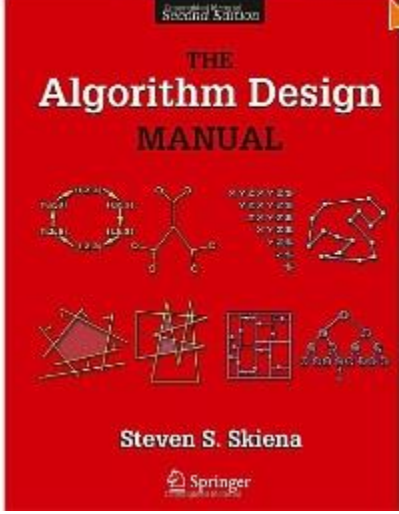
Curs 4:

Analiza eficienței algoritmilor (II)

Motivație

S. Skiena – The Algorithm Design Manual



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.

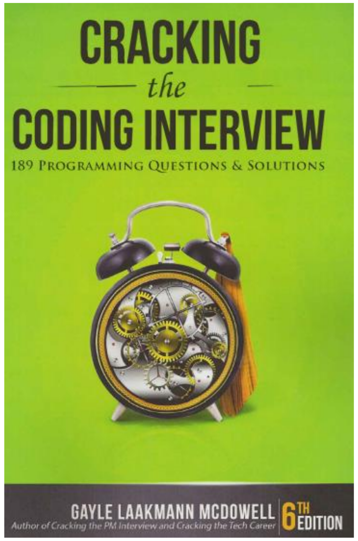
(a) Let S be an *unsorted* array of n integers. Give an algorithm that finds the pair $x, y \in S$ that *maximizes* $|x - y|$. Your algorithm must run in $O(n)$ worst-case time.

(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.

Motivație



Cracking the Coding Interview

Big O

This is such an important concept that we are dedicating an entire (long!) chapter to it.

Big O time is the language and metric we use to describe the efficiency of algorithms. Not understanding it thoroughly can really hurt you in developing an algorithm. Not only might you be judged harshly for not really understanding big O, but you will also struggle to judge when your algorithm is getting faster or slower.

Master this concept.

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 ?

- In 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 comportamentul 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: $a \log n$

$T_3(n)=a\ n^2+bn+c$ Termen dominant: $a\ n^2$

$T_4(n)=a^n+b\ n+c$ $(a>1)$ Termen dominant: a^n

Ce este ordinul de creștere ?

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

$T_1(n)=an$ $T_1(kn)= a\ kn=k\ T_1(n)$ (crește de atâtea ori de câte ori crește dimensiunea problemei – creștere liniară)

$T_2(n)=a \log n$ $T_2(kn)=a \log(kn)=T_2(n)+a \log k$ (se adaugă un termen constant în raport cu dimensiunea problemei)

$T_3(n)=a\ n^2$ $T_3(kn)=a\ (kn)^2=k^2\ T_3(n)$ (factorul de creștere este pătratic)

$T_4(n)=a^n$ $T_4(kn)=a^{kn}=(a^n)^k =T_4(n)^k$ (factorul de creștere intervine la exponent)

Ce este ordinul de creștere ?

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

	Ordin de creștere
$T'_1(kn)= a\ kn=k\ T'_1(n)$	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

$T_1(n)=10n+10$ (ordin liniar de creștere)

$T_2(n)=n^2$ (ordin pătratic de creștere)

