

Curs 8:

Tehnica divizării

In cursul anterior am văzut...

... cum se analizează eficiența algoritmilor recursivi

- Se scrie relația de recurență corespunzătoare timpului de execuție
- Se rezolvă relația de recurență folosind tehnica substituției directe sau a celei inverse

... cum se pot rezolva probleme folosind tehnica reducerii

- Descreștere prin reducerea dimensiunii problemei cu o constantă / variabilă
- Descreștere prin împărțirea dimensiunii problemei cu un factor constant/variabil

... uneori tehnica reducerii conduce la algoritmi mai eficienți decât cei obținuți aplicând tehnica forței brute

Structura

- Ideea de bază a tehnicii divizării
- Exemple
- Teorema Master pentru estimarea ordinului de complexitate al algoritmilor bazați pe tehnici reducere/divizare
- Sortare prin interclasare
- Sortare rapidă

Ideea de bază a tehnicii divizării

- Problema curentă este divizată în mai multe subprobleme de același tip dar de dimensiune mai mica
 - Subproblemele componente trebuie să fie **independente** (fiecare dintre aceste subprobleme va fi rezolvată cel mult o dată)
 - Subproblemele trebuie să aibă dimensiuni apropiate
- Subproblemele sunt rezolvate aplicând aceeași strategie (algoritmii proiectați folosind tehnica divizării pot fi descriși ușor în manieră recursivă)
 - Dacă dimensiunea problemei este mai mică decât o anumită valoare (**dimensiune critică**) atunci problema este rezolvată direct, altfel este rezolvată aplicând din nou tehnica divizării (de exemplu, recursiv)
- Dacă este necesar, soluțiile obținute prin rezolvarea subproblemelor sunt combinate

Ideea de bază a tehnicii divizării

Divide&conquer (n)

IF $n \leq n_c$ THEN <rezolvă $P(n)$ direct pt a obține rezultatul r >

ELSE

<descompune $P(n)$ in $P(n_1), \dots, P(n_k)$ >

FOR $i \leftarrow 1, k$ DO

$r_i \leftarrow$ **Divide&conquer**(n_i) // rezolvă subproblema $P(n_i)$

ENDFOR

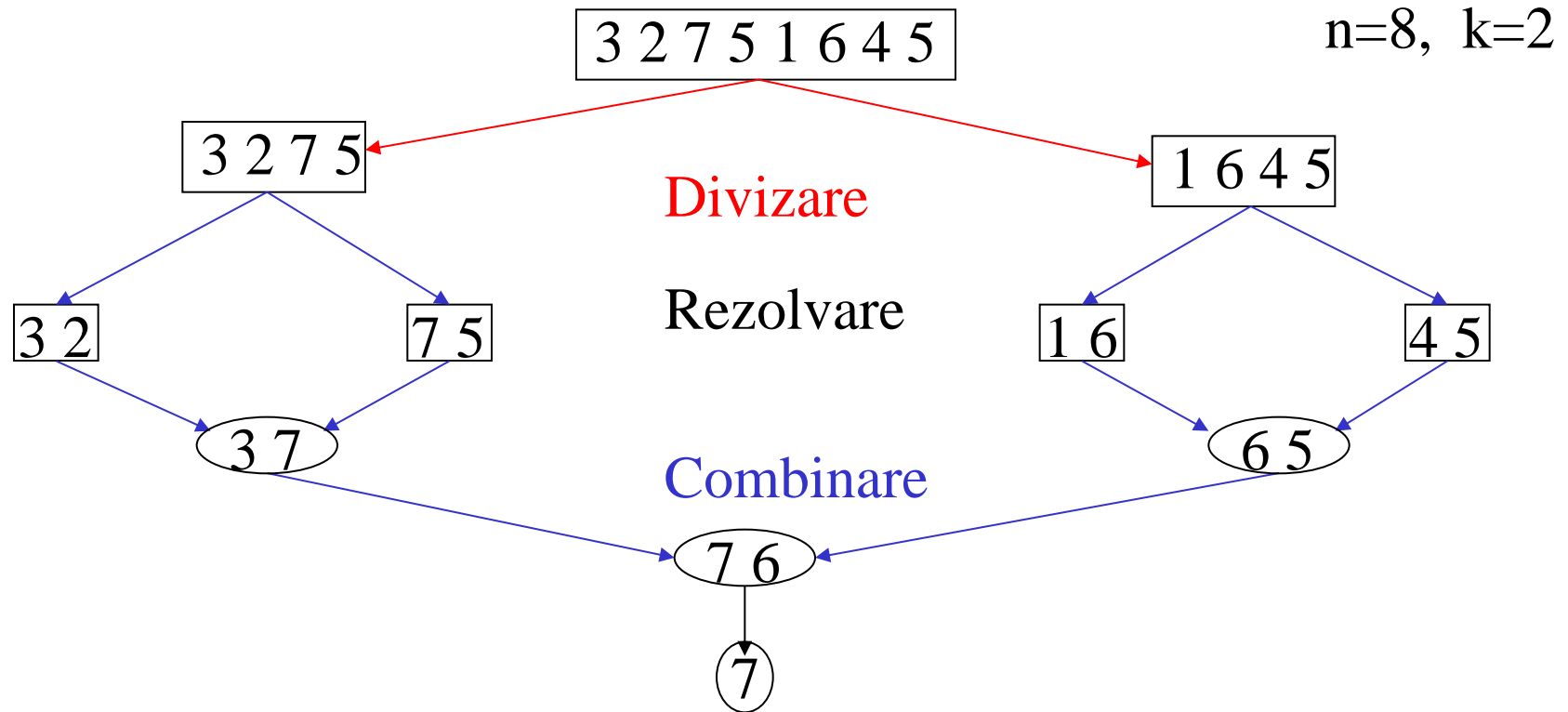
$r \leftarrow$ combinare(r_1, \dots, r_k)

