SEMANA DE LA CIENCIA 2019

Ficha 2. Algoritmos Recursivos

El objetivo de esta ficha es mostrar la programación de **algoritmos recursivos**.

Una de las funciones recursivas más famosas es **factorial**:

$$fact(n) = \begin{cases} 1 & si\ n = 0 \\ n * fact(n-1) & si\ n \geq 1 \end{cases} \qquad \begin{cases} 0! = 1 \\ n! = n * (n-1) * (n-2) * \ldots * 2 * 1 \end{cases}$$

Factorial es útil en la composición de *permutaciones*, por ejemplo. El cálculo es sencillo y su veremos que el código en un lenguaje de programación es casi cortar y pegar la fórmula matemática.

Otra función recursiva un poquito más difícil es la que hemos mostrado en la presentación: la sucesión de **Fibonacci**.

$$fib(n) = \begin{cases} 1 & si\ n = 1\ o\ 2 \\ fib(n-1) + fib(n-2) & si\ n \geq 3 \end{cases} \qquad \textit{Ecuación original}$$

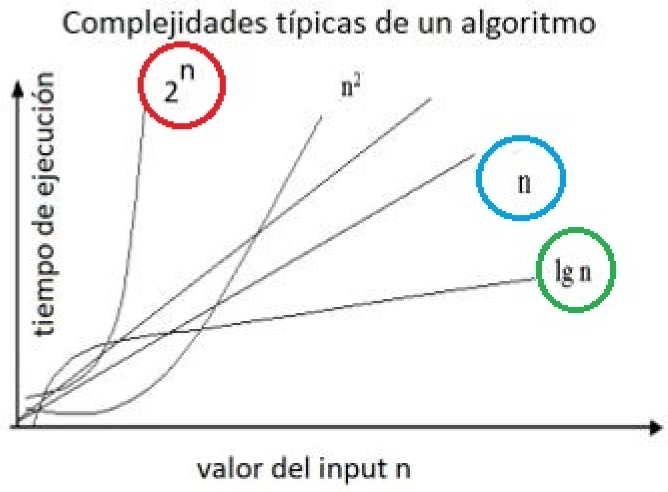
Sucesion de Fibonacci: **1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …**

Una primera implementación puede ser precisamente la *fórmula matemática con recursión doble*, pero veremos que es **muy poco eficiente**, pues ni siguiera un buen ordenador será capaz de calcular el Fibonacci de 50, por ejemplo.

Con ayuda de las matemáticas y de la imaginación vamos a implementar otra solución que calcula cualquier Fibonacci en **tiempo record**, lo que llamamos *'tiempo real'*.

$$fib\_final(n, f1, f2) = \begin{cases} f2 & si\ n = 1 \\ fib\_final(n-1, f2, f1+f2) & si\ n \geq 2 \end{cases} \qquad \textit{Ecuación generalizada}$$

Para calcular números de Fibonacci con esta función hacemos: **fib_final(n,0,1)**



En la práctica implementaremos varios algoritmos con diferentes complejidades como se marca en esta gráfica:

- Rojo: El tiempo de ejecución es tan alto que no podríamos vivir para conocer el resultado.
- Azul: El tiempo de ejecución es proporcional al número que actúa como input.
- Verde: El tiempo de ejecución será el logaritmo del número que actúa como input.