Eficiència d'algorismes

Salvador Roura

1 Introducció

La qualitat fonamental i imprescindible per a un algorisme és que aquest sigui correcte, és a dir, que produeixi el resultat desitjat en un temps finit. Els algorismes, però, haurien de presentar altres qualitats addicionals, com ara ser eficients, clars, ben estructurats, de fàcil us, etcètera. Un dels criteris més importants a l'hora d'avaluar la qualitat d'un algorisme es la seva *eficiència*. Aquesta s'acostuma a mesurar en funció de dos paràmetres: la velocitat de l'algorisme (eficiència temporal) i la quantitat de memòria que requereix la seva execució (eficiència espacial). De manera informal, volem algorismes ràpids que no ocupin més memòria de la estrictament necessària.

Nosaltres ens interessarem bàsicament en la velocitat dels algorismes, i per tant, quan parlem d'eficiència entendrem que ens referim a l'eficiència temporal. Per mesurar-la, hi ha bàsicament dues possibilitats:

- Proves empíriques
- Anàlisi matemàtica

La primera possibilitat consisteix a traduïr l'algorisme a un llenguatge de programació concret, i seguidament realitzar mesures dels temps d'execució observats del programa quan s'aplica a un conjunt de mostres d'entrada. A partir d'aquestes mesures s'intenta inferir el rendiment del programa (i, indirectament, de l'algorisme) per a dades d'entrada que no es trobin en les mostres utilitzades.

Aquest mètode és poc fiable, ja que depèn del subconjunt de proves escollit, i també de l'entorn de programació emprat (ordinador, llenguatge de programació, compilador). A més, en molts casos s'avalua l'eficiència d'un algorisme precisament per saber si valdria la pena construir-ne un programa.

És per això que introduïm l'anàlisi matemàtica d'algorismes. Aquesta es basa en la utilització de tècniques matemàtiques per a l'anàlisi rigurosa de l'eficiència dels algorismes. Recordem que els algorismes són mètodes abstractes per a la resolució de problemes, i per tant la seva eficiència no pot dependre de qüestions pràctiques com ara la velocitat de l'ordinador del què disposem. Per tant, les conclusions que s'obtenen han de ser inherents als algorismes analitzats, i independents del llenguatge de programació o de l'ordinador que s'utilitzi.

Veiem-ne un exemple concret. Suposem que volem avaluar l'eficiència de la funció següent, la qual calcula on es troba la clau més petita d'una taula (recordem que un element anomenat "clau" no es pot trobar repetit).

Precondició: { La taula T conté n claus a les posicions $0..n-1$, amb $n \geq 1$ }

enter posició_mínim(const taula de claus &T; enter n)

Postcondició: { posició_mínim(T, n) = pos
$\Rightarrow 0 \leq \text{pos} < n \wedge \text{T[pos]} = \mathit{Mín}\{T[i]\}_{0 \leq i < n}$ }

```
(1)     pos := 0
(2)     min := T[0]
(3)     i := 1
(4)     mentre i < n
(5)         si T[i] < min
(6)             pos := i
(7)             min := T[i]
(8)         ++i
        retorna pos
```

Analitzem aquesta funció detalladament, considerent que les úniques operacions que suposen algun temps d'execució són les assignacions, les comparacions i els increments. Sigui T_a el temps necessari per fer una assignació, T_c el temps necessari per fer una comparació, i T_i el temps necessari per incrementar el valor d'una variable. Les línies (1), (2) i (3) s'executen un sol cop, i cadascuna suposa una assignació. Per tant, en conjunt contribueixen amb un terme $3T_a$ al temps d'execució de la funció. La comparació de la línia (4) s'executa n vegades (per a valors de la variable "i" entre 1 i n), i això afegeix una contribució nT_c al temps d'execució. La comparació de la línia (5) i l'increment de la línia (8) s'executen $n-1$ vegades, i per tant aporten $(n-1)T_c$ i $(n-1)T_i$ al temps d'execució, respectivament.

Calcular quantes vegades s'executen les línies (6) i (7) és més difícil, i de fet depèn de l'ordre en què es trobin les claus a la taula. En *el cas pitjor* haurem d'executar aquestes línies $n-1$ vegades (això succeeix quan la taula està ordenada decreixentment), així que la contribució al temps d'execució serà com a molt $2(n-1)T_a$. En *el cas millor* no s'executaran mai (quan la clau més petita de totes es trobi a la primera posició de la taula), i en aquest cas la contribució de les línies (6) i (7) al temps total d'execució serà nul·la. Així doncs, tenim que la variable T_n, corresponent al temps d'execució de la funció posició_mínim() amb una taula amb n claus, està fitada superiorment per

$$\begin{aligned} T_n &\leq 3T_a + nT_c + (n-1)T_c + (n-1)T_i + 2(n-1)T_a \\ &= (2T_a + 2T_c + T_i)n + (T_a - T_c - T_i), \end{aligned}$$

i inferiorment per

$$\begin{aligned} T_n &\geq 3T_a + nT_c + (n-1)T_c + (n-1)T_i \\ &= (2T_c + T_i)n + (3T_a - T_c - T_i). \end{aligned}$$

Com es pot veure, el temps real d'execució de la funció posició_mínim() depèn, a part de la disposició dels elements a la taula, dels temps d'execució de cadascuna de les tres operacions elementals considerades: assignació, comparació

i increment. Per exemple, si una assignació requereix $4\mu s$, una comparació requereix $2\mu s$ i un increment requereix $1\mu s$, llavors el temps d'execució està fitat per $(5n+9)\mu s \leq T_n \leq (13n+1)\mu s$. Per a cada combinació de valors de T_a, T_c i T_i podem obtenir fites inferiors i superiors per al valor de la variable T_n, fites que sempre seran bàsicament funcions lineals en n, amb constants depenents de la velocitat de l'ordinador utilitzat.

Aquest tipus d'anàlisi, que considera el nombre precís d'execucions de cadascuna de les operacions elementals, és una opció raonable per avaluar l'eficiència dels algorismes. Ara bé, també presenta algunes dificultats: Per una banda, realitzar aquesta anàlisi exacta pot ser molt difícil o molt laboriós si l'algorisme és complicat. Per una altra banda, no és fàcil determinar amb precisió el conjunt d'operacions elementals que cal mesurar. Així, en l'exemple anterior no hem tingut en compte el cost del pas dels paràmetres, de la creació de variables locals o de l'última instrucció de retorn del resultat; tampoc hem considerat quant costa accedir a un element d'una taula; finalment, hem suposat el cost de les operacions entre claus idèntic al cost de les operacions entre enters, hipòtesi que pot ser falsa depenent de la definició del tipus clau. A més, un cop feta l'anàlisi obtenim fites pel temps total d'execució que depenen dels temps d'execució de cada operació elemental, que per la seva banda depenen de l'ordinador que fem servir, del llenguatge de programació escollit o del compilador emprat.

