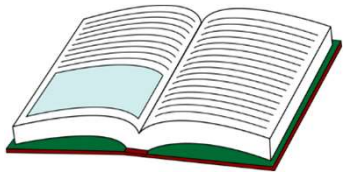
Tehnici de programare
2020

ovidiu.banias@aut.upt.ro

Scurtă prezentare



Curs
(14 săptămâni)

Test 1
(săptămâna x)

Proiect
(săptămâna 12-13)

Lucrări practice
(14 săptămâni)

Test 2
(săptămâna y)

Grilă
(sesiune)

Tehnici de programare

Analiza algoritmilor

Pointeri. Lucru pe biți

Stiva

Backtracking

Recursivitate

Divide at Impera

Greedy

Programare dinamică

Bibliografie selectivă

- Tudor Sorin, *Tehnici de programare*, Teora, 1995
- Cormen T.H., Leiserson C.E., Rivest R.R., *Introducere în algoritmi*, Agora, 2000
- Ivaşc C., Prună M., *Bazele informaticii*, Petrion, 1995
- Atanasiu. A, Pintea R., *Culegere de probleme pascal*, Petrion, 1996
- Manz. D, et. al., *Informatica. Culegere de probleme rezolvate şi propuse*, Mirton, 2005
- Ionescu C., et. al., *Informatica pentru grupele de performanță*, Dacia Educațional, 2004
- Ciocârlie H., Ciocârlie R., *Tehnici de programare şi structuri de date*, Eurostampa, 2010
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00sc-introduction-to-computer-science-and-programming-spring-2011/
- http://videolectures.net/mit6046jf05_introduction_algorithms/
- https://class.coursera.org/algo-004/lecture

Principii generale. Pași implementare

Analiză Proiectare Implementare

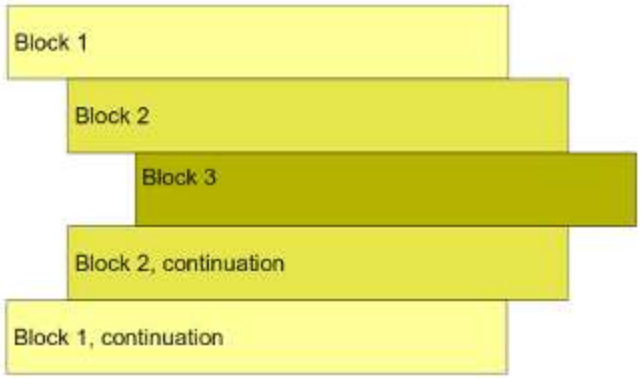
- Înțelegerea corectă a cerinței problemei
- Verificarea constrângerilor (timp de execuție, memorie, ...)

- În funcție de timpul alocat rezolvării problemei

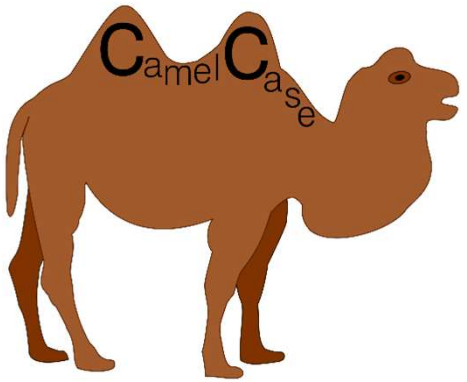
- Scriere cod
- Depanare
- Testare

Principii generale.Identare/Notații cod/Comentare

Identare (K&R, KNF, GNU)



Notații cod (Camel Case / Hungarian)



Comentare

Stil de programare

modularitate

claritate

aspect

lizibilitate

robustețe

Analiza algoritmilor

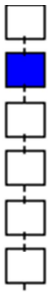
Problemă

Se dă o listă de n elemente, $A[1..n]$ și un element auxiliar x.

Să se găsească indexul i, cu proprietatea că $A[i]=x$, $1 \leq i \leq n$.

- **exemple**
- **2 metode**

Căutare liniară



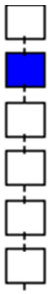
Algoritm: LINEARSEARCH

Input: vector A[1..n] de n elemente și un element x.

Output: i dacă x = A[i], 1 ≤ i ≤ n, și 0 în caz contrar.

- câte comparații minim?
- câte comparații maxim?

