Práctica 3: Medición experimental de tiempos de ejecución en programas

Objetivos:

- Conocer en la práctica la diferencia en tiempo de ejecución entre dos operaciones sobre matrices cuadradas que tienen distinta complejidad temporal (suma y multiplicación).

Programa a realizar:

Primeramente, deberéis crear una librería matriz (matriz.h y matriz.c) que, al menos, de soporte a las operaciones de creación de una matriz dinámica (a partir de su tamaño), destrucción, inicialización de una matriz con valores aleatorios (consultad uso de srand), acceso a un elemento, y suma y multiplicación de matrices. Estas dos últimas operaciones entre matrices han de implementarse de manera directa (es decir, sin utilizar ningún tipo de estrategia divide vencerás sino que se deben implementar de modo directo utilizando técnicas iterativas). Supondremos tipo long para los elementos de la matriz. Dado que pretendemos tratar matrices del mayor tamaño posible (para llevar cada algoritmo a "su situación de estrés"), debéis aseguraos de que la estructura dinámica que almacena los elementos de la matriz puede direccionar suficiente número de elementos (en particular, no utilicéis internamente, por ejemplo, un tipo short o int para referiros al tamaño o al índice sobre el que actuamos en la matriz, sino un tipo long).

Seguidamente, debéis desarrollar un programa que genere iterativamente pares de matrices de distinto tamaño y realice dos tipos de operaciones sobre los mismos:

- Suma de matrices
- Multiplicación de matrices

El programa deberá medir el tiempo necesario para realizar cada tipo de operación y guardar los resultados en un fichero. Usaremos dicho fichero para realizar con MATLAB un análisis básico de los resultados obtenidos, mediante su representación gráfica.

Medición de tiempos de ejecución:

Para medir el tiempo de ejecución de un programa disponemos en C de la biblioteca <**time.h**>. Dicha biblioteca dispone de funciones/procedimientos, constantes, tipos de datos y registros que permiten evaluar el número de períodos o "tics" de reloj que consume una secuencia de instrucciones. La función **clock()** permite acceder a dicha información. Su prototipo es el siguiente:

clock_t clock (void)

Práctica 3: Medición experimental de tiempos de ejecución en programas

clock() devuelve el número de "tics" de reloj que han transcurrido desde un determinado evento de referencia (la creación del proceso desde donde se invoca). Mediante **clock()** podemos medir el tiempo transcurrido en una secuencia de instrucciones sin más que invocar dicha función al inicio y al final de dicha secuencia, guardar ambos resultados y restarlos.
El tipo de dato que devuelve **clock()** es el tipo opaco **clock_t**, que en nuestro caso equivale a **long int** (aunque en otros sistemas puede ser **double**).
Para convertir el número de periodos de reloj a segundos de forma fácil y general puede utilizarse la constante

int CLOCKS_PER_SEC;

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char** argv) {
clock_t inicio=-1, fin=-1;

inicio=clock(); //TOMAMOS LA PRIMERA REFERENCIA
/* AQUÍ SE INICIA EL BLOQUE DE INSTRUCCIONES CUYO TIEMPO DE EJECUCIÓN QUEREMOS MEDIR */
        AQUÍ VA EL CÓDIGO QUE QUEREMOS
                "CRONOMETRAR"
/* AQUÍ FINALIZA EL BLOQUE DE INSTRUCCIONES CUYO TIEMPO DE EJECUCIÓN QUEREMOS MEDIR */
fin=clock(); //TOMAMOS LA SEGUNDA REFERENCIA

//IMPRIMIMOS EL RESULTADO
printf("%u\t%lf\n", fin-inicio, (fin-inicio)/(double)CLOCKS_PER_SEC);

