

Tema 1. Anàlisi d'algorismes

Estructures de Dades i Algorismes

FIB

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

Q1 2021 – 22

- **Teoria:** Albert Oliveras
- **Problemes:** Gabriel Valiente (P31), Salvador Roura (P32)
- **Laboratori:** Gabriel Valiente (L31), Salvador Roura (L32), Antoni Lozano (L33)
- **Email:** oliveras@cs.upc.edu
- **Despatx:** 114, edifici Ω
- **Avaluació:**
 - NPP = nota de l'examen parcial de paper (entre 0 i 10)
 - NFO = nota de l'examen final d'ordinador (entre 0 i 10)
 - NFE = nota de l'examen final escrit (entre 0 i 10)
 - NJ = nota del joc (entre 0 i 10)
 - NOTA = $\min(10, \max(30\% \text{ NPP} + 30\% \text{ NFO} + 30\% \text{ NFE} + 20\% \text{ NJ}, 30\% \text{ NFO} + 60\% \text{ NFE} + 20\% \text{ NJ}))$

- ➊ **Anàlisi d'Algorismes.** Cost en temps i espai. Cas pitjor, millor i mitjà. Notació asimptòtica. Càlcul del cost d'algorismes iteratius i recursius.
- ➋ **Algorismes de Dividir i Vèncer.** Mergesort. Quicksort. Algorisme de Karatsuba per multiplicar nombres llargs. Algorisme de Strassen per multiplicar matrius. Altres exemples.
- ➌ **Diccionaris.** Operacions. Taules. Taules ordenades. Llistes. Llistes ordenades. Taules de dispersió. Arbres binaris de cerca. Arbres AVL.
- ➍ **Cues amb Prioritats.** Operacions. Heaps. Heapsort.
- ➎ **Grafs.** Operacions. Representació en matrius d'adjacència. Representació en llistes d'adjacència. Representació implícita. Recorregut en fondària (DFS). Recorregut en amplada (BFS). Ordenació topològica. Algorisme de Dijkstra per camins mínims un-a-tots.
- ➏ **Algorismes de Generació i Cerca Exhaustiva.** Principis. Exemples.
- ➐ **Nocions d'Intractabilitat i d'Indecidibilitat.** Introducció bàsica a les classes P i NP, a les reduccions i a la NP-Completesa.

Tema 1. Anàlisi d'algorismes

1 Temps de càlcul i espai de memòria

- Eficiència dels algorismes
- Mida de l'entrada i cost
- Ordre de magnitud

2 Notació asimptòtica

- Notació asimptòtica: definicions
- Notació asimptòtica: propietats
- Formes de creixement

3 Cost dels algorismes

- Algorismes no recursius
- Algorismes recursius
- Teoremes mestres

Tema 1. Anàlisi d'algorismes

1 Temps de càlcul i espai de memòria

- Eficiència dels algorismes
- Mida de l'entrada i cost
- Ordre de magnitud

2 Notació asimptòtica

- Notació asimptòtica: definicions
- Notació asimptòtica: propietats
- Formes de creixement

3 Cost dels algorismes

- Algorismes no recursius
- Algorismes recursius
- Teoremes mestres

Objectius:

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

Objectius:

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

Objectius:

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

Consideracions sobre l'eficiència:

- Depèn de la mida de les entrades
- És un concepte relatiu: hi intervenen compilador, màquina, ...
- Aquests factors afecten de forma lineal

Consideracions sobre l'eficiència:

- Depèn de la mida de les entrades
- És un concepte relatiu: hi intervenen compilador, màquina, ...
- Aquests factors afecten de forma lineal

Consideracions sobre l'eficiència:

- Depèn de la mida de les entrades
- És un concepte relatiu: hi intervenen compilador, màquina, ...
- 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 en un vector i ordenar-los de forma decreixent.
Cada element següent es tracta per separat:

- si és més petit que el k -èsim del vector, es descarta;
- si no, se situa correctament en el vector i s'elimina el més petit.

Retornar l'element de la k -èsima posició.

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 en un vector i ordenar-los de forma decreixent.
Cada element següent es tracta per separat:

- si és més petit que el k -èsim del vector, es descarta;
- si no, se situa correctament en el vector i s'elimina el més petit.

Retornar l'element de la k -èsima posició.

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 en un vector i ordenar-los de forma decreixent. Cada element següent es tracta per separat:

- si és més petit que el k -èsim del vector, es descarta;
- si no, se situa correctament en el vector i s'elimina el més petit.

Retornar l'element de la k -èsima posició.

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 ja 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
- (recorrem sempre un metre més en direcció contrària)



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
- (recorrem sempre el doble de distància en direcció contrària)



Donat un algorisme A amb conjunt d'entrades E ,
el **cost** d' A (en temps, o en espai) es pot expressar com una funció

$$T : E \rightarrow \mathbb{R}^+.$$

Però calcular T **per cada entrada** pot ser complicat i poc manejable.

