# Complexitatea algoritmilor

P.C. Pop[1]

[1] Departmentul de Matematică şi Informatică

Universitatea Tehnică Cluj-Napoca, Centrul Universitar Nord din Baia Mare

# Outline

# Complexitatea algoritmilor

- Un algoritm este o listă de instrucţiuni care rezolvă orice instanţă a unei probleme într-un număr finit de paşi.
- O instanţă a unei probleme este un set de date care este obţinut când toţi parametrii care definesc problema sunt fixaţi.
- Output-ul unui algoritm este yes în cazul în care obţinem o soluţie şi no altfel.
- Nu orice problema poate fi rezolvată algoritmic. Considerăm un numar natural $n \in N$ şi următoarele probleme:
  - Să se construiască mulţimea divizorilor lui $n$,
  - Să se construiască mulţimea multiplilor lui $n$.
- Presupunem că fiecare instanţa este descrisă ca o inşiruire într-un cod binar. Prin mărimea (dimensiunea) unei instanţe a unei probleme vom întelege numărul de biţi necesari pentru a reprezenta instanţa.

# Complexitatea algoritmilor

- Un algoritm este o listă de instrucţiuni care rezolvă orice instanţă a unei probleme într-un număr finit de paşi.
- O instanţă a unei probleme este un set de date care este obţinut când toţi parametrii care definesc problema sunt fixaţi.
- Output-ul unui algoritm este yes în cazul în care obţinem o soluţie şi no altfel.
- Nu orice problema poate fi rezolvată algoritmic. Considerăm un numar natural $n \in N$ şi următoarele probleme:
  - Să se construiască mulţimea divizorilor lui $n$.
  - Să se construiască mulţimea multiplilor lui $n$.
- Presupunem că fiecare instanţa este descrisă ca o inşiruire într-un cod binar. Prin mărimea (dimensiunea) unei instanţe a unei probleme vom întelege numărul de biţi necesari pentru a reprezenta instanţa.

# Complexitatea algoritmilor

- Un algoritm este o listă de instrucţiuni care rezolvă orice instanţă a unei probleme într-un număr finit de paşi.
- O instanţă a unei probleme este un set de date care este obţinut când toţi parametrii care definesc problema sunt fixaţi.
- Output-ul unui algoritm este yes în cazul în care obţinem o soluţie şi no altfel.
- Nu orice problema poate fi rezolvată algoritmic. Considerăm un numar natural $n \in N$ şi următoarele probleme:
  - Să se construiască mulţimea divizorilor lui $n$.
  - Să se construiască mulţimea multiplilor lui $n$.
- Presupunem că fiecare instanţa este descrisă ca o inşiruire într-un cod binar. Prin mărimea (dimensiunea) unei instanţe a unei probleme vom întelege numărul de biţi necesari pentru a reprezenta instanţa.

# Complexitatea algoritmilor

- Un algoritm este o listă de instrucţiuni care rezolvă orice instanţă a unei probleme într-un număr finit de paşi.
- O instanţă a unei probleme este un set de date care este obţinut când toţi parametrii care definesc problema sunt fixaţi.
- Output-ul unui algoritm este yes în cazul în care obţinem o soluţie şi no altfel.
- Nu orice problema poate fi rezolvată algoritmic. Considerăm un numar natural $n \in N$ şi următoarele probleme:
  - Să se construiască mulţimea divizorilor lui $n$,
  - Să se construiască mulţimea multiplilor lui $n$.
- Presupunem că fiecare instanţa este descrisă ca o inşiruire într-un cod binar. Prin mărimea (dimensiunea) unei instanţe a unei probleme vom întelege numărul de biţi necesari pentru a reprezenta instanţa.

# Complexitatea algoritmilor

- Un algoritm este o listă de instrucţiuni care rezolvă orice instanţă a unei probleme într-un număr finit de paşi.
- O instanţă a unei probleme este un set de date care este obţinut când toţi parametrii care definesc problema sunt fixaţi.
- Output-ul unui algoritm este yes în cazul în care obţinem o soluţie şi no altfel.
- Nu orice problema poate fi rezolvată algoritmic. Considerăm un numar natural $n \in N$ şi următoarele probleme:
  - Să se construiască mulţimea divizorilor lui $n$,
  - Să se construiască mulţimea multiplilor lui $n$.
- Presupunem că fiecare instanţa este descrisă ca o inşiruire într-un cod binar. Prin mărimea (dimensiunea) unei instanţe a unei probleme vom întelege numărul de biţi necesari pentru a reprezenta instanţa.

# Complexitatea algoritmilor

- Un algoritm este o listă de instrucţiuni care rezolvă orice instanţă a unei probleme într-un număr finit de paşi.
- O instanţă a unei probleme este un set de date care este obţinut când toţi parametrii care definesc problema sunt fixaţi.
- Output-ul unui algoritm este yes în cazul în care obţinem o soluţie şi no altfel.
- Nu orice problema poate fi rezolvată algoritmic. Considerăm un numar natural $n \in N$ şi următoarele probleme:
  - Să se construiască mulţimea divizorilor lui $n$,
  - Să se construiască mulţimea multiplilor lui $n$.
- Presupunem că fiecare instanţa este descrisă ca o inşiruire într-un cod binar. Prin mărimea (dimensiunea) unei instanţe a unei probleme vom întelege numărul de biţi necesari pentru a reprezenta instanţa.

# Complexitatea algoritmilor

- Un algoritm este o listă de instrucţiuni care rezolvă orice instanţă a unei probleme într-un număr finit de paşi.
- O instanţă a unei probleme este un set de date care este obţinut când toţi parametrii care definesc problema sunt fixaţi.
- Output-ul unui algoritm este yes în cazul în care obţinem o soluţie şi no altfel.
- Nu orice problema poate fi rezolvată algoritmic. Considerăm un numar natural $n \in N$ şi următoarele probleme:
  - Să se construiască mulţimea divizorilor lui $n$,
  - Să se construiască mulţimea multiplilor lui $n$.
- Presupunem că fiecare instanţa este descrisă ca o inşiruire într-un cod binar. Prin mărimea (dimensiunea) unei instanţe a unei probleme vom întelege numărul de biţi necesari pentru a reprezenta instanţa.

# Complexitatea algoritmilor

- O întrebare pe care ne-o putem pune imediat este: cât de bun este un algoritm?
- Nu se poate scrie un altul mai bun (care să rezolve aceeaşi problemă, fireşte)?
- Pentru a putea răspunde, trebuie să cădem de acord asupra unei metode prin care măsurăm calităţile unui algoritm; putem măsura timpul lui de execuţie pentru o anumită problemă, sau cantitatea de memorie folosită, sau numărul de instrucţiuni care descriu programul, sau poate o altă măsură.
- Complexitatea unui algoritm se poate defini ca fiind numărul de operaţii în funcţie de dimensiunea datelor de intrare (input-ul).
- Complexitatea unui algoritm se referă la cantitatea de resurse consumate la execuţie - adică timp de procesare şi spaţiu de memorie.

