Algoritmos:
Análisis de Algoritmos

Alberto Valderruten

Dept. de Ciencias de la Computación y Tecnologías de la Información,
Universidade da Coruña

alberto.valderruten@udc.es

UNIVERSIDADE DA CORUÑA

Contenido

1. Análisis de la eficiencia de los algoritmos
2. Notaciones asintóticas
3. Cálculo de los tiempos de ejecución

Índice

1 Análisis de la eficiencia de los algoritmos

2 Notaciones asintóticas

3 Cálculo de los tiempos de ejecución

Análisis de la eficiencia de los algoritmos (1)

- **Objetivo:** *Predecir* el *comportamiento* del algoritmo
  $\Rightarrow$ aspectos cuantitativos:
  tiempo de ejecución, cantidad de memoria
- Disponer de una *medida* de su eficiencia:
  - "teórica"
  - no exacta: *aproximación* suficiente para *comparar*, *clasificar*
  $\Rightarrow$ acotar $T(n)$: tiempo de ejecución,
  $n =$ *tamaño del problema* (a veces, de la entrada)
  $n \to \infty$: *comportamiento asintótico*
  $\Rightarrow T(n) = O(f(n))$
  $f(n)$: una **cota superior** de $T(n)$ suficientemente ajustada
  $f(n)$ crece más deprisa que $T(n)$

Análisis de la eficiencia de los algoritmos (2)

- **Aproximación?**
  1. *Ignorar factores constantes*:
  20 multiplicaciones por iteración $\rightarrow$ 1 **operación** por iteración
  ¿cuántas iteraciones? $\rightarrow$ iteraciones en función de n
  2. *Ignorar términos de orden inferior*: $n + cte \rightarrow n$

- **Ejemplo 1**:
  2 algoritmos (A1 y A2) para un mismo problema A
  - algoritmo A1: $100n$ pasos $\rightarrow$ un recorrido de la entrada
    $T(n) = O(n)$: algoritmo *lineal*
  - algoritmo A2: $2n^2 + 50$ pasos $\rightarrow$ n recorridos de la entrada
    $T(n) = O(n^2)$: algoritmo *cuadrático*

Análisis de la eficiencia de los algoritmos (3)

- **Ejemplo 1 (Cont.)**:
  $\Rightarrow$ A1 lineal y A2 cuadrático:
  - *Comparar*: A2 "más lento" que A1,
    aunque con $n \leq 49$ sea más rápido
    $\Rightarrow$ **A1 es mejor**
  - *Clasificar*: lineales, cuadráticos...
    Tasas de crecimiento características:
    $O(1), O(logn), O(n), O(nlogn), O(n^2), O(n^3), \ldots O(2^n), \ldots$
- **Ejemplo 2**: (aproximación $\Rightarrow$ limitaciones)
  2 algoritmos (B1 y B2) para un mismo problema B:
  - algoritmo B1: $2n^2 + 50$ pasos $\rightarrow O(n^2)$
  - algoritmo B2: $100n^{1,8}$ pasos $\rightarrow O(n^{1,8})$
    $\Rightarrow$ B2 es "mejor"...
  pero a partir de algún valor de n **entre** 310 **y** $320 * 10^6$

Índice

1. Análisis de la eficiencia de los algoritmos
2. Notaciones asintóticas
3. Cálculo de los tiempos de ejecución

Notaciones asintóticas

- **Objetivo:** Establecer un orden relativo entre las funciones, comparando sus tasas de crecimiento
- **La notación O:**
  $T(n), f(n) : Z^+ \rightarrow R^+$
  Definición: $T(n) = O(f(n))$
  si $\exists$ constantes $c > 0$ y $n_0 > 0$: $T(n) \leq c * f(n) \; \forall n \geq n_0$
  $\square$
  n_0: umbral
  $T(n)$ es $O(f(n))$, $T(n) \in O(f(n))$
  "la tasa de crecimiento de $T(n) \leq$ que la de $f(n)$"
  $\rightarrow f(n)$ es *una* **cota superior** de $T(n)$
- **Ejemplo:** ¿$5n^2 + 15 = O(n^2)$?
  $< c, n_0 > = < 6, 4 >$ en la definición: $5n^2 + 15 \leq 6n^2 \; \forall n \geq 4$;
  $\exists$ infinitos $< c, n_0 >$ que satisfacen la desigualdad