És més útil agrupar les entrades amb la mateixa **mida**,
i estudiar el cost sobre aquestes entrades en conjunt.

Donat un algorisme A amb conjunt d'entrades E ,
el **cost** d' A (en temps, o en espai) es pot expressar com una funció

$$T : E \rightarrow \mathbb{R}^+.$$

Però calcular T **per cada entrada** pot ser complicat i poc manejable.

És més útil agrupar les entrades amb la mateixa **mida**,
i estudiar el cost sobre aquestes entrades en conjunt.

Donat un algorisme A amb conjunt d'entrades E ,
el **cost** d' A (en temps, o en espai) es pot expressar com una funció

$$T : E \rightarrow \mathbb{R}^+.$$

Però calcular T **per cada entrada** pot ser complicat i poc manejable.

És més útil agrupar les entrades amb la mateixa **mida**,
i estudiar el cost sobre aquestes entrades en conjunt.

Mida

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

Tipus d'entrades

- Nombres naturals \rightarrow codificació en valor (en “unari”)

$$|15| = 15$$

- Nombres naturals \rightarrow codificació en binari

$$|15| = 4 \text{ perquè } 15_{10} = 1111_2$$

- Llistes, vectors \rightarrow nombre de components

$$|(23, 1, 7, 0, 12)| = 5$$

Mida de l'entrada i cost

Mida

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

Tipus d'entrades

- Nombres naturals \rightarrow codificació en valor (en “unari”)

$$|15| = 15$$

- Nombres naturals \rightarrow codificació en binari

$$|15| = 4 \text{ perquè } 15_{10} = 1111_2$$

- Llistes, vectors \rightarrow nombre de components

$$|(23, 1, 7, 0, 12)| = 5$$

- **Cas pitjor.** $T_{pitjor}(n) = \max\{T(x) \mid x \in E \wedge |x| = n\}$

Ofereix garanties sobre uns límits que l'algorisme no superarà.

- **Cas millor.** $T_{millor}(n) = \min\{T(x) \mid x \in E \wedge |x| = n\}$

Poc útil.

- **Cas mig.** $T_{mig}(n) = \sum_{x \in E, |x|=n} Pr(x) T(x),$

on $Pr(x)$ és la probabilitat d'escollir l'entrada x entre totes les de mida n

Cal definir com es distribueix la probabilitat, i sol ser difícil de calcular.

- **Cas pitjor.** $T_{pitjor}(n) = \max\{T(x) \mid x \in E \wedge |x| = n\}$

Ofereix garanties sobre uns límits que l'algorisme no superarà.

- **Cas millor.** $T_{millor}(n) = \min\{T(x) \mid x \in E \wedge |x| = n\}$

Poc útil.

- **Cas mig.** $T_{mig}(n) = \sum_{x \in E, |x|=n} Pr(x) T(x),$

on $Pr(x)$ és la probabilitat d'escollir l'entrada x entre totes les de mida n

Cal definir com es distribueix la probabilitat, i sol ser difícil de calcular.

- **Cas pitjor.** $T_{pitjor}(n) = \max\{T(x) \mid x \in E \wedge |x| = n\}$

Ofereix garanties sobre uns límits que l'algorisme no superarà.

- **Cas millor.** $T_{millor}(n) = \min\{T(x) \mid x \in E \wedge |x| = n\}$

Poc útil.

- **Cas mig.** $T_{mig}(n) = \sum_{x \in E, |x|=n} Pr(x) T(x),$

on $Pr(x)$ és la probabilitat d'escollir l'entrada x entre totes les de mida n

Cal definir com es distribueix la probabilitat, i sol ser difícil de calcular.

Ordre de magnitud

Necessitem una notació que:

- permeti expressar una fita superior de

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

(sabrem que l'algorisme no superarà la fita)

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

Notació O gran

Donada una funció f , $\mathcal{O}(f)$ representa la classe de funcions que “creixen com f o més a poc a poc”.

Formalment, $g \in \mathcal{O}(f)$ si existeixen $c > 0$ i $n_0 \in \mathbb{N}$ tals que

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

En lloc de $f \in \mathcal{O}(g)$, s'escriu molt sovint “ f és $\mathcal{O}(g)$ ” o, també, $f = \mathcal{O}(g)$.

Ordre de magnitud

Necessitem una notació que:

- permeti expressar una fita superior de

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

(sabrem que l'algorisme no superarà la fita)

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

Notació O gran

Donada una funció f , $\mathcal{O}(f)$ representa la classe de funcions que “creixen com f o més a poc a poc”.