# Complexitatea algoritmilor

- O întrebare pe care ne-o putem pune imediat este: cât de bun este un algoritm?
- Nu se poate scrie un altul mai bun (care să rezolve aceeaşi problemă, fireşte)?
- Pentru a putea răspunde, trebuie să cădem de acord asupra unei metode prin care măsurăm calităţile unui algoritm; putem măsura timpul lui de execuţie pentru o anumită problemă, sau cantitatea de memorie folosită, sau numărul de instrucţiuni care descriu programul, sau poate o altă măsură.
- Complexitatea unui algoritm se poate defini ca fiind numărul de operaţii în funcţie de dimensiunea datelor de intrare (input-ul).
- Complexitatea unui algoritm se referă la cantitatea de resurse consumate la execuţie - adică timp de procesare şi spaţiu de memorie.

# Complexitatea algoritmilor

- O întrebare pe care ne-o putem pune imediat este: cât de bun este un algoritm?
- Nu se poate scrie un altul mai bun (care să rezolve aceeaşi problemă, fireşte)?
- Pentru a putea răspunde, trebuie să cădem de acord asupra unei metode prin care măsurăm calităţile unui algoritm; putem măsura timpul lui de execuţie pentru o anumită problemă, sau cantitatea de memorie folosită, sau numărul de instrucţiuni care descriu programul, sau poate o altă măsură.
- Complexitatea unui algoritm se poate defini ca fiind numărul de operaţii în funcţie de dimensiunea datelor de intrare (input-ul).
- Complexitatea unui algoritm se referă la cantitatea de resurse consumate la execuţie - adică timp de procesare şi spaţiu de memorie.

# Complexitatea algoritmilor

- O întrebare pe care ne-o putem pune imediat este: cât de bun este un algoritm?
- Nu se poate scrie un altul mai bun (care să rezolve aceeaşi problemă, fireşte)?
- Pentru a putea răspunde, trebuie să cădem de acord asupra unei metode prin care măsurăm calităţile unui algoritm; putem măsura timpul lui de execuţie pentru o anumită problemă, sau cantitatea de memorie folosită, sau numărul de instrucţiuni care descriu programul, sau poate o altă măsură.
- Complexitatea unui algoritm se poate defini ca fiind numărul de operaţii în funcţie de dimensiunea datelor de intrare (input-ul).
- Complexitatea unui algoritm se referă la cantitatea de resurse consumate la execuţie - adică timp de procesare şi spaţiu de memorie.

# Complexitatea algoritmilor

- O întrebare pe care ne-o putem pune imediat este: cât de bun este un algoritm?
- Nu se poate scrie un altul mai bun (care să rezolve aceeaşi problemă, fireşte)?
- Pentru a putea răspunde, trebuie să cădem de acord asupra unei metode prin care măsurăm calităţile unui algoritm; putem măsura timpul lui de execuţie pentru o anumită problemă, sau cantitatea de memorie folosită, sau numărul de instrucţiuni care descriu programul, sau poate o altă măsură.
- Complexitatea unui algoritm se poate defini ca fiind numărul de operaţii în funcţie de dimensiunea datelor de intrare (input-ul).
- Complexitatea unui algoritm se referă la cantitatea de resurse consumate la execuţie - adică timp de procesare şi spaţiu de memorie.

# Complexitatea algoritmilor

- Deci analiza complexităţii are ca sop stabilirea resurselor necesare pentru execuţia algoritmului pe o maşina de calcul. Presupunem că fiecare algoritm este executat pe aceeaşi maşina de calcul, aşa numita maşină Turing.
- Prin resurse intelegem:
  - spaţiul de memorie necesar pentru stocarea datelor care prelucrează algoritmul,
  - timpul necesar pentru execuţia tuturor prelucrărilor specificate în algoritm.
- Deci volumul resurselor depinde de dimensiunea problemei.

# Complexitatea algoritmilor

- Deci analiza complexităţii are ca sop stabilirea resurselor necesare pentru execuţia algoritmului pe o maşina de calcul. Presupunem că fiecare algoritm este executat pe aceeaşi maşina de calcul, aşa numita maşină Turing.
- Prin resurse intelegem:
  - spaţiul de memorie necesar pentru stocarea datelor care prelucrează algoritmul,
  - timpul necesar pentru execuţia tuturor prelucrărilor specificate în algoritm.
- Deci volumul resurselor depinde de dimensiunea problemei.

# Complexitatea algoritmilor

- Deci analiza complexităţii are ca sop stabilirea resurselor necesare pentru execuţia algoritmului pe o maşina de calcul. Presupunem că fiecare algoritm este executat pe aceeaşi maşina de calcul, aşa numita maşină Turing.
- Prin resurse intelegem:
  - spaţiul de memorie necesar pentru stocarea datelor care prelucrează algoritmul,
  - timpul necesar pentru execuţia tuturor prelucrărilor specificate în algoritm.
- Deci volumul resurselor depinde de dimensiunea problemei.

# Complexitatea algoritmilor

- Deci analiza complexităţii are ca sop stabilirea resurselor necesare pentru execuţia algoritmului pe o maşina de calcul. Presupunem că fiecare algoritm este executat pe aceeaşi maşina de calcul, aşa numita maşină Turing.
- Prin resurse intelegem:
  - spaţiul de memorie necesar pentru stocarea datelor care prelucrează algoritmul,
  - timpul necesar pentru execuţia tuturor prelucrărilor specificate în algoritm.
- Deci volumul resurselor depinde de dimensiunea problemei.

# Complexitatea algoritmilor

- Deci analiza complexităţii are ca sop stabilirea resurselor necesare pentru execuţia algoritmului pe o maşina de calcul. Presupunem că fiecare algoritm este executat pe aceeaşi maşina de calcul, aşa numita maşină Turing.
- Prin resurse intelegem:
  - spaţiul de memorie necesar pentru stocarea datelor care prelucrează algoritmul,
  - timpul necesar pentru execuţia tuturor prelucrărilor specificate în algoritm.
- Deci volumul resurselor depinde de dimensiunea problemei.

# Timp de execuţie

- Cel mai interesant atribut al performanţei a fost judecat a fi timpul de execuţie al unui algoritm.
- Timpul este apoi asimilat cu numărul de operaţii elementare pe care le efectuează un algoritm pentru a rezolva o problema.
- Pentru a estima timpul de execuţie al unui algoritm trebuie stabilit un model de calcul. Vom presupune că:
  - prelucrările se efectuează în mod secvenţial,
  - operaţiile elementare (cele aritmetice: $+, -, \cdot, :$, comparaţiile şi cele logice (negaţia, conjuncţia şi disjuncţia) sunt efectuate în timp constant indiferent de valoarea operanzilor.
- Scopul calculului timpului de execuţie este de a permite compararea algoritmilor.
- Vom nota cu $T(n)$ timpul de execuţie al unui algoritm destinat rezolvării unei probleme de dimensiune $n$.

