Tema 1. Anàlisi d'algorismes

Estructures de Dades i Algorismes

FIB

Transparències d'Antoni Lozano
(amb edicions d'altres professors)

Q1 2023–2024
Versió de 20 de setembre de 2023

Eficiència dels algorismes

El *software*, a més de ser correcte, hauria de ser eficient.

L'anàlisi de l'eficiència permet:

- Comparar solucions algorísmiques alternatives
- Millorar els algorismes existents
- Predir els recursos que farà servir un algorisme

Eficiència dels algorismes

El *software*, a més de ser correcte, hauria de ser eficient.

L'anàlisi de l'eficiència permet:

- Comparar solucions algorísmiques alternatives
- Millorar els algorismes existents
- Predir els recursos que farà servir un algorisme

Eficiència dels algorismes

El *software*, a més de ser correcte, hauria de ser eficient.

L'anàlisi de l'eficiència permet:

- Comparar solucions algorísmiques alternatives
- Millorar els algorismes existents
- Predir els recursos que farà servir un algorisme

Eficiència dels algorismes

El *software*, a més de ser correcte, hauria de ser eficient.

L'anàlisi de l'eficiència permet:

- Comparar solucions algorísmiques alternatives
- Millorar els algorismes existents
- Predir els recursos que farà servir un algorisme

Eficiència dels algorismes

Consideracions sobre l'eficiència:

- Depèn de la mida de les entrades
- És un concepte relatiu (màquina, compilador, llibreries)
- Aquests factors afecten de forma lineal

Eficiència dels algorismes

Consideracions sobre l'eficiència:

- Depèn de la mida de les entrades
- És un concepte relatiu (màquina, compilador, llibreries)
- Aquests factors afecten de forma lineal

Eficiència dels algorismes

Consideracions sobre l'eficiència:

- Depèn de la mida de les entrades
- És un concepte relatiu (màquina, compilador, llibreries)
- Aquests factors afecten de forma lineal

Exemple 1: el problema de selecció

Problema de selecció

Donada una llista de n naturals, determinar el k-èsim més gran.

Primera solució

Ordenar els nombres en un vector de forma decreixent i retornar el k-èsim.

Segona solució

Escriure els k primers nombres en un vector $V[0 \cdots k-1]$ i ordenar-los decreixentment. Per a cada element restant:

- si és més petit que $V[k-1]$, es descarta
- si no, se situa correctament en V i s'elimina el més petit

Retornar $V[k-1]$.

Exemple 1: el problema de selecció

Problema de selecció

Donada una llista de n naturals, determinar el k-èsim més gran.

Primera solució

Ordenar els nombres en un vector de forma decreixent i retornar el k-èsim.

Segona solució

Escriure els k primers nombres en un vector $V[0 \cdots k-1]$ i ordenar-los decreixentment. Per a cada element restant:

- si és més petit que $V[k-1]$, es descarta
- si no, se situa correctament en V i s'elimina el més petit

Retornar $V[k-1]$.

Exemple 1: el problema de selecció

Problema de selecció

Donada una llista de n naturals, determinar el k-èsim més gran.

Primera solució

Ordenar els nombres en un vector de forma decreixent i retornar el k-èsim.

Segona solució

Escriure els k primers nombres en un vector $V[0 \cdots k-1]$ i ordenar-los decreixentment. Per a cada element restant:

- si és més petit que $V[k-1]$, es descarta
- si no, se situa correctament en V i s'elimina el més petit

Retornar $V[k-1]$.

Exemple 2: el mur infinit

Mur infinit

Estem davant d'un mur que s'allarga indefinidament en totes dues direccions. Volem trobar l'única porta que el travessa, però no sabem a quina distància és ni en quina direcció. Tot i que és fosc, portem una espelma que ens permet veure la porta quan hi som a prop.

Exemple 2: el mur infinit

Primera solució

- Avancem 1 metre i tornem a l'origen
- Retrocedim 2 metres i tornem a l'origen
- Avancem 3 metres i tornem a l'origen
- Retrocedim 4 metres i tornem a l'origen
- ...

Exemple 2: el mur infinit

Segona solució

- Avancem 1 metre i tornem a l’origen
- Retrocedim 2 metres i tornem a l’origen
- Avancem 4 metres i tornem a l’origen
- Retrocedim 8 metres i tornem a l’origen
- ...

Mida de l’entrada i cost

Donat un algorisme *A* amb conjunt d’entrades $\mathcal{E}$, l’eficiència o cost d’*A* es pot expressar com una funció $T : \mathcal{E} \to \mathbb{R}^+$.

Però calcular T per cada entrada pot ser complicat i de poca utilitat. És més útil agrupar les entrades amb la mateixa mida i estudiar el cost sobre aquestes entrades en conjunt.

Mida de l'entrada i cost

Donat un algorisme *A* amb conjunt d'entrades $\mathcal{E}$, l'eficiència o cost d'*A* es pot expressar com una funció $T : \mathcal{E} \to \mathbb{R}^+$.

Però calcular *T* per cada entrada pot ser complicat i de poca utilitat. És més útil agrupar les entrades amb la mateixa mida i estudiar el cost sobre aquestes entrades en conjunt.

Mida de l’entrada i cost

Mida

La mida (o talla) d’una entrada x és el nombre de símbols necessari per codificar-la. Es representa amb $|x|$.

Convencions segons el tipus d’entrada

- Nombres naturals $\longrightarrow$ codificació en binari

$$|27| = 5 \text{ perquè } \langle 27 \rangle_2 = 11011$$

- Llistes, vectors $\longrightarrow$ nombre de components

$$|(23, 1, 7, 0, 12, 500, 2, 11)| = 8$$

Mida de l'entrada i cost

Mida

La mida (o talla) d'una entrada x és el nombre de símbols necessari per codificar-la. Es representa amb $|x|$.

Convencions segons el tipus d'entrada

- Nombres naturals $\longrightarrow$ codificació en binari

$$|27| = 5 \;\text{ perquè }\; \langle 27 \rangle_2 = 11011$$

- Llistes, vectors $\longrightarrow$ nombre de components

$$|(23, 1, 7, 0, 12, 500, 2, 11)| = 8$$

Mida de l'entrada i cost

Exercici

Demostreu que $|\langle x \rangle_2| = \lfloor \log_2 x \rfloor + 1$, on $\langle x \rangle_2$ és la codificació binària de x.

Pista: expresseu x en binari

$$\langle x \rangle_2 = b_{k-1} b_{k-2} \ldots b_0$$

on $b_{k-1} \neq 0$ i calculeu els valors mínim i màxim de x en funció de k.

Notació

A partir d'ara, escriurem $\log$ en lloc de $\log_2$.

Mida de l'entrada i cost

Exercici

Demostreu que $|\langle x \rangle_2| = \lfloor \log_2 x \rfloor + 1$, on $\langle x \rangle_2$ és la codificació binària de x.

Pista: expresseu x en binari

$$\langle x \rangle_2 = b_{k-1} b_{k-2} \dots b_0$$

on $b_{k-1} \neq 0$ i calculeu els valors mínim i màxim de x en funció de k.

Notació

A partir d'ara, escriurem $\log$ en lloc de $\log_2$.

Mida de l'entrada i cost

Suposem que A és un algorisme i $T(x)$ el cost d'executar A amb entrada $x \in \mathcal{E}$. Es defineixen 3 funcions de cost segons la mida de l'entrada:

- **Cas pitjor.** $T_{pitjor}(n) = \max\{T(x) \mid x \in \mathcal{E} \wedge |x| = n\}$
  Dona garanties sobre límits que l'algorisme no superarà.
- **Cas millor.** $T_{millor}(n) = \min\{T(x) \mid x \in \mathcal{E} \wedge |x| = n\}$
  Poc útil.
- **Cas mig.** $T_{mig}(n) = \sum_{x \in A, |x|=n} Pr(x)T(x)$,
  on $Pr(x)$ és la probabilitat de l'ocurrència de l'entrada x en $\mathcal{E}$
  Cal definir la distribució de probabilitat. Sol ser difícil de calcular.

Mida de l'entrada i cost

Suposem que A és un algorisme i $T(x)$ el cost d'executar A amb entrada $x \in \mathcal{E}$. Es defineixen 3 funcions de cost segons la mida de l'entrada:

- **Cas pitjor.** $T_{pitjor}(n) = \max\{T(x) \mid x \in \mathcal{E} \wedge |x| = n\}$
  Dona garanties sobre límits que l'algorisme no superarà.
- **Cas millor.** $T_{millor}(n) = \min\{T(x) \mid x \in \mathcal{E} \wedge |x| = n\}$
  Poc útil.
- **Cas mig.** $T_{mig}(n) = \sum_{x \in A, |x| = n} Pr(x) T(x)$,
  on $Pr(x)$ és la probabilitat de l'ocurrència de l'entrada x en $\mathcal{E}$
  Cal definir la distribució de probabilitat. Sol ser difícil de calcular.

Mida de l'entrada i cost

Suposem que A és un algorisme i $T(x)$ el cost d'executar A amb entrada $x \in \mathcal{E}$. Es defineixen 3 funcions de cost segons la mida de l'entrada:

- **Cas pitjor.** $T_{pitjor}(n) = \max\{T(x) \mid x \in \mathcal{E} \wedge |x| = n\}$
  Dona garanties sobre límits que l'algorisme no superarà.
- **Cas millor.** $T_{millor}(n) = \min\{T(x) \mid x \in \mathcal{E} \wedge |x| = n\}$
  Poc útil.
- **Cas mig.** $T_{mig}(n) = \sum_{x \in A, |x|=n} Pr(x)T(x)$,
  on $Pr(x)$ és la probabilitat de l'ocurrència de l'entrada x en $\mathcal{E}$
  Cal definir la distribució de probabilitat. Sol ser difícil de calcular.