ENDIF

RETURN r

Exemplu 1

Calculul maximului unui tablou $x[1..n]$



Exemplu 1

Algoritm:

```
maxim(x[s..d])
IF s=d then RETURN x[s]
ELSE
  m ← (s+d) DIV 2 //divizare
  max1 ← maxim(x[s..m]) // rezolvare
  max2 ← maxim(x[m+1..d])
  if max1 > max2 // combinare
    THEN RETURN max1
    ELSE RETURN max2
  ENDIF
ENDIF
```

Analiza eficienței

Dimensiunea pb: n

Operație dominantă: comparația

Relație recurență:

$$T(n) = \begin{cases} 0, & n=1 \\ T(\lfloor n/2 \rfloor) + T(n - \lfloor n/2 \rfloor) + 1, & n > 1 \end{cases}$$

Exemplu 1

$$T(n) = \begin{cases} 0, & n=1 \\ T(\lfloor n/2 \rfloor) + T(n - \lfloor n/2 \rfloor) + 1, & n > 1 \end{cases}$$

Caz particular: $n=2^m$

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

Substituție inversă:

$$T(2^m) = 2T(2^{m-1}) + 1$$

$$T(2^{m-1}) = 2T(2^{m-2}) + 1 \quad | * 2$$

...

$$T(2) = 2T(1) + 1 \quad | * 2^{m-1}$$

$$T(1) = 0$$

$$T(n) = 1 + \dots + 2^{m-1} = 2^m - 1 = n - 1$$

Exemplu 1

$$T(n) = \begin{cases} 0, & n=1 \\ T(\lfloor n/2 \rfloor) + T(n - \lfloor n/2 \rfloor) + 1, & n > 1 \end{cases}$$

Caz particular:

$$n = 2^m \Rightarrow T(n) = n - 1$$

Caz general.

(a) Demonstrație prin inducție matematică completă

Verificare. $n=1 \Rightarrow T(n)=0=n-1$

Pasul de inducție.

Presupunem că $T(k)=k-1$ pentru orice $k < n$.

Atunci

$$T(n) = \lfloor n/2 \rfloor - 1 + n - \lfloor n/2 \rfloor - 1 + 1 = n - 1$$

Deci $T(n) = n - 1 \Rightarrow$

$T(n)$ aparține lui $\Theta(n)$.

Exemplu 1

Caz general.

