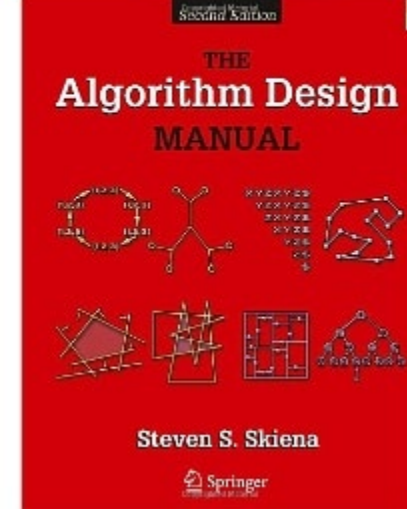


# Curs 5:

## Analiza eficienței algoritmilor (II)

# Motivatie



S. Skiena – The Algorithm Design Manual

<http://sist.sysu.edu.cn/~isslxm/DSA/textbook/Skienna.-TheAlgorithmDesignManual.pdf>

4-2. [3] For each of the following problems, give an algorithm that finds the desired numbers **within the given amount of time.** To keep your answers brief, feel free to use algorithms from the book as subroutines. For the example,  $S = \{6, 13, 19, 3, 8\}$ ,  $19 - 3$  maximizes the difference, while  $8 - 6$  minimizes the difference.

(a) Let  $S$  be an *unsorted* array of  $n$  integers. Give an algorithm that finds the pair  $x, y \in S$  that *maximizes*  $|x - y|$ . **Your algorithm must run in  $O(n)$  worst-case time.**

(b) Let  $S$  be a *sorted* array of  $n$  integers. Give an algorithm that finds the pair  $x, y \in S$  that *maximizes*  $|x - y|$ . **Your algorithm must run in  $O(1)$  worst-case time.**

(c) Let  $S$  be an *unsorted* array of  $n$  integers. Give an algorithm that finds the pair  $x, y \in S$  that *minimizes*  $|x - y|$ , for  $x \neq y$ . **Your algorithm must run in  $O(n \log n)$  worst-case time.**

(d) Let  $S$  be a *sorted* array of  $n$  integers. Give an algorithm that finds the pair  $x, y \in S$  that *minimizes*  $|x - y|$ , for  $x \neq y$ . **Your algorithm must run in  $O(n)$  worst-case time.**

# In cursul anterior...

... am văzut care sunt etapele principale ale analizei eficienței algoritmilor:

- Identificarea dimensiunii problemei
- Identificarea operației dominante
- Estimarea timpului de execuție (determinarea numărului de execuții ale operației dominante)
- Dacă timpul de execuție depinde de proprietățile datelor de intrare atunci se analizează:
  - Cel mai favorabil caz => margine inferioară a timpului de execuție
  - Cel mai defavorabil caz => margine superioară a timpului de execuție
  - Caz mediu=> timp mediu de execuție

# Azi vom vedea că...

- ... scopul principal al analizei eficienței algoritmilor este să se determine modul în care timpul de execuție al algoritmului crește o dată cu creșterea dimensiunii problemei
- ... pentru a obține această informație nu este necesar să se cunoască expresia detaliată a timpului de execuție ci este suficient să se identifice :
  - **Ordinul de creștere** al timpului de execuție (în raport cu dimensiunea problemei)
  - **Clasa de eficiența (complexitate)** căreia îi aparține algoritmul

# Structura

- Ce este ordinul de creștere ?
- Ce este analiza asimptotică ?
- Câteva notații asimptotice
- Analiza eficienței structurilor fundamentale de prelucrare
- Clase de eficiență
- Analiza empirică a eficienței algoritmilor

# Ce este ordinul de creștere ?

În expresia timpului de execuție există de regulă un termen care devine semnificativ mai mare decât ceilalți termeni atunci când dimensiunea problemei crește.

Acest termen este denumit **termen dominant** și el dictează comportarea algoritmului în cazul în care dimensiunea problemei devine mare

$$T_1(n) = an + b$$

Termen dominant:  $a n$

$$T_2(n) = a \log n + b$$

Termen dominant:  $a \log n$

$$T_3(n) = a n^2 + b n + c$$

Termen dominant:  $a n^2$

$$T_4(n) = a^n + b n + c$$

( $a > 1$ )

Termen dominant:  $a^n$

# Ce este ordinul de creștere ?

Sa analizăm ce se întâmplă cu termenul dominant când dimensiunea problemei crește de k ori :

$$T_1(n)=an$$

$$T'_1(kn)= a kn=k T_1(n)$$

$$T_2(n)=a \log n$$

$$T'_2(kn)=a \log(kn)=T_2(n)+a \log k$$

$$T_3(n)=a n^2$$

$$T'_3(kn)=a (kn)^2=k^2 T_3(n)$$

$$T_4(n)=a^n$$

$$T'_4(kn)=a^{kn}=(a^n)^k =T_4(n)^k$$

# Ce este ordinul de creștere ?

Ordinul de creștere exprimă cum crește termenul dominant al timpului de execuție în raport cu dimensiunea problemei

$$T'_1(kn) = a kn = k T'_1(n)$$

Ordin de creștere

Liniar

$$T'_2(kn) = a \log(kn) = T'_2(n) + a \log k$$

Logaritmic

$$T'_3(kn) = a (kn)^2 = k^2 T'_3(n)$$

Pătratic

$$T'_4(kn) = a^{kn} = (a^n)^k = (T'_4(n))^k$$

Exponențial



# Cum poate fi interpretat ordinul de creștere?

Când se compară doi algoritmi, cel având ordinul de creștere mai mic este considerat a fi mai eficient