2 Les notacions $\mathcal{O}()$, $\Theta()$ i $o()$

El nostre objectiu és poder fer afirmacions absolutes sobre l'eficiència d'un algorisme donat, observacions que siguin independents del procés de traducció de l'algorisme a un llenguatge de programació concret sobre una determinada màquina. Informalment parlant, el què volem és poder expressar que la funció posició_mínim() requereix un temps d'execució que és fonamentalment proporcional a n, el nombre de claus a la taula T, ja que aquest és un fet algorísmic independent de l'ordinador utilitzat. Per això introduim les notacions matemàtiques $\mathcal{O}()$, $\Theta()$ i $o()$.

Definició 1. *Sigui* $g : \mathbb{N} \longrightarrow \mathbb{R}$. *Definim el conjunt de funcions* $\mathcal{O}(g)$ *com*

$$\mathcal{O}(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} \mid [\exists c > 0, \exists n_0 \in \mathbb{N} \mid \forall n \geq n_0 : |f(n)| \leq c \cdot |g(n)|]\}.$$

De manera intuïtiva, $\mathcal{O}(g)$ és el conjunt de funcions que *asimptòticament* (és a dir, per a valors suficientment grans d'n) creixen com a molt de manera proporcional a la funció g. Per exemple,

$$\mathcal{O}(n^3) = \{n^2, -n, \sqrt{n}, 7n^2-4, n/2, n^3, -10n^3+5n, n^3/20, 8, 0, -2/n, 7/n^2, \dots\}.$$

Per a les funcions $f, g : \mathbb{N} \longrightarrow \mathbb{R}$ que ens trobarem a la pràctica, podrem dir que $f \in \mathcal{O}(g)$ sii $0 \leq \lim_{n\to\infty} |f(n)/g(n)| < \infty$.

Definició 2. *Sigui* $g : \mathbb{N} \longrightarrow \mathbb{R}$. *Definim el conjunt de funcions* $\Theta(g)$ *com*

$$\Theta(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} \mid [f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f)]\}.$$

El conjunt $\Theta(g)$ està format per les funcions que asimptòticament creixen de manera bàsicament proporcional a la funció g. Es per aquest motiu que quan $f \in \Theta(g)$ es diu que f i g són del mateix ordre (de creixement). Per exemple,

$$\Theta\big(n^3\big) = \{n^3, -10n^3 + 5n, n^3/20, \dots\}.$$

Un altre cop, hi ha una definició alternativa del tot natural que es pot aplicar a les funcions $f, g : \mathbb{N} \longrightarrow \mathbb{R}$ amb què ens trobarem a la pràctica: direm que $f \in \Theta(g)$ sii $0 < \lim_{n\to\infty} |f(n)/g(n)| < \infty$.

Definició 3. *Sigui $g : \mathbb{N} \longrightarrow \mathbb{R}$. Definim el conjunt de funcions $o(g)$ com*

$$o(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} \mid [\forall \varepsilon > 0\ \exists n_0 \in \mathbb{N} \mid \forall n \geq n_0 \Rightarrow |f(n)| \leq \varepsilon \cdot |g(n)|]\}.$$

Una manera alternativa de formular aquesta definició és dir que $f \in o(g)$ sii $\lim_{n\to\infty} f(n)/g(n) = 0$. Així doncs, les funcions dins del conjunt $o(g)$ són aquelles que asimptòticament creixen estrictament més a poc a poc que la funció g. Per això, si $f \in o(g)$ es diu que f és d'ordre inferior a g. Per exemple,

$$o\big(n^3\big) = \{n^2, n, -\sqrt{n}, 7n^2 - 4, n/2, -8, 0, -2/n, 7/n^2 \dots\}.$$

Per a les funcions que ens podrem trobar a la pràctica, hi ha una altra definició del conjunt $o(g)$: podrem dir que $f \in o(g)$ sii $f \in \mathcal{O}(g)$ però $g \notin \mathcal{O}(f)$.

Les propietats incloses a les proposicions següents es poden demostrar amb facilitat.

Proposició 4. *Siguin $f, g : \mathbb{N} \longrightarrow \mathbb{R}$, i siguin $a, b \in \mathbb{R}$ tals que $a \neq 0$, $b \neq 0$.*

- *Els operadors $\Theta()$ i $\mathcal{O}()$ són reflexius, i l'operador $o()$ és antireflexiu: $f \in \Theta(f)$, $f \in \mathcal{O}(f)$ i $f \notin o(f)$.*
- *L'operador $\Theta()$ és simètric: $f \in \Theta(g)$ sii $g \in \Theta(f)$.*
- *L'operador $o()$ és antisimètric: Si $f \in o(g)$ llavors $g \notin o(f)$.*
- *Si $f \in \Theta(g)$ o $f \in o(g)$ llavors $f \in \mathcal{O}(g)$. (De fet, per a les funcions "normals" es compleix que $\mathcal{O}(g) = o(g) \cup \Theta(g)$.)*
- *Si $f \in \Theta(g)$ llavors $a \cdot f \in \Theta(b \cdot g)$.*
- *Si $f \in \mathcal{O}(g)$ llavors $a \cdot f \in \mathcal{O}(b \cdot g)$.*
- *Si $f \in o(g)$ llavors $a \cdot f \in o(b \cdot g)$.*

Proposició 5. *Siguin $f, g, h : \mathbb{N} \longrightarrow \mathbb{R}$.*

- *L'operador $\Theta()$ és transitiu: Si $f \in \Theta(g)$ i $g \in \Theta(h)$ llavors $f \in \Theta(h)$.*
- *Si $f \in \Theta(g)$ i $g \in \mathcal{O}(h)$ llavors $f \in \mathcal{O}(h)$.*
- *Si $f \in \mathcal{O}(g)$ i $g \in \Theta(h)$ llavors $f \in \mathcal{O}(h)$.*

- *L'operador $\mathcal{O}()$ és transitiu: Si $f \in \mathcal{O}(g)$ i $g \in \mathcal{O}(h)$ llavors $f \in \mathcal{O}(h)$.*
- *Si $f \in \Theta(g)$ i $g \in o(h)$ llavors $f \in o(h)$.*
- *Si $f \in o(g)$ i $g \in \Theta(h)$ llavors $f \in o(h)$.*
- *Si $f \in \mathcal{O}(g)$ i $g \in o(h)$ llavors $f \in o(h)$.*
- *Si $f \in o(g)$ i $g \in \mathcal{O}(h)$ llavors $f \in o(h)$.*
- *L'operador $o()$ és transitiu: Si $f \in o(g)$ i $g \in o(h)$ llavors $f \in o(h)$.*

La proposició següent estableix les relacions d'igualtat, inclusió i inclusió estricte entre els conjunts definits amb $\Theta()$, $\mathcal{O}()$ i $o()$.

Proposició 6.

- *Suposem que $f \in \Theta(g)$. Llavors $\Theta(f) = \Theta(g)$, $\Theta(f) \subset \mathcal{O}(g)$, $\mathcal{O}(f) = \mathcal{O}(g)$, $o(f) \subset \mathcal{O}(g)$, i $o(f) = o(g)$.*
- *Suposem que $f \in \mathcal{O}(g)$. Llavors $\Theta(f) \subset \mathcal{O}(g)$, $\mathcal{O}(f) \subseteq \mathcal{O}(g)$, $o(f) \subset \mathcal{O}(g)$, i $o(f) \subseteq o(g)$.*
- *Suposem que $f \in o(g)$. Llavors $\Theta(f) \subset \mathcal{O}(g)$, $\Theta(f) \subset o(g)$, $\mathcal{O}(f) \subset \mathcal{O}(g)$, $\mathcal{O}(f) \subset o(g)$, $o(f) \subset \mathcal{O}(g)$, i $o(f) \subset o(g)$.*

A continuació es presenten una sèrie de convencions relatives a les notacions $\mathcal{O}()$, $\Theta()$ i $o()$, les quals simplifiquen la seva utilització pràctica.

