

Analiza eficienței algoritmilor.

SD 2015/2016

Analiza eficienței algoritmilor

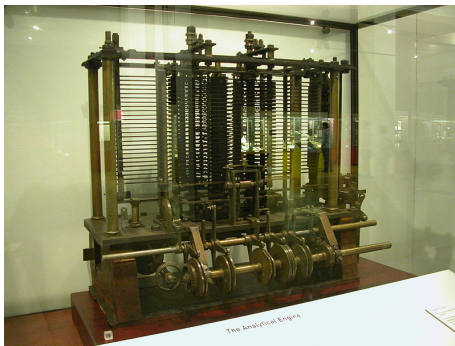
Exemple de calcul

Ordin de creștere

Notație asimptotică

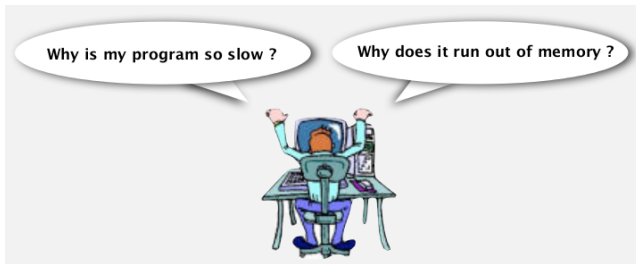
Timp de execuție

“As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time?”
Charles Babbage (1864)



Provocare

Pentru date de intrare de dimensiuni mari, programul rezolva problema?



Knuth (1970): Să utilizăm metode științifice pentru a înțelege performanța algoritmilor.

Analiza eficienței algoritmilor

- ▶ Analiza complexității.
- ▶ Estimarea volumului de **resurse de calcul** necesare execuției algoritmului:
 - ▶ **spațiu de memorie**: spațiul necesar stocării datelor;
 - ▶ **timp de execuție**: timpul necesar execuției algoritmului.
- ▶ **Algoritm eficient**: necesită un volum rezonabil de resurse de calcul:
 - ▶ eficiența se măsoară în raport cu spațiul de memorie, sau cu timpul de execuție;
- ▶ Utilitate:
 - ▶ pentru a stabili performanța algoritmului și a furniza garanții asupra acesteia;
 - ▶ pentru a compara algoritmi.

"It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one. We may count up the number of times that various elementary operations are applied in the whole process and then given them various weights. We might, for instance, count the number of additions, subtractions, multiplications, divisions, recording of numbers, and extractions of figures from tables. In the case of computing with matrices most of the work consists of multiplications and writing down numbers, and we shall therefore only attempt to count the number of multiplications and recordings."

— Alan Turing, *Bounding-off errors in matrix processes*, 1947

Trebuie să stabilim

I. Un model de calcul.

Modelul von Neumann – RAM (random access machine):

- ▶ prelucrările sunt executate secvențial;
- ▶ memoria constă dintr-o mulțime infinită de celule;
- ▶ timpul pentru accesarea datelor este același (nu depinde de locația acestora în memorie);
- ▶ celulele de memorie păstrează valori "mici" (limitate polinomial în dimensiune)
- ▶ timpul de execuție al unei prelucrări nu depinde de valorile operanzilor.

Modelul de calcul

- ▶ Implică o abstractizare, o simplificare brută.
- ▶ Memoria externă:
 - ▶ într-o mașină reală există o ierarhie complexă a memoriei;
 - ▶ există algoritmi special proiectați pentru seturi de date mari care trebuie păstrate în memoria externă;
 - ▶ memoria rapidă – de dimensiune limitată / memoria externă – nelimitată;
 - ▶ există operații speciale de intrare/ieșire care transferă informație între cele două tipuri.
- ▶ Procesare paralelă:
 - ▶ (*SIMD (Single Instruction, Multiple Data)*) - execuție paralelă a unei instrucțiuni, pe date diferite;
 - ▶ *multithreading* simultan - rularea mai multor fire de execuție pe un procesor;
 - ▶ procesoare multiple, procesoare *multicore*, etc.;
 - ▶ sisteme distribuite.

II. O unitate de măsură a timpului de execuție

- ▶ Pseudocod (curs 1):
 - ▶ variabile și tipuri elementare de date; instrucțiuni; proceduri și funcții.
- ▶ Timpul necesar execuției unei prelucrări elementare:
 - ▶ operații elementare: atribuire, operații aritmetice, comparații, operații logice;
 - ▶ fiecare operație elementară necesită o unitate de timp pentru a fi executată.
- ▶ Timpul total de execuție este egal cu numărul prelucrărilor elementare executate.