Formalment, $g \in \mathcal{O}(f)$ si existeixen $c > 0$ i $n_0 \in \mathbb{N}$ tals que

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

En lloc de $f \in \mathcal{O}(g)$, s'escriu molt sovint “ f és $\mathcal{O}(g)$ ” o, també, $f = \mathcal{O}(g)$.

Ordre de magnitud

Necessitem una notació que:

- permeti expressar una fita superior de

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

(sabrem que l'algorisme no superarà la fita)

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

Notació O gran

Donada una funció f , $\mathcal{O}(f)$ representa la classe de funcions que “creixen com f o més a poc a poc”.

Formalment, $g \in \mathcal{O}(f)$ si existeixen $c > 0$ i $n_0 \in \mathbb{N}$ tals que

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

En lloc de $f \in \mathcal{O}(g)$, s'escriu molt sovint “ f és $\mathcal{O}(g)$ ” o, també, $f = \mathcal{O}(g)$.

Ordre de magnitud

Necessitem una notació que:

- permeti expressar una fita superior de

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

(sabrem que l'algorisme no superarà la fita)

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

Notació O gran

Donada una funció f , $\mathcal{O}(f)$ representa la classe de funcions que “creixen com f o més a poc a poc”.

Formalment, $g \in \mathcal{O}(f)$ si existeixen $c > 0$ i $n_0 \in \mathbb{N}$ tals que

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

En lloc de $f \in \mathcal{O}(g)$, s'escriu molt sovint “ f és $\mathcal{O}(g)$ ” o, també, $f = \mathcal{O}(g)$.

Exemple

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

Per justificar-ho, només cal trobar c i n_0 que compleixin:

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

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 $f(n) = \mathcal{O}(n^3)$ amb $c = 9$ i $n_0 = 4$.

(De fet, és fàcil trobar constants c i n_0 més petites.)

Tema 1. Anàlisi d'algorismes

1 Temps de càlcul i espai de memòria

- Eficiència dels algorismes
- Mida de l'entrada i cost
- Ordre de magnitud

2 Notació asimptòtica

- Notació asimptòtica: definicions
- Notació asimptòtica: propietats
- Formes de creixement

3 Cost dels algorismes

- Algorismes no recursius
- Algorismes recursius
- Teoremes mestres

Notació asimptòtica: definicions

- La **notació asimptòtica** permet classificar les funcions segons com creixen “a la llarga”.
- Se centra en el comportament de les funcions per a entrades grans. Per exemple, $10^6 n > n^2$ fins a un cert valor de n que podem trobar així:

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

Per a $n \geq 10^6$, doncs, n^2 és més gran que $10^6 n$. En aquest cas, direm que la funció $g(n) = 10^6 n$ està fitada per $f(n) = n^2$ **asimptòticament**.

- La notació $\mathcal{O}(g)$, anomenada “O gran”, representa el conjunt de funcions fitades asimptòticament per g .

Notació asimptòtica: definicions

Notació Θ ((a): fita exacta asimptòtica)

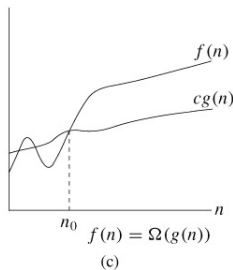
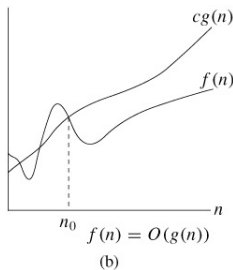
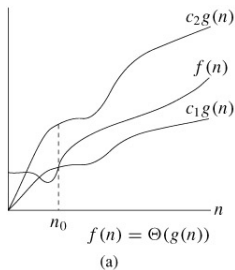
$$\Theta(g) = \{f : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists c_1, c_2 \in \mathbb{R}^+ \exists n_0 \in \mathbb{N} \forall n \geq n_0 \ c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

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

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

Notació Ω ((c): fita inferior asimptòtica)

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



Notació Θ (fita exacta asimptòtica)

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

Exemples

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

Notació O gran (fita superior asimptòtica)

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

Exemples

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

Notació Ω ((c): fita inferior asimptòtica)

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

Exemples

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

Relacions entre \mathcal{O} , Ω i Θ

Donades dues funcions f i g :

- $f \in \Omega(g) \iff g \in \mathcal{O}(f)$
- $\Theta(f) = \mathcal{O}(f) \cap \Omega(f)$
- $f \in \Theta(g) \iff f \in \mathcal{O}(g) \text{ i } f \in \Omega(g)$

Regles del límit

- $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f \in \mathcal{O}(g)$ però $f \notin \Omega(g)$
- $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f \in \Omega(g)$ però $f \notin \mathcal{O}(g)$
- $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$, on $0 < c < \infty \Rightarrow f \in \Theta(g)$ i $g \in \Theta(f)$

