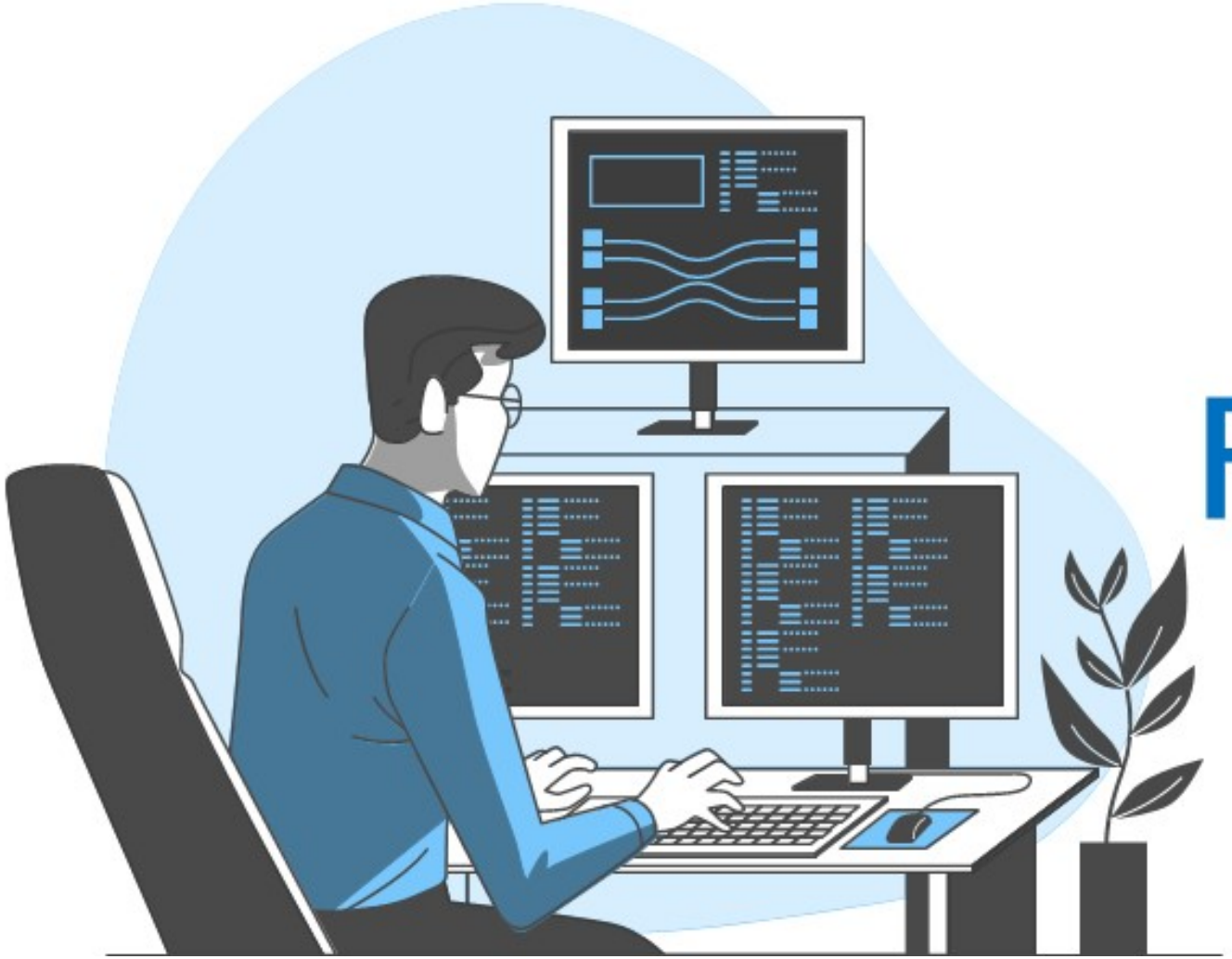
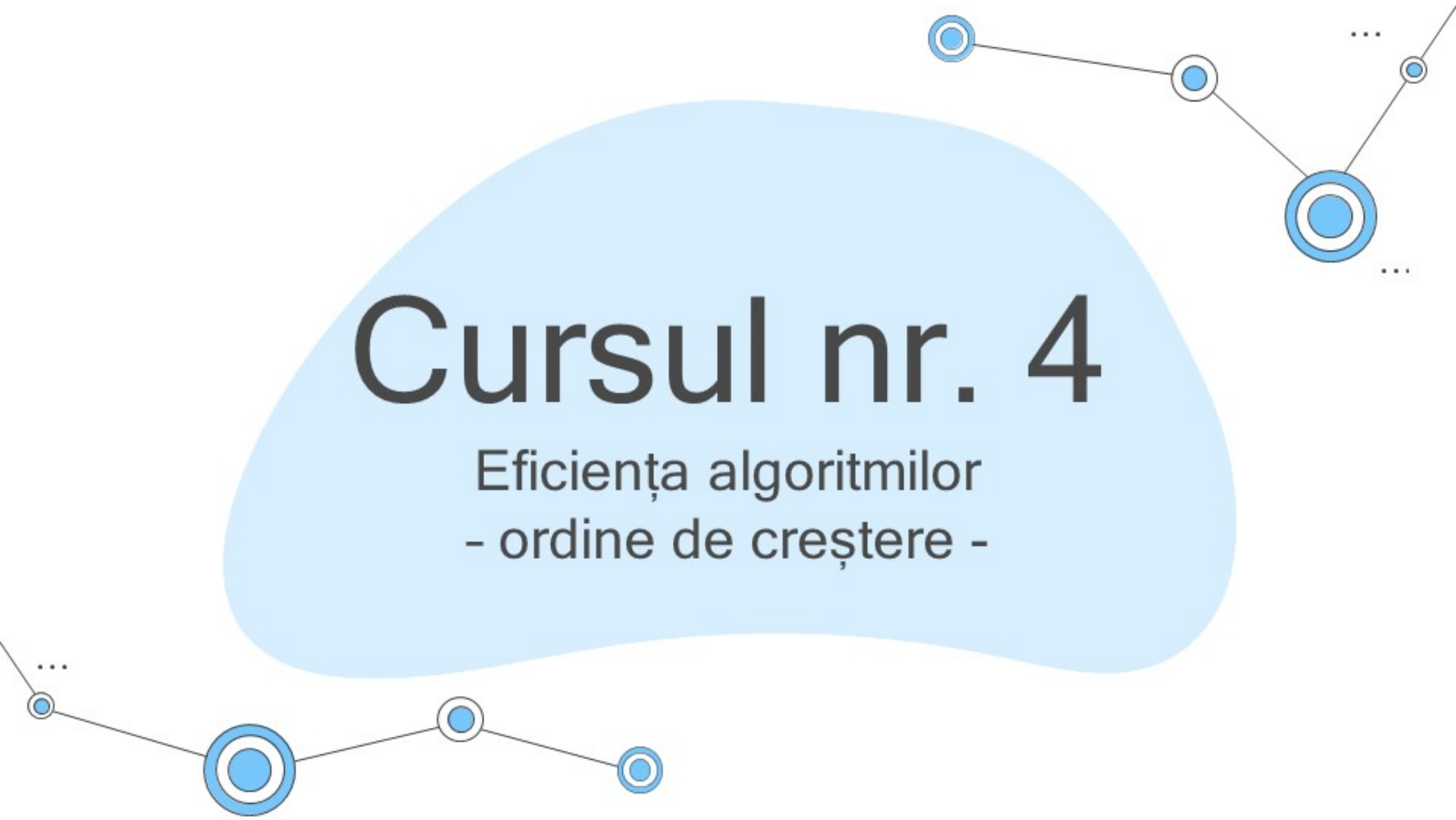


