

1.	Objetivo.....	2
2.	Introducción. Análisis de algoritmos.	2
3.	Enunciado.....	2
4.	Estudio práctico.	2
4.1.	Cálculo del Tiempo Teórico.....	2
4.1.1.	Caso mejor.	3
4.1.2.	Caso peor.	3
4.1.3.	Caso medio.	3
4.1.4.	Conclusión.	3
4.1.5.	Cómo Mostrar los Datos.....	3
4.2.	Cálculo del Tiempo de la Eficiencia Empírica	5
4.2.1.	Cálculo del tiempo.	5
4.2.2.	Cómo Mostrar los Datos.....	7
5.	Descripción del problema	8
5.1.	Definición de la clase base ConjuntoInt.	8
5.2.	La clase TestAlgoritmo.....	8
5.3.	La clase Mtime.	8
5.4.	Las Gráficas.	8
6.	Actividades a desarrollar.	8
6.1.	Programa principal.	9
6.2.	Conclusiones.	12
7.	Entrega de la práctica.....	12
7.1.	Esquema para la memoria.	12
8.	Consideraciones para la implementación:.....	13
9.	Anexo. Resumen de los pasos a seguir para realizar la práctica 1.	14

1. Objetivo.

El objetivo de estas sesiones es comprender la importancia de analizar la eficiencia de los algoritmos y familiarizarse con las formas de llevarlo a cabo.

La práctica consiste en la programación de una aplicación que calcule el tiempo de ejecución de un algoritmo.

2. Introducción. Análisis de algoritmos.

Para analizar la eficiencia se mostrará cómo realizar un análisis del algoritmo mediante estudio teórico y empírico. Este estudio será requerido en el resto de prácticas de la asignatura.

El análisis del algoritmo conlleva:

Estudio teórico: En vista de la implementación, habrá que obtener la expresión de la función complejidad temporal ($T(n)$) correspondiente al algoritmo.

Estudio Empírico: estudiar experimentalmente el comportamiento del algoritmo. Para ello mediremos el tiempo para cada tamaño dado de las entradas, obteniendo los puntos de la función complejidad para los casos peor, mejor y medio (o único si no es por casos).

Conclusión: para finalizar el análisis habrá que realizar una gráfica para cada una de las tres funciones obtenidas en el estudio empírico, si el análisis es por casos (sino sólo una), y comentarlas, es decir, si son coherentes el resultado empírico con el teórico obtenido, justificando la respuesta.

3. Enunciado.

En esta práctica realizaremos el estudio teórico y práctico para el algoritmo de Búsqueda Lineal (secuencial) visto en clase.

4. Estudio práctico.

4.1. Cálculo del Tiempo Teórico

A partir de la expresión del algoritmo, se aplicarán las reglas conocidas para contar el número de operaciones que realiza un algoritmo. Este valor será expresado como una función de $T(n)$ que dará el número de operaciones requeridas para un caso concreto del problema caracterizado por tener un tamaño n . El análisis lo realizaremos para los casos mejor, peor y medio.

ALGORITMO DE BÚSQUEDA SECUENCIAL. Para determinar el tiempo de ejecución, calculamos primero el número de operaciones elementales (OE) que se realizan (en pseudocódigo o en código C++):

líneas	<code>int BusquedaSecuencial(int T[],int n,int valor)</code>	nº OE	
	{		
1)	<code>int i=0;</code>	1	asignación
2)	<code>while (T[i] != valor && i<n) {</code>	4	Condición del Bucle (2comp., 1 ac. vector, 1 lóg.)
3)	<code> i=i+1;</code>	2	incremento y asignación
4)	<code>}</code>		
5)	<code>if (T[i]==valor)</code>	2	1 condición y 1 acc. al vector
6)	<code> return i;</code>	1	si la condición se cumple
7)	<code>else return -1;</code>	1	cuando condición es falsa.
	<code>}</code>		

El tiempo de ejecución del algoritmo será:

$$T_{\text{BSecuencial}}(n) = T_{\text{Asig}}(1) + T_{\text{Bucle}}(2) + T_{\text{Si}}(5)$$

Para calcularlo lo haremos por partes:

$$T_{\text{Asig}(1)} = 1$$

$$T_{\text{Si}(5)} = T_{\text{condSi}} + T_{\text{cuerpoSi}} = 2 + \text{máx/mín/medio}(T_{\text{return}(6\text{ó}7)}) = 2 + 1 = 3$$

$$T_{\text{Bucle}(2)} = T_{\text{condB}} + T_{\text{saltoB}} + \sum(i=1;?) T_{\text{cicloBucle}} = 4 + 1 + \sum(i=1;?) T_{\text{cicloBucle}}$$

$$T_{\text{cicloBucle}} = T_{\text{condB}} + T_{\text{cuerpoB}} (= 0 \text{ sólo instrucción de incremento del bucle}) + T_{\text{incrementoB}} + T_{\text{saltoCicloB}} = 4 + 2 + 1 = 7$$

