero ← mitad + 1
                         fsi
                    fsi
               fsi
          fmientras

          si encontrado
               devolver mitad
          sino
               devolver -1
          fsi
     termina
```

Para cada caso aplicaremos diferentes cálculos de tiempo:

Para el caso mejor: *T(n) = 14*

Para el caso medio: *T(n) = 8 +6*n*

Para el caso peor: *T(n) = 8 +12*n*

### 2.1.3. Algoritmo Interpolación Iterativo

```
Algoritmo BusquedaInterpolacionIt(V[1...n], size, clave);
     comienza
          p : int
          primero ← 1
          último ← size
          mientras (V[último] ≥ clave AND V[primero] < clave) hacer
  p ← primero + ((último - primero) * (clave - V[primero])/(V[último] -
V[primero]))
               si clave > V[p] hacer
                    primero ← p + 1
               sino
                    si clave < V[p] hacer
                         último ← p - 1
                    sino
                         primero ← p
                    fsi
               fsi
          fmientras
```

```
        si V[primero] == clave hacer
              devolver primero
        sino
              devolver -1
        fsi
    termina
```

Por la complejidad de este algoritmo, no realizamos el estudio de sus diferentes casos.

## 2.2. Tablas y Gráficas de coste

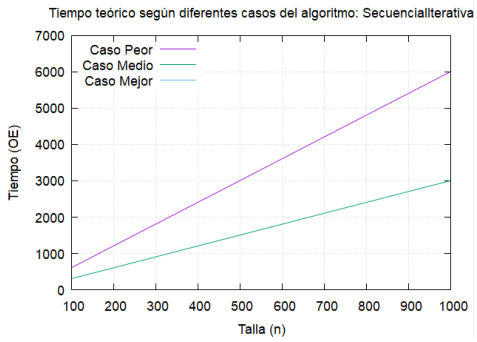
### 2.2.1. Algoritmo Secuencial Iterativo

```
Tiempo en operacion elementales segun diferentes casos del algoritmo: SecuencialIterativa
Operaciones elementales (OE)

       Talla   Tiempo Caso Peor       Tiempo Caso Medio      Tiempo caso Mejor

       100             6.1e+02        3.1e+02             7
       200             1.2e+03        6.1e+02             7
       300             1.8e+03        9.1e+02             7
       400             2.4e+03        1.2e+03             7
       500               3e+03        1.5e+03             7
       600             3.6e+03        1.8e+03             7
       700             4.2e+03        2.1e+03             7
       800             4.8e+03        2.4e+03             7
       900             5.4e+03        2.7e+03             7
       1000              6e+03          3e+03             7

Datos guardados en el fichero SecuencialIterativaTeorico.dat
```



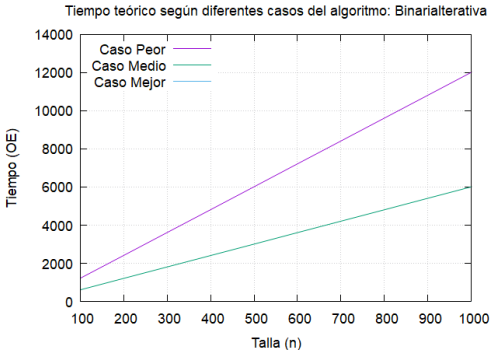

### 2.2.2. Algoritmo Binario Iterativo

```
Tiempo en operacion elementales segun diferentes casos del algoritmo: BinariaIterativa
Operaciones elementales (OE)

       Talla   Tiempo Caso Peor       Tiempo Caso Medio      Tiempo caso Mejor

       100             1.2e+03        6.1e+02             14
       200             2.4e+03        1.2e+03             14
       300             3.6e+03        1.8e+03             14
       400             4.8e+03        2.4e+03             14
       500               6e+03          3e+03             14
       600             7.2e+03        3.6e+03             14
       700             8.4e+03        4.2e+03             14
       800             9.6e+03        4.8e+03             14
       900             1.1e+04        5.4e+03             14
       1000            1.2e+04          6e+03             14

Datos guardados en el fichero BinariaIterativaTeorico.dat
```




### 2.2.3. Algoritmo Interpolación Iterativo

Puesto que este algoritmo es muy complejo, no podemos sacar sus fórmulas de complejidad para obtener posteriormente las tablas y gráficas de coste.

## 2.3. Conclusiones

Observamos que el *Algoritmo de Interpolación Iterativo* es claramente el más eficiente, con una única excepción, que es cuando usamos el *Algoritmo Secuencial Iterativo* y el elemento a buscar se encuentra en la primera posición, en este caso solo se produce una operación y, por tanto, no puede haber nada más eficiente.

# 3. Cálculo del tiempo experimental

## 3.1. Tablas y gráficas de coste

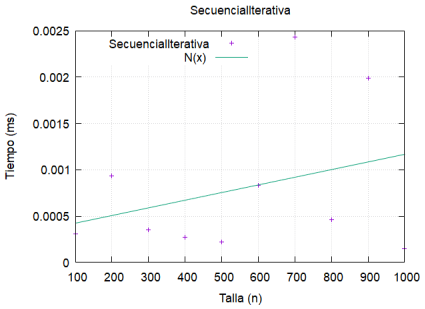
### 3.1.1. Algoritmo Secuencial Iterativo

```
Tiempo promedio de búsqueda por SecuencialIterativa
Tiempos de ejecucion

        Talla           Tiempo (ms)

        100               0.00031
        200               0.00093
        300               0.00035
        400               0.00027
        500               0.00022
        600               0.00083
        700                0.0024
        800               0.00046
        900                 0.002
        1000              0.00015