Algoritmi Fundamentali

Lector dr.
Dorin IORDACHE



Cursul nr. 4

Eficiența algoritmilor
- ordine de creștere -

ALGORITM

Analiza eficientei algoritmilor inseamna:

estimarea volumului de resurse de calcul necesare executiei algoritmilor

Observatie: uneori se foloseste termenul de analiza a complexitatii

Utilitate: analiza eficientei este utila pentru a compara algoritmii intre ei si pentru a obtine informatii privind resursele de calcul necesare pentru executia algoritmilor

Analiza eficientei

etapele principale ale analizei eficientei algoritmilor:

- Identificarea dimensiunii problemei
- Identificarea operatiei dominante
- Estimarea timpului de executie (determinarea numarului de executii ale operatiei dominante)
- Daca timpul de executie depinde de proprietatile datelor de intrare atunci se analizeaza:
  - Cel mai favorabil caz
    - => margine inferioara a timpului de executie
  - Cel mai defavorabil caz
    - => margine superioara a timpului de executie
  - Caz mediu
    - => timp mediu de executie

Rezumat

Tabel de costuri:

Operatie	Cost	Nr. repetari
1	c_1	n_1
2	c_2	n_2
3	c_3	n_3
....	....	.....
o	c_o	n_o

Timp de executie:

$$T(n) = \sum_{op=1}^{o} Cost_{op} * Nr.Repetariop$$

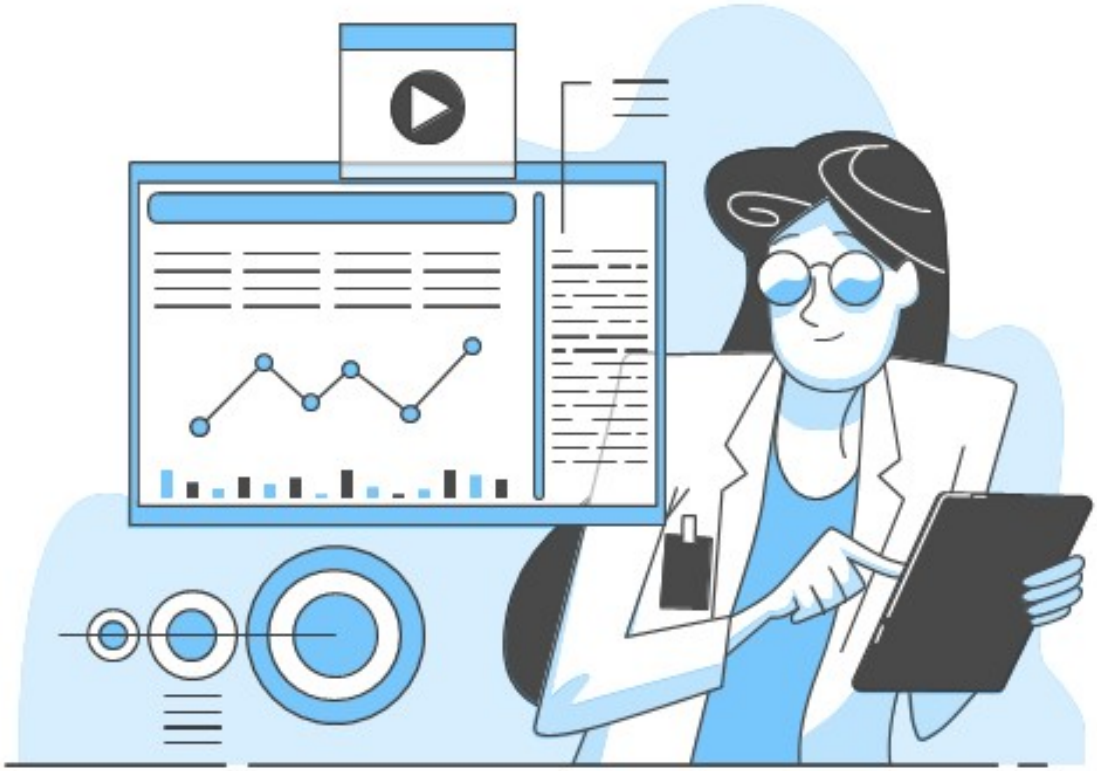
Agenda

01 Ordinul de creștere

02 Analiza asimptotică

03 Notația asimptotică

04 Clase de eficiență



01

Ordin de crestere

Ordin de crestere

In expresia timpului de executie exista de regula un termen care devine semnificativ mai mare decat ceilalti termeni atunci cand dimensiunea problemei creste.
Acest termen este denumit termen dominant si el dicteaza comportarea algoritmului in cazul in care dimensiunea problemei devine mare

