TEORIA

PROGRAMACIÓ CIENTÍFICA

T9

Anàlisi del cost algorísmic

ANÀLISI DEL

COST ALGORÍSMIC

Anàlisi del cost algorísmic

Avui

- Aprendre a mesurar els ordres de creixement dels algorismes
- La "Big O" notation
- Fórmules per calcular la complexitat d’un algorisme a partir de les seves instruccions

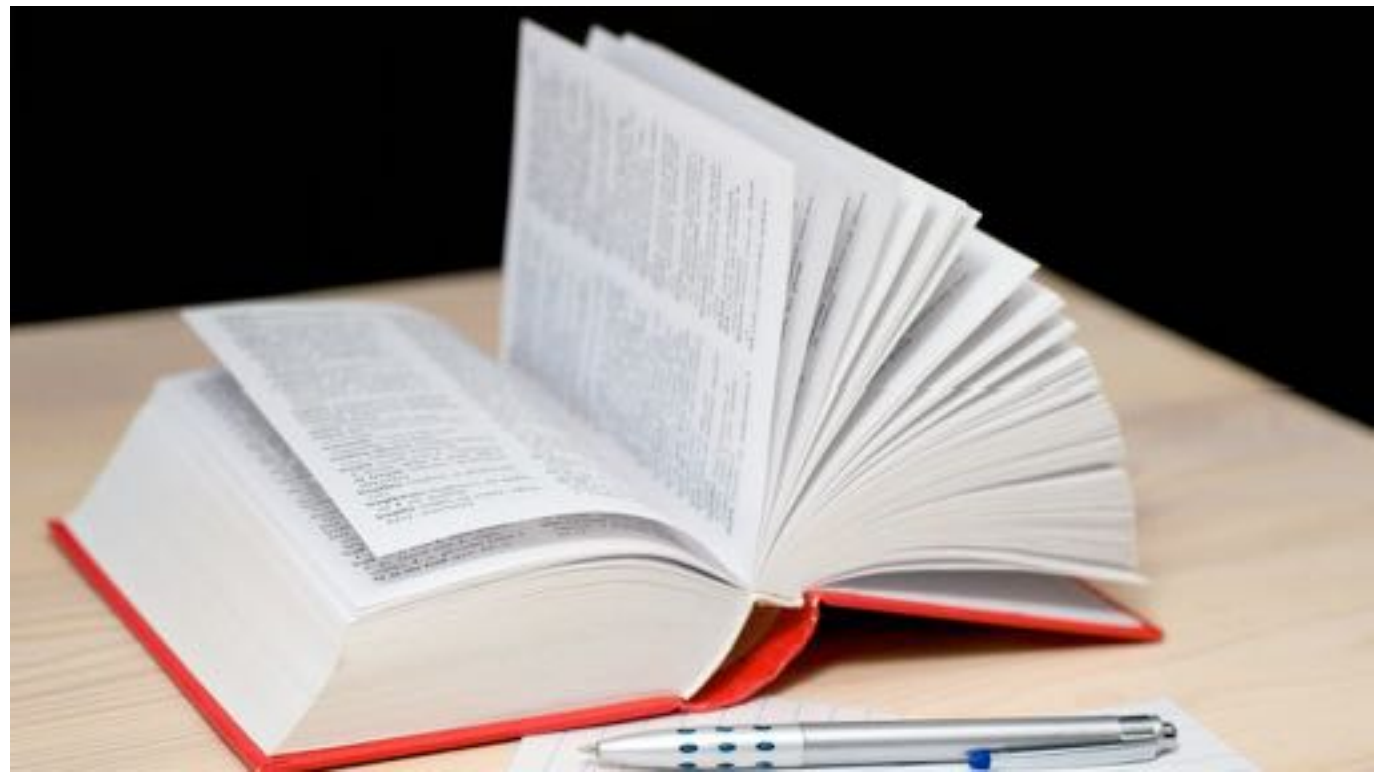
Anàlisi del cost algorísmic

Avui

- Aprendre a mesurar els ordres de creixement dels algorismes
- La "Big O" notation
- Fórmules per calcular la complexitat d'un algorisme a partir de les seves instruccions

Conceptes

- **Eficàcia**: Que té la virtut de produir l'efecte volgut.
- **Eficiència**: relació entre el treball efectuat per una màquina i els recursos que consumeix per produir aquest treball.



RECORDEU EL CAS DE BUSCAR PARAULES AL DICCIONARI?

Anàlisi del cost algorísmic

Què volem fer?

- Com podem raonar sobre un algorisme per tal de predir la quantitat de **temps** que necessitarà per resoldre un problema d'una **mida** determinada?
- Ens interessa poder relacionar les decisions que nosaltres prenem quan dissenyem un algorisme a la eficiència temporal de l'algorisme. E.g. si faig un bucle d’una manera en comptes d’una altra, quina eficiència tindrà?
  - Hi ha límits fonamentals en la quantitat de temps que necessitarem per resoldre un problema determinat?

Anàlisi del cost algorísmic

Què volem fer?

- Com podem raonar sobre un algorisme per tal de predir la quantitat de **temps** que necessitarà per resoldre un problema d'una **mida** determinada?
- Ens interessa poder relacionar les decisions que nosaltres prenem quan dissenyem un algorisme a la eficiència temporal de l'algorisme. E.g. si faig un bucle d'una manera en comptes d'una altra, quina eficiència tindrà?
  - Hi ha límits fonamentals en la quantitat de temps que necessitarem per resoldre un problema determinat?

Per què cal fer-ho?

- Abans, els ordinadors eren molt lents, s'havien de construir programes molt eficients per poder tenir algorismes que s'executessin en un temps raonable.
- Ara els ordinadors són molt més ràpids, són capaços d'executar programes molt més ràpidament
- Però el **volum de les dades** amb què tractem avui dia també ha crescut moltíssim.

Anàlisi del cost algorísmic

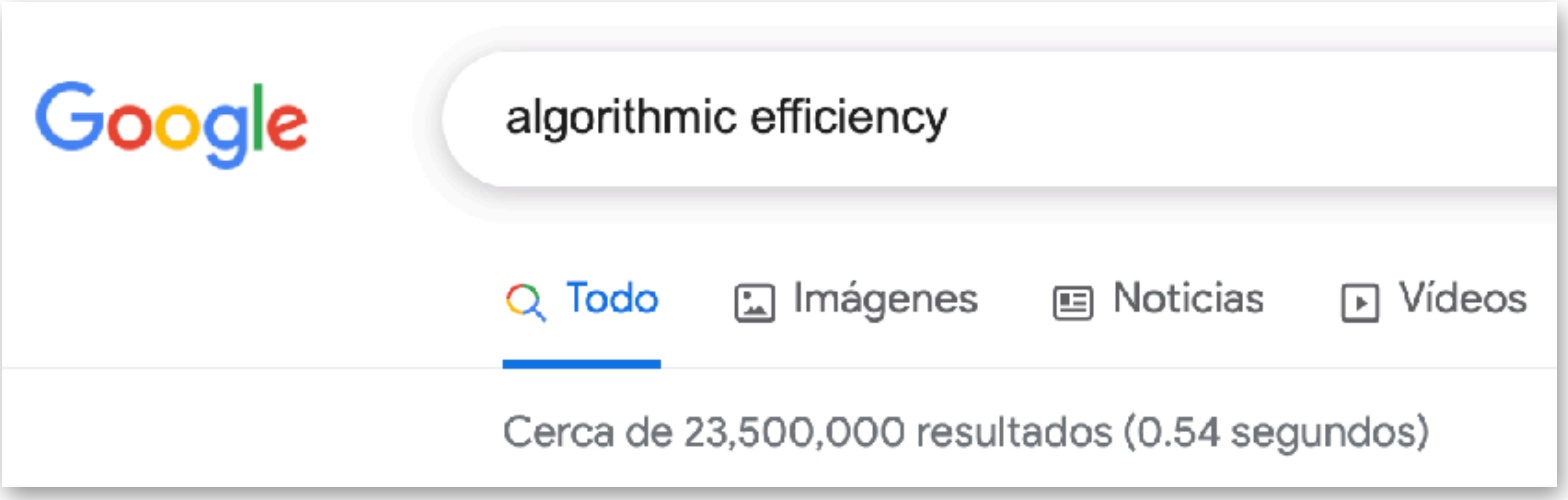
Què volem fer?