Obs: comparația se realizează pentru **valori mari ale dimensiunii problemei (cazul asimptotic)**

**Exemplu.** Considerăm următoarele două expresii ale timpului de execuție

$$T_1(n) = 10n + 10 \quad (\text{ordin liniar de creștere})$$

$$T_2(n) = n^2 \quad (\text{ordin pătratic de creștere})$$

Dacă  $n \leq 10$  atunci  $T_1(n) > T_2(n)$

**In acest caz ordinul de creștere este relevant doar pentru  $n > 10$**

# O comparație a ordinelor de creștere

Diferite tipuri de dependență a timpului de execuție în raport cu dimensiunea problemei

$n$	$\log_2 n$	$n \log_2 n$	$n^2$	$2^n$	$n!$
10	3.3	33	100	1024	3628800
100	6.6	664	10000	$10^{30}$	$10^{157}$
1000	10	9965	1000000	$10^{301}$	$10^{2567}$
10000	13	132877	100000000	$10^{3010}$	$10^{35659}$

# O comparație a ordinelor de creștere

**Ipoteză:** fiecare operație este executată în  $10^{-9}$  sec

Obs: pt timpi de execuție care depind exponențial sau factorial de dimensiunea problemei prelucrarea devine imposibil de executat dacă  $n > 10$

n	$\log_2 n$	$n \log_2 n$	$n^2$	$2^n$	n!
10 $10^{-8}$ sec	3.3 $3 \cdot 10^{-9}$ sec	33 $3 \cdot 10^{-8}$ sec	100 $10^{-7}$ sec	1024 $10^{-6}$ sec	3628800 0.003 sec
100 $10^{-7}$ sec	6.6 $6 \cdot 10^{-9}$ sec	664 $6 \cdot 10^{-7}$ sec	10000 $10^{-5}$ sec	$10^{30}$ $10^{13}$ ani	$10^{157}$ $10^{140}$ ani
1000 $10^{-6}$ sec	10 $10^{-8}$ sec	9965 $9 \cdot 10^{-6}$ sec	1000000 0.001 sec	$10^{301}$ $10^{284}$ ani	$10^{2567}$ $10^{2550}$ ani
10000 $10^{-5}$ sec	13 $1.3 \cdot 10^{-8}$ sec	132877 $10^{-3}$ sec	100000000 0.1 sec	$10^{3010}$ $10^{2993}$ ani	$10^{35659}$ $10^{35642}$ ani

# Compararea ordinelor de creștere

Ordinele de creștere a doi timpi de execuție  $T1(n)$  și  $T2(n)$  pot fi comparate prin **calculul limitei raportului  $T1(n)/T2(n)$  când  $n$  tinde la infinit**

Daca limita este **0** atunci se poate spune ca  $T1(n)$  are un ordin de creștere mai mic decât  $T2(n)$

Daca limita este o **constantă finită strict pozitivă  $c$  ( $c > 0$ )** atunci se poate spune că  $T1(n)$  și  $T2(n)$  au același ordin de creștere

Daca limita este **infinită** atunci se poate spune ca  $T1(n)$  are un ordin de creștere mai mare decât  $T2(n)$

# Structura

- Ce este ordinul de creștere ?
- Ce este analiza asimptotică ?
- Cateva notații asimptotice
- Analiza eficienței structurilor fundamentale de prelucrare
- Clase de eficiență
- Analiza empirică a eficienței algoritmilor

# Ce este analiza asimptotică ?

- Analiza timpilor de execuție pentru valori mici ale dimensiunii problemei nu permite diferențierea dintre algoritmi eficienți și cei ineficienți
- Diferențele dintre ordinele de creștere devin din ce în ce mai semnificative pe măsură ce crește dimensiunea problemei
- **Analiza asimptotică** are ca scop studiul proprietăților timpului de execuție atunci când dimensiunea problemei tinde către infinit (problemă de dimensiune mare)

# Ce este analiza asimptotică ?

- In funcție de proprietățile timpului de execuție când dimensiunea problemei devine mare, algoritmul poate fi încadrat in diferite clase identificate prin niște **notații standard**
- Notațiile standard utilizate în identificarea diferitelor clase de eficiență sunt:
  - $\Theta$  (Theta)
  - $O$  (O)
  - $\Omega$  (Omega)

# Notația $\Theta$

Fie  $f, g: \mathbb{N} \rightarrow \mathbb{R}_+$  două funcții care depind de dimensiunea problemei și iau valori pozitive

**Definiție.**

$f(n) \in \Theta(g(n))$  dacă există  $c_1, c_2 > 0$  și  $n_0 \in \mathbb{N}$  astfel încât  
 $c_1 g(n) \leq f(n) \leq c_2 g(n)$  pentru orice  $n \geq n_0$

**Notație.** Frecvent, în locul simbolului de apartenență se folosește cel de egalitate:

$f(n) = \Theta(g(n))$  ( $f(n)$  are același ordin de creștere ca și  $g(n)$ )

**Exemple.**

1.  $T(n) = 3n+3 \Rightarrow T(n) \in \Theta(n)$  (sau  $T(n) = \Theta(n)$ )