# Timp de execuţie

- Cel mai interesant atribut al performanţei a fost judecat a fi timpul de execuţie al unui algoritm.
- Timpul este apoi asimilat cu numărul de operaţii elementare pe care le efectuează un algoritm pentru a rezolva o problema.
- Pentru a estima timpul de execuţie al unui algoritm trebuie stabilit un model de calcul. Vom presupune că:
  - prelucrările se efectuează în mod secvenţial,
  - operaţiile elementare (cele aritmetice: $+, -, \cdot, :$, comparaţiile şi cele logice (negaţia, conjuncţia şi disjuncţia) sunt efectuate în timp constant indiferent de valoarea operanzilor.
- Scopul calculului timpului de execuţie este de a permite compararea algoritmilor.
- Vom nota cu $T(n)$ timpul de execuţie al unui algoritm destinat rezolvării unei probleme de dimensiune $n$.

# Timp de execuţie

- Cel mai interesant atribut al performanţei a fost judecat a fi timpul de execuţie al unui algoritm.
- Timpul este apoi asimilat cu numărul de operaţii elementare pe care le efectuează un algoritm pentru a rezolva o problema.
- Pentru a estima timpul de execuţie al unui algoritm trebuie stabilit un model de calcul. Vom presupune că:
  - prelucrările se efectuează în mod secvenţial,
  - operaţiile elementare (cele aritmetice: $+, -, \cdot, :$, comparaţiile şi cele logice (negaţia, conjuncţia şi disjuncţia) sunt efectuate în timp constant indiferent de valoarea operanzilor.
- Scopul calculului timpului de execuţie este de a permite compararea algoritmilor.
- Vom nota cu $T(n)$ timpul de execuţie al unui algoritm destinat rezolvării unei probleme de dimensiune $n$.

# Timp de execuţie

- Cel mai interesant atribut al performanţei a fost judecat a fi timpul de execuţie al unui algoritm.
- Timpul este apoi asimilat cu numărul de operaţii elementare pe care le efectuează un algoritm pentru a rezolva o problema.
- Pentru a estima timpul de execuţie al unui algoritm trebuie stabilit un model de calcul. Vom presupune că:
  - prelucrările se efectuează în mod secvenţial,
  - operaţiile elementare (cele aritmetice: $+, -, \cdot, :$, comparaţiile şi cele logice (negaţia, conjuncţia şi disjuncţia) sunt efectuate în timp constant indiferent de valoarea operanzilor.
- Scopul calculului timpului de execuţie este de a permite compararea algoritmilor.
- Vom nota cu $T(n)$ timpul de execuţie al unui algoritm destinat rezolvării unei probleme de dimensiune $n$.

# Timp de execuţie

- Cel mai interesant atribut al performanţei a fost judecat a fi timpul de execuţie al unui algoritm.
- Timpul este apoi asimilat cu numărul de operaţii elementare pe care le efectuează un algoritm pentru a rezolva o problema.
- Pentru a estima timpul de execuţie al unui algoritm trebuie stabilit un model de calcul. Vom presupune că:
  - prelucrările se efectuează în mod secvenţial,
  - operaţiile elementare (cele aritmetice: $+, -, \cdot, :$, comparaţiile şi cele logice (negaţia, conjuncţia şi disjuncţia) sunt efectuate în timp constant indiferent de valoarea operanzilor.
- Scopul calculului timpului de execuţie este de a permite compararea algoritmilor.
- Vom nota cu $T(n)$ timpul de execuţie al unui algoritm destinat rezolvării unei probleme de dimensiune $n$.

# Timp de execuţie

- Cel mai interesant atribut al performanţei a fost judecat a fi timpul de execuţie al unui algoritm.
- Timpul este apoi asimilat cu numărul de operaţii elementare pe care le efectuează un algoritm pentru a rezolva o problema.
- Pentru a estima timpul de execuţie al unui algoritm trebuie stabilit un model de calcul. Vom presupune că:
  - prelucrările se efectuează în mod secvenţial,
  - operaţiile elementare (cele aritmetice: $+, -, \cdot, :$, comparaţiile şi cele logice (negaţia, conjuncţia şi disjuncţia) sunt efectuate în timp constant indiferent de valoarea operanzilor.
- Scopul calculului timpului de execuţie este de a permite compararea algoritmilor.
- Vom nota cu $T(n)$ timpul de execuţie al unui algoritm destinat rezolvării unei probleme de dimensiune $n$.

# Timp de execuţie

- Cel mai interesant atribut al performanţei a fost judecat a fi timpul de execuţie al unui algoritm.
- Timpul este apoi asimilat cu numărul de operaţii elementare pe care le efectuează un algoritm pentru a rezolva o problema.
- Pentru a estima timpul de execuţie al unui algoritm trebuie stabilit un model de calcul. Vom presupune că:
  - prelucrările se efectuează în mod secvenţial,
  - operaţiile elementare (cele aritmetice: $+, -, \cdot, :$, comparaţiile şi cele logice (negaţia, conjuncţia şi disjuncţia) sunt efectuate în timp constant indiferent de valoarea operanzilor.
- Scopul calculului timpului de execuţie este de a permite compararea algoritmilor.
- Vom nota cu $T(n)$ timpul de execuţie al unui algoritm destinat rezolvării unei probleme de dimensiune $n$.

# Timp de execuţie

Exemplul 1. Să se calculeze suma primelor n numere.

- Utilizarea unei formule de calcul:

$$S_n = \frac{n(n+1)}{2}$$

  În acest caz complexitatea algoritmului este constantă.

- Utilizarea unui algoritm.

# Timp de execuţie

Exemplul 1. Să se calculeze suma primelor n numere.

- Utilizarea unei formule de calcul:

$$S_n = \frac{n(n+1)}{2}$$

  În acest caz complexitatea algoritmului este constantă.

- Utilizarea unui algoritm.

# Timp de execuţie

| | *suma(n)* | *cost* | *Nr. repetări* |
|---|---|---|---|
| 1. | S:=0 | $c_1$ | 1 |
| 2. | i:=1 | $c_2$ | 1 |
| 3. | **while** i<= n **do** | $c_3$ | n+1 |
| 4. | S:=S+i | $c_4$ | n |
| 5. | i:=i+1 | $c_5$ | n |
| 6. | **end while** | | |
| | Return S | | |

$$T(n) = n(c_3 + c_4 + c_5) + c_1 + c_2 + c_3 = k_1 \cdot n + k_2$$

Complexitatea acestui algoritm iterativ (complexitate liniară) este $O(n)$.

# Timp de execuţie

Exemplul 2. Considerăm problema sortării unui tablou $s[1..n]$ prin metoda interschimbării.

