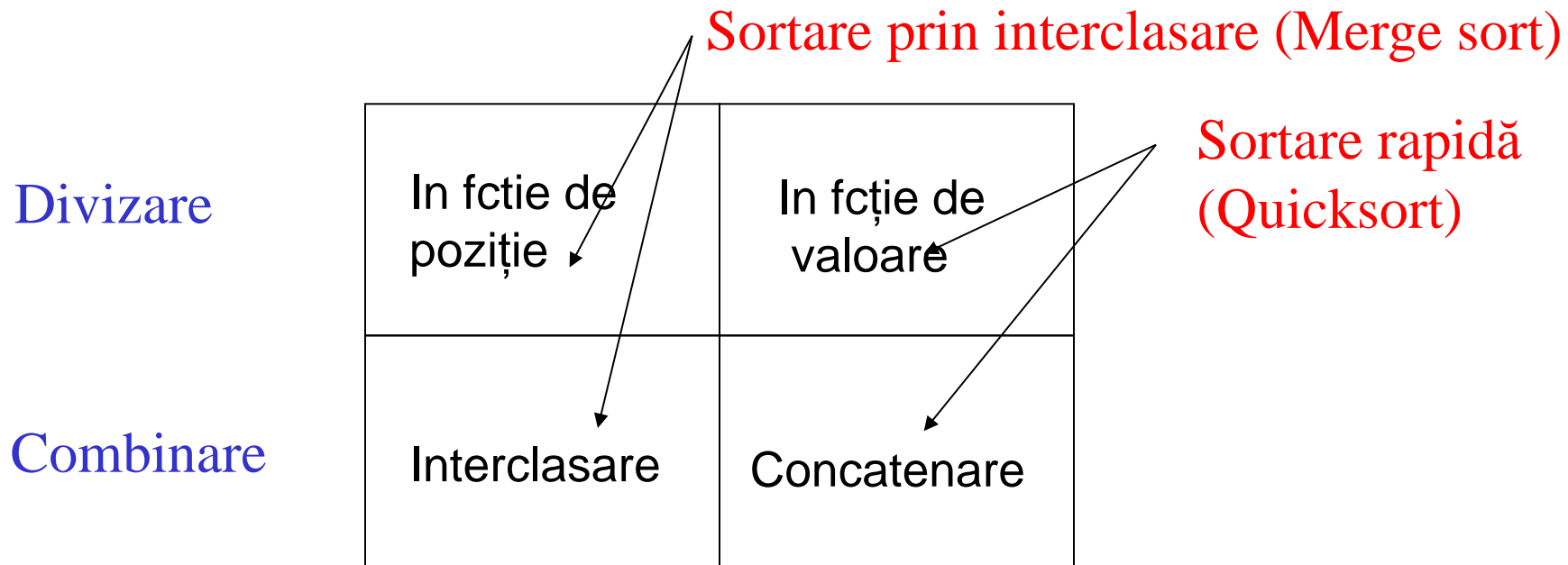


Curs 11:

Tehnica divizării (II).