La notación O (1)

- **Observación**:
  Según la definición, $T(n)$ podría estar muy por debajo:
  ¿$5n^2 + 15 = O(n^3)$?
  $< c, n_0 > = < 1, 6 >$ en la definición: $5n^2 + 15 \leq 1n^3 \; \forall n \geq 6$
  pero es más preciso decir $= O(n^2) \equiv$ **ajustar cotas**
  $\Rightarrow$ **Para el análisis de algoritmos, usar las aproximaciones:**
  $5n^2 + 4n \rightarrow O(n^2)$
  $log_2 n \rightarrow O(logn)$
  $13 \rightarrow O(1)$
- **Observación**:
  La notación O también se usa en expresiones como $3n^2 + O(n)$
- **Ejemplo 3**:
  ¿Cómo se consigue una mejora más drástica,
  - mejorando la eficiencia del algoritmo, o
  - mejorando el ordenador?

La notación O (2)

- **Ejemplo 3** (cont.):

$T(n)$	tiempo$_1$ 1000 pasos/s	tiempo$_2$ 2000 pasos/s	tiempo$_3$ 4000 pasos/s	tiempo$_4$ 8000 pasos/s
$log_2 n$	0,010	0,005	0,003	0,001
n	1	0,5	0,25	0,125
$nlog_2 n$	10	5	2,5	1,25
$n^{1,5}$	32	16	8	4
n^2	1.000	500	250	125
n^3	1.000.000	500.000	250.000	125.000
$1,1^n$	10^{39}	10^{39}	10^{38}	10^{38}

Tabla: Tiempos de ejecución (en s) para 7 algoritmos de distinta complejidad (n=1000).

- **Ejemplo 4**: Ordenar 100.000 enteros aleatorios:
  * 17 s en un 386 + Quicksort
  * 17 min en un procesador 100 veces más rápido + Burbuja

La notación O (3)

Reglas prácticas para trabajar con la O:

Definición: $f(n)$ es **monótona creciente** si $n_1 \geq n_2 \Rightarrow f(n_1) \geq f(n_2)$ $\square$

- **Teorema**: $\forall c > 0, a > 1, f(n)$ monótona creciente:

$$(f(n))^c = O(a^{f(n)})$$

$\equiv$ *"Una función exponencial (ej: 2^n) crece más rápido que una función polinómica (ej: n^2)"*

$$\rightarrow \begin{cases} n^c = O(a^n) \\ (log_a n)^c = O(a^{log_a n}) = O(n) \end{cases}$$

$$\rightarrow (logn)^k = O(n) \ \forall k \text{ cte.}$$

$\equiv$ *"n crece más rápido que cualquier potencia de logaritmo"*

$\equiv$ *"los logaritmos crecen muy lentamente"*

La notación O (4)

Reglas prácticas para trabajar con la O (Cont.):

- **Suma y multiplicación**:

  $$T_1(n) = O(f(n)) \wedge T_2(n) = O(g(n)) \Rightarrow$$
  $$\begin{cases} (1) \quad T_1(n) + T_2(n) = O(f(n) + g(n)) = max(O(f(n)), O(g(n))) \\ (2) \quad T_1(n) * T_2(n) = O(f(n) * g(n)) \end{cases}$$

  $$\text{Aplicación: } \begin{cases} \text{(1) Secuencia:} & 2n^2 = O(n^2) \wedge 10n = O(n) \\ & \Rightarrow 2n^2 + 10n = O(n^2) \\ \text{(2) Bucles} & \end{cases}$$

  Observación: **No** extender la regla: ni resta, ni división

  $\leftarrow$ relación $\leq$ en la definición de la O

... suficientes para ordenar la mayoría de las funciones.