$$T_{\text{Bucle}(2)} = 4 + 1 + \sum(i=1;?) 7 = 5 + 7 \sum(i=1;?)$$

$$T_{\text{BSecuencial}}(n) = T_{\text{Asig}(1)} + T_{\text{Bucle}(2)} + T_{\text{Si}(5)} = 1 + 5 + 7 \sum(i=1;?) + 3 = 9 + 7 \sum(i=1;?)$$

$$T_{\text{BSecuencial}}(n) = 9 + 7 \sum(i=1;?)$$

4.1.1. Caso mejor.

En el caso mejor para el algoritmo, se efectuará la línea (1) y la línea (2) que supone 4 OE. Tras ellas la función acaba ejecutando las líneas (5) a (7). En consecuencia, el ciclo del bucle no se ejecuta nunca:

$\sum(i=1;0) = 0$ y por tanto,

$$T_{\text{BSecuencial}}(n) = 9 + 7 \sum(i=1;0) = 9$$

4.1.2. Caso peor.

En el caso peor sucede cuando el valor no se encuentra en el vector. Se efectúa la línea (1), el bucle se repite n veces hasta que se cumple la segunda condición, después se efectúa la condición de la línea (5) y la función acaba al ejecutarse la línea (7). Cada iteración del bucle está compuesta por las líneas 2 (4OE) y 3 (2OE), junto con una ejecución adicional de la línea (2) que es la que ocasiona la salida del bucle. En consecuencia, el ciclo del bucle se ejecuta n veces:

$\sum(i=1;n) = n$ y por tanto,

$$T_{\text{BSecuencial}}(n) = 9 + 7 \sum(i=1;n) = 7n + 9$$

4.1.3. Caso medio.

En el caso medio el bucle se ejecutará un n^0 de veces entre 1 y n , y suponemos que cada una de ellas tiene la misma probabilidad de suceder. En consecuencia, el ciclo del bucle se ejecuta $n/2$ veces:

$\sum(i=1;n/2) = n/2$ y por tanto,

$$T_{\text{BSecuencial}}(n) = 9 + 7 \sum(i=1;n/2) = 9 + 7(n/2) = (7/2)n + 9$$

4.1.4. Conclusión.

Su **caso mejor** se da cuando el elemento está en la primera posición del vector $O(1)$.

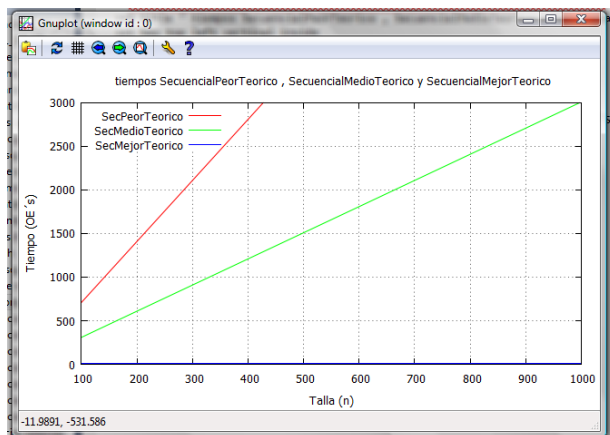
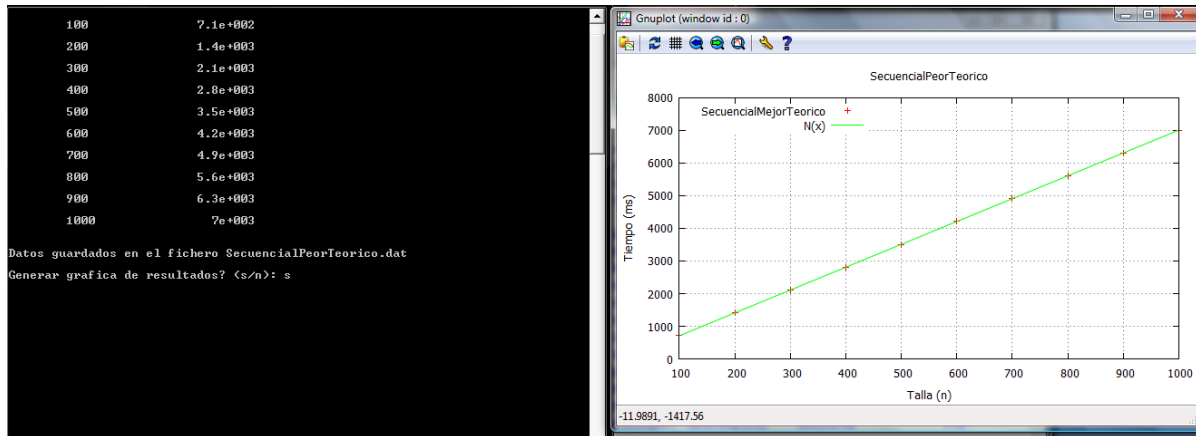
El **caso peor** se produce cuando el elemento no está en el vector.