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

Ordin de crestere

Sa analizam ce se intampla cu termenul dominant cand dimensiunea problemei creste de k ori :

$T1(n)=an+b$	$T'_1(kn)= a\ kn=k\ T_1(n)$
$T2(n)=a\ \log\ n+b$	$T'_2(kn)=a\ \log(kn)=T_2(n)+a\ \log\ k$
$T3(n)=a\ n^2+bn+c$	$T'_3(kn)=a\ (kn)^2=k^2\ T_3(n)$
$T4(n)=a^n+b\ n\ +c$ $(a>1)$	$T'_4(kn)=a^{kn}=(a^n)^k\ =T_4(n)^k$

Ordin de crestere

Ordinul de crestere exprima cum creste termenul dominant al timpului de executie in raport cu dimensiunea problemei

		Ordin de crestere
$T1(n)=an+b$	$T'_1(kn)= a\ kn=k\ T_1(n)$	Liniar
$T2(n)=a\ \log\ n+b$	$T'_2(kn)=a\ \log(kn)=T_2(n)+a\ \log\ k$	Logaritmic
$T3(n)=a\ n^2+bn+c$	$T'_3(kn)=a\ (kn)^2=k^2\ T_3(n)$	Patratic
$T4(n)=a^n+b\ n\ +c$ $(a>1)$	$T'_4(kn)=a^{kn}=(a^n)^k\ =T_4(n)^k$	Exponential

Ordin de crestere – interpretare

Cand se compara doi algoritmi, cel avand ordinul de crestere mai mic este considerat a fi mai eficient

Obs: comparatia se realizeaza pentru dimensiuni mari ale dimensiunii problemei (cazul asimptotic)

Exemplu. Consideram urmatoarele doua expresii ale timpului de executie
$T1(n) = 10n+10$ (ordin liniar de crestere)
$T2(n) = n^2$ (ordin patratic de crestere)

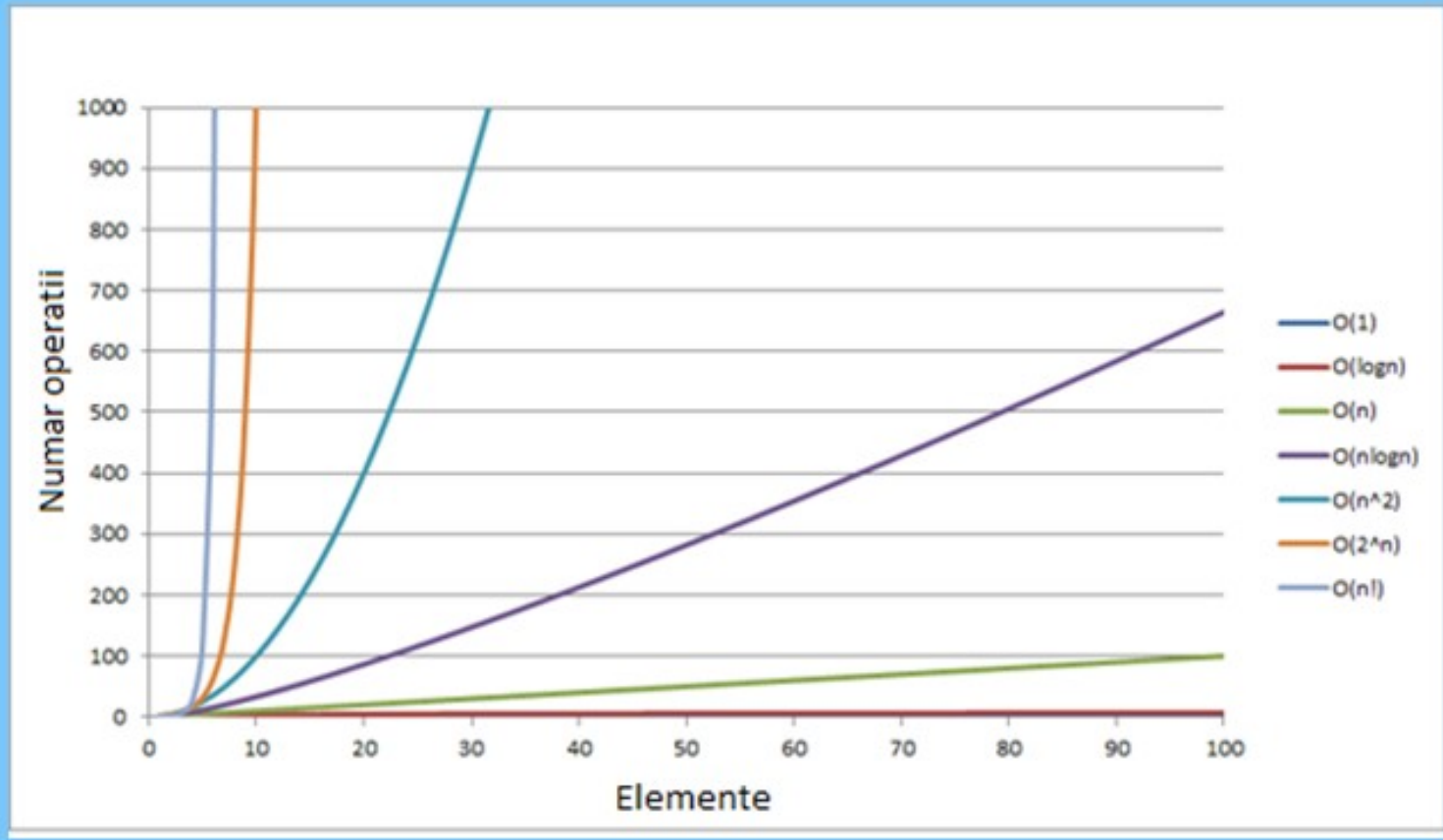
Daca $n<=10$ atunci $T1(n)>T2(n)$
In acest caz ordinul de crestere este relevant doar pentru $n>10$