(b) Regula funcțiilor “netede”

Dacă $T(n)$ aparține lui $\Omega(f(n))$ pentru $n=b^m$

$T(n)$ este crescătoare pentru valori mari ale lui n

$f(n)$ este “netedă” ($f(cn)$ aparține lui $\Theta(f(n))$ pentru orice constantă pozitivă c)

atunci $T(n)$ aparține lui $\Omega(f(n))$ pentru orice n

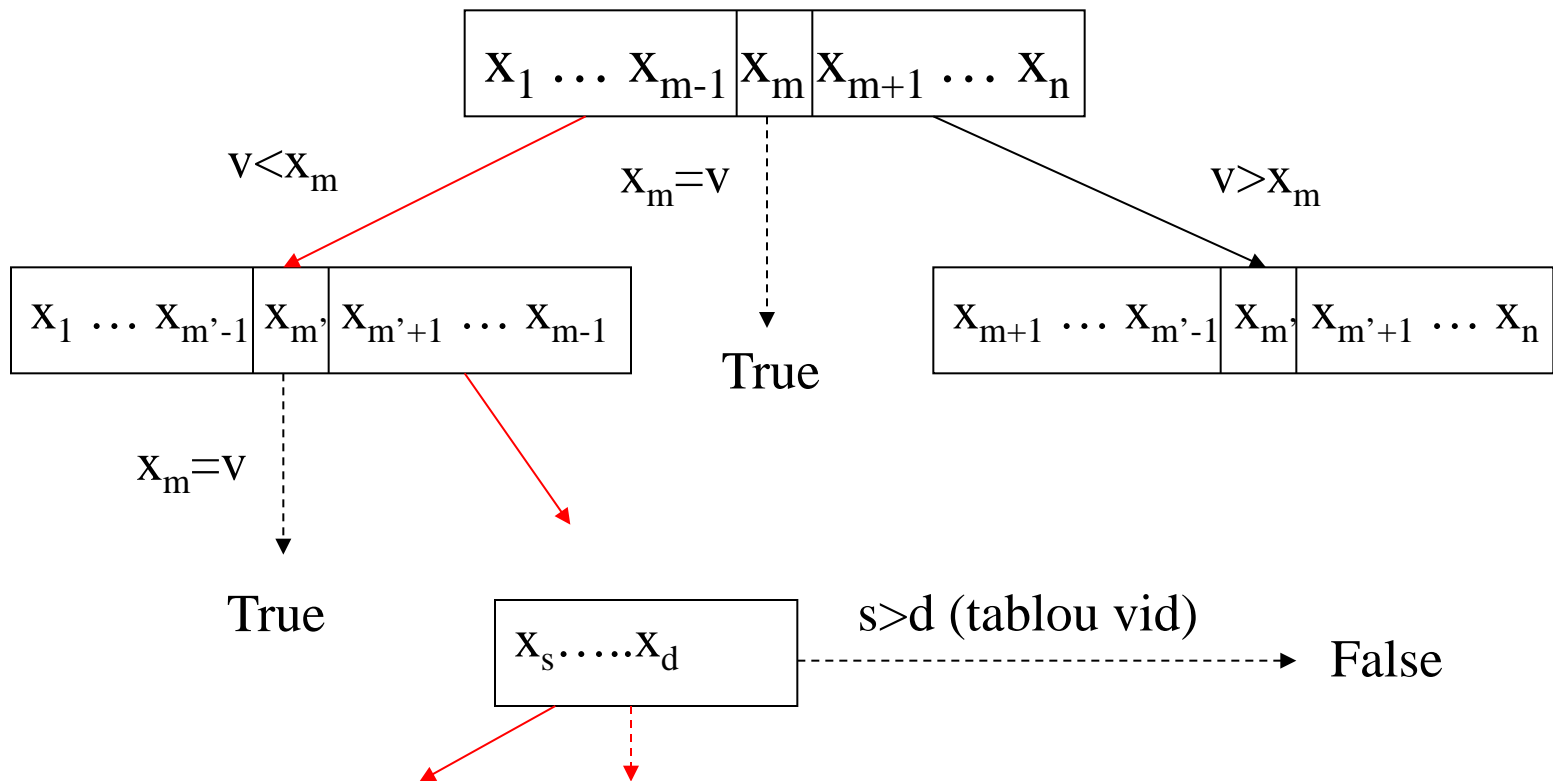
Observații.