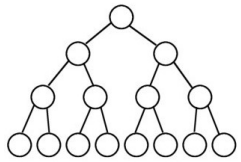
Căutare liniară



Algoritm: LINEARSEARCH

```
1. i ← 1

2. while (i < n) and (x ≠ A[i])

3.     i ← i+1

4. end while

5. if (x = A[i]) then return i else return 0
```

Căutare binară



Algoritm: BINARYSEARCH

Input: vector A[1..n] de n elemente sortate crescător și elementul x

Output: i dacă x == A[i], 1 ≤ i ≤ n, și 0 în caz contrar.

Observație: fie **li** și **lf** doi indecși ai vectorului **A[1..n]**, reprezentând limita inițială și respectiv limita superioară a unui subvector **A[li..lf]**.

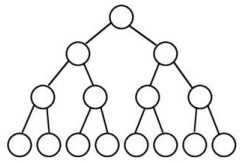
Se observă că:

A[li] ≤ A[(li+lf)/2] ≤ A[lf]

Rezolvare: la fiecare pas se va împărți intervalul în jumătate și elementul **x** se va căuta în doar unul din cele două intervale de indecși **A[li..(li+lf)/2]** şi **A[(li+lf)/2+1..lf]**, în funcție de valoarea **A[(li+lf)/2]** .

- câte comparații minim? câte comparații maxim?

Căutare binară



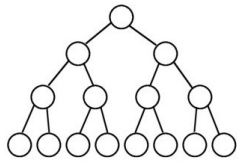
Algoritm: BINARYSEARCH – exemplificare (element căutat x=37)

Index (i)	1	2	3	4	5	6	7	8	9	10
A[1..10]	3	7	12	21	25	26	27	29	37	39

- li=1, lf=10
- pas 1: k=[(li+lf)/2] = 5
- pas 2: se obțin 2 intervale [1..5] și [6..10]
- pas 3: pt. că A[k]=25 < 37 se alege intervalul A[6..10]
- pas 4: li=6, lf=10, repeta pasul 1 până (li==lf) sau A[k]==x

Căutare binară

Algoritm: BINARYSEARCH – exemplificare (element căutat x=37)



Index (i)	6	7	8	9	10
A[6..10]	26	27	29	37	39

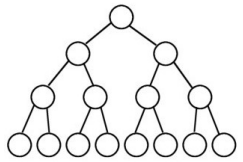
k=8, (A[k]=29)<37→ A[8..10]

Index (i)	6	7	8	9	10
A[8..10]	26	27	29	37	39

k=9, A[k]=37→ OK

http://www.infoarena.ro/blog/cautare-binara

Căutare binară

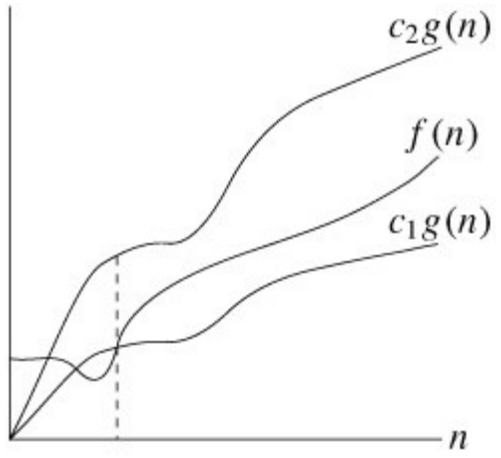


Algoritm: BINARYSEARCH – pseudocod

```
1. li ← 1; lf ← n;
2. while (li ≤ lf)
3.    k ← [(li + lf) / 2]
4.    if (x = A[k]) then return 1
5.    else if (x < A[k]) then lf ← k-1
6.                       else li ← k+1
7. end while
8. return 0
```

- după fiecare pas i, numărul de elemente se înjumătățeşte: pas i → $\frac{n}{2^{i-1}}$ elemente
- la ultimul pas, $\left\lfloor \frac{n}{2^{i-1}} = 1 \right\rfloor \Rightarrow i = \lfloor \log_2 n + 1 \rfloor$