Dimensiunea problemei

- ▶ Ipoteză: volumul resurselor de calcul necesare depinde de volumul datelor de intrare.
- ▶ **Dimensiunea problemei:** volumul de memorie necesar stocării datele de intrare.
 - ▶ Este exprimată în:
 - ▶ numărul de componente ale datelor de intrare sau
 - ▶ numărul de biți necesari stocării datelor de intrare.
 - ▶ Numărul de biți necesari stocării valorii n este $\lceil \log_2 n \rceil + 1$.

Dimensiunea problemei - exemple

- ▶ Testul de primalitate pentru un număr n : n (sau $\log_2 n$).
- ▶ Minimul unui tablou: $x[0..n-1]$: n .
- ▶ Suma a două matrici ($m \times n$): $m \times n$.

Analiza eficienței algoritmilor

Exemple de calcul

Ordin de creștere

Notație asimptotică

Exemplul 1. Suma primelor n numere întregi

Intrare: $n \geq 1$

Ieșire: suma $s = 1 + 2 + \dots + n$

Dimensiunea problemei: n

Exemplul 1. Suma primelor n numere întregi

Intrare: $n \geq 1$

Ieșire: suma $s = 1 + 2 + \dots + n$

Dimensiunea problemei: n

Function $\text{suma}(n)$

begin

1 $s \leftarrow 0$

2 $i \leftarrow 1$

3 **while** $i \leq n$ **do**

4 $s \leftarrow s + i$

5 $i \leftarrow i + 1$

6 **return** s

end

Operație	Cost	Nr. repetări
1	c_1	1
2	c_2	1
3	c_3	$n+1$
4	c_4	n
5	c_5	n

$$\begin{aligned} T(n) &= (c_3 + c_4 + c_5)n + (c_1 + c_2 + c_3) \\ &= a * n + b \end{aligned}$$

Exemplul 1. Suma primelor n numere întregi

Considerăm că toate prelucrările elementare au același **cost unitar**.

- ▶ $T(n) = 3(n + 1)$;
- ▶ Constantele ce intervin în expresie nu sunt importante.
- ▶ Timpul de execuție depinde **liniar** de dimensiunea problemei.
- ▶ Algoritm echivalent:

$s \leftarrow 0$

for $i \leftarrow 1$ **to** n **do**

$s \leftarrow s + i$

- ▶ gestiunea contorului: $2(n + 1)$ operații;
- ▶ calculul sumei: $(n + 1)$ operații (inițializarea și actualizarea lui s).

Exemplul 2. Înmulțirea a două matrici

Intrare: $A(m \times n), B(n \times p)$

Ieșire: $C = A * B, \quad C_{i,j} = \sum_{k=1}^n A_{ik} B_{kj}, \quad i = 1, \dots, m, j = 1, \dots, p$

Dimensiunea problemei: mnp

Exemplul 2. Înmulțirea a două matrici

Intrare: $A(m \times n), B(n \times p)$

Ieșire: $C = A * B, \quad C_{i,j} = \sum_{k=1}^n A_{ik} B_{kj}, \quad i = 1, \dots, m, j = 1, \dots, p$

Dimensiunea problemei: mnp

Function *produs*($a[0..m-1, 0..n-1], b[0..n-1, 0..p-1]$)

begin

```
1   for  $i \leftarrow 0$  to  $m-1$  do  
2       for  $j \leftarrow 0$  to  $p-1$  do  
3            $c[i, j] \leftarrow 0$   
4           for  $k \leftarrow 0$  to  $n-1$  do  
5                $c[i, j] \leftarrow c[i, j] + a[i, k] * b[k, j]$   
6   return  $c[0..m-1, 0..p-1]$ 
```

end

Exemplul 2. Înmulțirea a două matrici

Operație	Cost	Nr. repetări
1	$2(m+1)$	1
2	$2(p+1)$	m
3	1	mp
4	$2(n+1)$	mp
5	2	mpn

$$T(m, n, p) = 4mnp + 5mp + 4m + 2$$

Exemplul 2. Înmulțirea a două matrici

Operație	Cost	Nr. repetări
1	$2(m+1)$	1
2	$2(p+1)$	m
3	1	mp
4	$2(n+1)$	mp
5	2	mpn

$$T(m, n, p) = 4mnp + 5mp + 4m + 2$$

Observație: nu este necesar să se completeze întreg tabelul; este suficient să se contorizeze **operația dominantă**.

- ▶ Cea mai frecventă (costisitoare) operație: $a[i, k] * b[k, j]$.
- ▶ Estimare timp de execuție: $T(m, n, p) = mnp$.