El **caso medio** ocurre cuando consideramos equiprobables cada una de las posiciones en las que puede encontrarse el elemento dentro del vector (incluyendo la posición especial -1, que indica que el elemento a buscar no se encuentra en el vector) pero ambos son de $O(n)$.

4.1.5. Cómo Mostrar los Datos

Para mostrar la eficiencia teórica haremos uso de los ficheros (tablas) que recojan el tiempo invertido para cada caso y para cada algoritmo.

También utilizaremos la representación gráfica realizada a partir de los datos almacenados en los ficheros.



4.2. Cálculo del Tiempo de la Eficiencia Empírica

Se trata de llevar a cabo un estudio puramente empírico, es decir, estudiar experimentalmente el comportamiento del algoritmo. Para ello mediremos los recursos empleados (tiempo) para cada tamaño dado de las entradas.

El estudio experimental de la eficiencia de un algoritmo conlleva la elaboración de conjuntos de prueba que reflejen condiciones diferentes de funcionamiento del mismo (caso en que el algoritmo se comporta mejor, en que se comporta peor, aleatorio, etc.) realizado para volúmenes de datos, o tallas, crecientes del problema. A medida que se generan los distintos casos de prueba, o bien posteriormente, se somete el, o los algoritmos, a dichos casos, determinándose para cada uno de ellos una medida del esfuerzo computacional necesario para su resolución (dicha medida suele ser el tiempo de ejecución).

En el caso de **estudios del tiempo promedio** será necesario generar, para cada posible talla, diferentes instancias aleatorias, de forma que la medida final sería un promedio de las efectuadas individualmente

Por último, los resultados se tabulan y se presentan adecuadamente en pantalla y se graban en **ficheros de datos y ficheros gráficos**.

En el caso del algoritmo de búsqueda el tamaño viene dado por el número de componentes del vector a ordenar (n).

4.2.1. Cálculo del tiempo.

Para obtener el tiempo empírico de una ejecución de un algoritmo lo que vamos a hacer es calcular el tiempo de CPU. El tiempo que toma una función es muy simple:

1. tomamos el valor del reloj antes de realizar la llamada (t_{ini}),
2. llamamos a la rutina en cuestión, y
3. tomamos nuevamente el valor del reloj (t_{fin}).
4. La diferencia entre $t_{fin} - t_{ini}$ nos da el total de tiempo que tomó:
 1. hacer la llamada a la rutina,
 2. que esta haga su trabajo,
 3. que devuelva el resultado.

Ahora hay algunos pequeños detalles de implementación. Por ejemplo, ¿qué función usar para tomar el tiempo del reloj? Y más importante, ¿qué precisión obtenemos con dicha función?.

Para tomar el tiempo podemos usar la rutina `clock()`, que devuelve el tiempo aproximado de CPU que transcurrió desde que nuestro programa fue iniciado, dicho tiempo representado en un valor de tipo `clock_t`: un valor entero que indica una cantidad de "tics" de reloj.

La precisión que tenemos con dicha rutina es de `CLOCKS_PER_SEC` (tics de reloj por segundo), lo que significa que por cada segundo que pasa, la función `clock()` nos devolverá `CLOCKS_PER_SEC` unidades más que el valor anterior. Según la plataforma utilizada dicho valor `CLOCKS_PER_SEC` varía entre 1000 a 1000000.

Una vez dicho esto, el código de arriba en la mayoría de los casos no funciona. Así que tenemos que buscar una función con mayor precisión, y además, promediar varias muestras.

Esta alternativa en Windows no sirve y la razón es sencilla, en la misma MSDN explican que el temporizador del sistema corre aproximadamente a unos 10 milisegundos, por lo tanto, cualquier función que lo utilice nos estará dando la misma poca precisión (incluso al utilizar `GetSystemTimeAsFileTime` y `FILETIME`). Por lo tanto la solución es utilizar lo que se conoce en el mundo de Windows como el "contador de rendimiento de alta resolución" (high-resolution performance counter: que tendremos que utilizar así:

```

#ifndef _LIB_MTIME
#define _LIB_MTIME

#include <stdio.h>
#include <windows.h>

/* retorna "a - b" en segundos */
double performancecounter_diff(LARGE_INTEGER *a, LARGE_INTEGER *b)
{
    LARGE_INTEGER freq;
    QueryPerformanceFrequency(&freq);
    return (double)(a->QuadPart - b->QuadPart) / (double)freq.QuadPart;
}

#endif

/* Uso
int main(int argc, char *argv[])
{
    LARGE_INTEGER t_ini, t_fin;
    double secs;

    QueryPerformanceCounter(&t_ini);
    /* ...hacer algo... */
    /*
    QueryPerformanceCounter(&t_fin);

    secs = performancecounter_diff(&t_fin, &t_ini);
    printf("%.16g milliseconds\n", secs * 1000.0);
    return 0;
}
*/

```

En este caso, `QueryPerformanceCounter` es como `clock()` y `QueryPerformanceFrequency` es como `CLOCKS_PER_SEC`. Es decir, la primera función nos da el valor del contador, y la segunda su frecuencia (en ciclos por segundo, `hertz`). Cabe aclarar que un `LARGE_INTEGER` es una forma de representar un entero de 64 bits por medio de una unión (*union*).