$c_1=2, c_2=4, n_0=3, g(n)=n$

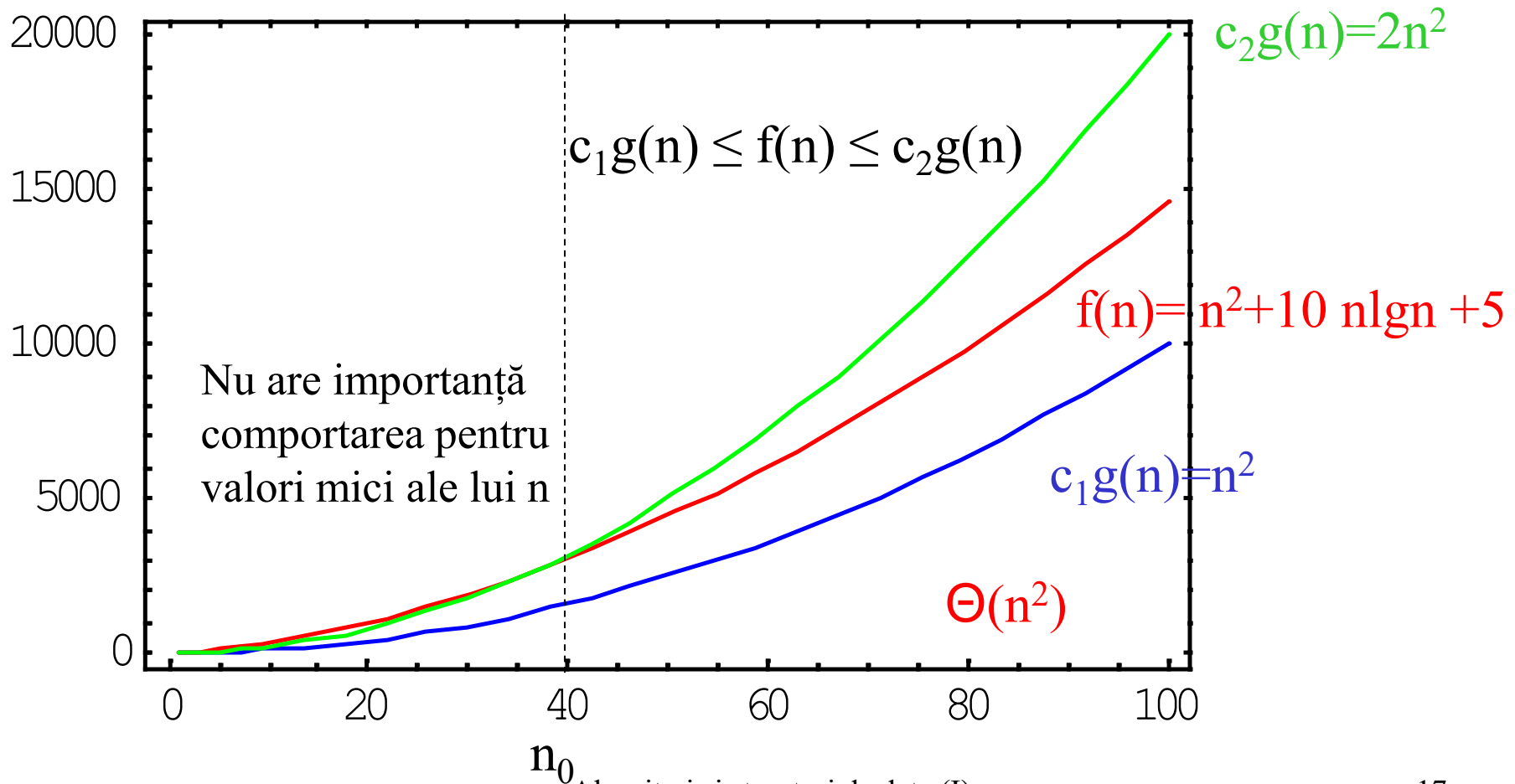
2.  $T(n) = n^2 + 10n \lg n + 5 \Rightarrow T(n) \in \Theta(n^2)$

$c_1=1, c_2=2, n_0=40, g(n)=n^2$



# Notația $\Theta$

Ilustrare grafică. Pentru valori mari ale lui  $n$ ,  $f(n)$  este mărginită, atât superior cât și inferior de  $g(n)$  înmulțit cu niște constante pozitive



# Notăția $\Theta$ . Proprietăți

1. Dacă  $T(n)=a_k n^k+a_{k-1} n^{k-1}+\dots+a_1 n+a_0$  atunci  $T(n) \in \Theta(n^k)$

**Dem.** Intrucât  $T(n)>0$  pentru orice  $n$  rezultă ca  $a_k>0$ .

Deci  $T(n)/n^k \rightarrow a_k$  (cand  $n \rightarrow \infty$ ).

Deci pentru orice  $\varepsilon>0$  (sufficient de mic) exista  $N(\varepsilon)$  astfel încât

$$|T(n)/n^k - a_k| < \varepsilon \Rightarrow a_k - \varepsilon < T(n)/n^k < a_k + \varepsilon \text{ pentru orice } n > N(\varepsilon)$$

Să presupunem că  $a_k - \varepsilon > 0$ .

Considerând  $c_1 = a_k - \varepsilon$ ,  $c_2 = a_k + \varepsilon$  și  $n_0 = N(\varepsilon)$  se obține

$$c_1 n^k < T(n) < c_2 n^k \text{ pentru orice } n > n_0, \text{ adică } T(n) \in \Theta(n^k)$$

# Notatia $\Theta$ . Proprietăți

2.  $\Theta(c g(n)) = \Theta(g(n))$  pentru orice constantă  $c$

Dem. Fie  $f(n) \in \Theta(cg(n))$ .

Atunci  $c_1 cg(n) \leq f(n) \leq c_2 cg(n)$  pentru orice  $n \geq n_0$ .

Considerând  $c'_1 = c c_1$  și  $c'_2 = c c_2$  se obține că  $f(n) \in \Theta(g(n))$ .

Astfel rezultă că  $\Theta(cg(n)) \subseteq \Theta(g(n))$ .

