Notación asintótica y análisis de algoritmos

Clase 18

IIC 1253

Prof. Cristian Riveros

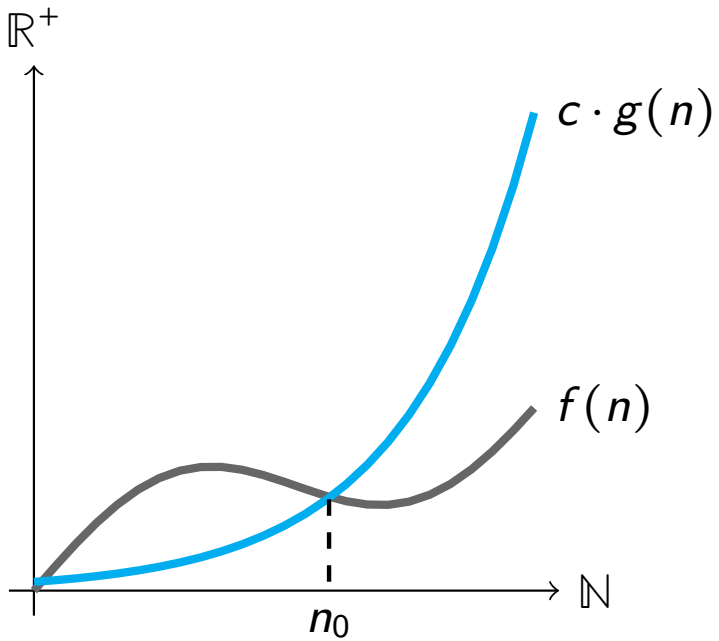
Recordatorio: Notación $\mathcal{O}$

Sea $f : \mathbb{N} \to \mathbb{R}^+$ y $g : \mathbb{N} \to \mathbb{R}^+$.

Definición

Se define el conjunto $\mathcal{O}(g)$ de todas las funciones $f : \mathbb{N} \to \mathbb{R}^+$ tal que **existe** $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que:

$$f(n) \leq c \cdot g(n) \quad \textbf{para todo } n \geq n_0$$



Recordatorio: Notación $\mathcal{O}$

Sea $f : \mathbb{N} \to \mathbb{R}^+$ y $g : \mathbb{N} \to \mathbb{R}^+$.

Definición

Se define el conjunto $\mathcal{O}(g)$ de todas las funciones $f : \mathbb{N} \to \mathbb{R}^+$ tal que **existe** $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que:

$$f(n) \leq c \cdot g(n) \quad \textbf{para todo } n \geq n_0$$

En notación lógica:

$$\mathcal{O}(g) = \{ f : \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+.\ \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ \ f(n) \leq c \cdot g(n) \}$$

Pensar en $f \in \mathcal{O}(g)$ como decir que f "**crece más lento o igual**" que g.

Recordatorio: Algunas propiedades de la notación $\mathcal{O}$

Definición

$$\mathcal{O}(g) = \{ f : \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+.\ \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ \ f(n) \leq c \cdot g(n) \}$$

Propiedades

- Si $f(n) \leq g(n)$ para todo $n \in \mathbb{N}$, entonces $f \in \mathcal{O}(g)$.
- Para todo $k \in \mathbb{N}$, si $f \in \mathcal{O}(g)$ entonces $k \cdot f \in \mathcal{O}(g)$.
- Para todo g creciente y $k \in \mathbb{N}$, si $f \in \mathcal{O}(g)$ entonces $f + k \in \mathcal{O}(g)$.
- Si $f \in \mathcal{O}(g)$ y $g \in \mathcal{O}(h)$, entonces $f \in \mathcal{O}(h)$.

Recordatorio: Notación $\mathcal{O}$ para algunas funciones

Teorema

1. Sea $f(x) = a_k x^k + \ldots + a_1 x + a_0$ un polinomio sobre $\mathbb{N}$, entonces:

$$f \in \mathcal{O}(x^k)$$

2. $x^{k+1} \notin \mathcal{O}(x^k)$ para todo $k \in \mathbb{N}$.