Para obtener la eficiencia empírica deberemos ejecutar el mismo algoritmo para diferentes ejemplos. Así para un algoritmo como el de búsqueda lo ejecutaremos para diferentes tamaños del vector y obtendremos los tiempos. Estos tiempos los almacenaremos en un fichero.

Para medir el tiempo con la función anterior y su utilización en las prácticas de la asignatura, se ha implementado la clase **Mtime**, ficheros **Mtime.h** y **Mtime.cpp**, su uso es como en el ejemplo anterior.

4.2.2. Cómo Mostrar los Datos

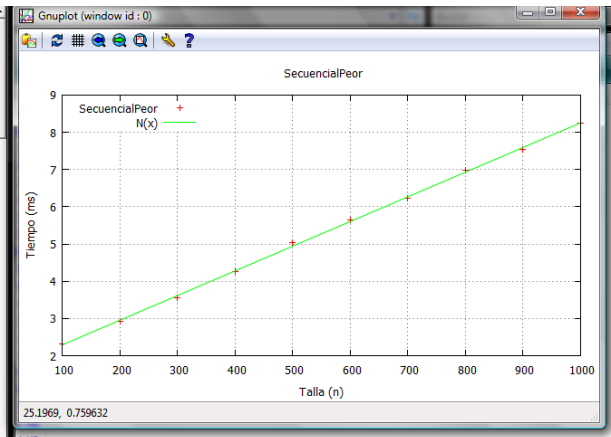
Para mostrar la eficiencia empírica haremos uso de los ficheros (tablas) que recojan el tiempo invertido para cada caso y para cada algoritmo.

También utilizaremos la representación gráfica realizada a partir de los datos almacenados en los ficheros.

Busqueda SecuencialPeor. Tiempos de ejecucion promedio

Talla	Tiempo (nseg)
100	2.3
200	2.9
300	3.6
400	4.3
500	5
600	5.7
700	6.2
800	7
900	7.5
1000	8.2

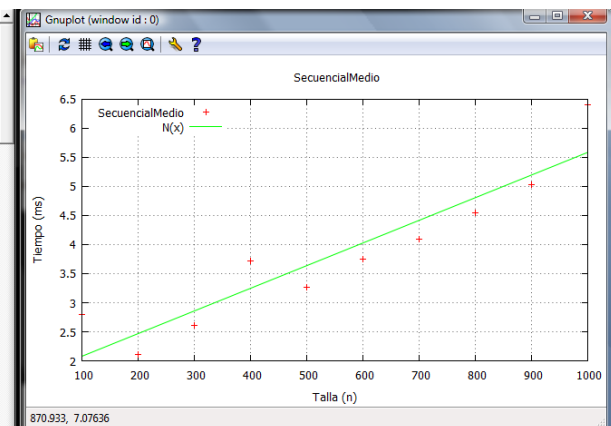
Datos guardados en el fichero SecuencialPeor.dat
Generar grafica de resultados? (s/n): s



Busqueda SecuencialMedio. Tiempos de ejecucion promedio

Talla	Tiempo (nseg)
100	2.8
200	2.1
300	2.6
400	3.7
500	3.3
600	3.8
700	4.1
800	4.6
900	5
1000	6.4

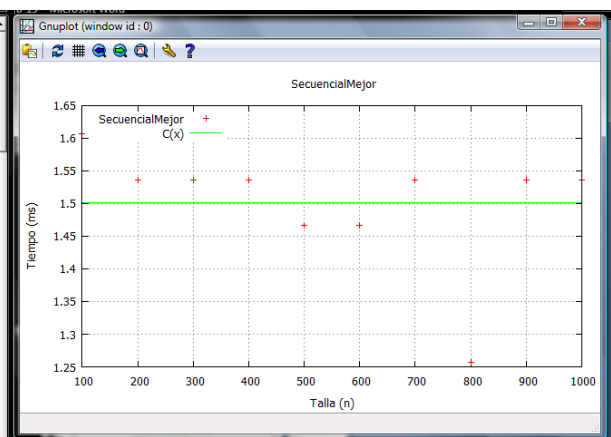
Datos guardados en el fichero SecuencialMedio.dat
Generar grafica de resultados? (s/n): s