Ordin de crestere

Obs: Constanta multiplicativa ce apare in cadrul termenului dominant poate fi ignorata

n	$\log_2 n$	$n\log_2 n$	n^2	2^n
10	3,3	33	100	1024
100	6,64	664	10000	12^{30}
1000	9,965	9965	1000000	10^{300}
10000	13,2877	132877	100000000	2^{3010}

Ordin de crestere



Ordin de crestere

Ordinele de crestere a doi timpi de executie T1(n) si T2(n) pot fi comparate prin calculul limitei raportului T1(n)/T2(n) cand n tinde la infinit

Daca limita este 0 atunci se poate spune ca T1(n) are un ordin de crestere mai mic decat T2(n)

Daca limita este o constanta finita strict pozitiva c (c>0) atunci se poate spune caT1(n) si T2(n) au acelasi ordin de crestere

Daca limita este infinita atunci se poate spune ca T1(n) are un ordin de crestere mai mare decat T2(n)

02

Analiza asimptotica

Analiza asimptotica

- Analiza timpilor de executie pentru valori mici ale dimensiunii problemei nu permite diferentierea dintre algoritmii eficienti si cei ineficienti
- Diferentele dintre ordinele de crestere devin din ce in ce mai semnificative pe masura ce creste dimensiunea problemei
- Analiza asimptotica are ca scop studiul proprietatilor timpului de executie atunci cand dimensiunea problemei tinde catre infinit (probleme de dimensiune mare)

Analiza asimptotica

In functie de proprietatile timpului de executie cand dimensiunea problemei devine mare, algoritmul poate fi incadrat in diferite clase identificate prin niste notatii standard

Notatiile standard utilizate in identificarea diferitelor clase de eficienta sunt:

- Θ (Theta)
- O (big O)
- Ω (Omega)

03

Notatii asimptotice

Notatii asimptotice

Θ (Theta)
O (big O)
Ω (Omega)

Notația Θ

Fie algoritmul de cautare secventiala de mai jos:

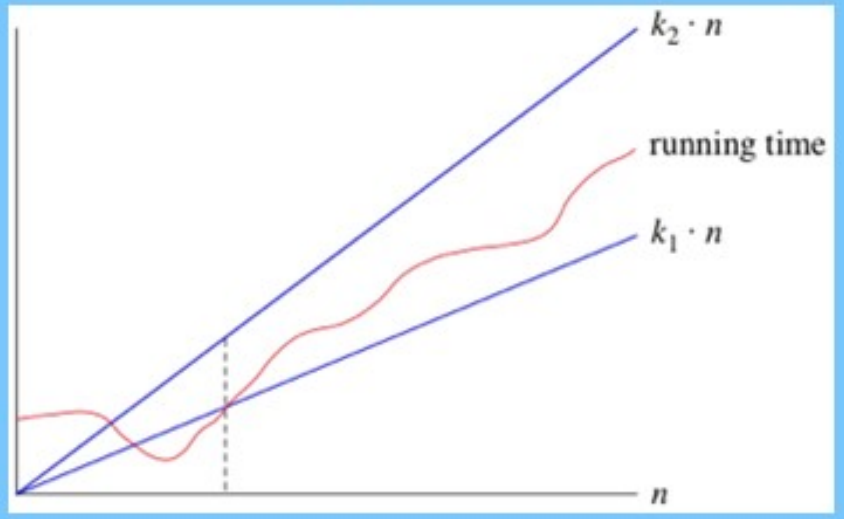
```
int CautareSecventiala(int v[], int n, int valoare){
for (int i = 0; i < n; i++)
  {
     if (v[i] == valoare)
        return 1;
  }
  return 0;
}
```

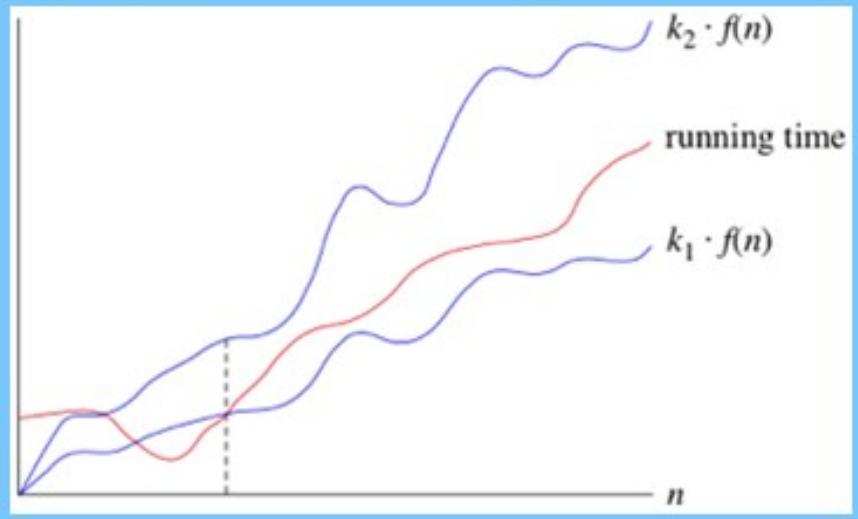
Dimeniunea maxima a vectorului este n
Numarul maxim de executii este n, aceasta se intampla in cazul cel mai nefavorabil (valoarea de cautat nu se regaseste in elemenetele vectorului)

Notația Θ

Definiție:

Fie f și g, 2 funcții definite pe mulțimea numerelor naturale. Funcția f se spune că este Θ(g), dacă există constantele k_1, $k_2 > 0$ și un număr natural n_0 astfel încât $k_1 * g(n) \leq f(n) \leq k_2 * g(n)$ pentru toate $n \geq n_0$