Mida de l’entrada i cost

Taula 1 (Garey/Johnson, *Computers and Intractability*)

Comparació de funcions polinòmiques i exponencials.

cost	10	20	30	40	50
n	0.00001 s	0.00002 s	0.00003 s	0.00004 s	0.00005 s
n^2	0.0001 s	0.0004 s	0.0009 s	0.0016 s	0.0025 s
n^3	0.001 s	0.008 s	0.027 s	0.064 s	0.125 s
n^5	0.1 s	3.2 s	24.3 s	1.7 min	5.2 min
2^n	0.001 s	1.0 s	17.9 min	12.7 dies	35.7 anys
3^n	0.059 s	58 min	6.5 anys	3855 segles	2×10^8 segles

Mida de l'entrada i cost

Taula 2 (Garey/Johnson, *Computers and Intractability*)

Efecte de les millores en la tecnologia sobre algorismes polinòmics i exponencials.

S'indiquen les mides de les entrades processades per unitat de temps

cost	tecnologia actual	tecnologia $\times 100$	tecnologia $\times 1000$
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31.6N_2$
n^3	N_3	$4.64N_3$	$10N_3$
2^n	N_4	$N_4 + 6.64$	$N_4 + 9.97$
3^n	N_5	$N_5 + 4.19$	$N_5 + 6.29$

Ordres de magnitud

Necessitem una notació que:

- permeti donar una fita superior de

$$T_{pitjor}(n) = \max\{T(x) \mid x \in \mathcal{A} \wedge |x| = n\}.$$

  (sabrem que l'algorisme mai superarà la fita)

- que sigui independent dels factors constants

  (així no dependrà de la implementació)

Notació O gran

Donada una funció g, $O(g)$ és la classe de funcions f que "no creixen més de pressa que g". Formalment, $f \in O(g)$ si existeixen $c > 0$ i $n_0 \in \mathbb{N}$ tals que

$$\forall n \geq n_0 \quad f(n) \leq c \cdot g(n).$$

En lloc de $f \in O(g)$, s'escriu sovint "f és $O(g)$" o, també, $f = O(g)$.

Ordres de magnitud

Necessitem una notació que:

- permeti donar una fita superior de

$$T_{pitjor}(n) = \max\{T(x) \mid x \in \mathcal{A} \wedge |x| = n\}.$$

  (sabrem que l'algorisme mai superarà la fita)

- que sigui independent dels factors constants
  (així no dependrà de la implementació)

Notació O gran

Donada una funció g, $O(g)$ és la classe de funcions f que "no creixen més de pressa que g". Formalment, $f \in O(g)$ si existeixen $c > 0$ i $n_0 \in \mathbb{N}$ tals que

$$\forall n \geq n_0 \quad f(n) \leq c \cdot g(n).$$

En lloc de $f \in O(g)$, s'escriu sovint "f és $O(g)$" o, també, $f = O(g)$.

Ordres de magnitud

Necessitem una notació que:

- permeti donar una fita superior de

$$T_{pitjor}(n) = \max\{T(x) \mid x \in \mathcal{A} \wedge |x| = n\}.$$

  (sabrem que l'algorisme mai superarà la fita)

- que sigui independent dels factors constants
  (així no dependrà de la implementació)

Notació O gran

Donada una funció g, $O(g)$ és la classe de funcions f que "no creixen més de pressa que g". Formalment, $f \in O(g)$ si existeixen $c > 0$ i $n_0 \in \mathbb{N}$ tals que

$$\forall n \geq n_0 \quad f(n) \leq c \cdot g(n).$$

En lloc de $f \in O(g)$, s'escriu sovint "f és $O(g)$" o, també, $f = O(g)$.

Ordres de magnitud

Necessitem una notació que:

- permeti donar una fita superior de

$$T_{pitjor}(n) = \max\{T(x) \mid x \in \mathcal{A} \wedge |x| = n\}.$$

  (sabrem que l'algorisme mai superarà la fita)

- que sigui independent dels factors constants
  (així no dependrà de la implementació)

Notació O gran

Donada una funció g, $O(g)$ és la classe de funcions f que "no creixen més de pressa que g". Formalment, $f \in O(g)$ si existeixen $c > 0$ i $n_0 \in \mathbb{N}$ tals que

$$\forall n \geq n_0 \quad f(n) \leq c \cdot g(n).$$

En lloc de $f \in O(g)$, s'escriu sovint "f és $O(g)$" o, també, $f = O(g)$.

Ordres de magnitud

Exemple

Sigui $f(n) = 3n^3 + 5n^2 - 7n + 41$. Llavors, podem afirmar que $f \in O(n^3)$.

Per justificar-ho, només cal trobar constants c i n_0 tals que

$$\forall n \geq n_0 \;\; f(n) \leq cn^3.$$

Però $3n^3 + 5n^2 - 7n + 41 \leq 8n^3 + 41$. Triem $c = 9$. Llavors,

$$8n^3 + 41 \leq 9n^3 \iff 41 \leq n^3,$$

que es compleix a partir de $n_0 = 4$. Per tant, $\forall n \geq 4 \quad f(n) \leq 9n^3$ i, llavors, $f(n) = O(n^3)$ amb $c = 9$ i $n_0 = 4$.

Exercici

Trobeu una constant n_0 que, juntament amb $c = 4$, demostri que $f \in O(n^3)$ per a la funció f de l'exemple.

Ordres de magnitud

Exemple

Sigui $f(n) = 3n^3 + 5n^2 - 7n + 41$. Llavors, podem afirmar que $f \in O(n^3)$.

Per justificar-ho, només cal trobar constants c i n_0 tals que

$$\forall n \geq n_0 \;\; f(n) \leq cn^3.$$

Però $3n^3 + 5n^2 - 7n + 41 \leq 8n^3 + 41$. Triem $c = 9$. Llavors,

$$8n^3 + 41 \leq 9n^3 \iff 41 \leq n^3,$$

que es compleix a partir de $n_0 = 4$. Per tant, $\forall n \geq 4 \quad f(n) \leq 9n^3$ i, llavors, $f(n) = O(n^3)$ amb $c = 9$ i $n_0 = 4$.

Exercici

Trobeu una constant n_0 que, juntament amb $c = 4$, demostri que $f \in O(n^3)$ per a la funció f de l'exemple.

Ordres de magnitud

Exemple

Sigui $f(n) = 3n^3 + 5n^2 - 7n + 41$. Llavors, podem afirmar que $f \in O(n^3)$.

Per justificar-ho, només cal trobar constants c i n_0 tals que

$$\forall n \geq n_0 \;\; f(n) \leq cn^3.$$

Però $3n^3 + 5n^2 - 7n + 41 \leq 8n^3 + 41$. Triem $c = 9$. Llavors,

$$8n^3 + 41 \leq 9n^3 \iff 41 \leq n^3,$$

que es compleix a partir de $n_0 = 4$. Per tant, $\forall n \geq 4 \quad f(n) \leq 9n^3$ i, llavors, $f(n) = O(n^3)$ amb $c = 9$ i $n_0 = 4$.

Exercici

Trobeu una constant n_0 que, juntament amb $c = 4$, demostri que $f \in O(n^3)$ per a la funció f de l'exemple.

Exemple 1: el problema de selecció

Problema de selecció

Donada una llista de n naturals, determinar el k-èsim més gran.

Primera solució

Ordenar els nombres en un vector de forma decreixent i retornar el k-èsim.

Segona solució

Escriure els k primers nombres en un vector $V[0 \cdots k-1]$ i ordenar-los decreixentment. Per a cada element restant:

- si és més petit que $V[k-1]$, es descarta
- si no, se situa correctament en V i s'elimina el més petit

Retornar $V[k-1]$.

Exemple 1: el problema de selecció

Problema de selecció

Donada una llista de n naturals, determinar el k-èsim més gran.

Primera solució

Ordenar els nombres en un vector de forma decreixent i retornar el k-èsim.

Segona solució

Escriure els k primers nombres en un vector $V[0 \cdots k-1]$ i ordenar-los decreixentment. Per a cada element restant:

- si és més petit que $V[k-1]$, es descarta
- si no, se situa correctament en V i s'elimina el més petit

Retornar $V[k-1]$.

Exemple 1: el problema de selecció

Problema de selecció

Donada una llista de n naturals, determinar el k-èsim més gran.

Primera solució

Ordenar els nombres en un vector de forma decreixent i retornar el k-èsim.

Segona solució

Escriure els k primers nombres en un vector $V[0 \cdots k-1]$ i ordenar-los decreixentment. Per a cada element restant:

- si és més petit que $V[k-1]$, es descarta
- si no, se situa correctament en V i s'elimina el més petit

Retornar $V[k-1]$.

Exemple 1: el problema de selecció

Problema de selecció

Donada una llista de n naturals, determinar el k-èsim més gran.

Primera solució

Ordenar els nombres en un vector de forma decreixent i retornar el k-èsim.

- amb un algorisme d'ordenació bàsic (bombolla, inserció): $O(n^2)$
- amb un algorisme d'ordenació eficient: $O(n \log n)$

Exemple 1: el problema de selecció

Problema de selecció

Donada una llista de n naturals, determinar el k-èsim més gran.

Segona solució

Escriure els k primers nombres en un vector $V[0 \cdots k-1]$ i ordenar-los decreixentment. Per a cada element restant:

- si és més petit que $V[k-1]$, es descarta
- si no, se situa correctament en V i s'elimina el més petit

Retornar $V[k-1]$.