Otras notaciones asintóticas (1)

1. $T(n), f(n) : Z^+ \rightarrow R^+$, **Definición**: O
2. **Definición**: $T(n) = \Omega(f(n))$
   ssi $\exists$ constantes c y n_0: $T(n) \geq cf(n)\ \forall n \geq n_0$
   $\square$
   $f(n)$: **cota inferior** de $T(n) \equiv$ trabajo mínimo del algoritmo
   Ejemplo: $3n^2 = \Omega(n^2)$: cota inferior más ajustada...
   pero $3n^2 = O(n^2)$ también! $(O \wedge \Omega)$
3. **Definición**: $T(n) = \Theta(f(n))$
   ssi $\exists$ constantes c_1, c_2 y n_0: $c_1 f(n) \leq T(n) \leq c_2 f(n)\ \forall n \geq n_0$
   $\square$
   $f(n)$: **cota exacta** de $T(n)$, del orden exacto
   Ejemplo: $5nlog_2 n - 10 = \Theta(nlogn)$:
   $$\begin{cases} (1)\ \text{demostrar } O \rightarrow < c, n_0 > \\ (2)\ \text{demostrar } \Omega \rightarrow < c', n_0' > \end{cases}$$

Otras notaciones asintóticas (2)

4. **Definición**: $T(n) = o(f(n))$
   ssi $\forall$ constante $C > 0$, $\exists n_0 > 0$: $T(n) < Cf(n)\ \forall n \geq n_0$
   $\square$
   $\equiv O \wedge \neg\Theta \equiv O \wedge \neg\Omega$
   $f(n)$: **cota estrictamente superior** de $T(n)$: $\lim_{n\to\infty} \frac{T(n)}{f(n)} = 0$
   Ejemplos: $\frac{n}{log_2 n} = o(n)$ $\frac{n}{10} \neq o(n)$
5. **Definición**: $T(n) = \omega(f(n))$
   ssi $\forall$ constante $C > 0$, $\exists n_0 > 0$: $T(n) > Cf(n)\ \forall n \geq n_0$
   $\square$
   $\leftrightarrow f(n) = o(T(n))$
   $\rightarrow f(n)$: **cota estrictamente inferior** de $T(n)$
6. **Notación OO** [Manber]: $T(n) = OO(f(n))$ si es $O(f(n))$
   pero con constantes demasiado grandes para casos prácticos
   Ref: Ejemplo 2 (p. 4): $B1 = O(n^2)$, $B2 = OO(n^{1,8})$

Otras notaciones asintóticas (3)

Reglas prácticas (Cont.):

- $T(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_k n^k \Rightarrow T(n) = \Theta(n^k)$
  (polinomio de grado k)
- **Teorema**: $\forall c > 0, a > 1, f(n)$ monótona creciente:

$$(f(n))^c = o(a^{f(n)})$$

$\equiv$ *"Una función exponencial **crece más rápido** que una función polinómica"*
$\rightarrow$ no llegan a igualarse

Índice

1 Análisis de la eficiencia de los algoritmos

2 Notaciones asintóticas

3 Cálculo de los tiempos de ejecución

Modelo de computación (1)

- Calcular O para $T(n) \equiv$ número de "pasos" $\rightarrow f(n)$? ¿*paso*?
- **Modelo de computación**:
  - ordenador secuencial
  - instrucción $\leftrightarrow$ paso (no hay instrucciones complejas)
  - entradas: tipo único ("entero") $\rightarrow$ sec(n)
  - memoria infinita + *"todo está en memoria"*
- Alternativas: Un *paso* es...
  1. **Operación elemental**:
     Operación cuyo tiempo de ejecución está acotado superiormente por una constante que sólo depende de la implementación
     $\rightarrow = O(1)$
  2. **Operación principal** [Manber]:
     Operación *representativa* del trabajo del algoritmo:
     El número de operaciones principales que se ejecutan debe ser *proporcional* al número total de operaciones (verificarlo!).
     Ejemplo: la comparación en un algoritmo de ordenación

