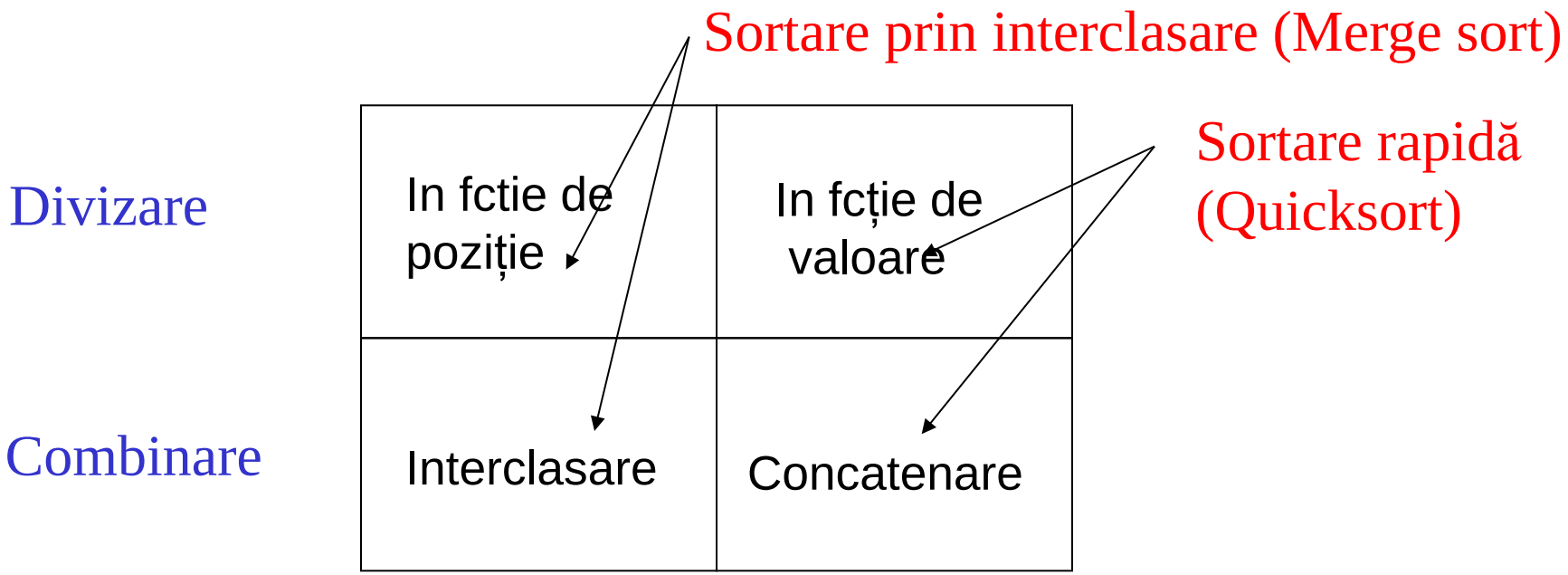
Curs 9:

Tehnica divizării.
Sortare prin interclasare și sortare rapidă

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

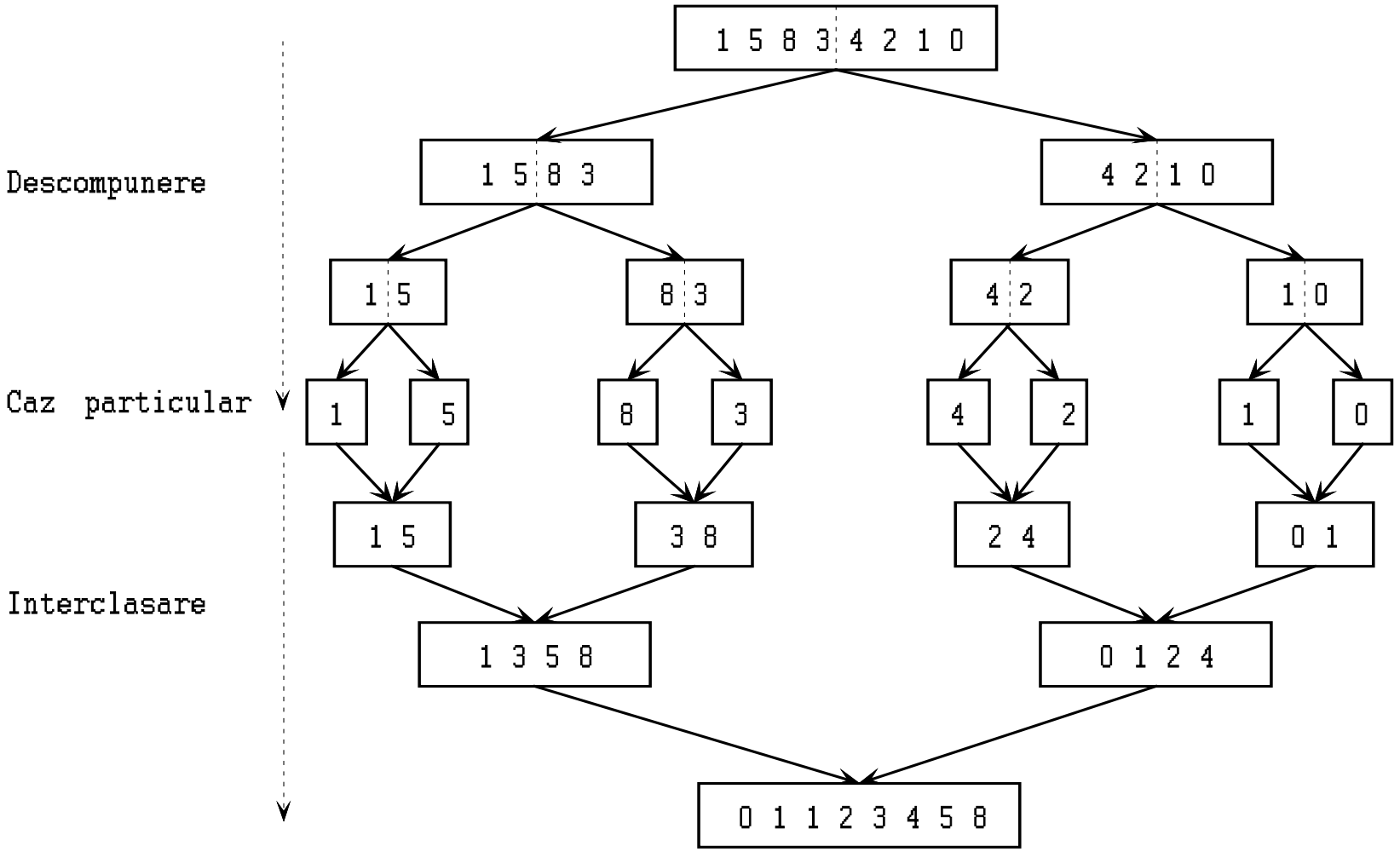
Idee de bază:

- 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]] si 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

Algoritm:

```
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]=max, b[q+1]= max

```
interclasare(a[1..p],b[1..q])
a[p+1] ← max;  b[q+1] ← max
  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=[n/2], q=n-[n/2]):

T(n)<=[n/2]+n-[n/2]=n

Deci T(n) aparține lui O(n)

(interclasarea este de complexitate liniară)

Sortare prin interclasare

Analiza sortării prin interclasare:

$$T(n)=\begin{cases} 0 & n=1 \\ T([n/2])+T(n-[n/2])+T_M(n) & n>1 \end{cases}$$

Intrucât k=2, m=2, d=1 ($T_M(n)$ aparține lui O(n)) rezultă (folosind al doilea caz din teorema “master”) că T(n) aparține lui O(nlgn). De fapt T(n) aparține lui Θ(nlgn)

Observații.

1. Principalul dezavantaj al sortării prin interclasare este faptul că utilizează un tablou adițional de dimensiunea tabloului de sortat
2. Dacă în pasul de interclasare se folosește inegalitate de tip <= atunci sortarea prin interclasare este stabilă

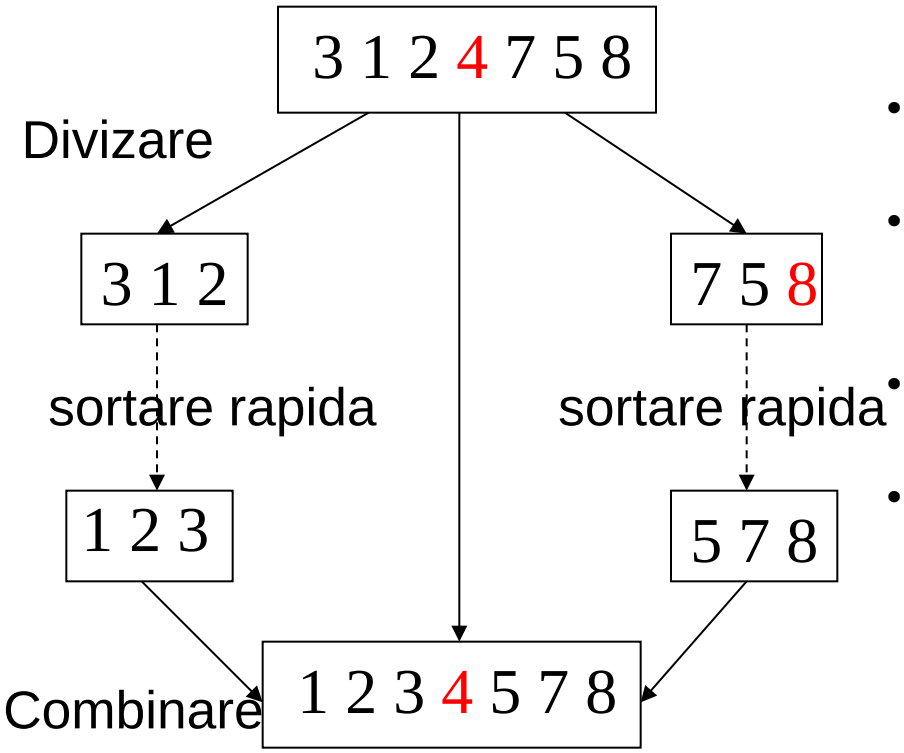
Sortare rapidă

Idee:

- Se reorganizează și se imparte tabloul x[1..n] în două subtablouri x[1..q] și x[q+1..n] astfel încât elementele lui x[1..q] sunt mai mici decât x[q+1..n]
- Se sortează fiecare dintre subtablouri aplicând aceeași strategie
- Se concatenează subtablourile sortate
- Creator: Tony Hoare (1962)

Sortare rapidă

Exemplu 1



- Un element x[q] avnd proprietățile:

(a) x[q]>=x[i], for all i<q

(b) x[q]<=x[i], for all i>q

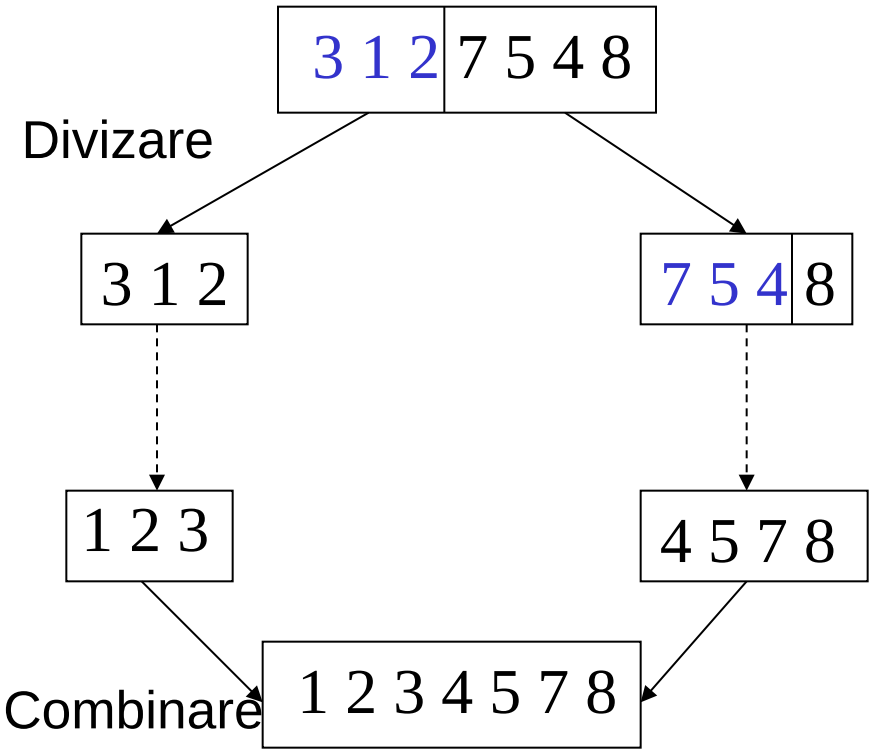
este denumit pivot

- Un pivot este un element aflat pe poziția sa finală
- Un bun pivot împarte tabloul curent în două subtablouri având dimensiuni apropiate (partiționare echilibrată)
- Uneori pivotul împarte tabloul în mod neechilibrat
- Alteori nu există un astfel de pivot

(de ex. (3 1 2)). In acest caz trebuie creat un pivot prin interschimbarea elementelor

Sortare rapidă

Exemplu 2



- O poziție q având proprietatea:

  (a) x[i]<=x[j], pentru 1<=i<=q si q+1<=j<=n

  Este denumită poziție de partiționare

- O poziție de partiționare bună divide tabloul curent în subtablouri de dimensiuni apropiate
- Uneori poziția de partiționare divide tabloul în mod neechilibrat
- Alteori nu există o poziție de partiționare. In acest caz se creează o astfel de poziție prin interschimbarea unor elemente

Sortare rapidă

Varianta ce utilizează pivot:

```
quicksort1(x[s..d])
IF s<d THEN
  q ← pivot(x[s..d])
  x[s..q-1] ← quicksort1(x[s..q-1])
  x[q+1..d] ← quicksort1(x[q+1..d])
ENDIF
RETURN x[s..d]
```

Varianta ce utilizează poziție de partiționare:

```
quicksort2(x[s..d])
IF s<d THEN
 q ← partitie(x[s..d])
 x[s..q] ← quicksort2(x[s..q])
 x[q+1..d] ← quicksort2(x[q+1..d])
ENDIF
RETURN x[s..d]
```

Sortare rapidă

Construirea unui pivot:

- Se alege o valoare arbitrară din tablou (prima, ultima sau una aleatoare) – aceasta va reprezenta valoarea pivotului

- Se rearanjează elementele tabloului astfel încât toate elementele care care sunt mai mici decât valoarea aleasă să se afle în prima parte a tabloului iar valorile mai mari decât pivotul sunt în partea a doua a tabloului

- Se plasează valoarea pivotului pe poziția sa finală (astfel încât toate elementele din stânga sa să fie mai mici iar toate elementele din dreapta sa să fie mai mari)

Sortare rapidă

O idee de rearanjare a elementelor:

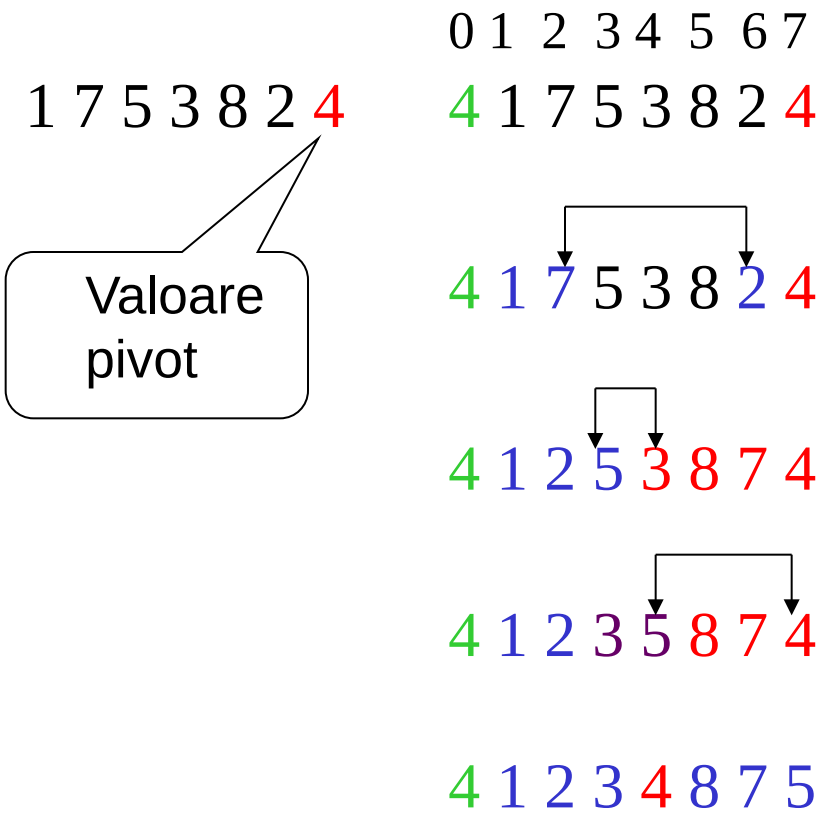
- Se folosesc doi indicatori: unul care pornește de la primul element iar celălalt care pornește de la ultimul element
- Se măresc respectiv micșorează indicatorii până când se identifică o inversiune.