Notații asimptotice



O limita sperioară

Θ limita inferioară == limita superioară

Ω limita inferioară

Notații asimptotice - O

Definiție:

fie $f(n), g(n) : \mathrm{N} \to R$

spunem că complexitatea lui $f(n) \overset{(not)}{=} O(g(n))$

dacă $\exists\, n_0 \in N$ şi $c = const.$

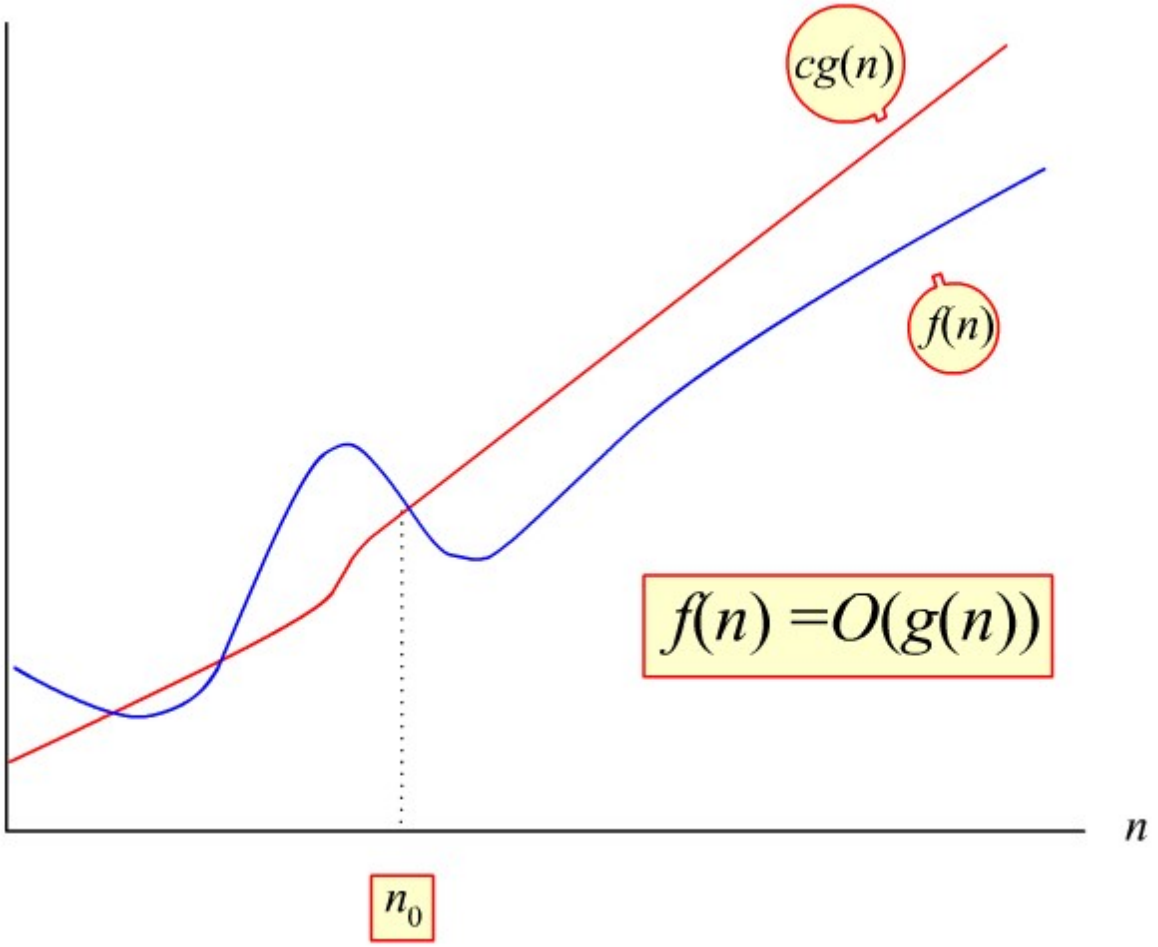
a.î. $\forall n \geq n_0, f(n) \leq cg(n).$

Fie $f(n) = 2n^3 + 10n^2 + 2, \forall n \geq 1$

$f(n) \leq 12n^3 \Rightarrow$ complex. $f(n) \overset{(not)}{=} O(n^3)$,

pt. că $\exists n_0 = 1$ şi $c = 12$ a.î. $\forall n \geq n_0, f(n) \leq cg(n).$

Notații asimptotice - O



$f(n), g(n) : \mathrm{N} \to R$, complexitatea lui $f(n) \overset{(not)}{=} O(g(n)) \overset{(not)}{=} g(n)$
dacă $\exists\, n_0 \in N$ şi $c = const.$ a.î. $\forall n \geq n_0, f(n) \leq cg(n).$