Propietats de l'O gran

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

- **Reflexivitat.** $f \in \mathcal{O}(f)$
- *Transitivitat.* $h \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f) \Rightarrow h \in \mathcal{O}(f)$
- *Caracterització.* $g \in \mathcal{O}(f) \iff \mathcal{O}(g) \subseteq \mathcal{O}(f)$
- *Suma.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 + g_2 \in \mathcal{O}(f_1 + f_2) = \mathcal{O}(\max(f_1, f_2))$
- *Producte.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 \cdot g_2 \in \mathcal{O}(f_1 \cdot f_2)$
- *Invariància multiplicativa.* Per a tota constant $c \in \mathbb{R}^+$, $\mathcal{O}(f) = \mathcal{O}(c \cdot f)$

Propietats de l'O gran

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

- *Reflexivitat.* $f \in \mathcal{O}(f)$
- *Transitivitat.* $h \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f) \Rightarrow h \in \mathcal{O}(f)$
- *Caracterització.* $g \in \mathcal{O}(f) \iff \mathcal{O}(g) \subseteq \mathcal{O}(f)$
- *Suma.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 + g_2 \in \mathcal{O}(f_1 + f_2) = \mathcal{O}(\max(f_1, f_2))$
- *Producte.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 \cdot g_2 \in \mathcal{O}(f_1 \cdot f_2)$
- *Invariància multiplicativa.* Per a tota constant $c \in \mathbb{R}^+$, $\mathcal{O}(f) = \mathcal{O}(c \cdot f)$

Propietats de l'O gran

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

- *Reflexivitat.* $f \in \mathcal{O}(f)$
- *Transitivitat.* $h \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f) \Rightarrow h \in \mathcal{O}(f)$
- *Caracterització.* $g \in \mathcal{O}(f) \iff \mathcal{O}(g) \subseteq \mathcal{O}(f)$
- *Suma.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 + g_2 \in \mathcal{O}(f_1 + f_2) = \mathcal{O}(\max(f_1, f_2))$
- *Producte.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 \cdot g_2 \in \mathcal{O}(f_1 \cdot f_2)$
- *Invariància multiplicativa.* Per a tota constant $c \in \mathbb{R}^+$, $\mathcal{O}(f) = \mathcal{O}(c \cdot f)$

Propietats de l'O gran

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

- *Reflexivitat.* $f \in \mathcal{O}(f)$
- *Transitivitat.* $h \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f) \Rightarrow h \in \mathcal{O}(f)$
- *Caracterització.* $g \in \mathcal{O}(f) \iff \mathcal{O}(g) \subseteq \mathcal{O}(f)$
- *Suma.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 + g_2 \in \mathcal{O}(f_1 + f_2) = \mathcal{O}(\max(f_1, f_2))$
- *Producte.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 \cdot g_2 \in \mathcal{O}(f_1 \cdot f_2)$
- *Invariància multiplicativa.* Per a tota constant $c \in \mathbb{R}^+$, $\mathcal{O}(f) = \mathcal{O}(c \cdot f)$

Propietats de l'O gran

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

- *Reflexivitat.* $f \in \mathcal{O}(f)$
- *Transitivitat.* $h \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f) \Rightarrow h \in \mathcal{O}(f)$
- *Caracterització.* $g \in \mathcal{O}(f) \iff \mathcal{O}(g) \subseteq \mathcal{O}(f)$
- *Suma.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 + g_2 \in \mathcal{O}(f_1 + f_2) = \mathcal{O}(\max(f_1, f_2))$
- *Producte.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 \cdot g_2 \in \mathcal{O}(f_1 \cdot f_2)$
- *Invariància multiplicativa.* Per a tota constant $c \in \mathbb{R}^+$, $\mathcal{O}(f) = \mathcal{O}(c \cdot f)$

Propietats de l'O gran

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

- *Reflexivitat.* $f \in \mathcal{O}(f)$
- *Transitivitat.* $h \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f) \Rightarrow h \in \mathcal{O}(f)$
- *Caracterització.* $g \in \mathcal{O}(f) \iff \mathcal{O}(g) \subseteq \mathcal{O}(f)$
- *Suma.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 + g_2 \in \mathcal{O}(f_1 + f_2) = \mathcal{O}(\max(f_1, f_2))$
- *Producte.* $g_1 \in \mathcal{O}(f_1) \wedge g_2 \in \mathcal{O}(f_2) \Rightarrow g_1 \cdot g_2 \in \mathcal{O}(f_1 \cdot f_2)$
- *Invariància multiplicativa.* Per a tota constant $c \in \mathbb{R}^+$, $\mathcal{O}(f) = \mathcal{O}(c \cdot f)$