Convenció 7. *Siguin $f, g : \mathbb{N} \longrightarrow \mathbb{R}$. Podrem dir que $f = \mathcal{O}(g)$ sii $f \in \mathcal{O}(g)$. Podrem dir que $f = \Theta(g)$ sii $f \in \Theta(g)$. Podrem dir que $f = o(g)$ sii $f \in o(g)$.*

Així, per exemple, tenim que $10n^2 - 5n \in \mathcal{O}\big(n^3\big)$, i per tant podrem escriure $10n^2 - 5n = \mathcal{O}\big(n^3\big)$. Igualment podrem dir que $-n^3 + 4 \neq \mathcal{O}\big(n^2\big)$.

(Cal remarcar que aquesta convenció, com totes les que veurem a continuació, no és commutativa. Quan escrivim $n = \mathcal{O}\big(n^2\big)$ el què de fet volem dir és que la funció n pertany al conjunt de funcions $\mathcal{O}\big(n^2\big)$. Lògicament, no podem fer l'afirmació contrària: dir que un conjunt de funcions pertany a una funció. Per tant, no té cap sentit escriure $\mathcal{O}\big(n^3\big) = 10n^2 - 5n$, o dir que $\mathcal{O}\big(n^2\big) \neq -n^3 + 4$.)

Com altres exemples, tenim que $n^3/20 \in \Theta\big(n^3\big)$, i per tant podrem escriure $n^3/20 = \Theta\big(n^3\big)$. També podrem escriure $-n^4 \neq \Theta\big(n^3\big)$. De la mateixa manera, podrem dir que $-\sqrt{n} = o\big(n^3\big)$ o que $n^4 \neq o\big(n^3\big)$.

Convenció 8. *Siguin $f, g : \mathbb{N} \longrightarrow \mathbb{R}$, i suposem que es compleix la relació d'inclusió $\mathcal{N}(f)\ \mathcal{R}\ \mathcal{M}(g)$, on $\mathcal{N}, \mathcal{M} \in \{\mathcal{O}, \Theta, o\}$ i $\mathcal{R} \in \{\subset, \subseteq\}$. Llavors podrem dir que $\mathcal{N}(f) = \mathcal{M}(g)$, volent indicar que qualsevol funció que pertanyi a $\mathcal{N}(f)$ ha de pertànyer a $\mathcal{M}(g)$.*

Per exemple, podríem escriure

$$3(n-1)(n+2) = 3n^2 + 3n - 6 = \mathcal{O}\big(n^2\big) = \mathcal{O}\big(n^3\big)\,.$$

La primera de les igualtats és la convencional entre nombres enters, la segona igualtat utilitza la Convenció 7 (ja que $3n^2+3n-6 \in \mathcal{O}\big(n^2\big)$), i l'última igualtat utilitza la Convenció 8 (ja que $\mathcal{O}\big(n^2\big) \subset \mathcal{O}\big(n^3\big)$, i per tant qualsevol funció que pertanyi a $\mathcal{O}\big(n^2\big)$ ha d'ésser a $\mathcal{O}\big(n^3\big)$).

Fixem-nos que la utilització de la notació asimptòtica produeix, en general, una pèrdua d'informació sobre la funció en estudi (precisament per aquest motiu les convencions no són commutatives). Aquesta pèrdua de informació no ha de ser forçosament perjudicial. De fet, sovint s'utilitzen les notacions $\mathcal{O}()$, $\Theta()$ i $o()$ per remarcar les propietats interessants de les funcions, ignorant detalls com ara funcions de menor ordre o constants que poden no ser rellevants (com quan analitzem l'eficiència dels algorismes per obtenir el seu ordre asimptòtic de creixement).

Com un altre exemple de com les notacions $\mathcal{O}()$, $\Theta()$ i $o()$ permeten proporcionar informació parcial sobre les funcions, considerem la funció $f(n) = 3n^2 + 7$. La taula següent mostra graus d'informació decreixents sobre f, cadascun amb unes quantes de les funcions que en són consistents.

Informació	Funcions candidates
$f(n) = 3n^2 + 7$	$3n^2 + 7$
$f(n) = 3n^2 + \Theta(1)$	$3n^2 + 7, 3n^2 + 1 + 1/\sqrt{n}, 3n^2 - 40, \ldots$
$f(n) = 3n^2 + \mathcal{O}(1)$	Totes les anteriors, més $3n^2 - 1/\sqrt{n}, 3n^2, \ldots$
$f(n) = 3n^2 + \mathcal{O}(n)$	Totes les anteriors, més $3n^2 + n, 3n^2 + 5\sqrt{n}, \ldots$
$f(n) = 3n^2 + o\big(n^2\big)$	Totes les anteriors, més $3n^2 + 1000n^{1.99}, \ldots$
$f(n) = \Theta\big(n^2\big)$	Totes les anteriors, més $-n^2 + 5, 100n^2, \ldots$
$f(n) = \mathcal{O}\big(n^2\big)$	Totes les anteriors, més $\sqrt{n}, -8, 0, \ldots$
$f(n) = o\big(n^3\big)$	Totes les anteriors, més $100n^3/\ln n, \ldots$

D'acord amb les convencions anteriors, segons les quals un conjunt de funcions definit amb $\mathcal{O}()$, $\Theta()$ o $o()$ es pot interpretar com qualsevol de les funcions del conjunt, podem reformular abreujadament la Proposició 5, tal i com es mostra a continuació. (Per exemple, la propietat $\Theta(\mathcal{O}(f)) = \mathcal{O}(f)$ s'ha d'interpretar com "les funcions que es poden obtenir escollint qualsevol funció g amb ordre de creixement com a molt igual a f, i després escollint qualsevol funció proporcional a g, són les mateixes que es poden obtenir escollint qualsevol funció amb ordre de creixement com a molt igual a f".)

Proposició 9. *Siguin $f, g : \mathbb{N} \longrightarrow \mathbb{R}$. Llavors les igualtats següents són certes en el sentit més estricte ("$=$" significa "$=$" i no pas "$\subseteq$" o "$\subset$"):*

- $\Theta(\Theta(f)) = \Theta(f)$,
- $\Theta(\mathcal{O}(f)) = \mathcal{O}(\Theta(f)) = \mathcal{O}(\mathcal{O}(f)) = \mathcal{O}(f)$,
- $\Theta(o(f)) = o(\Theta(f)) = \mathcal{O}(o(f)) = o(\mathcal{O}(f)) = o(o(f)) = o(f)$.

Les notacions $\mathcal{O}()$, $\Theta()$ i $o()$ presenten algunes propietats interessants respecte del producte i de la suma, expressades el la proposició següent utilitzant les Convencions 7 i 8.

