Cost dels algorismes

Programació 2

Abril 2016

1 Motivació

La primera condició que s'ha d'exigir a un algorisme és que sigui correcte, és a dir, que resolgui satisfactòriament el problema plantejat, però també és natural demanar-li una sèrie de condicions addicionals com ara la llegibilitat, la modularitat, la reusabilitat o un bon cost de desenvolupament, sense oblidar l'eficiència. Tot algorisme fa ús d'una sèrie de **recursos**, fonamentalment el temps de càlcul i l'espai de memòria (que abreujarem com **temps** i **espai**). Parlarem, doncs, d'**eficiència** o **complexitat** d'un algorisme per referir-nos al bon ús que fa dels recursos de temps i d'espai. En particular, quan no es diu res sobre la mena d'eficiència, s'entendrà eficiència en temps.

Hi ha diversos motius pels quals existeixen algorismes ineficients. D'una banda, hi ha el compromís entre temps i espai, que ja era ben conegut en els inicis de la Informàtica: molt sovint es pot dissenyar un algorisme ràpid a condició que consumeixi força memòria; si, en canvi, volem que faci servir poca memòria, augmentarà la seva complexitat en temps. Actualment, la disponibilitat de grans quantitats de memòria fa que aquesta sigui una raó de menys pes per a la ineficiència (en temps), però encara queden altres raons que poden empitjorar el temps d'execució d'un algorisme. Pot passar que existeixin algorismes més eficients per al nostre problema però que siguin difícils de trobar o de programar, però també pot passar que hi hagi dubtes sobre si el nostre problema admet solucions més eficients o simplement no existeixen. Un exemple del segon tipus és la factorització de naturals, sobre la qual no se sap si existeixen solucions molt més eficients que les actuals però, en cas que es trobessin, obligarien a revisar la seguretat a internet. En qualsevol cas, l'eficiència és un aspecte fonamental dels algorismes per al qual convé disposar de notació i conceptes que permetin fer afirmacions acurades.

2 Cost d'un algorisme

Abans d'establir la noció del temps d'execució d'un programa, sorgeixen unes quantes preguntes. En quina màquina el mesurarem? Amb quin compilador? I fins i tot, amb quines entrades? Respecte de l'última pregunta, és ben possible que un mateix programa trigui temps molt diferent en funció de l'entrada concreta i no de la mida (o grandària, o talla) de l'entrada, com és el cas de l'algorisme d'ordenació per inserció.

```
// Pre: cert
// Post: v conté els elements inicials
         i està ordenat creixentment
(1) void ordena_per_insercio(vector<double>& v) {
          // Inv: v[0..i-1] està ordenat creixentment
(2)       for (int i = 1; i < v.size(); ++i) {
(3)           double x = v[i];
(4)           int j = i;
(5)           while (j > 0 and v[j - 1] > x) {
(6)               v[j] = v[j - 1];
(7)               --j;
(8)           }
(9)           v[j] = x;
(10) }    }
```

Figura 1: Algorisme d'inserció

És fàcil comprovar que si el vector d'entrada està ordenat de forma creixent, l'algorisme no executarà el bucle de les línies 5-8 i que, en canvi, si l'ordenació és decreixent, el mateix bucle s'executarà el màxim nombre de vegades. Per tant, malgrat que la mida de l'entrada sigui idèntica, els dos casos d'ordenació esmentats fan que l'algorisme tinguin un comportament molt diferent pel que fa al temps d'execució. Per aquest motiu es pot analitzar el temps que triga un algorisme per a les entrades d'una mateixa mida de tres maneres diferents:

- en el *cas millor*, el que fa que l'algorisme trigui el mínim temps possible (en l'algorisme anterior, un vector ordenat creixentment)
- en el *cas mitjà*, que té en compte la probabilitat de cada possible entrada
- en el *cas pitjor*, el que fa que l'algorisme trigui el màxim temps possible (en l'algorisme anterior, un vector ordenat decreixentment)

L'**anàlisi en el cas pitjor** té l'avantatge de ser habitualment més fàcil de fer i de donar la seguretat que no se sobrepassarà un cert llindar d'ineficiència, de manera que aquí ens centrarem en aquesta mena d'anàlisi de cost.