Exemplul 3. Minimul unui tablou

Intrare: $x[0..n-1]$, $n \geq 1$

Ieșire: $m = \min(x[0..n-1])$

Dimensiunea problemei: n

Exemplul 3. Minimul unui tablou

Intrare: $x[0..n-1]$, $n \geq 1$

leșire: $m = \min(x[0..n-1])$

Dimensiunea problemei: n

Function *minim*($x[0..n-1]$)
begin

```
1   $m \leftarrow x[0]$ 
2   $i \leftarrow 1$ 
3  while  $i < n$  do
4      if  $x[i] < m$  then
5           $m \leftarrow x[i]$ 
6       $i \leftarrow i + 1$ 
7  return  $m$ 
end
```

Operație	Cost	Nr. repetări
1	1	1
2	1	1
3	1	n
4	1	$n-1$
5	1	$t(n)$
6	1	$n-1$

$$T(n) = 3n + t(n)$$

Exemplul 3. Minimul unui tablou

Timpul de execuție depinde de:

- ▶ dimensiunea problemei;
- ▶ proprietățile datelor de intrare.

Trebuie analizate cazurile extreme:

- ▶ cazul cel mai favorabil
 - ▶ $x[0] \leq x[i], i = 0, \dots, n-1 \Rightarrow t(n) = 0 \Rightarrow T(n) = 3n$
- ▶ cazul cel mai nefavorabil
 - ▶ $x[0] > x[1] > \dots > x[n-1] \Rightarrow t(n) = n-1 \Rightarrow T(n) = 4n-1$
- ▶ $3n \leq T(n) \leq 4n-1$
Limita inferioară și limita superioară depind liniar de dimensiunea problemei.
- ▶ Dacă se ia în calcul doar operația de bază (comparația $x[i] < m$):
 $T(n) = n-1$

Exemplul 4. Căutarea secvențială

Intrare: $x[0..n-1]$, $n \geq 1$ și v o valoare (cheie de căutare)

Ieșire: valoarea de adevăr a afirmației “ v face parte din $x[0..n-1]$ ”

Dimensiunea problemei: n

Exemplul 4. Căutarea secvențială

Intrare: $x[0..n-1]$, $n \geq 1$ și v o valoare (cheie de căutare)

Ieșire: valoarea de adevăr a afirmației “ v face parte din $x[0..n-1]$ ”

Dimensiunea problemei: n

Function *cauta*($x[0..n-1], v$)

begin

```
1    $i \leftarrow 0$ 
2   while  $x[i] \neq v$  and  $i < n-1$ 
     do
3        $i \leftarrow i + 1$ 
4   if  $x[i] == v$  then
5        $gasit \leftarrow true$ 
     else
6        $gasit \leftarrow false$ 
     return  $gasit$ 
end
```

Operație	Cost	Nr. repetări
1	1	1
2	2	$t(n)+1$
3	1	$t(n)$
4	1	1
5	1	1
6	1	1

$$T(n) = 1 + 3t(n) + 4$$

Exemplul 4. Căutarea secvențială

Timpul de execuție depinde de:

- ▶ dimensiunea problemei;
- ▶ proprietățile datelor de intrare.

- ▶ Cazul cel mai favorabil
 - ▶ $x[0] = v \Rightarrow t(n) = 0 \Rightarrow T(n) = 5$

- ▶ cazul cel mai nefavorabil
 - ▶ $x[n-1] = v$ sau
($v! = x[0], \dots, v! = x[n-1]$) $\Rightarrow t(n) = n-1 \Rightarrow T(n) = 3n+2$

- ▶ Dacă se consideră ca operație dominantă comparația $x[i]! = v$:
 - ▶ cazul cel mai favorabil: $T(n) = 2$;
 - ▶ cazul cel mai nefavorabil: $T(n) = n+2$.

Exemplul 5. Sortarea prin insertie

Intrare: o secvență de numere (a_1, \dots, a_n)

Ieșire: o permutare $(a_{\sigma_1}, \dots, a_{\sigma_n})$ astfel încât $a_{\sigma_1} \leq a_{\sigma_2} \leq \dots \leq a_{\sigma_n}$

Dimensiunea problemei: n

Exemplul 5. Sortarea prin insertie

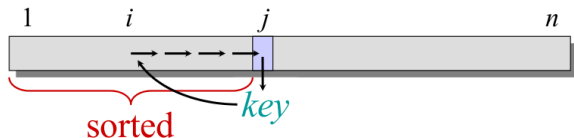
Intrare: o secvență de numere (a_1, \dots, a_n)

leșire: o permutare ($a_{\sigma_1}, \dots, a_{\sigma_n}$) astfel încât $a_{\sigma_1} \leq a_{\sigma_2} \leq \dots \leq a_{\sigma_n}$

