STRUCTURI DE DATE

SEMESTRUL II

2019/2020

Lect. dr. Trîmbițaș Maria-Gabriela

Dedicat memoriei lui Mihai Pătraşcu

Mihai Pătraşcu, considerat cel mai bun cercetător în informatică teoretică din ultimii 10 ani

Născut pe 17 iulie 1982 la Craiova, **Mihai Pătraşcu** și-a finalizat studiile universitare la *Massachusetts Institute of Technology din Statele Unite*, unde a și lucrat ulterior.

Teza lui de doctorat, susținută aici, a fost premiată drept cea mai bună teza științifică de la MIT, care este cea mai bine cotată universitate la informatică din lume.

Mihai a câștigat multe medalii la olimpiadele de specialitate, ajungând pe locul 20 în topul medaliaților la nivel internațional. A obținut de 9 ori premiul I la nivel național și 7 medalii la fazele internaționale: 4 de aur și 3 de argint.

A fost membru al Comitetului Stiințific al Olimpiadei Internaționale de Informatică, presedintele Comitetului științific al Balcaniadei de Informatică (2011) și al Olimpiadei Europene de Informatică (2009).

In 2012, Mihai Patrascu a primit premiul *Presburger*, de la Asociația Europeană de Informatică Teoretică, pentru *„revoluționarea domeniului de structuri de date”.*

În vârstă de numai 29 de ani, a murit pe data de 5 iunie 2012.

Cuprinsul cursului

1. **Introducere. Structuri de date. Algoritmi.**
2. **Complexități**
3. **Tipuri de date. TAD Colecție - Concepte legate de colecție**
4. **TAD – Tablou**
5. **TAD Dicționar – Concepte legate de dicționare**
6. **TAD Lista - Concepte legate de liste**
7. **TAD Stiva – Concepte legate de stivă**
8. **TAD Coadă - Concepte legate de coadă**
9. **TAD Coadă cu priorităţi – Concepte legate de coada cu priorități**
10. **Tabela de dispersie (hash-table)**
11. **TAD Arbore – Concepte legate de arbori**
12. **Heap-uri**
13. **Arbori binari de căutare. Arbori echilibrați**

Ordinea ar putea fi modificată

Bibliografie

1. CORMEN, THOMAS H. - LEISERSON, CHARLES - RIVEST, RONALD R.: Introducere în algoritmi. Cluj-Napoca: Editura Computer Libris Agora, 2000.
2. SIMONAS SALTENIS:Algorithms and Data Structures, 2002.
3. STANDISH, T.A.: Data Structures, Algorithms & Software Principles in C, Addison-Wesley, 1995
4. FRENTIU M., POP H.F., SERBAN G.: Programming Fundamentals, Ed.Presa Universitara Clujeana, Cluj-Napoca, 2006, 234 pagini
5. SEDGEWICK, R.: Algorithmen, Addison-Wesley,1998
6. WIRTH, N.: Algorithmen und Datenstrukturen, Pascal Version, 5 Auflage, B.G. Teubner Stuttgart,1998
7. NICULESCU V., CZIBULA G.: Structuri fundamentale de date. O perspective orientate obiect. Editura Casa Cartii de Stiinta, Cluj-Napoca,2011
8. HOROWITZ, E.: Fundamentals of Data Structures in C++. Computer Science Press, 1995.
9. MOUNT, DAVID M.: Data Structures. University of Maryland, 1993.
10. AMBSBURY, WAYNE: Data Structures. From Arrays to Priority Queues, 1993.
11. BRUNO R. PREISS, Data Structures and Algorithms with Object-Oriented Design Patterns in C++, 1997.

1. Introducere

- Structuri de date
- Algoritmi

Algoritm

- În acest timbru sovietic apare *Muhammad ibn Musa al-Khwarizmi* (născut aproximativ în 780; mort între 835 şi 850), matematician persan şi astronom din Khorasan provincie a Uzbekistanului de azi. Cuvântul “algoritm” provine din numele lui.

❖Definiție (generală) Un algoritm este o ***descriere precisă și finită*** a unui proces constând din ***pași elementari.***

❖Definiție (Computer Science) Un algoritm este o descriere precisă și finită a unui proces care este

(a) dată într-un ***limbaj formal*** și

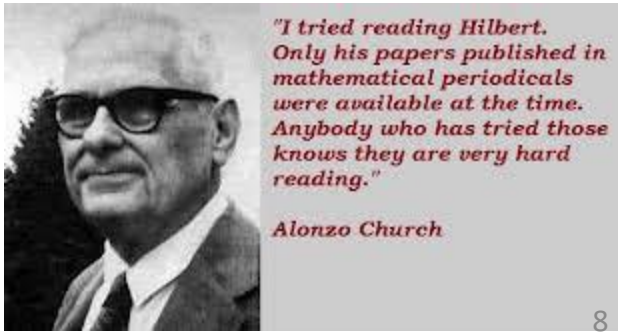
(b) constă din ***pasi*** elementari și ***executabili pe mașină.***

Reprezentarea algoritmilor

- Descriere verbală
- Grafică: Structograme, diagrame de flux, etc.
- Pseudocod
- Limbaj de programare
- ….
- Teza lui Church ⇒Toate formele de reprezentare ale algoritmilor sunt echivalente.