Donada una entrada x, representarem amb $n = |x|$ la mida de x. A la pràctica, el concepte de **mida de l'entrada** dependrà de quina mena de dades estiguem tractant. En el cas dels vectors —com en els algorismes d'ordenació—, la mida de l'entrada serà el nombre d'elements del vector. En el dels nombres —com ara un algorisme per decidir la primalitat—, serà el nombre de dígits necessari per representar el nombre (que és logarítmic en el valor) i, en general, es pot prendre com a mida de l'entrada el nombre de símbols necessaris per descriure-la —per exemple, en els algorismes que tracten text.

Definició 1. Sigui $T_{\mathcal{A}}(x)$ el temps d'execució de l'algorisme $\mathcal{A}$ en un determinat **entorn** (que entenem com tot allò que permet l'execució: ordinador, llenguatge, compilador, etc.) quan la seva entrada és x. Definim la *funció de cost en el cas pitjor* (aquí, simplement **funció de cost**) de l'algorisme $\mathcal{A}$ com

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

Fixem-nos que, segons la definició anterior, la funció de cost en el cas pitjor de l'algorisme d'inserció seria una funció que tindria, per a cada n, un valor equivalent al cas en què els n nombres d'entrada estiguessin ordenats de forma decreixent. Si per a diferents entorns possibles féssim gràfiques del temps en funció de la mida de l'entrada, veuríem que les gràfiques obtingudes serien força semblants i, de fet, correspondrien totes a una funció quadràtica. Ens interessa, doncs, per poder-nos abstreure de l'entorn concret en què s'executarà un algorisme, disposar d'una notació que no consideri significatives les diferències constants ni lineals entre diferents funcions de cost.

Exemple 2. Considerem la funció següent:

```
(1)    int valors_no_nuls(vector<int>& v) {
(2)        int n = v.size();
(3)        int k = 0;
(4)        for (int i = 0; i < n; ++i)
(5)            if (v[i] != 0)
(6)                ++k;
(7)        return k;
(8)    }
```

Per expressar el temps d'execució en termes de la mida de l'entrada $n =$ `v.size()`, assumirem que cada instrucció elemental té un cost constant. Com

que ens interessa el cost en el cas pitjor, sumem el cost d'execució de les línies 5-6, C_4, per a cada iteració del bucle, malgrat que la línia 6 no s'executarà quan es compleixi `v[i] = 0`. Al resultat només faltarà sumar-li els costos constants del pas del paràmetre (línia 1), C_1, de les assignacions (línies 2-3), C_2, i del retorn del resultat (línia 7), C_3:

$$C_1 + C_2 + C_3 + \sum_{i=0}^{n-1} C_4 = C_1 + C_2 + C_3 + C_4 \cdot n.$$

Si ara executéssim aquest algorisme en dos entorns diferents, l'únic que variaria seria el valor de les constants additives (C_1, C_2, C_3) i de la multiplicativa (C_4), però continuaria sent una funció lineal en n. Necessitem, doncs, una notació matemàtica que expressi el cost en funció de la mida de l'entrada i que ho faci indicant la **taxa de creixement** (lineal, en el cas d'aquest exemple) de la funció de cost en lloc de la funció de cost concreta.

3 Notació asimptòtica

La notació asimptòtica que veurem en aquesta secció ens permetrà fer afirmacions generals sobre l'eficiència d'un algorisme en el cas pitjor donant una funció que representi una fita superior. El fet de demanar només una fita superior té dos avantatges: d'una banda, assegura que no sobrepassarem un cert cost i, de l'altra, la seva obtenció no requerirà habitualment de càlculs complexos. En el cas de l'exemple 2, volem expressar que la funció `valors_no_nuls()` és lineal en la mida de l'entrada n sense fer referència als paràmetres de la funció lineal (és a dir, a constants additives o multiplicatives). Això ho aconseguim amb la notació $\mathcal{O}()$, anomenada de l'**O gran**, que és part d'un conjunt més ampli de notacions anomenades *asimptòtiques*. Per a la resta d'aquest document, suposarem que totes les funcions són $\mathbb{N} \to \mathbb{R}^+$ i contínues.

Definició 3. Donada una funció g, definim $\mathcal{O}(g)$ com el conjunt de funcions f tals que

$$0 \leq \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty.$$

Intuïtivament, $\mathcal{O}(g)$ és el conjunt de funcions que **asimptòticament** (és a dir, en el límit) creixen més lentament o igual de ràpid que g.

Convenció 4. Si $f \in \mathcal{O}(g)$, també s'afirma que f és $\mathcal{O}(g)$ i que $f = \mathcal{O}(g)$.