Dimensiunea problemei: n

Procedure *insertion-sort*($a[0..n-1], n$)
begin

```
1   for  $j \leftarrow 1$  to  $n - 1$  do  
2        $key \leftarrow a[j]$   
3        $i \leftarrow j - 1$   
4       while  $i \geq 0$  and  $a[i] > key$  do  
5            $a[i + 1] \leftarrow a[i]$   
6            $i \leftarrow i - 1$   
7        $a[i + 1] \leftarrow key$   
end
```



Sortare prin inserție - exemplu

8 1 4 9 2 6

Sortare prin inserție - exemplu

8 1 4 9 2 6

8 1 4 9 2 6

Sortare prin inserție - exemplu

8 1 4 9 2 6

8	1	4	9	2	6
1	8	4	9	2	6

Sortare prin inserție - exemplu

8	1	4	9	2	6
8	1	4	9	2	6
1	8	4	9	2	6
1	4	8	9	2	6

Sortare prin inserție - exemplu

8	1	4	9	2	6
8	1	4	9	2	6
1	8	4	9	2	6
1	4	8	9	2	6
1	4	8	9	2	6

Sortare prin inserție - exemplu

8	1	4	9	2	6
8	1	4	9	2	6
1	8	4	9	2	6
1	4	8	9	2	6
1	4	8	9	2	6
1	2	4	8	9	6

Sortare prin inserție - exemplu

8	1	4	9	2	6
8	1	4	9	2	6
1	8	4	9	2	6
1	4	8	9	2	6
1	4	8	9	2	6
1	2	4	8	9	6
1	2	4	6	8	9
1	2	4	6	8	9

Exemplul 5. Sortarea prin inserție

Operație	Cost	Nr. repetări
1	c_1	n
2	c_2	$n - 1$
3	c_3	$n - 1$
4	c_4	$\sum_{j=2}^n t_j$
5	c_5	$\sum_{j=2}^n (t_j - 1)$
6	c_6	$\sum_{j=2}^n (t_j - 1)$
7	c_7	$n - 1$

$$T(n) = c_1 n + c_2(n - 1) + c_3(n - 1) + c_4 \sum_{j=2}^n t_j + c_5 \sum_{j=2}^n (t_j - 1) + c_6 \sum_{j=2}^n (t_j - 1) + c_7(n - 1)$$

Exemplul 5. Sortarea prin inserție

- ▶ Timpul de execuție depinde de:
 - ▶ dimensiunea problemei;
 - ▶ proprietățile datelor de intrare.

Exemplul 5. Sortarea prin inserție

- ▶ Timpul de execuție depinde de:
 - ▶ dimensiunea problemei;
 - ▶ proprietățile datelor de intrare.
- ▶ Cazul cel mai favorabil: tabloul este sortat.

$$t_j = 1, \quad j = 2, \dots, n$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

Exemplul 5. Sortarea prin inserție

- ▶ Timpul de execuție depinde de:
 - ▶ dimensiunea problemei;
 - ▶ proprietățile datelor de intrare.
- ▶ Cazul cel mai favorabil: tabloul este sortat.

$$t_j = 1, \quad j = 2, \dots, n$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

- ▶ Cazul cel mai nefavorabil: tabloul este sortat în ordine inversă.

$$t_j = j, \quad j = 2, \dots, n$$

$$\begin{aligned} T(n) &= c_1 n + (n-1)(c_2 + c_3 + c_7) + c_4 \left(\frac{n(n+1)}{2} - 1 \right) + c_5 \frac{n(n-1)}{2} + c_6 \frac{n(n-1)}{2} \\ &= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} \right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7 \right) n - (c_2 + c_3 + c_4 + c_7) \end{aligned}$$

Exemplul 5. Sortarea prin inserție

- ▶ Timpul de execuție depinde de:
 - ▶ dimensiunea problemei;
 - ▶ proprietățile datelor de intrare.
- ▶ Cazul cel mai favorabil: tabloul este sortat.

$$t_j = 1, \quad j = 2, \dots, n$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

- ▶ Cazul cel mai nefavorabil: tabloul este sortat în ordine inversă.

$$t_j = j, \quad j = 2, \dots, n$$

$$\begin{aligned} T(n) &= c_1 n + (n-1)(c_2 + c_3 + c_7) + c_4 \left(\frac{n(n+1)}{2} - 1 \right) + c_5 \frac{n(n-1)}{2} + c_6 \frac{n(n-1)}{2} \\ &= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2} \right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7 \right) n - (c_2 + c_3 + c_4 + c_7) \end{aligned}$$

- ▶ Cazul mediu: toate permutările au aceeași probabilitate de apariție.
- ▶ Este sortarea prin inserție un algoritm rapid?