$O((k \log k) + (n-k) \cdot k)$

- Si k és constant, és $O(k \cdot n) = O(n)$
- Si $k = \lceil n/2 \rceil$, és $O(\frac{n}{2} \cdot \frac{n}{2}) = O(n^2)$

Exemple 1: el problema de selecció

Problema de selecció

Donada una llista de n naturals, determinar el k-èsim més gran.

Exercici

- Proposa una tercera solució del problema de selecció consistent en repetir k vegades una certa acció sobre la llista de naturals.
- Dona una fita superior del seu cost.

Exemple 2: el mur infinit

Mur infinit

Estem davant d'un mur que s'allarga indefinidament en totes dues direccions. Volem trobar l'única porta que el travessa, però no sabem a quina distància està ni en quina direcció. Tot i que és fosc, portem una espelma que ens permet veure la porta quan hi som a prop.

Exemple 2: el mur infinit

Primera solució

- Avancem 1 metre i tornem a l'origen
- Retrocedim 2 metres i tornem a l'origen
- Avancem 3 metres i tornem a l'origen
- Retrocedim 4 metres i tornem a l'origen
- ...

Temps quan la porta és a distància n:

$$T(n) = 2\sum_{i=1}^{n-1} i + n = 2\frac{(n-1)n}{2} + n = n^2 \in O(n^2).$$

$$\left(\text{recordem que } \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}\right)$$

Exemple 2: el mur infinit

Segona solució

- Avancem 1 metre i tornem a l'origen
- Retrocedim 2 metres i tornem a l'origen
- Avancem 4 metres i tornem a l'origen
- Retrocedim 8 metres i tornem a l'origen
- ...

Si la porta és a distància $n = 2^k$, aleshores

$$T(n) = 2\sum_{i=0}^{k-1} 2^i + 2^k = 2(2^k - 1) + 2^k = 3n - 2 \in O(n).$$

$$\left(\text{recordem que } \sum_{i=0}^{k-1} 2^i = 2^k - 1\right)$$

Notació asimptòtica: definicions

- La notació asimptòtica permet classificar les funcions d'acord amb la seva taxa relativa de creixement.
- Té en compte el comportament de les funcions per a entrades grans. Per exemple, $n^2 \geq 10^6 n$ a partir d'un cert valor de n:

$$n^2 \geq 10^6 n \iff n \geq 10^6.$$

  Per a $n \geq 10^6$, doncs, n^2 creix més de pressa que $10^6 n$. En aquest cas, diem que la funció $f(n) = 10^6 n$ està fitada per $g(n) = n^2$ asimptòticament.
- La notació $O(g)$, anomenada "O gran", representa el conjunt de funcions fitades asimptòticament per g.

Notació asimptòtica: definicions

Notació Θ ((a): fita exacta asimptòtica)

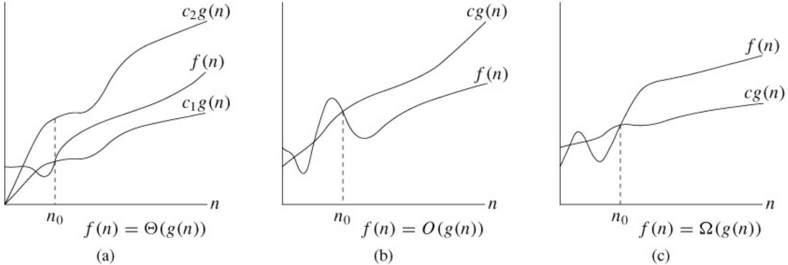
$$\Theta(g) = \{f : \mathbb{N} \to \mathbb{R}^+ \mid \exists c_1, c_2 \in \mathbb{R}^+ \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 \; c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Notació O gran ((b): fita superior asimptòtica)

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

Notació Ω ((c): fita inferior asimptòtica)

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



Notació Θ

Notació Θ (fita exacta asimptòtica)

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

Exemples

- $75n \in \Theta(n)$
- $1023n^2 \notin \Theta(n)$
- $n^2 \notin \Theta(n)$
- $2^n \notin \Theta(2^{n^2})$
- $\Theta(n) \neq \Theta(n^2)$

Exercici

Demostreu que $2^n \notin \Theta(2^{n^2})$.

Notació Θ

Notació Θ (fita exacta asimptòtica)

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

Exemples

- $75n \in \Theta(n)$
- $1023n^2 \notin \Theta(n)$
- $n^2 \notin \Theta(n)$
- $2^n \notin \Theta(2^{n^2})$
- $\Theta(n) \neq \Theta(n^2)$

Exercici

Demostreu que $2^n \notin \Theta(2^{n^2})$.

Notació O gran

Notació O gran (fita superior asimptòtica)

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

Exemples

- $7n^2 + 5n - 7 \in O(n^2)$
- $n + 15 \in O(n)$
- $O(n^5) \subseteq O(n^6)$
- $n^3 \notin O(n^2)$
- $n^3 \in O(2^n)$

Exercici

Demostreu que $p(n) = 7n^2 + 4n - 2$ és $O(n^2)$.

Notació O gran

Notació O gran (fita superior asimptòtica)

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

Exemples

- $7n^2 + 5n - 7 \in O(n^2)$
- $n + 15 \in O(n)$
- $O(n^5) \subseteq O(n^6)$
- $n^3 \notin O(n^2)$
- $n^3 \in O(2^n)$

Exercici

Demostreu que $p(n) = 7n^2 + 4n - 2$ és $O(n^2)$.

Notació Ω

Notació Ω (fita inferior asimptòtica)

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

Exemples

- $2^n \in \Omega(n)$
- $n^2 - n \in \Omega(n)$
- $n \in \Omega(n)$
- $n \notin \Omega(n^2)$
- $\Omega(n^6) \subseteq \Omega(n^5)$

Exercici

Demostreu que $n^2 - n$ és $\Omega(n)$.

Notació Ω

Notació Ω (fita inferior asimptòtica)

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

Exemples

- $2^n \in \Omega(n)$
- $n^2 - n \in \Omega(n)$
- $n \in \Omega(n)$
- $n \notin \Omega(n^2)$
- $\Omega(n^6) \subseteq \Omega(n^5)$

Exercici

Demostreu que $n^2 - n$ és $\Omega(n)$.

Notació asimptòtica: propietats

Relacions entre O, Ω i Θ

Donades dues funcions f i g:

- $f \in \Omega(g) \iff g \in O(f)$
- $\Theta(f) = O(f) \cap \Omega(f)$
- $O(f) = O(g) \iff \Omega(f) = \Omega(g) \iff \Theta(f) = \Theta(g)$

Notació asimptòtica: propietats

Regla del límit

- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \;\Rightarrow\; f \in O(g)$ però $g \notin O(f)$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \;\Rightarrow\; g \in O(f)$ però $f \notin O(g)$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c,$ on $0 < c < \infty \;\Rightarrow\; O(f) = O(g)$

Exercicis

Siguin $k \geq 1$ i $c > 1$. Demostreu:

1. $n^k \in O(c^n)$ i $n^k \notin \Omega(c^n)$
2. $\log^k n \in O(n)$

Notació asimptòtica: propietats

Regla del límit

- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \;\Rightarrow\; f \in O(g)$ però $g \notin O(f)$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \;\Rightarrow\; g \in O(f)$ però $f \notin O(g)$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, on $0 < c < \infty \;\Rightarrow\; O(f) = O(g)$

Exercicis

Siguin $k \geq 1$ i $c > 1$. Demostreu:

1. $n^k \in O(c^n)$ i $n^k \notin \Omega(c^n)$
2. $\log^k n \in O(n)$

Notació asimptòtica: propietats

Propietats de l'O gran

Donades les funcions f, f_1, f_2, g, g_1, g_2 i h:

- *Reflexivitat.* $f \in O(f)$
- *Transitivitat.* $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$
- *Caracterització.* $f \in O(g) \iff O(f) \subseteq O(g)$
- *Regla de la suma.* $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(\max(g_1, g_2))$
- *Regla del producte.* $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \Rightarrow f_1 \cdot f_2 \in O(g_1 \cdot g_2)$
- *Invariança multiplicativa.* Per a tota constant $c \in \mathbb{R}^+$, $O(f) = O(c \cdot f)$

Notació asimptòtica: propietats

Propietats de l'O gran

Donades les funcions f, f_1, f_2, g, g_1, g_2 i h:

- *Reflexivitat.* $f \in O(f)$
- *Transitivitat.* $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$
- *Caracterització.* $f \in O(g) \iff O(f) \subseteq O(g)$
- *Regla de la suma.* $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(\max(g_1, g_2))$
- *Regla del producte.* $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \Rightarrow f_1 \cdot f_2 \in O(g_1 \cdot g_2)$
- *Invariança multiplicativa.* Per a tota constant $c \in \mathbb{R}^+$, $O(f) = O(c \cdot f)$

Notació asimptòtica: propietats

Propietats de l'O gran

Donades les funcions f, f_1, f_2, g, g_1, g_2 i h:

- *Reflexivitat.* $f \in O(f)$
- *Transitivitat.* $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$
- *Caracterització.* $f \in O(g) \iff O(f) \subseteq O(g)$
- *Regla de la suma.* $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(\max(g_1, g_2))$
- *Regla del producte.* $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \Rightarrow f_1 \cdot f_2 \in O(g_1 \cdot g_2)$
- *Invariança multiplicativa.* Per a tota constant $c \in \mathbb{R}^+$, $O(f) = O(c \cdot f)$

Notació asimptòtica: propietats

Propietats de l'O gran

Donades les funcions f, f_1, f_2, g, g_1, g_2 i h:

- *Reflexivitat*. $f \in O(f)$
- *Transitivitat*. $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$
- *Caracterització*. $f \in O(g) \iff O(f) \subseteq O(g)$
- *Regla de la suma*. $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \Rightarrow f_1 + f_2 \in O(\max(g_1, g_2))$
- *Regla del producte*. $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \Rightarrow f_1 \cdot f_2 \in O(g_1 \cdot g_2)$
- *Invariança multiplicativa*. Per a tota constant $c \in \mathbb{R}^+$, $O(f) = O(c \cdot f)$

Notació asimptòtica: propietats

Exercici

Feu servir la regla del límit per demostrar la transitivitat de l'O gran, és a dir, que si f, g, h són funcions, llavors $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$.