Proposició 10. *Siguin $f, g : \mathbb{N} \longrightarrow \mathbb{R}$. Llavors les igualtats següents són certes en el sentit més estricte:*

- $\Theta(f) \cdot \Theta(g) = \Theta(f \cdot g)$,
- $\Theta(f) \cdot \mathcal{O}(g) = \mathcal{O}(f) \cdot \mathcal{O}(g) = \mathcal{O}(f \cdot g)$,
- $\Theta(f) \cdot o(g) = \mathcal{O}(f) \cdot o(g) = o(f) \cdot o(g) = o(f \cdot g)$,
- $\Theta(f) + \Theta(g) = \Theta(f + g) = \Theta(\mathit{Màx}\{f, g\})$,
- $\Theta(f) + \mathcal{O}(g) =^{(*)} \left\{ \begin{array}{ll} \Theta(f) & , \textit{ si } g = \mathcal{O}(f) \\ \mathcal{O}(g) & , \textit{ si } g \neq \mathcal{O}(f) \end{array} \right\} \subseteq \mathcal{O}(f + g)$,
- $\mathcal{O}(f) + \mathcal{O}(g) = \mathcal{O}(f + g) = \mathcal{O}(\mathit{Màx}\{f, g\})$,
- $\Theta(f) + o(g) =^{(*)} \left\{ \begin{array}{ll} \Theta(f) & , \textit{ si } g = \mathcal{O}(f) \\ o(g) & , \textit{ si } g \neq \mathcal{O}(f) \end{array} \right\} \subset \mathcal{O}(f + g)$,
- $\mathcal{O}(f) + o(g) =^{(*)} \left\{ \begin{array}{ll} \mathcal{O}(f) & , \textit{ si } g = \mathcal{O}(f) \\ o(g) & , \textit{ si } g \neq \mathcal{O}(f) \end{array} \right\} \subseteq \mathcal{O}(f + g)$,
- $o(f) + o(g) = o(f + g) = o(\mathit{Màx}\{f, g\})$.

(Parlant rigurosament, les tres igualtats marcades amb un asterisc només són certes per a les funcions que hom pot trobar a la pràctica.)

Les tres igualtats $\Theta(f + g) = \Theta(\mathit{Màx}\{f, g\})$, $\mathcal{O}(f + g) = \mathcal{O}(\mathit{Màx}\{f, g\})$, i $o(f + g) = o(\mathit{Màx}\{f, g\})$ es poden generalitzar fàcilment a qualsevol nombre de sumands, sempre i quan aquest nombre sigui constant (és a dir, no depengui d'n). Això es pot comprovar en el contraexemple següent:

$$\Theta(n) = \Theta\left(\sum_{1 \leq i \leq n} 1\right) \neq \Theta(\mathit{Màx}\{1, \ldots, 1\}) = \Theta(1)\,.$$

L'exemple següent mostra com es poden utilitzar les notacions i convencions asimptòtiques introduides en aquesta secció. (Fixem-nos que el pas "$\Theta\big(n^2\big) + o\big(n^2\big) = \Theta\big(n^2\big)$" es troba justificat per la Proposició 10.)

$$3n^2 + 3n - 6 = 3n^2 + \Theta(n) = 3n^2 + o\big(n^2\big) = \Theta\big(n^2\big) + o\big(n^2\big) = \Theta\big(n^2\big) = \mathcal{O}\big(n^3\big)\,,$$

ja que $3n - 6 \in \Theta(n)$, qualsevol funcio dins de $\Theta(n)$ ha de pertànyer a $o\big(n^2\big)$, $3n^2 \in \Theta\big(n^2\big)$, qualsevol funció a $\Theta\big(n^2\big)$ sumada a qualsevol funció a $o\big(n^2\big)$ es troba dins de $\Theta\big(n^2\big)$, i qualsevol funcio a $\Theta\big(n^2\big)$ ha de pertànyer a $\mathcal{O}\big(n^3\big)$.

Els ordres de creixement de les funcions més usuals segueixen l'ordenació implícita en la pròxima proposició.

Proposició 11. *Les propietats següents són certes:*

- Siguin $a, b \in \mathbb{R}$ tals que $a > 0$, $b > 1$. Llavors $1 = o((\log_b n)^a)$.
- Siguin $a, b, \beta \in \mathbb{R}$ tals que $a > 0$, $b > 1$, $\beta > 1$.
  Llavors $(\log_b n)^a = \Theta\big((\log_\beta n)^a\big)$.
- Siguin $a, b, \alpha, \beta \in \mathbb{R}$ tals que $\alpha > a > 0$, $b > 1$, $\beta > 1$.
  Llavors $(\log_b n)^a = o\big((\log_\beta n)^\alpha\big)$.
- Siguin $a, b, c \in \mathbb{R}$ tals que $a > 0$, $b > 1$, $c > 0$. Llavors $(\log_b n)^a = o(n^c)$.
- Siguin $c, \gamma \in \mathbb{R}$ tals que $\gamma > c > 0$. Llavors $n^c = o(n^\gamma)$.
- Siguin $c, d \in \mathbb{R}$ tals que $c > 0$, $d > 1$. Llavors $n^c = o(d^n)$.
- Siguin $d, \delta \in \mathbb{R}$ tals que $\delta > d > 1$. Llavors $d^n = o(\delta^n)$.
- Sigui $d \in \mathbb{R}$ tal que $d > 1$. Llavors $d^n = o(n!)$.
- $n! = o(n^n)$.

Segons la segona i la tercera de les propietats esmentades, la base dels logaritmes no és important quan s'usen les notacions $\mathcal{O}()$, $\Theta()$ i $o()$. Consistentment, podrem dir que una funció és $\mathcal{O}(\log n)$ en lloc d' $\mathcal{O}(\log_2 n)$, per exemple, sense especificar quina és la base del logaritme.

3 Complexitat dels algorismes

En aquesta secció veurem com es poden utilitzar les notacions matemàtiques introduïdes a la secció anterior per calcular el cost dels algorismes o procediments. Aquest cost s'expressa normalment en funció de la *grandària*, *talla* o *mida* $|\mathrm{x}|$ del problema a resoldre x. Aquesta noció de grandària normalment té una definició natural per a cada problema concret. Així, per exemple, la grandària de l'entrada d'un algorisme de cerca en taules o d'ordenació de taules s'identifica amb el nombre d'elements a la taula, i la grandària de l'entrada d'un algorisme de tractament de seqüències amb el nombre d'elements a la seqüència.