Inversiune: pereche de indici i<j cu proprietatea că x[i]>pivot și x[j]<pivot

- Se repară inversiunea prin interschimbarea elementelor
- Se continuă procesul până când indicatorii se “întâlnesc”

Sortare rapidă

Construirea unui pivot



- Se alege valoarea pivotului: 4 (ultima valoare din tablou)
- Se plasează o santinelă înaintea primei poziții a tabloului (doar pentru tabloul inițial)

i=0, j=7

i=2, j=6

i=3, j=4

i=4, j=3 (indicatorii s-au încrucișat)

Pivotul este plasat pe poziția sa finală

Sortare rapidă

```
pivot(x[s..d])
   v ← x[d]
   i ← s-1
   j ← d
   WHILE i<j DO
      REPEAT i ← i+1 UNTIL x[i]>=v
      REPEAT j ← j-1 UNTIL x[j]<=v
      IF i<j THEN x[i]↔x[j] ENDIF
  ENDWHILE
  x[i] ↔ x[d]
  RETURN i
```

Observatii:

- x[d] joacă rolul unei santinele la extremitatea dreaptă
- La extremitatea stângă se plasează explicit o valoare santinelă x[0]=v (doar pentru tabloul inițial x[1..n])
- Condițiile x[i]>=v, x[j]<=v permit oprirea căutarii când sunt întâlnite santinelele. De asemenea permit obținerea unei partiționări echilibrate atunci cand tabloul conține elemente identice
- La sfârșitul ciclului while indicatorii satisfac fie i=j fie i=j+1

Sortare rapidă

```
pivot(x[s..d])
   v ← x[d]
   i ← s-1
   j ← d
   WHILE i<j DO
      REPEAT i ← i+1 UNTIL x[i]>=v
      REPEAT j ← j-1 UNTIL x[j]<=v
      IF i<j THEN x[i] ↔x[j] ENDIF
  ENDWHILE
  x[i] ↔x[d]
  RETURN i
```

Corectitudine:

Invariant:

Dacă i<j atunci x[k]<=v for k=s..i
x[k]>=v for k=j..d

Dacă i>=j atunci x[k]<=v for k=s..i
x[k]>=v for k=j+1..d

Sortare rapidă

```
pivot(x[s..d])
   v ← x[d]
   i ← s-1
   j ← d
   WHILE i<j DO
      REPEAT i ← i+1 UNTIL x[i]>=v
      REPEAT j ← j-1 UNTIL x[j]<=v
      IF i<j THEN x[i] ↔x[j] ENDIF
   ENDWHILE
  x[i] ↔x[d]
  RETURN i
```

Eficiența:

Dimensiune pb.: n=d-s+1

Op. dominantă: comparația în care intervin elemente ale lui x

T(n)=n+c,

c=0 daca i=j

și

c=1 daca i=j+1

Deci T(n) aparține lui Θ (n)

Sortare rapidă

Obs: poziția pivotului nu împarte întotdeauna tabloul în mod echilibrat

Partiționare echilibrată:

- Tabloul este împărțit în două subtablouri de dimensiuni apropiate de n/2
- Dacă fiecare partiționare este echilibrată atunci algoritmul execută mai puține operații (corespunde celui mai favorabil caz)

Partiționare neechilibrată:

- Tabloul este împărțit într-un subtablou cu (n-1) elemente, pivotul și un subtablou vid
- Dacă fiecare partiționare este neechilibrată atunci algoritmul execută mai multe operații (corespunde celui mai defavorabil caz)

Sortare rapidă

Analiza în cazul cel mai defavorabil:

$$T(n)=\begin{cases}0 & \text{if } n=1\\ T(n-1)+n+1, & \text{if } n>1\end{cases}$$

Substituție inversă:

T(n)=T(n-1)+(n+1)

T(n-1)=T(n-2)+n

…

T(2)=T(1)+3

T(1)=0

T(n)=(n+1)(n+2)/2-3

In cel mai defavorabil caz algoritmul este de complexitate pătratică

Deci sortarea rapidă aparține lui $O(n^2)$

Sortare rapidă

Analiza în cazul cel mai favorabil:

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

Aplicând cazul al doilea al teoremei "master" (pentru k=2,m=2,d=1) rezultă că în cel mai favorabil caz ordinul de complexitate este nlog(n)

Deci algoritmul de sortare rapidă aparține lui $\Omega(n\log(n))$ si lui $O(n^2)$

Analiza în cazul mediu ar putea fi utilă

Sortare rapidă

Analiza în cazul mediu.

Ipoteze:

- Fiecare pas de partiționare necesită cel mult (n+1) comparații
- Există n poziții posibile pentru pivot. Presupunem că fiecare dintre aceste poziții are aceeași șansă de a fi selectată (Prob(q)=1/n)
- Dacă pivotul se află pe poziția q atunci numărul de comparații satisface

$$T_q(n)=T(q-1)+T(n-q)+(n+1)$$

Sortare rapidă

Numărul mediu de comparații este

$T_a(n)=(T_1(n)+\ldots+T_n(n))/n$

$=((T_a(0)+T_a(n-1))+(T_a(1)+T_a(n-2))+\ldots+(T_a(n-1)+T_a(0)))/n + (n+1)$

$=2(T_a(0)+T_a(1)+\ldots+T_a(n-1))/n+(n+1)$

Deci

$n\ T_a(n) = 2(T_a(0)+T_a(1)+\ldots+T_a(n-1))+n(n+1)$

$(n-1)T_a(n-1)= 2(T_a(0)+T_a(1)+\ldots+T_a(n-2))+(n-1)n$

Calculând diferența dintre ultimele două egalitati:

$nT_a(n)=(n+1)T_a(n-1)+2n$

$T_a(n)=(n+1)/n\ T_a(n-1)+2$

Sortare rapidă

Analiza în cazul mediu.