- Com podem raonar sobre un algorisme per tal de predir la quantitat de **temps** que necessitarà per resoldre un problema d'una **mida** determinada?
- Ens interessa poder relacionar les decisions que nosaltres prenem quan dissenyem un algorisme a la eficiència temporal de l'algorisme. E.g. si faig un bucle d'una manera en comptes d'una altra, quina eficiència tindrà?
  - Hi ha límits fonamentals en la quantitat de temps que necessitarem per resoldre un problema determinat?

Per què cal fer-ho?

- Abans, els ordinadors eren molt lents, s'havien de construir programes molt eficients per poder tenir algorismes que s'executessin en un temps raonable.
- Ara els ordinadors són molt més ràpids, són capaços d'executar programes molt més ràpidament
- Però el **volum de les dades** amb què tractem avui dia també ha crescut moltíssim.

  - E.g. al 2014, Google tenia indexades 30.000.000.000.000 pàgines (ocupant 100.000.000 GB) i tot i així és capaç de donar-nos un resultat de cerca en segons.
  - Quant hauríem trigat si haguéssim recorregut totes les pàgines fent servir força bruta?
  - Tot i tenir capacitat computacional, hem de dissenyar algorismes eficients preparats per treballar amb un gran volum de dades



Anàlisi del cost algorísmic

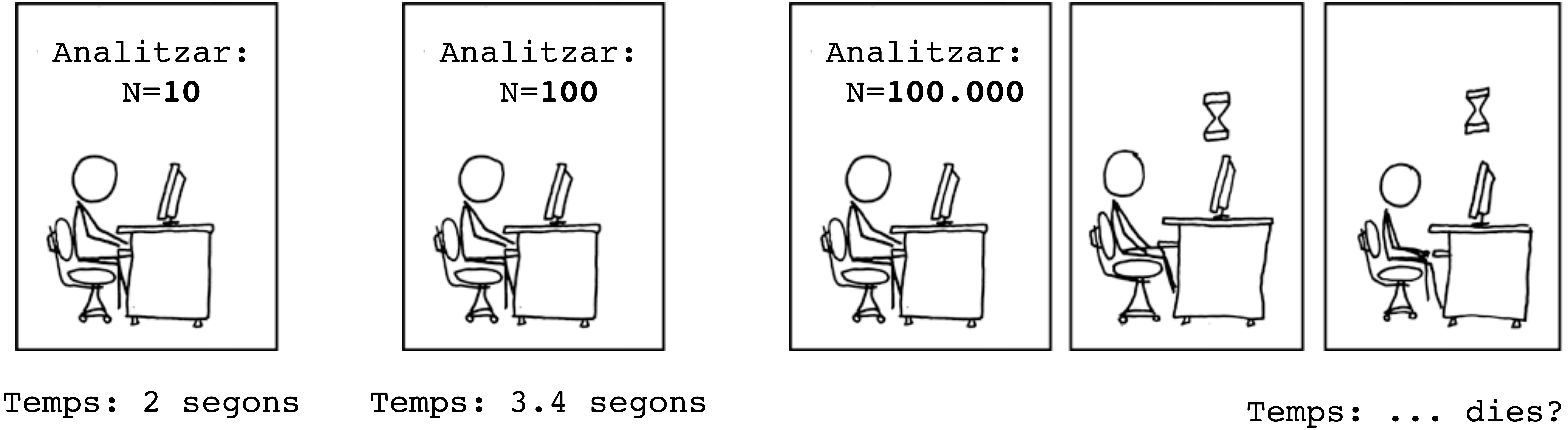
Tipus d'eficiència

- **Eficiència temporal**: quant triga l'algorisme a executar-se en funció de la mida de dades que ha de tractar

Anàlisi del cost algorísmic

Tipus d'eficiència

- **Eficiència temporal**: quant triga l'algorisme a executar-se en funció de la mida de dades que ha de tractar



Anàlisi del cost algorísmic

Tipus d'eficiència

- **Eficiència temporal**: quant triga l'algorisme a executar-se en funció de la mida de dades que ha de tractar
- **Eficiència espaial**: com d'eficient és l'ús de recursos de memòria per resoldre un problema donades unes dades

Anàlisi del cost algorísmic

Tipus d'eficiència

- **Eficiència temporal**: quant triga l'algorisme a executar-se en funció de la mida de dades que ha de tractar
- **Eficiència espaial**: com d'eficient és l'ús de recursos de memòria per resoldre un problema donades unes dades

```
Desar matriu d'adjacència de xarxa social
```

Anàlisi del cost algorísmic

Tipus d'eficiència

- **Eficiència temporal**: quant triga l'algorisme a executar-se en funció de la mida de dades que ha de tractar
- **Eficiència espaial**: com d'eficient és l'ús de recursos de memòria per resoldre un problema donades unes dades

`Desar matriu d'adjacència de xarxa social`

```
matriu =

     0     0     1     0     0
     0     0     0     1     0
     1     0     0     0     0
     0     1     0     0     1
     0     0     0     1     0
```

Anàlisi del cost algorísmic

Tipus d'eficiència

- **Eficiència temporal**: quant triga l'algorisme a executar-se en funció de la mida de dades que ha de tractar
- **Eficiència espaial**: com d'eficient és l'ús de recursos de memòria per resoldre un problema donades unes dades

```
Desar matriu d'adjacència de xarxa social
```

```
matriu =

      0      0      1      0      0
      0      0      0      1      0
      1      0      0      0      0
      0      1      0      0      1
      0      0      0      1      0
```

```
llista =

   (3,1)        1
   (4,2)        1
   (1,3)        1
   (2,4)        1
   (5,4)        1
   (4,5)        1
```

Anàlisi del cost algorísmic

Tipus d'eficiència

- **Eficiència temporal**: quant triga l'algorisme a executar-se en funció de la mida de dades que ha de tractar
- **Eficiència espaial**: com d'eficient és l'ús de recursos de memòria per resoldre un problema donades unes dades
- **Trade-off** entre eficiència temporal i espaial:
  - Normalment, aquelles solucions que tenen molt bon temps d'execució, requeriran l'ús de valors precalculats o tenir més dades en memòria.
  - En aquesta lliçó, ens centrarem en eficiència temporal.