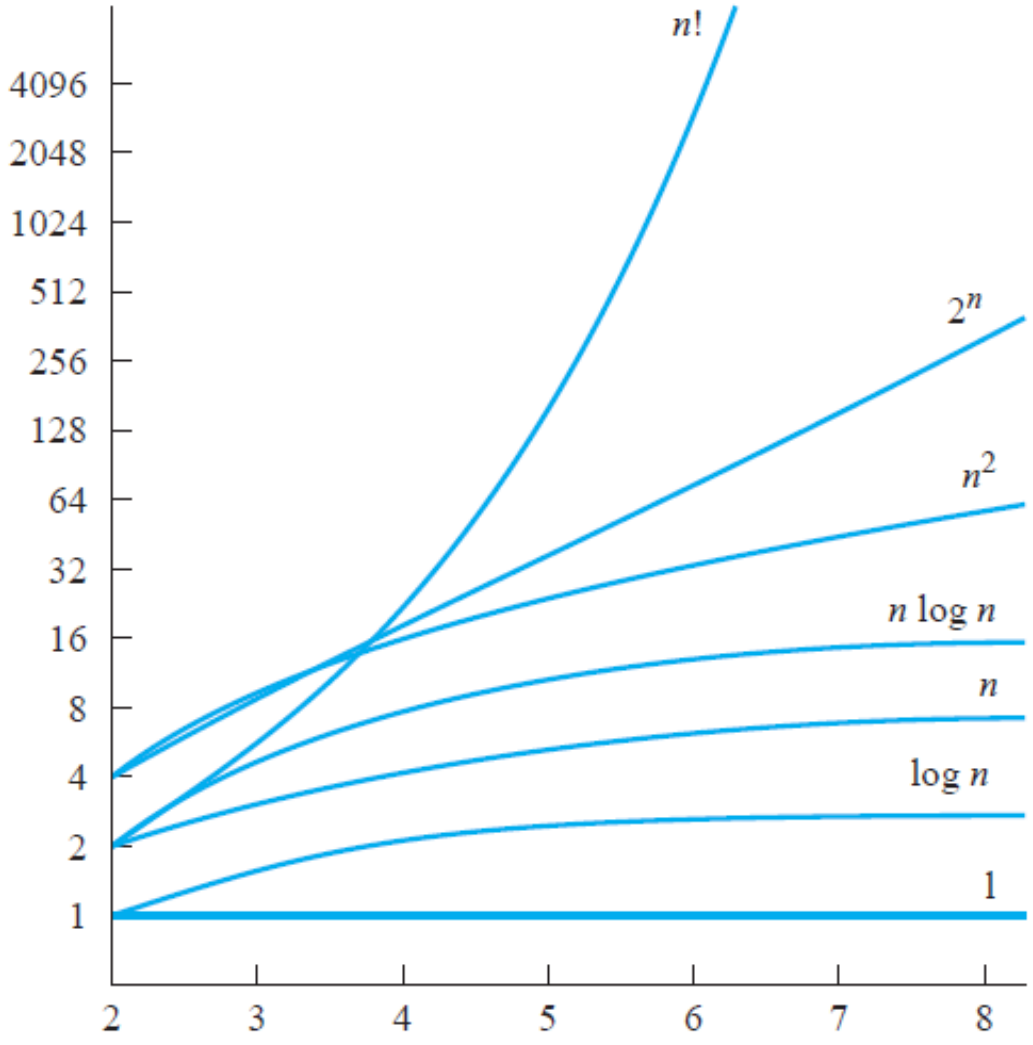
3. Para todo $a, b > 1$, se tiene que $\log_a(n) \in \mathcal{O}(\log_b(n))$.

4. Para todo $a < b$ con $a, b \in \mathbb{N}$, se tiene que $a^n \in \mathcal{O}(b^n)$ y $b^n \notin \mathcal{O}(a^n)$.

5. Para todo $a \in \mathbb{N}$, se tiene que $a^n \in \mathcal{O}(n!)$ y $n! \notin \mathcal{O}(a^n)$.

6. $n! \in \mathcal{O}(2^{n \cdot \log(n)})$.