În mod similar se poate demonstra că  $\Theta(g(n)) \subseteq \Theta(cg(n))$ , adică  
 $\Theta(cg(n)) = \Theta(g(n))$ .

Cazuri particulare:

a)  $\Theta(c) = \Theta(1)$

b)  $\Theta(\log_a h(n)) = \Theta(\log_b h(n))$  pentru orice  $a, b > 1$

**Obs.** În stabilirea ordinului de complexitate, baza logaritmilor nu este relevantă, astfel că se va considera în majoritatea cazurilor că se lucrează cu baza 2.

# Notația $\Theta$ . Proprietăți

3.  $f(n) \in \Theta(f(n))$  (reflexivitate)

4.  $f(n) \in \Theta(g(n)) \Rightarrow g(n) \in \Theta(f(n))$  (simetrie)

5.  $f(n) \in \Theta(g(n))$ ,  $g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$  (tranzitivitate)

6.  $\Theta(f(n)+g(n)) = \Theta(\max\{f(n),g(n)\})$

# Notăția $\Theta$ . Alte exemple

3.  $3n \leq T(n) \leq 4n-1 \Rightarrow T(n) \in \Theta(n)$   
 $c_1=3, c_2=4, n_0=1$

4. Inmulțirea a doua matrici:  $T(m,n,p)=4mnp+5mp+4m+2$

Extinderea definiției (în cazul în care dimensiunea problemei depinde de mai multe valori):

$f(m,n,p) \in \Theta(g(m,n,p))$  dacă există

$c_1, c_2 > 0$  și  $m_0, n_0, p_0 \in \mathbb{N}$  astfel încât

$c_1 g(m,n,p) \leq f(m,n,p) \leq c_2 g(m,n,p)$  pentru orice  $m \geq m_0, n \geq n_0, p \geq p_0$

Astfel  $T(m,n,p) \in \Theta(mnp)$

5. Căutare secvențială:  $6 \leq T(n) \leq 3(n+1)$  (sau  $4 \leq T(n) \leq 2n+2$ )

Dacă  $T(n)=6$  atunci nu se poate găsi  $c_1$  astfel încât  $6 \geq c_1 n$  pentru valori suficient de mari ale lui  $n$ . Rezultă că  $T(n)$  nu aparține lui  $\Theta(n)$ .

Obs: Există timpi de execuție (algoritmi) care nu aparțin unei clase de tip  $\Theta$

# Notația O

Definiție.

$f(n) \in O(g(n))$  dacă există  $c > 0$  și  $n_0 \in \mathbb{N}$  astfel încât  
 $f(n) \leq c \cdot g(n)$  pentru orice  $n \geq n_0$

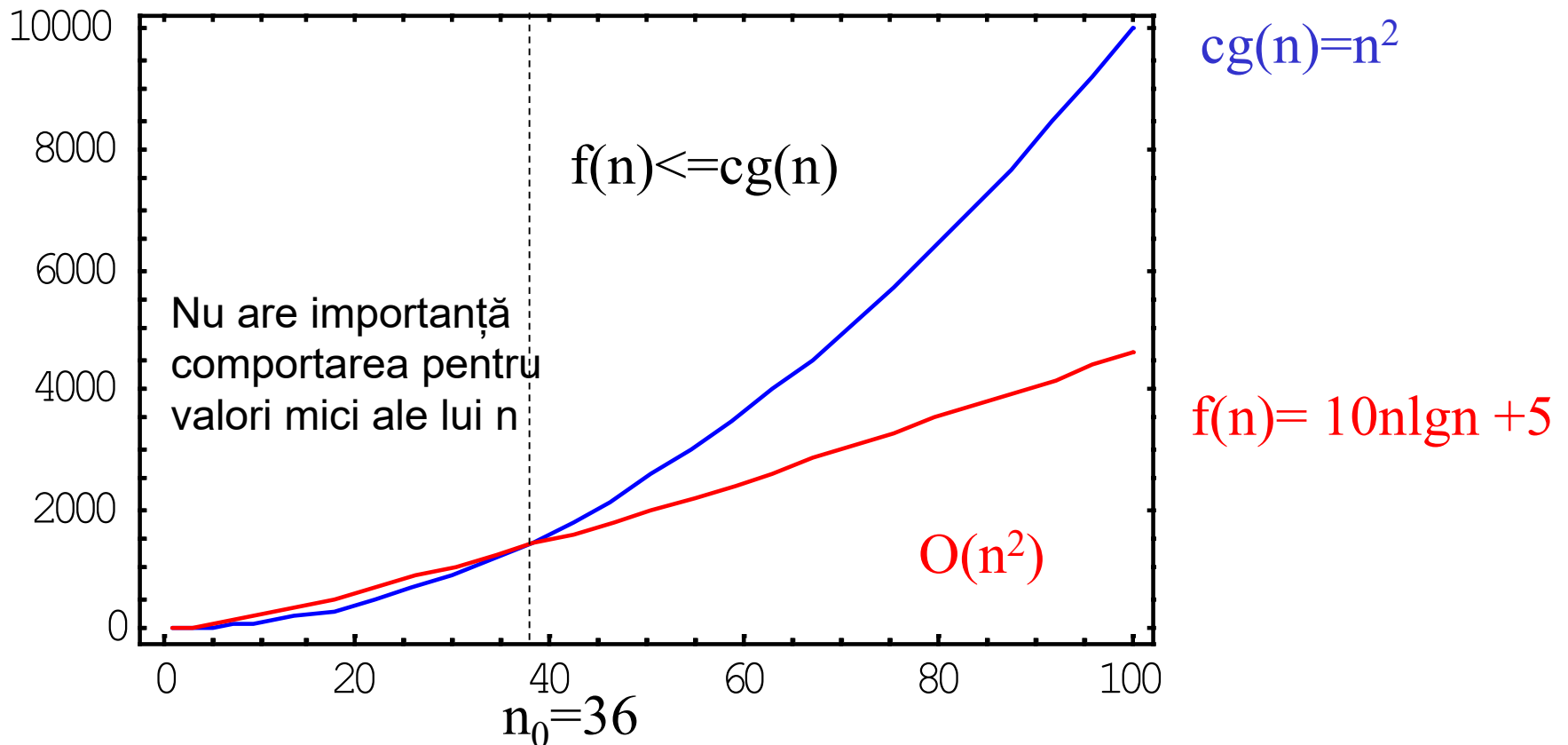
Notație.  $f(n) = O(g(n))$  ( $f(n)$  are un ordin de creștere cel mult egal cu cel al lui  $g(n)$ )