► Cazul cel mai favorabil

- O margine inferioară a timpului de execuție;
- Identificarea algoritmilor ineficienți:
 - dacă un algoritm are un cost ridicat în cazul cel mai favorabil, atunci el nu reprezintă o soluție acceptabilă.

► Cazul cel mai nefavorabil

- Cel mai mare timp de execuție în raport cu toate datele de intrare posibile;
- O margine superioară a timpului de execuție;
- Marginea superioară este mai importantă decât cea inferioară.

Analiza în cazul mediu

- ▶ Există situații când cazul cel mai favorabil și cel mai nefavorabil sunt cazuri rare (excepții):
 - ▶ analiza în cele două cazuri nu furnizează suficientă informație.
- ▶ **Analiza în cazul mediu** furnizează informații privind comportarea algoritmului în cazul unor date de intrare arbitrare.
 - ▶ Se bazează pe cunoașterea **distribuției de probabilitate** a datelor de intrare.
 - ▶ Cunoașterea (estimarea) probabilității de apariție a fiecăreia dintre instanțele posibile ale datelor de intrare.
 - ▶ **Timpul mediu de execuție** este media timpilor de execuție corespunzători instanțelor datelor de intrare.

Analiza în cazul mediu

- ▶ Ipoteze privind distribuția de probabilitate a datelor de intrare:
 - ▶ datele de intrare pot fi grupate în clase (timpul de execuție este același pentru datele din aceeași clasă);
 - ▶ avem m clase cu date de intrare;
 - ▶ probabilitatea de apariție a unei date din clasa k este P_k ;
 - ▶ timpul de execuție pentru date din clasa k este $T_k(n)$.
- ▶ Timpul mediu de execuție este:

$$T_a(n) = P_1 T_1(n) + P_2 T_2(n) + \dots + P_m T_m(n)$$

- ▶ Dacă toate clasele au aceeași probabilitate de apariție:

$$T_a(n) = (T_1(n) + T_2(n) + \dots + T_m(n))/m$$

Exemplul 4. Căutarea secvențială (revizitat)

► Ipoteze:

- probabilitatea ca v să se afle în tablou: p
 - v apare cu aceeași probabilitate pe fiecare poziție din tablou;
 - probabilitatea ca v să se afle pe poziția k : p/n ;
- probabilitatea ca v să nu se afle în tablou: $1 - p$.

$$\text{► } T_a(n) = \frac{p(1+2+\dots+n)}{n} + (1-p)n = \frac{p(n+1)}{2} + (1-p)n = \left(1 - \frac{p}{2}\right)n + \frac{p}{2}$$

- dacă $p = 0.5$, atunci $T_a(n) = \frac{3}{4}n + \frac{1}{4}$;
- timpul mediu depinde liniar de dimensiunea datelor de intrare.

- Observație: timpul mediu nu este în mod necesar media aritmetică a timpilor de execuție corespunzători cazurilor extreme.

Etapele analizei algoritmilor

1. Identificarea dimensiunii problemei.
2. Identificarea operației dominante.
3. Estimarea timpului de execuție (determinarea numărului de execuții ale operației dominante).
4. dacă timpul de execuție depinde de proprietățile datelor de intrare, atunci se analizează:
 - ▶ cazul cel mai favorabil;
 - ▶ cazul cel mai nefavorabil;
 - ▶ cazul mediu.
5. Se stabilește ordinul (clasa) de complexitate.

Analiza eficienței algoritmilor

- ▶ Scopul principal: determinarea modului în care timpul de execuție crește odată cu creșterea dimensiunii problemei.
- ▶ Nu e necesar să se cunoască expresia detaliată a timpului de execuție
- ▶ Este suficient să se identifice:
 - ▶ **ordinul de creștere** al timpului de execuție;
 - ▶ **clasa de eficiență (complexitate)** căreia îi aparține algoritmul.

Analiza eficienței algoritmilor

Exemple de calcul

Ordin de creștere

Notăție asimptotică

Ordin de creștere

- ▶ **Termen dominant:** termen care devine semnificativ mai mare decât ceilalți atunci când dimensiunea problemei crește.
 - ▶ Diktează comportarea algoritmului când dimensiunea problemei crește.

Timp de execuție	Termen dominant
$T1(n) = an + b$	an
$T2(n) = a \log n + b$	$a \log n$
$T3(n) = an^2 + bn + c$	an^2
$T4(n) = a^n + bn + c$	a^n
$(a > 1)$	

- ▶ **Ordin de creștere:** caracterizează creșterea termenului dominant al timpului de execuție în raport cu dimensiunea problemei.