Modelo de computación (2)

- La hipótesis de la op. principal supone una aproximación mayor!
- En general, **usaremos la hipótesis de la operación elemental**.
- En cualquier caso, se ignora: lenguaje de programación, procesador, sistema operativo, carga...
  $\Rightarrow$ Sólo se considera el algoritmo, el tamaño del problema, ...
- **Debilidades**:
  - operaciones de coste diferente
    (“todo en memoria” $\Rightarrow$ lectura en disco = asignación)
    $\rightarrow$ contar separadamente según tipo de instrucción y luego *ponderar* $\equiv$ factores $\equiv$ dependiente de la implementación
    $\Rightarrow$ costoso y generalmente inútil
  - faltas de página ignoradas
  - etc.

$\rightarrow$ *Aproximación*

Análisis de casos

- **Análisis de casos**:
  Consideramos distintas funciones para $T(n)$:
  $$\begin{cases} T_{mejor}(n) & \\ T_{medio}(n) & \leftarrow \text{representativa, más complicada de obtener} \\ T_{peor}(n) & \leftarrow \text{en general, la más utilizada} \end{cases}$$
  $$T_{mejor}(n) \leq T_{medio}(n) \leq T_{peor}(n)$$
- ¿El tiempo de respuesta es crítico?
  $\rightarrow$ *Sistemas de Tiempo Real*

Ordenación por Inserción (1)

```
procedimiento Ordenación por Inserción (var T[1..n])
    para i:=2 hasta n hacer
        x:=T[i];
        j:=i-1;
        mientras j>0 y T[j]>x hacer
            T[j+1]:=T[j];
            j:=j-1
        fin mientras;
        T[j+1]:=x
    fin para
fin procedimiento
```

Ordenación por Inserción (2)

3	$\boxed{1}$	4	1	2	9	5	6	5	3
1	3	$\boxed{4}$	1	2	9	5	6	5	3
1	3	**4**	$\boxed{1}$	2	9	5	6	5	3
1	**1**	3	4	$\boxed{2}$	9	5	6	5	3
1	1	**2**	3	4	$\boxed{9}$	5	6	5	3
1	1	2	3	4	**9**	$\boxed{5}$	6	5	3
1	1	2	3	4	**5**	9	$\boxed{6}$	5	3
1	1	2	3	4	5	**6**	9	$\boxed{5}$	3
1	1	2	3	4	5	**5**	6	9	$\boxed{3}$
1	1	2	3	**3**	4	5	5	6	9

Análisis de casos: Ordenación por Inserción

- **Peor caso** $\rightarrow$ "insertar siempre en la primera posición"
  $\equiv$ entrada en orden inverso
  $\Rightarrow$ el bucle interno se ejecuta 1 vez en la primera iteración,
  2 veces en la segunda, ..., $n-1$ veces en la última:
  $\Rightarrow \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ iteraciones del bucle interno

  $\boxed{\sum_{i=1}^{n} i = \frac{n(n+1)}{2}}$

  $\Rightarrow T(n) = \frac{n(n-1)}{2}c_1 + (n-1)c_2 + c_3$: polinomio de grado 2
  $\Rightarrow \boxed{T(n) = \Theta(n^2)}$
- **Mejor caso** $\rightarrow$ "no insertar nunca" $\equiv$ entrada ordenada
  $\Rightarrow$ el bucle interno no se ejecuta
  $\Rightarrow T(n) = (n-1)c_1 + c_2$: polinomio de grado 1
  $\Rightarrow \boxed{T(n) = \Theta(n)}$

$\Rightarrow T(n)$ depende *también* del estado inicial de la entrada

Ordenación por Selección (1)

```
procedimiento Ordenación por Selección (var T[1..n])
    para i:=1 hasta n-1 hacer
        minj:=i;
        minx:=T[i];
        para j:=i+1 hasta n hacer
            si T[j]<minx entonces
                minj:=j;
                minx:=T[j]
            fin si
        fin para;
        T[minj]:=T[i];
        T[i]:=minx
    fin para
fin procedimiento
```

