# Práctica 1.2: Complejidad teórica: algoritmos iterativos

Algoritmia y optimización

Curso 2024–25

## Índice

# 1. Introducción

En esta Práctica 1.2 continuamos analizando la complejidad de los algoritmos. En esta ocasión, además de seguir practicando con el análisis empírico, incluiremos el **análisis teórico** de algoritmos iterativos.

# 2. Objetivos

- Comprender la complejidad teórica en algoritmos iterativos.
- Comparar la complejidad teórica con la complejidad empírica.

# 3. Ejercicio 1

Dado que en esta práctica compararemos las complejidades teóricas con respecto a las complejidades empíricas, es importante tener claro cómo visualizar una complejidad teórica. Representa las siguientes funciones con Matplotlib:

- $\sqrt{n}$
- $10^n$
- $n^{1,5}$
- $2\sqrt{\log_2 n}$
- $n^2 \log_2 n$
- $2^n$
- $n^{\log_2 n}$
- $n^2$
- $2^{\log_2 n}$
- $2^{2^{\log_2 n}}$
- $n^{\frac{5}{2}}$
- $n^2 \log_2 n$

## 4. Ejercicio 2

En la Práctica 1.1 se presentaron dos algoritmos de ordenación. Ahora, calcula la complejidad teórica del algoritmo *ordenacion_uno()* y compara el resultado obtenido con la complejidad empírica de la práctica anterior.[1]

```
función ordenacion_uno(arr):
    n := |arr|
    para i desde 0 hasta n - 1:
        para j desde 0 hasta n - i - 1:
            si arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    devuelve arr
```

## 5. Ejercicio 3

Calcula la complejidad teórica y empírica del algoritmo *producto_matrices_cuadradas()*. Muestra ambas complejidades en una misma gráfica.

```
función producto_matrices_cuadradas(A, B):
    n := dimensión(A)
    C := ceros(n, n)
    para i desde 0 hasta n - 1:
        para j desde 0 hasta n - 1:
            suma := 0
            para k desde 0 hasta n - 1:
                suma := suma + A[i][k] * B[k][j]
            C[i][j] := suma
    devuelve C
```

## 6. Ejercicio 4

El siguiente pseudocódigo representa una versión iterativa del algoritmo denominado *búsqueda_binaria()*. Nota que este algoritmo presenta mejor y peor caso. Por tanto, averigua sus complejidades teóricas y después calcula su complejidad empírica.

[1] Recuerda que en la asignatura estamos utilizando la notación "Big O", que indica el orden de magnitud de la complejidad pero no su función exacta. Es por ello que, al dibujar en un mismo gráfico la complejidad empírica y la teórica, tendrás que aplicar algún factor para mantener la proporción de ambas curvas.

```
función busqueda_binaria(arr, n, target):
    L := 0
    R := n - 1
    mientras L <= R:
        m := redondeo((L + R) / 2)
        si arr[m] < target entonces:
            L := m + 1
        si_no si arr[m] > target entonces:
            R := m - 1
        si_no:
            devuelve m
    devuelve NO_ENCONTRADO
```

### 6.1. Cómo medir el algoritmo *búsqueda_binaria*

Como has podido comprobar calculando la complejidad teórica de la *búsqueda_binaria()*, encontramos mejor y peor caso. Para una correcta visualización del comportamiento empírico realiza las siguientes pruebas:

- Ejecuta con listas aleatorias y busca valores al azar.
- Ejecuta con listas aleatorias y busca un número que no se encuentre.
- Ejecuta con listas aleatorias y busca el número que está en la primera posición.

Una vez implementados estos experimentos, muestra el rango de los tiempos obtenidos en la gráfica como un área sombreada junto a las líneas que representan las complejidades teóricas del mejor y peor caso. Recuerda que, para cada caso, debes medir la misma prueba varias veces y almacenar el promedio para obtener una medida más precisa de la complejidad. Recuerda también obtener mediciones a tallas crecientes de la lista.