Suposant que $f \in O(g)$ i $g \in O(h)$, tenim que

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \ \wedge \ \lim_{n\to\infty} \frac{g(n)}{h(n)} < \infty.$$

Aleshores,

$$\lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} \frac{f(n) \cdot g(n)}{g(n) \cdot h(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} \cdot \lim_{n\to\infty} \frac{g(n)}{h(n)} < \infty$$

i, per tant, $f \in O(h)$.

Exercici

Feu servir la regla del límit per demostrar les altres propietats de l'O gran.

Notació asimptòtica: propietats

Exercici

Feu servir la regla del límit per demostrar la transitivitat de l'O gran, és a dir, que si f, g, h són funcions, llavors $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$.

Suposant que $f \in O(g)$ i $g \in O(h)$, tenim que

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \;\wedge\; \lim_{n\to\infty} \frac{g(n)}{h(n)} < \infty.$$

Aleshores,

$$\lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} \frac{f(n)\cdot g(n)}{g(n)\cdot h(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} \cdot \lim_{n\to\infty} \frac{g(n)}{h(n)} < \infty$$

i, per tant, $f \in O(h)$.

Exercici

Feu servir la regla del límit per demostrar les altres propietats de l'O gran.

Notació asimptòtica: propietats

Exercici

Feu servir la regla del límit per demostrar la transitivitat de l'O gran, és a dir, que si f, g, h són funcions, llavors $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$.

Suposant que $f \in O(g)$ i $g \in O(h)$, tenim que

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \ \wedge \ \lim_{n\to\infty} \frac{g(n)}{h(n)} < \infty.$$

Aleshores,

$$\lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} \frac{f(n)\cdot g(n)}{g(n)\cdot h(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} \cdot \lim_{n\to\infty} \frac{g(n)}{h(n)} < \infty$$

i, per tant, $f \in O(h)$.

Exercici

Feu servir la regla del límit per demostrar les altres propietats de l'O gran.

Notació asimptòtica: propietats

Exercici

Argumenteu per què l'afirmació $f \in O(g)$ és equivalent a

$$\exists c \in \mathbb{R}^+ \ \overset{\infty}{\forall} n \quad f(n) \leq c \cdot g(n).$$

Recordeu que, per definició, $f \in O(g)$ si

$$\exists c \in \mathbb{R}^+ \ \exists n_0 \in \mathbb{N} \ \ \forall n \geq n_0 \quad f(n) \leq c \cdot g(n).$$

Nota

La notació $\overset{\infty}{\forall} n \ P(n)$ representa que $P(n)$ es compleix per a tots els valors de n excepte per a un nombre finit.

Notació asimptòtica: propietats

Propietats de Θ

Donades les funcions f, f_1, f_2, g, g_1, g_2 i h:

- *Reflexivitat*. $f \in \Theta(f)$
- *Transitivitat*. $f \in \Theta(g) \wedge g \in \Theta(h) \Rightarrow f \in \Theta(h)$
- *Simetria*. $f \in \Theta(g) \iff g \in \Theta(f) \iff \Theta(f) = \Theta(g)$
- *Regla de la suma*. $f_1 \in \Theta(g_1) \wedge f_2 \in \Theta(g_2) \Rightarrow f_1 + f_2 \in \Theta(\max(g_1, g_2))$
- *Regla del producte*. $f_1 \in \Theta(g_1) \wedge f_2 \in \Theta(g_2) \Rightarrow f_1 \cdot f_2 \in \Theta(g_1 \cdot g_2)$
- *Invariança multiplicativa*. Per a tota constant $c \in \mathbb{R}^+$, $\Theta(f) = \Theta(c \cdot f)$

Notació asimptòtica: propietats

Notació de classes

Si $\mathcal{F}_1$ i $\mathcal{F}_2$ són classes de funcions (com ara $O(f)$ o $\Omega(f)$), definim:

- $\mathcal{F}_1 + \mathcal{F}_2 = \{f + g \mid f \in \mathcal{F}_1 \wedge g \in \mathcal{F}_2\}$
- $\mathcal{F}_1 \cdot \mathcal{F}_2 = \{f \cdot g \mid f \in \mathcal{F}_1 \wedge g \in \mathcal{F}_2\}$

Regles de la suma i el producte (segona versió)

Donades dues funcions f i g:

- $O(f) + O(g) = O(f + g) = O(\max\{f, g\})$
- $O(f) \cdot O(g) = O(f \cdot g)$
- $\Theta(f) + \Theta(g) = \Theta(f + g) = \Theta(\max\{f, g\})$
- $\Theta(f) \cdot \Theta(g) = \Theta(f \cdot g)$

Notació asimptòtica: propietats

Notació de classes

Si $\mathcal{F}_1$ i $\mathcal{F}_2$ són classes de funcions (com ara $O(f)$ o $\Omega(f)$), definim:

- $\mathcal{F}_1 + \mathcal{F}_2 = \{f + g \mid f \in \mathcal{F}_1 \wedge g \in \mathcal{F}_2\}$
- $\mathcal{F}_1 \cdot \mathcal{F}_2 = \{f \cdot g \mid f \in \mathcal{F}_1 \wedge g \in \mathcal{F}_2\}$

Regles de la suma i el producte (segona versió)

Donades dues funcions f i g:

- $O(f) + O(g) = O(f + g) = O(\max\{f, g\})$
- $O(f) \cdot O(g) = O(f \cdot g)$
- $\Theta(f) + \Theta(g) = \Theta(f + g) = \Theta(\max\{f, g\})$
- $\Theta(f) \cdot \Theta(g) = \Theta(f \cdot g)$

Formes de creixement

Costos freqüents

- Constant: $\Theta(1)$
  - Decidir la paritat
  - Sumar dues variables numèriques
- Logarítmic: $\Theta(\log n)$
  - Cerca binària
- Lineal: $\Theta(n)$
  - Recorregut seqüencial
    (p. ex., calcular el màxim, el mínim, la mitjana)
- Quasilineal: $\Theta(n \log n)$
  - Ordenacions per fusió (*Mergesort*) i ràpida (*Quicksort*) en cas mitjà

Formes de creixement

Costos freqüents

- Constant: $\Theta(1)$
  - Decidir la paritat
  - Sumar dues variables numèriques
- Logarítmic: $\Theta(\log n)$
  - Cerca binària
- Lineal: $\Theta(n)$
  - Recorregut seqüencial
    (p. ex., calcular el màxim, el mínim, la mitjana)
- Quasilineal: $\Theta(n \log n)$
  - Ordenacions per fusió (*Mergesort*) i ràpida (*Quicksort*) en cas mitjà

Formes de creixement

Costos freqüents

- Constant: $\Theta(1)$
  - Decidir la paritat
  - Sumar dues variables numèriques
- Logarítmic: $\Theta(\log n)$
  - Cerca binària
- Lineal: $\Theta(n)$
  - Recorregut seqüencial
    (p. ex., calcular el màxim, el mínim, la mitjana)
- Quasilineal: $\Theta(n \log n)$
  - Ordenacions per fusió (*Mergesort*) i ràpida (*Quicksort*) en cas mitjà

Formes de creixement

Costos freqüents

- Constant: $\Theta(1)$
  - Decidir la paritat
  - Sumar dues variables numèriques
- Logarítmic: $\Theta(\log n)$
  - Cerca binària
- Lineal: $\Theta(n)$
  - Recorregut seqüencial
    (p. ex., calcular el màxim, el mínim, la mitjana)
- Quasilineal: $\Theta(n \log n)$
  - Ordenacions per fusió (*Mergesort*) i ràpida (*Quicksort*) en cas mitjà

Formes de creixement

Costos freqüents

- Quadràtic: $\Theta(n^2)$
  - Suma de dues matrius quadrades
  - Ordenació per selecció i bombolla
- Cúbic: $\Theta(n^3)$
  - Producte de dues matrius quadrades
  - Enumeració de triples
- Polinòmic: $\Theta(n^k)$, per a $k \geq 1$ constant
  - Enumerar combinacions
  - Test de primalitat
    (amb variants de l'algorisme AKS que van de $\Theta(n^{12})$ a $\Theta(n^6)$)
- Exponencial: $\Theta(k^n)$, per a $k > 1$ constant
  - Cerca en un espai de configuracions (d'amplada k i profunditat n)
- Altres funcions: $\Theta(\sqrt{n})$, $\Theta(n!)$, $\Theta(n^n)$

Formes de creixement

Costos freqüents

- Quadràtic: $\Theta(n^2)$
  - Suma de dues matrius quadrades
  - Ordenació per selecció i bombolla
- Cúbic: $\Theta(n^3)$
  - Producte de dues matrius quadrades
  - Enumeració de triples
- Polinòmic: $\Theta(n^k)$, per a $k \geq 1$ constant
  - Enumerar combinacions
  - Test de primalitat (amb variants de l'algorisme AKS que van de $\Theta(n^{12})$ a $\Theta(n^6)$)
- Exponencial: $\Theta(k^n)$, per a $k > 1$ constant
  - Cerca en un espai de configuracions (d'amplada k i profunditat n)
- Altres funcions: $\Theta(\sqrt{n})$, $\Theta(n!)$, $\Theta(n^n)$