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Cronometrar amb un timer**

```
#include <time.h>

    clock_t start, end;
    double cpu_time_used;

    start = clock();
    ... /* Do the work. */
    end = clock();
    cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
```

La llibreria <time.h> ens permet cronometrar un programa

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Cronometrar amb un timer**
- El temps d'execució varia entre diferents **algorismes** ✓

```c
#include <time.h>

    clock_t start, end;
    double cpu_time_used;

    start = clock();
    ... /* Do the work. */
    end = clock();
    cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
```

La llibreria <time.h> ens permet cronometrar un programa

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Cronometrar amb un timer**
- El temps d'execució varia entre diferents **algorismes**
- El temps d'execució varia en diferents **ordinadors**

```c
#include <time.h>

    clock_t start, end;
    double cpu_time_used;

    start = clock();
    ... /* Do the work. */
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

La llibreria <time.h> ens permet cronometrar un programa

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Cronometrar amb un timer**
- El temps d'execució varia entre diferents **algorismes**
- El temps d'execució varia en diferents **ordinadors**
- El temps d'execució **no es pot predir** a partir de proves que fem amb inputs petits (sabem que el temps serà diferent per mides d'input diferents, però no sabem quina és la relació entre la mida de l'input i el temps d'execució, o sigui, no podem fer una predicció).

```
#include <time.h>

     clock_t start, end;
     double cpu_time_used;

     start = clock();
     ... /* Do the work. */
     end = clock();
     cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
```

La llibreria <time.h> ens permet cronometrar un programa

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Cronometrar amb un timer**
- El temps d'execució varia entre diferents **algorismes**
- El temps d'execució varia en diferents **ordinadors**
- El temps d'execució **no es pot predir** a partir de proves que fem amb inputs petits (sabem que el temps serà diferent per mides d'input diferents, però no sabem quina és la relació entre la mida de l'input i el temps d'execució, o sigui, no podem fer una predicció).

```
#include <time.h>

    clock_t start, end;
    double cpu_time_used;

    start = clock();
    ... /* Do the work. */
    end = clock();
    cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
```

La llibreria <time.h> ens permet cronometrar un programa

Cronometrar avalua l'eficiència de l'algorisme per una implementació i una màquina concreta

Nosaltres volem avaluar com escala el temps d'execució d'un algorisme en funció de la mida de l'input

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- Calcular l'ordre de creixement d'un algorisme

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Calcular l'ordre de creixement d'un algorisme**
- Cada instrucció té un **cost**

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Calcular l'ordre de creixement d'un algorisme**
- Cada instrucció té un **cost**
- Només ens preocuparem de com és la performance de l'algorisme quan la mida del problema es fa molt gran: **comportament asimptòtic**

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Calcular l'ordre de creixement d'un algorisme**
- Cada instrucció té un **cost**
- Només ens preocuparem de com és la performance de l'algorisme quan la mida del problema es fa molt gran: **comportament asimptòtic**
- Necessitarem saber què mesurar, ja que les instruccions que executarem poden ser unes o d'altres en funció del flux

- Necessitarem saber què mesurar, ja que les instruccions que executarem poden ser unes o d'altres en funció del flux

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;
  trobat := fals;
  mentre (i<N) i (no(trobat)) fer
    si (t[i]==elem)
      trobat := cert;
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

Algorisme simple de cerca seqüencial

- Necessitarem saber què mesurar, ja que les instruccions que executarem poden ser unes o d'altres en funció del flux

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;
  trobat := fals;
  mentre (i<N) i (no(trobat)) fer
    si (t[i]==elem)
      trobat := cert;
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

Algorisme simple de cerca seqüencial

Millor cas

Si l'element **`elem`** està a la primera posició de la taula

Només hem d'explorar la primera posició de la taula

Pitjor cas

Si l'element **`elem`** no és a la taula

Haurem hagut de recórrer la taula sencera (N)

Cas average

Si l'element **`elem`** està al mig

Haurem hagut de recórrer mitja taula (N/2)

- Necessitarem saber què mesurar, ja que les instruccions que executarem poden ser unes o d'altres en funció del flux

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;
  trobat := fals;
  mentre (i<N) i (no(trobat)) fer
    si (t[i]==elem)
      trobat := cert;
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

Algorisme simple de cerca seqüencial

Millor cas

Si l'element `elem` està a la primera posició de la taula

Només hem d'explorar la primera posició de la taula

Pitjor cas

Si l'element `elem` no és a la taula

Haurem hagut de recórrer la taula sencera (N)

Cas average

Si l'element `elem` està al mig

Haurem hagut de recórrer mitja taula (N/2)

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Calcular l'ordre de creixement d'un algorisme**
- Cada instrucció té un **cost**
- Només ens preocuparem de com és la performance de l'algorisme quan la mida del problema es fa molt gran: **comportament asimptòtic**
- Necessitarem saber què mesurar, ja que les instruccions que executarem poden ser unes o d'altres en funció del flux
- Ens centrarem en el **pitjor cas**, que ens donarà la **cota superior** del temps que pot trigar en funció de l'entrada.

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Calcular l'ordre de creixement d'un algorisme**
- Cada instrucció té un **cost**
- Només ens preocuparem de com és la performance de l'algorisme quan la mida del problema es fa molt gran: **comportament asimptòtic**
- Necessitarem saber què mesurar, ja que les instruccions que executarem poden ser unes o d'altres en funció del flux
- Ens centrarem en el **pitjor cas**, que ens donarà la **cota superior** del temps que pot trigar en funció de l'entrada.
- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;
  trobat := fals;
  mentre (i<N) i (no(trobat)) fer
    si (t[i]==elem)
      trobat := cert;
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;
  trobat := fals;
  mentre (i<N) i (no(trobat)) fer
    si (t[i]==elem)
      trobat := cert;
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

1 instrucció (assignació)

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;                          1 instrucció (assignació)
  trobat := fals;                  1 instrucció (assignació)
  mentre (i<N) i (no(trobat)) fer
    si (t[i]==elem)
      trobat := cert;
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;
  trobat := fals;
  mentre (i<N) i (no(trobat)) fer
    si (t[i]==elem)
      trobat := cert;
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

1 instrucció (assignació)

1 instrucció (assignació)

2 instruccions (comparacions)

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;                                  → 1 instrucció (assignació)
  trobat := fals;                          → 1 instrucció (assignació)
  mentre (i<N) i (no(trobat)) fer          → 2 instruccions (comparacions)
    si (t[i]==elem)                        → 1 instrucció (comparació)
      trobat := cert;
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;                         →  1 instrucció (assignació)
  trobat := fals;                 →  1 instrucció (assignació)
  mentre (i<N) i (no(trobat)) fer →  2 instruccions (comparacions)
    si (t[i]==elem)               →  1 instrucció (comparació)
      trobat := cert;             →  1 instrucció (assignació)
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;                              → 1 instrucció (assignació)
  trobat := fals;                      → 1 instrucció (assignació)
  mentre (i<N) i (no(trobat)) fer      → 2 instruccions (comparacions)
    si (t[i]==elem)                    → 1 instrucció (comparació)
      trobat := cert;                  → 1 instrucció (assignació)
    fsi
    i:=i+1;                            → 1 instrucció (suma)
  fmentre
  retorna (trobat);
ffunció
```