Propietats de Θ

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

- *Reflexivitat.* $f \in \Theta(f)$
- *Transitivitat.* $h \in \Theta(g) \wedge g \in \Theta(f) \Rightarrow h \in \Theta(f)$
- *Simetria.* $g \in \Theta(f) \iff f \in \Theta(g) \iff \Theta(g) = \Theta(f)$
- *Suma.* $g_1 \in \Theta(f_1) \wedge g_2 \in \Theta(f_2) \Rightarrow g_1 + g_2 \in \Theta(f_1 + f_2) = \Theta(\max(f_1, f_2))$
- *Producte.* $g_1 \in \Theta(f_1) \wedge g_2 \in \Theta(f_2) \Rightarrow g_1 \cdot g_2 \in \Theta(f_1 \cdot f_2)$
- *Invariància 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 $\mathcal{O}(f)$ 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 :

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

Notació asimptòtica: propietats

Notació de classes

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

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

Notació asimptòtica: propietats

Polinomis

Sigui $p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_0$ amb $a_k > 0$. Aleshores $p(n) \in \Theta(n^k)$

Logarismes

Siguin $a, b > 1$. Aleshores $\log_a(n) \in \Theta(\log_b(n))$

Logarismes, polinomis i exponencials

Siguin $a > 0, b > 0, c > 1$:

- $(\ln n)^a \in \mathcal{O}(n^b)$ però $(\ln n)^a \notin \Omega(n^b)$ [de fet, $\lim_{n \rightarrow \infty} \frac{(\ln n)^a}{n^b} = 0$]
- $n^b \in \mathcal{O}(c^n)$ però $n^b \notin \Omega(c^n)$ [de fet, $\lim_{n \rightarrow \infty} \frac{n^b}{c^n} = 0$]

Exercici: Compara asimptòticament

n^{412}	vs	1.02^n	$243n^2$	vs	n^3	2^n	vs	3^n
$(\log n)^{923}$	vs	$n^{0.0001}$	$n(\ln n)^4$	vs	$n^2 \ln n$	$(\log_2 7)^n$	vs	$(\log_4 7)^n$
$(\ln n)^{7842}$	vs	1.001^n	$n 2^n$	vs	$n^3 1.5^n$	$2^{\sqrt{\log n}}$	vs	n

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions:** $\Theta(n!)$, $\Theta(n^n)$.

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions:** $\Theta(n!)$, $\Theta(n^n)$.

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions**: $\Theta(n!)$, $\Theta(n^n)$.

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions:** $\Theta(n!)$, $\Theta(n^n)$.

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions:** $\Theta(n!)$, $\Theta(n^n)$.

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions:** $\Theta(n!)$, $\Theta(n^n)$.

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions:** $\Theta(n!)$, $\Theta(n^n)$.

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions**: $\Theta(n!)$, $\Theta(n^n)$.

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions:** $\Theta(n!)$, $\Theta(n^n)$.

Costos freqüents

- **Constant** $\Theta(1)$. Decidir si un nombre és parell o senar.
- **Logarítmic** $\Theta(\log n)$. Cerca binària.
- **Radical** $\Theta(\sqrt{n})$. Test bàsic de primalitat.
- **Lineal** $\Theta(n)$. Cerca seqüencial en un vector.
- **Quasilineal** $\Theta(n \log n)$. Ordenació eficient d'un vector.
- **Quadràtic** $\Theta(n^2)$. Suma de dues matrius quadrades de mida $n \times n$.
- **Cúbic** $\Theta(n^3)$. Producte de dues matrius quadrades de mida $n \times n$.
- **Polinòmic** $\Theta(n^k)$, per a $k \geq 1$ constant. Enumerar combinacions (n elements presos de k en k).
- **Exponencial** $\Theta(k^n)$, per a k constant. Cerca en un espai de configuracions (d'amplada k i alçada n).
- **Altres funcions**: $\Theta(n!)$, $\Theta(n^n)$.

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

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

Efecte de les millores tecnològiques en algorismes polinòmics i exponencials

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	$N_5 + 4.19$	$N_5 + 6.29$

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 en un vector i ordenar-los de forma decreixent.
Cada element següent es tracta per separat:

- si és més petit que el k -èsim del vector, es descarta;
- si no, se situa correctament en el vector i s'elimina el més petit.

Retornar l'element de la k -èsima posició.

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 en un vector i ordenar-los de forma decreixent.
Cada element següent es tracta per separat:

- si és més petit que el k -èsim del vector, es descarta;
- si no, se situa correctament en el vector i s'elimina el més petit.

Retornar l'element de la k -èsima posició.

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 en un vector i ordenar-los de forma decreixent. Cada element següent es tracta per separat:

- si és més petit que el k -èsim del vector, es descarta;
- si no, se situa correctament en el vector i s'elimina el més petit.

Retornar l'element de la k -èsima posició.