**begin**
  **for** $i = 1$ to $n - 1$ **do**
    **for** $j = i + 1$ to $n$ **do**
      **if** $(s[j] < s[i])$ **then**
        $s[i] \leftrightarrow s[j]$
      **end if**
**end**

Notăm cu $T(n)$ numărul de paşi executaţi de algoritm.

Operaţiile de bază sunt: comparaţia $s[i]$ cu $s[j]$ şi schimbarea $s[i]$ cu $s[j]$.

# Timp de execuţie

- Bucla exterioară se execută de $n - 1$ ori.
- La primul pas al buclei exterioare se vor executa $n - 1$ paşi în bucla interioară.
- La al doilea pas al buclei exterioare vor fi $n - 2$ paşi în bucla interioară. Apoi $n - 3$, $n - 4$ paşi, etc.
- Deci timpul de execuţie al algoritmului este:

$$T(n) = (n-1) + (n-2) + (n-3) + ... + 2 + 1 = \frac{n(n-1)}{2}$$

- Complexitatea acestui algoritm este pătratică $O(n^2)$.

# Timp de execuţie

- Bucla exterioară se execută de $n-1$ ori.
- La primul pas al buclei exterioare se vor executa $n-1$ paşi în bucla interioară.
- La al doilea pas al buclei exterioare vor fi $n-2$ paşi în bucla interioară. Apoi $n-3$, $n-4$ paşi, etc.
- Deci timpul de execuţie al algoritmului este:

$$T(n) = (n-1) + (n-2) + (n-3) + ... + 2 + 1 = \frac{n(n-1)}{2}$$

- Complexitatea acestui algoritm este pătratică $O(n^2)$.

# Timp de execuţie

- Bucla exterioară se execută de $n-1$ ori.
- La primul pas al buclei exterioare se vor executa $n-1$ paşi în bucla interioară.
- La al doilea pas al buclei exterioare vor fi $n-2$ paşi în bucla interioară. Apoi $n-3$, $n-4$ paşi, etc.
- Deci timpul de execuţie al algoritmului este:

$$T(n) = (n-1) + (n-2) + (n-3) + ... + 2 + 1 = \frac{n(n-1)}{2}$$

- Complexitatea acestui algoritm este pătratică $O(n^2)$.

# Timp de execuţie

- Bucla exterioară se execută de $n-1$ ori.
- La primul pas al buclei exterioare se vor executa $n-1$ paşi în bucla interioară.
- La al doilea pas al buclei exterioare vor fi $n-2$ paşi în bucla interioară. Apoi $n-3$, $n-4$ paşi, etc.
- Deci timpul de execuţie al algoritmului este:

$$T(n) = (n-1) + (n-2) + (n-3) + ... + 2 + 1 = \frac{n(n-1)}{2}$$

- Complexitatea acestui algoritm este pătratică $O(n^2)$.

# Timp de execuţie

- Bucla exterioară se execută de $n-1$ ori.
- La primul pas al buclei exterioare se vor executa $n-1$ paşi în bucla interioară.
- La al doilea pas al buclei exterioare vor fi $n-2$ paşi în bucla interioară. Apoi $n-3$, $n-4$ paşi, etc.
- Deci timpul de execuţie al algoritmului este:

$$T(n) = (n-1) + (n-2) + (n-3) + ... + 2 + 1 = \frac{n(n-1)}{2}$$

- Complexitatea acestui algoritm este pătratică $O(n^2)$.

# Timp de executie

Exemplul 3. Considerăm problema determinării produsului a două matrici: $A$ de dimensiune $m \times n$ şi $B$ de dimensiune $n \times p$.

| | | *Nr. repetări* |
|---|---|---|
| 1. | **for** $i = 1$ to $m$ **do** | 1 |
| 2. |     **for** $j = 1$ to $p$ **do** | m |
| 3. |         $c[i,j] \leftarrow 0$ | m x p |
| 4. |         **for** $k = 1$ to $n$ **do** | m x p |
| 5. |             $c[i,j] \leftarrow c[i,j] + a[i,k] \times b[k,j]$ | m x p x n |
| 6. |         **end for** | |
| 7. |     **end for** | |
| 8. | **end for** | |

- Timpul de execuţie este: $T(m,n,p) = mnp + 2mp + m + 1$.
- Operaţia dominantă (operaţia care contribuie cel mai mult la timpul de execuţie a algoritmului) este operaţia de înmulţire. Complexitatea algoritmului este $O(mnp)$.

# Timp de executie

Exemplul 3. Considerăm problema determinării produsului a două matrici: $A$ de dimensiune $m \times n$ şi $B$ de dimensiune $n \times p$.

| | *Nr. repetări* |
|---|---|
| **for** $i = 1$ to $m$ **do** | 1 |
|     **for** $j = 1$ to $p$ **do** | m |
|         $c[i,j] \leftarrow 0$ | m x p |
|         **for** $k = 1$ to $n$ **do** | m x p |
|             $c[i,j] \leftarrow c[i,j] + a[i,k] \times b[k,j]$ | m x p x n |
|         **end for** | |
|     **end for** | |
| **end for** | |

- Timpul de execuţie este: $T(m,n,p) = mnp + 2mp + m + 1$.
- Operaţia dominantă (operaţia care contribuie cel mai mult la timpul de execuţie a algoritmului) este operaţia de înmulţire. Complexitatea algoritmului este $O(mnp)$.

# Timp de execuţie

Exemplul 4. Considerăm problema determinării valorii minime într-un tablou $x[1..n]$.

| | *minim x[1..n]* | *Nr. repetări* |
|---|---|---|
| 1. | x[1]:=m | 1 |
| 2. | **for** i=2 to n **do** | 1 |
| 3. |     **if** m > x[i] **then** | n-1 |
| 4. |         x[i]:=m | $\tau(n)$ |
| 5. |     **end if** | |
| 6. | **end for** | |
| | Return m | |

- Spre deosebire de exemplele anterioare, timpul de execuţie nu poate fi calculat explicit, deoarece numărul de repetări ale prelucrării numerotate cu 4 depinde de valorile aflate în tablou.

# Timp de execuţie

- Dacă cea mai mică valoare din tablou se află chiar pe prima poziţie atunci prelucrarea 4 nu se efectuează niciodată, iar $\tau(n) = 0$. Acest caz este considerat cazul cel mai favorabil.
- Dacă, în schimb, elementele tabloului sunt în ordine strict descrescătoare atunci prelucrarea 4 se efectuează la fiecare iteraţie, adică $\tau(n) = n - 1$. Acesta este cazul cel mai defavorabil.
- Timpul de execuţie în acest caz poate fi încadrat între două limite:

$$n + 1 \leq T(n) \leq 2n$$

- În aprecierea şi compararea algoritmilor ne interesează în special cel mai defavorabil caz deoarece furnizează cel mai mare timp de execuţie relativ la orice date de intrare de dimensiune fixă.
- Exerciţiu. Să se determine timpul de execuţie în cazul problemei căutarii unei valori $v$ într-un tablou.