Definició 12. *Sigui $T_{\mathcal{A}}(\mathrm{x})$ el temps d'execució en un determinat ordinador de l'algorisme $\mathcal{A}$ quan la seva entrada és* x*. Definim el cost o complexitat en el cas pitjor de l'algorisme $\mathcal{A}$ com l'ordre de creixement (en funció de n):*

$$\Theta\big(\mathit{Màx}\{T_{\mathcal{A}}(\mathrm{x})\}_{|\mathrm{x}|=n}\big)\,.$$

Per a cada element x *de talla n, sigui $Prob(\mathrm{x})$ la probabilitat que té* x *d'ésser l'entrada de l'algorisme, condicionada al fet que l'entrada de l'algorisme té talla n. Definim el cost o complexitat en el cas mitjà de l'algorisme $\mathcal{A}$ com l'ordre de creixement (en funció de n):*

$$\Theta\left(\sum_{|\mathrm{x}|=n} \mathit{Prob}(\mathrm{x}) \cdot T_{\mathcal{A}}(\mathrm{x})\right).$$

Com ja hem vist en les seccions anteriors, el què ens interessa en molts casos és l'ordre de creixement de les funcions de cost, és a dir, saber si el cost d'un algorisme creix de manera proporcional a la grandària de l'entrada, o al seu quadrat, etcètera. Es per aquest motiu que hem utilitzat la notació $\Theta()$ en la definició del costs en els casos pitjor i mitjà, ignorant les constants de proporcionalitat. Això no obstant, a vegades computarem amb precisió el nombre d'algunes operacions elementals, sempre que aquesta informació sigui significativa i l'anàlisi sigui possible.

Per una altra banda, cal dir que el cas mitjà no sempre está ben definit, ja que hi ha vegades en què seria arbitrari suposar una certa distribució de probabilitats per a les possibles entrades de l'algorisme.

Finalment, cal dir que es pot definir el cost en el cas millor de manera anàloga al cost en el cas pitjor. El cost en el cas millor no es fa servir gaire, però, ja que normalment aporta poca informació sobre l'eficiència d'un algorisme.

A continuació es presenten unes quantes regles pràctiques per a l'avaluació del cost (en el cas pitjor o mitjà) dels algorismes. Alhora es donen exemples de com utilitzar aquestes regles per calcular el cost de la funció posició_mínim(), suposant que el tipus clau és de grandària constant respecte de n.

Regles pràctiques per l'avaluació de la complexitat d'un algorisme o procediment:

- *Les comparacions, operacions aritmètiques, increments, lectures o escriptures d'objectes de tipus elementals, i els accessos a components elementals d'una taula tenen cost* $\Theta(1)$*, o sigui, cost constant que no depèn de la grandària del problema a resoldre.*

  Així, per exemple, la comparació "i $< n$" de la línia (4), l'accés a la primera posició de la taula T de la línia (2), o l'accés a la i-èssima posició de la taula T de la línia (7), requereixen cost $\Theta(1)$ cadascun.

- *Avaluar una expressió té cost igual a la suma dels costs de les operacions que s'hi realitzen, inclosos els costs de les crides a les funcions, si n'hi ha.*

  En el nostre exemple, avaluar la part dreta de les assignacions de les línies (2) i (7) té cost $\Theta(1)$. Avaluar la part dreta de les assignacions de les línies (1), (3) i (6) té en canvi cost nul.

- *Per calcular el cost d'una assignació d'una expressió* E *a una variable* v*,*

  $$\text{v} := \text{E} \ ,$$

  cal sumar el cost de l'avaluació de E*, més* $\Theta(1)$ *si* v *és de tipus elemental, o bé* $\Theta(n)$ *si* v *és (o conté) una taula de la grandària del problema.*

  En el cas que ens ocupa, les assignacions de les línies (1), (2), (3), (6) i (7) tenen totes cost $\Theta(1)$.

- *Regla de les sumes: El cost d'una seqüència d'accions*

$$\mathrm{A}_1,\ \mathrm{A}_2,\ \ldots,\ \mathrm{A}_k$$

és senzillament

$$\sum_{1\leq i\leq k} C_{A_i}(n),$$

on $C_{A_i}(n)$ *denota el cost de* A_i*.*

Així doncs, el cost de l'execució de les tres primeres línies de la funció posició_mínim() és $\Theta(1)+\Theta(1)+\Theta(1)=\Theta(1)$, i el cost de cada execució de les línies (6) i (7) és $\Theta(1)+\Theta(1)=\Theta(1)$.

- *El cost d'una estructura alternativa amb una sola branca,*

$$\underline{\text{si}}\quad \mathrm{E}\quad \underline{\text{llavors}}\quad \mathrm{A}\ ,$$

es pot calcular com la suma del cost de l'avaluació de l'expressió booleana E*, més potser el cost de l'execució de l'única branca* A*. És a dir, en general, tenim l'expressió*

$$C_E(n)+\mathcal{O}(C_A(n))\,.$$

El cost del <u>si</u> de la línia (5) és doncs $\Theta(1)+\mathcal{O}(\Theta(1))=\Theta(1)+\mathcal{O}(1)=\Theta(1)$.

- *El cost d'una estructura alternativa amb dues branques,*

$$\underline{\text{si}}\quad \mathrm{E}\quad \underline{\text{llavors}}\quad \mathrm{A}_1$$
$$\underline{\text{sino}}\quad \mathrm{A}_2\ ,$$

està fitat superiorment per la suma del cost de l'avaluació de l'expressió booleana E*, més el cost de l'execució de la branca més costosa, sigui* A_1 *o* A_2*. En general, tenim l'expressió*

$$C_E(n)+\mathcal{O}(C_{A_1}(n)+C_{A_2}(n))\,.$$

Pel cas particular en què $C_{A_1}(n)=\Theta(f)$ *i* $C_{A_2}(n)=\Theta(f)$ *per una mateixa* f*, o sigui, quan seguint qualsevol de les dues branques el cost és proporcional a una mateixa funció, disposem d'una expressió pel cost més acurada,*

$$C_E(n)+\Theta(f)\,.$$

- *Regla dels productes: Per calcular el cost d'una estructura iterativa,*

$$\underline{\text{mentre}}\quad \mathrm{E}\quad \underline{\text{fer}}\quad \mathrm{A}\ ,$$

en principi cal considerar quantes iteracions es realitzen i quant costa cada una. Suposem que el nombre de vegades que s'executa el cos del <u>mentre</u>

és $I(n)$. Llavors l'expressió booleana E *s'avalua $I(n)+1$ cops, i el cost es pot expressar com*

$$(I(n)+1)C_E(n) + I(n)C_A(n),$$

on $C_E(n)$ i $C_A(n)$ són els costs de l'avaluació de l'expressió booleana E *i de l'execució del cos* A*, respectivament,* **a cada iteració**.

Per exemple, podem calcular el cost del mentre de la línia (4): La comparació "i $< n$" té cost $\Theta(1)$. Per la regla de les sumes, la complexitat del cos del mentre a qualsevol iteració és $\Theta(1) + \Theta(1) + \Theta(1) = \Theta(1)$. El nombre d'iteracions és $n = \Theta(n)$. Per tant, la complexitat total del mentre és $(\Theta(n) + 1)\Theta(1) + \Theta(n) \cdot \Theta(1) = \Theta(n)$.

En la gran majoria de situacions, en el cas pitjor el cos del mentre s'executarà com a mínim un cop, és a dir, tindrem $I(n) \geq 1$. Si és així, el cost es pot expressar simplement com

$$I(n)(C_E(n) + C_A(n)).$$

En el nostre exemple, el cost del mentre de la línia (4) és senzillament $\Theta(n)\,(\Theta(1) + \Theta(1)) = \Theta(n)$.