- Toate funcțiile care nu cresc foarte rapid (ex: funcția logaritmică și cea polinomială) sunt funcții netede. În schimb funcția exponențială nu are această proprietate: a^{cn} nu aparține lui $\Theta(a^n)$
- Pentru algoritmul “maxim” : $T(n)$ este crescătoare, $f(n)=n$ este netedă, deci $T(n)$ este din $\Theta(n)$ pentru orice valoare a lui n

Exemplu 2 – căutare binară

Să se verifice dacă o valoare dată, v , aparține sau nu unui tablou **ordonat crescător**, $x[1..n]$ ($x[i] \leq x[i+1]$, $i=1..(n-1)$)

Idee: se compară v cu elementul din mijloc și se continuă căutarea fie în subtabloul stâng fie în cel drept



Exemplu 2 – căutare binară

Varianta recursivă:

```
cautbin(x[s..d],v)
IF s>d THEN RETURN False
ELSE
  m ← (s+d) DIV 2
  IF v=x[m] THEN RETURN True
  ELSE
    IF v<x[m]
      THEN RETURN cautbin(x[s..m-1],v)
      ELSE RETURN cautbin(x[m+1..d],v)
    ENDIF
  ENDIF
ENDIF
```

Apel funcție:

cautbin(x[1..n],v)

(la început s=1, d=n)

Observație:

$n_c=0$

k=2

Doar una dintre
subprobleme este
rezolvată

Căutarea binară se
bazează de fapt
pe tehnica
reducerii

Exemplu 2 – căutare binară

Varianta iterativă 1:

```
cautbin1(x[1..n],v)
```

```
  s ← 1
```

```
  d ← n
```

```
  WHILE s<=d DO
```

```
    m ←(s+d) DIV 2
```

```
    IF v=x[m] THEN RETURN True
```

```
    ELSE
```

```
      IF v<x[m]
```

```
        THEN d ← m-1
```

```
        ELSE s ← m+1
```

```
      ENDIF / ENDIF/ ENDWHILE
```

```
  RETURN False
```

Varianta iterativă 2:

```
cautbin2(x[1..n],v)
```

```
  s ← 1
```

```
  d ← n
```

```
  WHILE s<d DO
```

```
    m ←(s+d) DIV 2
```

```
    IF v<=x[m]
```

```
      THEN d ← m
```

```
      ELSE s ← m+1
```

```
    ENDIF / ENDWHILE
```

```
  IF x[s]=v THEN RETURN True
```

```
    ELSE RETURN False
```

```
  ENDIF
```

Exemplu 2 – căutare binară

Varianta iterativă 2:

cautbin2(x[1..n],v)

s ← 1

d ← n

WHILE s < d DO

 m ← (s+d) DIV 2

 IF v ≤ x[m]

 THEN d ← m

 ELSE s ← m+1

 ENDIF / ENDWHILE

IF x[s]=v THEN RETURN True

 ELSE RETURN False

ENDIF

Corectitudine

Precondiție: $n \geq 1$

Postcondiție:

“returnează True dacă v este în x[1..n] și False în caz contrar”

Invariant: “v este în x[1..n] dacă și numai dacă v este în x[s..d]”

- (i) $s=1, d=n \Rightarrow$ invariantul e adevărat
- (ii) Rămâne adevărat prin execuția corpului ciclului
- (iii) când $s=d$ se obține postcondiția

Exemplu 2 – căutare binară

Varianta iterativă 2:

cautbin2(x[1..n],v)

s ← 1

d ← n

WHILE s<d DO

 m ← (s+d) DIV 2

 IF v ≤ x[m]

 THEN d ← m

 ELSE s ← m+1

 ENDIF / ENDWHILE

IF x[s]=v THEN RETURN True

 ELSE RETURN False

ENDIF

Eficiența:

Caz defavorabil: x nu conține pe v

Caz particular: $n=2^m$

$$T(n) = \begin{cases} 1 & n=1 \\ T(n/2)+1 & n>1 \end{cases}$$

$$T(n) = T(n/2) + 1$$

$$T(n/2) = T(n/4) + 1$$

...

$$T(2) = T(1) + 1$$

$$T(1) = 1$$

$$T(n) = \lg n + 1 \quad O(\lg n)$$

Exemplu 2 – căutare binară

Observație:

- Aplicând regula funcțiilor “netede” rezultă că algoritmul `cautbin2` (similar se poate arăta pentru celelalte variante) are ordinul de complexitate $O(\log n)$ pentru orice valoare a lui n
- Analiza eficienței algoritmilor proiectați utilizând tehnicile de reducere și divizare poate fi ușurată prin folosirea teoremei master

Teorema “master”