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): $\mathcal{O}(n^2)$
- amb un algorisme d'ordenació eficient: $\mathcal{O}(n \log n)$

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 en un vector i ordenar-los. Cada element següent es tracta per separat:

- si és més petit que el k -èsim del vector, es descarta;
- si no, se situa correctament en el vector i s'elimina el més petit.

Retornar l'element de la k -èsima posició.

$$\mathcal{O}((k \log k) + (n - k) \cdot k)$$

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

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 ja 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
- (recorrem sempre un metre més en direcció contrària)

Temps quan la porta està a distància n :

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

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

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
- (recorrem sempre el doble de distància en direcció contrària)

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

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

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

Tema 1. Anàlisi d'algorismes

1 Temps de càlcul i espai de memòria

- Eficiència dels algorismes
- Mida de l'entrada i cost
- Ordre de magnitud

2 Notació asimptòtica

- Notació asimptòtica: definicions
- Notació asimptòtica: propietats
- Formes de creixement

3 Cost dels algorismes

- Algorismes no recursius
- Algorismes recursius
- Teoremes mestres

Càlcul del cost (en temps):

- El cost d'una **operació elemental** és $\Theta(1)$. Això inclou:
 - una assignació de tipus bàsics (`int`, `bool`, `double`, ...)
 - un increment o decrement d'una variable de tipus bàsic
 - una operació aritmètica
 - una lectura o escriptura de tipus bàsic
 - una comparació
 - l'accés a una component d'un vector
- 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 `return E` és la suma del cost de l'avaluació de l'expressió E més el cost de copiar (assignar) el resultat.
- El cost del pas de paràmetres per referència és $\Theta(1)$
- El cost de construir o copiar un vector de mida n (assignació, pas per valor, return) és $\Theta(n)$

- 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 cada fragment F_k té cost C_k , el cost de la composició seqüencial

$$F_1; F_2; \dots; F_N$$

és $C_1 + C_2 + \dots + C_N$

- 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**

if (B) F_1 ; else F_2

és $D + C_1$ si es compleix B , i $D + C_2$ altrament.

En tot cas, el cost és $\leq D + \max(C_1, C_2)$.

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

- 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**

`if (B) F_1 ; else F_2`

és $D + C_1$ si es compleix B , i $D + C_2$ altrament.

En tot cas, el cost és $\leq D + \max(C_1, C_2)$.

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

Exemple d'ordenació per selecció

Passos per ordenar 5, 6, 1, 2, 0, 7, 4, 3 segons l'algorisme de selecció.

En vermell, els elements ja ordenats.

En groc, 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

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: $m - 1 + 1 = m$, $n - 1 + 1 = n$.

7 Cost $\Theta(i)$.

altres Instruccions de cost constant: $\Theta(1)$.

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

Exemple d'ordenació per inserció

Passos per ordenar 5, 6, 1, 2, 0, 7, 4, 3 segons l'algorisme d'inserció.

En vermell, els elements ja ordenats.

Entre parèntesis, 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)

