# Introducción a los Algoritmos

MSc Edson Ticona Zegarra

Taller de Programación 2025

# Contenido

# Contenido

# Definiciones

- Un algoritmo se define como un procedimiento, definido por una serie de instrucciones o pasos, que recibe un conjunto de valores de entrada y retorna un conjunto de valores de salida.

# ¿Cómo se mide la eficiencia de un algoritmo?

- Dado un problema, y dos algoritmos que resuelven dicho problema, intuitivamente podemos decir que el algoritmo que resuelve el problema más rápido es el mejor de los dos, o el más *eficiente*.
- En general se puede decir que aquel algoritmo que resuelve el problema en la *menor* cantidad de pasos es el más eficiente.
- Cuando hablamos del *análisis* de un algoritmo, nos referimos a estimar los recuros que requiere el algoritmo, en términos de memoria y tiempo.

# Ejemplo

- La cantidad de pasos que un algoritmo requiere para terminar su ejecución depende, por lo general, de la cantidad de datos de entrada.
- Ejemplo: diseñar un algoritmo que, dado un conjunto de números, encuentre el menor de todos ellos.

# Ejemplo

**input** : $A$ es un conjunto de $n$ números.
**output:** $min$ es el menor elemento de $A$.

$min \leftarrow A[0]$;
**for** $a \in A$ **do**
  **if** $a < min$ **then**
    $min \leftarrow a$
  **end**
**end**
**return** $min$

- Denotemos como $T(n)$ la cantidad de pasos necesarios para la ejecución total del algoritmo.
- Donde $n$ representa el número de elementos de entrada.

# Ejemplo

**input :** $A$ es un conjunto de $n$ números.
**output:** $min$ es el menor elemento de $A$.

$min \leftarrow A[0]$ ; /* $c_1$ */
**for** $a \in A$ **do** ; /* $n$ veces */

  **if** $a < min$ **then** ; /* $c_2$ */

    $min \leftarrow a$ ; /* $c_3$ */
  **end**
**end**
**return** $min$

Sumando, queda $T(n) = c_1 + n * (c_2 + c_3)$

# Contenido

# Notación *Big-O*

- Para facilitar el análisis introducimos la notación asintótica

**Definition**

Decimos que $T(n) = O(f(n))$ si $T(n) \leq f(n)$ para todo $n \geq n_0$

- Es decir, la notación *Big-O* marca una cota superior.
- Entonces, en el ejemplo previo, $T(n) = O(n)$

# Notación *Big-Omega*

- Analogamente, podemos definir limites inferiores

## Definition

Decimos que $T(n) = \Omega(f(n))$ si $T(n) \geq f(n)$ para todo $n \geq n_0$

- Es decir, la notación *Big-Omega* marca una cota inferior.
- Entonces, en el ejemplo previo, $T(n) = \Omega(n)$

# Notación *Big-Theta*

- Finalmente, se puede usar ambos límites

**Definition**

Decimos que $T(n) = \Theta(f(n))$ si $c_1 f(n) \leq T(n) \leq c_2 f(n)$ para todo $n \geq n_0$

- Es decir, la notación *Big-Theta* marca cotas inferiores y superiores.
- Entonces, en el ejemplo previo, $T(n) = \Theta(n)$

# Complejidad Temporal y Espacial

- Se define como *complejidad temporal* de un algoritmo al tiempo necesario por un algoritmo para su ejecución.
- Se define como *complejidad espacial* de un algoritmo a la memoria requerida por un algoritmo para su ejecución.
- En este curso, cuando hablemos de *complejidad* nos estamos refiriendo a la complejidad temporal.
- Siempre utilizamos notación asintótica para expresar cualquier complejidad.
- Usualmente estamos interesados en las cotas superiores, pues marcan el peor de los casos de la ejecución de los algoritmos, y nos referimos a esta cuando se habla simplemente de complejidad.

# Algoritmos eficientes

## Definition

Se considera un algoritmo como eficiente cuando su complejidad es polinómica.

- Por ejemplo, sea $A$ un algoritmo cuya complejidad es $O(n^{100})$ y $B$ un algoritmo cuya complejidad es $O(2^n)$
- entonces, $A$ se dice que a es eficiente y $B$, de complejidad exponencial, no lo es.