- *El cost de la instrucció de retorn de resultats d'una funció,*

  retorna E ,

  és la suma del cost de l'avaluació de l'expressió E *més el cost de copiar (assignar) el resultat.*

  Com que l'avaluació del valor retornat per la funció posició_mínim() en l'última línia té cost nul, només cal carregar cost $\Theta(1)$ pel retorn d'un resultat de tipus elemental.

- *El pas de paràmetres per referència té cost associat $\Theta(1)$.*

- *El pas de paràmetres per valor i la creació de variables locals tenen cost associat $\Theta(1)$, a no ser que siguin (o continguin) alguna taula de la grandària del problema; en aquest cas el cost és $\Theta(n)$.*

  Per tant, cal afegir cost $\Theta(1)$ als costs calculats fins el moment.

Tot plegat, per la regla de les sumes, el cost de la funció posició_mínim() és $\Theta(n)$. El càlcul ha estat tant senzill que ni tant sols hem hagut de definir quina de les complexitats hem mesurat (en el cas pitjor o en el cas mitjà), ja que en aquest cas coincideixen. Veurem que no ha de ser així forçosament.

La taula següent presenta els costs més comuns i alguns dels algorismes més coneguts per a cada cost.

Complexitat	Nom comú	Exemples
$\Theta(1)$	Constant	Cerca en taules de dispersió
$\Theta(\log n)$	Logarítmic	Cerca en arbres binaris, cerca dicotòmica
$\Theta(n)$	Linial	Cerca i recorregut seqüencials, mitjana
$\Theta(n \log n)$	Quasilinial	Quicksort, mergesort, heapsort
$\Theta(n^2)$	Quadràtic	Mètodes elementals d'ordenació
$\Theta(n^c)$, $c > 0$	Polinòmic	Problemes "tractables"
$\Theta(2^n)$	Exponencial	Problemes intractables, backtracking
$\Theta(n!)$	Factorial	Problemes "encara més intractables"

Els algorismes amb complexitat superpolinòmica no són utilitzables a la pràctica, excepte per a n's extremadament petites. Per contraposició, comunment es defineix com a problema tractable aquell que es pot resoldre en temps polinòmic. Cal matisar que aquesta divisió entre problemes tractables i intractables és certament discutible, ja que un algorisme amb complexitat $\Theta(n^{1000})$, per exemple, difícilment serà utilitzat a la pràctica.

4 Exemples de càlcul de la complexitat

Ara ja estem preparats per calcular el cost de molts algorismes i procediments. Per exemple, considerem la funció següent, la qual retorna la posició on es troba una clau a una taula, o -1 si la clau no hi és.

Precondició: { La taula T conté n claus a les posicions $0..n-1$, amb $n \geq 0$ }

enter posició(const taula de claus &T; enter n; clau x)

Postcondició: { posició(T, n, x) = pos $\Rightarrow$ $(0 \leq \text{pos} < n \wedge \text{T[pos]} = \text{x})$
$\vee(\text{pos} = -1 \wedge (\forall\ 0 \leq \text{i} < n : \text{T[i]} \neq \text{x}))$ }

```
(1)     i := 0
(2)     trobat := fals
(3)     mentre i < n i no trobat
(4)         si T[i] = x
(5)             trobat := cert
              sino
(6)             ++i
(7)     si no trobat llavors i := -1
(8)     retorna i
```

Calculem-ne el cost en el cas pitjor. El pas de paràmetres, la creació de variables locals i les línies (1), (2), i (8) clarament tenen cost $\Theta(1)$. La línia (7) també, ja que realitza una comparació, més, possiblement, una assignació. Considerem el cost del mentre. Tant la línia (5) com la (6) tenen cost constant. Per tant, el cost del si de la línia (4) és $\Theta(1) + \Theta(1) = \Theta(1)$ —cost constant de l'avaluació de la condició, més cost constant de seguir qualsevol de les dues branques, sigui quina sigui. Per calcular el cost en el cas pitjor del mentre de la línia (3), cal

conèixer el cost de cada iteració i el nombre d'iteracions. Una iteració costa $\Theta(1)$, ja que s'avalua la condició del mentre (cost constant) i s'executa el si (també cost constant). En el cas pitjor, la variable "i" pot arribar a valer $n-1$ (o n), així que el nombre d'iteracions és $\Theta(n)$. Per tant, el cost en el cas pitjor del mentre és $\Theta(n) \cdot \Theta(1) = \Theta(n)$, i el cost en el cas pitjor de la funció posició() també.

Cal dir que en molts casos no cal realitzar una anàlisi tant detallada per conèixer el cost d'un procediment. Per exemple, per calcular el cost en el cas pitjor de la funció posició(), n'hi hauria prou amb adonar-nos que es recorre la taula fins a trobar la clau, o fins a arribar al final de la taula. Si la clau es troba a l'última posició o no hi és, llavors cal visitar n posicions, i a cada posició el cost de les operacions és constant. Per tant, el cost en el cas pitjor de la funció posició() és $\Theta(n)$.

El càlcul del cost en el cas pitjor de la funció següent és força il·lustratiu. La funció compta quants elements positius tenen en comú dues taules donades.

Precondició: { Les taules V i T contenen n claus a les posicions $0..n-1$ cadascuna, amb $n \geq 0$ }

enter nombre_positius_comuns(const taula de claus &V, &T; enter n)

Postcondició: { nombre_positius_comuns(V, T, n) $= \text{Card}\{\text{V[i]}\}_{0 \leq \text{i} < n \wedge \text{V[i]} \geq 0 \wedge \text{V[i]} \in \text{T}}$ }

```
(1)     c := 0
(2)     per i en [1..n]
(3)         si V[i] ≥ 0
(4)             si posició(T, n, V[i]) ≠ -1
(5)                 ++c
(7)     retorna c
```

L'única contribució al cost major que $\Theta(1)$ pot venir del per de la línia (2). Aquest s'executa n vegades, així que només ens falta saber el cost de cada iteració. Per una banda, el per de la línia (2) suposa implícitament l'avaluació de la condició "i $< n$" al principi de cada iteració, i l'execució de l'increment "++i" al final de cada iteració. Ambdues operacions tenen cost $\Theta(1)$, i el nombre d'iteracions és $\Theta(n)$. El cost del per sense les línies (3), (4) i (5) és per tant $\Theta(n) \cdot \Theta(1) = \Theta(n)$.

Ara ens cal computar la complexitat del cos del per. Ja hem vist que la funció posició() té cost $\Theta(n)$ en el cas pitjor. Per tant, podríem estar temptats de dir que, en el cas pitjor de la funció nombre_positius_comuns(), cada crida a la funció posició() té cost $\Theta(n)$. Peró això només és cert si podem garantir que existeix una situació en la qual *cada crida* a posició() té cost $\Theta(n)$. Sense haver trobat aquesta situació, podria passar que algunes crides tinguessin cost $\Theta(n)$ però d'altres cost $o(n)$, i l'únic que es pot afirmar és que el cost de la funció posició() és $\mathcal{O}(n)$ a qualsevol crida. Així doncs, una anàlisi poc acurada (però correcta) ens diu que el cost del si de la línia (4) és $\mathcal{O}(n) + \mathcal{O}(\Theta(1)) = \mathcal{O}(n)$,

el cost del si de la línia (3) és $\Theta(1) + \mathcal{O}(\mathcal{O}(n)) = \mathcal{O}(n)$ —en algunes iteracions es fa una crida amb cost com a molt linial—, i el cost del per de la línia (2) és $\Theta(n) + \Theta(n) \cdot \mathcal{O}(n) = \mathcal{O}\big(n^2\big)$.