Daca $n<=10$ 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 $3*10^{-9}$ sec	33 $3*10^{-8}$ sec	100 10^{-7} sec	1024 10^{-6} sec	3628800 0.003 sec
100 10^{-7} sec	6.6 $6*10^{-9}$ sec	664 $6*10^{-7}$ sec	10000 10^{-5} sec	10^{30} 10^{13} ani	10^{157} 10^{140} ani
1000 10^{-6} sec	10 10^{-8} sec	9965 $9*10^{-6}$ sec	1000000 0.001 sec	10^{301} 10^{284}ani	10^{2567} 10^{2550} ani
10000 10^{-5} sec	13 $1.3*10^{-8}$ 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 $T_1(n)$ și $T_2(n)$ pot fi comparate prin calculul limitei raportului $T_1(n)/T_2(n)$ când n tinde la infinit
- Dacă limita este 0 atunci se poate spune că $T_1(n)$ are un ordin de creștere mai mic decât $T_2(n)$
- Dacă limita este o constantă finită strict pozitivă c (c>0) atunci se poate spune că $T_1(n)$ și $T_2(n)$ au același ordin de creștere
- Dacă limita este infinită atunci se poate spune că $T_1(n)$ are un ordin de creștere mai mare decât $T_2(n)$

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 algoritmii 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:

  Θ (Theta)
  O (O)
  Ω (Omega)

Notația Θ

Fie f,g: N-> R_+ două funcții care depind de dimensiunea problemei și iau valori pozitive

Definiție.

f(n) ∈ Θ(g(n)) dacă există c_1, c_2 > 0 si n_0 ∈ 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)= Θ (g(n)) (f(n) are același ordin de creștere ca și g(n))

Exemple.