Ordenació per inserció

```
0 void ordena_insercio (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: $(n - 1) - 1 + 1 = n - 1 = \Theta(n)$.

1,2 Condició d'iteració i línia 2: $\Theta(1)$.

3 Iteracions bucle: entre 0 i $k - 1 - 0 + 1 = k$.

4,5 Assignacions de cost $\Theta(1)$.

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

Tenim 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

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

Aleshores,

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

i

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

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

Definició

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

Exemple

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

Per exemple, $C(1) = 1$, $C(2) = 1 + 2 = 3$ i $C(3) = 3 + 3 = 6$.

Però voldríem una fórmula no recurrent per calcular el valor!

Resoldre la recurrència vol dir donar una forma tancada pel terme n -èsim.

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

Definició

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

Exemple

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

Per exemple, $C(1) = 1$, $C(2) = 1 + 2 = 3$ i $C(3) = 3 + 3 = 6$.

Però voldríem una fórmula no recurrent per calcular el valor!

Resoldre la recurrència vol dir donar una forma tancada pel terme n -èsim.

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

Solució

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

Per trobar una recurrència que descrigui el cost d'un algorisme recursiu, hem de determinar:

- el paràmetre de recursió n (típicament la mida de l'entrada),
- el cost del cas inductiu
 - nombre de crides recursives
 - valors del paràmetre recursiu de les crides
 - cost dels càlculs extra no recursius

Cerca lineal recursiva

Mirar si un nombre x apareix en un vector a entre les posicions 0 i $n - 1$ comparant-lo amb $a[0], a[1], \dots, a[n - 1]$.

Si el troba, retorna l'índex de la posició que conté x .

Altrament, retorna -1 .

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

El paràmetre de la recursió és n , la mida del vector.

Definim la recurrència per a $T(n)$, el cost (en el cas pitjor) de l'algorisme:

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

$$T(n) = T(n - 1) + \Theta(1) \text{ per a } n \geq 1 \quad (\text{i } T(0) = \Theta(1))$$

Solució

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

Cerca binària recursiva

Mirar si un nombre x apareix en un vector ordenat a entre les posicions i i j per cerca binària.

Si el troba, retorna l'índex de la posició que conté x .

Altrament, retorna -1 .

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

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

El paràmetre de recursió és $n = j - i + 1$, la mida de l'interval a explorar. Definim la recurrència per a $T(n)$, el cost (en cas pitjor) de l'algorisme:

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

$$T(n) = T(n/2) + \Theta(1) \quad \text{per a } n \geq 1 \quad (\text{i } T(0) = \Theta(1)).$$

Solució

$$\begin{aligned} T(n) &= T(n/2) + \Theta(1) \\ &= T(n/4) + 2 \cdot \Theta(1) \\ &= T(n/8) + 3 \cdot \Theta(1) \\ &\vdots \\ &= T(n/2^{\log_2 n}) + \log_2 n \cdot \Theta(1) \\ &= T(1) + \log_2 n \cdot \Theta(1) \\ &= T(0) + (\log_2 n + 1) \cdot \Theta(1) \\ &= (\log_2 n + 2) \cdot \Theta(1) = \Theta(\log n + 2) = \Theta(\log n). \end{aligned}$$

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

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

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.

Teorema mestre de recurrències subtractives

Sigui $T(n)$ la recurrència

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

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,

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

Teorema mestre de recurrències subtractives

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

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:

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

Teorema mestre de recurrències subtractives

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

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

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

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

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

Teorema mestre de recurrències subtractives

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

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

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

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

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

Teorema mestre de recurrències divisores

Sigui $T(n)$ la recurrència

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

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,

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

Teorema mestre de recurrències divisores

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

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 1

Hem vist que el cost de l'algorisme recursiu de **cerca binària** es pot descriure amb la recurrència $T(n) = T(n/2) + \Theta(1)$ per a $n \geq 1$ 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:

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

Exemple 2

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

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

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

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

Teorema mestre de recurrències divisores

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

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,

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

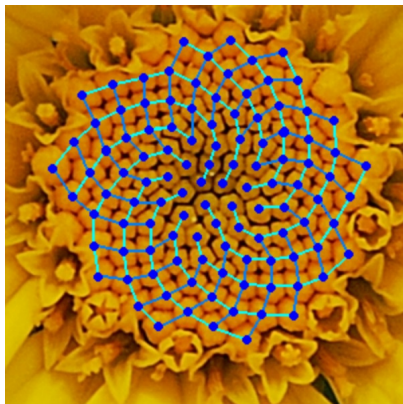
Exemple 2

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:

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

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



1a solució

```
int fib(int k) {  
    if (k <= 1) return 1;  
    else return fib(k-1) + fib(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!

Podem aproximar i llavors aplicar el teorema mestre:
(noteu les inequacions i les O/Ω)

- $T(k) = T(k-1) + T(k-2) + \Theta(1) \leq 2T(k-1) + \Theta(1)$ dóna
 $T(k) = O(2^k)$, i
- $T(k) = T(k-1) + T(k-2) + \Theta(1) \geq 2T(k-2) + \Theta(1)$ dóna
 $T(k) = \Omega(\sqrt{2}^k)$

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

Noteu que $\sqrt{2} = 1.414213562\dots$ i $\phi = 1.618033988\dots$

2a solució

```
int fib(int k) {  
    if (k <= 1) return 1;  
    int cur = 1;  
    int pre = 1;  
    for (int i = 1; i < k; ++i) {  
        int tmp = pre;  
        pre = cur;  
        cur = cur + tmp;  
    }  
    return cur;  
}
```

- Cada volta costa temps $\Theta(1)$
- El cost és proporcional al nombre de voltes: $\Theta(k)$

Algorisme d'exponenciació ràpida

```
template <typename T>
T pow(const T& a, int k) {
    if (k == 0) return identity(a);
    T b = pow(a, k/2);
    if (k % 2 == 0) return b*b;
    else          return a*b*b;
}
```

- Serveix per a calcular a^k
- $a^0 = 1$
- Si k és parell, $a^k = (a^{\frac{k}{2}})^2 = (a^{k \div 2})^2$
- Si k és senar, $a^k = a \cdot a^{k-1} = a \cdot (a^{\frac{k-1}{2}})^2 = a \cdot (a^{k \div 2})^2$

Algorisme d'exponenciació ràpida

```
template <typename T>
T pow(const T& a, int k) {
    if (k == 0) return identity(a);
    T b = pow(a, k/2);
    if (k % 2 == 0) return b*b;
    else          return a*b*b;
}
```