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



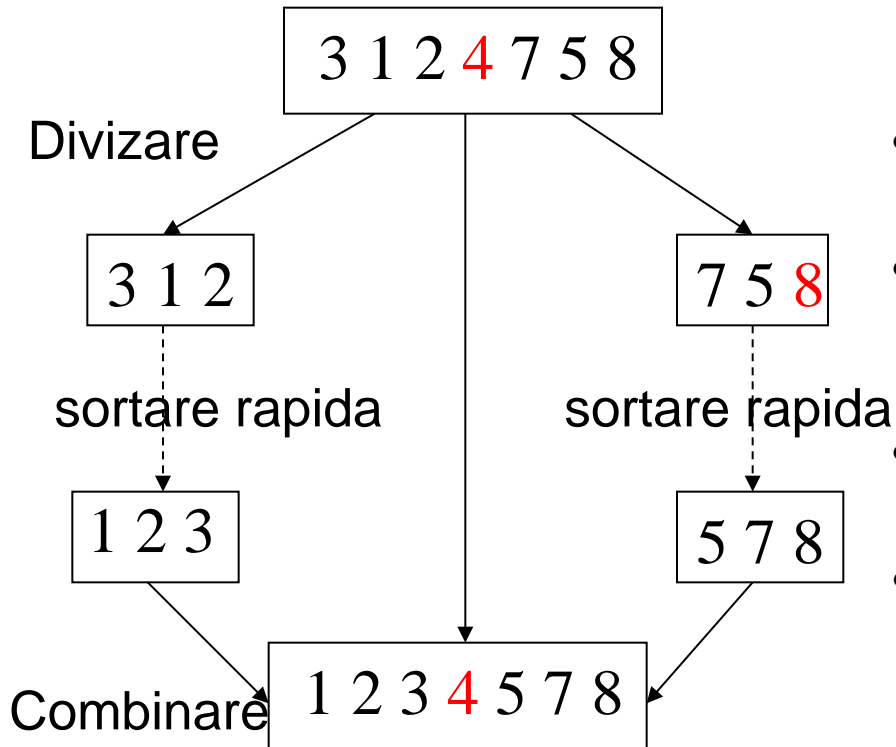
Sortare rapidă (quicksort)

Idee:

- Se reorganizează și se împarte 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

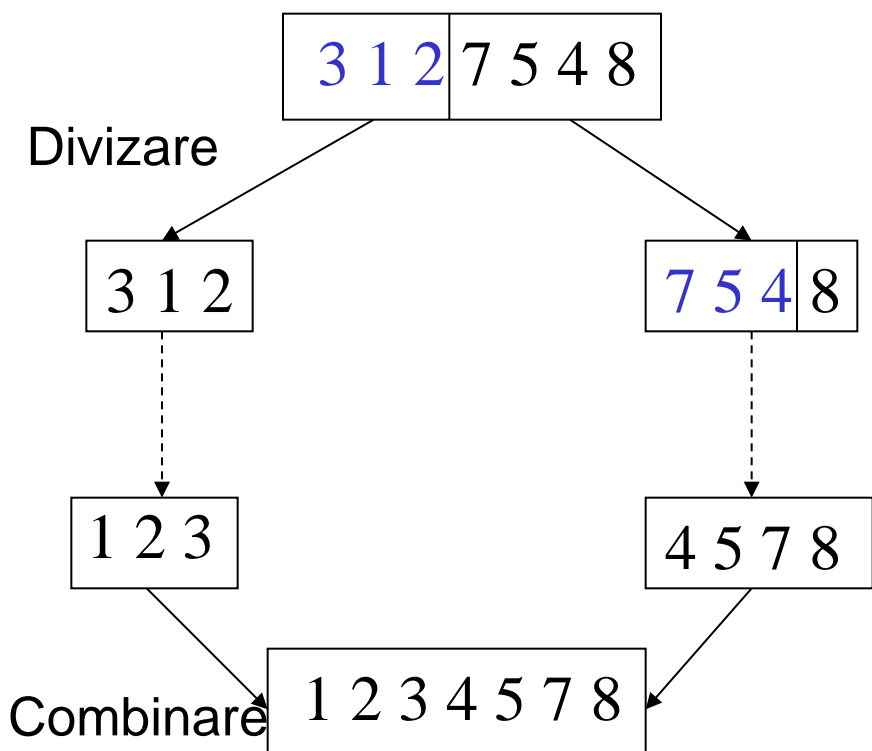
Exemplu 1



- Un element $x[q]$ având proprietățile:
 - (a) $x[q] \geq x[i]$, for all $i < q$
 - (b) $x[q] \leq x[i]$, 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)). În acest caz trebuie creat un pivot prin interschimbarea elementelor

Sortare rapidă

Exemplu 2



- O poziție q având proprietatea:
(a) $x[i] \leq x[j]$, pentru $1 \leq i \leq q$ și $q+1 \leq j \leq n$
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. În 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 sunt mai mici decât valoarea aleasă să se afle în prima parte a tabloului iar valorile mai mari decât pivotul să se afle 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 $x[i] > \text{pivot}$ și $x[j] < \text{pivot}$