Exemple.

1.  $T(n) = 3n + 3 \Rightarrow T(n) \in O(n)$   
 $c=4, n_0=3, g(n)=n$
2.  $6 \leq T(n) \leq 3(n+1) \Rightarrow T(n) \in O(n)$  (interesează doar marginea superioară a timpului de execuție)  
 $c=4, n_0=3, g(n)=n$

# Notatia O

Ilustrare grafica. Pentru valori mari ale lui  $n$ ,  $f(n)$  este marginită superior de  $g(n)$  multiplicată cu o constantă pozitivă



# Notația O. Proprietăți

1. Dacă  $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$   
atunci  $T(n) \in O(n^d)$  pentru orice  $d \geq k$

**Dem.** Intrucât  $T(n) > 0$  pentru orice  $n$ , rezultă că  $a_k > 0$ .

Atunci  $T(n)/n^k \rightarrow a_k$  (când  $n \rightarrow \infty$ ).

Deci pentru orice  $\varepsilon > 0$  rezultă că există  $N(\varepsilon)$  astfel încât

$$T(n)/n^k \leq a_k + \varepsilon \text{ pentru orice } n > N(\varepsilon)$$

Prin urmare  $T(n) \leq (a_k + \varepsilon)n^k \leq (a_k + \varepsilon)n^d$

Considerând  $c = a_k + \varepsilon$  și  $n_0 = N(\varepsilon)$  rezultă că

$$T(n) < cn^d \text{ pentru orice } n > n_0, \text{ adică } T(n) \in O(n^d)$$

**Exemplu.**

$n \in O(n^2)$  (afirmația este corectă matematic însă este mai util în practică să se considere o margine **mai strânsă**, adică  $n \in O(n)$ )



# Notatia O. Proprietăți

2.  $f(n) \in O(f(n))$  (reflexivitate)

3.  $f(n) \in O(g(n))$  ,  $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$  (tranzitivitate)

4.  $\Theta(g(n))$  este inclusă în  $O(g(n))$

**Obs.** Incluziunea de mai sus este strictă: există elemente din  $O(g(n))$  care nu aparțin lui  $\Theta(g(n))$

**Exemplu:**

$$f(n)=10n\lg n+5, \quad g(n)=n^2$$

$$f(n) \leq g(n) \text{ pentru orice } n \geq 36 \Rightarrow f(n) \in O(g(n))$$

Dar nu există constante  $c$  și  $n_0$  astfel încât:

$$cn^2 \leq 10n\lg n+5 \text{ pentru orice } n \geq n_0 \text{ (deci } f(n) \text{ nu este și din } \Theta(g(n)))$$

# Notatia O. Proprietăți

Dacă prin analizarea celui mai defavorabil caz se obține:

$T(n) \leq g(n)$  atunci se poate spune despre  $T(n)$  că aparține lui  $O(g(n))$

**Exemplu.** Căutare secvențială:  $6 \leq T(n) \leq 3(n+1)$  (sau  $4 \leq T(n) < 2(n+1)$  - în funcție de varianta de algoritm folosită și de operațiile contorizate – vezi Curs 4)

Deci algoritmul căutării secvențiale este din clasa  $O(n)$   
(sau are ordin liniar de complexitate)

# Notația $\Omega$

Definiție.

$f(n) \in \Omega(g(n))$  dacă există  $c > 0$  și  $n_0 \in \mathbb{N}$  astfel încât  
 $cg(n) \leq f(n)$  pentru orice  $n \geq n_0$