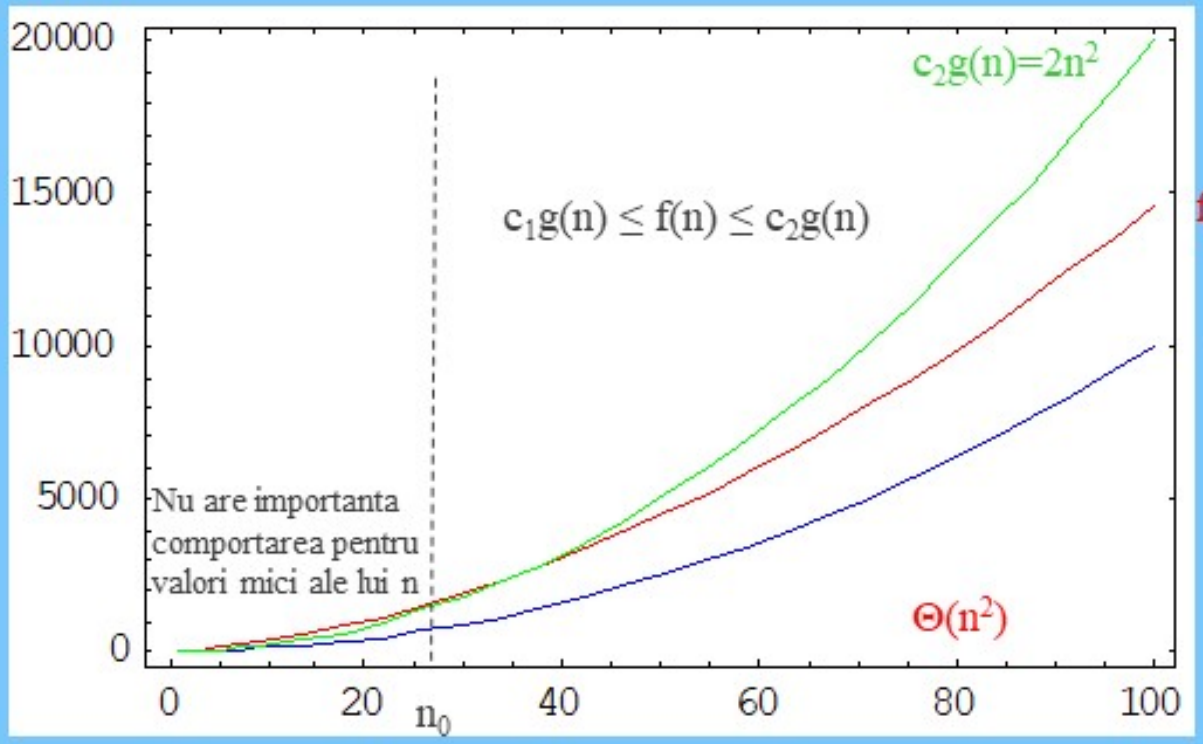


Notatie. Frecvent, in locul simbolului de apartenenta se foloseste cel de egalitate:

$f(n) = \Theta(g(n))$ (f(n) are acelasi ordin de crestere ca si g(n))

Notația Θ

Ilustrare grafică: Pentru valori mari ale lui n, f(n) este marginita, atat superior cat si inferior de g(n) inmultit cu niste constante pozitive



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

$c_1g(n)=n^2$

Proprietăți Θ

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

Dem. Intrucat $T(n)>0$ pentru orice n rezulta ca $a_k>0$.
Deci $T(n)/n^k \to a_k$ (cand $n \to 0$).

Deci pentru orice $\varepsilon>0$ exista $N(\varepsilon)$ astfel incat
$|T(n)/n^k - a_k| < \varepsilon \Rightarrow a_k - \varepsilon < T(n)/n^k < a_k + \varepsilon$ pentru orice $n>N(\varepsilon)$

Sa presupunem ca $a_k - \varepsilon > 0$.

Considerand $c_1=(a_k - \varepsilon)$, $c_2=a_k + \varepsilon$ si $n_0=N(\varepsilon)$ se obtine
$c_1 n^k < T(n) < c_2 n^k$ pentru orice $n>n_0$, adica $T(n) \in \Theta(nk)$

Proprietăți Θ

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

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

Atunci $c_1 cg(n) \leq f(n) \leq c_2 cg(n)$ pentru orice $n \geq n0$.
Considerand $c'_1 = cc_1$ si $c'_2 = c\ c_2$ se obtine ca $f(n) \in \Theta(g(n))$.
Astfel rezulta ca $\Theta(cg(n)) \in \Theta(g(n))$.
In mod similar se poate demonstra ca $\Theta(g(n)) \in \Theta(cg(n))$, adica $\Theta(cg(n)) = \Theta(g(n))$.

Cazuri particulare:

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

Obs. Baza logaritmilor nu este relevanta, astfel ca se va considera in majoritatea cazurilor ca se lucreaza cu baza 2.

Proprietăți Θ

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

4. $f(n) = \Theta(g(n))$ daca si numai daca $g(n) = \Theta(f(n))$ simetrie

5. $f(n) = \Theta(g(n))$ și $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$ tranzitivitate

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

Exemple Θ

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

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

Extinderea definitiei (in cazul in care dimensiunea problemei depinde de mai multe valori):

$f(m,n,p) \in \Theta(g(m,n,p))$ daca exista
$c_1, c_2 > 0$ si $m_0, n_0, p_0 \in N$ astfel incat
$c_1 g(m,n,p) <= f(m,n,p) <= c_2 g(m,n,p)$ pentru orice $m >= m_0,\ n >= n_0,\ p >= p_0$

Astfel $T(m,n,p) \in \Theta(mnp) \in \Theta(n^3)$ (m,n,p sunt de același ordin)

3. Cautare secventiala: $6 <= T(n) <= 3(n+1)$ (sau $4 <= T(n) <= 2n+2$)

Daca $T(n)=6$ atunci nu se poate gasi c_1 astfel incat $6 >= c_1 n$ pentru valori suficient de mari ale lui n. Rezulta ca T(n) nu apartine lui $\Theta(n)$.