return (EXIT_SUCCESS);
}
```

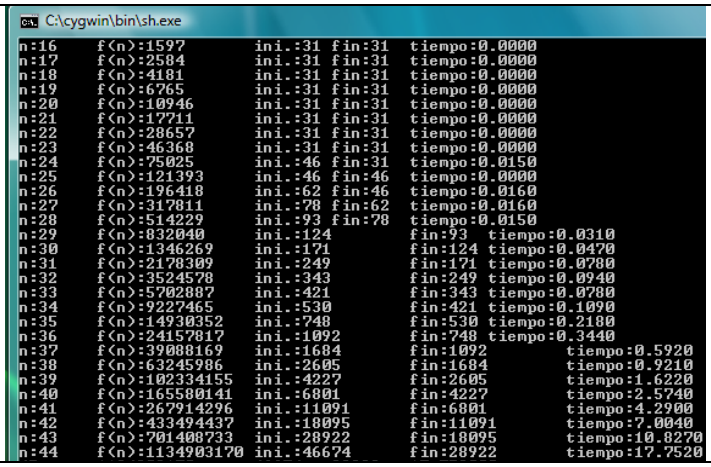
Ejemplo: medida experimental de tiempos en la implementación recursiva de la sucesión de Fibonacci.

En este ejemplo debéis ejecutar el proyecto **P3_FibonacciRecursivo**, que incluye la implementación recursiva de la serie de Fibonacci. Dicha serie se define como:

$$f(n) = \begin{cases} 1 & n = 0,1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

Práctica 3: Medición experimental de tiempos de ejecución en programas

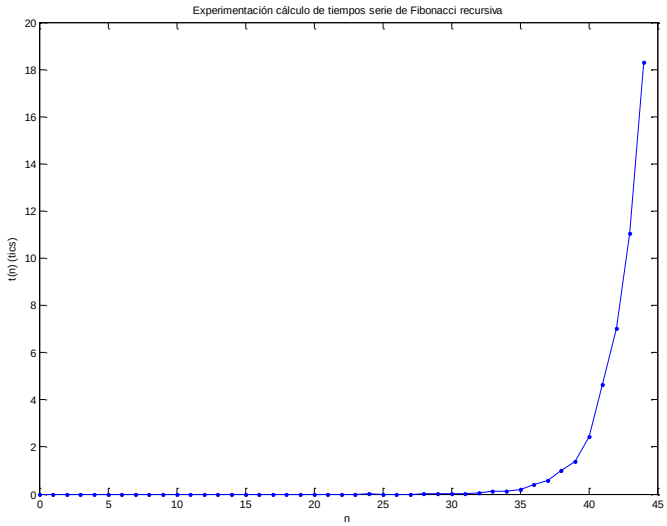
Podéis verificar experimentalmente como progresivamente la obtención de los resultados se ralentiza (debido a que el número de llamadas recursivas, y por lo tanto el tiempo de ejecución, crece exponencialmente)[1]. Fijaos, en particular en el tiempo de obtención de los últimos términos de la sucesión (n=43, 44).



El programa genera un fichero de nombre "**tiemposFibonacciRecursivo.txt**" con dos columnas de datos que son respectivamente **n** y **t(n)**. Podéis visualizarlos gráficamente desde Matlab ejecutando el programa **repreTiemposFibo.m**.

[1] **Como curiosidad, comentar que el tiempo de obtención del término n-ésimo t(n) tiende a verificar también la sucesión de Fibonacci: t(n)→ t(n-1)+t(n-2) cuando n→∞.**

Práctica 3: Medición experimental de tiempos de ejecución en programas



Para valorar la importancia de basar la implementación de los programas en algoritmos lo más eficientes posible, podéis experimentar ahora sobre el programa anterior, cambiando la implementación recursiva por esta otra iterativa:

```
unsigned long fibonacciIterativo(unsigned char n){

    unsigned long a=1, b=1, c=-1, k=-1;