Alonzo Church a fost un matematician american și logician care a avut contribuții majore în logica matematică și fundamentele teoretice ale informaticii.

Wikipedia
Born: June 14, 1903, Washington, D.C., United States
Died: August 11, 1995, Hudson, Ohio, United States
Books: Introduction to Mathematical Logic



Algoritmul lui Euclid

în Pascal

```
PROGRAM cmmdc (Input,Output);
   VAR a,b: Integer;
BEGIN
   ReadLn(a,b);
      WHILE (a > 0) AND (b > 0)     DO
         IF a > b THEN
            a := a-b;
          ELSE b := b-a;
      IF b=0 THEN
        WriteLn(a)
      ELSE WriteLn(b)
END.
```

Observatie:

Algoritmii lucrează asupra datelor de intrare, generează date intermediare, iar în final produc un rezultat (dată)
O **structură de date** este modul în care data este reprezentată în interiorul masinii, în memorie sau pe disc (vezi și cursul BD)

Structurile de date determină ce pot să facă algoritmii și cu ce cost. Mai precis: … cât costă un anumit pas al unui algoritm

Complexitatea algoritmilor este strâns legată de structurile de date pe care ei le utilizează; atât de strâns încat unii subsumează ambele concepte sub termenul de “algoritm”.

I(nformaţia) = **D**(ata) + **S**(emantica)

P(rogram) = **D**(ate) + **A**(lgoritm)

Date (atomicitatea):

elementare;

structurate.

Tipuri de SD: (dpdv al memoriei și locului în memorie)

statice: articol (înregistrare), tablou, mulţime, etc;

semistatice: stivă, coadă, tabelă de dispersie;

dinamice: listă dinamică, arbore, graf;

Clasificarea SD (dpdv al abstractizării/implementării)

SD Logice

Descrierea legăturilor între elementele tipului (ex: matrice, stivă, coadă, arbore, etc.)

SD Fizice:

Se referă la reprezentarea fizică în memorie a elementelor domeniului în memorie.

Stil în proiectarea SD

Un program poate rezolva corect o problemă, dar poate fi un program „slab” din multe puncte de vedere:

- are un timp de execuţie relativ „mare”;
- utilizează spaţiu de memorie excedentar;
- dificil de depanat, de citit, de modificat;
- greu reutilizabil sau nereutilizabil în alte proiecte (programe).

SD trebuie concepute astfel încât să înlăture neajunsurile de mai sus;
3 reguli generale:

R1: Rafinarea pas cu pas (Stepwise refinement);

R2: Abstractizare:

- grupare unor componente în mod logic;
- să poată fi reutilizate de alte SD, sau alte programe;
- detaliile de implementare să fie amânate până la codificare;
- separarea descrierii logice de implementare.

R3: Independenţă de limbaj.

Exemplu rafinare: paşii în **abstractizarea** (prin rafinare) unei stive de elemente de tipul **TE**

step 1: stiva de elemente de tipul **TE**

step 2: stivă cu alocarea dinamică a elementelor de tipul **TE**

step 3: stivă cu alocarea dinamică o obiectelor care reprezintă elemente de tipul **TE**

Exemplu abstractizare

Versiunea 1

```
Nume = String[15];
Persoană =
        Record
            NumeFam,Prenume:Nume;
            CodTelefon:          0..1000;
            NrTelefon:           String[9];
            Ziua:                1..31;
            Luna:                1..12;
            Anul:                1900..2100;
        End;
```

Versiunea 2

```
Nume = String[15];
TipTelefon =
        Record
            CodTelefon: 0..1000;
            NrTelefon:  String[9];
        End;
DataCalend =
        Record
            Ziua:1..31;
            Luna:1..12;
            Anul:1900..2100;
        End;
Persoana =
        Record
            NumeFam,Prenume:   Nume;
            Telefon:       TipTelefon;
            DataNasterii:  DataCalend;
        End;
```

2. Complexități

Observații preliminare:

1. Afirmațiile despre complexitatea algoritmilor și a problemelor trebuie să fie de regulă independente de un anumit model de mașină și anumite proprietăți ale implementării, ca și de detaliile technologice

2. La studiul funcțiilor de complexitate nu interesează atât de mult evoluția exactă a valorilor unei funcții $f:N->R^+$ ci „tendința“ acesteia, comportamentul ei de creștere (comportament asimptotic) atunci când crește argumentul