Formes de creixement

Costos freqüents

- Quadràtic: $\Theta(n^2)$
  - Suma de dues matrius quadrades
  - Ordenació per selecció i bombolla
- Cúbic: $\Theta(n^3)$
  - Producte de dues matrius quadrades
  - Enumeració de triples
- Polinòmic: $\Theta(n^k)$, per a $k \geq 1$ constant
  - Enumerar combinacions
  - Test de primalitat
    (amb variants de l'algorisme AKS que van de $\Theta(n^{12})$ a $\Theta(n^6)$)
- Exponencial: $\Theta(k^n)$, per a $k > 1$ constant
  - Cerca en un espai de configuracions (d'amplada k i profunditat n)
- Altres funcions: $\Theta(\sqrt{n})$, $\Theta(n!)$, $\Theta(n^n)$

Formes de creixement

Costos freqüents

- Quadràtic: $\Theta(n^2)$
  - Suma de dues matrius quadrades
  - Ordenació per selecció i bombolla
- Cúbic: $\Theta(n^3)$
  - Producte de dues matrius quadrades
  - Enumeració de triples
- Polinòmic: $\Theta(n^k)$, per a $k \geq 1$ constant
  - Enumerar combinacions
  - Test de primalitat
    (amb variants de l'algorisme AKS que van de $\Theta(n^{12})$ a $\Theta(n^6)$)
- Exponencial: $\Theta(k^n)$, per a $k > 1$ constant
  - Cerca en un espai de configuracions (d'amplada k i profunditat n)
- Altres funcions: $\Theta(\sqrt{n})$, $\Theta(n!)$, $\Theta(n^n)$

Formes de creixement

Costos freqüents

- Quadràtic: $\Theta(n^2)$
  - Suma de dues matrius quadrades
  - Ordenació per selecció i bombolla
- Cúbic: $\Theta(n^3)$
  - Producte de dues matrius quadrades
  - Enumeració de triples
- Polinòmic: $\Theta(n^k)$, per a $k \geq 1$ constant
  - Enumerar combinacions
  - Test de primalitat
    (amb variants de l'algorisme AKS que van de $\Theta(n^{12})$ a $\Theta(n^6)$)
- Exponencial: $\Theta(k^n)$, per a $k > 1$ constant
  - Cerca en un espai de configuracions (d'amplada k i profunditat n)
- Altres funcions: $\Theta(\sqrt{n})$, $\Theta(n!)$, $\Theta(n^n)$

Formes de creixement

Notació

Donades dues funcions f i g, escrivim $f \prec g$ per indicar que $f \in O(g)$ però $g \notin O(f)$.

Exercici

Trobeu dos costos f, g de l'escala anterior per als quals $f \prec \sqrt{n} \prec g$.

Solució

Triem $f(n) = \log n$ i $g(n) = n$ i apliquem la regla del límit:

1. $\log n \prec \sqrt{n}$. Per la regla de L'Hôpital,

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{1/(\ln 2 \cdot n)}{1/2 \cdot n^{-1/2}} = \frac{2}{\ln 2} \cdot \lim_{n\to\infty} \frac{n^{1/2}}{n} = \frac{2}{\ln 2} \cdot \lim_{n\to\infty} \frac{1}{n^{1/2}} = 0.$$

2. $\sqrt{n} \prec n$. Trivialment, $\lim_{n\to\infty} \frac{\sqrt{n}}{n} = \lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$.

Formes de creixement

Notació

Donades dues funcions f i g, escrivim $f \prec g$ per indicar que $f \in O(g)$ però $g \notin O(f)$.

Exercici

Trobeu dos costos f, g de l'escala anterior per als quals $f \prec \sqrt{n} \prec g$.

Solució

Triem $f(n) = \log n$ i $g(n) = n$ i apliquem la regla del límit:

1. $\log n \prec \sqrt{n}$. Per la regla de L'Hôpital,

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{1/(\ln 2 \cdot n)}{1/2 \cdot n^{-1/2}} = \frac{2}{\ln 2} \cdot \lim_{n\to\infty} \frac{n^{1/2}}{n} = \frac{2}{\ln 2} \cdot \lim_{n\to\infty} \frac{1}{n^{1/2}} = 0.$$

2. $\sqrt{n} \prec n$. Trivialment, $\lim_{n\to\infty} \frac{\sqrt{n}}{n} = \lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$.

Càlcul del cost:

- El cost d'una operació elemental és $\Theta(1)$. Això inclou:
  - una assignació entre tipus bàsics (`int`, `bool`, `double`,...)
  - una lectura o escriptura d'un tipus bàsic
  - una comparació
  - una operació aritmètica
  - l'accés a un component d'un vector
  - el pas d'un paràmetre per referència
- Avaluar una expressió té cost igual a la suma dels costos de les operacions que s'hi fan (incloses les crides a les funcions, si n'hi ha).
- El cost de construir o copiar un vector de mida n (assignació, pas per valor, return) és $\Theta(n)$.

Càlcul del cost:

- Si el cost d'un fragment F_1 és C_1 i el d'un fragment F_2 és C_2, llavors el cost de la composició seqüencial

$$F_1; F_2;$$

és $C_1 + C_2$.

En general, si N és constant i el fragment F_k té cost C_k, el cost de la composició seqüencial

$$F_1; F_2; \ldots; F_k;$$

és $C_1 + C_2 + \cdots + C_N$.

Càlcul del cost:

- Si el cost d'un fragment F és C i el cost d'avaluar B és D, llavors el cost de la composició alternativa d'una branca

  $$\texttt{if } (B)\ F;$$

  és $\leq D + C$.

- Si el cost d'un fragment F_1 és C_1, el d'un fragment F_2 és C_2 i el d'avaluar B és D, llavors el cost de la composició alternativa de dues branques

  $$\texttt{if } (B)\ F_1;\ \texttt{else } F_2;$$

  és $\leq D + \max(C_1, C_2)$.

Algorismes iteratius

Càlcul del cost:

- Si el cost d'un fragment F és C i el cost d'avaluar B és D, llavors el cost de la composició alternativa d'una branca

  $$\texttt{if } (B)\ F;$$

  és $\leq D + C$.

- Si el cost d'un fragment F_1 és C_1, el d'un fragment F_2 és C_2 i el d'avaluar B és D, llavors el cost de la composició alternativa de dues branques

  $$\texttt{if } (B)\ F_1;\ \texttt{else }\ F_2;$$

  és $\leq D + \max(C_1, C_2)$.

Algorismes iteratius

Càlcul del cost:

- Si el cost de F durant la k-èsima iteració és C_k, el d'avaluar B és D_k i el nombre d'iteracions és N, llavors el cost de la composició iterativa

  `while` (B) F;

  és $\left(\sum_{k=1}^{N} C_k + D_k\right) + D_{N+1}$.

Algorismes iteratius

Exemple d'ordenació per selecció

Passos per ordenar la seqüència $5, 6, 1, 2, 0, 7, 4, 3$ segons l'algorisme de selecció. En vermell, els elements ja ordenats. En blau, els elements intercanviats pel màxim.

5	6	1	2	0	7	4	3
5	6	1	2	0	3	4	7
5	4	1	2	0	3	6	7
3	4	1	2	0	5	6	7
3	0	1	2	4	5	6	7
2	0	1	3	4	5	6	7
1	0	2	3	4	5	6	7
0	1	2	3	4	5	6	7

Algorismes iteratius

Ordenació per selecció

```
0 int posicio_maxim(const vector<int>& v, int m) {
1   int k = 0;
2   for (int i = 1; i <= m; ++i)
3      if (v[i] > v[k]) k = i;
4   return k; }

5 void ordena_seleccio (vector<int>& v, int n) {
6     for (int i = n-1; i > 0; --i) {
7         int k = posicio_maxim(v,i);
8         swap(v[k],v[i]); }}}
```

- **2, 6** Iteracions bucles: $m-1+1 = m \in \Theta(m)$, $(n-1)-1+1 = n-1 \in \Theta(n)$.
- **7** Cost $\Theta(i)$.
- **altres** Instruccions de cost constant: $\Theta(1)$.

$$t_{sel}(n) = \Theta(1) + \sum_{i=1}^{n-1}(\Theta(i)+\Theta(1)) = \Theta(\sum_{i=1}^{n-1} i) = \Theta(\frac{(n-1)n}{2}) = \Theta(n^2)$$

Exemple d'ordenació per inserció

Passos per ordenar la seqüència 5, 6, 1, 2, 0,7,4,3 segons l'algorisme d'inserció. En vermell, els elements ja ordenats. En blau, el nombre de posicions que s'ha desplaçat l'element inserit.