Segons la convenció anterior podem escriure, per exemple, que

$$n^2 + 7 = \mathcal{O}(n^2) \subseteq \mathcal{O}(n^3),$$

la qual cosa és fàcil de demostrar aplicant la definició. En el cas de les funcions $f(n) = \log n^2$ i $g(n) = \log^2 n$ (és a dir, $(\log n)^2$) podem començar observant, fent ús de les propietats dels logaritmes, que $f(n) = 2 \log n$ i, per tant,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{2\log n}{\log^2 n} = \lim_{n\to\infty} \frac{2}{\log n} = 0.$$

Així doncs, podem dir que f és $\mathcal{O}(g)$.

La notació asimptòtica de l'O gran compleix les propietats següents (podeu trobar-ne una demostració a l'apèndix).

Proposició 5. *Donades les funcions f, g, f_1, f_2, g_1, g_2, es compleixen les propietats següents:*

1. *Reflexivitat: $f \in \mathcal{O}(f)$*
2. *Transitivitat: si $f \in \mathcal{O}(g)$ i $g \in \mathcal{O}(h)$, llavors $f \in \mathcal{O}(h)$*
3. *Criteri de caracterització: $f \in \mathcal{O}(g)$ si i només si $\mathcal{O}(f) \subseteq \mathcal{O}(g)$*
4. *Invariància multiplicativa: $\mathcal{O}(f) = \mathcal{O}(c \cdot f)$ per a tot $c \in \mathbb{R}$*
5. *Regla de la suma: si $f_1 \in \mathcal{O}(g_1)$ i $f_2 \in \mathcal{O}(g_2)$, llavors $f_1 + f_2 \in \mathcal{O}(g_1 + g_2)$*
6. *Caracterització de la suma: $\mathcal{O}(f + g) = \mathcal{O}(\max(f, g))$*
7. *Regla del producte: si $f_1 \in \mathcal{O}(g_1)$ i $f_2 \in \mathcal{O}(g_2)$, llavors $f_1 \cdot f_2 \in \mathcal{O}(g_1 \cdot g_2)$*
8. *Invariància additiva: $\mathcal{O}(f) = \mathcal{O}(c + f)$ per a tot $c \in \mathbb{R}$ i tota funció f tal que $\lim_{n\to\infty} f(n) > 0$*

Les dues primeres propietats de la proposició 5 caracteritzen l'O gran com un ordre parcial sobre les funcions. Introduïm la notació següent per referir-nos-hi més còmodament.

Convenció 6. És habitual fer servir expressions amb notació asimptòtica. Per exemple, $\mathcal{O}(f) \cdot \mathcal{O}(g)$ representa el producte de qualsevol parell de funcions que pertanyen, respectivament, a $\mathcal{O}(f)$ i a $\mathcal{O}(g)$. Com que, per la regla del producte, el resultat serà $\mathcal{O}(f \cdot g)$, escriurem senzillament $\mathcal{O}(f) \cdot \mathcal{O}(g) = \mathcal{O}(f \cdot g)$.

La definició de l'O gran i el criteri de caracterització ens permeten identificar quan dues funcions tenen la mateixa taxa de creixement.

Observació 7. Si f i g són dues funcions tals que $0 < \lim_{n\to\infty} f(n)/g(n) < \infty$, llavors $\mathcal{O}(f) = \mathcal{O}(g)$.

Les propietats següents corresponen a casos de funcions molt habituals en el càlcul de costos (en podeu trobar les demostracions a l'apèndix).

Proposició 8. *Si $p(n)$ és un polinomi de grau k, llavors $\mathcal{O}(p) = \mathcal{O}(n^k)$.*

Proposició 9. *Per a qualsevol constant $c > 1$,*

$$\mathcal{O}(1) \subsetneq \mathcal{O}(\log n) \subsetneq \mathcal{O}(n) \subsetneq \mathcal{O}(n \log n) \subsetneq \mathcal{O}(n^2) \subsetneq \mathcal{O}(n^3) \subsetneq \cdots \subsetneq \mathcal{O}(c^n).$$

4 Càlcul del cost

La notació asimptòtica vista en la secció anterior ens permet expressar d'una manera molt còmoda el cost d'un algorisme. En el cas concret de l'exemple 2 s'obtenia un cost de $C_1+C_2+C_3+C_4 \cdot n$, però fent ús de la notació asimptòtica podrem eliminar les constants C_1, ..., C_4 per referir-nos estrictament a la taxa de creixement, que és de $\mathcal{O}(n)$. Les regles següents ens permetran calcular el cost asimptòtic sense necessitat d'introduir constants prèviament:

- **Operacions elementals.** Una operació elemental té cost $\mathcal{O}(1)$, és a dir, un cost que no depèn de la mida de l'entrada i que, per tant, anomenem constant. Es pot considerar com operacions elementals les lectures i escriptures, els accessos a components elementals d'un vector, les operacions aritmètiques senzilles o les comparacions i assignacions de components elementals.

