# ANÁLISIS Y DISEÑO DE ALGORITMOS

## Divide y vencerás

### Práctica 5 de laboratorio

**Entrega: Hasta el domingo 8 de marzo, 23:55h. A través de Moodle**

La técnica de diseño de algoritmos *Divide y vencerás* consiste en resolver un problema reduciéndolo sucesivamente a subproblemas más sencillos hasta que la solución de estos resulte trivial. La complejidad temporal del algoritmo resultante depende, entre otros factores, del número de subproblemas que se generan.[1] El objetivo de esta práctica es comprobar empíricamente con un problema concreto cómo influye en la eficiencia del algoritmo resultante la forma en la que el problema inicial se divide en subproblemas.

El trabajo a realizar en esta práctica es el siguiente:

1. Implementa tres funciones distintas que obtengan el valor de la potencia enésima de 2 ($2^n$ donde $n \in \mathbb{N}$) de manera que cada una de ellas presente una complejidad temporal asintótica distinta a las demás. Incluye las tres funciones en un único fichero fuente con nombre **`pow2.cc`**. Los prototipos de las funciones han de ser los siguientes:

   - `unsigned long pow2_1(unsigned)`
   - `unsigned long pow2_2(unsigned)`
   - `unsigned long pow2_3(unsigned)`

   Completa el fichero fuente con la función `main()` para realizar un análisis empírico de la complejidad temporal de las tres funciones implementadas (véase el siguiente apartado).

2. De manera similar al trabajo realizado en las prácticas 1 y 2, realiza un análisis empírico de la complejidad temporal de las tres funciones implementadas. Puede hacerse cronometrando tiempo de ejecución, contando pasos de programa o de cualquier otra forma que se considere apropiada. El programa resultante debe llamarse `pow2` y su ejecución debe mostrar por pantalla una tabla con los resultados de la medición para cada valor de $n$ y cada algoritmo.

3. A partir de la mencionada tabla y mediante la herramienta *Gnuplot* obtén una única gráfica que contenga las tres funciones de coste obtenidas. La gráfica debe llamarse *pow2.png* y se creará al ejecutar la orden *make* sin necesidad de añadir nada más. El archivo de órdenes de gnuplot debe llamarse *pow2.gnuplot*.

**Normas para la entrega.**

**ATENCIÓN: Estas normas son de obligado cumplimiento para que esta práctica sea evaluada.**

1. Se debe entregar los ficheros `pow2.cc`, `pow2.gnuplot` y `makefile`. **No hay que entregar nada más.**

2. Es imprescindible que no presente errores ni de compilación ni de interpretación (según corresponda), en los ordenadores del laboratorio asignado y en el sistema operativo *Linux*.[2] Se tratará de evitar también cualquier tipo de *warning*. A partir de la orden *make* (sin añadir nada más) debe crearse la imagen *pow2.png*.

3. Todos los ficheros que se entregan deben contener el nombre del autor y su DNI (o NIE) en su primera línea (entre comentarios apropiados según el tipo de archivo).

4. Se comprimirán en un archivo *.tar.gz* cuyo nombre será el DNI del alumno, compuesto de 8 dígitos y una letra (o NIE, compuesto de una letra seguida de 7 dígitos y otra letra). Por ejemplo: 12345678A.tar.gz o X1234567A.tar.gz. **Solo se admite este formato de compresión y solo es válida esta forma de nombrar el archivo.**

5. En el archivo comprimido **no debe existir subcarpetas**, es decir, al extraer sus archivos estos deben quedar guardados en la misma carpeta donde está el archivo que los contiene.

6. La práctica hay que subirla a *Moodle* respetando las fechas expuestas en el encabezado de este enunciado.

[1] El coste de obtener los subproblemas y el de combinar sus soluciones parciales también pueden afectar a la complejidad del algoritmo resultante.

[2] Si trabajas con tu propio ordenador o con otro sistema operativo asegúrate de que este requisito se cumple.