1. T(n) = 3n+3 =>T(n) ∈ Θ (n) (sau T(n) = Θ (n))

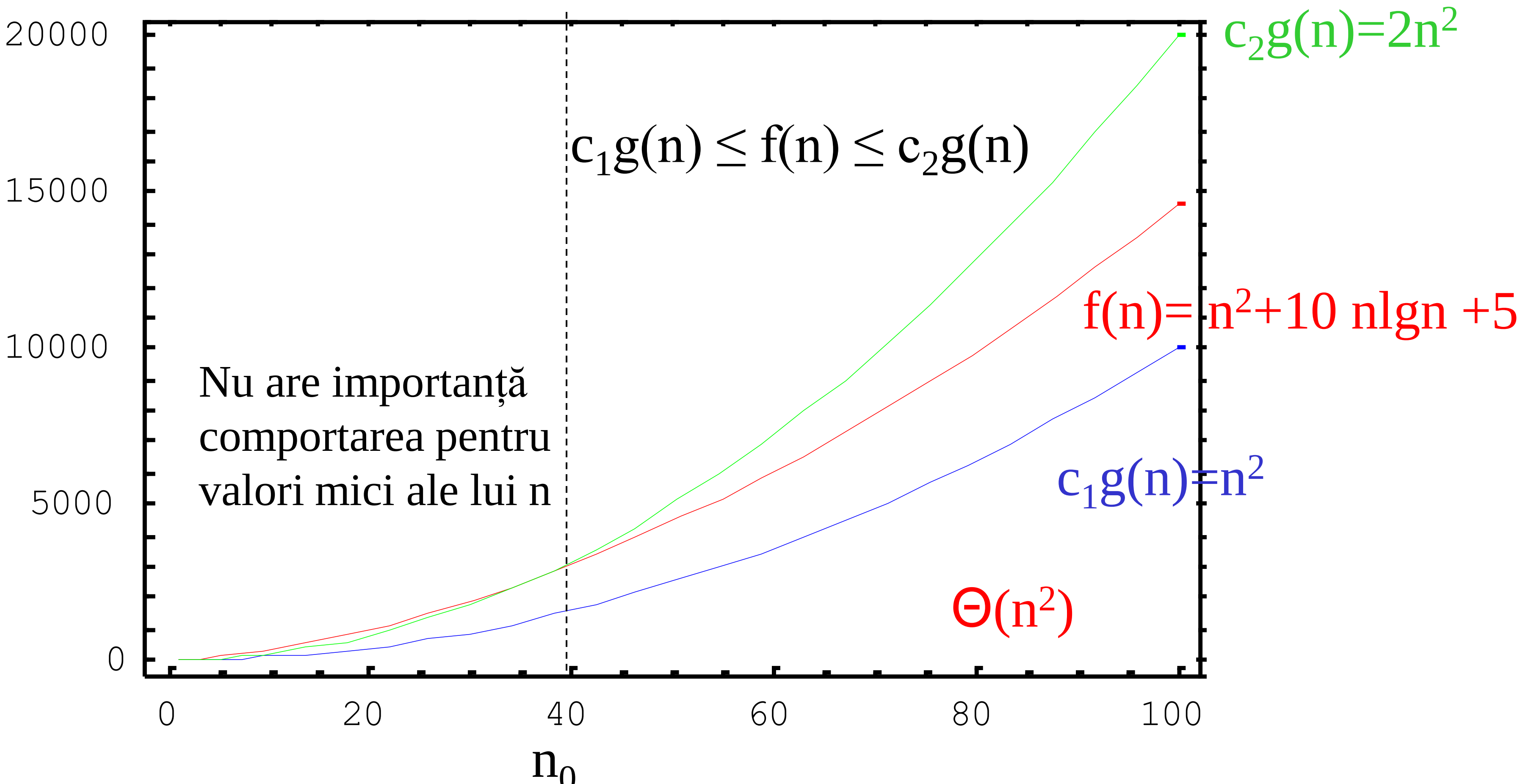
   $c_1=2$, $c_2=4$, $n_0=3$, $g(n)=n$

2. $T(n)= n^2+10 \text{ nlgn} +5$ => T(n) ∈ Θ (n^2)

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

Notația Θ

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



Notația Θ. Proprietăți

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

Dem. Intrucât $T(n)>0$ pentru orice n rezultă ca $a_k>0$.
Deci $T(n)/n^k \to a_k$ (cand n->□).

Deci pentru orice $\varepsilon>0$ (sufficient de mic) exista $N(\varepsilon)$ astfel încât
$|T(n)/n^k- a_k|< \varepsilon \Rightarrow a_k- \varepsilon<T(n)/n^k<a_k+ \varepsilon$ 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
$c_1 n^k < T(n) <c_2 n^k$ pentru orice $n>n_0$, adică $T(n) \in \Theta(n^k)$

Notatia Θ. Proprietăți

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

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

Atunci $c_1cg(n) \le f(n) \le c_2cg(n)$ pentru orice $n \ge n0$.
Considerând $c'_1 = cc_1$ si $c'_2 = c\, c_2$ se obține că $f(n) \in \Theta(g(n))$.
Astfel rezultă că $\Theta(cg(n)) \in \Theta(g(n))$.
In mod similar se poate demonstra că $\Theta(g(n)) \in \Theta(cg(n))$, 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. In stabilirea ordinului de complexitate, baza logaritmilor nu este relevantă, astfel că se va considera în majoritatea cazurilor că se lucrează cu baza 2

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

Notația Θ. Alte exemple

3. $3n <= T(n) <= 4n-1 => T(n) \in \Theta(n)$

   $c_1=3$, $c_2=4$, $n_0=1$

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

$c_1, c_2 > 0$ si $m_0, n_0, p_0 \in N$ astfel încât

$c_1 g(m,n,p) <= f(m,n,p) <= c_2 g(m,n,p)$ pentru orice $m >= m_0$, $n >= n_0$, $p >= p_0$

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

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

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

Obs: Există timpi de executie (algoritmi) care nu aparțin unei clase de tip

Notația O

Definiție.

$f(n) \in O(g(n))$ dacă există $c > 0$ și $n_0 \in N$ astfel încât

$$f(n) <= c\, g(n) \quad \text{pentru orice } n >= n_0$$

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

Exemple.

