

# Complexitatea algoritmilor

**Facultatea de Matematică și Informatică**

Feb. 2019

# Complexitatea algoritmilor

- Dorim să **comparăm algoritmii**: care este mai “bun”, verificând algoritmul pentru aceleași date de intrare.
- Cum evaluăm un algoritm (criterii de comparație):

→ **Timp de execuție**

=> **Complexitatea în timp**

→ **Memoria ocupată**

=> **Complexitate spațiu**

De multe ori este nevoie de un compromis între complexitatea timp/spațiu

# Complexitatea timp

→ Timpul de execuție dat în secunde?  
Depinde de mașina pe care rulăm.

→ Tipul de date influențează (int, long long, long double, etc...)?  
Da, dar nu semnificativ (exemplu)

=> Avem nevoie de ceva independent de mașină și totodată semnificativ:  
numărul de operații (exemplu)

- nu va fi necesar numărul exact de operații, ci “ordinul” numărului de operații executate de algoritm pentru un set de date cu dimensiune fixată (termenul dominant).

## Dimensiunea datelor de intrare

# of records	10	20	50	100	1000	5000
algorithm 1	0.00s	0.01s	0.05s	0.47s	23.92s	47min
algorithm 2	0.05s	0.05s	0.06s	0.11s	0.78s	14.22s

- Valoarea  **$M$**  pentru care trebuie formulat răspunsul
- Număr de elemente: vector cu  **$N$**  elemente
- Set de variabile  **$N_1, N_2, \dots, N_k$**
- Determinăm complexitatea timp ca funcție de aceste variabile

$$f(N_1, N_2, \dots, N_k)$$

- O unitate de timp (constantă) pentru operațiile matematice / logice sau atribuire.

## Definiții formale

- Big-O
  - Spunem că  $f(N)$  este  $O(g(N))$  dacă există  $c$  și  $N_0$  astfel încât: pentru orice  $N > N_0$  avem  $f(N) < c * g(N)$
- Big- $\Omega$ 
  - Spunem că  $f(N)$  este  $\Omega(g(N))$  dacă  $g(N)$  este  $O(f(N))$
- Big- $\Theta$ 
  - Spunem că  $f(N)$  este  $\Theta(g(N))$  dacă  $f(N)$  este  $O(g(N))$  și  $g(N)$  este  $O(f(N))$

Spunem că un algoritm (sau un program) are complexitatea  $O(g(N))$  dacă funcția sa de complexitate  $f(N)$  este  $O(g(N))$ .

# Exemple

→ Pentru două seturi de instrucțiuni succesive, cu numărul de operații **N**, respectiv **M**: numărul de operații va fi **N+M**:

```
for(int i=0;i<N;i++)    x++;  
for(int j=0;j<M;j++)    y++;
```

→ Pentru două bucle imbricate (cu numărul de iterații **N**, respectiv **M**): numărul de operații va fi **N\*M**:

```
for (i = 0; i < N; i + +)  
    for (j = 0; j < M; j + +)  
        {x + +; y + +;}
```

→ Dacă algoritmul nu conține bucle (for, while, repeat, do etc.) sau recursii => număr de operații constant (în majoritatea cazurilor)

# Numărarea pas-cu-pas

## C

```
• int result=0,N,M;
• for (int i=0; i<N; i++)
•   for (int j=i; j<N; j++)
•     {
•       for (int k=0; k<M; k++)
•         {
•           int x=0;
•           while (x<N)
•             {
•               result++;
•               x+=3;
•             }
•         }
•       for (int k=0; k<2*M; k++)
•         if (k%7 == 4) result++;
•     }
```

## PASCAL

```
• var result,i,j,k,x,N,M:integer;
• result:=0;
• for i := 0 to n-1 do
•   for j := i to n-1 do
•     begin
•       for k := 0 to M-1 do
•         begin
•           x:=0;
•           while x<N do
•             begin
•               inc(result);
•               x:=x+3;
•             end;
•         end;
•       for k := 0 to 2*M-1 do
•         if k mod 7 = 4 then inc(result);
•       end;
```

Calculați funcția de complexitate  $f(N,M)$  a programului de mai sus.

## Complexitatea timp

- cazul cel mai defavorabil;
- cazul cel mai favorabil;
- timpul mediu de execuție:
  - se determină valoarea medie a timpului de execuție după distribuția de probabilitate a datelor de intrare: avem valorile posibile și fiecare set de valori are o probabilitate de apariție și un timp de execuție => o medie ponderată.



# Funcții uzuale de complexitate

- $f(N)$  este  $O(\log N)$ : complexitate logaritmică (nu contează baza!)
- $f(N)$  este  $O(N)$ : complexitate liniară
- $f(N)$  este  $O(N^2)$ : complexitate pătratică
- $f(N)$  este  $O(N^3)$ : complexitate cubică
- $f(N)$  este  $O(N^k)$ ,  $k$  oarecare: complexitate polinomială
- $f(N)$  este  $O(2^N)$ : complexitate exponențială (nu contează baza!)
- $f(N)$  este  $O(1)$ : complexitate constantă

# Complexitate timp

- Timp fixat de rezolvare a unei probleme
- Se aproximează care este dimensiunea maximă a problemei ce se poate rezolva în durata de timp impusă:

complexity	maximum $N$
$\Theta(N)$	100 000 000
$\Theta(N \log N)$	40 000 000
$\Theta(N^2)$	10 000
$\Theta(N^3)$	500
$\Theta(N^4)$	90
$\Theta(2^N)$	20
$\Theta(N!)$	11

Table 3. Approximate maximum problem size solvable in 8 seconds.

## Natura datelor de intrare

- Complexitatea spațiu și complexitatea timp depind de natura datelor de intrare. :

### Exemplu:

$T$  turiști vizitează orașe, iar o vizită este dată prin tripletul (**id\_calator**, **coord1\_oras**, **coord2\_oras**), unde **id\_calator**  $\in \{1, \dots, T\}$ ,  $1 \leq \text{coord1\_oras}, \text{coord2\_oras} \leq N$ . Se citesc  $V$  vizite (triplete). Determinați orașul cel mai vizitat.

Cazuri:

- 1)  $1 \leq N \leq 100, 1 \leq T \leq 10^{10}, 1 \leq V \leq 10^{11}$
- 2)  $1 \leq N \leq 10^{19}, 1 \leq T \leq 10^4, 1 \leq V \leq 10^4$

**Soluții posibile? Complexitatea?**

## Alt exemplu

### C

- `int j=0,N,D;`
- `for (int i=0; i<N; i++)`
- `{`
- `while((j<N-1)&&(A[i]-A[j]>D))`
- `j++;`
- `if (A[i]-A[j] == D)`
- `return 1;`
- `}`

### PASCAL

- `var i,j,N,D:integer;`
- `j:=0;`
- `for i := 0 to N-1 do`
- `begin`
- `while (j<N-1)&&(A[i]-A[j]>D) do`
- `inc(result);`
- `if (A[i]-A[j] == D) then exit;`
- `end;`

Calculați funcția de complexitate  **$f(N)$**  a programului de mai sus.