Reprezentări diferite pentru acelaşi algoritm (şi programele sunt reprezentări)

În plus: algoritmi diferiţi pentru aceeaşi problemă

Scop: Compararea algoritmilor între ei (analiza complexităţii)

Criterii: Timpul de execuţie şi spaţiul de memorie

Complexitatea se exprimă în funcţie de mulţimea şi dimensiunea datelor

Determinarea timpului de execuție

Posibilităţi de măsurare a eficienţei unui algoritm:

1. Implementarea unui algoritm pe un calculator real şi măsurarea timpului
   Dezavantaj: prea multe mărimi care influenţează timpul, abstracţie făcând de algoritmul însuşi
2. Numărarea paşilor de calcul care se efectuează pentru un set de date de intrare.
   - Întrebare: ce este un pas de calcul?
   - Model formal de calcul de exemplu maşină Turing sau RAM
   - "programarea" algoritmului într-un limbaj de programare artificial şi numărarea şi evaluarea (ponderea) operaţiilor

   Dezavantaj: Efort mare, portabilitate incertă
3. Numărarea operaţiilor la nivel înalt (de exemplu *numărul comparaţiilor* la căutarea într-o listă sau numărul de perechi de elemente care se interschimbă la sortare)

   Dezavantaj: Evaluare prea grosolană

Vom folosi ca metodă simplă de măsurare timpul de execuţie

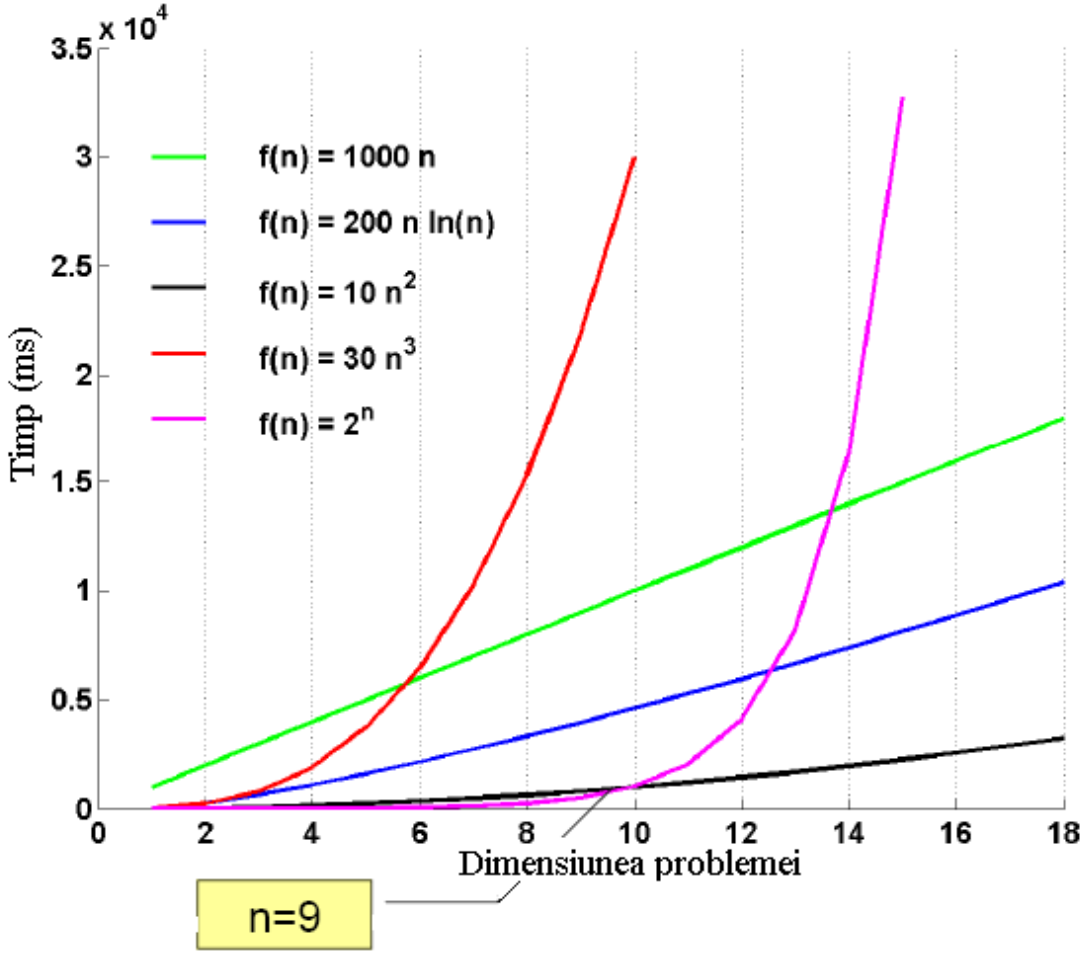
Fie dată o problemă de dimensiune *n*

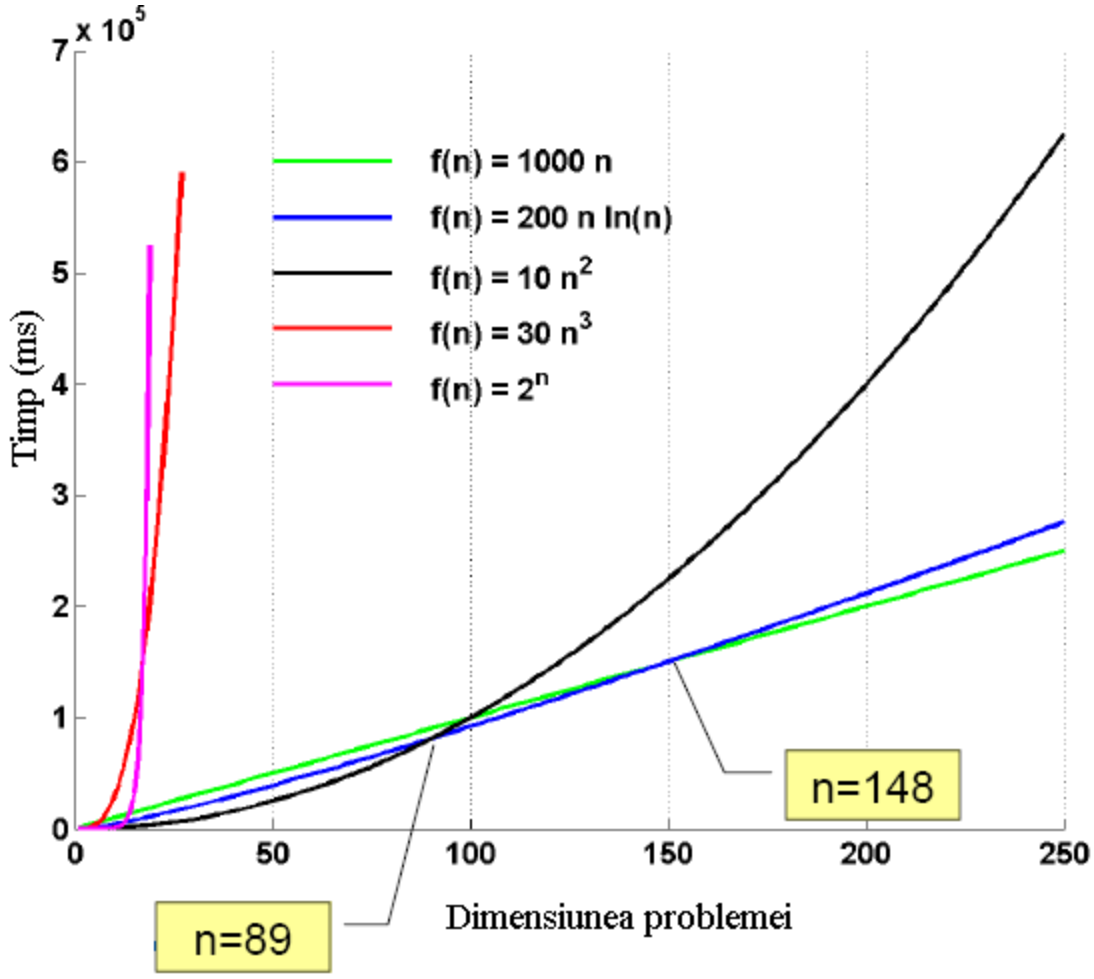
De exemplu sortarea a n valori

- Există mai mulţi algoritmi pentru rezolvarea problemei

- Care algoritm este mai rapid depinde de dimensiunea *n* a problemei

- Pentru valori foarte mari ale lui n, cel mai rapid algoritm va fi întotdeauna acela al cărui timp de execuţie creşte cel mai puţin în funcţie de *n*.


x 10^4
3.5
3
2.5
2
1.5
1
0.5
0
Timp (ms)
f(n) = 1000 n
f(n) = 200 n ln(n)
f(n) = 10 n^2
f(n) = 30 n^3
f(n) = 2^n
0
2
4
6
8
10
12
14
16
18
Dimensiunea problemei
n=9


x 10^5
7
6
5
4
3
2
1
0
Timp (ms)
f(n) = 1000 n
f(n) = 200 n ln(n)
f(n) = 10 n^2
f(n) = 30 n^3
f(n) = 2^n
0
50
100
150
200
250
Dimensiunea problemei
n=148
n=89

De regulă nu suntem interesaţi de numărul exact de operaţii ci de clase de complexitate: cum se modifică efortul de calcul, atunci când se măreşte volumul datelor de intrare cu un anumit factor?

Care este comportamentul calitativ al funcţiei de timp?