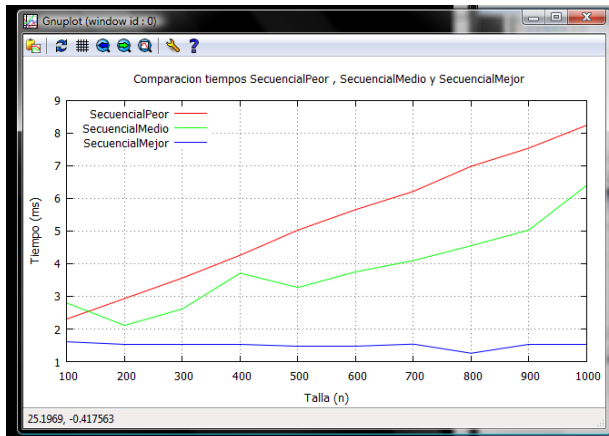


Busqueda SecuencialMejor. Tiempos de ejecucion promedio

Talla	Tiempo (nseg)
100	1.6
200	1.5
300	1.5
400	1.5
500	1.5
600	1.5
700	1.5
800	1.3
900	1.5
1000	1.5

Datos guardados en el fichero SecuencialMejor.dat
Generar grafica de resultados? (s/n): s





5. Descripción del problema

Este estudio de costes se va a realizar sobre *vectores* de enteros generados aleatoriamente.

5.1. Definición de la clase base `ConjuntoInt`.

Definición de la clase `ConjuntoInt` para que se pueda especificar (y pueda variar) el tamaño máximo del vector.

La declaración e implementación está en la carpeta `Ficheros_Practica_1_FAA_19_20`

5.2. La clase `TestAlgoritmo`.

Definición de la clase `TestAlgoritmo` que contiene los métodos para el análisis del algoritmo para la búsqueda secuencial:

- Comprueba que el algoritmo funciona adecuadamente.
- Calcula la eficiencia teórica y experimental para los casos del algoritmo de búsqueda secuencial, permitiendo guardar los datos e imprimir la gráfica correspondiente.
- Comparar el coste temporal de los casos del método de búsqueda secuencial teórico y experimental, permitiendo guardar los datos e imprimir la gráfica correspondiente.

La declaración e implementación está en la carpeta `Ficheros_Practica_1_FAA_19_20`

5.3. La clase `Mtime`.

Definición de la clase `Mtime` que contiene la función para medir el tiempo.

La declaración e implementación está en la carpeta `Ficheros_Practica_1_FAA_19_20`

5.4. Las Gráficas.

Las **Gráficas** las podemos hacer a partir de los ficheros de datos con Excel o con los ficheros por lotes (comandos) de gnuplot que están en la subcarpeta `datos` en `Ficheros_Practica_1_FAA_19_20` para generar los ficheros gráficos correspondientes a:

- Caso medio de un algoritmo con ajuste a su Orden de complejidad.
- Comparación de los Casos de un algoritmo.

6. Actividades a desarrollar.

Para el desarrollo de esta práctica, Análisis experimental del Algoritmo de Búsqueda_Secuencial, se detalla y se proponen para su realización las siguientes actividades:

6.1. Programa principal.

Completar el programa principal incorporando los menús como se especifica a continuación.

El método **main** proporciona las opciones del menú para verificar que el algoritmo implementado funciona correctamente, calcular su coste teórico y empírico en el caso peor, medio y mejor, así como comparar los casos.

El programa principal está compuesto por una cabecera con el **nombre del alumno/a** (donde pone Prof. poned Alumno/a) y un menú cuya estructura y descripción es la siguiente:

El programa principal está compuesto por un menú cuya estructura y descripción es la siguiente:

```
*** FAA. Practica 1. Curso 19/20 ***  
  
Prof. Teresa Santos  
  
*** ESTUDIO DE LA COMPLEJIDAD DEL ALGORITMO BUSQUEDA SECUENCIAL ***  
  
1.- ESTUDIO TEORICO  
2.- ESTUDIO EMPIRICO  
0.- Salir  
-----  
Elige opcion: _
```

- **El Menú** consta de 3 opciones incluida la opción de salir. El programa no finalizará hasta que la opción salir sea seleccionada.
 - **Opción 1:** Esta opción abrirá un submenú para realizar el estudio teórico del algoritmo de búsqueda secuencial implementado en la clase TestAlgoritmo.
 - **Opción 2:** Esta opción abrirá un submenú para realizar el estudio empírico del algoritmo de búsqueda secuencial que deberá implementarse en la clase TestAlgoritmo.
 - **Opción 0:** Esta opción hace que el programa termine.

1. Submenú de la Opción 1: **ESTUDIO TEÓRICO**.

El Menú Teórico consta de 4 opciones incluida la opción de salir. El programa no finalizará hasta que la opción salir sea seleccionada.

- **Opción 1:** Esta opción **comprueba** el funcionamiento del algoritmo de búsqueda secuencial (implementado en la clase TestAlgoritmo).
- **Opción 2:** Esta opción abrirá un submenú para efectuar el cálculo de los tiempos en el **caso peor, medio y mejor** para el algoritmo de búsqueda secuencial y realizar la **gráfica** con su función de orden O.
- **Opción 3:** Esta opción realiza la **gráfica** comparativa de todos los casos.
- **Opción 0:** Esta opción hace que el programa vuelva al menú principal.