Ordenación por Selección (2)

3	$\boxed{1}$	4	1	2	9	5	6	5	3
1	3	4	$\boxed{1}$	2	9	5	6	5	3
1	**1**	4	3	$\boxed{2}$	9	5	6	5	3
1	1	**2**	$\boxed{3}$	4	9	5	6	5	3
1	1	2	**3**	4	9	5	6	5	$\boxed{3}$
1	1	2	3	**3**	9	5	6	5	$\boxed{4}$
1	1	2	3	3	**4**	$\boxed{5}$	6	5	9
1	1	2	3	3	4	**5**	6	$\boxed{5}$	9
1	1	2	3	3	4	5	**5**	$\boxed{6}$	9
1	1	2	3	3	4	5	5	**6**	$\boxed{9}$

Análisis de casos: Ordenación por Selección

- $\boxed{T(n) = \Theta(n^2)}$ sea cual sea el orden inicial (ejercicio)
  $\leftrightarrow$ la comparación interna se ejecuta las mismas veces
  Empíricamente: $T(n)$ no fluctúa más del 15 %

algoritmo	mínimo	máximo
Inserción	0,004	5,461
Selección	4,717	5,174

Tabla: Tiempos (en segundos) obtenidos para $n = 4000$

- **Comparación**:

algoritmo	peor caso	caso medio	mejor caso
Inserción	$\Theta(n^2)$	$\Theta(n^2)$	$\boxed{\Theta(n)}$
Selección	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Quicksort	$O(n^2)$	$\boxed{O(nlogn)}$	$O(nlogn)$

Análisis de casos: exponenciación (1)

- Potencia1: $x^n = x * x * \ldots * x$ (bucle, n veces x)
  Operación principal: multiplicación
  ¿Número de multiplicaciones? $f_1(n) = n - 1 \Rightarrow T(n) = \Theta(n)$

- Potencia2 (recursivo):
  $$x^n = \begin{cases} x^{\lfloor n/2 \rfloor} * x^{\lfloor n/2 \rfloor} & \text{si n par} \\ x^{\lfloor n/2 \rfloor} * x^{\lfloor n/2 \rfloor} * x & \text{si n impar} \end{cases}$$
  ¿Número de multiplicaciones? ¿$f_2(n)$?

Análisis de casos: exponenciación (2)

- Potencia2 (recursivo) (Cont.)
  Cálculo de $f_2(n)$:

$$\begin{cases} \text{mín: n par en cada llamada} & \rightarrow n = 2^k, k \in Z^+ \leftrightarrow \text{mejor caso} \\ \text{máx: n impar en cada llamada} & \rightarrow n = 2^k - 1, k \in Z^+ \leftrightarrow \text{peor caso} \end{cases}$$

  - *Mejor caso*: $f_2(2^k) = \begin{cases} 0 & \text{si } k = 0 \\ f_2(2^{k-1}) + 1 & \text{si } k > 0 \text{ (1)} \end{cases}$
  - *Peor caso*: $f_2(2^k - 1) = \begin{cases} 0 & \text{si } k = 1 \\ f_2(2^{k-1} - 1) + 2 & \text{si } k > 1 \text{ (2)} \end{cases}$

$\rightarrow$ *relaciones de recurrencia*

Análisis de casos: exponenciación (3)

- *Mejor caso*: $f_2(2^k) = \begin{cases} 0 & \text{si } k = 0 \\ f_2(2^{k-1}) + 1 & \text{si } k > 0 \ (1) \end{cases}$

$$\begin{array}{llclcl} k = & 0 & \rightarrow f_2(& 1 &) = & 0 \\ & 1 & & 2 & & 1 \\ & 2 & & 4 & & 2 \\ & 3 & & 8 & & 3 \\ & \ldots & & & & \end{array}$$

$\Rightarrow$ Hipótesis de inducción: $\boxed{f_2(2^\alpha) = \alpha}$: $0 \leq \alpha \leq k - 1$