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;                                  1 instrucció (assignació)
  trobat := fals;                          1 instrucció (assignació)
  mentre (i<N) i (no(trobat)) fer          2 instruccions (comparacions)
    si (t[i]==elem)                        1 instrucció (comparació)
      trobat := cert;                      1 instrucció (assignació)
    fsi
    i:=i+1;                                1 instrucció (suma)
  fmentre
  retorna (trobat);                        1 instrucció (retorn)
ffunció
```

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;                                 1 instrucció (assignació)
  trobat := fals;                         1 instrucció (assignació)
  mentre (i<N) i (no(trobat)) fer         2 instruccions (comparacions)
    si (t[i]==elem)                       1 instrucció (comparació)
      trobat := cert;                     1 instrucció (assignació)
    fsi
    i:=i+1;                               1 instrucció (suma)
  fmentre
  retorna (trobat);                       1 instrucció (retorn)
ffunció
```

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;                              → 1 instrucció (assignació)
  trobat := fals;                      → 1 instrucció (assignació)
  mentre (i<N) i (no(trobat)) fer      → 2 instruccions (comparacions)
    si (t[i]==elem)                    → 1 instrucció (comparació)
      trobat := cert;                  → 1 instrucció (assignació)
    fsi
    i:=i+1;                            → 1 instrucció (suma)
  fmentre
  retorna (trobat);                    → 1 instrucció (retorn)
ffunció
```

(2 instruccions (comparacions), 1 instrucció (comparació), 1 instrucció (assignació), 1 instrucció (suma)) → **n** vegades (pitjor cas)

Algorisme simple de cerca

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;
  trobat := fals;
  mentre (i<N) i (no(trobat)) fer
    si (t[i]==elem)
      trobat := cert;
    fsi
    i:=i+1;
  fmentre
  retorna (trobat);
ffunció
```

Algorisme simple de cerca

Codi	Cost	
`i := 0;`	1 instrucció (assignació)	
`trobat := fals;`	1 instrucció (assignació)	
`mentre (i<N) i (no(trobat)) fer`	2 instruccions (comparacions)	**n** vegades (pitjor cas)
`si (t[i]==elem)`	1 instrucció (comparació)	**n** vegades (pitjor cas)
`trobat := cert;`	1 instrucció (assignació)	**n** vegades (pitjor cas)
`i:=i+1;`	1 instrucció (suma)	**n** vegades (pitjor cas)
`retorna (trobat);`	1 instrucció (retorn)	

$$\mathbf{T(n)} = 1 + 1 + n(2+1+1+1) + 1 = \mathbf{5n + 3}$$

Funció de cost temporal

- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

```
funció cercar_element (t: taula[] d'enter,
N: enter, elem: enter) retorna booleà és
var
  i: enter;
  trobat: booleà;
fvar
inici
  i := 0;                          → 1 instrucció (assignació)
  trobat := fals;                  → 1 instrucció (assignació)
  mentre (i<N) i (no(trobat)) fer  → 2 instruccions (comparacions)
    si (t[i]==elem)                → 1 instrucció (comparació)
      trobat := cert;              → 1 instrucció (assignació)
    fsi
    i:=i+1;                        → 1 instrucció (suma)
  fmentre
  retorna (trobat);                → 1 instrucció (retorn)
ffunció
```

Algorisme simple de cerca

Les instruccions del bucle → **n** vegades (pitjor cas)

Terme dominant

$$\mathbf{T(n)} = 1 + 1 + n(2+1+1+1) + 1 = \mathbf{5n + 3}$$

Anàlisi del cost algorísmic

Com podem mesurar l'eficiència?

- **Calcular l'ordre de creixement d'un algorisme**
- Cada instrucció té un **cost**
- Només ens preocuparem de com és la performance de l'algorisme quan la mida del problema es fa molt gran: **comportament asimptòtic**
- Necessitarem saber què mesurar, ja que les instruccions que executarem poden ser unes o d'altres en funció del flux
- Ens centrarem en el **pitjor cas**, que ens donarà la **cota superior** del temps que pot trigar en funció de l'entrada.
- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)

Anàlisi del cost algorísmic

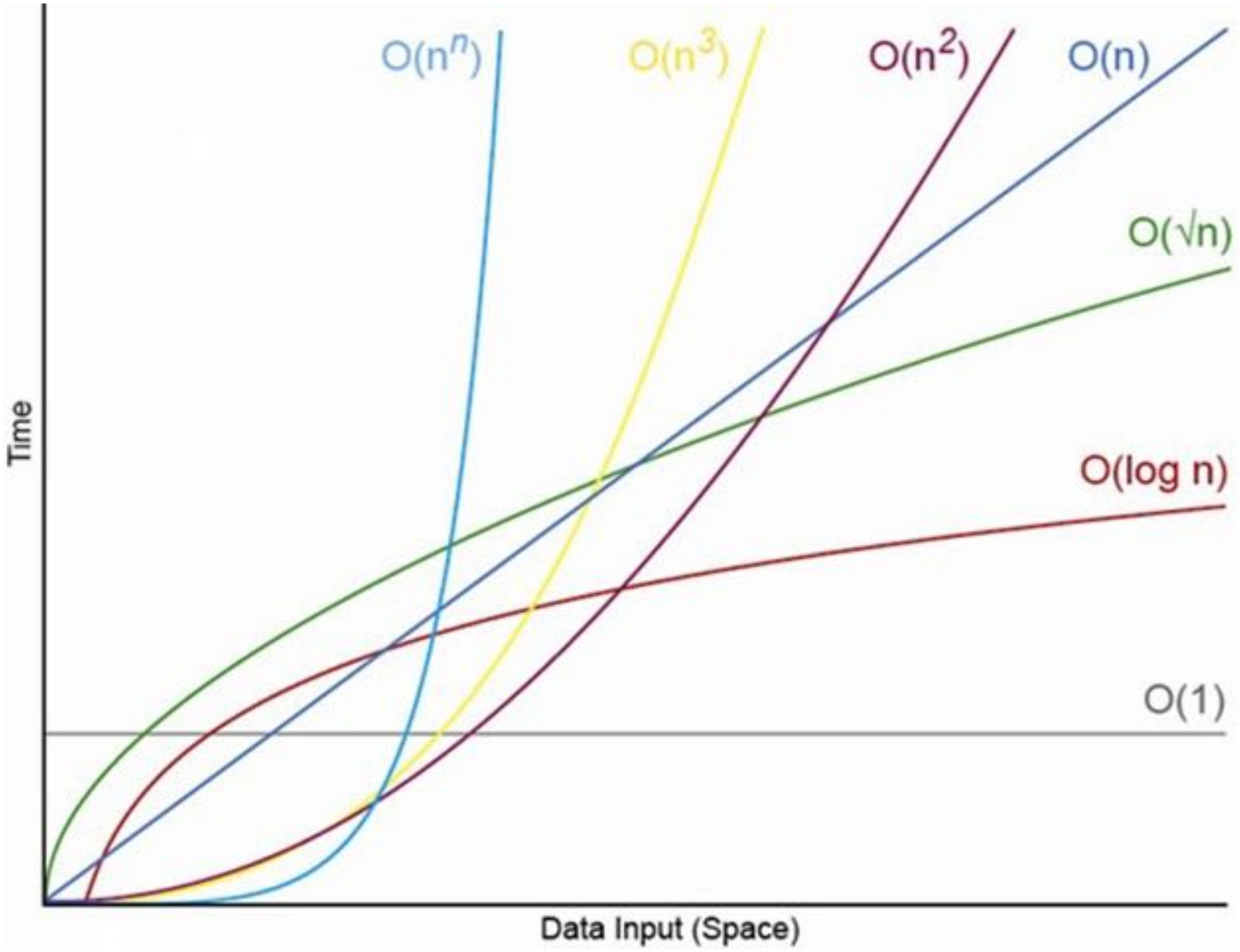
Com podem mesurar l'eficiència?