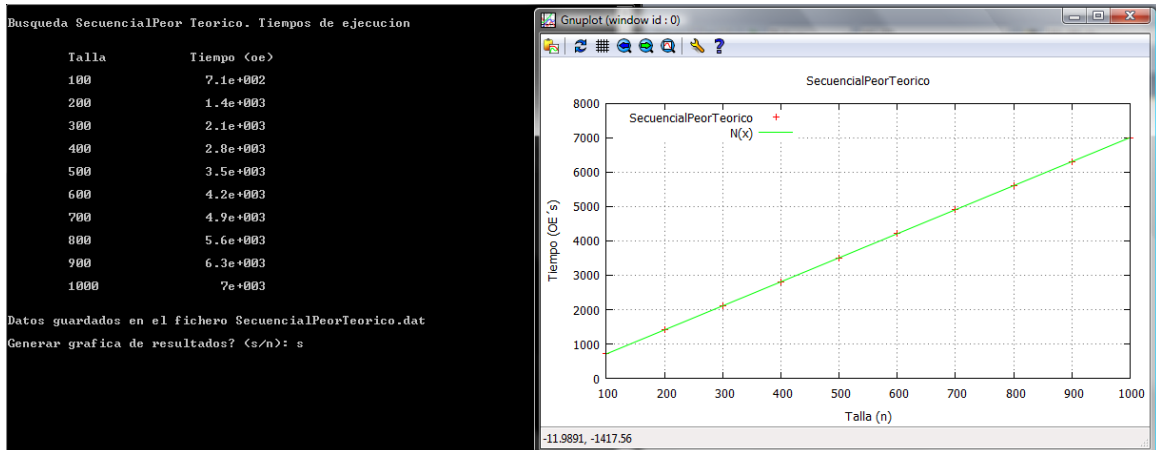
<pre>*** MENU TEORICO DEL ALGORITMO DE BUSQUEDA SECUENCIAL*** 1.- Probar el algoritmo busqueda secuencial 2.- Obtener los casos del metodo de busqueda secuencial 3.- Comparar los casos 0.- Volver al menu principal ----- Elige opcion: 1_</pre>	<pre>Introduce la talla: 5 vector para el algoritmo : 331, 841, 615, 195, 560 Introduce la clave a buscar: 331 posicion de 331 buscado con el algoritmo de busqueda secuencial : 0 Presione una tecla para continuar . . .</pre>
--	---

1.2. El submenú correspondiente de la **Opción 2** es el siguiente:

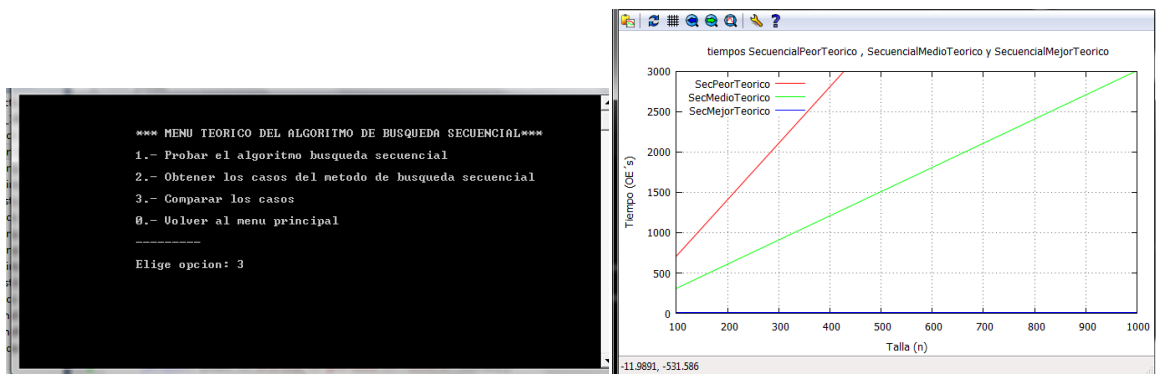
```
*** MENU TEORICO DEL ALGORITMO DE BUSQUEDA SECUENCIAL***
1.- Probar el algoritmo busqueda secuencial
2.- Obtener los casos del metodo de busqueda secuencial
3.- Comparar los casos
0.- Volver al menu principal
-----
Elige opcion: 2

*** Caso a estudiar para la busqueda secuencial ***
0: Caso peor
1: Caso medio
2: caso mejor
-----
Elige opcion: 0
```

➤ Como resultado de la opción 0 (Caso peor), por ejemplo, se mostrará la información siguiente:



1.3. Como resultado de la **opción 3** se realiza la **gráfica** comparativa de todos los casos.



2. Submenú de la **Opción 2: ESTUDIO EMPÍRICO**. Para el estudio experimental los menús son iguales que para el teórico.

```
*** FAA, Práctica 1. Curso 19/20 ***  
  
Prof. Teresa Santos  
  
*** ESTUDIO DE LA COMPLEJIDAD DEL ALGORITMO BUSQUEDA SECUENCIAL ***  
  
1.- ESTUDIO TEORICO  
2.- ESTUDIO EMPIRICO  
0.- Salir  
-----  
Elige opcion: 2
```

```
*** MENU EMPIRICO DEL ALGORITMO DE BUSQUEDA SECUENCIAL***  
1.- Probar el algoritmo busqueda secuencial  
2.- Obtener los casos del metodo de busqueda secuencial  
3.- Comparar los casos  
0.- Volver al menu principal  
-----  
Elige opcion: 2_
```

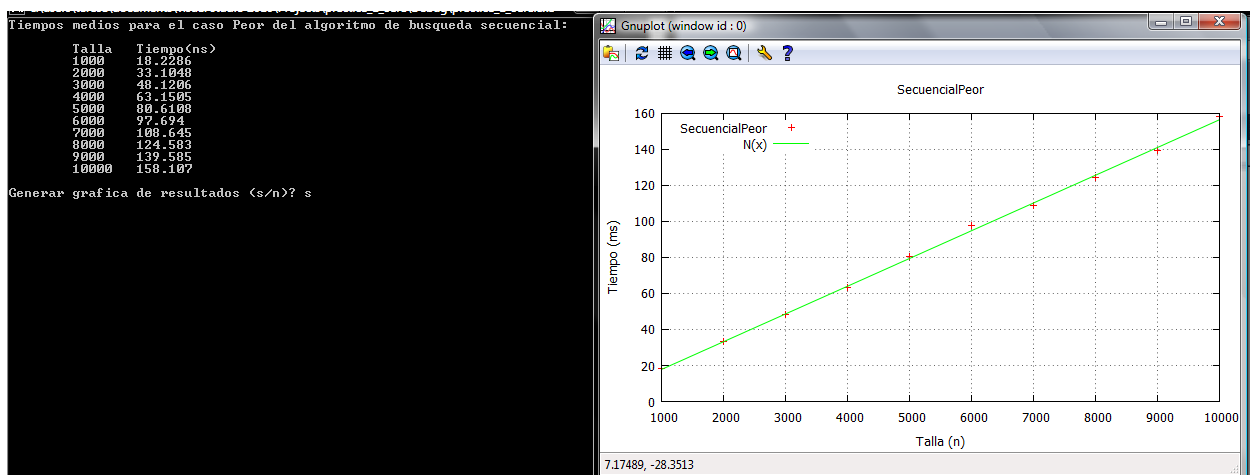
- 2.1. Como resultado de la opción 1 dará:

```
Introduce la talla: 10  
  
vector para el algoritmo :  
903, 749, 239, 258, 185, 98, 468, 53, 933, 95  
  
Introduce la clave a buscar: 95  
  
posicion de 95 buscado con el algoritmo de busqueda secuencial : 9  
  
tiempo de ejecucion= 0.195556 ms  
Presione una tecla para continuar . . .
```

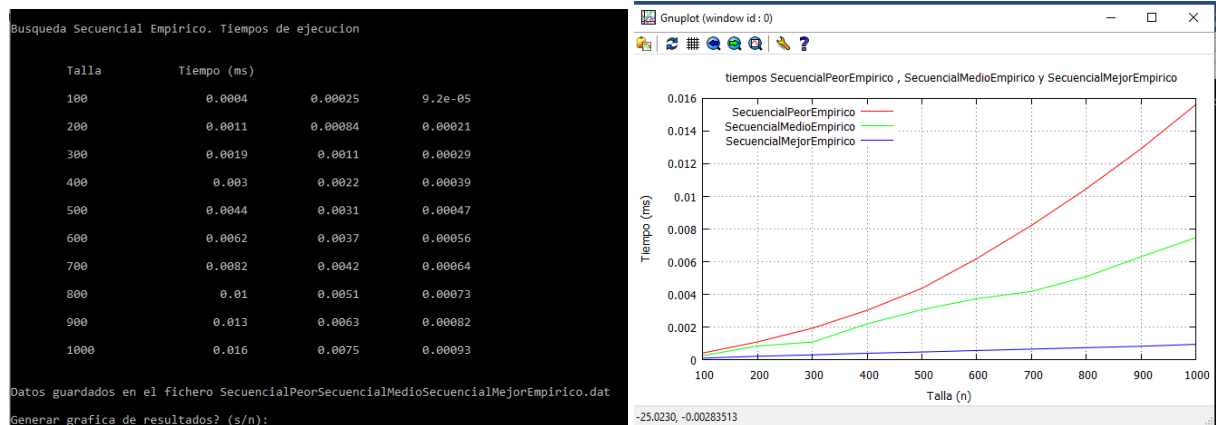
- 2.2. El submenú correspondiente de la **Opción 2** es el siguiente:

```
*** Caso a estudiar para la busqueda secuencial ***  
0: Caso peor  
1: Caso medio  
2: caso mejor  
-----  
Elige opcion: _
```

- Como resultado de la opción 0 (Caso peor), por ejemplo, se mostrará la información siguiente:



2.3. Como resultado de la **opción 3** se realiza la **gráfica** comparativa de todos los casos.



6.2. Conclusiones.

Para el algoritmo habrá que realizar una gráfica para cada una de las tres curvas obtenidas y del orden de complejidad y comentarlas tal y como se mostró anteriormente. ¿Son coherentes con el resultado teórico obtenido? Justificar la respuesta.