# Timp de execuţie

- Dacă cea mai mică valoare din tablou se află chiar pe prima poziţie atunci prelucrarea 4 nu se efectuează niciodată, iar $\tau(n) = 0$. Acest caz este considerat cazul cel mai favorabil.
- Dacă, în schimb, elementele tabloului sunt în ordine strict descrescătoare atunci prelucrarea 4 se efectuează la fiecare iteraţie, adică $\tau(n) = n - 1$. Acesta este cazul cel mai defavorabil.
- Timpul de execuţie în acest caz poate fi încadrat între două limite:

$$n + 1 \leq T(n) \leq 2n$$

- În aprecierea şi compararea algoritmilor ne interesează în special cel mai defavorabil caz deoarece furnizează cel mai mare timp de execuţie relativ la orice date de intrare de dimensiune fixă.
- Exerciţiu. Să se determine timpul de execuţie în cazul problemei căutarii unei valori $v$ într-un tablou.

# Timp de execuţie

- Dacă cea mai mică valoare din tablou se află chiar pe prima poziţie atunci prelucrarea 4 nu se efectuează niciodată, iar $\tau(n) = 0$. Acest caz este considerat cazul cel mai favorabil.
- Dacă, în schimb, elementele tabloului sunt în ordine strict descrescătoare atunci prelucrarea 4 se efectuează la fiecare iteraţie, adică $\tau(n) = n - 1$. Acesta este cazul cel mai defavorabil.
- Timpul de execuţie în acest caz poate fi încadrat între două limite:

$$n + 1 \leq T(n) \leq 2n$$

- În aprecierea şi compararea algoritmilor ne interesează în special cel mai defavorabil caz deoarece furnizează cel mai mare timp de execuţie relativ la orice date de intrare de dimensiune fixă.
- Exerciţiu. Să se determine timpul de execuţie în cazul problemei căutarii unei valori $v$ într-un tablou.

# Timp de execuţie

- Dacă cea mai mică valoare din tablou se află chiar pe prima poziţie atunci prelucrarea 4 nu se efectuează niciodată, iar $\tau(n) = 0$. Acest caz este considerat cazul cel mai favorabil.
- Dacă, în schimb, elementele tabloului sunt în ordine strict descrescătoare atunci prelucrarea 4 se efectuează la fiecare iteraţie, adică $\tau(n) = n - 1$. Acesta este cazul cel mai defavorabil.
- Timpul de execuţie în acest caz poate fi încadrat între două limite:

$$n + 1 \leq T(n) \leq 2n$$

- În aprecierea şi compararea algoritmilor ne interesează în special cel mai defavorabil caz deoarece furnizează cel mai mare timp de execuţie relativ la orice date de intrare de dimensiune fixă.
- Exerciţiu. Să se determine timpul de execuţie în cazul problemei căutarii unei valori $v$ într-un tablou.

# Timp de execuţie

- Dacă cea mai mică valoare din tablou se află chiar pe prima poziţie atunci prelucrarea 4 nu se efectuează niciodată, iar $\tau(n) = 0$. Acest caz este considerat cazul cel mai favorabil.
- Dacă, în schimb, elementele tabloului sunt în ordine strict descrescătoare atunci prelucrarea 4 se efectuează la fiecare iteraţie, adică $\tau(n) = n - 1$. Acesta este cazul cel mai defavorabil.
- Timpul de execuţie în acest caz poate fi încadrat între două limite:

$$n + 1 \leq T(n) \leq 2n$$

- În aprecierea şi compararea algoritmilor ne interesează în special cel mai defavorabil caz deoarece furnizează cel mai mare timp de execuţie relativ la orice date de intrare de dimensiune fixă.
- Exerciţiu. Să se determine timpul de execuţie în cazul problemei căutarii unei valori $v$ într-un tablou.

# Timp mediu de execuţie

- O altă măsură a complexităţii algoritmilor o reprezintă timpul mediu de execuţie.
- Timpul mediu de execuţie reprezintă o valoare medie a timpilor de execuţie calculată în raport cu distribuţia de probabilitate corespunzătoare spaţiului datelor de intrare.
- Dacă $\nu(n)$ reprezintă numărul variantelor posibile, $P_k$ probabilitatea de apariţie a cazului $k$, iar $T_k(n)$ este timpul de execuţie corespunzător cazului $k$ atunci timpul mediu de execuţie este dat de relaţia:

$$T_m(n) = \sum_{k=1}^{\nu(n)} T_k(n) \cdot P_k$$

# Timp mediu de execuţie

- O altă măsură a complexităţii algoritmilor o reprezintă timpul mediu de execuţie.
- Timpul mediu de execuţie reprezintă o valoare medie a timpilor de execuţie calculată în raport cu distribuţia de probabilitate corespunzătoare spaţiului datelor de intrare.
- Dacă $\nu(n)$ reprezintă numărul variantelor posibile, $P_k$ probabilitatea de apariţie a cazului $k$, iar $T_k(n)$ este timpul de execuţie corespunzător cazului $k$ atunci timpul mediu de execuţie este dat de relaţia:

$$T_m(n) = \sum_{k=1}^{\nu(n)} T_k(n) \cdot P_k$$

# Timp mediu de execuţie

- O altă măsură a complexităţii algoritmilor o reprezintă timpul mediu de execuţie.
- Timpul mediu de execuţie reprezintă o valoare medie a timpilor de execuţie calculată în raport cu distribuţia de probabilitate corespunzătoare spaţiului datelor de intrare.
- Dacă $\nu(n)$ reprezintă numărul variantelor posibile, $P_k$ probabilitatea de apariţie a cazului $k$, iar $T_k(n)$ este timpul de execuţie corespunzător cazului $k$ atunci timpul mediu de execuţie este dat de relaţia:

$$T_m(n) = \sum_{k=1}^{\nu(n)} T_k(n) \cdot P_k$$

# Timp mediu de execuţie

- În cazul în care toate cazurile sunt echiprobabile: $P_k = \frac{1}{\nu(n)}$, atunci:

$$T_m(n) = \sum_{k=1}^{n} \frac{T_k(n)}{\nu(n)}$$

- Aceasta tehnică este mult mai rar folosită, pentru că:
  - Este greu de argumentat o distribuţie de probabilitate pentru un set de date de intrare (practic distribuţia afirmă ce şansă are fiecare instanţă de a fi întâlnită când se rulează algoritmul). De exemplu, pentru un algoritm pe grafuri, care este probabilitatea de a primi un arbore?
  - În general este mult mai greu de evaluat analitic formula obţinută decat in cazul folosirii maximumului.

# Timp mediu de execuţie

- În cazul în care toate cazurile sunt echiprobabile: $P_k = \frac{1}{\nu(n)}$, atunci:

$$T_m(n) = \sum_{k=1}^{n} \frac{T_k(n)}{\nu(n)}$$

- Aceasta tehnică este mult mai rar folosită, pentru că:
  - Este greu de argumentat o distribuţie de probabilitate pentru un set de date de intrare (practic distribuţia afirmă ce şansă are fiecare instanţă de a fi întâlnită când se rulează algoritmul). De exemplu, pentru un algoritm pe grafuri, care este probabilitatea de a primi un arbore?
  - În general este mult mai greu de evaluat analitic formula obţinută decat in cazul folosirii maximumului.