Recordatorio: Jerarquía en notación $\mathcal{O}$



Combinaciones de funciones en notación $\mathcal{O}$

Teorema

Si $f_1 \in \mathcal{O}(g_1)$ y $f_2 \in \mathcal{O}(g_2)$, entonces $f_1 + f_2 \in \mathcal{O}(\max\{g_1, g_2\})$.

Demostración

Suponga que:

- existe $C_1 \in \mathbb{R}^+$, $n_0^1 \in \mathbb{N}$ tal que $f_1(n) \leq C_1 \cdot g_1(n)$ para todo $n \geq n_0^1$.
- existe $C_2 \in \mathbb{R}^+$, $n_0^2 \in \mathbb{N}$ tal que $f_2(n) \leq C_2 \cdot g_2(n)$ para todo $n \geq n_0^2$.

Si $n_0 = \max\{n_0^1, n_0^2\}$ y $C = C_1 + C_2$, entonces para todo $n \geq n_0$:

$$\begin{aligned} f_1(n) + f_2(n) &\leq C_1 \cdot g_1(n) + C_2 \cdot g_2(n) \\ &\leq C_1 \cdot \max\{g_1(n), g_2(n)\} + C_2 \cdot \max\{g_1(n), g_2(n)\} \\ &\leq (C_1 + C_2) \cdot \max\{g_1(n), g_2(n)\} \end{aligned}$$

Notar que $f_1 \in \mathcal{O}(g)$ y $f_2 \in \mathcal{O}(g)$ implica $f_1 + f_2 \in \mathcal{O}(g)$

Combinaciones de funciones en notación $\mathcal{O}$

Teorema

Si $f_1 \in \mathcal{O}(g_1)$ y $f_2 \in \mathcal{O}(g_2)$, entonces $f_1 \cdot f_2 \in \mathcal{O}(g_1 \cdot g_2)$.

Demostración

Suponga que:

- existe $C_1 \in \mathbb{R}^+$, $n_0^1 \in \mathbb{N}$ tal que $f_1(n) \leq C_1 \cdot g_1(n)$ para todo $n \geq n_0^1$.
- existe $C_2 \in \mathbb{R}^+$, $n_0^2 \in \mathbb{N}$ tal que $f_2(n) \leq C_2 \cdot g_2(n)$ para todo $n \geq n_0^2$.

Si $n_0 = \max\{n_0^1, n_0^2\}$ y $C = C_1 \cdot C_2$, entonces para todo $n \geq n_0$:

$$\begin{aligned} f_1(n) \cdot f_2(n) &\leq C_1 \cdot g_1(n) \cdot C_2 \cdot g_2(n) \\ &\leq (C_1 \cdot C_2) \cdot (g_1(n) \cdot g_2(n)) \end{aligned}$$

Combinaciones de funciones en notación $\mathcal{O}$

Ejemplo

De un buen estimador del orden de la siguientes funciones:

- $(x+1)\cdot\log(x^2+1) \;+\; 3\cdot x^2$
- $3\cdot x\cdot\log(x!) \;+\; (x^2+3)\cdot\log(x)$