5	6	1	2	0	7	4	3	(0)
5	6	1	2	0	7	4	3	(0)
1	5	6	2	0	7	4	3	(2)
1	2	5	6	0	7	4	3	(2)
0	1	2	5	6	7	4	3	(4)
0	1	2	5	6	7	4	3	(0)
0	1	2	4	5	6	7	3	(3)
0	1	2	3	4	5	6	7	(4)

Algorismes iteratius

Ordenació per inserció

```
0 void ordena_inserció(vector<int>& v,int n) {
1     for (int k = 1; k <= n-1; ++k) {
2         int t = k-1;
3         while (t >= 0 and v[t+1] < v[t]) {
4             swap(v[t], v[t+1]);
5             --t;  }}}
```

- 0 Pas de paràmetres: $\Theta(1)$.
- 1 Iteracions bucle: $(n-1)-1+1 = n-1 \in \Theta(n)$.
- 1,2 Condició d'iteració i línia 2: $\Theta(1)$.
- 3 Iteracions bucle: entre $0 \in \Theta(1)$ i $k-1-0+1 = k \in \Theta(k)$.
- 4,5 Assignacions amb cost $\Theta(1)$.

$$\Theta(1) + (\Theta(n) \times \Theta(1)) \leq t_{ins}(n) \leq \Theta(1) + \sum_{k=1}^{n-1} \Theta(k)$$

Algorismes iteratius

Hem vist que el cost d'ordenar per inserció n elements és $t_{ins}(n)$, on

$$\Theta(1) + (\Theta(n) \times \Theta(1)) \le t_{ins}(n) \le \Theta(1) + \sum_{k=1}^{n-1} \Theta(k).$$

Però sabem que

$$\sum_{k=1}^{n-1} k = 1 + 2 + \cdots + (n-1) = \frac{(n-1)n}{2}.$$

Aleshores,

$$\sum_{k=1}^{n-1} \Theta(k) = \Theta\Big(\sum_{k=1}^{n-1} k\Big) = \Theta\Big(\frac{(n-1)n}{2}\Big) = \Theta(n^2)$$

i, per tant,

$$\Theta(n) \le t_{ins}(n) \le \Theta(n^2).$$

Algorismes iteratius

Hem vist que el cost d'ordenar per inserció n elements és $t_{ins}(n)$, on

$$\Theta(1) + (\Theta(n) \times \Theta(1)) \leq t_{ins}(n) \leq \Theta(1) + \sum_{k=1}^{n-1} \Theta(k).$$

Però sabem que

$$\sum_{k=1}^{n-1} k = 1 + 2 + \cdots + (n-1) = \frac{(n-1)n}{2}.$$

Aleshores,

$$\sum_{k=1}^{n-1} \Theta(k) = \Theta\left(\sum_{k=1}^{n-1} k\right) = \Theta\left(\frac{(n-1)n}{2}\right) = \Theta(n^2)$$

i, per tant,

$$\Theta(n) \leq t_{ins}(n) \leq \Theta(n^2).$$

Algorismes iteratius

Hem vist que el cost d'ordenar per inserció n elements és $t_{ins}(n)$, on

$$\Theta(1) + (\Theta(n) \times \Theta(1)) \leq t_{ins}(n) \leq \Theta(1) + \sum_{k=1}^{n-1} \Theta(k).$$

Però sabem que

$$\sum_{k=1}^{n-1} k = 1 + 2 + \cdots + (n-1) = \frac{(n-1)n}{2}.$$

Aleshores,

$$\sum_{k=1}^{n-1} \Theta(k) = \Theta\left(\sum_{k=1}^{n-1} k\right) = \Theta\left(\frac{(n-1)n}{2}\right) = \Theta(n^2)$$

i, per tant,

$$\Theta(n) \leq t_{ins}(n) \leq \Theta(n^2).$$

Algorismes recursius

El cost d'un algorisme recursiu s'expressa sovint en forma de recurrència.

Definició

Una recurrència és una equació o una desigualtat que descriu una funció expressada en termes del seu valor per a entrades més petites.

Exemple

$$C(n) = \begin{cases} 1, & \text{si } n = 1 \\ C(n-1) + n, & \text{si } n \geq 2 \end{cases}$$

Resoldre una recurrència vol dir donar-ne una forma tancada o, almenys, fites Θ o O de la seva solució.

Algorismes recursius

El cost d'un algorisme recursiu s'expressa sovint en forma de recurrència.

Definició

Una recurrència és una equació o una desigualtat que descriu una funció expressada en termes del seu valor per a entrades més petites.

Exemple

$$C(n) = \begin{cases} 1, & \text{si } n = 1 \\ C(n-1) + n, & \text{si } n \geq 2 \end{cases}$$

Resoldre una recurrència vol dir donar-ne una forma tancada o, almenys, fites Θ o O de la seva solució.

Algorismes recursius

Exemple

$$C(n) = \begin{cases} 1, & \text{si } n = 1 \\ C(n-1) + n, & \text{si } n \geq 2 \end{cases}$$

Idea

Podem observar que

- $C(1) = 1$
- $C(2) = 1 + 2 = 3$
- $C(3) = 3 + 3 = 6$
- $C(n) = C(n-1) + n = C(n-2) + (n-1) + n = \cdots$

Algorismes recursius

Exemple

$$C(n) = \begin{cases} 1, & \text{si } n = 1 \\ C(n-1) + n, & \text{si } n \geq 2 \end{cases}$$

Idea

Podem observar que

- $C(1) = 1$
- $C(2) = 1 + 2 = 3$
- $C(3) = 3 + 3 = 6$
- $C(n) = C(n-1) + n = C(n-2) + (n-1) + n = \cdots$

Algorismes recursius

Exemple

$$C(n) = \begin{cases} 1, & \text{si } n = 1 \\ C(n-1) + n, & \text{si } n \geq 2 \end{cases}$$

Idea

Podem observar que

- $C(1) = 1$
- $C(2) = 1 + 2 = 3$
- $C(3) = 3 + 3 = 6$
- $C(n) = C(n-1) + n = C(n-2) + (n-1) + n = \cdots$

Algorismes recursius

Exemple

$$C(n) = \begin{cases} 1, & \text{si } n = 1 \\ C(n-1) + n, & \text{si } n \geq 2 \end{cases}$$

Idea

Podem observar que

- $C(1) = 1$
- $C(2) = 1 + 2 = 3$
- $C(3) = 3 + 3 = 6$
- $C(n) = C(n-1) + n = C(n-2) + (n-1) + n = \cdots$

Algorismes recursius

Exemple

$$C(n) = \begin{cases} 1, & \text{si } n = 1 \\ C(n-1) + n, & \text{si } n \geq 2 \end{cases}$$

Solució

$$\begin{aligned} C(n) &= C(n-1) + n \\ &= C(n-2) + (n-1) + n \\ &= C(n-3) + (n-2) + (n-1) + n \\ &\ \vdots \\ &= C(1) + 2 + \cdots + (n-2) + (n-1) + n \\ &= 1 + 2 + \cdots + n \\ &= \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \in \Theta(n^2). \end{aligned}$$

Algorismes recursius

Per descriure una recurrència que expressi el cost d'un algorisme recursiu, n'hi ha prou a determinar:

- el paràmetre de recursió n,
- el cost del cas base ($n = 0$, $n = 1$,...)
- el cost del cas inductiu
  - nombre de crides recursives
  - valor del paràmetre recursiu en les crides
  - cost dels càlculs addicionals no recursius

Cerca lineal recursiva

Comprovar si un nombre x apareix en un vector v entre les posicions 0 i $n-1$ comparant-lo amb $v[0], v[1], \ldots, v[n-1]$.
Si es troba x, retornar la seva posició en v. Altrament, retornar -1.

```
int cerca_lineal(const vector<int>& v,int n,int x) {
    if (n == 0) return -1;
    else if (v[n-1] == x) return n-1;
         else return cerca_lineal(v,n-1,x);
}
```

El paràmetre de recursió és n, la mida del vector. Definim la recurrència $T(n)$ que representa el cost (en cas pitjor) de l'algorisme:

$$T(n) = T(n-1) + \Theta(1)$$

Algorismes recursius

Recurrència per a la cerca lineal

$T(n) = T(n-1) + \Theta(1)$ per a $n \geq 1$, i
$T(0) = \Theta(1)$.

Solució

$$
\begin{aligned}
T(n) &= T(n-1) + \Theta(1) \\
&= T(n-2) + 2 \cdot \Theta(1) \\
&= T(n-3) + 3 \cdot \Theta(1) \\
&\ \vdots \\
&= T(0) + n \cdot \Theta(1) \\
&= (n+1) \cdot \Theta(1) \\
&= \Theta(n+1) = \Theta(n).
\end{aligned}
$$

Algorismes recursius

Cerca binària recursiva

Comprovar si un nombre *x* apareix en un vector ordenat *v* entre les posicions *i* i *j* per cerca binària.
Si es troba *x*, retornar la seva posició en *v*. Altrament, retornar -1.

```
int cerca_binaria(const vector<int>& v,int i,int j,int x)
{  if (i <= j) {
       int k = (i + j) / 2;
       if (x == v[k])
           return k;
       else if (x < v[k])
           return cerca_binaria(v,i,k-1,x);
       else
           return cerca_binaria(v,k+1,j,x);
   }
   else return -1;
}
```

Algorismes recursius

```
int cerca_binaria(const vector<int>& v,int i,int j,int x)
{   if (i <= j) {
        int k = (i + j) / 2;
        if (x == v[k])
            return k;
        else if (x < v[k])
            return cerca_binaria(v,i,k-1,x);
        else
            return cerca_binaria(v,k+1,j,x);
    }
    else return -1;
}
```

El paràmetre de recursió és $n = j - i$, la mida de l'interval a explorar.
Definim la recurrència $T(n)$, el cost (en cas pitjor) de l'algorisme:

$$T(n) = T(n/2) + \Theta(1)$$

Algorismes recursius

Recurrència per a la cerca binària

$T(n) = T(n/2) + \Theta(1)$ per a $n \geq 1$, i
$T(0) = \Theta(1)$.

Solució

$$\begin{aligned}
T(n) &= T(n/2) + \Theta(1) \\
&= T(n/4) + 2 \cdot \Theta(1) \\
&= T(n/8) + 3 \cdot \Theta(1) \\
&\vdots \\
&= T(n/2^{\log n}) + \log n \cdot \Theta(1) \\
&= T(1) + \log n \cdot \Theta(1) \\
&= T(0) + (\log n + 1) \cdot \Theta(1) \\
&= (\log n + 2) \cdot \Theta(1) = \Theta(\log n + 2) = \Theta(\log n).
\end{aligned}$$

Teoremes mestres

Per sistematitzar l'anàlisi del cost dels algorismes recursius, els classifiquem en dos grups en funció de com divideixen el problema d'entrada en subproblemes en les crides recursives.