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

Sortare rapidă

Construirea unui pivot

1 7 5 3 8 2 4

Valoare
pivot

0 1 2 3 4 5 6 7

4 1 7 5 3 8 2 4

4 1 7 5 3 8 2 4

4 1 2 5 3 8 7 4

4 1 2 3 5 8 7 4

4 1 2 3 4 8 7 5

- 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 $x[i] \geq v$, $x[j] \leq v$ permit oprirea căutării 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 $x[k] \leq v$ for $k = s..i$
 $x[k] \geq v$ for $k = j..d$

Dacă $i \geq j$ atunci $x[k] \leq v$ for $k = s..i$
 $x[k] \geq v$ 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

$T(n)=n+c,$
 $c=0$ dacă $i=j$
 și
 $c=1$ dacă $i=j+1$

Deci $T(n)$ aparține lui $\Theta(n)$

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 $n \log(n)$

Deci algoritmul de sortare rapidă aparține lui $\Omega(n \log(n))$ și 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 ($\text{Prob}(q)=1/n$)
- 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

$$\begin{aligned}T_a(n) &= (T_1(n) + \dots + T_n(n)) / n \\ &= ((T_a(0) + T_a(n-1)) + (T_a(1) + T_a(n-2)) + \dots + (T_a(n-1) + T_a(0))) / n + (n+1) \\ &= 2(T_a(0) + T_a(1) + \dots + T_a(n-1)) / n + (n+1)\end{aligned}$$

Deci

$$\begin{aligned}n T_a(n) &= 2(T_a(0) + T_a(1) + \dots + T_a(n-1)) + n(n+1) \\ (n-1)T_a(n-1) &= 2(T_a(0) + T_a(1) + \dots + T_a(n-2)) + (n-1)n\end{aligned}$$

Calculând diferența dintre ultimele două egalități:

$$\begin{aligned}nT_a(n) &= (n+1)T_a(n-1) + 2n \\ T_a(n) &= (n+1)/n T_a(n-1) + 2\end{aligned}$$

Sortare rapidă

Analiza în cazul mediu.

Prin substituție inversă:

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

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

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

...

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

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

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

In cazul mediu ordinul de complexitate este $n \log(n)$

Sortare rapidă - variante

Alta variantă de construire a pivotului

3 7 5 2 1 4 8

v=3, i=1, j=2

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

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: $O(n)$

Invariant: $x[k] \leq v$ pentru $s \leq k \leq i$
 $x[k] > v$ pentru $i < k \leq j-1$

Sortare rapidă-variante

Construirea unei poziții de partitionare

3 7 5 2 1 4 8

v=3

Partiție(x[s..d])

3 7 5 2 1 4 8

i=2, j=5

v ← x[s]

i ← s-1

j ← d+1

3 1 5 2 7 4 8

i=3, j=4

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

3 1 2 5 7 4 8

i=4, j=3

ENDWHILE

RETURN j

Poziție de partiționare: 3

Ordin complexitate: O(n)

Obs: Algoritmul de partiționare este folosit în algoritmul quicksort2

Sumar

MergeSort – $O(n \log(n))$

QuickSort - $O(n \log(n))$

Merge sort

Quicksort

Divizare

$O(1)$
Determinare
indice mijloc

$O(n)$
Construire
pivot

Combinare

$O(n)$
Interclasare

$O(1)$
Concatenare

Intrebare (intermediară)

Există algoritm de sortare bazat pe compararea elementelor din tablou care să necesite mai puțin de $n \log(n)$ comparații (în cel mai defavorabil caz)?