Pe noi ne intereseaza cum depinde o resursa consumata de un algoritm de n (dimensiunea problemei) pentru o valoare mare a lui n. Pentru a putea exprima aceasta, matematic corect, se folosesc simbolurile lui Landau.

Simbolurile lui Landau - O, Ω, Θ, o si ω

Definiție: g(n) este marginea asimptotică superioară a lui f(n), dacă există o constantă $c > 0$ și un $n_0 \in N$, astfel încât pentru toți $n \geq n_0$ are loc $f(n) \leq cg(n)$
Mulțimea tuturor funcțiilor f(n), pentru care o funcție g(n) dată este margine asimptotică superioară se notează cu $O(g(n))$.

Analog se defineste marginea asimptotică inferioară, marginea asimptotică strânsă, margine superioară tare, respectiv margine inferioară tare.

Definiție: Fie f și g funcții $N \rightarrow R^+$.
• Marginea superioară:
$O(g(n)) = \{f(n) \mid \exists c > 0\ \exists n_0\ \forall n \geq n_0\ \ f(n) \leq cg(n)\}$
• Marginea inferioară:
$\Omega(g(n)) = \{f(n) \mid \exists c > 0\ \exists n_0\ \forall n \geq n_0\ \ cg(n) \leq f(n)\}$
• Marginea strânsă (aceeași creștere):
$\Theta(g(n)) = O(g(n) \cap \Omega(g(n)))$
• Margine superioară tare:
$o(g(n)) = \{f(n) \mid \forall c > 0\ \exists n_0\ \forall n \geq n_0\ \ f(n) \leq cg(n)\}$
• Margine inferioară tare:
$\omega(g(n)) = \{f(n) \mid \forall c > 0\ \exists n_0\ \forall n \geq n_0\ \ cg(n) \leq f(n)\}$
Atenție: Se folosește adesea în locul scrierii corecte $f(n) \in O(g(n))$ pentru marginea asimptotică superioară, notația neglijentă $f(n) = O(g(n))$.

Proprietățile notațiilor asimptotice

Tranzitivitate:

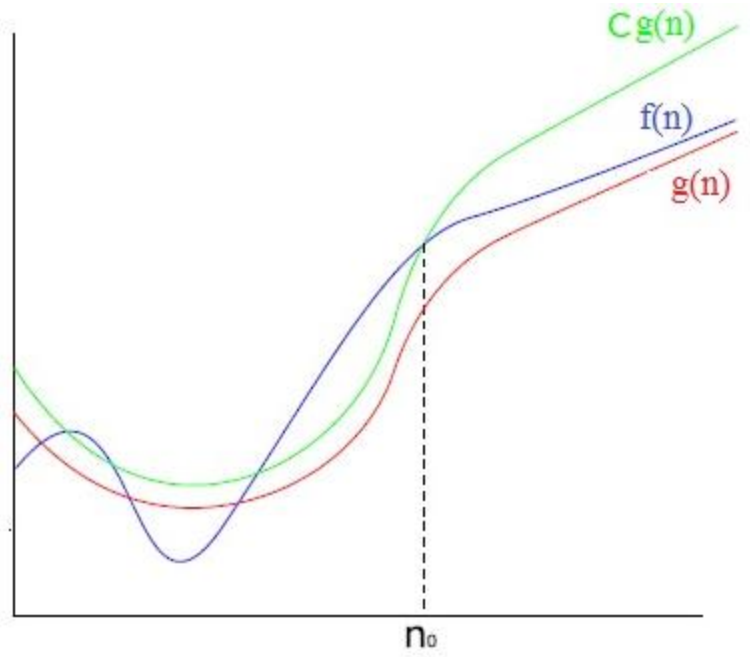
$$
\begin{array}{lllll}
f(n) \in O(g(n)) & \text{și} & g(n) \in O(h(n)) & \Rightarrow & f(n) \in O(h(n)) \\
f(n) \in o(g(n)) & \text{și} & g(n) \in o(h(n)) & \Rightarrow & f(n) \in o(h(n)) \\
f(n) \in \Omega(g(n)) & \text{și} & g(n) \in \Omega(h(n)) & \Rightarrow & f(n) \in \Omega(h(n)) \\
f(n) \in \omega(g(n)) & \text{și} & g(n) \in \omega(h(n)) & \Rightarrow & f(n) \in \omega(h(n)) \\
f(n) \in \Theta(g(n)) & \text{și} & g(n) \in \Theta(h(n)) & \Rightarrow & f(n) \in \Theta(h(n))
\end{array}
$$

Reflexivitate: $f(n) \in O(f(n)), \quad f(n) \in \Theta(f(n)), \quad f(n) \in \Omega(f(n))$