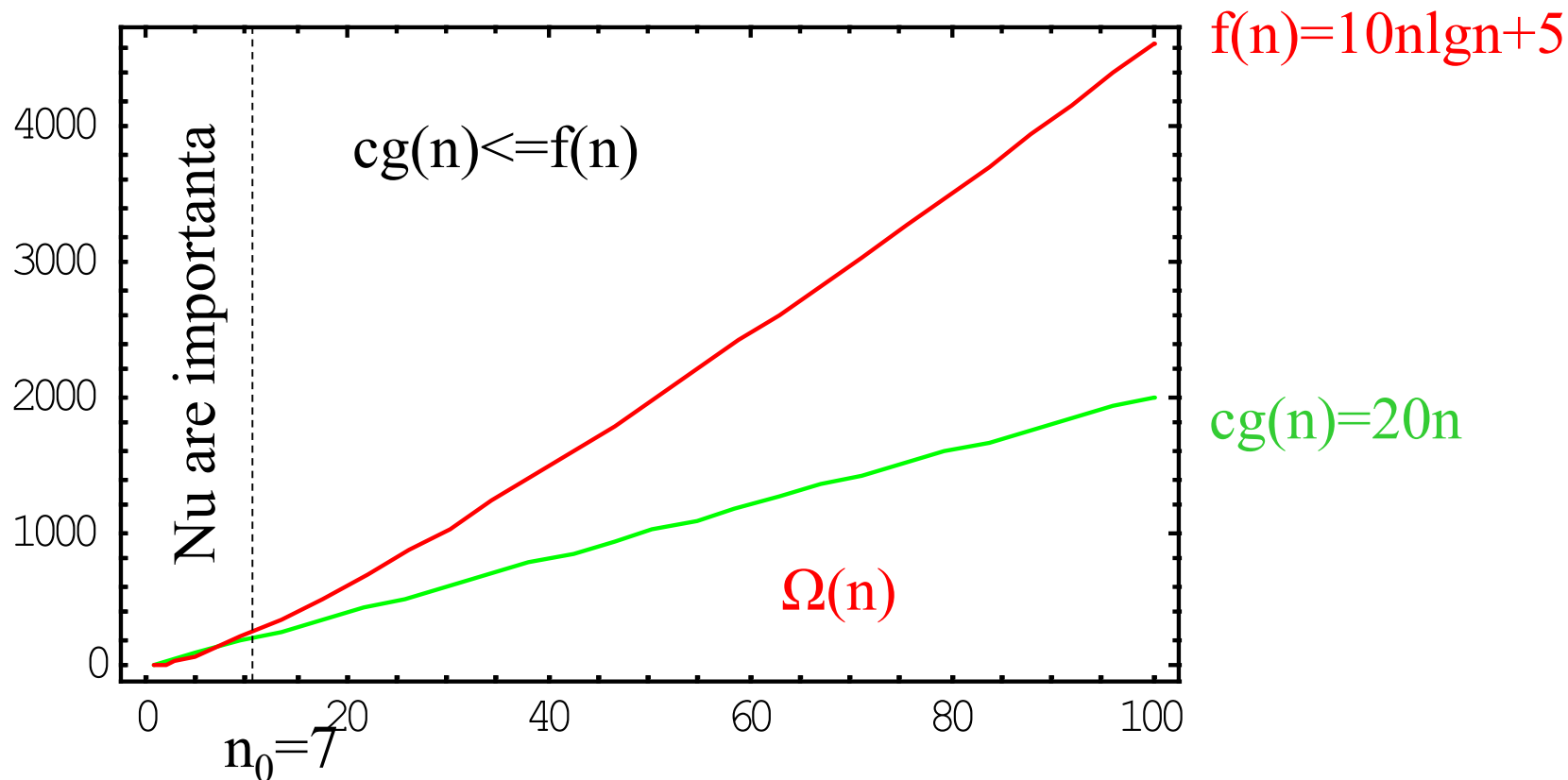
Notație.  $f(n) = \Omega(g(n))$  (ordinul de creștere al lui  $f(n)$  este cel puțin la fel de mare ca cel al lui  $g(n)$ )

Exemple.

1.  $T(n) = 3n+3 \Rightarrow T(n) \in \Omega(n)$   
 $c=3, n_0=1, g(n)=n$
2.  $6 \leq T(n) \leq 3(n+1) \Rightarrow T(n) \in \Omega(1)$   
 $c=6, n_0=1, g(n)=1$

# Notația $\Omega$

Ilustrare grafică. Pentru valori mari ale lui  $n$ , funcția  $f(n)$  este marginită inferior de  $g(n)$  multiplicată eventual de o constantă pozitivă



# Notatia $\Omega$ . Proprietăți

1. Dacă  $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$   
atunci  $T(n) \in \Omega(n^d)$  pentru orice  $d \leq k$

Dem. Intrucât  $T(n) > 0$  pentru orice  $n$  rezultă că  $a_k > 0$ .

Atunci  $T(n)/n^k \rightarrow a_k$  (cand  $n \rightarrow \infty$ ).

Astfel pentru orice  $\varepsilon > 0$  există  $N(\varepsilon)$  astfel încât

$$a_k - \varepsilon \leq T(n)/n^k \quad \text{pentru orice } n > N(\varepsilon)$$

In ipoteza că  $a_k - \varepsilon > 0$ , rezultă că  $(a_k - \varepsilon)n^d \leq (a_k - \varepsilon)n^k \leq T(n)$

Considerând  $c = a_k - \varepsilon$  si  $n_0 = N(\varepsilon)$  se obține

$$cn^d \leq T(n) \text{ pentru orice } n > n_0, \text{ adică } T(n) \in \Omega(n^d)$$

Exemplu.

$$n^2 \in \Omega(n)$$

# Notația $\Omega$ . Proprietăți

2.  $\Theta(g(n)) \subseteq \Omega(g(n))$

**Dem.** Este suficient să se ia în considerare marginea inferioară din definiția notației  $\Theta$

**Obs.** Incluziunea este strictă: există elemente ale lui  $\Omega(g(n))$  care nu aparțin lui  $\Theta(g(n))$

**Exemple:**

$$f(n) = 10n \lg n + 5, \quad g(n) = n$$

$$f(n) \geq 10g(n) \text{ pentru orice } n \geq 1 \Rightarrow f(n) \in \Omega(g(n))$$

Dar nu există constante  $c$  și  $n_0$  astfel încât:

$$10n \lg n + 5 \leq cn \text{ pentru orice } n \geq n_0$$

3.  $\Theta(g(n)) = O(g(n)) \cup \Omega(g(n))$

# Structura

- Ce este ordinul de creștere ?
- Ce este analiza asimptotică ?
- Cateva notații asimptotice
- Analiza eficienței structurilor fundamentale de prelucrare
- Clase de eficiență
- Analiza empirică a eficienței algoritmilor