- **Composició seqüencial.** Si el cost d'un fragment de codi `F1` és $\mathcal{O}(f_1)$ i el d'un fragment `F2` és $\mathcal{O}(f_2)$, llavors el cost del fragment

  ```
  F1; F2;
  ```

  és $\mathcal{O}(f_1) + \mathcal{O}(f_2)$, que és igual a $\mathcal{O}(f_1 + f_2)$ per la regla de la suma i, per consegüent, a $\mathcal{O}(\max(f_1, f_2))$ per la caracterització de la suma.

- **Composició alternativa.** Si el cost de `F1` és $\mathcal{O}(f_1)$, el de `F2` és $\mathcal{O}(f_2)$ i el d'avaluar una condició `B` és $\mathcal{O}(f_0)$, aleshores el cost del fragment

  ```
  if (B) F1; else F2;
  ```

  serà de $\mathcal{O}(f_0) + \mathcal{O}(\max(f_1, f_2))$ (el màxim proporciona el cas pitjor), que és igual a $\mathcal{O}(\max(f_0, f_1, f_2))$ per la regla i la caracterització de la suma. El fragment sense `else` té un cost equivalent a l'anterior tenint en compte que $f_2 \in \mathcal{O}(1)$.

- **Composició iterativa.** En general, cal multiplicar el nombre màxim d'iteracions pel cost de cada iteració i aplicar la regla del producte. En el cas d'un fragment com

  ```
  while (B) F;
  ```

  en el qual es fa un màxim de h iteracions, el cost de `F` és $\mathcal{O}(f_i)$ durant la i-èsima iteració i el d'avaluar la condició `B` durant la i-èsima iteració és $\mathcal{O}(b_i)$, aleshores el `while` tindrà cost $h \cdot \mathcal{O}(\max_{1 \leq i \leq h}(b_i, f_i) + b_{h+1}) = \mathcal{O}(h \cdot (\max_{1 \leq i \leq h}(b_i, f_i) + b_{h+1}))$ per la regla del producte. En el cas en què tots els b_i són iguals a un valor b i tots els f_i a un valor f, o bé quan b i f són fites superiors útils de tots els valors b_i i f_i, respectivament, podem expressar el cost més simplement com $\mathcal{O}(h \cdot \max(b, f))$.

  Si la composició és de la forma

  ```
  for (int i = 0; i < h; ++i) F;
  ```

  i $\mathcal{O}(f_i)$ és el cost de `F` en la iteració i-èsima, llavors el cost del `for` és $h \cdot \mathcal{O}(\max_{1 \leq i \leq h}(f_i)) = \mathcal{O}(h \cdot \max_{1 \leq i \leq h}(f_i))$ per la regla del producte.

- **Retorn de resultats.** El cost del retorn de resultats d'una funció

  ```
  return E;
  ```

  és la suma del cost d'avaluar l'expressió `E` més el de copiar el resultat.

- **Pas de paràmetres.** Si és per referència, té un cost associat $\mathcal{O}(1)$. Si és per valor, té un cost $\mathcal{O}(1)$ més el de la mida del paràmetre (que és $\mathcal{O}(1)$ per a objectes elementals, $\mathcal{O}(n)$ si té la mida de l'entrada, etc.)

Convenció 10. Es destaca en cursiva la terminologia habitual per descriure el cost d'algorismes amb diferents fites asimptòtiques. Cada cas va seguit d'un exemple.

- $\mathcal{O}(1)$, cost *constant*: suma de dues variables numèriques.
- $\mathcal{O}(\log n)$, cost *logarítimic*: cerca dicotòmica en un vector de mida n.
- $\mathcal{O}(n)$, cost *lineal*: recorregut en un vector de mida n.
- $\mathcal{O}(n \log n)$, cost *quasilineal*: *quicksort* o altres algorismes d'ordenació eficients sobre un vector de mida n.
- $\mathcal{O}(n^2)$, cost *quadràtic*: mètode de la bombolla o altres algorismes d'ordenació ineficients sobre un vector de mida n.
- $\mathcal{O}(n^3)$, cost *cúbic*: multiplicació elemental de matrius.