Outline

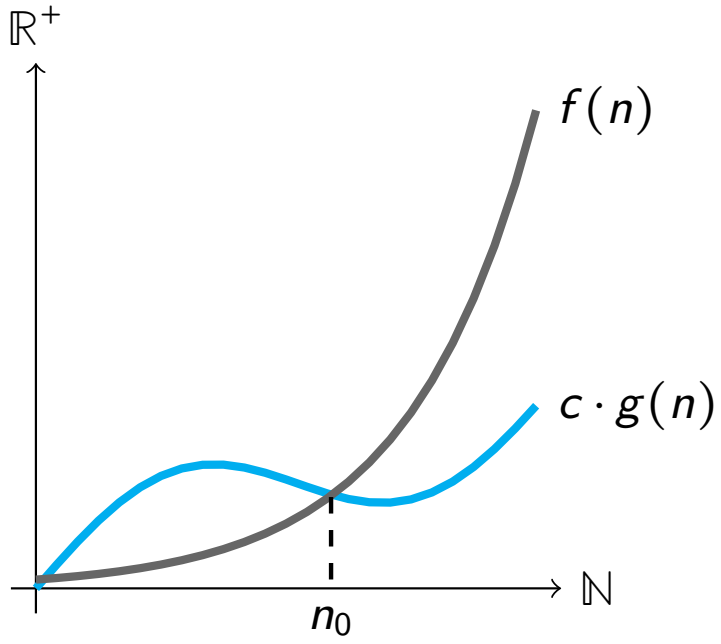
Notación Ω

Sea $f : \mathbb{N} \to \mathbb{R}^+$ y $g : \mathbb{N} \to \mathbb{R}^+$.

Definición

Se define el conjunto $\Omega(g)$ de todas las funciones $f : \mathbb{N} \to \mathbb{R}^+$ tal que **existe** $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que:

$$f(n) \geq c \cdot g(n) \quad \textbf{para todo } n \geq n_0$$



Notación Ω

Sea $f : \mathbb{N} \to \mathbb{R}^+$ y $g : \mathbb{N} \to \mathbb{R}^+$.

Definición

Se define el conjunto $\Omega(g)$ de todas las funciones $f : \mathbb{N} \to \mathbb{R}^+$ tal que **existe** $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que:

$$f(n) \geq c \cdot g(n) \quad \textbf{para todo } n \geq n_0$$

En notación matemática:

$$\Omega(g) = \{ f : \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+.\ \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ f(n) \geq c \cdot g(n) \}$$

Notación

Cuando $f \in \Omega(g)$ diremos que "f es $\Omega(g)$".

Intuitivamente, $f \in \Omega(g)$ si "f **crece más rápido o igual** que g".

Notación Ω

Definición

$$\Omega(g) = \{ f : \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+.\ \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ \ f(n) \geq c \cdot g(n) \}$$

Ejemplo

Considere la función $f(x) = x^4 + 2x^2 + 5$ y $g(x) = 5x^4$.

$$\text{¿}\ \ x^4 + 2x^2 + 5 \in \Omega(5x^4)\ \ ?$$

Para $n \geq 1$ tenemos que:

$$n^4 + 2n^2 + 5 \ \geq\ \frac{1}{5} \cdot 5n^4$$

Si tomamos $c = \frac{1}{5}$ y $n_0 = 1$ entonces para todo $n \geq n_0$:

$$f(n) \ = \ n^4 + 2n^2 + 5 \ \geq\ \frac{1}{5} \cdot 5n^4 \ = \ c \cdot g(n)$$

Por lo tanto, $x^4 + 2x^2 + 5 \in \Omega(5x^4)$.

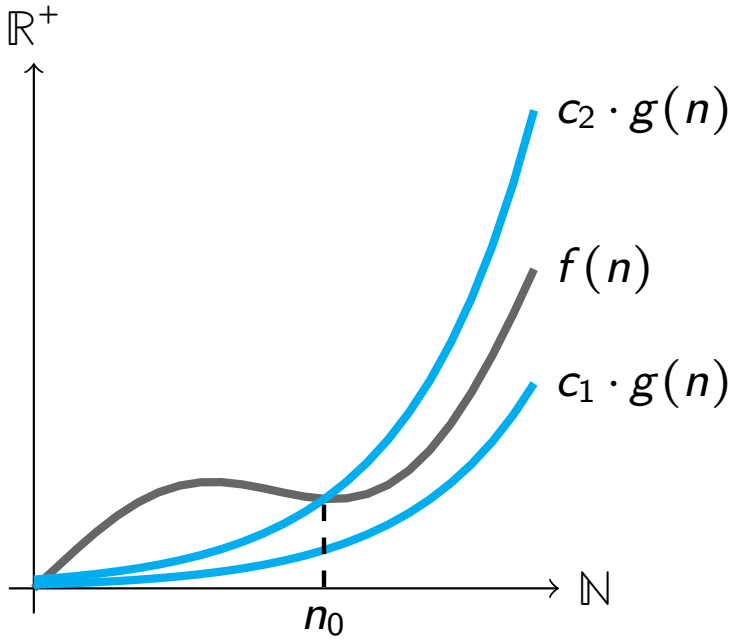
Notación Θ

Sea $f : \mathbb{N} \to \mathbb{R}^+$ y $g : \mathbb{N} \to \mathbb{R}^+$.