Prin substituție inversă:

$T_a(n) = (n+1)/n \; T_a(n-1)+2$

$T_a(n-1) = n/(n-1) \; T_a(n-2)+2 \quad |*(n+1)/n$

$T_a(n-2) = (n-1)/(n-2) \; T_a(n-3)+2 \quad |*(n+1)/(n-1)$

…

$T_a(2) = 3/2 \; T_a(1)+2 \quad |*(n+1)/3$

$T_a(1) = 0 \quad |*(n+1)/2$

$T_a(n) = 2+2(n+1)(1/n+1/(n-1)+\ldots+1/3) \approx 2(n+1)(\ln n-\ln 3)+2$

In cazul mediu ordinul de complexitate este nlog(n)

Sortare rapidă -variante

Alta variantă de construire a pivotului

```
pivot(x[s..d])
   v ← x[s]
   i ← s
   FOR j ← s+1,d
      IF x[j]<=v THEN
         i ← i+1
         x[i] ↔ x[j]
      ENDIF
   ENDFOR
x[s] ↔x[i]
RETURN i
```

Invariant: x[k]<=v pentru s<=k<=i
x[k]>v pentru i<k<=j-1

3 7 5 2 1 4 8 v=3, i=1,j=2

3 7 5 2 1 4 8 i=2, j=4

3 2 5 7 1 4 8 i=3, j=5

3 2 1 7 5 4 8 i=3, j=8

Plasare pivot:

1 2 3 7 5 4 8

Pozitie pivot: 3

Ordin complexitate construire pivot: O(n)

Sortare rapidă-variante

Construirea unei poziții de partitionare

```
Partiție(x[s..d])
   v ← x[s]
   i ← s-1
   j ← d+1
   WHILE i<j DO
      REPEAT i ← i+1 UNTIL x[i]>=v
      REPEAT j ← j-1 UNTIL  x[j]<=v
      IF i<j THEN x[i] ↔x[j]
      ENDIF
  ENDWHILE
RETURN j
```

3 7 5 2 1 4 8 v=3

3 7 5 2 1 4 8 i=2, j=5

3 1 5 2 7 4 8 i=3, j=4

3 1 2 5 7 4 8 i=4, j=3

Poziție de partiționare: 3

Ordin complexitate: O(n)

Obs: Algoritmul de partiționare este folosit în algoritmul quicksort2

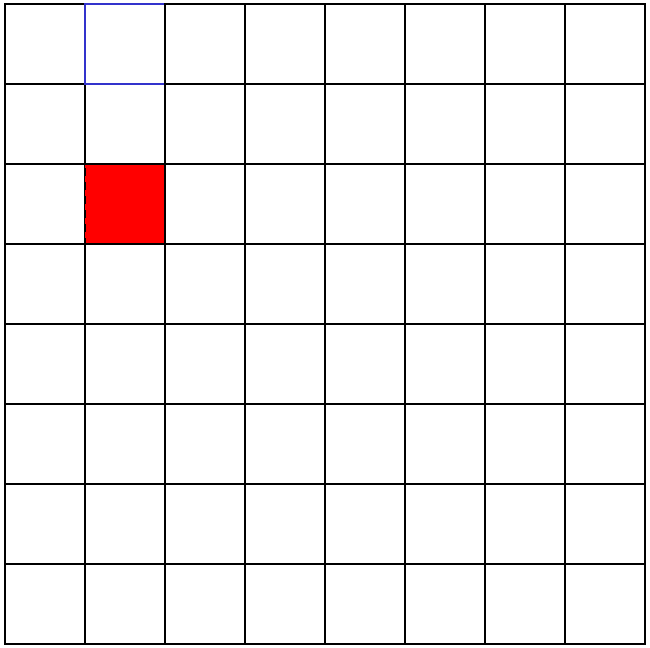
Sumar

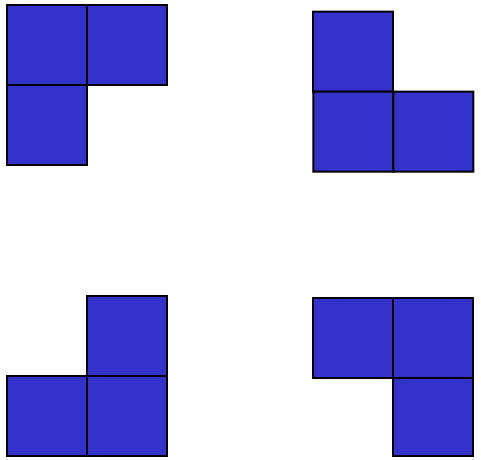
MergeSort si QuickSort

	Merge sort	Quicksort
Divizare	O(1) Determinare indice mijloc	O(n) Construire pivot
Combinare	O(n) Interclasare	O(1) Concatenare

Triomino (Tromino)