Paso inductivo:

$$\begin{aligned} (1) \rightarrow f_2(2^k) &= f_2(2^{k-1}) + 1 \\ &= (k-1) + 1 \\ &= k \end{aligned}$$

forma explícita correcta de la relación de recurrencia

Análisis de casos: exponenciación (4)

- *Peor caso*: $f_2(2^k-1)=\begin{cases}0 & \text{si } k=1\\ f_2(2^{k-1}-1)+2 & \text{si } k>1 \text{ (2)}\end{cases}$

$k=$	1	$\to f_2($	1	$)=$	0
	2		3		2
	3		7		4
	4		15		6
	5		31		8
	6		63		10
	...				

$\Rightarrow$ Hipótesis de inducción: $\boxed{f_2(2^\alpha-1)=2(\alpha-1)}: 1\leq\alpha\leq k-1$

Paso inductivo: $(2)\to f_2(2^k-1)$

$$\begin{aligned} f_2(2^k-1) &= f_2(2^{k-1}-1)+2\\ &= 2(k-1-1)+2\\ &= 2(k-1)\end{aligned}$$

Análisis de casos: exponenciación (5)

- $n = 2^k$ (mejor caso):
  $f_2(2^k) = k$ para $k \geq 0$
  $\rightarrow f_2(n) = log_2 n$ para $n = 2^k$ y $k \geq 0$ (ya que $log_2 2^k = k$)
  $\Rightarrow \boxed{f_2(n) = \Omega(logn)}$
- $n = 2^k - 1$ (peor caso):
  $f_2(2^k - 1) = 2(k-1)$ para $k \geq 1$
  $\rightarrow f_2(n) = 2[log_2(n+1) - 1]$ para $n = 2^k - 1$ y $k \geq 1$
  $\Rightarrow \boxed{f_2(n) = O(logn)}$
- $\Rightarrow f_2(n) = \Theta(logn)$
  Modelo de computación: operación principal = multiplicación
  $\Rightarrow \boxed{T(n) = \Theta(logn)}$

mejor caso $\leftrightarrow \Omega$

peor caso $\leftrightarrow O$

Reglas para calcular O (1)

1. operación elemental = 1 $\leftrightarrow$ Modelo de Computación

Reglas para calcular O (2)

2. **secuencia**: $S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
$\Rightarrow \boxed{S_1; S_2} = O(f_1(n) + f_2(n)) = O(max(f_1(n), f_2(n)))$

- *También con* Θ

Reglas para calcular O (3)

3. **condición**: $B = O(f_B(n)) \wedge S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
   $\Rightarrow$ $\boxed{\textbf{si } B \textbf{ entonces } S_1 \textbf{ sino } S_2} = O(max(f_B(n), f_1(n), f_2(n)))$

- Si $f_1(n) \neq f_2(n)$ y $max(f_1(n), f_2(n)) > f_B(n) \leftrightarrow$ **Peor caso**
- ¿Caso medio?
  $\rightarrow f(n)$: promedio de f_1 y f_2 ponderado con las frecuencias de cada rama
  $\rightarrow O(max(f_B(n), f(n)))$

Reglas para calcular O (4)

4. **iteración**: $B; S = O(f_{B,S}(n)) \wedge$ nº iter$= O(f_{iter}(n))$

$\Rightarrow$ $\boxed{\textbf{mientras } B \textbf{ hacer } S} = O(f_{B,S}(n) * f_{iter}(n))$
ssi el coste de las iteraciones no varía, sino: $\sum$ costes indiv.

$\Rightarrow$ $\boxed{\textbf{para } i \leftarrow x \textbf{ hasta } y \textbf{ hacer } S} = O(f_S(n) *$nº iter$)$
ssi el coste de las iteraciones no varía, sino: $\sum$ costes indiv.