Definición

Se define el conjunto $\Theta(g)$ de todas las funciones $f : \mathbb{N} \to \mathbb{R}^+$ tal que **existen** $c_1, c_2 \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que:

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \textbf{para todo } n \geq n_0$$



Notación Θ

Sea $f : \mathbb{N} \to \mathbb{R}^+$ y $g : \mathbb{N} \to \mathbb{R}^+$.

Definición

Se define el conjunto $\Theta(g)$ de todas las funciones $f : \mathbb{N} \to \mathbb{R}^+$ tal que **existen** $c_1, c_2 \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que:

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \textbf{para todo } n \geq n_0$$

En notación matemática:

$$\Theta(g) = \{ f : \mathbb{N} \to \mathbb{R}^+ \mid \exists c_1, c_2 \in \mathbb{R}^+ . \exists n_0 \in \mathbb{N} . \forall n \geq n_0 . c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \}$$

En otras palabras, $f \in \Theta(g)$ si, y solo si, $f \in \Omega(g)$ y $f \in \mathcal{O}(g)$.

(demuestre esta afirmación)

Notación Θ

Definición

$$\Theta(g) = \{ f : \mathbb{N} \to \mathbb{R}^+ \mid \exists c_1, c_2 \in \mathbb{R}^+ . \exists n_0 \in \mathbb{N} . \forall n \geq n_0 . c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \}$$

Ejemplo

Considere la función $g(x) = x^k$ y $f(x) = a_k x^k + \ldots + a_1 x + a_0$.

$$¿ \quad a_k x^k + \ldots + a_1 x + a_0 \in \Theta(x^k) \quad ?$$

Ya sabemos que $f \in \mathcal{O}(g)$ por lo que queda demostrar que $f \in \Omega(g)$.

Para $n \geq n_0 = 1$ y $c = a_k$ tenemos que:

$$a_k n^k + \ldots + a_1 n + a_0 \geq c \cdot n^k + \ldots + 0 \cdot n + 0 = c \cdot n^k$$

Por lo tanto, $f \in \Omega(g)$.

Propiedades de notación Θ

Teorema

1. Sea $f(n) = a_k n^k + \ldots + a_1 n + a_0$ un polinomio sobre $\mathbb{N}$, entonces:

$$f \in \Theta(n^k)$$

2. $n^k \notin \Omega(n^{k+1})$ para todo $k > 0$.

Demostración (ejercicio)

Propiedades de notación Θ

Teorema

1. Para todo $a, b > 1$, se tiene que $\log_a(n) \in \Theta(\log_b(n))$.
2. Si $f_1 \in \Theta(g)$ y $f_2 \in \Theta(g)$, entonces $f_1 + f_2 \in \Theta(g)$.
3. Si $f_1 \in \Theta(g_1)$ y $f_2 \in \Theta(g_2)$, entonces $f_1 \cdot f_2 \in \Theta(g_1 \cdot g_2)$.

Demostración (ejercicio)

Sobre la notación Θ

Podemos ver Θ como una relación entre funciones:

$$\text{“}(f, g) \in R_\Theta\text{”} \quad \text{si, y solo si,} \quad f \in \Theta(g)$$

¿es Θ una “relación de equivalencia”?

- Refleja?
- Simétrica?
- Transitiva?

Outline

Recordatorio: ¿qué es un algoritmo?

Definición

Un **algoritmo** es una secuencia finita de instrucciones precisas para realizar una computación o resolver un problema.

Un algoritmo puede estar dado por cualquier lenguaje:

- Lenguaje de programación.
  - Python, Java, C++, etc
- Lenguaje natural.
- Pseudo-código.

Nuestros algoritmos serán generalmente en **pseudo-código**.