# Analiza eficienței structurilor fundamentale de prelucrare

- Structura secvențială

A:

$A_1$	$\Theta(g_1(n))$	$O(g_1(n))$	$\Omega(g_1(n))$
$A_2$	$\Theta(g_2(n))$	$O(g_2(n))$	$\Omega(g_2(n))$
...	...	...	
$A_k$	$\Theta(g_k(n))$	$O(g_k(n))$	$\Omega(g_k(n))$

---

$$\Theta(\max\{g_1(n), g_2(n), \dots, g_k(n)\})$$

$$O(\max\{g_1(n), g_2(n), \dots, g_k(n)\})$$

$$\Omega(\max\{g_1(n), g_2(n), \dots, g_k(n)\})$$

**Obs:** se ia în considerare cea mai costisitoare prelucrare din secvență



# Analiza eficienței structurilor fundamentale de prelucrare

- Structura condițională

P:

IF <conditie>

THEN  $P_1$        $\Theta(g_1(n))$        $O(g_1(n))$        $\Omega(g_1(n))$

ELSE  $P_2$        $\Theta(g_2(n))$        $O(g_2(n))$        $\Omega(g_2(n))$

---

$O(\max\{g_1(n), g_2(n)\})$

$\Omega(\min\{g_1(n), g_2(n)\})$

**Obs:** dacă  $\Theta(g_1(n))$  este diferit de  $\Theta(g_2(n))$  prelucrarea nu poate fi încadrată într-o clasă de tip  $\Theta$  ci doar în celelalte două clase

# Analiza eficienței structurilor fundamentale de prelucrare

- Prelucrarea repetitivă

P:

```
FOR i ← 1, n DO
```

```
    P1                 $\Theta(1)$     →     $\Theta(n)$ 
```

```
FOR i ← 1, n DO
```

```
    FOR j ← 1, n DO
```

```
        P1                 $\Theta(1)$     →     $\Theta(n^2)$ 
```

**Obs:** In cazul a k cicluri for suprapuse a căror contor variază între 1 și n ordinul de complexitate este  $\Theta(n^k)$

# Analiza eficienței structurilor fundamentale de prelucrare

Obs.

Dacă limitele contorului sunt **variabile** atunci numărul de operații efectuate trebuie calculat explicit pentru fiecare dintre ciclurile suprapuse

Exemplu:

$m \leftarrow 1$

FOR  $i \leftarrow 1, n$  DO

$m \leftarrow 3 * m$                        $\{m=3^i\}$

FOR  $j \leftarrow 1, m$  DO

prelucrare de cost  $\Theta(1)$        $\{aceasta\ e\ operatia\ dominanta\}$

Ordinul de complexitate al prelucrării este:

$$3 + 3^2 + \dots + 3^n = (3^{n+1} - 1) / 2 - 1$$

adica  $\Theta(3^n)$

# Structura

- Ce este ordinul de creștere ?
- Ce este analiza asimptotică ?
- Cateva notații asimptotice
- Analiza eficienței structurilor fundamentale de prelucrare
- Clase de eficiență
- Analiza empirică a eficienței algoritmilor

# Clase de eficiență

Câteva dintre cele mai frecvente clase de eficiență (complexitate):

Denumire clasă	Notație asimptotică	Exemplu
logaritmic	$O(\lg n)$	Căutare binară → curs 9
liniar	$O(n)$	Căutare secvențială → curs 4
pătratic	$O(n^2)$	Sortare prin inserție → curs 6
cubic	$O(n^3)$	Inmulțirea a două matrici $n \times n$ → curs 4
exponențial	$O(2^n)$	Prelucrarea tuturor submultimilor unei mulțimi cu $n$ elemente → curs 13
factorial	$O(n!)$	Prelucrarea tuturor permutărilor de ordin $n$ → curs 9

# Exemplu

Se consideră un tablou cu  $n$  elemente,  $x[1..n]$  având valori din  $\{1, \dots, n\}$ . Tabloul poate avea toate elementele distincte sau poate exista o pereche de elemente cu aceeași valoare (o **singură** astfel de pereche). Să se verifice dacă elementele tabloului sunt toate distincte sau există o pereche de elemente identice.

**Exemplu:**  $n=5$ ,  $x=[2,1,4,1,3]$  nu are toate elementele distincte

$x=[2,1,4,5,3]$  are toate elementele distincte

Se pune problema identificării unui algoritm cât mai eficient din punct de vedere al timpului de execuție

# Exemplu

## Varianta 1:

verific(x[1..n])

$i \leftarrow 1$

$d \leftarrow \text{True}$

while (d==True) and (i<n) do

$d \leftarrow \text{NOT}(\text{caut}(x[i+1..n], x[i]))$

$i \leftarrow i+1$

endwhile

return d

Dim. problemei: n

$1 \leq T(n) \leq T'(n-1) + T'(n-2) + \dots + T'(1)$

$1 \leq T(n) \leq n(n-1)/2$

$T(n) \in \Omega(1)$ ,  $T(n) \in O(n^2)$

caut(x[s..f],v)

$i \leftarrow s$

while  $i < f$  AND  $x[i] \neq v$  do

$i \leftarrow i+1$

endwhile

if  $x[i] == v$  then return True

    else return False

endif

Dim. subproblemei:  $k = f - s + 1$

$1 \leq T'(k) \leq k$

Caz favorabil:  $x[1] = x[2]$

Caz defavorabil: elemente  
distincte

# Exemplu

## Varianta 2:

```
verific(x[1..n])
int f[1..n] // tabel frecvente
f[1..n] ← 0
for i ← 1 to n do
    f[x[i]] ← f[x[i]]+1
i ← 1
while f[i]<2 AND i<n do i ← i+1
if f[i]>=2 then return False
    else return True
endif
```