# Timp mediu de execuţie

- În cazul în care toate cazurile sunt echiprobabile: $P_k = \frac{1}{\nu(n)}$, atunci:

$$T_m(n) = \sum_{k=1}^{n} \frac{T_k(n)}{\nu(n)}$$

- Aceasta tehnică este mult mai rar folosită, pentru că:
  - Este greu de argumentat o distribuţie de probabilitate pentru un set de date de intrare (practic distribuţia afirmă ce şansă are fiecare instanţă de a fi întâlnită când se rulează algoritmul). De exemplu, pentru un algoritm pe grafuri, care este probabilitatea de a primi un arbore?
  - În general este mult mai greu de evaluat analitic formula obţinută decat in cazul folosirii maximumului.

# Timp mediu de execuţie

- În cazul în care toate cazurile sunt echiprobabile: $P_k = \dfrac{1}{\nu(n)}$, atunci:

$$T_m(n) = \sum_{k=1}^{n} \frac{T_k(n)}{\nu(n)}$$

- Aceasta tehnică este mult mai rar folosită, pentru că:
  - Este greu de argumentat o distribuţie de probabilitate pentru un set de date de intrare (practic distribuţia afirmă ce şansă are fiecare instanţă de a fi întâlnită când se rulează algoritmul). De exemplu, pentru un algoritm pe grafuri, care este probabilitatea de a primi un arbore?
  - În general este mult mai greu de evaluat analitic formula obţinută decat in cazul folosirii maximumului.

# Ordin de creştere

- Pentru a aprecia eficienţa unui algoritm nu este necesară cunoaşterea detaliată a timpului de execuţie. Mai degrabă ne interesează modul în care timpul de execuţie creşte o data cu creşterea dimensiunii problemei.
- O măsură utilă în acest sens este ordinul de creştere.
- Ordinul de creştere este determinat de termenul dominant din expresia timpului de execuţie.
- Când dimensiunea problemei este mare, valoarea termenului dominant depăşeşte semnificativ valorile celorlalţi termeni astfel încât aceştia pot fi neglijaţi.

# Ordin de creştere

- Pentru a aprecia eficienţa unui algoritm nu este necesară cunoaşterea detaliată a timpului de execuţie. Mai degrabă ne interesează modul în care timpul de execuţie creşte o data cu creşterea dimensiunii problemei.
- O măsură utilă în acest sens este ordinul de creştere.
- Ordinul de creştere este determinat de termenul dominant din expresia timpului de execuţie.
- Când dimensiunea problemei este mare, valoarea termenului dominant depăşeşte semnificativ valorile celorlalţi termeni astfel încât aceştia pot fi neglijaţi.

# Ordin de creştere

- Pentru a aprecia eficienţa unui algoritm nu este necesară cunoaşterea detaliată a timpului de execuţie. Mai degrabă ne interesează modul în care timpul de execuţie creşte o data cu creşterea dimensiunii problemei.
- O măsură utilă în acest sens este ordinul de creştere.
- Ordinul de creştere este determinat de termenul dominant din expresia timpului de execuţie.
- Când dimensiunea problemei este mare, valoarea termenului dominant depăşeşte semnificativ valorile celorlalţi termeni astfel încât aceştia pot fi neglijaţi.

# Ordin de creştere

- Pentru a aprecia eficienţa unui algoritm nu este necesară cunoaşterea detaliată a timpului de execuţie. Mai degrabă ne interesează modul în care timpul de execuţie creşte o data cu creşterea dimensiunii problemei.
- O măsură utilă în acest sens este ordinul de creştere.
- Ordinul de creştere este determinat de termenul dominant din expresia timpului de execuţie.
- Când dimensiunea problemei este mare, valoarea termenului dominant depăşeşte semnificativ valorile celorlalţi termeni astfel încât aceştia pot fi neglijaţi.

# Ordin de crestere

- Dacă $T(n) = an + b$, $(a > 0)$, când dimensiunea problemei creşte de $k$ ori şi termenul dominant creşte de acelaşi număr de ori.

$$T(kn) = (ka) \cdot n + b$$

  În acest caz vom spune că avem un ordin liniar de creştere.

- Dacă $T(n) = an^2 + bn + c$, $(a > 0)$, atunci

$$T(kn) = (k^2 a) \cdot n^2 + (kb) \cdot n + c$$

  În acest caz, observăm că termenul dominant creşte de $k^2$ ori şi vom spune că avem un ordin pătratic de creştere.

# Ordin de crestere

- Dacă $T(n) = an + b$, $(a > 0)$, când dimensiunea problemei creşte de $k$ ori şi termenul dominant creşte de acelaşi număr de ori.

$$T(kn) = (ka) \cdot n + b$$

  În acest caz vom spune că avem un ordin liniar de creştere.
- Dacă $T(n) = an^2 + bn + c$, $(a > 0)$, atunci

$$T(kn) = (k^2 a) \cdot n^2 + (kb) \cdot n + c$$

  În acest caz, observăm că termenul dominant creşte de $k^2$ ori şi vom spune că avem un ordin pătratic de creştere.

# Ordin de creştere

- Dacă $T(n) = a \cdot lgn$, ($a > 0$), atunci

$$T(kn) = a \cdot lg(kn) = a \cdot lgn + a \cdot lgk$$

Observăm că în acest caz termenul dominant nu se modifică (timpul de execuţie crescând cu o constantă). Deci un ordin logaritmic de creştere reprezintă o comportare bună.

- Dacă $T(n) = a2^n$, ($a > 0$), atunci

$$T(kn) = a2^{kn} = a(2^n)^k$$

În acest caz, observăm că termenul dominant creşte de exponenţial şi vom spune că avem un ordin exponenţial de creştere.