- **Calcular l'ordre de creixement d'un algorisme**
- Cada instrucció té un **cost**
- Només ens preocuparem de com és la performance de l'algorisme quan la mida del problema es fa molt gran: **comportament asimptòtic**
- Necessitarem saber què mesurar, ja que les instruccions que executarem poden ser unes o d'altres en funció del flux
- Ens centrarem en el **pitjor cas**, que ens donarà la **cota superior** del temps que pot trigar en funció de l'entrada.
- Només ens preocuparem dels **factors més grans** (quins trossos de codi són els que determinen el cost del programa?)
- Així podrem calcular l'**ordre de creixement del temps d'execució** (no el propi temps d'execució) **en funció de la mida de l'entrada.**

Notació asimptòtica

Big O notation

- La "Big O" mesura l'upper bound (cota superior) del creixement asimptòtic, també anomenat ordre de creixement



Ordres de complexitat

Constant	$\mathcal{O}(1)$
Logarítmic	$\mathcal{O}(\log_2 n)$
Lineal	$\mathcal{O}(n)$
Quasilineal	$\mathcal{O}(n \log_2 n)$
Quadràtic	$\mathcal{O}(n^2)$
Cúbic	$\mathcal{O}(n^3)$
Polinòmic	$\mathcal{O}(n^k)$, amb k conegut
Exponencial	$\mathcal{O}(k^n)$, amb k conegut
Factorial	$\mathcal{O}(n!)$
No afitat	$\mathcal{O}(\infty)$

- Comparant els ordres de creixement de diferents algorismes podem saber quins es comportaran millor quan l'input sigui molt gran.

Exemples

Classe de complexitat		n = 10	n = 100	n = 1000	n = 1,000,000
Constant	O(1)	1	1	1	1
Logarítmic	O(log n)	1	2	3	6
Lineal	O(n)	10	100	1000	1,000,000
Quasi-lineal	O(n log n)	10	200	3000	6,000,000
Quadràtic	O(n^2)	100	10,000	1,000,000	1,000,000,000,000
Exponencial	O(2^n)	1024	126765060 022822940 149670320 5376	1071508607186267320948425049060001 8105614048117055336074437503883703 5105112493612249319837881569585812 7594672917553146825187145285692314 0435984577574698574803934567774 8 2423098542107460506237114187795418 2153046474983581941267398767559165 5439460770629145711964776865421676 604298316526243868372056680693 76	☠️

Notació asimptòtica

Com es calcula l'ordre de complexitat?

Notació asimptòtica

Com es calcula l'ordre de complexitat?

- Analitzar cadascuna de les instruccions
- Aplicar unes regles i prendre el cost màxim (terme dominant)

Notació asimptòtica

Com es calcula l'ordre de complexitat?

- Analitzar cadascuna de les instruccions
- Aplicar unes regles i prendre el cost màxim (terme dominant)

Regla de la suma:

- Es fa servir en instruccions que s'executen seqüencialment

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Rightarrow f_1 + f_2 \in O(\text{MÀXIM}(g_1, g_2)).$$

Notació asimptòtica

Com es calcula l'ordre de complexitat?

- Analitzar cadascuna de les instruccions
- Aplicar unes regles i prendre el cost màxim (terme dominant)

Regla de la suma:

- Es fa servir en instruccions que s'executen seqüencialment

$$\left.\begin{aligned} f_1 &\in O(g_1) \\ f_2 &\in O(g_2) \end{aligned}\right\} \Rightarrow f_1 + f_2 \in O(\text{MÀXIM}(g_1, g_2)).$$

```
...
per (i:=0; i<n; i++)
  escriure("a");
fper
per (i:=0; i<n*n; i++)
  escriure("b");
fper
...
```

Notació asimptòtica

Com es calcula l'ordre de complexitat?

- Analitzar cadascuna de les instruccions
- Aplicar unes regles i prendre el cost màxim (terme dominant)

Regla de la suma:

- Es fa servir en instruccions que s'executen seqüencialment

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Rightarrow f_1 + f_2 \in O(\text{MÀXIM}(g_1, g_2)).$$

```
...
per (i:=0; i<n; i++)
  escriure("a");          → O(n)
fper
per (i:=0; i<n*n; i++)   → O(n²)
  escriure("b");
fper
...
```

Notació asimptòtica

Com es calcula l'ordre de complexitat?

- Analitzar cadascuna de les instruccions
- Aplicar unes regles i prendre el cost màxim (terme dominant)

Regla de la suma:

- Es fa servir en instruccions que s'executen seqüencialment

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Rightarrow f_1 + f_2 \in O(\text{MÀXIM}(g_1, g_2)).$$

```
...
per (i:=0; i<n; i++)
  escriure("a");
fper
per (i:=0; i<n*n; i++)
  escriure("b");
fper
...
```

$O(n)$

$O(n^2)$

$O(n + n^2) =$
$= O(n^2)$

Notació asimptòtica

Com es calcula l'ordre de complexitat?

- Analitzar cadascuna de les instruccions
- Aplicar unes regles i prendre el cost màxim (terme dominant)

Regla de la suma:

- Es fa servir en instruccions que s'executen seqüencialment

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Rightarrow f_1 + f_2 \in O(\text{MÀXIM}(g_1, g_2)).$$

```
...
per (i:=0; i<n; i++)
   escriure("a");
fper
per (i:=0; i<n*n; i++)
   escriure("b");
fper
...
```

$\texttt{escriure("a");} \rightarrow O(n)$

$\texttt{per (i:=0; i<n*n; i++)} \rightarrow O(n^2)$

$O(n) , O(n^2) \rightarrow O(n + n^2) =$

$= O(n^2)$

Regla del producte

- Es fa servir en instruccions que estan anidades

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Longrightarrow f_1 * f_2 \in O(g_1 * g_2)$$

Notació asimptòtica

Com es calcula l'ordre de complexitat?