Obs: Exista timpi de executie (algoritmi) care nu apartin unei clase de tip

Notația O

Definitie
$f(n) \in O(g(n))$ daca exista $c > 0$ si $n_0 \in N$ astfel incat
$f(n) <= c\, g(n)$ pentru orice $n >= n_0$

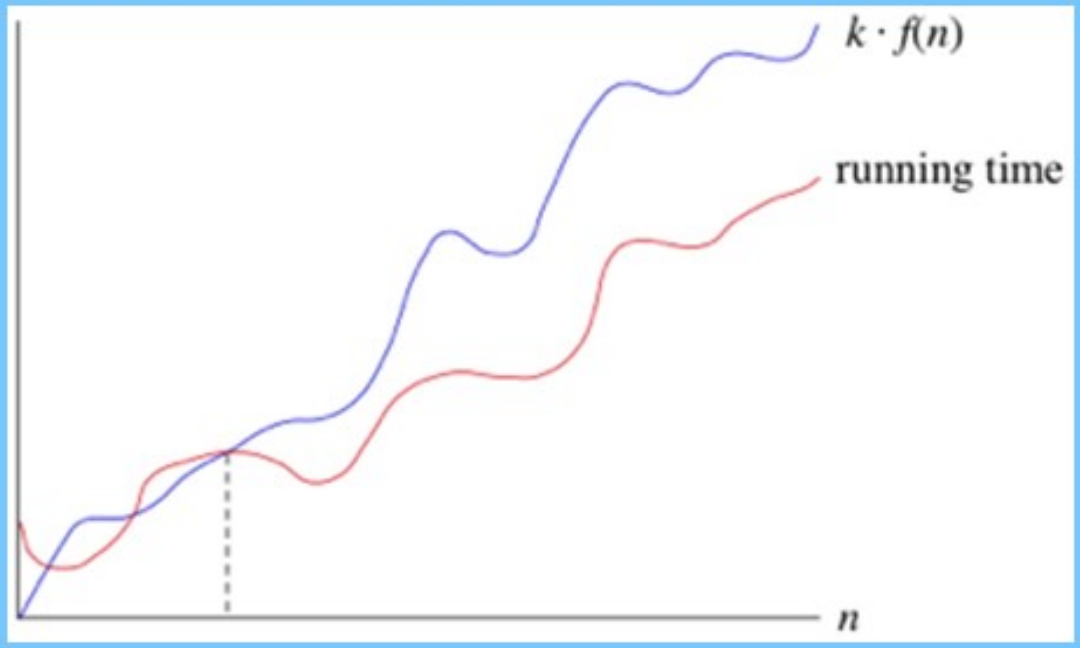
Notatie. $f(n) = O(g(n))$ ($f(n)$ are un ordin de crestere cel mult egal cu cel al lui $g(n)$)

Exemple.

1. $T(n) = 3n+3 \in T(n) \in O(n)$
   $c=4,\ n_0=3,\ g(n)=n$
2. $6 <= T(n) <= 3(n+1) \in T(n) \in O(n)$
   $c=4,\ n_0=3,\ g(n)=n$

Notația O

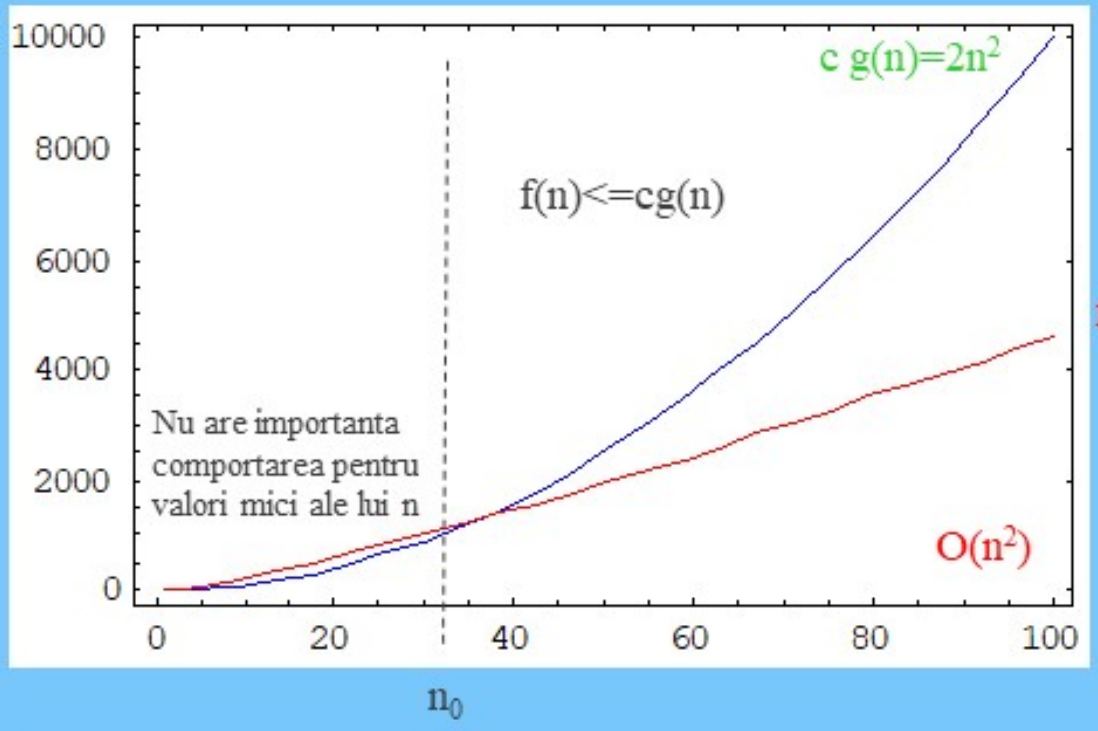
Dacă un timp de rulare este O(f(n)), atunci pentru n suficient de mare, timpul de rulare este de cel mult k·f(n), pentru o constantă k, ca in imaginea de mai jos.