Ordin de creștere

- ▶ **Ordin de creștere:** caracterizează creșterea termenului dominant al timpului de execuție în raport cu dimensiunea problemei.
- ▶ Ce se întâmplă cu termenul dominant când dimensiunea problemei crește de k ori?

$T_1(n) = an$	$T'_1(kn) = akn$	$= kT_1(n)$	liniar
$T_2(n) = a \log n$	$T'_2(kn) = a \log(kn)$	$= T_2(n) + a \log k$	logaritm
$T_3(n) = an^2$	$T'_3(kn) = a(kn)^2$	$= k^2 T_3(n)$	pătratic
$T_4(n) = a^n$	$T'_4(kn) = a^{kn} = (a^n)^k$	$= T_4(n)^k$	exponențial

O comparație a ordinelor de creștere

Dependența timpilor de execuție a diferiți algoritmi în raport cu dimensiunea problemei (considerăm un procesor ce realizează 10^6 instrucțiuni pe secundă); dacă timpul de execuție depășește 10^{25} ani, afișăm “nu”.

n	$\log_2 n$	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
30	< 1 sec	< 1 sec	< 1 sec	< 1sec	18 min	nu
50	< 1 sec	< 1 sec	< 1 sec	< 1sec	36 ani	nu
10^2	< 1 sec	< 1 sec	< 1 sec	1 sec	10^{17} ani	nu
10^3	< 1 sec	< 1 sec	1 sec	18 min	nu	nu
10^4	< 1 sec	< 1 sec	2 min	12 zile	nu	nu
10^5	< 1 sec	2 sec	3 ore	32 ani	nu	nu
10^6	< 1 sec	20 sec	12 zile	31710 ani	nu	nu

Pentru a compara ordinele de creștere a doi timpi de execuție $T1(n)$ și $T2(n)$, calculăm $\lim_{n \rightarrow \infty} \frac{T1(n)}{T2(n)}$

- ▶ dacă $\lim = 0$: $T1(n)$ are un ordin de creștere mai mic decât $T2(n)$;
- ▶ dacă $\lim = c, c > 0$ constantă: $T1(n)$ și $T2(n)$ au același ordin de creștere;
- ▶ dacă $\lim = \infty$: $T1(n)$ are un ordin de creștere mai mare decât $T2(n)$.

Analiza eficienței algoritmilor

Exemple de calcul

Ordin de creștere

Notăție asimptotică

- ▶ Analiza timpilor de execuție pentru valori **mici** ale dimensiunii problemei **nu** permite diferențierea între algoritmi eficienți și ineficienți.
- ▶ Diferențele dintre ordinele de creștere devin din ce în ce mai semnificative pe măsură ce dimensiunea problemei crește.
- ▶ **Analiza asimptotică**: studiul proprietăților timpului de execuție atunci când dimensiunea problemei tinde către infinit (probleme de dimensiune mare).
 - ▶ algoritmul poate fi încadrat în diferite clase identificate prin notații: O , Ω , Θ

Ordine de creștere asimptotică. Notăția O

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

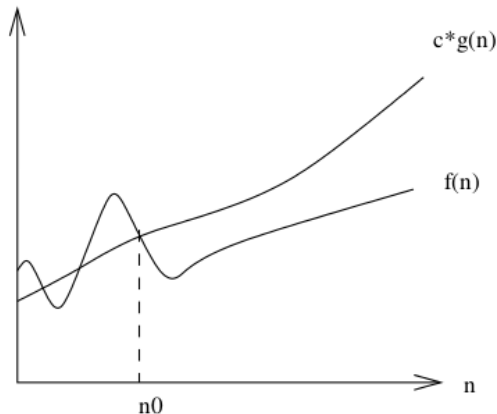
Definiție

$$O(g(n)) = \{f(n) : \exists c > 0, \exists n_0 \in \mathbb{N} \text{ a.î. } \forall n \geq n_0, 0 \leq f(n) \leq cg(n)\}.$$

Notăție: $f(n) = O(g(n))$

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

Notăția O



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

Exemple:

1. $T(n) = 3n + 3 \Rightarrow T(n) \in O(n)$
 $4n \geq 3n + 3, c = 4, n_0 = 3, g(n) = n$

Exemple:

1. $T(n) = 3n + 3 \Rightarrow T(n) \in O(n)$
 $4n \geq 3n + 3, c = 4, n_0 = 3, g(n) = n$
2. $3n^2 - 100n + 6 = O(n^2)$
 $3n^2 > 3n^2 - 100n + 6$