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

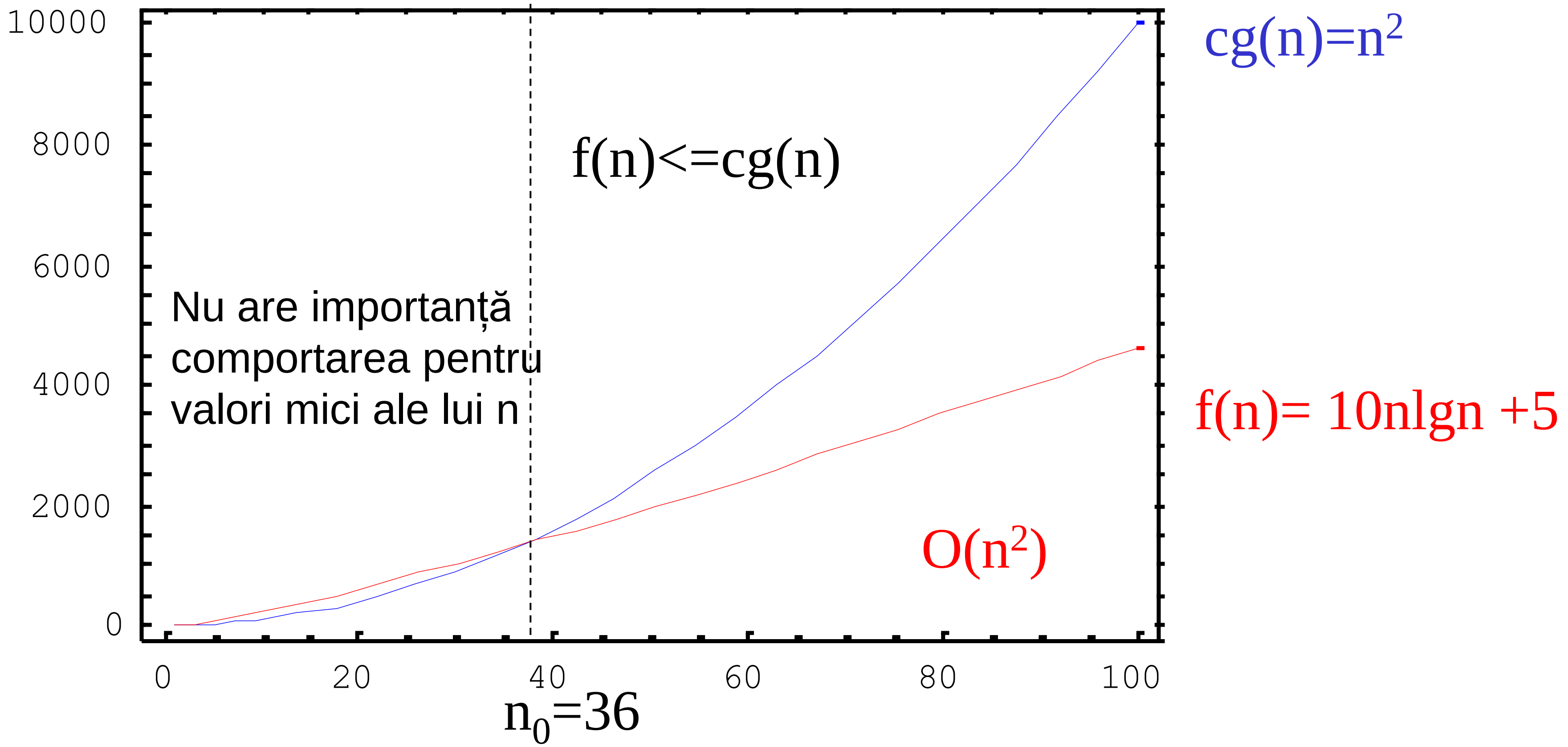
   $c=4$, $n_0=3$, $g(n)=n$

2. $6 <= T(n) <= 3(n+1) \Rightarrow T(n) \in O(n)$ (interesează doar marginea superioară a timpului de execuţie)