Per fer un càlcul més precís del cost en el cas pitjor hem de caracteritzar una situació en la qual el cost sigui el màxim possible. Per exemple, podem considerar què succeeix quan la taula V conté només elements positius. En aquest cas, per a cada una de les n claus de V cal fer una crida a la funció posició(). Però amb això no n'hi ha prou per obtenir una complexitat més elevada que la del càlcul anterior, ja que el cost del si de la línia (3) seguiria sent $\Theta(1) + \Theta(\mathcal{O}(n)) = \mathcal{O}(n)$, i el del per seria $\Theta(n) + \Theta(n) \cdot \mathcal{O}(n) = \mathcal{O}\big(n^2\big)$.

També podem calcular el cost suposant només que cap element de la taula V es troba present a la taula T. En aquest cas el cost del si de la línia (4) és $\Theta(n)+0 = \Theta(n)$, però el cost del si de la línia (3) seguiria sent $\Theta(1)+\mathcal{O}(\Theta(n)) = \mathcal{O}(n)$, i el del per seguiria sent $\Theta(n) + \Theta(n) \cdot \mathcal{O}(n) = \mathcal{O}\big(n^2\big)$.

Finalment, considerem el cost quan la taula V conté únicament elements positius que no es troben a la taula T. En aquest cas, per a cada una de les n claus de V es fa una crida a la funció posició(), i es recorre T fins al final. Per tant, el cost del si de la línia (4) és $\Theta(n) + 0 = \Theta(n)$, el cost del si de la línia (3) és $\Theta(1) + \Theta(\Theta(n)) = \Theta(n)$, i el cost del per és ara $\Theta(n) + \Theta(n) \cdot \Theta(n) = \Theta\big(n^2\big)$.

Una imprecisió habitual quan es parla del cost d'un algorisme o procediment és dir "el cost és $\mathcal{O}(f)$", ja que la frase és inherentment ambigua. Fixem-nos que pot voler significar que el cost en el cas pitjor és proporcional a f, o bé que no som capaços de fer un càlcul més precís del cost en el cas pitjor o mitjà, i que només podem donar-ne una fita superior. Per desfer l'ambigüitat, en el primer cas direm "el cost en el cas pitjor és $\Theta(f)$", i en el segon cas direm "el cost en el cas (pitjor/mitjà) és $\mathcal{O}(f)$".

Per exemple, acabem de realitzar quatre anàlisis, tots diferents, però tots correctes, del cost en el cas pitjor de la funció nombre_positius_comuns(). En els tres primers casos hem conclòs que era $\mathcal{O}\big(n^2\big)$, en l'últim hem estat més acurats i hem conclòs que era $\Theta\big(n^2\big)$. La frase informal "el cost és $\mathcal{O}\big(n^2\big)$" podria voler descriure tant els tres primers casos (el cost en el cas pitjor no pot ser superior a $\Theta\big(n^2\big)$, però potser és inferior) com l'últim (el cost en el cas pitjor és proporcional a n^2).

Finalment, també podrem dir que el cost d'un algorisme és $\Theta(f)$, indicant que qualsevol cost (millor, pitjor o mitjà) és proporcional a f, o el què és equivalent, que qualsevol execució de l'algorisme requereix temps proporcional a f. Per exemple, la funció posició_mínim() té cost $\Theta(n)$. En canvi, seria inexacte dir que la funció posició() té cost $\Theta(n)$, ja que en el cas millor només costa $\Theta(1)$.

A continuació es presenta una anàlisi del cost en el cas pitjor de la funció nombre_positius_comuns(), suposant que sabem que hi ha només $\Theta(\sqrt{n}\,)$ nombres positius a la taula V. Un primer càlcul (imprecís), aniria així: Suposem el cas pitjor en què cap element de la taula V es troba present a la taula T. Llavors el cost del si de la línia (4) és $\Theta(n) + 0 = \Theta(n)$, el cost del si de la línia (3) és $\Theta(1) + \mathcal{O}(\Theta(n)) = \mathcal{O}(n)$ —no sempre es fa una crida a la funció posició()—, i el cost del per de la línia (2) és $\Theta(n) + \Theta(n) \cdot \mathcal{O}(n) = \mathcal{O}\big(n^2\big)$.

A vegades, però, es poden realitzar càlculs més exactes si es consideren per separat quants cops s'executa cadascuna de les possibles branques dins d'una estructura iterativa. En el nostre cas, és fàcil analitzar el cost més acuradament si mesurem (apart del cost constant del pas de paràmetres, de les instruccions que s'executen un sol cop, etcètera) les contribucions al cost dels elements positius de V i del elements negatius de V per separat. Per als primers, tenim que $\Theta(\sqrt{n}\,)$ vegades es fa una crida a la funció posició(), la qual en el cas pitjor té cost $\Theta(n)$. Així doncs, la contribució al cost d'aquests elements és $\Theta(\sqrt{n}\,)\cdot\Theta(n) = \Theta(n\sqrt{n}\,)$. Per altra banda, els $n-\Theta(\sqrt{n}\,) = \Theta(n)-o(n) = \Theta(n)$ elements negatius aporten cost constant cadascun, ja que no generen cap crida a la funció posició(). La seva contribució és, per tant, $\Theta(n)\cdot\Theta(1) = \Theta(n)$. El cost en el cas pitjor és doncs només $\Theta(1)+\Theta(n\sqrt{n}\,)+\Theta(n) = \Theta(n\sqrt{n}\,)$.

Per acabar amb aquest exemple, també podríem analitzar el cost acuradament si tenim la informació que només hi ha $\mathcal{O}(1)$ elements positius a la taula V. Ara, la contribució dels elements positius al cost en el cas pitjor és $\mathcal{O}(1)\cdot\Theta(n) = \mathcal{O}(n)$. El nombre d'elements negatius és $n-\mathcal{O}(1) = \Theta(n)-o(n) = \Theta(n)$, i la seva contribució és $\Theta(n)\cdot\Theta(1) = \Theta(n)$. El cost en el cas pitjor és ara tant sols $\Theta(1)+\mathcal{O}(n)+\Theta(n) = \Theta(n)$, és a dir, el mínim possible de qualsevol algorisme de recorregut d'una seqüència.

Acabem de veure que per calcular el cost en el cas pitjor cal identificar precisament quin o quins són els casos pijors. En canvi, a l'hora d'avaluar el cost en el cas mitjà, el què cal és definir amb precisió la distribució de probabilitats sobre les possibles entrades de l'algorisme.