Considerăm următoarea relație de recurență:

$$T(n) = \begin{cases} T_0 & n \leq n_c \\ kT(n/m) + T_{DC}(n) & n > n_c \end{cases}$$

Dacă $T_{DC}(n)$ aparține lui $\Theta(n^d)$ ($d \geq 0$) atunci

$$T(n) \text{ aparține lui } \begin{cases} \Theta(n^d) & \text{daca } k < m^d \\ \Theta(n^d \log(n)) & \text{daca } k = m^d \\ \Theta(n^{\log(k)/\log(m)}) & \text{daca } k > m^d \end{cases}$$

Obs:

1. m reprezintă nr de subprobleme în care se descompune problema inițială iar k e nr de subprobleme care se rezolvă efectiv
2. Un rezultat similar există pentru clasele O și Ω

Teorema “master”

Utilitate:

- Poate fi aplicată în analiza algoritmilor bazați pe tehnica reducerii sau a divizării
- Evită rezolvarea explicită a relației de recurență corespunzătoare timpului de execuție
- În multe aplicații practice etapele de divizare (reducere) și de combinare sunt de complexitate polinomială
- Spre deosebire de variantele de rezolvare explicită a relației de recurență furnizează doar ordinul de complexitate nu și constantele ce intervin în estimarea timpului de execuție

Teorema “master”

Exemplu1 : calcul maxim:

$k=2$ (pb inițială se divide în două subprobleme, iar ambele subprobleme trebuie rezolvate)

$m=2$ (dimensiunea fiecărei subprobleme este aproximativ $n/2$)

$d=0$ (etapele de divizare și de combinare a rezultatelor au cost constant)

Intrucât $k > m^d$ prin aplicarea celui de al treilea caz al teoremei “master” rezulta ca $T(n)$ aparține lui $\Theta(n^{\log(k)/\log(m)}) = \Theta(n)$

Teorema “master”

Exemplu 2: căutare binară

$k=1$ (doar una dintre subprobleme trebuie rezolvată)

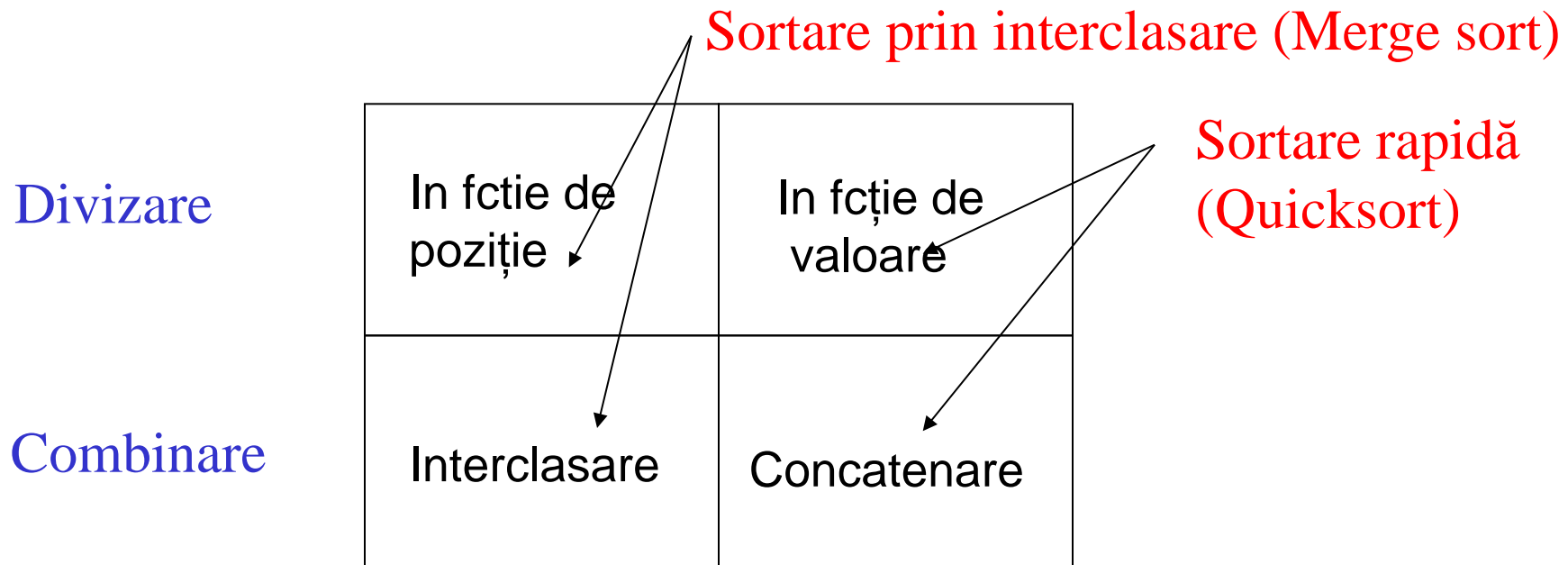
$m=2$ (dimensiunea problemei este $n/2$)