- Analitzar cadascuna de les instruccions
- Aplicar unes regles i prendre el cost màxim (terme dominant)

Regla de la suma:

- Es fa servir en instruccions que s'executen seqüencialment

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Rightarrow f_1 + f_2 \in O(\text{MÀXIM}(g_1, g_2)).$$

```
...
per (i:=0; i<n; i++)
   escriure("a");
fper
per (i:=0; i<n*n; i++)
   escriure("b");
fper
...
```

$O(n)$

$O(n^2)$

$O(n + n^2) =$
$= O(n^2)$

Regla del producte

- Es fa servir en instruccions que estan anidades

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Longrightarrow f_1 * f_2 \in O(g_1 * g_2)$$

```
...
per (i:=0; i<n; i++)
   per (j:=0; i<n; i++)
      escriure("Hola");
   fper
fper
...
```

Notació asimptòtica

Com es calcula l'ordre de complexitat?

- Analitzar cadascuna de les instruccions
- Aplicar unes regles i prendre el cost màxim (terme dominant)

Regla de la suma:

- Es fa servir en instruccions que s'executen seqüencialment

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Rightarrow f_1 + f_2 \in O(\text{MÀXIM}(g_1, g_2)).$$

```
...
per (i:=0; i<n; i++)
   escriure("a");
fper
per (i:=0; i<n*n; i++)
   escriure("b");
fper
...
```

escriure("a") → $O(n)$

per (i:=0; i<n*n; i++) → $O(n^2)$

→ $O(n + n^2) =$
$= O(n^2)$

Regla del producte

- Es fa servir en instruccions que estan anidades

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Longrightarrow f_1 * f_2 \in O(g_1 * g_2)$$

```
...
per (i:=0; i<n; i++)
   per (j:=0; i<n; i++)
      escriure("Hola");
   fper
fper
...
```

per (i:=0; i<n; i++) → $O(n)$

per (j:=0; i<n; i++) → $O(n)$

Notació asimptòtica

Com es calcula l'ordre de complexitat?

- Analitzar cadascuna de les instruccions
- Aplicar unes regles i prendre el cost màxim (terme dominant)

Regla de la suma:

- Es fa servir en instruccions que s'executen seqüencialment

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Rightarrow f_1 + f_2 \in O(\text{MÀXIM}(g_1, g_2)).$$

```
...
per (i:=0; i<n; i++)
   escriure("a");
fper
per (i:=0; i<n*n; i++)
   escriure("b");
fper
...
```

$\texttt{escriure("a");} \rightarrow O(n)$

$\texttt{per (i:=0; i<n*n; i++)} \rightarrow O(n^2)$

$\rightarrow O(n + n^2) = O(n^2)$

Regla del producte

- Es fa servir en instruccions que estan anidades

$$\left.\begin{array}{l} f_1 \in O(g_1) \\ f_2 \in O(g_2) \end{array}\right\} \Longrightarrow f_1 * f_2 \in O(g_1 * g_2)$$

```
...
per (i:=0; i<n; i++)
   per (j:=0; i<n; i++)
      escriure("Hola");
   fper
fper
...
```

$\texttt{per (i:=0; i<n; i++)} \rightarrow O(n)$

$\texttt{per (j:=0; i<n; i++)} \rightarrow O(n)$

$\rightarrow O(n*n) = O(n^2)$

Notació asimptòtica

Costos de les construccions algorísmiques

Assignació, comparació, lectura, escriptura

- Cost constant: O(1)

Notació asimptòtica

Costos de les construccions algorísmiques

Assignació, comparació, lectura, escriptura

- Cost constant: O(1)

Seqüència d'instruccions i_1, i_2, i_3,…

- Aplicar regla de la suma: $\texttt{cost}(i_1, i_2, i_3) = \texttt{max}(\ \texttt{cost}(i_1), \texttt{cost}(i_2), \texttt{cost}(i_3)\ )$

Notació asimptòtica

Costos de les construccions algorísmiques

Assignació, comparació, lectura, escriptura

- Cost constant: O(1)

Seqüència d'instruccions `i₁, i₂, i₃,…`

- Aplicar regla de la suma: `cost(i₁, i₂, i₃) = max( cost(i₁), cost(i₂), cost(i₃) )`

Condicionals

<u>si</u> condició <u>llavors</u> ≡
<u>sino</u> ≡
<u>fsi</u>

- `Cost (condició) + max (cost(llavors), cost(sino) )`

Notació asimptòtica

Costos de les construccions algorísmiques

Iteracions

Notació asimptòtica

Costos de les construccions algorísmiques

Iteracions

- Si sabem el nombre d'iteracions

```
per x des de A fins a Z fer
    ...
fper
```

$$\sum_{num\_iteracions} \texttt{cost(condició + cos) = num\_iteracions * cost(condició + cos)}$$

Notació asimptòtica

Costos de les construccions algorísmiques

Iteracions

- Si sabem el nombre d'iteracions

<u>per</u> x <u>des de</u> A <u>fins a</u> Z <u>fer</u>

<u>fper</u>

$$\sum_{num\_iteracions} \texttt{cost(condició + cos) = num\_iteracions * cost(condició + cos)}$$

- Si **no** sabem el nombre d'iteracions

<u>mentre</u> condició <u>fer</u>

<u>fimentre</u>

`max_iteracions * cost (cos en el pitjor cas)`

Notació asimptòtica

Exercici: calculeu el cost asimptòtic d'aquest algorisme

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

Notació asimptòtica

Solució

Comencem per les instruccions simples més internes

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

Notació asimptòtica

Solució

Comencem per les instruccions simples més internes

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

O(1)

Notació asimptòtica

Solució

Comencem per les instruccions simples més internes

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];            → O(1)
    k := i;               → O(1)
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

Notació asimptòtica

Solució

Comencem per les instruccions simples més internes

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];            → O(1)
    k := i;               → O(1)
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];          → O(1)
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

Notació asimptòtica

Solució

Comencem per les instruccions simples més internes

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];            → O(1)
    k := i;               → O(1)
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];          → O(1)
        k:=j;             → O(1)
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

Notació asimptòtica

Solució

Comencem per les instruccions simples més internes

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];                                O(1)
    k := i;                                   O(1)
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];                              O(1)
        k:=j;                                 O(1)
      fsi
    fper
    A[k] := A[j];                             O(1)
    A[i] := x;
  fper
facció
```

Notació asimptòtica

Solució

Comencem per les instruccions simples més internes

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];                                  O(1)
    k := i;                                     O(1)
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];                                O(1)
        k:=j;                                   O(1)
      fsi
    fper
    A[k] := A[j];                               O(1)
    A[i] := x;                                  O(1)
  fper
facció
```

Notació asimptòtica

Solució

Anem pujant un nivell: aquí ens trobem un condicional.

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];                O(1)
    k := i;                   O(1)
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];              O(1)
        k:=j;                 O(1)
      fsi
    fper
    A[k] := A[j];             O(1)
    A[i] := x;                O(1)
  fper
facció
```

Notació asimptòtica

Solució

Anem pujant un nivell: aquí ens trobem un condicional.

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

O(1)

O(1)

O(1)

O(1)

O(1)

O(1)

Notació asimptòtica

Solució

Anem pujant un nivell: aquí ens trobem un condicional.

- La comparació és una instrucció simple d'ordre O(1)

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];            → O(1)
    k := i;               → O(1)
    per (j:=i+1; j<n; j++)
      si (A[j] < x)       → O(1)
        x:=A[j];          → O(1)
        k:=j;             → O(1)
      fsi
    fper
    A[k] := A[j];         → O(1)
    A[i] := x;            → O(1)
  fper
facció
```

Notació asimptòtica

Solució

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

O(1)

O(1)

O(1)

O(1)

O(1)

max = **O(1)**

O(1)

O(1)

Anem pujant un nivell: aquí ens trobem un condicional.