Notația O oferă doar limita asimptotică superioară

Notația O

Ilustrare grafică: Pentru valori mari ale lui n, f(n) este marginita superior de g(n) multiplicata cu o constanta pozitiva



f(n)= 10 nlgn +5

Proprietăți O

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

Dem. Intrucat $T(n)>0$ pentru orice n, rezulta ca $a_k>0$.
Atunci $T(n)/n^k \rightarrow a_k$
Deci pentru orice $\varepsilon>0$ rezulta ca exista $N(\varepsilon)$ astfel incat
$T(n)/n^k <= a_k+\varepsilon$ pentru orice $n>N(\varepsilon)$
Prin urmare $T(n) <= (a_k+\varepsilon)n^k <= (a_k+\varepsilon)n^d$
Considerand $c=a_k+\varepsilon$ si $n_0=N(\varepsilon)$ rezulta ca
$T(n) <cn^d$ pentru $n>n_0$, i.e. $T(n) \in O(n^d)$

Exemplu.
$n \in O(n^2)$

Proprietăți O

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 inclusa in $O(g(n)$

Obs. Incluziunea de mai sus este stricta: exista elemente din $O(g(n))$ care nu apartin lui $\Theta(g(n))$

Exemplu:

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

$f(n)<=g(n)$ pentru orice $n>=36$ Θ $f(n) \in O(g(n))$

Dar nu exista constante c si n0 astfel incat:

$cn^2 <= 10n\lg n+5$ pentru orice $n >= n_0$

Proprietăți O

Daca prin analizarea celui mai defavorabil caz se obtine:

$T(n) <= g(n)$ atunci se poate spune despre T(n) ca apartine lui $O(g(n))$

Exemplu. Cautare secventiala:

$6 <= T(n) <= 3(n+1)$ (sau $4 <= T(n) < 2(n+1)$)

Deci, algoritmul cautarii secventiale este din clasa O(n)

$T(n) \in O(n)$

Notația Ω

Definitie

$f(n) \in \Omega(g(n))$ daca exista $c > 0$ si $n0 \in N$ astfel incat

$cg(n) <= f(n)$ pentru orice $n >= n_0$

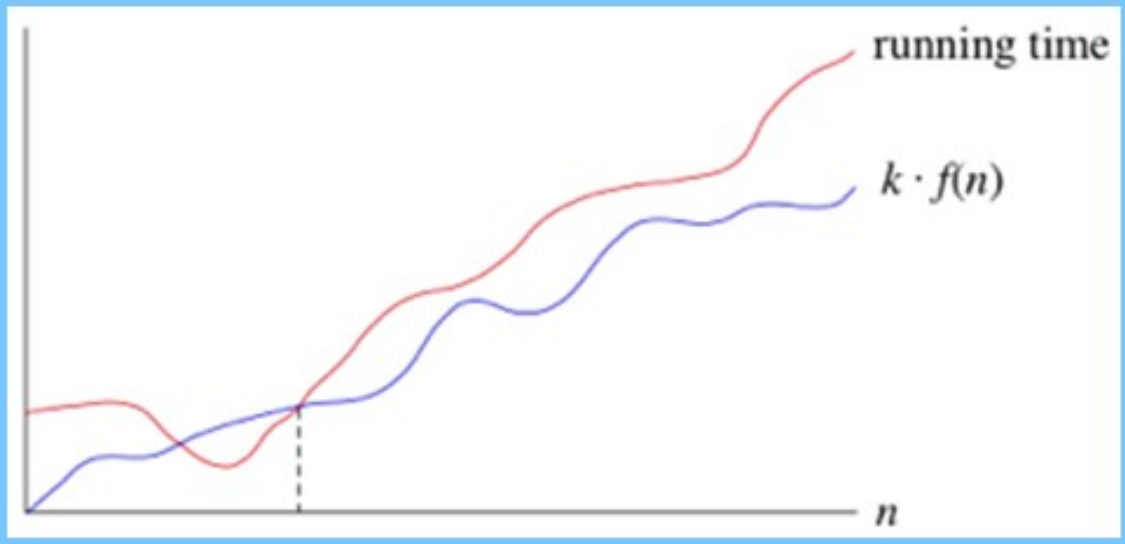
Notatie. $f(n) = \Omega(g(n))$ (ordinul de crestere al lui f(n) este cel putin la fel de mare ca cel al lui g(n))

Exemple.

1. $T(n) = 3n+3 \in T(n) \in \Omega(n)$
   $c=3,\ n0=1,\ g(n)=n$
2. $6 <= T(n) <= 3(n+1) \in T(n) \in \Omega(1)$
   $c=6,\ n0=1,\ g(n)=1$