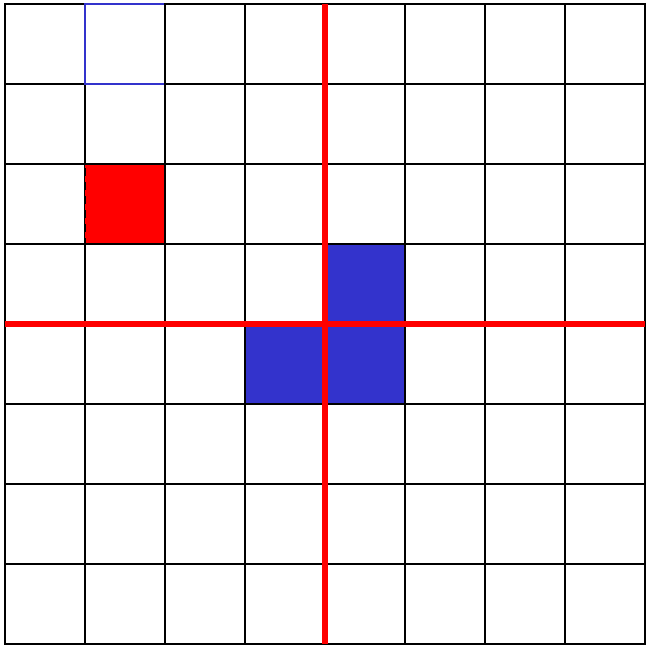
Se consideră o grilă pătratică de latură $n=2^k$ în care una dintre celule este marcată (interzisă). Se pune problema acoperirii grilei cu piese constituite din 3 celule plasate în forma de L (patru variante ce pot fi obținute prin rotire cu câte 90 grade)

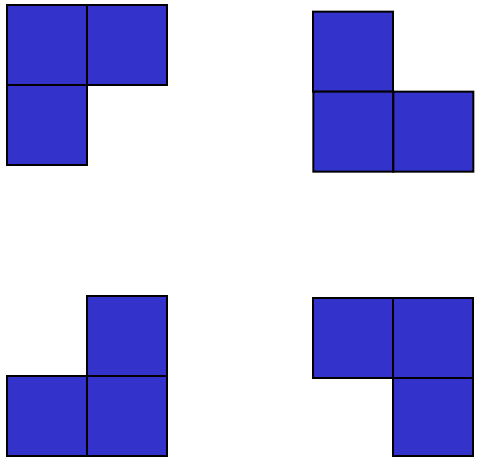




Triomino (Tromino)

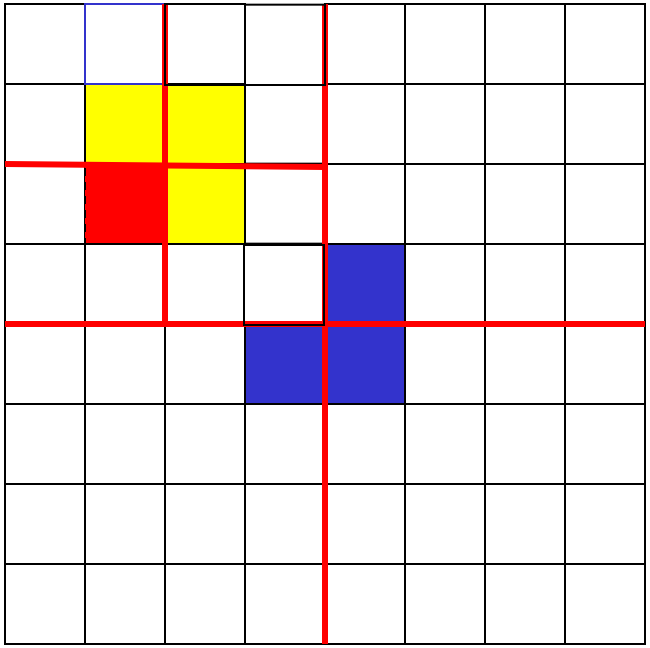
Idee: se reduce problema la 4 probleme similare de dimensiune 2^k prin plasarea unei piese în zona centrală, astfel încât să se ocupe o celulă în fiecare dintre cele 3 zone care au toate celulele libere

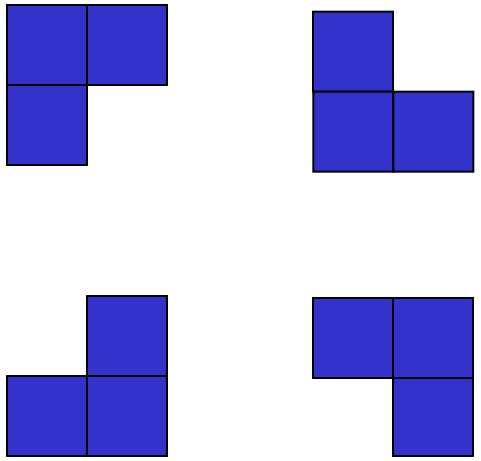




Triomino (Tromino)

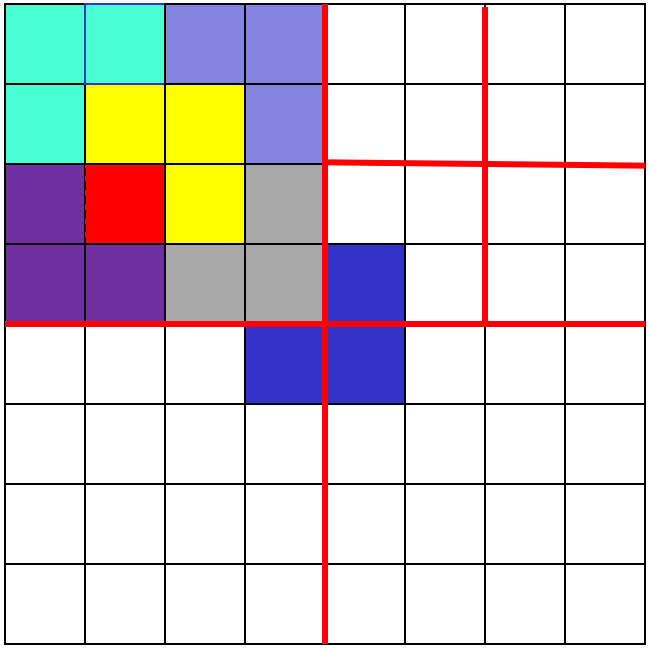
Idee: se aplică aceeași strategie pentru zona din stânga sus