- $\mathcal{O}(n^k)$, per a una constant $k > 0$, cost *polinòmic*: test de primalitat (amb variants de l'algorisme AKS que van de $\mathcal{O}(n^{12})$ a $\mathcal{O}(n^6)$)
- $\mathcal{O}(c^{n^k})$, per a constants $c, k > 0$, cost *exponencial*: problemes de *backtracking*

Acabem amb uns exemples de càlcul del cost segons les regles donades abans.

Exemple 11. Si apliquem les regles anteriors a l'exemple 2, obtenim un cost $\mathcal{O}(1)$ per a l'`if`, que implica un cost $\mathcal{O}(n)$ per al `for`. La resta d'operacions sumades suposen un cost $\mathcal{O}(1)$, de manera que s'obté $\mathcal{O}(1)+\mathcal{O}(n) = \mathcal{O}(n+1)$ que, per la caracterització de la suma, és igual a $\mathcal{O}(n)$. Per tant, es tracta d'una funció de temps lineal.

Exemple 12. Si hem declarat les variables

```
int i;
vector<int> v;
```

aleshores la instrucció

```
return i;
```

té cost $\mathcal{O}(1)$, mentre que la instrucció

```
return v;
```

té cost $\mathcal{O}(\texttt{v.size()})$.

Exemple 13. Considerem l'algorisme d'inserció vist en la figura 1. Definint $n = \texttt{v.size()}$, podem observar que:

- El pas del paràmetre per referència de la línia 1 té cost $\mathcal{O}(1)$.
- El bucle de la línia 2 s'executa $n - 1$ cops.
- Les línies 3, 4, 6, 7-10 representen un cost $\mathcal{O}(1)$.
- El bucle de la línia 5 s'executa un màxim de i vegades; com que $i < n$, podem dir que s'executa $\mathcal{O}(n)$ vegades.

Expressem el cost de l'algorisme com:

$$\mathcal{O}(1) + (n-1) \cdot \big(\mathcal{O}(1) + \mathcal{O}(n) \cdot \mathcal{O}(1)\big)$$

que, donat que $n - 1 \in \mathcal{O}(n)$, la convenció 6 ens permet expressar com

$$\mathcal{O}(1) + \mathcal{O}(n) \cdot \big(\mathcal{O}(1) + \mathcal{O}(n) \cdot \mathcal{O}(1)\big)$$

Fent ús ara de les regles de la suma, del producte i de la caracterització de la suma, obtenim

$$\mathcal{O}(1) + \mathcal{O}(n) \cdot \mathcal{O}(1 + n \cdot 1) = \mathcal{O}(1) + \mathcal{O}(n) \cdot \mathcal{O}(n) = \mathcal{O}(n^2).$$

Per tant, podem afirmar que l'algorisme d'inserció té un cost quadràtic en el cas pitjor.

5 Apèndix

Demostració. (Proposició 5)

1. *Reflexivitat.* Immediata a partir de la definició.

2. *Transitivitat.* Suposem que

   $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c_1 < \infty \quad \text{i} \quad \lim_{n\to\infty} \frac{g(n)}{h(n)} = c_1 < \infty.$$

   Aleshores, $\lim_{n\to\infty} \frac{f(n)}{h(n)}$ es pot expressar com

   $$\lim_{n\to\infty} \frac{f(n)}{h(n)} \frac{g(n)}{g(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} \frac{g(n)}{h(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} \cdot \lim_{n\to\infty} \frac{g(n)}{h(n)} = c_1 \cdot c_2 < \infty$$

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

3. *Criteri de caracterització.* D'esquerra a dreta, es demostra aplicant la transitivitat. De dreta a esquerra, aplicant la reflexivitat.

4. *Invariància multiplicativa.* Aplicant el criteri de caracterització, només cal veure que $f \in \mathcal{O}(c \cdot f)$ i que $c \cdot f \in \mathcal{O}(f)$, però els dos límits de la definició de l'O gran són menors que infinit.