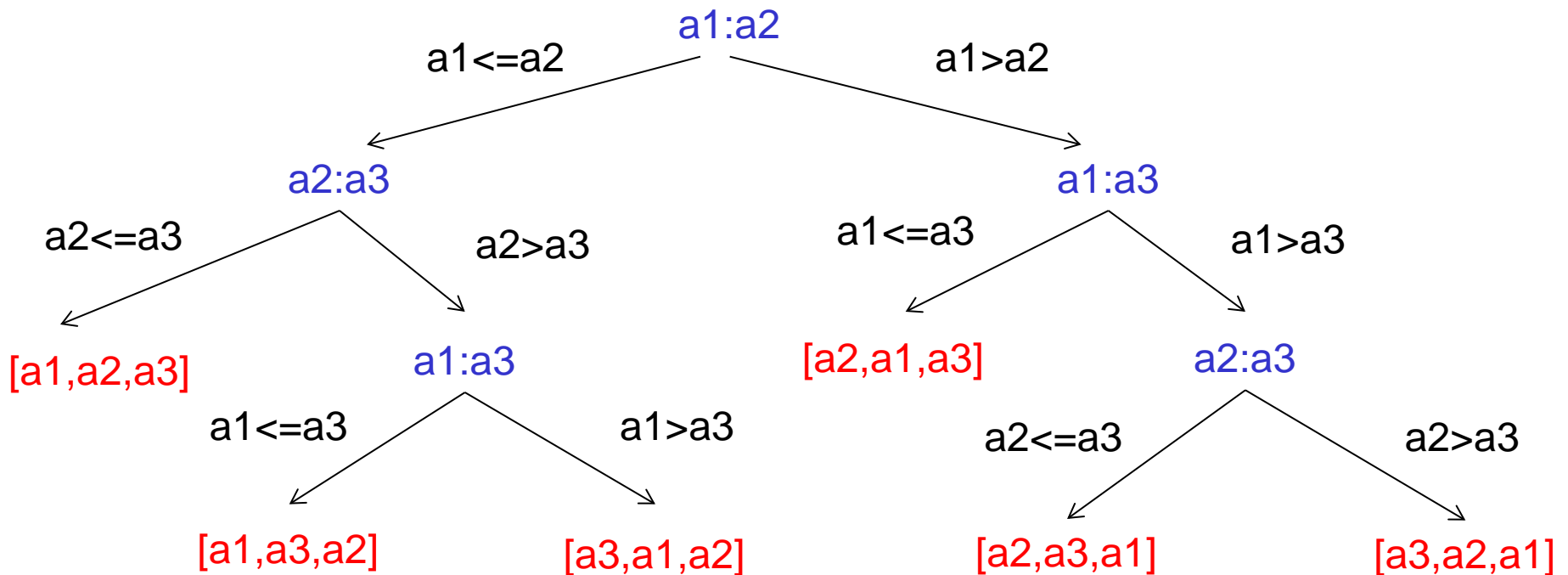
Răspuns: NU

Justificare:

- algoritmi de sortare bazați pe comparații efectuează la fiecare etapă o comparație pentru a decide dacă schimbă sau nu poziția unor elemente;
- principiul funcționării acestor algoritmi poate fi descris utilizând un **arbore binar de decizie**

Complexitatea sortării bazate pe comparații

Exemplu de arbore binar de decizie ($n=3$, $[a_1, a_2, a_3]$):



Obs: fiecare dintre cele $n!$ variante de rearanjare a listei de valori trebuie să apară în cel puțin o frunză a arborelui

Complexitatea sortării bazate pe comparații

Obs:

- fiecare dintre cele $n!$ variante de rearanjare a listei de valori trebuie să apară în cel puțin o frunză a arborelui
- Procesul de sortare corespunzător unei liste date corespunde parcurgerii unei ramuri în arbore pornind de la rădăcină până la un nod frunză
- Numărul de comparații în cazul cel mai defavorabil este corelat cu lungimea celei mai lungi ramuri din arbore = înălțimea arborelui = h
- Numărul maxim de frunze ale unui arbore binar de înălțime h este 2^h

Deci

$$n! \leq \text{nr frunze} \leq 2^h$$

Complexitatea sortării bazate pe comparații

Deci

$$n! \leq \text{nr frunze} \leq 2^h$$

Adică

$$\log n! \leq h$$

Folosind aproximarea $n! \cong \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$ (formula lui Stirling)

Rezultă că $h \geq \log n! = \Theta(n \log n)$

Alte aplicații ale divizării: selecția celui de al k-lea element

Fiind dat un tablou $x[1..n]$ (neordonat), se consideră problema determinării celui de al k-lea element în ordine crescătoare

Exemplu: $x=[4,1,5,3,8,6]$

$k=1 \Rightarrow 1$ (cel mai mic element)

$k=3 \Rightarrow 4$

$k=6 \Rightarrow 8$ (cel mai mare element)

Variante de rezolvare:

- Sortare $x[1..n] \Rightarrow O(n \log n)$
- Sortare parțială prin selecție $\Rightarrow O(kn)$ – eficient doar pentru k mic (sau apropiat de n)

Există variantă mai eficientă?