Sigui A un algorisme que, amb una entrada de mida n, fa a crides recursives i una feina addicional no recursiva de cost $g(n)$. Llavors, si en les crides recursives els subproblemes tenen mida

- $n - c$, el cost d'A ve descrit per la recurrència

$$T(n) = a \cdot T(n - c) + g(n)$$

- n/b, el cost d'A ve descrit per la recurrència

$$T(n) = a \cdot T(n/b) + g(n)$$

Teoremes mestres

Les dues menes de recurrències anteriors:

- subtractives: $T(n) = a \cdot T(n - c) + g(n)$
- divisores: $T(n) = a \cdot T(n/b) + g(n)$

es poden resoldre amb els teoremes mestres que veurem a continuació.

Teoremes mestres

Teorema mestre de recurrències subtractives

Sigui $T(n) = \begin{cases} f(n), & \text{si } 0 \leq n < n_0 \\ a \cdot T(n-c) + g(n), & \text{si } n \geq n_0 \end{cases}$

on $n_0 \in \mathbb{N}$, $c \geq 1$, f és una funció arbitrària i $g \in \Theta(n^k)$ per a $k \geq 0$.

Aleshores

$$T(n) \in \begin{cases} \Theta(n^k), & \text{si } a < 1 \\ \Theta(n^{k+1}), & \text{si } a = 1 \\ \Theta(a^{n/c}), & \text{si } a > 1 \end{cases}$$

Teoremes mestres

Teorema mestre de recurrències subtractives

Sigui $T(n) = \begin{cases} f(n), & \text{si } 0 \leq n < n_0 \\ a \cdot T(n-c) + g(n), & \text{si } n \geq n_0 \end{cases}$

on $n_0 \in \mathbb{N}$, $c \geq 1$, f és una funció arbitrària i $g \in \Theta(n^k)$ per a $k \geq 0$.

Aleshores

$$T(n) \in \begin{cases} \Theta(n^k), & \text{si } a < 1 \\ \Theta(n^{k+1}), & \text{si } a = 1 \\ \Theta(a^{n/c}), & \text{si } a > 1 \end{cases}$$

Exemple 1

Hem vist que el cost de l'algorisme recursiu de cerca lineal es pot descriure amb la recurrència $T(n) = T(n-1) + \Theta(1)$ per a $n \geq 1$, i $T(0) = \Theta(1)$.

Per tant, $n_0 = 1$, $a = 1$, $c = 1$, $k = 0$. Llavors, $T(n)$ pertany al segon cas:

$$T(n) \in \Theta(n^{k+1}) = \Theta(n).$$

Teoremes mestres

Teorema mestre de recurrències subtractives

Sigui $T(n) = \begin{cases} f(n), & \text{si } 0 \leq n < n_0 \\ a \cdot T(n-c) + g(n), & \text{si } n \geq n_0 \end{cases}$

on $n_0 \in \mathbb{N}$, $c \geq 1$, f és una funció arbitrària i $g \in \Theta(n^k)$ per a $k \geq 0$.

Aleshores

$$T(n) \in \begin{cases} \Theta(n^k), & \text{si } a < 1 \\ \Theta(n^{k+1}), & \text{si } a = 1 \\ \Theta(a^{n/c}), & \text{si } a > 1 \end{cases}$$

Exemple 2

En la recurrència $T(n) = T(n-1) + \Theta(n)$, tenim els valors

$$a = 1,\ c = 1,\ k = 1.$$

Llavors, $T(n)$ pertany al segon cas:

$$T(n) \in \Theta(n^{k+1}) = \Theta(n^2).$$

Teoremes mestres

Teorema mestre de recurrències subtractives

Sigui $T(n) = \begin{cases} f(n), & \text{si } 0 \leq n < n_0 \\ a \cdot T(n-c) + g(n), & \text{si } n \geq n_0 \end{cases}$

on $n_0 \in \mathbb{N}$, $c \geq 1$, f és una funció arbitrària i $g \in \Theta(n^k)$ per a $k \geq 0$.

Aleshores

$$T(n) \in \begin{cases} \Theta(n^k), & \text{si } a < 1 \\ \Theta(n^{k+1}), & \text{si } a = 1 \\ \Theta(a^{n/c}), & \text{si } a > 1 \end{cases}$$

Exemple 3

En la recurrència $T(n) = 2 \cdot T(n-1) + \Theta(n)$, tenim els valors

$$a = 2,\ c = 1,\ k = 1.$$

Llavors, $T(n)$ pertany al tercer cas:

$$T(n) \in \Theta(2^n).$$

Teoremes mestres

Exemple 4

Els nombres de Fibonacci estan definits per la recurrència
$f(k) = f(k-1) + f(k-2)$ per a $k \geq 2$, amb $f(0) = f(1) = 1$.

La solució recursiva és evident.

```
int fibonacci (int k) {
  if (k <= 1) return 1;
  else return fibonacci(k-1) + fibonacci(k-2);
}
```

- El cost segueix la recurrència $T(k) = T(k-1) + T(k-2) + \Theta(1)$
- No podem aplicar directament el teorema mestre per resoldre-la!

Teoremes mestres

Exemple 4

Els nombres de Fibonacci estan definits per la recurrència
$f(k) = f(k-1) + f(k-2)$ per a $k \geq 2$, amb $f(0) = f(1) = 1$.

La solució recursiva és evident.

```
int fibonacci (int k) {
  if (k <= 1) return 1;
  else return fibonacci(k-1) + fibonacci(k-2);
}
```

- El cost segueix la recurrència $T(k) = T(k-1) + T(k-2) + \Theta(1)$
- No podem aplicar directament el teorema mestre per resoldre-la!

Teoremes mestres

Podem aplicar el teorema mestre a dues fites de $T(k)$:

- $T(k) = T(k-1) + T(k-2) + \Theta(1) \leq 2T(k-1) + \Theta(1)$ dona $T(k) \in O(2^k)$
- $T(k) = T(k-1) + T(k-2) + \Theta(1) \geq 2T(k-2) + \Theta(1)$ dona $T(k) \in \Omega(2^{k/2}) = \Omega(\sqrt{2}^k)$

Es pot demostrar que $T(k) = \Theta(\phi^k)$, on $\phi = \frac{1+\sqrt{5}}{2}$ (*nombre d'or*).
Noteu que $\sqrt{2} = 1.414213562...$ i $\phi = 1.618033988...$

Teoremes mestres

Podem aplicar el teorema mestre a dues fites de $T(k)$:

- $T(k) = T(k-1) + T(k-2) + \Theta(1) \leq 2T(k-1) + \Theta(1)$ dona $T(k) \in O(2^k)$
- $T(k) = T(k-1) + T(k-2) + \Theta(1) \geq 2T(k-2) + \Theta(1)$ dona $T(k) \in \Omega(2^{k/2}) = \Omega(\sqrt{2}^k)$

Es pot demostrar que $T(k) = \Theta(\phi^k)$, on $\phi = \frac{1+\sqrt{5}}{2}$ (*nombre d'or*).
Noteu que $\sqrt{2} = 1.414213562...$ i $\phi = 1.618033988...$

Teoremes mestres

Teorema mestre de recurrències divisores

Sigui $T(n) = \begin{cases} f(n), & \text{si } 0 \le n < n_0 \\ a \cdot T(n/b) + g(n), & \text{si } n \ge n_0 \end{cases}$

on $n_0 \in \mathbb{N}$, $b > 1$, f és una funció arbitrària i $g \in \Theta(n^k)$ per a $k \ge 0$.

Sigui $\alpha = \log_b(a)$. Aleshores,

$$T(n) \in \begin{cases} \Theta(n^k), & \text{si } \alpha < k \\ \Theta(n^k \log n), & \text{si } \alpha = k \\ \Theta(n^\alpha), & \text{si } \alpha > k \end{cases}$$

Exemple 1

Hem vist que el cost de l'algorisme recursiu de cerca binària es pot descriure amb $T(n) = T(n/2) + \Theta(1)$, $n \ge 1$, i $T(0) = \Theta(1)$.
Per tant, $n_0 = 1$, $a = 1$, $b = 2$, $k = 0$, $\alpha = 0$. Llavors, $T(n)$ pertany al 2n cas:

$$T(n) \in \Theta(n^k \log n) = \Theta(\log n).$$

Teoremes mestres

Teorema mestre de recurrències divisores

Sigui $T(n) = \begin{cases} f(n), & \text{si } 0 \le n < n_0 \\ a \cdot T(n/b) + g(n), & \text{si } n \ge n_0 \end{cases}$

on $n_0 \in \mathbb{N}$, $b > 1$, f és una funció arbitrària i $g \in \Theta(n^k)$ per a $k \ge 0$.