Dimensiune problema:  $n$

$n+1 \leq T(n) \leq 2n$

$T(n) \in \Theta(n)$

## Varianta 3:

```
verific3(x[1..n])
int f[1..n] // tabel frecvente
f[1..n] ← 0
i ← 1
while i<=n do
    f[x[i]] ← f[x[i]]+1
    if f[x[i]]>=2 then return False
    i ← i+1
endif
endwhile
return True
```

Dimensiune problema:  $n$

$4 \leq T(n) \leq 2n$

$T(n) \in O(n)$ ,  $T(n) \in \Omega(1)$



# Exemplu

## Varianta 4:

Variantele 2 și 3 necesită un spațiu suplimentar de memorie de dimensiune  $n$

Se poate rezolva problema în timp liniar dar fără a utiliza spațiu suplimentar de dimensiune  $n$  ci doar de dimensiune 1?

**Idee:** elementele sunt distincte doar dacă în tablou se află toate elementele din mulțimea  $\{1, 2, \dots, n\}$  adică suma lor este  $n(n+1)/2$

```
verific4(x[1..n])  
s ← 0  
for i ← 1 to n do s ← s + x[i] endfor  
if s = n(n+1)/2 then return True  
else return False
```

Endif

Dimensiune problema:  $n$

$T(n) = n$

$T(n) \in \Theta(n)$

Obs.

Varianta 4 este mai bună decât varianta 3 în raport cu spațiul de memorie utilizat însă în cazul mediu timpul de execuție este mai mic în varianta 3 decât în varianta 4

# Structura

- Ce este ordinul de creștere ?
- Ce este analiza asimptotică ?
- Cateva notații asimptotice
- Analiza eficienței structurilor fundamentale de prelucrare
- Clase de eficiență
- Analiza empirică a eficienței algoritmilor

# Analiza empirică a eficienței algoritmilor

Uneori analiza teoretică a eficienței este dificilă; în aceste cazuri poate fi utilă **analiza empirică**.

Analiza empirică poate fi utilizată pentru:

- Formularea unei ipoteze inițiale privind eficiența algoritmului
- Compararea eficienței mai multor algoritmi destinați rezolvării aceleiași probleme
- Analiza eficienței unei implementări a algoritmului (pe o anumită mașină)
- Verificarea acurateții unei afirmații privind eficiența algoritmului

# Structura generală a analizei empirice

1. Se stabilește **scopul analizei**
2. Se alege o **măsură a eficienței** (de exemplu, numărul de execuții ale unor operații sau timpul necesar execuției unor pași de prelucrare)
3. Se stabilesc **caracteristicile setului de date de intrare** ce va fi utilizat (dimensiune, domeniu de valori ...)
4. Se implementează algoritmul sau în cazul în care algoritmul este deja implementat **se adaugă instrucțiunile necesare efectuării analizei** (contoare, funcții de înregistrare a timpului necesar execuției etc)
5. Se **generează datele de intrare**
6. Se **execută programul** pentru fiecare dată de intrare și se înregistrează rezultatele
7. Se **analizează rezultatele obținute**

# Structura generală a analizei empirice

**Măsura eficienței:** este aleasă în funcție de scopul analizei

- Dacă scopul este să se identifice clasa de eficiență atunci se poate folosi numărul de operații care se execută
- Dacă scopul este să se analizeze/compare implementarea unui algoritm pe o anumită mașină de calcul atunci o măsură adecvată ar fi timpul fizic

# Structura generală a analizei empirice

**Set de date de intrare.** Trebuie generate diferite categorii de date de intrare pentru a surprinde diferitele cazuri de funcționare ale algoritmului

Câteva reguli de generare a datelor de intrare:

- Datele de intrare trebuie să fie de diferite dimensiuni și cu valori cât mai variate
- Setul de test trebuie să conțină date cât mai arbitrare (nu doar excepții)

# Structura generală a analizei empirice

**Implementarea algoritmului.** De regulă este necesară introducerea unor prelucrări de monitorizare

- **Variable contor** (în cazul în care eficiența este estimată folosind numărul de execuții ale unor operații)
- **Apelul unor funcții specifice** care returnează ora curentă (în cazul în care măsura eficienței este timpul fizic)

**Exemplu simplu** în Python

(o variantă mai bună ar fi utilizarea unui **profiler** – vezi **cprofile**):

```
import time
timpInitial = time.time()
< ... Prelucrari... >
timpFinal = time.time()
print(" Durata (sec):" , (timpFinal-timpInitial))
```

# Următorul curs va fi despre...

... algoritmi de sortare

... analiza corectitudinii lor

... analiza eficienței



# Intrebare de final

P:  $x[1..n]$  tablou de valori reale

Q:  $rez = \max\{x[i] - x[j]; 1 \leq i \leq n, 1 \leq j \leq n\}$

Alg( $x[1..n]$ )

max  $\leftarrow$   $x[1]$ ; min  $\leftarrow$   $x[1]$ ;

for  $i \leftarrow 2, n$  do

  if  $max < x[i]$  then max  $\leftarrow$   $x[i]$  endif

  if  $min > x[i]$  then min  $\leftarrow$   $x[i]$  endif

endfor

rez = max - min

return rez

Variante de raspuns:

- Algoritmul e incorect
- Algoritmul e corect si are ordinul de complexitate  $\Theta(1)$
- Algoritmul e corect si are ordinul de complexitate  $\Theta(n)$
- Algoritmul e corect si are ordinul de complexitate  $\Theta(n^2)$
- Nici unul dintre raspunsurile de mai sus nu e corect