Datos guardados en el fichero SecuencialIterativa.dat
```



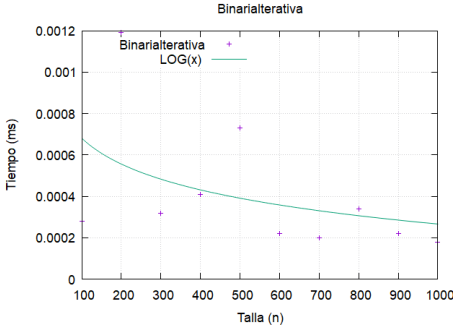

### 3.1.2. Algoritmo Binario Iterativo

```
Tiempo promedio de búsqueda por BinariaIterativa
Tiempos de ejecucion

        Talla           Tiempo (ms)

        100               0.00028
        200                0.0012
        300               0.00032
        400               0.00041
        500               0.00073
        600               0.00022
        700                0.0002
        800               0.00034
        900               0.00022
        1000              0.00018

Datos guardados en el fichero BinariaIterativa.dat
```



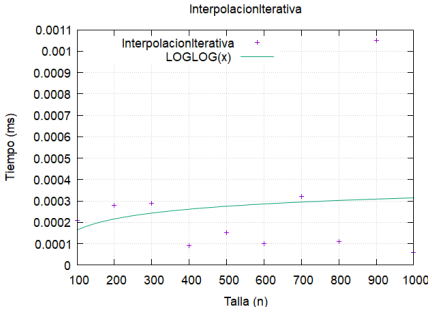

### 3.1.3. Algoritmo Interpolación Iterativo

```
Tiempo promedio de búsqueda por InterpolacionIterativa
Tiempos de ejecucion

        Talla           Tiempo (ms)

        100               0.00021
        200               0.00028
        300               0.00029
        400                 9e-05
        500               0.00015
        600                0.0001
        700               0.00032
        800               0.00011
        900                 0.001
        1000                6e-05

Datos guardados en el fichero InterpolacionIterativa.dat
```




## 3.2. Conclusiones

Al igual que en el análisis teórico el más veloz es el *Algoritmo de Interpolación Iterativo*, pues es este el que resuelve con mayor velocidad su caso medio para vectores de diferentes tamaños.

# 4. Comparación de los resultados teórico y experimental

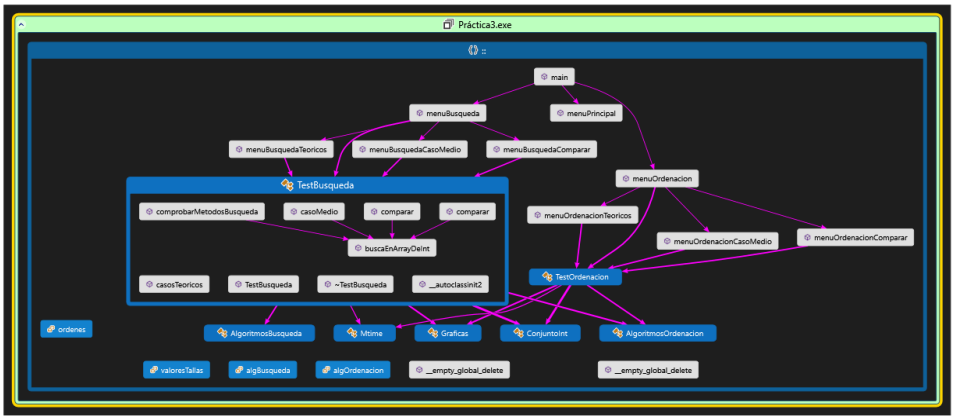
La diferencia general de los dos resultados es que estamos midiendo dos unidades diferentes: operaciones elementales (en el caso teórico) y tiempo en milisegundos (en el caso empírico). Ciertamente, estas operaciones elementales van a ser muchas más o menos según el algoritmo de búsqueda que esté en uso y la talla del vector a procesar.

# 5. Diseño de la aplicación

La aplicación muestra el estudio de las operaciones que realiza el algoritmo de búsqueda secuencial y el tiempo que tardan en ejecutarse según los diferentes casos. Está compuesta de los archivos de encabezado, de sus correspondientes archivos de origen, *ConjuntoInt.h/.cpp*, *Constantes.h*, *Mtime.h/.cpp*, *TestBusqueda.h/.cpp, TestOrdenacion.h/.cpp, Graficas.h/.cpp, AlgoritmosBusqueda.h/.cpp, AlgoritmosOrdenacion.h/.cpp* y *Principal.cpp*, además de otra serie de archivos donde se recogen los datos obtenidos en la aplicación y las gráficas generadas a partir de dichos datos.

La clase *ConjuntoInt* nos ayuda a trabajar los vectores para el estudio, con diferentes métodos, como generar o vaciar vectores. En la clase *AlgoritmosBusqueda* implementamos los diferentes algoritmos para su posterior uso. Mediante los métodos de la clase *TestBusqueda* podemos realizar los diferentes estudios de los algoritmos. Gracias a la clase *Mtime* calculamos el tiempo que tarda en ejecutarse cada caso.

En el fichero *Principal* se lleva a cabo la ejecución de la aplicación, mediante la navegación a través de varios menús en los que podemos decidir que estudio en concreto queremos realizar. A continuación, mostramos el mapa de código generado por la aplicación IDE.




# 6. Conclusiones y valoraciones personales de la práctica

En esta prácticas unimos lo visto anteriormente de algoritmos de búsqueda y algoritmos de ordenación para hacer un uso más complejo de ellos, ya que tanto para la búsqueda binaria y la interpolación era necesario que los vectores estuvieran ordenados. Puesto que la complejidad de todos los algoritmos estudiados es diferente, realizamos un estudio independiente de cada uno para su posterior comparación.