$d=0$ (etapele de divizare și combinare au cost constant)

Intrucât $k=m^d$ prin aplicarea celui de al doilea caz al teoremei “master” se obține ca $T(n)$ aparține lui $O(n^d \lg(n)) = \Theta(\lg n)$

Sortare eficientă

- Metodele elementare de sortare aparțin lui $O(n^2)$
- Idee de eficientizare a procesului de sortare:
 - Se împarte secvența inițială în două subsecvențe
 - Se sortează fiecare subsecvență
 - Se combină subsecvențele sortate



Sortare prin interclasare

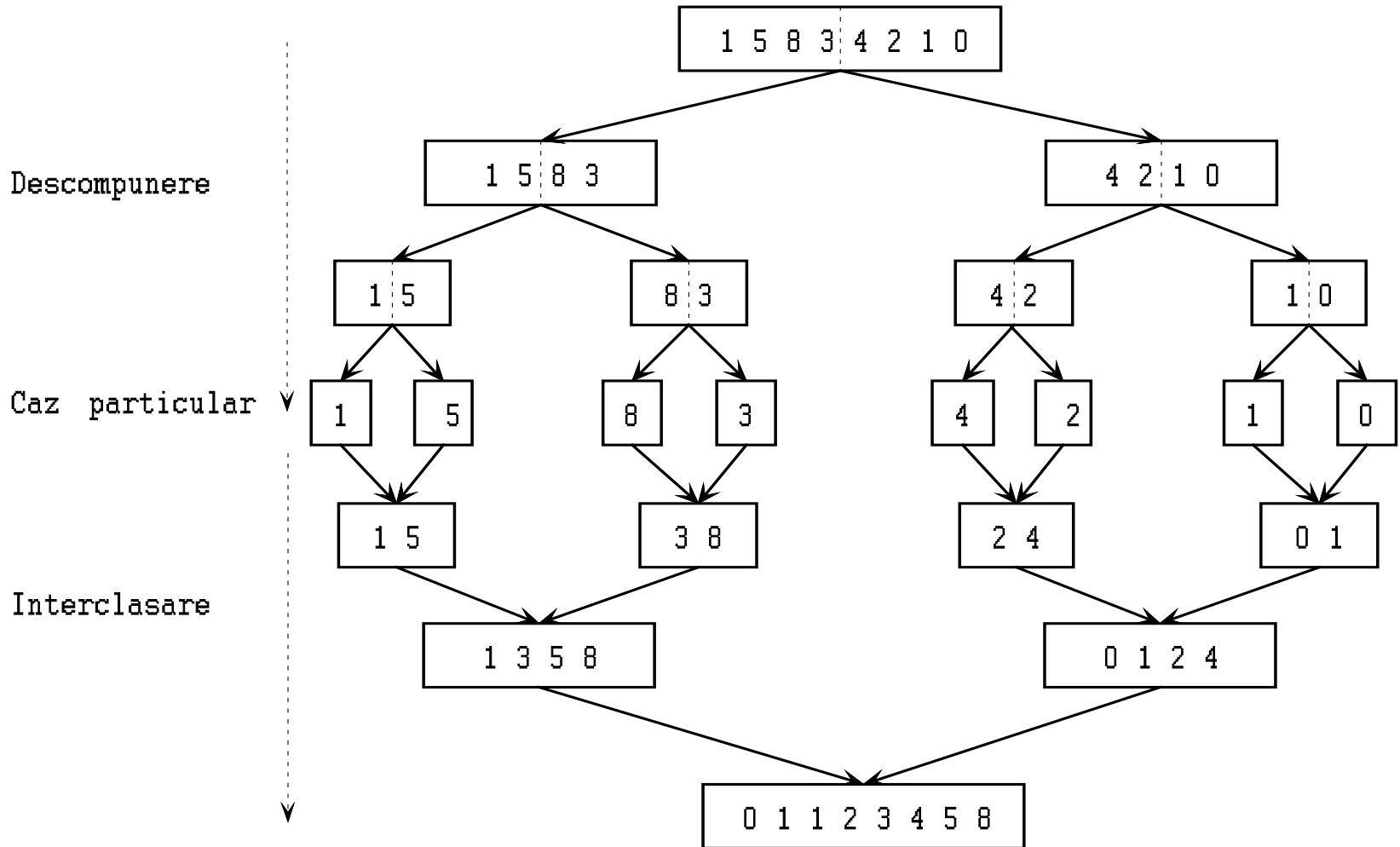
Idee de bază:

- Imparte $x[1..n]$ în două subtablouri $x[1..[n/2]]$ and $x[[n/2]+1..n]$
- Sortează fiecare subtablou
- Interclasează elementele subtablourilor $x[1..[n/2]]$ și $x[[n/2]+1..n]$ și construiește tabloul sortat $t[1..n]$. Transferă conținutul tabloului temporar t în $x[1..n]$

Observații:

- Valoarea critică: 1 (un tablou conținând un singur element este implicit sortat)
- Valoarea critică poate fi mai mare decât 1 (de exemplu, 10) iar sortarea subtablourilor cu un număr de elemente mai mic decât valoarea critică se poate realiza cu unul dintre alg. elementari (ex.: Sortare prin inserție).

Sortare prin interclasare



Sortare prin interclasare

Algorithm:

```
sortare(x[s..d])
IF s<d THEN
    m ← (s+d) DIV 2           //divizare
    x[s..m] ← sortare(x[s..m]) //rezolvare
    x[m+1..d] ← sortare(x[m+1..d])
    x[s..d] ← interclasare(x[s..m],x[m+1..d]) //combinare
ENDIF
RETURN x[s..d]
```

Obs:

algoritmul se apelează prin sortare(x[1..n])

Sortare prin interclasare

```
interclasare (x[s..m],x[m+1..d])
  i ← s; j ← m+1; k ← 0;
// se parcurg subtablourile în
// paralel și la fiecare pas se
// transferă cel mai mic
// element
WHILE i<=m AND j<=d DO
  k ← k+1
  IF x[i]<=x[j]
    THEN
      t[k] ← x[i]
      i ← i+1
    ELSE
      t[k] ← x[j]
      j ← j+1
  ENDIF
ENDWHILE
```

```
// se transferă eventualele elemente
// rămase în primul subtablou
WHILE i<=m DO
  k ← k+1
  t[k] ← x[i]
  i ← i+1
ENDWHILE
// se transferă eventualele elemente
// rămase în al doilea subtablou
WHILE j<=d DO
  k ← k+1
  t[k] ← x[j]
  j ← j+1
ENDWHILE
RETURN t[1..k]
```

Sortare prin interclasare

- Interclasarea este o prelucrare ce poate fi utilizată pentru construirea unui tablou sortat pornind de la alte două tablouri sortate ($a[1..p]$, $b[1..q]$)
- Varianta de interclasare bazată pe valori santinelă:
Se adaugă două valori mai mari decât elementele tablourilor $a[p+1]=\infty$, $b[q+1]=\infty$

```
interclasare(a[1..p],b[1..q])
```

```
a[p+1] ← ∞ ; b[q+1] ← ∞
```

```
i ← 1; j ← 1;
```

```
FOR k ← 1,p+q DO
```

```
  IF a[i] ≤ b[j]
```

```
    THEN c[k] ← a[i]
```

```
      i ← i+1
```

```
    ELSE c[k] ← b[j]
```

```
      j ← j+1
```

```
  ENDIF / ENDFOR
```

```
RETURN c[1..p+q]
```

Analiza eficienței interclasării

Operație dominantă: comparația

$$T(p,q)=p+q$$

In algoritmul de sortare ($p=\lfloor n/2 \rfloor$, $q=n-\lfloor n/2 \rfloor$):

$$T(n) \leq \lfloor n/2 \rfloor + n - \lfloor n/2 \rfloor = n$$

Deci $T(n)$ aparține lui $O(n)$

(interclasarea este de complexitate liniară)

Cursul următor va fi despre...

... sortare rapidă

... și alte aplicații ale tehnicii divizării