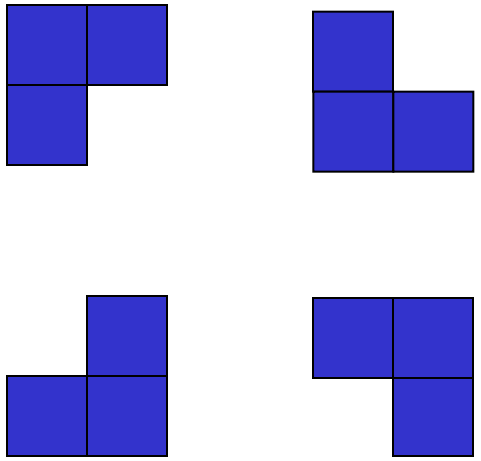




Triomino (Tromino)

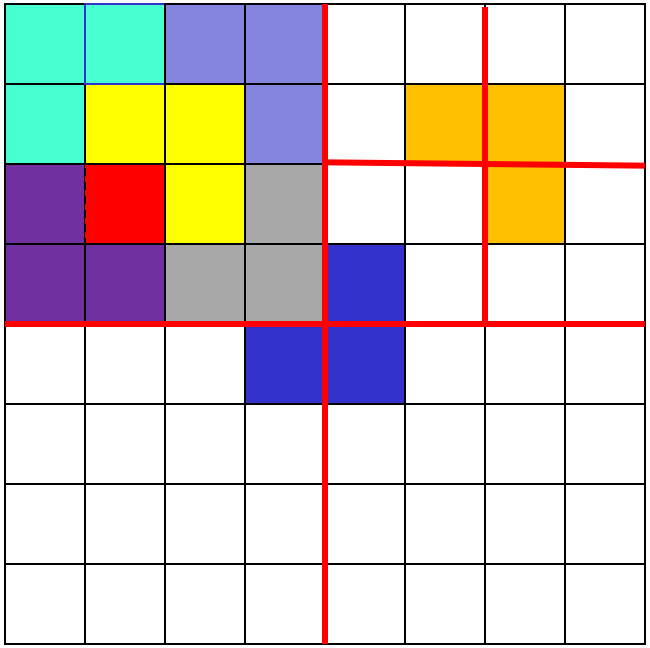
Idee: ... se aplică aceeași strategie pentru zona din dreapta sus

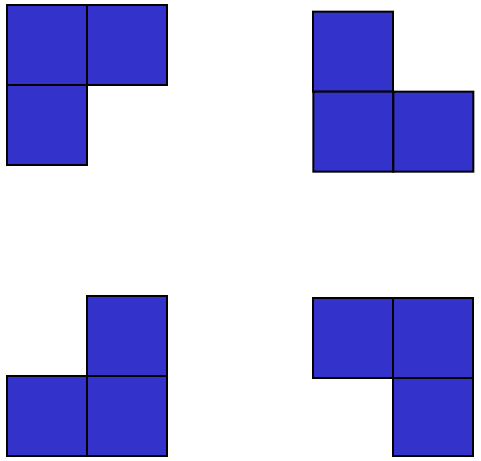




Triomino (Tromino)

Idee: ... se aplică aceeași strategie pentru zona din dreapta sus

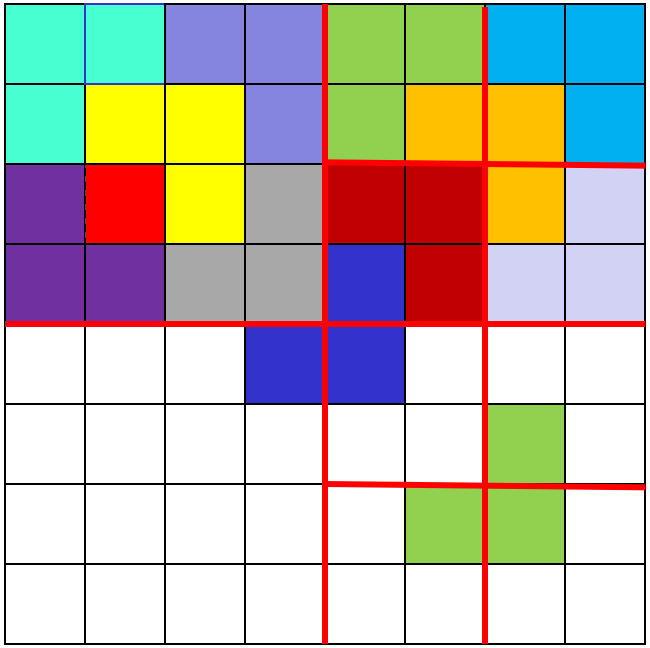


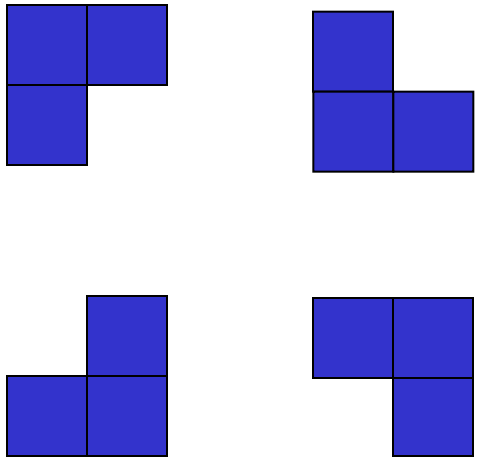


Triomino (Tromino)