- La comparació és una instrucció simple d'ordre O(1)
- El cos del "si" són dues instruccions seqüencials (r.suma)

Notació asimptòtica

Solució

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];                      O(1)
    k := i;                         O(1)
    per (j:=i+1; j<n; j++)
      si (A[j] < x)                 O(1)
        x:=A[j];                    O(1)
        k:=j;                       O(1)   max = O(1)
      fsi
    fper
    A[k] := A[j];                   O(1)
    A[i] := x;                      O(1)
  fper
facció
```

Anem pujant un nivell: aquí ens trobem un condicional.

- La comparació és una instrucció simple d'ordre O(1)
- El cos del "si" són dues instruccions seqüencials (r.suma)
- Com resolem un condicional:

  `cost (condició) + max (cost(llavors), cost(sino))`

Notació asimptòtica

Solució

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];                      → O(1)
    k := i;                         → O(1)
    per (j:=i+1; j<n; j++)
      si (A[j] < x)                 → O(1)
        x:=A[j];                    → O(1)
        k:=j;                       → O(1)
      fsi
    fper
    A[k] := A[j];                   → O(1)
    A[i] := x;                      → O(1)
  fper
facció
```

max = O(1)

Anem pujant un nivell: aquí ens trobem un condicional.

- La comparació és una instrucció simple d'ordre O(1)
- El cos del "si" són dues instruccions seqüencials (r.suma)
- Com resolem un condicional:

  `cost (condició) + max (cost(llavors), cost(sino))`

- Com que el '+' anterior denota la regla de la suma, realment el que fem és:

  `max (cost (condició), cost(llavors), cost(sino))`

Notació asimptòtica

Solució

```
acció qualsevol (A: taula d'enter, n: enter) és
var
   i,j,k,x : enter;
fvar
inici
   per (i:=0; i<n; i++)
      x := A[i];
      k := i;
      per (j:=i+1; j<n; j++)
         si (A[j] < x)
            x:=A[j];
            k:=j;
         fsi
      fper
      A[k] := A[j];
      A[i] := x;
   fper
facció
```

O(1)
O(1)
O(1)
O(1) O(1) max = **O(1)**
max = **O(1)**
O(1)
O(1)

Anem pujant un nivell: aquí ens trobem un condicional.

- La comparació és una instrucció simple d'ordre O(1)
- El cos del "si" són dues instruccions seqüencials (r.suma)
- Com resolem un condicional:

  `cost (condició) + max (cost(llavors), cost(sino))`

- Com que el '+' anterior denota la regla de la suma, realment el que fem és:

  `max (cost (condició), cost(llavors), cost(sino))`

Notació asimptòtica

Solució

Al nivell següent ens trobem un “per”

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

O(1)

O(1)

O(1)

O(1)

O(1)

max = **O(1)**

max = **O(1)**

O(1)

O(1)

Notació asimptòtica

Solució

Al nivell següent ens trobem un "per"

- Quantes vegades es fa aquest bucle? n-(i+1)= n-i-1

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];              → O(1)
    k := i;                 → O(1)
    per (j:=i+1; j<n; j++)  → num_iteracions = n - i -1 = O(n)
      si (A[j] < x)         → O(1)
        x:=A[j];            → O(1)
        k:=j;               → O(1)
      fsi
    fper
    A[k] := A[j];           → O(1)
    A[i] := x;              → O(1)
  fper
facció
```

max = **O(1)**

max = **O(1)**

Notació asimptòtica

Solució

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

Al nivell següent ens trobem un “per”

- Quantes vegades es fa aquest bucle? n-(i+1)= n-i-1
- Com que estan anidats fem servir la regla del producte

O(1)

O(1)

num_iteracions = n - i -1 = **O(n)**

O(1)

O(1)

max = **O(1)**

O(1)

max = **O(1)**

Regla del producte

O(n)*O(1) = O(n)

O(1)

O(1)

Notació asimptòtica

Solució

Ara podem calcular el cost de tota la seqüència de dins del "per"

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];          → O(1)
    k := i;             → O(1)
    per (j:=i+1; j<n; j++)   → num_iteracions = n - i -1 = O(n)
      si (A[j] < x)     → O(1)
        x:=A[j];        → O(1)
        k:=j;           → O(1)
      fsi
    fper
    A[k] := A[j];       → O(1)
    A[i] := x;          → O(1)
  fper
facció
```

max = **O(1)**

max = **O(1)**

Regla del producte
O(n)*O(1) = O(n)