Notații asimptotice – clasificare algoritmi

$O(1)$ = constant

$O(\log n)$ = logaritmic

$O(n)$ = liniar

$O(n \log n)$ = supraliniar

$O(n^2)$ = patratic

$O(n^c)$ = polinomial

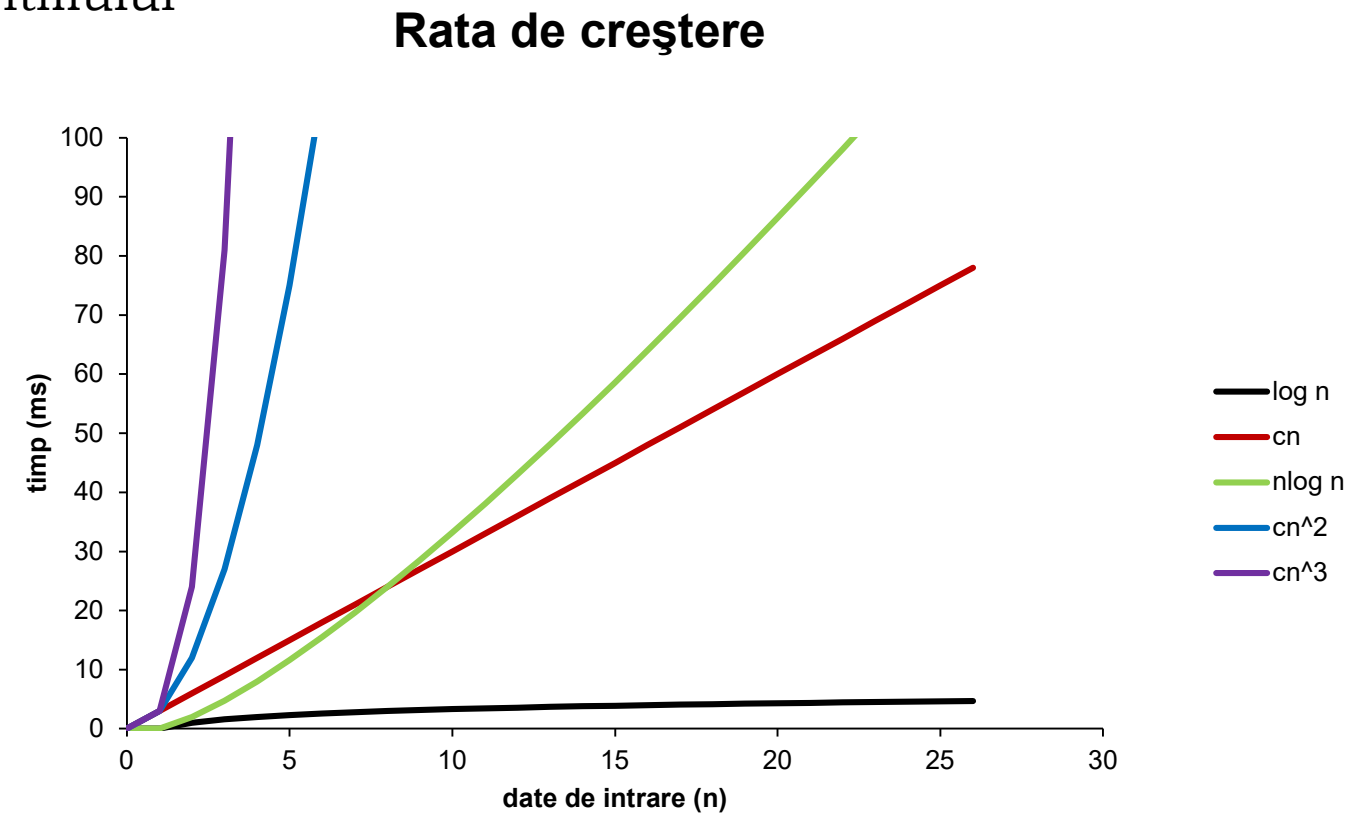
$O(c^n)$ = exponential

$O(n!)$ = factorial

c - constantă

Ordinul/rata de creştere a unui algoritm

- timpul de execuție a unui algoritm este o funcție de dimensiunea datelor de intrare și de complexitatea algoritmului



- $f(n)=n^3+n^2$
- $f(n)=n\log n+n+3$
- cu cât n creşte cu atât scade importanța termenilor de grad inferior