Simetrie: $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$

Antisimetrie: $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$

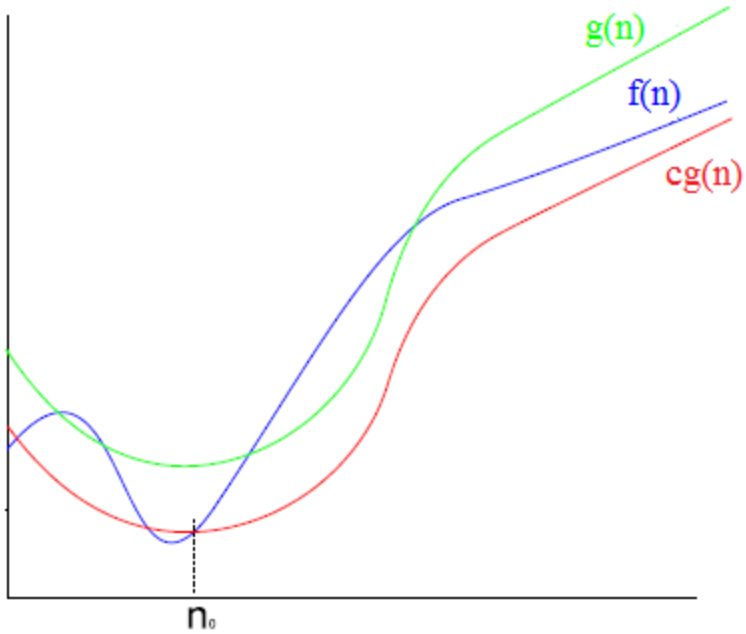
Notația $O(g(n))$



Exemplu:

Fie $f(n) = n^2+1000n$ demonstrăm că $f \in O(n^2)$ cu $g(n)=n^2$
deci se caută c>0 și $n_0 \in N$ *astfel încât pentru toți* $n \geq n_0 f(n) \leq cn^2$
$f(n) = n^2+1000n \leq n^2+1000n^2=1001n^2$ => c=1001, $n_0=1$

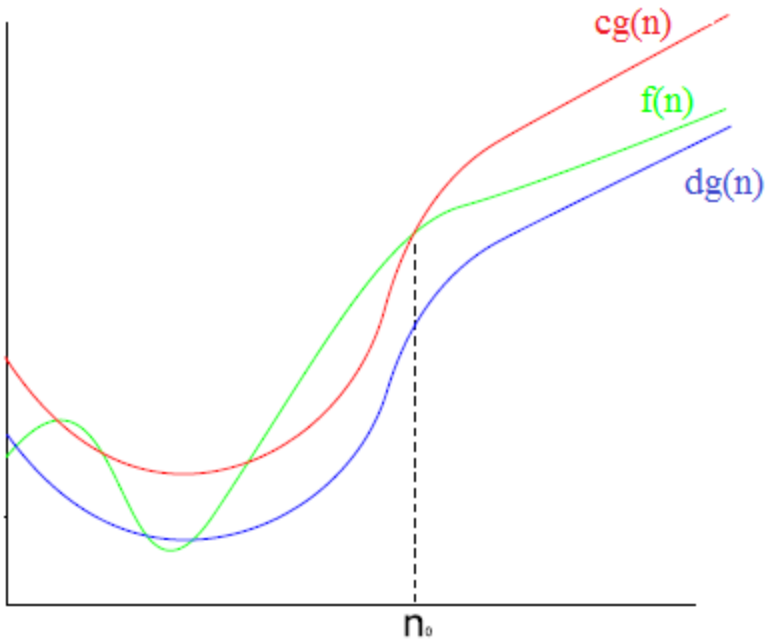
Notația $\Omega(g(n))$



Exemplu:

Fie aceeași funcție
$f(n) = n^2+1000n$ demonstrăm că $f \in \Omega(n^2)$ *cu* $g(n)=n^2$
deci se caută c>0 și $n_0 \in N$ *astfel încât pentru toți* $n \geq n_0 f(n) \geq cn^2$
$f(n) = n^2+1000n \geq 1n^2$ => c=1, $n_0=1$

Notația $\Theta(g(n))$



Din definițiile notațiilor asimptotice rezultă că
$f(n) = n^2+1000n \; f \in \Theta(n^2)$

TEMA: Să se demonstreze că $f(n) = n^2+1000n \; f \in o(n^2 \log n)$

O(g) este mulţimea tuturor funcţiilor care cresc asimptotic cel mult la fel de repede ca c·g.

De exemplu:

$$2n^2 + 5n + 13 \in O(n^2)$$

$$2n^2 + 5n + 13 \notin O(n)$$

$$5n + 13 \in O(n^2)$$

$$5n + 13 \in O(n)$$

La sume se impune termenul cu creşterea cea mai rapidă.