   $c=4$, $n_0=3$, $g(n)=n$

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

Dem. Intrucât $T(n)>0$ pentru orice n, rezultă că $a_k>0$.
Atunci $T(n)/n^k \to a_k$ (cand $n\to\infty$).
Deci pentru orice $\varepsilon>0$ rezultă că există $N(\varepsilon)$ astfel încât

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

Prin urmare $T(n) <= (a_k+ \varepsilon)n^k <= (a_k+\varepsilon)n^d$
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:

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

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

Dar nu există constante c si n_0 astfel incat:

$cn^2 <= 10n\lg n+5$ pentru orice $n >= n_0$ (deci $f(n)$ nu este şi din $\Theta(g(n))$)

Notatia O. Proprietăți

Dacă prin analizarea celui mai defavorabil caz se obține:

T(n) <= g(n) atunci se poate spune despre T(n) că aparține lui O(g(n))

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

Deci algoritmul căutarii secvențiale este din clasa O(n)

(sau are ordin liniar de complexitate)

Notația Ω

Definiție.

$f(n) \in \Omega(g(n))$ dacă există $c > 0$ și $n_0 \in N$ astfel încât

$$cg(n) <= f(n) \text{ pentru orice } n>=n_0$$

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. $T(n) = 3n+3 \Rightarrow T(n) \in \Omega(n)$