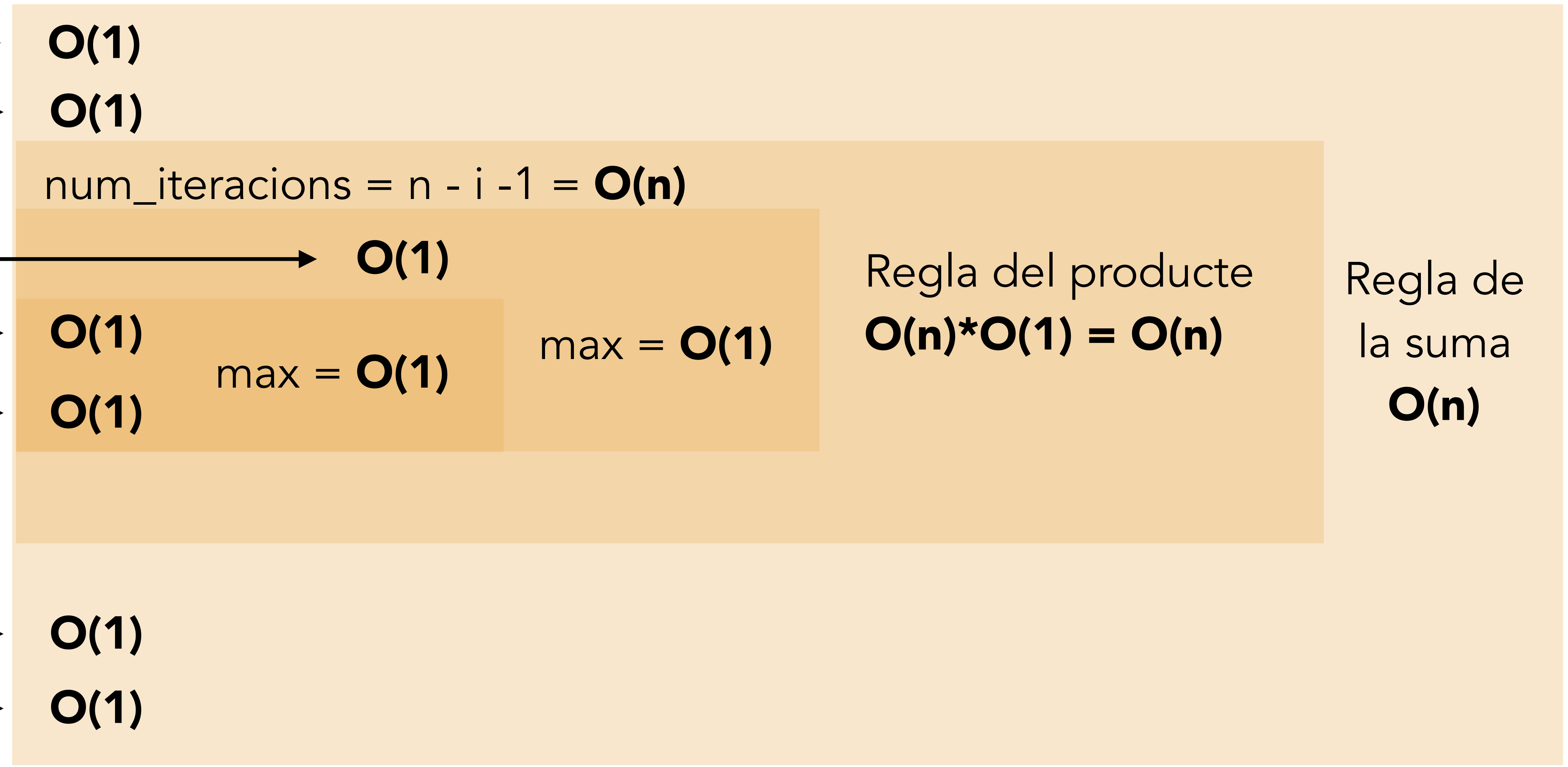
Notació asimptòtica

Solució

> Ara podem calcular el cost de tota la seqüència de dins del "per"

- Com que son instruccions seqüencials, es fa servir la regla de la suma: O(1) + O(1) + O(n) + O(1) + O(1) = O(n)

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

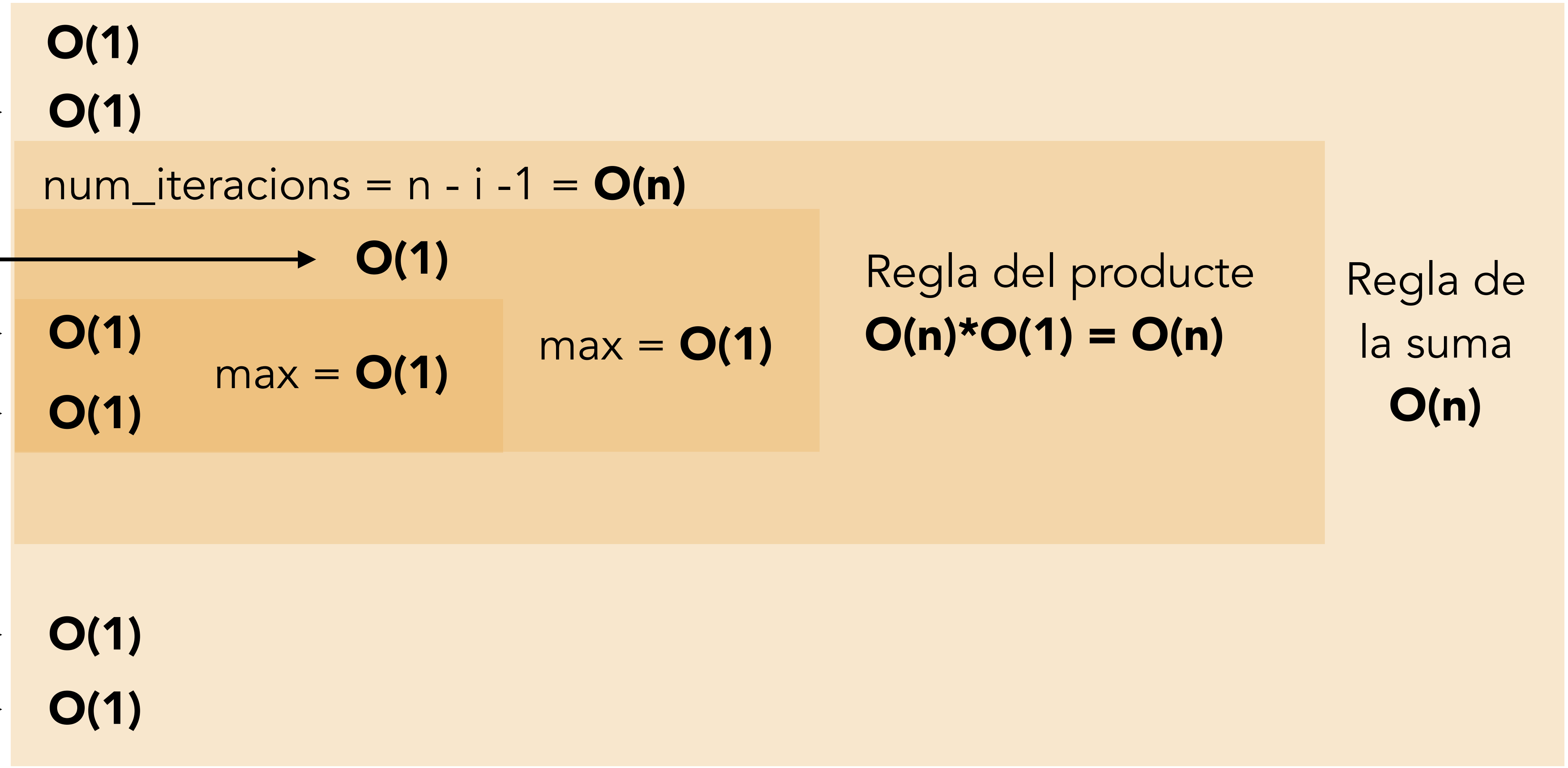


Notació asimptòtica

Solució

Ara ens centrem en el “per” més extern. Quantes vegades es fa? n. Per tant, O(n)

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```



Notació asimptòtica

Solució

Ara ens centrem en el “per” més extern. Quantes vegades es fa? n. Per tant, O(n)

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

num_iteracions = n = **O(n)**

O(1)

O(1)

num_iteracions = n - i -1 = **O(n)**

O(1)

O(1)

O(1)

max = **O(1)**

max = **O(1)**

Regla del producte
O(n)*O(1) = O(n)

Regla de la suma
O(n)

O(1)

O(1)

Notació asimptòtica

Solució

```
acció qualsevol (A: taula d'enter, n: enter) és
var
  i,j,k,x : enter;
fvar
inici
  per (i:=0; i<n; i++)
    x := A[i];
    k := i;
    per (j:=i+1; j<n; j++)
      si (A[j] < x)
        x:=A[j];
        k:=j;
      fsi
    fper
    A[k] := A[j];
    A[i] := x;
  fper
facció
```

Ara ens centrem en el "per" més extern. Quantes vegades es fa? n. Per tant, O(n)

- Com que el codi anterior que era O(n) està dins del bucle que es fa n vegades i també és O(n), aplicant la regla del producte tenim: O(n)*O(n) = O(n^2)