$f(n) = 2n^2 + 7n + 10$

creste cel mai repede

$\Rightarrow f(n) \in O(n^2)$

Demonstraţie:

$2n^2+7n+10 \leq 3\,n^2$

pentru $n_0 = 9$

S-ar putea scrie şi
$f(n) = 2n^2 + 7n + 10 \in O(2n^2 + 7n + 10)$
interesează însă numai comparaţia cu funcţii simple ca de exemplu O(n), $O(n^2)$,...

Clasa	**Denumire**	**Exemplu**
1	constant	Instructiuni elementare
log n	logaritmic	Cautare binara
n	linear	Minimum unui sir
n^2	quadratic	Procedee simple de sortare
n^3	cubic	Inversarea matricelor
n^k	polinomial	Programare Lineara
n log n	superlinear	Procedee eficiente de sortare
2^{cn}	exponential	brute-force search, Backtracking
n!	Factorial	Permutari,Problema comis-voiajorului, Met.Kramer

Avem: $O(1) \subset O(n) \subset ... \subset O(n!)$

Regulă de bază:

Cea mai mică clasă de complexitate (în notatia *O*) rezultă din $T_A(n)$ prin:

- Extragerea termenului "dominant"(cel mai mare) şi prin
- Renunţarea la coeficienţi

De exemplu:

$T(n)=60n^2+ 4n \Rightarrow T \in O(n^2)$;

$T(n)=\lg(n) + 1 \Rightarrow T \in O(\lg(n))=O(\log(n))$,

de ce? 

Ω(f) este mulţimea tuturor funcţiilor asimptotice care cresc cel puţin la fel de repede ca şi c·g.

Exemplu:

$f(n) = 2n^2 + 7n + 10$

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

deoarece

$2n^2+7n+10 \geq 2n^2$

pentru $n_0 = 1$

Θ(g) Dacă f(n) ϵ Ω(g) şi f(n) ϵ O(g), atunci f(n) ϵ Θ(g) .

Există o limită superioară c_1şi o limită inferioară c_2, astfel încât din punct de vedere asimptotic pentru toţi $n>n_0$) este adevărată:
$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

Exemplu:

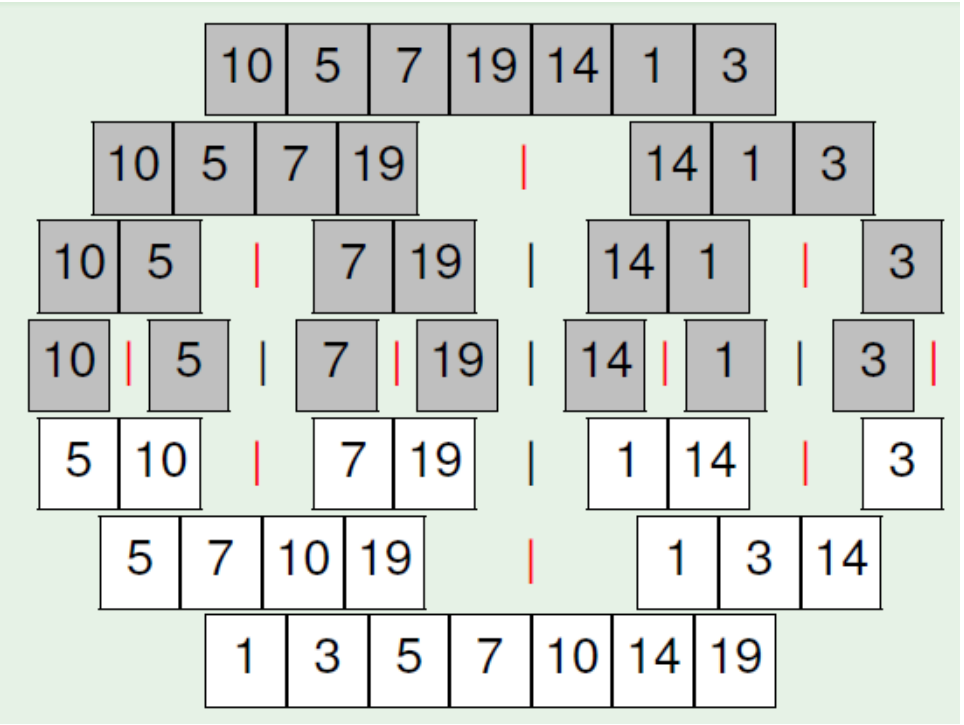
$f(n) = 2n^2 + 7n + 10$
$\Rightarrow f(n) \in \Theta(n^2)$ pentru $n_0 = 9$

Demonstraţie:

$3n^2 \geq 2n^2+7n+10 \geq 2n^2$ pentru $n_0 = 9$

Exemplu: Algoritm de tip “Divide et impera”

MergeSort




```
Procedure MergeSort (V,p,q)
        if  p ≤ q   then
                m=(p+q) div 2
                MergeSort (V,p,m)
                MergeSort (V,m+1,q)
                Merge (V,p,m,q)
        endif
EndProcedure
```