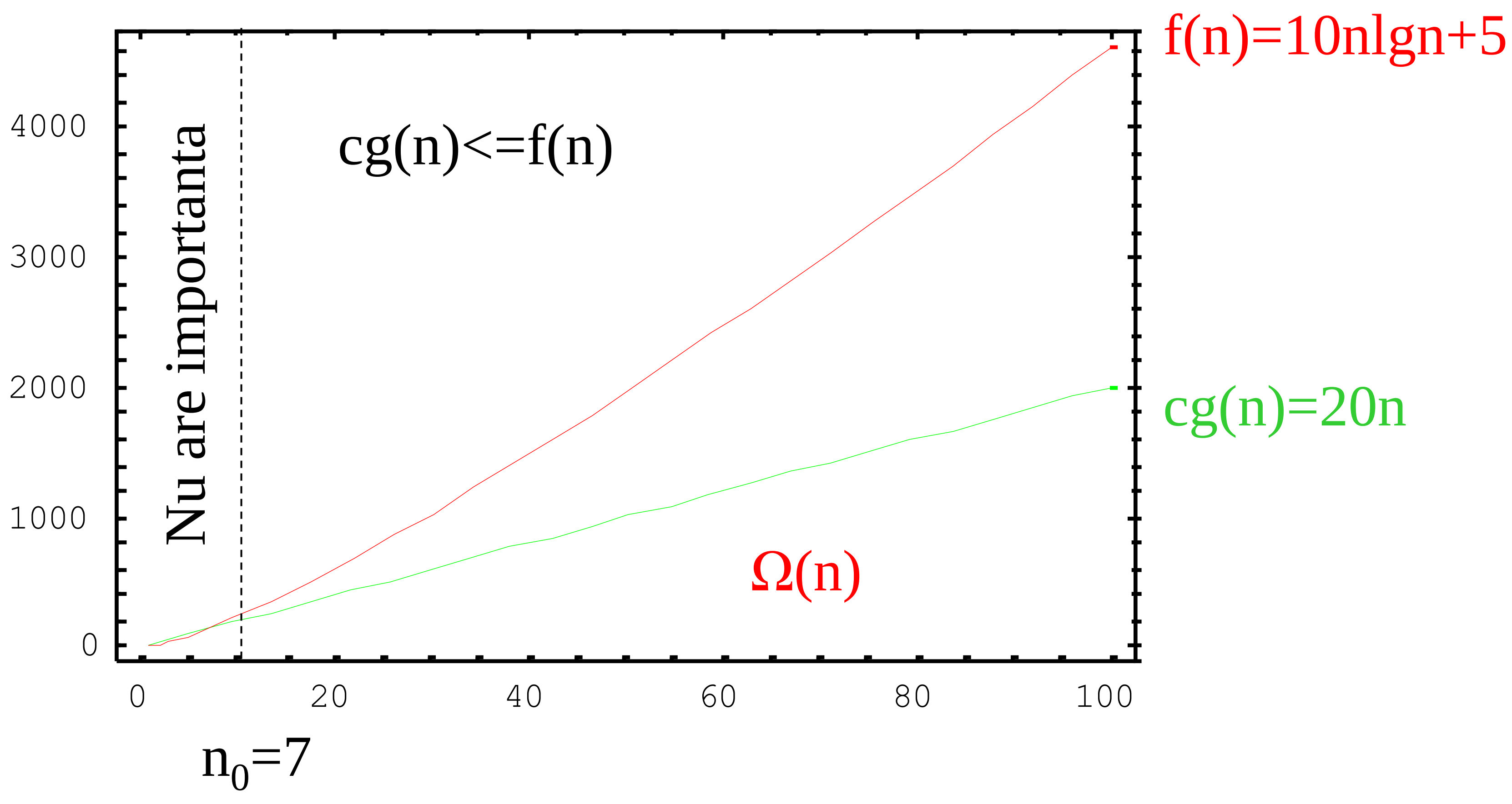
   $c=3$, $n_0=1$, $g(n)=n$

2. $6 <= T(n) <= 3(n+1) \Rightarrow T(n) \in \Omega(1)$

   $c=6$, $n_0=1$, $g(n)=1$

Notația Ω

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



Notatia Ω. Proprietăți

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

Atunci $T(n)/n^k \to a_k$ (cand $n\to\infty$).
Astfel pentru orice $\varepsilon>0$ există $N(\varepsilon)$ astfel încât

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

In ipoteza că $a_k-\varepsilon>0$, rezultă că $(a_k-\varepsilon)n^d <=(a_k-\varepsilon)n^k <= T(n)$
Considerând $c=a_k-\varepsilon$ si $n_0=N(\varepsilon)$ se obține

$cn^d <= T(n)$ pentru orice $n>n_0$, adică $T(n) \in \Omega(n^d)$

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

Notația Ω. Proprietăți

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

Dem. Este suficient să se ia în considerare marginea inferioară din definiția notației Θ

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

Exemple:

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

$f(n) >= 10g(n)$ pentru orice $n>=1$ => $f(n) \in \Omega(g(n))$

Dar nu există constante c și n_0 astfel încât:

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

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

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

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

Obs: se ia în considerare cea mai costisitoare prelucrare din secvenţă

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 Θ 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 ← 1
FOR i ← 1,n DO
    m ← 3*m                              {m=3^i}
    FOR j ← 1,m DO
        prelucrare de cost Θ(1)          {aceasta e operatia dominanta}
```

Ordinul de complexitate al prelucrării este:

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

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 3
pătratic	$O(n^2)$	Sortare prin inserție → curs 6
cubic	$O(n^3)$	Inmulțirea a două matrici nxn → curs 3
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,…,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←1

d ← True

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

  d ← NOT (caut(x[i+1..n],x[i]))

  i ← i+1

endwhile

return d

Dim. problemei: n

1<= T(n)<=T'(n-1)+T'(n-2)+…+T'(1)

1<=T(n)<=n(n-1)/2

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

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

i ← s

while i<f AND x[i]!=v do

  i ← i+1

endwhile

if x[i]==v then return True

else return False

endif

Dim. subproblemei: k=f-s+1

1<= T'(k)<=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<= T(n)<=2n

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<= T(n)<=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,…,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 in 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 măsura 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 cat 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

- Variabile contor (în cazul in 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…

… analiza corectitudinii algoritmilor

- Precondiții
- Postcondiții
- Proprietăți invariante