# Ordin de creştere

- Dacă $T(n) = a \cdot lgn$, ($a > 0$), atunci

$$T(kn) = a \cdot lg(kn) = a \cdot lgn + a \cdot lgk$$

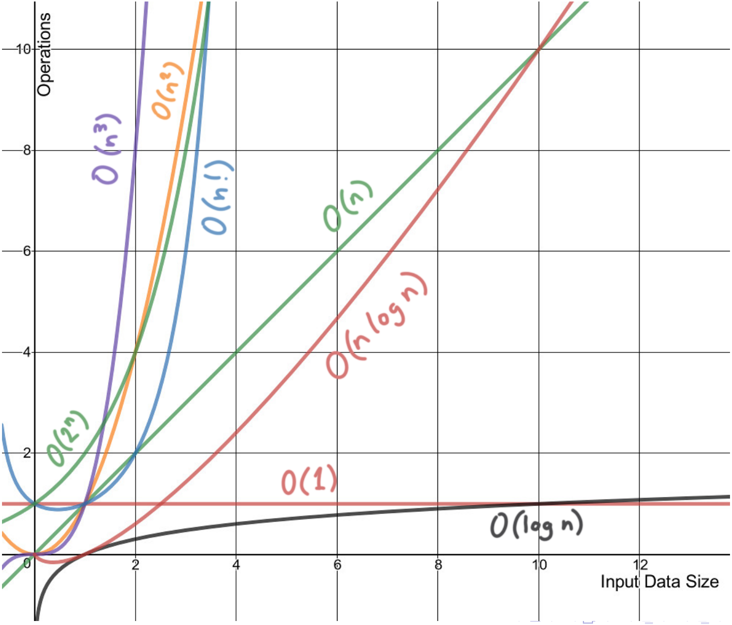
  Observăm că în acest caz termenul dominant nu se modifică (timpul de execuţie crescând cu o constantă). Deci un ordin logaritmic de creştere reprezintă o comportare bună.

- Dacă $T(n) = a2^n$, ($a > 0$), atunci

$$T(kn) = a2^{kn} = a(2^n)^k$$

  În acest caz, observăm că termenul dominant creşte de exponenţial şi vom spune că avem un ordin exponenţial de creştere.

# Ordin de creştere



# Analiză asimptotica

- Întrucât problema eficienţei devine critică pentru probleme de dimensiuni mari se face analiza complexităţii pentru cazul când $n$ este mare (teoretic considerăm $n \to \infty$).
- Acest tip de analiză se numeşte analiză asimptotică.
- În cadrul analizei asimptotice se consideră că un algoritm este mai eficient decât altul dacă ordinul de creştere al timpului de execuţie al primului este mai mic decât al celui de-al doilea.
- Relaţia dintre ordinele de creştere are semnificaţie doar pentru dimensiuni mari ale problemei.

$$T_1(n) = 10n + 10 \quad \text{si} \quad T_2(n) = n^2$$

$T_1(n) > T_2(n)$ pentru $n \leq 10$, deşi ordinul de creştere al lui $T_1$ este evident mai mic decât cel al lui $T_2$.

# Analiză asimptotica

- Întrucât problema eficienţei devine critică pentru probleme de dimensiuni mari se face analiza complexităţii pentru cazul când $n$ este mare (teoretic considerăm $n \to \infty$).
- Acest tip de analiză se numeşte analiză asimptotică.
- În cadrul analizei asimptotice se consideră că un algoritm este mai eficient decât altul dacă ordinul de creştere al timpului de execuţie al primului este mai mic decât al celui de-al doilea.
- Relaţia dintre ordinele de creştere are semnificaţie doar pentru dimensiuni mari ale problemei.

$$T_1(n) = 10n + 10 \quad \text{si} \quad T_2(n) = n^2$$

$T_1(n) > T_2(n)$ pentru $n \leq 10$, deşi ordinul de creştere al lui $T_1$ este evident mai mic decât cel al lui $T_2$.

# Analiză asimptotica

- Întrucât problema eficienţei devine critică pentru probleme de dimensiuni mari se face analiza complexităţii pentru cazul când $n$ este mare (teoretic considerăm $n \to \infty$).
- Acest tip de analiză se numeşte analiză asimptotică.
- În cadrul analizei asimptotice se consideră că un algoritm este mai eficient decât altul dacă ordinul de creştere al timpului de execuţie al primului este mai mic decât al celui de-al doilea.
- Relaţia dintre ordinele de creştere are semnificaţie doar pentru dimensiuni mari ale problemei.

$$T_1(n) = 10n + 10 \quad \text{si} \quad T_2(n) = n^2$$

$T_1(n) > T_2(n)$ pentru $n \leq 10$, deşi ordinul de creştere al lui $T_1$ este evident mai mic decât cel al lui $T_2$.

# Analiză asimptotica

- Întrucât problema eficienţei devine critică pentru probleme de dimensiuni mari se face analiza complexităţii pentru cazul când $n$ este mare (teoretic considerăm $n \to \infty$).
- Acest tip de analiză se numeşte analiză asimptotică.
- În cadrul analizei asimptotice se consideră că un algoritm este mai eficient decât altul dacă ordinul de creştere al timpului de execuţie al primului este mai mic decât al celui de-al doilea.
- Relaţia dintre ordinele de creştere are semnificaţie doar pentru dimensiuni mari ale problemei.

$$T_1(n) = 10n + 10 \quad \text{si} \quad T_2(n) = n^2$$

$T_1(n) > T_2(n)$ pentru $n \leq 10$, deşi ordinul de creştere al lui $T_1$ este evident mai mic decât cel al lui $T_2$.

# Analiză asimptotică

- Pentru a permite gruparea algoritmilor în clase în funcţie de ordinul de creştere a timpului de execuţie s-a introdus o notaţie pentru ordinul de mărime al unei funcţii, introdusde fizicianul L. D. Landau.
- Notatia foloseşte simbolul $O$ pentru a indica mulţimea funcţiilor care cresc mai repede decât o funcţie dată. Această notaţie compară numai funcţii la limită, în creşterea lor spre infinit.
- *Definitie*: Fie dată o funcţie $f : N \to R_+$, astfel încât de la un rang încolo $f(n) > 0$; atunci mulţimea funcţiilor dominate de $f$ se notează cu $O(f)$ şi se defineşte astfel:

$$O(f) = \left\{ g : N \to R_+ | \exists N_0, \exists c < \infty \text{ astfel incat } \frac{g(n)}{f(n)} < c, \forall n > N_0 \right\}.$$

# Analiză asimptotică

- Pentru a permite gruparea algoritmilor în clase în funcţie de ordinul de creştere a timpului de execuţie s-a introdus o notaţie pentru ordinul de mărime al unei funcţii, introdusde fizicianul L. D. Landau.
- Notatia foloseşte simbolul $O$ pentru a indica mulţimea funcţiilor care cresc mai repede decât o funcţie dată. Această notaţie compară numai funcţii la limită, în creşterea lor spre infinit.
- *Definitie*: Fie dată o funcţie $f : N \to R_+$, astfel încât de la un rang încolo $f(n) > 0$; atunci mulţimea funcţiilor dominate de $f$ se notează cu $O(f)$ şi se defineşte astfel:

$$O(f) = \{g : N \to R_+ | \exists N_0, \exists c < \infty \text{ astfel incat } \frac{g(n)}{f(n)} < c, \forall n > N_0\}.$$