Exemple:

1. $T(n) = 3n + 3 \Rightarrow T(n) \in O(n)$
 $4n \geq 3n + 3, c = 4, n_0 = 3, g(n) = n$
2. $3n^2 - 100n + 6 = O(n^2)$
 $3n^2 > 3n^2 - 100n + 6$
3. $3n^2 - 100n + 6 = O(n^3)$
 $0.01n^3 > 3n^2 - 100n + 6$

Notăția O - proprietăți

1. $f(n) \in O(f(n))$ (reflexivitate).
2. $f(n) \in O(g(n)), g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$ (tranzitivitate).
3. Dacă $T(n) = a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n + a_0$ atunci $T(n) \in O(n^k)$ pentru orice $k \geq d$.
 - ▶ exemplu: $n \in O(n^2)$
4. Dacă pentru cazul cel mai nefavorabil obținem $T(n) \leq g(n)$, atunci $T(n) \in O(g(n))$.
 - ▶ Căutarea secvențială: $5 \leq T(n) \leq 3n + 2 \Rightarrow$ algoritmul este din clasa $O(n)$.

Definiție

$$\Omega(g(n)) = \{f(n) : \exists c > 0, n_0 \in \mathbb{N} \text{ a.î. } \forall n \geq n_0 : f(n) \geq cg(n)\}$$

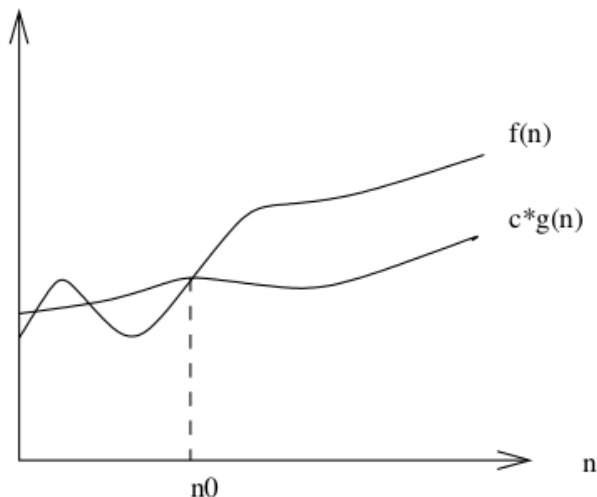
Notatie: $f(n) = \Omega(g(n))$

($f(n)$ are un ordin de creștere cel puțin la fel de mare ca cel al lui $g(n)$.)

Exemple:

1. $T(n) = 3n + 3 \Rightarrow T(n) \in \Omega(n)$
 $3n \leq 3n + 3, c = 3, n_0 = 1, g(n) = n$
2. $5 \leq T(n) \leq 3n + 2 \Rightarrow T(n) \in \Omega(1)$
 $c = 5, n_0 = 1, g(n) = 1$

Notăția Ω



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

Notăția Ω - proprietăți

1. $f(n) \in \Omega(f(n))$ (reflexivitate).
2. $f(n) \in \Omega(g(n)), g(n) \in \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n))$ (tranzitivitate).
3. Dacă $T(n) = a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n + a_0$ atunci $T(n) \in \Omega(n^k)$ pentru orice $k \leq d$.
 - ▶ exemplu: $n^2 \in \Omega(n)$

Definiție

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2 > 0, n_0 \in \mathbb{N} \text{ a.î. } \forall n \geq n_0 : c_1 g(n) \leq f(n) \leq c_2 g(n)\}.$$

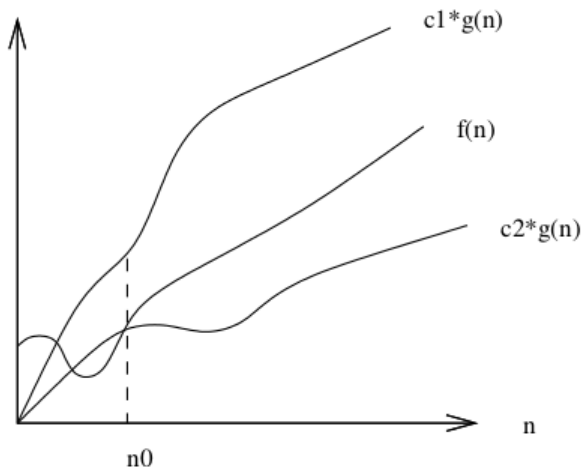
Notăție: $f(n) = \Theta(g(n))$

($f(n)$ are același ordin de creștere ca și $g(n)$.)