Recordatorio: eficiencia con respecto al tiempo

Definición

Para un **algoritmo** A sobre un conjunto de inputs $\mathcal{I}$ se define la función:

$$\text{tiempo}_A : \mathcal{I} \to \mathbb{N}$$

tal que para todo input $I \in \mathcal{I}$:

$$\text{tiempo}_A(I) \;=\; \text{número de } \textbf{pasos} \text{ realizados por } A \text{ con input } I$$

Uso de notación asintótica en algoritmos

Considere el siguiente fragmento de un algoritmo:

```
for i = 1 to n do
    for j = 1 to i do
        x := x + 1
```

¿cuántas veces se ejecuta la linea $x := x + 1$ según n?

Si el número de veces que se ejecuta $x := x + 1$ es $T(n)$, entonces:

$$T(n) = 1 + 2 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Por lo tanto, la cantidad de veces es $\Theta(n^2)$.

Uso de notación asintótica en algoritmos

Considere el siguiente fragmento de un algoritmo:

$j := n$
while $j \geq 1$ **do**
    for $i = 1$ **to** j **do**
        $x := x + 1$
    $j := \left\lfloor \frac{j}{2} \right\rfloor$

¿cuántas veces se ejecuta la linea $x := x + 1$ según n?

Si el número de veces que se ejecuta $x := x + 1$ es $T(n)$, entonces:

$$T(n) = n + \frac{n}{2} + \frac{n}{4} + \ldots + 1 \leq \sum_{j=0}^{\infty} \frac{n}{2^j} \leq 3 \cdot n$$

Por lo tanto, la cantidad de veces es $\mathcal{O}(n)$. ¿es $\Theta(n)$?

Uso de notación asintótica en algoritmos

input : Una secuencia $S = (a_1, \ldots, a_n)$, el largo n y un elemento a.
output: La primera posición donde aparece a y -1 si no aparece.

Function `BusquedaIngenua` *(S, n, a)*

```
k := 1
while k ≤ n do
    if a_k = a then
        return k
    k := k + 1
return −1
```

¿cuántas veces se ejecuta el **while** según n?

Depende:

- Si $a_1 = a$, entonces se ejecutará 1 vez.
- Si $a_n = a$ y $a_j \neq a$ para $j < n$, entonces se ejecutará n-veces.

¿es el tiempo del algoritmo $\Theta(1)$ o $\Theta(n)$?

Uso de notación asintótica en algoritmos

En el caso anterior tenemos dos **problemas**:

1. El *input* NO depende **solo** de n.
2. El tiempo depende del la **distribución/forma** del *input*.

Para esto debemos considerar:

- Como medir el **tamaño** de una instancia.
- Como medir el **tiempo del algoritmo** sin depender del *input*.

Tamaño del *input*

Definición

Para un conjunto de *inputs* $\mathcal{I}$ se define su función tamaño:

$$|\cdot| : \mathcal{I} \to \mathbb{N}$$

tal que para todo *input* $I \in \mathcal{I}$:

$$|I| = \text{es el tamaño de } I \text{ según su "}\textbf{representación}\text{".}$$

En general, $|I|$ será un valor que "representa" el tamaño de I y que nos será útil en nuestro análisis/modelación.

Tamaño del input

Ejemplos

- Para la palabra de bits $w \in \{0,1\}^*$:

$$|w| = \text{largo de la palabra } w \text{ (número de bits)}$$

- Para un número $n \in \mathbb{N}$:

$$|n| = \text{número de bits o símbolos para representar } n$$

Tamaño del input

Ejemplos

- Para una relación $R \subseteq A \times A$:

$$|R| = \text{número de tuplas en } R$$

- Para un grafo $G = (V, E)$:

$$|G| = \text{número de vertices } V + \text{número de aristas } E$$

- Para un secuencia $S = (a_1, \ldots, a_n)$, el largo n y un elemento a :

$$|(S, n, a)| = \sum_{i=0}^{n} |a_i| + n + |a| \quad o \quad |(S, n, a)| = n$$

¡El tamaño de las instancias depende del detalle del análisis!