Idee: apoi pentru zona din dreapta jos și în final pentru zona din stânga jos

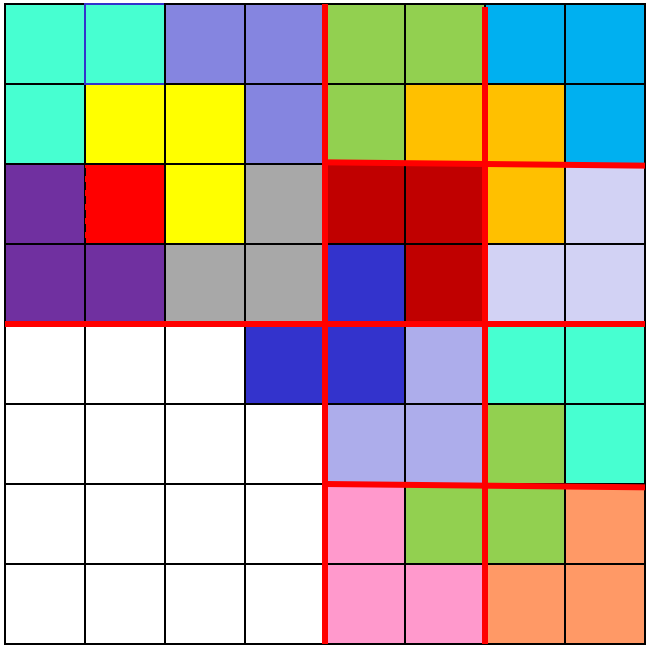
Obs: nu contează ordinea în care sunt rezolvate subproblemele

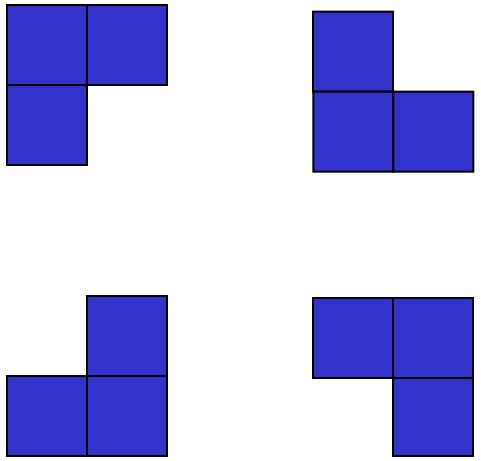




Triomino (Tromino)

Idee: apoi pentru zona din dreapta jos

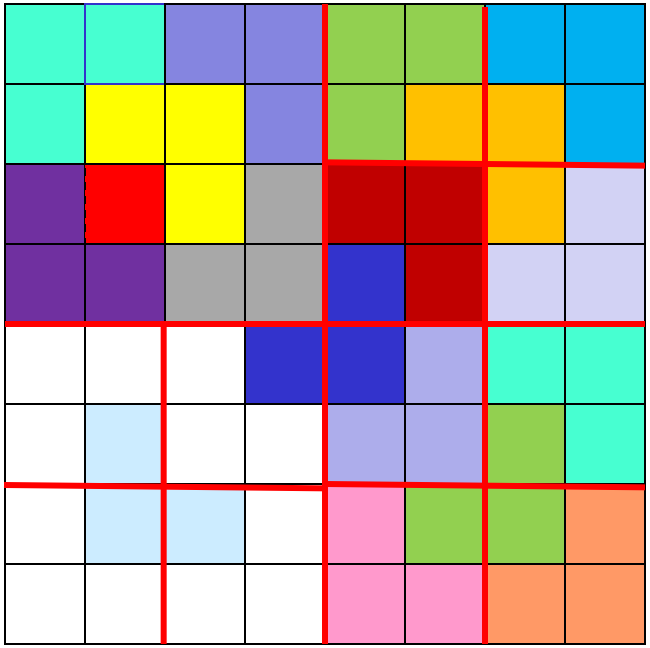


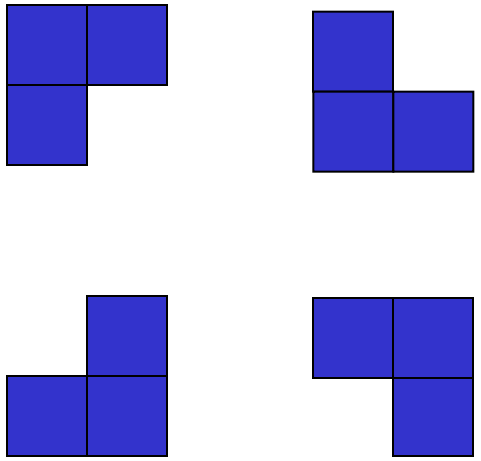


Triomino (Tromino)

Idee: ... și în final pentru zona din stânga jos

Obs: nu contează ordinea în care sunt rezolvate subproblemele





Triomino (Tromino)

Idee algoritm:

nr ← 0; // nr ordine piesa

Triomino(i1,j1,i2,j2,ih,jh) // i1,j1,i2,j2=indici colturi grila, ih,jh=indici celula ocupata

if ((i2-i1==1) and (j2-j1==1)) then <completare cele 3 celule libere>

else

imij ←(i1+i2)/2; jmij ←(j1+j2)/2 // calcul indici mijloc

if (ih<=imij) and (jh<=jmij) then // celula ocupata e in subgrila stanga sus

a[imij][jmij+1] ← nr; a[imij+1][jmij] ← nr; a[imij+1][jmij+1] ← nr; nr=nr+1;

triomino(i1,j1,imij,jmij,ih,jh); // subgrila stanga jos

triomino(i1,jmij+1,imij,j2,imij,jmij+1); // subgrila dreapta sus

triomino(imij+1,jmij+1,i2,j2,imij+1,jmij+1); // subgrila dreapta jos

triomino(imij+1,j1,i2,jmij,imij+1,jmij); // subgrila stanga jos

if ((ih<=imij) and (jh>jmij)) then …. // subgrila dreapta sus

if ((ih>imij) and (jh>jmij)) then …. // subgrila dreapta jos

if ((ih>imij) and (jh<=jmij)) then …. // subgrila stanga jos

Cursul următor va fi despre…

… tehnica căutarii local optimale

… și aplicații