# Analiză asimptotică

- Pentru a permite gruparea algoritmilor în clase în funcţie de ordinul de creştere a timpului de execuţie s-a introdus o notaţie pentru ordinul de mărime al unei funcţii, introdusde fizicianul L. D. Landau.
- Notatia foloseşte simbolul $O$ pentru a indica mulţimea funcţiilor care cresc mai repede decât o funcţie dată. Această notaţie compară numai funcţii la limită, în creşterea lor spre infinit.
- *Definitie*: Fie dată o funcţie $f : N \rightarrow R_{+}$, astfel încât de la un rang încolo $f(n) > 0$; atunci mulţimea funcţiilor dominate de $f$ se notează cu $O(f)$ şi se defineşte astfel:

$$O(f) = \{g : N \rightarrow R_{+} | \exists N_0, \, \exists c < \infty \text{ astfel incat } \frac{g(n)}{f(n)} < c, \, \forall n > N_0\}.$$

# Analiză asimptotică

- În cuvinte, o funcţie este în mulţimea $O(f)$ dacă ea "creşte mai încet" decât $f$ la infinit.
- De exemplu, funcţia $g(n) = n$ este în mulţimea $O(n^2)$, pentru că $\frac{g(n)}{n^2} \to 0$.
- În general, un polinom de grad mai mic decât $k$ este în mulţimea $O(n^k)$.
- Ce inseamnă deci că un algoritm "are o complexitate $O(nlogn)$"?
- Înseamnă că pe măsură ce datele de intrare cresc în marime ca $n$, numărul de operaţii facut de algoritm în raport cu măarimea datelor de intrare este mai mic decât $nlogn$ ori.
- Astfel complexitatea asimptotică exprimă concis o limită superioară a timpului de execuţie al unui algoritm.

# Analiză asimptotică

- În cuvinte, o funcţie este în mulţimea $O(f)$ dacă ea "creşte mai încet" decât $f$ la infinit.
- De exemplu, funcţia $g(n) = n$ este în mulţimea $O(n^2)$, pentru că $\frac{g(n)}{n^2} \to 0$.
- În general, un polinom de grad mai mic decât $k$ este în mulţimea $O(n^k)$.
- Ce inseamnă deci că un algoritm "are o complexitate $O(nlogn)$"?
- Înseamnă că pe măsură ce datele de intrare cresc în marime ca $n$, numărul de operaţii facut de algoritm în raport cu măarimea datelor de intrare este mai mic decât $nlogn$ ori.
- Astfel complexitatea asimptotică exprimă concis o limită superioară a timpului de execuţie al unui algoritm.

# Analiză asimptotică

- În cuvinte, o funcţie este în mulţimea $O(f)$ dacă ea "creşte mai încet" decât $f$ la infinit.
- De exemplu, funcţia $g(n) = n$ este în mulţimea $O(n^2)$, pentru că $\frac{g(n)}{n^2} \to 0$.
- În general, un polinom de grad mai mic decât $k$ este în mulţimea $O(n^k)$.
- Ce inseamnă deci că un algoritm "are o complexitate $O(nlogn)$"?
- Înseamnă că pe măsură ce datele de intrare cresc în marime ca $n$, numărul de operaţii facut de algoritm în raport cu măarimea datelor de intrare este mai mic decât $nlogn$ ori.
- Astfel complexitatea asimptotică exprimă concis o limită superioară a timpului de execuţie al unui algoritm.

# Analiză asimptotică

- În cuvinte, o funcţie este în mulţimea $O(f)$ dacă ea "creşte mai încet" decât $f$ la infinit.
- De exemplu, funcţia $g(n) = n$ este în mulţimea $O(n^2)$, pentru că $\frac{g(n)}{n^2} \to 0$.
- În general, un polinom de grad mai mic decât $k$ este în mulţimea $O(n^k)$.
- Ce inseamnă deci că un algoritm "are o complexitate $O(nlogn)$"?
- Înseamnă că pe măsură ce datele de intrare cresc în marime ca $n$, numărul de operaţii facut de algoritm în raport cu măarimea datelor de intrare este mai mic decât $nlogn$ ori.
- Astfel complexitatea asimptotică exprimă concis o limită superioară a timpului de execuţie al unui algoritm.

# Analiză asimptotică

- În cuvinte, o funcţie este în mulţimea $O(f)$ dacă ea "creşte mai încet" decât $f$ la infinit.
- De exemplu, funcţia $g(n) = n$ este în mulţimea $O(n^2)$, pentru că $\frac{g(n)}{n^2} \to 0$.
- În general, un polinom de grad mai mic decât $k$ este în mulţimea $O(n^k)$.
- Ce inseamnă deci că un algoritm "are o complexitate $O(nlogn)$"?
- Înseamnă că pe măsură ce datele de intrare cresc în marime ca $n$, numărul de operaţii facut de algoritm în raport cu măarimea datelor de intrare este mai mic decât $nlogn$ ori.
- Astfel complexitatea asimptotică exprimă concis o limită superioară a timpului de execuţie al unui algoritm.

# Analiză asimptotică

- În cuvinte, o funcţie este în mulţimea $O(f)$ dacă ea "creşte mai încet" decât $f$ la infinit.
- De exemplu, funcţia $g(n) = n$ este în mulţimea $O(n^2)$, pentru că $\frac{g(n)}{n^2} \to 0$.
- În general, un polinom de grad mai mic decât $k$ este în mulţimea $O(n^k)$.
- Ce inseamnă deci că un algoritm "are o complexitate $O(nlogn)$"?
- Înseamnă că pe măsură ce datele de intrare cresc în marime ca $n$, numărul de operaţii facut de algoritm în raport cu măarimea datelor de intrare este mai mic decât $nlogn$ ori.
- Astfel complexitatea asimptotică exprimă concis o limită superioară a timpului de execuţie al unui algoritm.

# Analiză asimptotică

| *Clasa de complexitate* | *Ordin de creştere (cazul cel mai defavorabil)* | *Exemplu* |
|---|---|---|
| logaritmică | $O(lgn)$ | căutarea binară |
| liniară | $O(n)$<br>$O(nlgn)$ | căutarea secvenţială<br>sortare prin interclasare |
| pătratică | $O(n^2)$ | sortarea prin inserţie |
| cubică | $O(n^3)$ | produsul a două matrice de ordinul n |
| exponenţială | $O(2^n)$ | prelucrarea tuturor submult. unei mulţimi cu n elemente |
| factorială | $O(n!)$ | prelucrarea tuturor pemutărilor unei mulţimi cu n elemente |

# For Further Reading I


- Alexander Schrijver, A Course in Combinatorial Optimization, February 1, 2006.
- William J. Cook, William H. Cunningham, William R. Pulleyblank, Alexander Schrijver, Combinatorial Optimization; John Wiley & Sons; 1 edition (November 12, 1997).
- Jon Lee, A First Course in Combinatorial Optimization; Cambridge University Press; 2004.
- Pierluigi Crescenzi, Viggo Kann, Magnús Halldórsson, Marek Karpinski, Gerhard Woeginger, A Compendium of NP Optimization Problems.
- Christos H. Papadimitriou and Kenneth Steiglitz, Combinatorial Optimization : Algorithms and Complexity; Dover Pubns; (paperback, Unabridged edition, July 1998).