La algoritmul de sortare prin interclasare avem divizarea în 2 subsecvenţe de lungime n/2; deci a=2, iar dimensiunea unei probleme este 1/2

dacă p = q avem timp constant θ(1) (n=1, deci **ε**=1);

D(n) = θ(1), pentru că se determină indicele de mijloc al secvenţei (p,q);

C(n) = θ(n), (pentru că interclasarea a 2 secvenţe de lungime p, respectiv q dă θ (p+q-1))

Avem deci:

$$T(n) = \begin{cases} \theta(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + \theta(1) + \theta(n), & n > 1 \end{cases} \quad \text{sau} \quad T(n) = \begin{cases} \theta(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + \theta(n), & n > 1 \end{cases}$$

În final avem recurenţa $$T(n) = \begin{cases} 0, & n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & n > 1 \end{cases}$$

Rezolvată la seminar prin metoda iteraţiei

Teorema master

Fie $a \geq 1$ și $b > 1$ constante, fie $f(n)$ o funcție oarecare și $T(n)$ o funcție definită pe $\mathbb{N}$ prin recurența

$$T(n) = aT(n/b) + f(n),$$

unde n/b se interpretează fie ca $\lfloor \frac{n}{b} \rfloor$ fie ca $\lceil \frac{n}{b} \rceil$. Atunci $T(n)$ poate fi delimitată asimptotic după cum urmează

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & \text{dacă } \exists \epsilon > 0 \text{ și } f(n) = O\left(n^{\log_b a - \epsilon}\right) \\ \Theta\left(n^{\log_b a} \log n\right) & \text{dacă } f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right) & \text{dacă } \exists \epsilon > 0 \text{ și } \exists c < 1 : af\left(n/b\right) \leq cf(n) \\ & \text{și } f(n) = \Omega\left(n^{\log_b a + \epsilon}\right). \end{cases}$$

- În fiecare caz comparăm $f(n)$ cu $n^{\log_b a}$. *Intuitiv*, soluția este determinată de cea mai mare dintre cele două funcții:
- Dacă $n^{\log_b a}$ este mai mare (cazul 1), atunci $T(n) = \Theta\left(n^{\log_b a}\right)$.
- Dacă $f(n)$ este mai mare (cazul 3), atunci soluția este $T(n) = \Theta(f(n))$.
- Dacă cele două funcții au același ordin de mărime (cazul 2), înmulțim cu un factor logaritmic, iar soluția este

$$f(n) = \Theta\left(n^{\log_b a} \log n\right) = \Theta(f(n) \log n).$$

Revenim la recurenţa obţinută la MERGESORT

$$T(n) = \begin{cases} 0, & n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & n > 1 \end{cases}$$

şi aplicând Teorema master avem a=b=2 => f(n)=θ(n), deci ne aflăm în cazul 2

=> T(n) = θ(n log n)

Problemă tratabilă = problema care poate fi rezolvată printr-un algoritm cu timp de execuţie polinomial

Cazul cel mai nefavorabil, mediu şi cel mai favorabil

La timpul de execuţie deosebim:

– cazul cel mai defavorabil (worstcase),

– cazul mediu (averagecase) şi

– cazul cel mai favorabil (best case).

• Cazul mediu este important atunci când, cazul cel mai favorabil şi cazul cel mai defavorabil sunt excepţii foarte rare

Notația *O* - are originea în lucrarea *Die Elemente der Zahlentheorie* din anul 1892 a lui
Paul Gustav Heinrich Bachmann (1837 - 1920) .

Puțin timp mai târziu, a utilizat-o și specialistul in Teoria numerelor Edmund Landau (1877 -1938) și pe lângă ‚*O'* a considerat și ‚o'. Din această cauză notația asimptotică este denumită și *„Simbolurile lui Landau"*

Tot lui Landau i se datorează și notația Z pentru numerele întregi.

Incă un exemplu de analiză a complexității unui algoritm

Algoritmul standard pentru înmulțirea a doua matrice

```
function MATMULT(A,B,l,m,n)
  C = zeroes(l,n)          // zeroizare matrice rezultat C
  for i = 1 to l           // ciclu dupa liniile lui C
    for k = 1 to n         // ciclu dupa coloanele lui C
      for j = 1 to m       // ciclu dupa coloanele A / liniile lui B
        C(i,k) = C(i,k) + A(i,j) * B(j,k)   // calculul sumei de produse
      end
    end
  end
return C
```

Ordinea pentru cele trei ciluri *for* poate fi schimbată. Deoarece cele trei cicluri sunt independente unele de altele, numărul operațiilor este de ordinul $O(l \cdot m \cdot n)$

Ca urmare timpul de execuție pentru matrice pătratice (l=m=n) este cubic, deci de ordinul $O(n^3)$