- Assumim cost d'`identity`, `*`, construcció i còpia pel tipus T és $\Theta(1)$
- El cost $T(k)$ segueix la recurrència $T(k) = T(k/2) + \Theta(1)$
- Aplicant el teorema mestre tenim que $T(k) = \Theta(\log(k))$

- Per inducció es pot demostrar que

$$\begin{pmatrix} f(k+1) \\ f(k) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^k \begin{pmatrix} f(1) \\ f(0) \end{pmatrix} \text{ per tot } k \geq 0$$

3a solució

```
typedef vector<vector<int>> matrix;  
  
int fib(int k) {  
    matrix f = {{1, 1}, {1, 0}};  
    matrix p = pow(f, k);  
    return p[1][0] + p[1][1];  
}
```

- Ja hem vist que el cost és $\Theta(\log(k))$

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

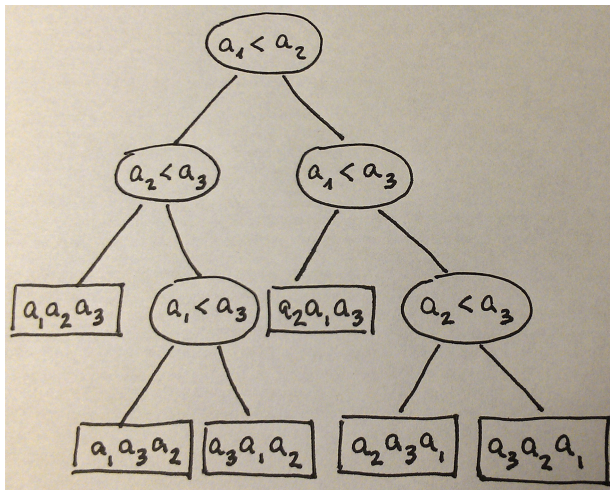
Proposició

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

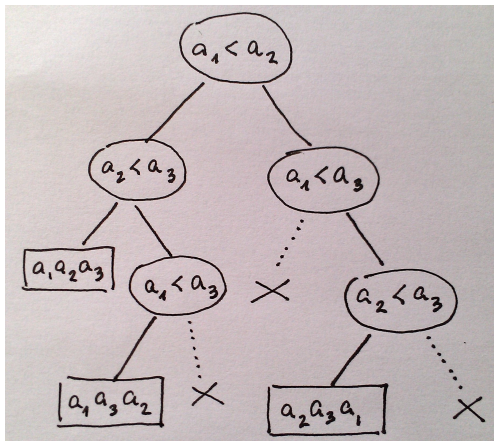
Suposem que volem ordenar a_1 , a_2 i a_3 . Si $a_1 < a_2$, seguim per la branca esquerra; si no, per la dreta. Els rectangles representen les ordenacions trobades. **L'alçada de l'arbre és el cost en cas pitjor.**



Fita inferior dels algorismes d'ordenació

Considerem un arbre que ordena n elements:

- cada fulla correspon a una permutació de $\{1, 2, \dots, n\}$
- cada permutació de $\{1, 2, \dots, n\}$ ha d'aparèixer en alguna fulla
(si una no hi aparegués, 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 d'alçada d té $\leq 2^d$ fulles
- per tant, l'alçada del nostre arbre és almenys de $\log_2 n!$

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

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

tenim que

$$\log_2 n! \geq \log_2 (n/2)^{(n/2)} = \frac{n}{2} \log_2 (n/2) \in \Omega(n \log n).$$

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 d'alçada d té $\leq 2^d$ fulles
- per tant, l'alçada del nostre arbre és almenys de $\log_2 n!$

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

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

tenim que

$$\log_2 n! \geq \log_2 (n/2)^{(n/2)} = \frac{n}{2} \log_2 (n/2) \in \Omega(n \log n).$$

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 d'alçada d té $\leq 2^d$ fulles
- per tant, l'alçada del nostre arbre és almenys de $\log_2 n!$

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

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

tenim que

$$\log_2 n! \geq \log_2 (n/2)^{(n/2)} = \frac{n}{2} \log_2 (n/2) \in \Omega(n \log n).$$

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 d'alçada d té $\leq 2^d$ fulles
- per tant, l'alçada del nostre arbre és almenys de $\log_2 n!$

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

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

tenim que

$$\log_2 n! \geq \log_2 (n/2)^{(n/2)} = \frac{n}{2} \log_2 (n/2) \in \Omega(n \log n).$$

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 d'alçada d té $\leq 2^d$ fulles
- per tant, l'alçada del nostre arbre és almenys de $\log_2 n!$

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

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

tenim que

$$\log_2 n! \geq \log_2 (n/2)^{(n/2)} = \frac{n}{2} \log_2 (n/2) \in \Omega(n \log n).$$

Proposició

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