    for(k=2; k<=n; k++){
        c=a+b;
        a=b;
        b=c;
    }
    return b;
}
```

En particular, comparad el tiempo invertido en obtener los últimos términos de la sucesión entre las dos implementaciones propuestas: esta nueva implementación iterativa y la anterior implementación recursiva. ¿Cuál es la diferencia para los términos n>40?

Práctica 3: Medición experimental de tiempos de ejecución en programas

Ejercicio 3.1: medida experimental de tiempos de suma de matrices.

En el ejercicio vamos a implementar un programa que nos permita medir los tiempos de ejecución para suma de matrices de distintos tamaños (n). Utilizaremos un nuevo TAD matriz que crearéis para esta actividad. Para cada valor de **n**, el programa medirá el tiempo de ejecución **t(n)** consumido en realizar la suma (supondremos tipo long para los elementos del vector).

El módulo principal se encargará de realizar la experimentación. Para ello debéis implementar las siguientes tareas:

1. generar pares de matrices de tamaño **n*n**. Podéis tomar valores de **n**=10.000, ..., 100.000 con paso 10.000 (u otros intervalos y pasos diferentes, en función del tiempo que tarde en vuestro computador las operaciones).
2. realizar la ordenación de suma de matrices
3. medir el tiempo **t(n)** que tarda en realizarse la ordenación.
4. guardar en un fichero dos columnas de datos, respectivamente **n** y **t(n)** (formato idéntico al del ejemplo)

El programa recibirá como argumentos en la línea de entrada los valores de tamaño inicial, tamaño final y paso necesarios para realizar la experimentación. En el ejemplo anterior recibiría 10000 100000 10000 pero el programa debe poder aceptar cualquier entrada de estos tres valores para poder realizar experimentos con tamaños iniciales, finales y pasos distintos.

Una vez ejecutado el programa, podéis visualizar una gráfica con los resultados **t(n)** frente a **n** utilizando el programa Matlab **repreTiemposSumaMatrices.m.** Ejecutad el programa para distintos rangos de tamaños de vectores y analizad con cuidado las tendencias obtenidas. El programa Matlab es únicamente para mostrar la gráfica con los resultados que saca a fichero vuestro programa C.

Ejercicio 3.2: medida experimental de tiempos de multiplicación de matrices.

El ejercicio consiste en realizar la misma experimentación que en el ejercicio anterior, para obtener los tiempos de ejecución de la multiplicación de matrices.

Una vez ejecutado el programa, podéis visualizar una gráfica con los resultados **t(n)** frente a **n** utilizando el programa Matlab **repreTiemposMultiplicacionMatrices.m**. Nuevamente, el programa Matlab es únicamente para mostrar la gráfica con los resultados que saca a fichero vuestro programa C.

Práctica 3: Medición experimental de tiempos de ejecución en programas

Entregables:

Se deberá realizar la entrega en el Campus Virtual. Las fechas de entrega se especifican en el Campus Virtual, y las instrucciones son las siguientes:

- En esta práctica hay que incluir en el fichero comprimido **tanto el código del programa realizado como un informe explicativo** de la experimentación realizada:
  - Al igual que en prácticas anteriores, **el código a entregar estará únicamente compuesto por los archivos fuentes** (main.c, matriz.h, matriz.c, ...) **y un Makefile** que compile perfectamente en Linux. Cualquier ejercicio que no compile directamente con el makefile (sin opciones) en Linux será evaluado con la calificación de 0. Este criterio se mantendrá en todas prácticas de la asignatura.
  - El informe **debe ser entregado en formato PDF**. La entrega del documento en otros formatos tendrá automáticamente la calificación de 0. El informe será un breve documento (2 o 3 páginas) introduciendo el problema de experimentación, comentando los resultados, mostrando las gráficas y contestando a preguntas como:
    - ¿qué tamaños de matrices son gestionables en tiempo "razonable" por cada algoritmo?
    - ¿Es más eficiente en general alguna de las dos operaciones? En caso afirmativo, indica si es así para todos los valores de **n** o únicamente para algunos. En este último caso, indica el rango de valores de n para los que es más ágil cada operación.
  - El fichero comprimido con el código y el PDF tendrá el nombre apellido1_apellido2 y la extensión .ZIP (**usad siempre .ZIP como formato de compresión**).

Algunos errores típicos detectados en años anteriores y que debéis evitar a la hora de redactar el informe (**serán tenidos en cuenta en la revisión del informe**):

-gráficas caóticas por no probar rangos de tamaños suficientemente grande. No se aprecia crecimiento esperado en los tiempos de ejecución.
-errores edición informe (p.e. falta de justificación derecha de los párrafos, falta de nombre de autor en el informe, errores ortográficos, falta de título en el documento, falta de introducción, edición no profesional –tipos de fuente heterogéneos, etc-).

-alguna de las preguntas planteadas en el guión no es contestada en el informe o de hacerlo se hace de manera genérica sin proporcionar datos experimentales.
-errores argumentativos respecto a conceptos básicos de complejidad computacional (por ejemplo, argumentar en el informe que superlineal=lineal, o que cuadrático=superlineal, o que cuadrático=exponencial). Es decir, tener confusión respecto a las clases de complejidad.
-la cronometración no cuenta sólo el coste de la operación sino también alguna operación adicional (creación, relleno de aleatorios o liberación de la matriz, por ejemplo).