Sigui $\alpha = \log_b(a)$. Aleshores,

$$T(n) \in \begin{cases} \Theta(n^k), & \text{si } \alpha < k \\ \Theta(n^k \log n), & \text{si } \alpha = k \\ \Theta(n^\alpha), & \text{si } \alpha > k \end{cases}$$

Exemple 1

Hem vist que el cost de l'algorisme recursiu de cerca binària es pot descriure amb $T(n) = T(n/2) + \Theta(1)$, $n \ge 1$, i $T(0) = \Theta(1)$.
Per tant, $n_0 = 1$, $a = 1$, $b = 2$, $k = 0$, $\alpha = 0$. Llavors, $T(n)$ pertany al 2n cas:

$$T(n) \in \Theta(n^k \log n) = \Theta(\log n).$$

Teoremes mestres

Exemple 2

Funció principal de l'ordenació per fusió (*mergesort*)

```
template <typename elem>
void mergesort(vector<elem>& v,int e,int d) {
    if (e < d) {
        int m = (e + d) / 2;
        mergesort(v, e, m);
        mergesort(v, m + 1, d);
        merge(v, e, m, d);
}   }
```

Tenint en compte que el cost de la crida `merge(v,e,m,d)` és $\Theta(n)$ (on $n = d - e + 1$), el cost total es pot expressar amb la recurrència:

$$T(n) = 2T(n/2) + \Theta(n) \text{ per a } n \geq 2, \text{ i } T(1) = \Theta(1).$$

Teoremes mestres

Teorema mestre de recurrències divisores

Sigui $T(n) = \begin{cases} f(n), & \text{si } 0 \leq n < n_0 \\ a \cdot T(n/b) + g(n), & \text{si } n \geq n_0 \end{cases}$

on $n_0 \in \mathbb{N}$, $b > 1$, f és una funció arbitrària i $g \in \Theta(n^k)$ per a $k \geq 0$.

Sigui $\alpha = \log_b(a)$. Aleshores,

$$T(n) \in \begin{cases} \Theta(n^k), & \text{si } \alpha < k \\ \Theta(n^k \log n), & \text{si } \alpha = k \\ \Theta(n^\alpha), & \text{si } \alpha > k \end{cases}$$

Exemple 2

Hem vist que el cost de l'ordenació per fusió es pot descriure amb la recurrència $T(n) = 2T(n/2) + \Theta(n)$ per a $n \geq 2$ i $T(1) = \Theta(1)$.

Per tant, $n_0 = 2$, $a = 2$, $b = 2$, $k = 1$, $\alpha = 1$. Llavors, $T(n)$ pertany al 2n cas:

$$T(n) \in \Theta(n^k \log n) = \Theta(n \log n).$$

Teoremes mestres

Exercici 1

Resoleu la recurrència $T(n) = T(\sqrt{n}) + 1$.

Pista

Fer canvi de variable $m = \log n$.

Teoremes mestres

Exercici 1

Resoleu la recurrència $T(n) = T(\sqrt{n}) + 1$.

Solució

Fem el canvi de variable $m = \log n$. Aleshores,

$$T(n) = T(2^m) = T(2^{m/2}) + 1.$$

Definim $S(m) = T(2^m)$, que compleix

$$S(m) = S(m/2) + 1.$$

Pel segon teorema mestre, tenim que $S(m) \in \Theta(\log m)$ i, per tant:

$$T(n) = T(2^m) = S(m) \in \Theta(\log m) = \Theta(\log \log n).$$

Teoremes mestres

Exercici 1

Resoleu la recurrència $T(n) = T(\sqrt{n}) + 1$.

Solució

Fem el canvi de variable $m = \log n$. Aleshores,

$$T(n) = T(2^m) = T(2^{m/2}) + 1.$$

Definim $S(m) = T(2^m)$, que compleix

$$S(m) = S(m/2) + 1.$$

Pel segon teorema mestre, tenim que $S(m) \in \Theta(\log m)$ i, per tant:

$$T(n) = T(2^m) = S(m) \in \Theta(\log m) = \Theta(\log\log n).$$

Exercici 2

Resoleu la recurrència $T(n) = 2T(\sqrt{n}) + \log n$.

Fita inferior dels algorismes d'ordenació

En termes de cost asimptòtic, l'algorisme d'ordenació per fusió és òptim:

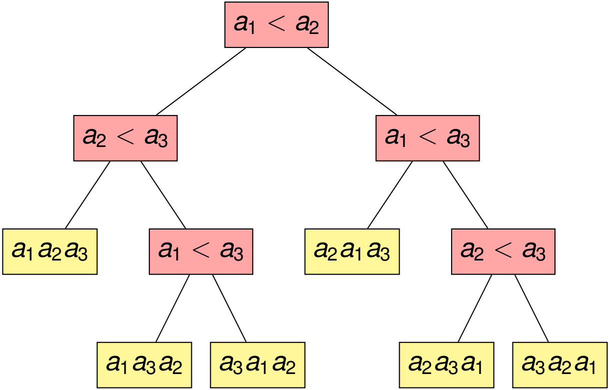
Proposició

Tot algorisme d'ordenació basat en comparacions té cost $\Omega(n \log n)$.

Es pot argumentar fent servir arbres per representar els algorismes d'ordenació basats en comparacions.

Fita inferior dels algorismes d'ordenació

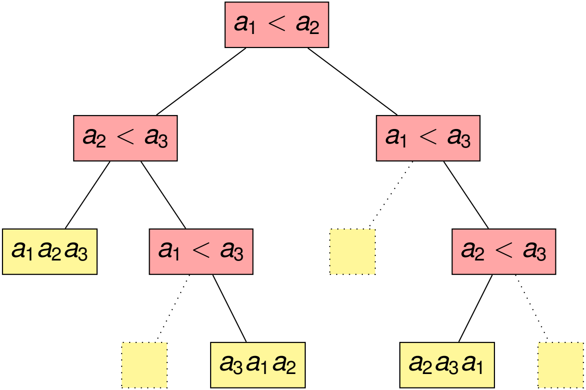
Suposem que volem ordenar a_1, a_2 i a_3. Si $a_1 < a_2$, seguim per la branca esquerra; si no, per la dreta. Els rectangles grocs representen les ordenacions trobades. L'alçària de l'arbre és el cost en cas pitjor.



Fita inferior dels algorismes d'ordenació

Considerem un arbre que ordena n elements:

- cada fulla correspon a una permutació de $\{1, 2, \ldots, n\}$
- cada permutació de $\{1, 2, \ldots, n\}$ ha d'aparèixer en alguna fulla
  (si una no hi fos, què passaria si es donés com a entrada?)



Fita inferior dels algorismes d'ordenació

- com que hi ha $n!$ permutacions de n elements, l'arbre té $\geq n!$ fulles
- tot arbre binari amb $\geq k$ fulles té alçària $\geq \log k$
- per tant, l'alçària del nostre arbre és almenys de $\log n!$

El cost de l'algorisme representat per l'arbre és, per tant, $\Omega(\log n!)$. Com que

$$n! \geq n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot \lfloor n/2 \rfloor \geq (n/2)^{(n/2)}$$

tenim que

$$\log n! \geq \log(n/2)^{(n/2)} = \frac{n}{2} \log(n/2) \in \Omega(n \log n).$$

Proposició

Tot algorisme d'ordenació basat en comparacions té cost $\Omega(n \log n)$.

Fita inferior dels algorismes d'ordenació

- com que hi ha $n!$ permutacions de n elements, l'arbre té $\geq n!$ fulles
- tot arbre binari amb $\geq k$ fulles té alçària $\geq \log k$
- per tant, l'alçària del nostre arbre és almenys de $\log n!$

El cost de l'algorisme representat per l'arbre és, per tant, $\Omega(\log n!)$. Com que

$$n! \geq n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot \lfloor n/2 \rfloor \geq (n/2)^{(n/2)}$$

tenim que

$$\log n! \geq \log(n/2)^{(n/2)} = \frac{n}{2} \log(n/2) \in \Omega(n \log n).$$

Proposició

Tot algorisme d'ordenació basat en comparacions té cost $\Omega(n \log n)$.

Fita inferior dels algorismes d'ordenació

- com que hi ha $n!$ permutacions de n elements, l'arbre té $\geq n!$ fulles
- tot arbre binari amb $\geq k$ fulles té alçària $\geq \log k$
- per tant, l'alçària del nostre arbre és almenys de $\log n!$

El cost de l'algorisme representat per l'arbre és, per tant, $\Omega(\log n!)$. Com que

$$n! \geq n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot \lfloor n/2 \rfloor \geq (n/2)^{(n/2)}$$

tenim que

$$\log n! \geq \log(n/2)^{(n/2)} = \frac{n}{2} \log(n/2) \in \Omega(n \log n).$$

Proposició

Tot algorisme d'ordenació basat en comparacions té cost $\Omega(n \log n)$.

Fita inferior dels algorismes d'ordenació

- com que hi ha $n!$ permutacions de n elements, l'arbre té $\geq n!$ fulles
- tot arbre binari amb $\geq k$ fulles té alçària $\geq \log k$
- per tant, l'alçària del nostre arbre és almenys de $\log n!$

El cost de l'algorisme representat per l'arbre és, per tant, $\Omega(\log n!)$. Com que

$$n! \geq n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot \lfloor n/2 \rfloor \geq (n/2)^{(n/2)}$$

tenim que

$$\log n! \geq \log(n/2)^{(n/2)} = \frac{n}{2} \log(n/2) \in \Omega(n \log n).$$

Proposició

Tot algorisme d'ordenació basat en comparacions té cost $\Omega(n \log n)$.

Fita inferior dels algorismes d'ordenació

- com que hi ha $n!$ permutacions de n elements, l'arbre té $\geq n!$ fulles
- tot arbre binari amb $\geq k$ fulles té alçària $\geq \log k$
- per tant, l'alçària del nostre arbre és almenys de $\log n!$

El cost de l'algorisme representat per l'arbre és, per tant, $\Omega(\log n!)$. Com que

$$n! \geq n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot \lfloor n/2 \rfloor \geq (n/2)^{(n/2)}$$

tenim que

$$\log n! \geq \log(n/2)^{(n/2)} = \frac{n}{2} \log(n/2) \in \Omega(n \log n).$$

Proposició

Tot algorisme d'ordenació basat en comparacions té cost $\Omega(n \log n)$.