- B es comparar 2 enteros $= O(1)$; nº iter $= y - x + 1$

Reglas para calcular O (5)

1. operación elemental = 1 $\leftrightarrow$ Modelo de Computación
2. **secuencia**: $S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
   $\Rightarrow \boxed{S_1; S_2} = O(f_1(n) + f_2(n)) = O(max(f_1(n), f_2(n)))$
   - También con Θ
3. **condición**: $B = O(f_B(n)) \wedge S_1 = O(f_1(n)) \wedge S_2 = O(f_2(n))$
   $\Rightarrow$ $\boxed{\textbf{si } B \textbf{ entonces } S_1 \textbf{ sino } S_2} = O(max(f_B(n), f_1(n), f_2(n)))$
   - Si $f_1(n) \neq f_2(n)$ y $max(f_1(n), f_2(n)) > f_B(n) \leftrightarrow$ **Peor caso**
   - ¿Caso medio? $\rightarrow f(n)$: promedio de f_1 y f_2 ponderado con las frecuencias de cada rama $\rightarrow O(max(f_B(n), f(n)))$
4. **iteración**: $B; S = O(f_{B,S}(n)) \wedge$ nº iter$= O(f_{iter}(n))$
   $\Rightarrow$ $\boxed{\textbf{mientras } B \textbf{ hacer } S} = O(f_{B,S}(n) * f_{iter}(n))$
   ssi el coste de las iteraciones no varía, sino: $\sum$ costes indiv.
   $\Rightarrow$ $\boxed{\textbf{para } i \leftarrow x \textbf{ hasta } y \textbf{ hacer } S} = O(f_S(n) *$nº iter$)$
   ssi el coste de las iteraciones no varía, sino: $\sum$ costes indiv.
   - B es comparar 2 enteros $= O(1)$; nº iter $= y - x + 1$

Reglas para calcular O (6)

- Uso de las reglas:
  - análisis “de adentro hacia afuera”
  - analizar primero los subprogramas
  - recursividad: intentar tratarla como un ciclo, sino resolver relación de recurrencia
- **Ejemplo**: $\sum_{i=1}^{n} i^3$

```
    función suma (n:entero) : entero
{1}   s:=0;
{2}   para i:=1 hasta n hacer
{3}        s:=s+i*i*i;
{4}   devolver s
    fin función
```

$\Theta(1)$ en $\{3\}$ y no hay variaciones
$\Rightarrow \Theta(n)$ en $\{2\}$ (regla 4)
$\Rightarrow T(n) = \Theta(n)$ (regla 2)

- *El razonamiento ya incluye las aproximaciones*

Ordenación por Selección (3)

```
    procedimiento Ordenación por Selección (var T[1..n])
{1}     para i:=1 hasta n-1 hacer
{2}         minj:=i; minx:=T[i];
{3}         para j:=i+1 hasta n hacer
{4}             si T[j]<minx entonces
{5}                 minj:=j; minx:=T[j]
                fin si
            fin para;
{6}         T[minj]:=T[i]; T[i]:=minx
        fin para
    fin procedimiento
```

Ordenación por Selección (4)

- $\Theta(1)$ en $\{5\}$ (regla 2)
  $\Rightarrow O(max(\Theta(1), \Theta(1), 0)) = \Theta(1)$ en $\{4\}$
  (regla 3: **no estamos en peor caso**)
- $S = \Theta(1)$; nº iter=$n - i \Rightarrow \Theta(n-i)$ en $\{3\}$ (regla 4)
- $\Theta(1)$ en $\{2\}$ y en $\{6\}$ (regla 2)
  $\Rightarrow \Theta(n-i)$ en $\{$2-6$\}$ (regla 2)
- $S = \Theta(n-i)$ **varía**: $\begin{cases} i = 1 & \rightarrow \Theta(n) \\ i = n-1 & \rightarrow \Theta(1) \end{cases}$

  $\Rightarrow \sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i$ en $\{1\}$ (regla 4)
  $= (n-1)n - \frac{n(n-1)}{2}$: polinomio de grado 2

  $\Rightarrow T(n) = \Theta(n^2)$ **en cualquier caso**