Tamaño del *input*

Definición

Para un conjunto de *inputs* $\mathcal{I}$ se define su función tamaño:

$$|\cdot| : \mathcal{I} \to \mathbb{N}$$

tal que para todo *input* $I \in \mathcal{I}$:

$$|I| = \text{es el tamaño de } I \text{ según su “}\textbf{representación}\text{”}.$$

En general

La definición más absoluta y general del tamaño $|I|$:

$$|I| = \text{número de bits de una } \textbf{codificación} \text{ “razonable” de I.}$$

Siempre vamos a depender de la codificación del *input*.

Tipos de complejidad

Definición

Para un algoritmo A y su conjunto de *inputs* I se definen las funciones:

$$\text{peor-caso}_A : \mathbb{N} \to \mathbb{N} \quad \text{y} \quad \text{mejor-caso}_A : \mathbb{N} \to \mathbb{N}$$

- Función de complejidad en el **peor caso** de A:

$$\text{peor-caso}_A(n) = \max_{I \in \mathcal{I}} \{ \text{ tiempo}_A(I) \mid |I| = n \}$$

- Función de complejidad en el **mejor caso** de A:

$$\text{mejor-caso}_A(n) = \min_{I \in \mathcal{I}} \{ \text{ tiempo}_A(I) \mid |I| = n \}$$

Tipos de complejidad

Ejemplo

input : Una secuencia $S = (a_1, \ldots, a_n)$, el largo n y elemento a.
output: La primera posición donde aparece a y -1 si no aparece.

Function `BusquedaIngenua` *(S, n, a)*
    $k := 1$
    while $k \leq n$ **do**
        if $a_k = a$ **then**
            return k
        $k := k + 1$
    return -1

- ¿cuál es su función de complejidad en el **peor-caso**?

$$\text{peor-caso}_{\text{Busquedad}}(n) = n$$

- ¿cuál es su función de complejidad en el **mejor-caso**?

$$\text{mejor-caso}_{\text{Busquedad}}(n) = 1$$

Tipos de complejidad

Estamos interesados en el comportamiento **asintótico** de

peor-caso$_A$ o mejor-caso$_A$

El análisis de la complejidad del algoritmo A corresponde a encontrar f:

- peor-caso$_A \in \mathcal{O}(f)$ o
- peor-caso$_A \in \Theta(f)$.

Diremos que f es la **complejidad** de A en el **peor caso**.

Análisis de complejidad de BusquedaIngenua

input : Una secuencia $S = (a_1, \ldots, a_n)$, el largo n y elemento a.
output: La primera posición donde aparece a y -1 si no aparece.

Function `BusquedaIngenua` *(S, n, a)*

```
k := 1
while k ≤ n do
    if a_k = a then
        return k
    k := k + 1
return −1
```

- Complejidad en el **peor caso**: $\Theta(n)$
- Complejidad en el **mejor caso**: $\Theta(1)$

Análisis de complejidad de BusquedaBinaria

input : Una sec. creciente $S = (a_1, \ldots, a_n)$, el largo n y elemento a.
output: Alguna posición donde aparece a y -1 si no aparece.

Function `BusquedaBinaria` *(S, n, a)*

```
i := 1, j := n
while i < j do
    m := ⌊(i+j)/2⌋
    if a_m < a then i := m + 1
    else j := m
if a_i = a then return i
else return −1
```

- ¿complejidad en el **peor caso**? $\Theta(\log(n))$
- ¿complejidad en el **mejor caso**? $\Theta(\log(n))$

Análisis de complejidad de EsPrimo

input : Un número n.
output: `TRUE` si n es primo y `FALSE` si no.

Function `EsPrimo` (n)
    for $i = 2$ **to** $n - 1$ **do**
        if $n \mod i = 0$ **then**
            return `FALSE`
    return `TRUE`

- ¿complejidad en el **peor caso**? $\mathcal{O}(n)$ (¿cuál es el peor caso?)
- ¿complejidad en el **mejor caso**? $\Omega(1)$ (¿cuál es el mejor caso?)

¿es correcto el análisis? ¿cuál es el tamaño de n?