Ordinul/rata de creştere a unui algoritm

n	O($\log n$)	O(n)	O($n \log n$)	O(n^2)	O(n^3)	O (2^n)
$2^7 = 128$	$0.007\mu s$	$0.128\mu s$	$0.89\mu s$	$16\mu s$	$2ms$	$10^{31} ani$
$2^8 = 256$	$0.008\mu s$	$0.256\mu s$	$2\mu s$	$65\mu s$	$16ms$	$10^{69} ani$
$2^{10} = 1024$	$0.01\mu s$	$1\mu s$	$10\mu s$	$1ms$	1sec	$10^{300} ani$
$2^{12} = 4096$	$0.012\mu s$	$4\mu s$	$49\mu s$	$16ms$	68sec	$10^{1221} ani$
2^{20}	$0.02\mu s$	$1ms$	$20ms$	18.3min	$37 ani$	$10^{314565} ani$

Operaţii elementare: (i) adunare, scadere, înmulțire şi împărțire

(ii) comparații şi operatori logici

(iii) atribuiri

Notații asimptotice - Ω, Θ

Definiție: fie $f(n), g(n) : \mathrm{N} \to R$

$$f(n) \overset{(not)}{=} \Omega(g(n))$$

dacă $\exists\, n_0 \in N$ şi $c = \text{const}$. a.î. $\forall n \geq n_0$,

$f(n) \geq cg(n)$.

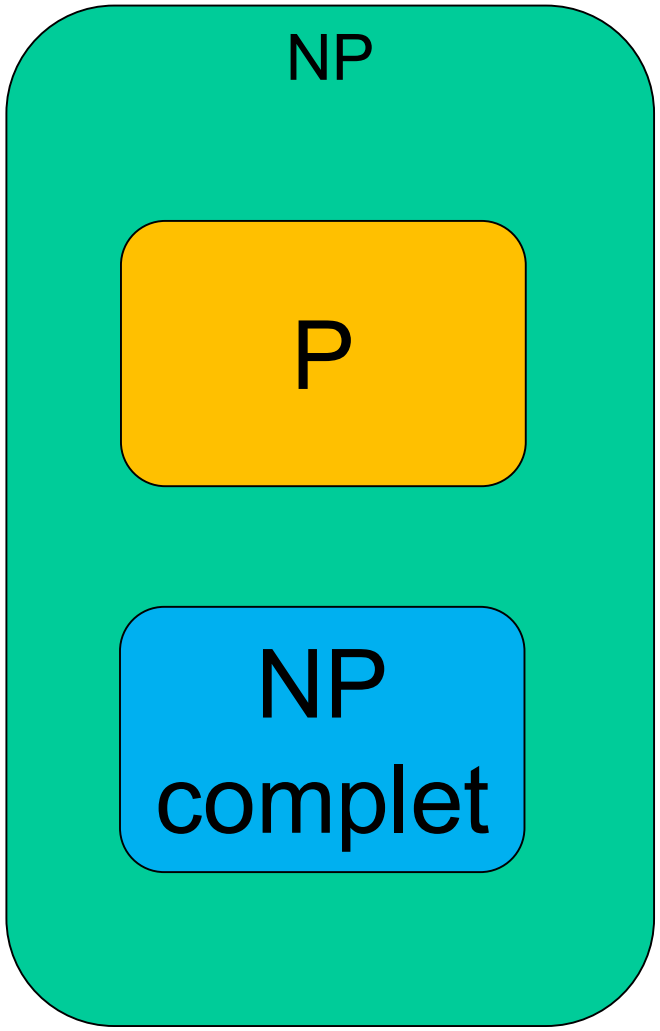
$$f(n) \overset{(not)}{=} \Theta(g(n))$$

dacă $\exists\, n_0 \in N$ şi $c_1, c_2 = \text{const}$. a.î. $\forall n \geq n_0$,

$c_1 g(n) \leq f(n) \leq c_2 g(n)$.

Complexitate P/NP/NP-complet

- Algoritmi deterministici
- Algoritmi nondeterministici
- P = Polinomial
- NP = Nondeterministic Polinomial
- NP-complet



$P \neq NP$

Rezumat. Analiza algoritmilor

- Căutare liniară vs. Căutare binară
- Notații asimptotice - O
- Ordinul/rata de creștere a unui algoritm
- Tipuri de algoritmi