Exemple:

1. $T(n) = 3n + 3 \Rightarrow T(n) \in \Theta(n)$
 $c_1 = 2, c_2 = 4, n_0 = 3, g(n) = n$
2. Determinarea minimului unui tablou:
 $3n \leq T(n) \leq 4n - 1 \Rightarrow T(n) \in \Theta(n)$
 $c_1 = 3, c_2 = 4, n_0 = 1$

Notăția Θ



Pentru valori suficient de mari ale lui n , $f(n)$ este marginită, atât superior cât și inferior de $g(n)$ multiplicată cu niște constante pozitive.

Notăția Θ - proprietăți

1. $f(n) \in \Theta(f(n))$ (reflexivitate).
2. $f(n) \in \Theta(g(n)), g(n) \in \Theta(h(n)) \Rightarrow f(n) \in \Theta(h(n))$ (tranzitivitate).
3. $f(n) \in \Theta(g(n)) \Rightarrow g(n) \in \Theta(f(n))$ (simetrie).
4. Dacă $T(n) = a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n + a_0$ atunci $T(n) \in \Theta(n^d)$.
5. $\Theta(cg(n)) = \Theta(g(n))$ pentru orice constantă c .
Cazuri particulare:
 - ▶ $\Theta(c) = \Theta(1)$
 - ▶ $\Theta(\log_a h(n)) = \Theta(\log_b h(n))$ pentru orice $a, b > 1$
6. $\Theta(f(n) + g(n)) = \Theta(\max\{f(n), g(n)\})$

Notăția Θ - proprietăți

7. $\Theta(g(n)) \subset O(g(n))$.

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

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

Dar nu există constante c și n_0 astfel încât $cn^2 \leq 10n \lg n + 5$ pentru orice $n \geq n_0$.

8. $\Theta(g(n)) \subset \Omega(g(n))$.

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

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

Dar nu există constante c și n_0 astfel încât $10n \lg n + 5 \leq cn$ pentru orice $n \geq n_0$.

9. $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$.

Notăția Θ - exemple

1. Înmulțirea a două matrici: $T(m, n, p) = 4mnp + 5mp + 4m + 2$.

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

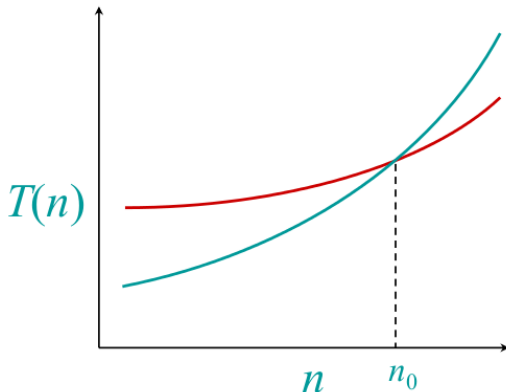
$f(m, n, p) \in \Theta(g(m, n, p))$ dacă există $c_1, c_2 > 0$ și $m_0, n_0, p_0 \in \mathbb{N}$ astfel încât $c_1 g(m, n, p) \leq f(m, n, p) \leq c_2 g(m, n, p)$ pentru orice $m \geq m_0, n \geq n_0, p \geq p_0$.

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

2. Căutare secvențială: $5 \leq T(n) \leq 3n + 2$.

Dacă $T(n) = 5$ atunci nu se poate găsi c_1 astfel încât $5 \geq c_1 n$ pentru valori suficient de mari ale lui $n \Rightarrow T(n)$ nu aparține lui $\Theta(n)$.

Notăția Θ - exemple



Când n este suficient de mare, un algoritm cu o complexitate $\Theta(n^2)$ este mai eficient decât unul cu complexitatea $\Theta(n^3)$.

Clasificarea algoritmilor folosind notația O

$$O(1) \subset O(\log n) \subset O(\log^k n) \subset O(n) \subset O(n^2) \subset \dots \subset O(n^{k+1}) \subset O(2^n)$$

$\{A \mid T_A(n) = O(1)\}$	=	clasa algoritmilor constanți;
$\{A \mid T_A(n) = O(\log n)\}$	=	clasa algoritmilor logaritmici;
$\{A \mid T_A(n) = O(\log^k n)\}$	=	clasa algoritmilor polilogaritmici;
$\{A \mid T_A(n) = O(n)\}$	=	clasa algoritmilor liniari;
$\{A \mid T_A(n) = O(n^2)\}$	=	clasa algoritmilor pătratici;
$\{A \mid T_A(n) = O(n^{k+1})\}$	=	clasa algoritmilor polinomiali;
$\{A \mid T_A(n) = O(2^n)\}$	=	clasa algoritmilor exponențiali.

$(k \geq 2)$