Para mostrar los datos en gráfica pondremos en el eje X (abscisa) el tamaño de los casos y en el eje Y (ordenada) el tiempo, medido en milisegundos(o microsegundos), requerido por la implementación del algoritmo.

Esto formará parte de la memoria de la práctica.

7. Entrega de la práctica

La fecha de entrega es el domingo **5 de abril**.

La entrega de la práctica se realizará por Moodle en la tarea puesta al efecto y tendrá el siguiente formato:

Crear y subir la carpeta **Apellido1Apellido2.rar** la cual debe contener:

Una subcarpeta con los fuentes del programa (.cpp y .h)

Una subcarpeta con el ejecutable y los datos.

La memoria, de **7 páginas como máximo**, en formato pdf. con el esquema especificado a continuación.

7.1. Esquema para la memoria.

1. Portada (todos los autores)
2. Índice
3. Introducción. Algoritmo Búsqueda secuencial.
4. Cálculo del tiempo teórico:
 - 4.1. Pseudocódigo (código) y análisis de coste
 - 4.2. Tablas y Gráficas de coste
 - 4.3. Conclusiones
5. Cálculo del tiempo experimental:
 - 5.1. Tablas y Gráficas de coste

5.2. Conclusiones

6. Comparación de los resultados teórico y experimental.
7. Diseño de la aplicación.

(Mostrar un esquema gráfico global de la estructura de tipos de datos existentes. Detallar la descomposición modular del programa, qué módulos existen, cuál es la responsabilidad de cada uno y la relación de uso. Documentar cualquier otra decisión de diseño que pueda resultar de interés. Funcionamiento y explicación de los métodos implementados en la práctica.)

8. Conclusiones y valoraciones personales de la práctica.

8. Consideraciones para la implementación:

La clase ConjuntoInt permite crear vectores de tamaño variable, para cada uno es tamaño fijo pues no hacemos inserciones ya que generamos el vector de forma aleatoria para ese tamaño.

Para generar los vectores con los diferentes tamaños podéis utilizar el siguiente esquema:

```
for (int i= 100; i<1000; i+=100) // para vectores de 100 a 900 elementos
{
    ConjuntoInt *c= new ConjuntoInt(i); /* crea un nuevo objeto usando el constructor
    por defecto. i= numero de elementos del vector, Tamaño.*/
    ...
    ...
    delete c; //Borra el objeto dinámico, usando el destructor.
};
```

Para la invocación de un método sobre el objeto dinámico se utiliza el operador flecha '->':

a->b equivale a (*a).b

Ejemplo:

```
c->generaVector (100);
c->escribe();
cout << c->busquedaSecuencial(88);
```

9. Anexo. Resumen de los pasos a seguir para realizar la práctica 1.

- En la carpeta FicherosPractica1.rar **subcarpeta fuentes** tenéis las siguientes clases que tendréis que incorporar a vuestro proyecto **ApellidoPractica1**:
 - La clase ConjuntoInt, ficheros ConjuntoInt.h y ConjuntoInt.cpp
 - La clase TestAlgoritmo, ficheros TestAlgoritmo.h y TestAlgoritmo.cpp
 - La clase **Mtime**, ficheros **Mtime.h** y **Mtime.cpp**. Fichero donde está la función de medir el tiempo.
 - Fichero **Constantes.h**, definiciones de las constantes utilizadas.
 - El main es el fichero Principal.cpp
 - En la carpeta FicherosPractica1.rar **subcarpeta datos** tenéis los ficheros de comandos por si queréis generar las gráficas en gnuplot .
1. Completar el programa principal incorporando los menús como se ha especificado anteriormente.
 2. Analizar y ejecutar los métodos dados `costeCasoTeorico` y `compararTeorico` de la clase `TestAlgoritmo`.
 3. Completar los métodos `costeCasoEmpirico` y `compararEmpirico` de la clase `TestAlgoritmo`.
 4. Una vez completado el código, crear en el directorio **Apellido1Apellido2** tres carpetas:
 - **fuentes**: con los fuentes .h y .cpp
 - **datos**: con el ejecutable y los datos
 - **memoria**: con el contenido de la memoria especificada.
 5. Subir **Apellido1Apellido1.rar**