5. *Regla de la suma.* Suposem que $\lim_{n\to\infty} \frac{f_1(n)}{g_1(n)} = c_1 < \infty$ i $\lim_{n\to\infty} \frac{f_2(n)}{g_2(n)} = c_1 < \infty$. Aleshores,

$$\lim_{n\to\infty} \frac{f_1(n)+f_2(n)}{g_1(n)+g_2(n)} = \lim_{n\to\infty} \Big(\frac{f_1(n)}{g_1(n)+g_2(n)} + \frac{f_2(n)}{g_1(n)+g_2(n)}\Big) =$$

$$= \lim_{n\to\infty} \frac{f_1(n)}{g_1(n)+g_2(n)} + \lim_{n\to\infty} \frac{f_2(n)}{g_1(n)+g_2(n)} \leq c_1 + c_2 < \infty$$

6. *Caracterització de la suma.* Aplicant la propietat reflexiva i el criteri de caracterització, la demostració es redueix a comprovar la finitud de dos límits, que es pot fer a partir de la propietat següent:

$$\max(f,g)(n) \leq f(n)+g(n) \leq 2\max(f,g)(n).$$

7. *Regla del producte.* El límit que demostra la conclusió es pot expressar com el producte dels límits que corresponen a la hipòtesi, de manera que quedarà expressat com a producte de dues constants.

8. *Invariància additiva.* És conseqüència immediata de la caracterització de la suma. La condició $\lim_{n\to\infty} f(n) > 0$ és necessària perquè $\mathcal{O}(f) = \mathcal{O}(\max(c,f))$ per a qualsevol constant $c \in \mathbb{R}$.

□

Demostració. (Proposició 8)
Si $p(n)$ és un polinomi de grau k, es pot expressar com

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0,$$

on $a_k > 0$. Fent ús de l'observació 7, n'hi ha prou a demostrar que $0 < \lim_{n\to\infty} p(n)/n^k < \infty$, però

$$\lim_{n\to\infty} \frac{a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0}{n^k} = \lim_{n\to\infty} \Big(a_k + \frac{a_{k-1}}{n} + \cdots + \frac{a_0}{n^k}\Big) = a_k$$

i, efectivament, $0 < a_k < \infty$. □

Demostració. (Proposició 9)
Per demostrar cada inclusió estricta $f \subsetneq \mathcal{O}(g)$, n'hi ha prou a verificar que $\lim_{n\to\infty} f(n)/g(n) = 0$. D'aquesta manera, $f \in \mathcal{O}(g)$ però, com que $\lim_{n\to\infty} g(n)/f(n) = \infty$, tenim que $g \notin \mathcal{O}(f)$. Aplicant ara el criteri de caracterització, es dedueix que $\mathcal{O}(f) \subseteq \mathcal{O}(g)$ i $\mathcal{O}(g) \not\subseteq \mathcal{O}(f)$, que equival a $\mathcal{O}(f) \subsetneq \mathcal{O}(g)$. Verifiquem, doncs, per a cada parell de funcions consecutives f, g, que $\lim_{n\to\infty} f(n)/g(n) = 0$:

- $\mathcal{O}(1) \subsetneq \mathcal{O}(\log n)$.
  Trivialment, $\lim_{n\to\infty} 1/\log n = 0$.

- $\mathcal{O}(\log n) \subsetneq \mathcal{O}(n)$.
  $\lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{1/(n\ln 2)}{1} = \lim_{n\to\infty} \frac{1}{n\ln 2} = 0$, després d'aplicar la regla de l'Hôpital en la primera igualtat.

- $\mathcal{O}(n) \subsetneq \mathcal{O}(n\log n)$.
  $\lim_{n\to\infty} \frac{n}{n\log n} = \lim_{n\to\infty} \frac{1}{\log n} = 0$.

- $\mathcal{O}(n\log n) \subsetneq \mathcal{O}(n^2)$.
  $\lim_{n\to\infty} \frac{n\log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = 0$ com hem vist abans.

- $\mathcal{O}(n^k) \subsetneq \mathcal{O}(n^{k+1})$, per a tot $k \geq 0$.
  Clarament, $\lim_{n\to\infty} \frac{n^k}{n^{k+1}} = \lim_{n\to\infty} \frac{1}{n} = 0$.

- $\mathcal{O}(n^k) \subsetneq \mathcal{O}(c^n)$, per a $k \geq 0$, $c > 0$.
  $\lim_{n\to\infty} \frac{n^k}{c^n} = \lim_{n\to\infty} \frac{k\cdot n^{k-1}}{\ln c\cdot c^n} = \cdots = \lim_{n\to\infty} \frac{k!}{(\ln c)^k\cdot c^n} = 0$, on els punts suspensius indiquen que s'ha aplicat l'Hôpital k vegades.

□