Considerem, per exemple, la funció posició(). Podríem suposar (potser arbitràriament) que els n elements de la taula es troben en qualsevol ordre amb la mateixa probabilitat (és a dir, que les $n!$ permutacions són equiprobables), i que busquem un element escollit a l'atzar que hi és segur. El cost mitjà de trobar l'element es podria calcular rigurosament com segueix. Sigui $\mathcal{P}$ el conjunt de les $n!$ possibles permutacions de les claus de la taula T. Per a tota permutació $p \in \mathcal{P}$, denotem amb $I(p)$ i amb $C(p)$ el nombre d'iteracions del mentre i el cost total de trobar el mínim, respectivament, quan la permutació dins la taula T és p. Llavors $C(n)$, el cost en el cas mitjà de la funció posició(), es pot expressar com

$$C(n) = \sum_{p\in\mathcal{P}} Prob(p)\cdot C(p) = \sum_{p\in\mathcal{P}} \frac{1}{n!}\big(\Theta(1)+I(p)\cdot\Theta(1)\,\big),$$

ja que les instruccions fora del mentre (inclosa l'última avaluació de la condició " i $< n$ i no trobat", la que fa que el mentre acabi) tenen en total cost $\Theta(1)$, i cada iteració del mentre costa $\Theta(1)$. Treient els factors comuns fora del sumatori, podem expressar $C(n)$ com

$$C(n) = \Theta(1) + \Theta\left(\sum_{p\in\mathcal{P}} \frac{I(p)}{n!}\right). \tag{1}$$

Ara ens cal avaluar la suma. Per això n'hi ha prou amb adonar-nos que per a $(n-1)!$ permutacions (aquelles que comencen amb l'element cercat) el nombre d'iteracions només és 1, que per a $(n-1)!$ permutacions (aquelles amb l'element cercat a la segona posició) tenim $I(n) = 2$, etcètera. Per tant,

$$C(n) = \Theta(1) + \Theta\left(\sum_{1\le i\le n} \frac{(n-1)!\cdot i}{n!}\right)$$

$$= \Theta(1) + \frac{1}{n}\cdot\Theta\left(\sum_{1\le i\le n} i\right) = \Theta(1) + \frac{1}{n}\cdot\Theta\left(\frac{n(n+1)}{2}\right) = \Theta(n)\,.$$

També podríem haver fet el càlcul molt més informalment, però de manera igualment correcta, com segueix. Les instruccions fora del mentre tenen en total cost $\Theta(1)$. Cada iteració del mentre costa $\Theta(1)$. Només ens falta saber quantes iteracions es fan en mitjana. Com que les permutacions són aleatòries, la posició esperada de l'element que busquem és $(n+1)/2$. Per tant el nombre esperat d'iteracions és $\Theta(n)$, i el cost mitjà és també $\Theta(n)$.

Fixem-nos que aquest últim càlcul es pot resumir així: en aquest exemple, el cost mitjà $C(n)$ és igual al nombre esperat d'iteracions del mentre, i mai inferior a $\Theta(1)$. Això és exactament el què ens diu l'Equació (1).

5 Exercicis

1) Ordeneu les funcions següents per ordre asimptòtic de creixement (algunes tenen el mateix ordre):

$$0, 8, n, n^2, \sqrt{n^2+n}, 1/n, \log_2 n, \ln n, (\ln n)^2, \ln(n^2),$$

$$(\ln n)/n, \ln(n/\ln n), n\ln n, \log_2(\ln n), 1+1/\ln n,$$

$$2^n, 2^{-n}, 2^{2n}, 3^n, 4^n, n2^n, n^{1+1/n}, n!, \ln(n!), n^n, \ln(n^n), 2^{\ln n}, 2^{\sqrt{n}}, 2^{\sqrt{\log_2 n}}.$$

2) Des d'un punt de vista matemàtic rigurós, no és cert que la Definició 1 del conjunt $\mathcal{O}(g)$ sigui equivalent a dir que una funció f pertany a $\mathcal{O}(g)$ sii $0 \le \lim_{n\to\infty} |f(n)/g(n)| < \infty$. Per exemple, considerem les funcions f i g definides com $g(n) = n$, $f(2n) = 6n$, $f(2n+1) = 7$. és cert que $f \in \mathcal{O}(g)$? I que $0 \le \lim_{n\to\infty} |f(n)/g(n)| < \infty$? Podeu donar algun exemple de parells de funcions f i g tals que $f \notin \mathcal{O}(g)$ i $g \notin \mathcal{O}(f)$?

3) Tampoc és estrictament cert que la Definició 2 del conjunt $\Theta(g)$ sigui equivalent a dir que f pertany a $\Theta(g)$ sii $0 < \lim_{n\to\infty} |f(n)/g(n)| < \infty$, ni que la Definició 3 del conjunt $o(g)$ sigui equivalent a dir que f pertany a $o(g)$ sii $f \in \mathcal{O}(g)$ però $g \notin \mathcal{O}(f)$. Podeu donar-ne alguns exemples?

4) Suposem que es disposa de cinc algorismes diferents per resoldre el mateix problema, i que el temps que triga cadascun ve definit, respectivament, per les

fórmules $T_1(n) = 1000n$, $T_2(n) = 100n \log_2 n$, $T_3(n) = 10n^2$, $T_4(n) = n^3$ i $T_5(n) = 2^n/100$. Per a cada valor d'n, quin algorisme farieu servir?

5)En la funció nombre_positius_comuns(), les dues condicions "V[i] $\geq$ 0" i "posició(T, n, V[i]) $\neq$ -1" s'han separat expressament per explicitar l'ordre en que s'avaluen, ja que assumirem que l'ordre d'avaluació de dues o més operacions amb la mateixa prioritat és indeterminat. Ara bé, com a norma general, sembla convenient realitzar les comprovacions menys costoses abans que les més cares. Aquesta intuició es pot quantificar en aquest exemple si considerem la funció nombre_positius_comuns() amb la modificació següent.

```
(3)     ...
(4)         si posició(T, n, V[i]) ≠ -1
(5)             si V[i] ≥ 0
(6)                 ...
```

Calculeu-ne el cost en el cas pitjor suposant que el nombre d'elements positius a la taula V és indeterminat, és $\Theta(\sqrt{n}\,)$ i és $\mathcal{O}(1)$. Compareu els resultats amb els calculs fets anteriorment per a la mateixa funció a la Secció 4. Per a les dues versions de la funció nombre_positius_comuns(), generalitzeu el càlcul suposant que hi ha $f(n)$ elements positius a la taula V.

6) Detecteu on és l'error en el raonament següent: "Les taules V i T tenen una capacitat màxima (anomenem-la MAX) cadascuna. Per tant, el cost en cas pitjor de la funció nombre_positius_comuns() és $\mathcal{O}\big(\mathrm{MAX}^2\big)$, ja que $n \leq$ MAX. Com que MAX és una constant, el cost en cas pitjor és senzillament $\mathcal{O}(1)$."

7) Suposem que el cost d'un algorisme en el cas mitjà és $\Theta(n)$, i que la variança del temps d'execució és moderada (és a dir, que diverses execucions requereixen temps semblants). Si l'algorisme triga temps t per resoldre un problema de mida n (suficientment gran), quant esperarieu que trigués amb una mida $2n$? I amb una mida $10n$? Feu els mateixos càlculs suposant que el cost de l'algorisme en el cas mitjà és $\Theta\big(n^2\big)$, $\Theta\big(n^3\big)$ i $\Theta(2^n)$.