Notația Ω

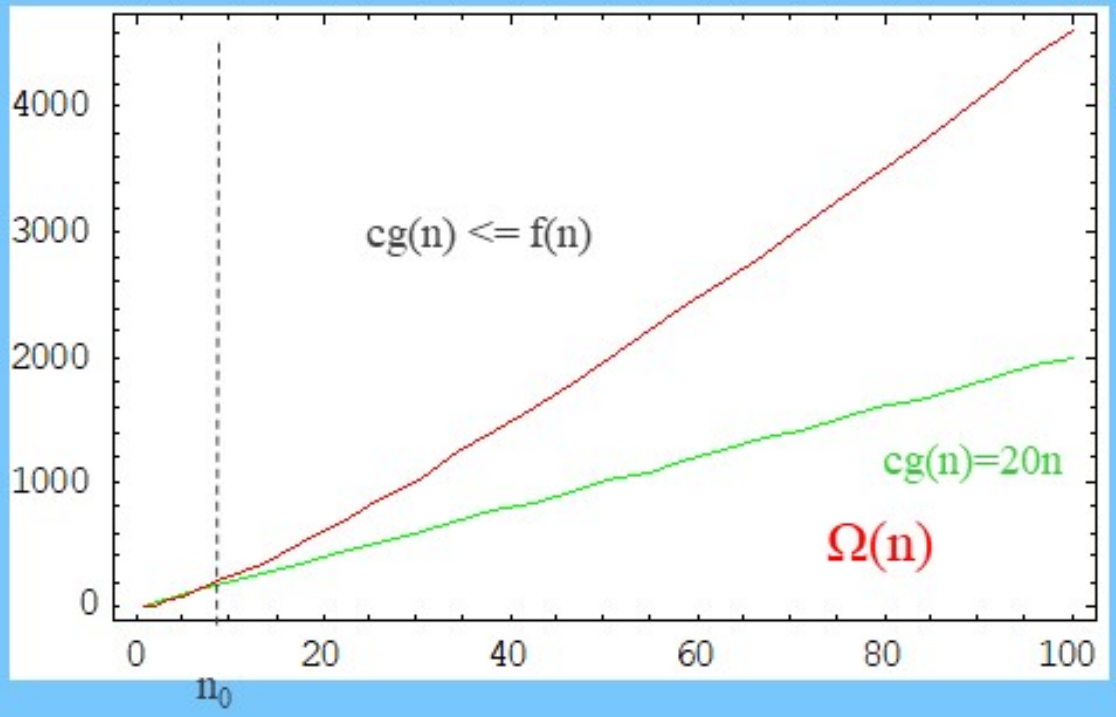
Notația Omega reprezintă limita inferioară a timpului de rulare a unui algoritm. Astfel, oferă cea mai bună complexitate de caz a unui algoritm



Dacă un timp de rulare este Ω(f(n), atunci pentru n suficient de mare, timpul de rulare este de cel puțin k·f(n) pentru o constantă k

Notația Ω

Ilustrare grafică: Pentru valori mari ale lui n, functia f(n) este marginita inferior de g(n) multiplicata eventual de o constanta pozitiva



f(n)= 10 nlgn +5

Proprietăți Ω

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

Dem. Intrucat $T(n)>0$ pentru orice n rezulta ca $a_k>0$.

Atunci $T(n)/n^k \rightarrow a_k$
Astfel pentru orice $\varepsilon>0$ exista $N(\varepsilon)$ astfel incat
$a_k - \varepsilon <= T(n)/n^k$ pentru orice $n>N(\varepsilon)$
Rezulta ca $(a_k-\varepsilon)n^d <=(a_k-\varepsilon)n^k <= T(n)$
Considerand $c=a_k-\varepsilon$ si $n_0=N(\varepsilon)$ se obtine
$cn^d <= T(n)$ pentru orice $n>n_0$, adica $T(n) \in \Omega(n^d)$

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

Analiza eficientei structurilor algoritmice

Structura secventiala

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	$\Theta(g_k(n))$	$O(g_k(n))$	$\Omega(g_k(n))$

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

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

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

Analiza eficientei structurilor algoritmice

Structura conditionala

P:

IF <condition> THEN

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

ELSE

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

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

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

Analiza eficientei structurilor algoritmice

Structura repetitiva

P:

FOR i←1, n DO

P1 O(1) ➔ O(n)

FOR i←1,n DO

FOR j ← 1,n DO

P1 O(1) ➔ $O(n^2)$

Obs: In cazul a k cicluri suprapuse a caror contor varianza intre 1 si n ordinul de complexitate este n^k

Analiza eficientei structurilor algoritmice

Obs.

Daca limitele contorului sunt variabile atunci numarul de operatii efectuate trebuie calculat explicit pentru fiecare dintre ciclurile suprapuse

Exemplu:

```
m ← 1
FOR i ← 1,n DO
        m ← 3*m              {m=3^i}
    FOR j ← 1,m DO
        prelucrare de cost O(1)
    ENDFOR
ENDFOR
```

Ordinul de complexitate al prelucrarii este:

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

adica $\Theta(3^n)$

04

Clase de eficienta

Clase de eficienta(complexitate)

Nume clasa	Notatie asimptotica	Exemplu
logaritmic	$O(\log n)$	Cautare binara
liniar	$O(n)$	Cautare secventiala
patratic	$O(n^2)$	Sortare prin insertie
cubic	$O(n^3)$	Inmultirea a doua matrice nxn
exponential	$O(2^n)$	Prelucrarea tuturor submultimilor unei multimi cu n elemente
factorial	$O(n!)$	Prelucrarea tuturor permutarilor de ordin n

Clase de eficienta(complexitate)

1. Se stabilește dimensiunea problemei.
2. Se identifică operația de bază (operația dominantă).
3. Se verifică dacă numărul de execuții ale operației de bază depinde doar de dimensiunea problemei.
   Da: se determină acest număr.
   Nu: se analizează cazul cel mai favorabil, cazul cel mai defavorabil și (dacă este posibil) cazul mediu.
4. Se stabilește clasa de complexitate căruia îi aparține algoritmul.

Analiza empirică

Uneori analiza teoretica a eficientei este dificila; in aceste cazuri poate fi utila analiza empirica.

Analiza empirica poate fi utilizata pentru:

- Formularea unei ipoteze initiale privind eficienta algoritmului
- Compararea eficientei a mai multor algoritmi destinati rezolvarii aceleiasi probleme
- Analiza eficientei unei implementari a algoritmului (pe o anumita masina)
- Verificarea acuratetii unei afirmatii privind eficienta algoritmului

Analiza empirică

1. Se stabileste scopul analizei
2. Se alege o masura a eficientei (de exemplu, numarul de executii ale unor operatii sau timpul necesar executiei unor pasi de prelucrare)
3. Se stabilesc caracteristicile setului de date de intrare ce va fi utilizat (dimensiune, domeniu de valori ...)
4. Se implementeaza algoritmul sau in cazul in care algoritmul este deja inmplementat se adauga instructiunile necesare efectuarii analizei (contoare, functii de inregistrare a timpului necesar executiei etc)
5. Se genereaza datele de intrare
6. Se executa programul pentru fiecare data de intrare si se inregistreaza rezultatele
7. Se analizeaza rezultatele obtinute

Mulțumesc

pentru atenție!

dorin.iordache@365.univ-ovidius.ro