Alte aplicații ale divizării: selecția celui de al k-lea element

Idee: se folosește strategia de partiționare de la quicksort și, în funcție de relația dintre valoarea curentă a lui k și numărul de elemente din $x[s..q]$ se continuă căutarea în prima parte ($x[s..q]$) sau în a doua parte ($x[q+1..d]$)

Obs: dacă pentru apelul inițial k este între 1 și n , la fiecare dintre apeluri, k va fi între 1 și $d-s+1$ (k e rangul unui element dar nu indicele elementului)

```
Selectie(x[s..d],k)
```

```
  if s==d then return x[s]
```

```
  else
```

```
    q←partitie(x[s..d])
```

```
    r ← q-s+1
```

```
    if k≤r then return selectie(x[s..q],k)
```

```
      else return selectie(x[q+1..d],k-r)
```

```
    endif
```

```
  endif
```

Alte aplicații ale divizării: selecția celui de al k-lea element

```
Selectie(x[s..d],k)
  if s==d then return x[s]
  else
    q←partitie(x[s..d])
    r ← q-s+1
    if k≤r then return selectie(x[s..q],k)
      else return selectie(x[q+1..d],k-r)
    endif
  endif
endif
```

Cazul cel mai favorabil
(partiționare echilibrată):

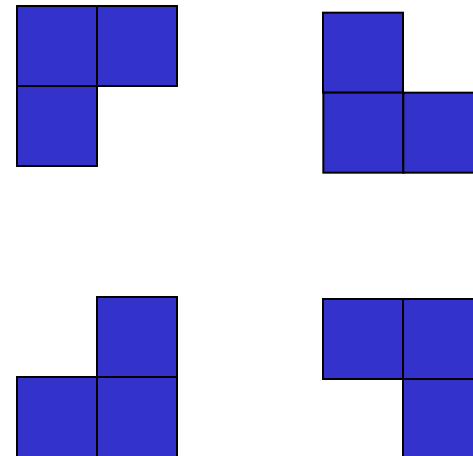
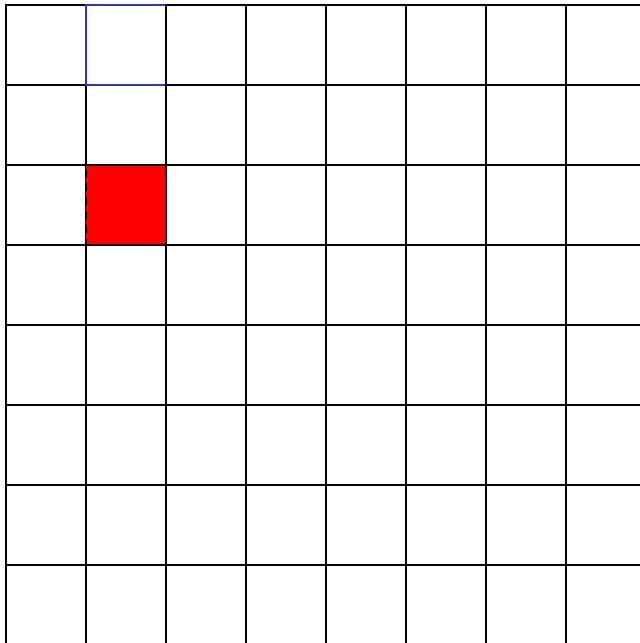
$$T(n) = \begin{cases} 0 & n=1 \\ T(n/2)+n & n>1 \end{cases}$$

⇒ (t. Master, caz 1:
m=2, k=1, d=1)

T(n) aparține lui O(n)

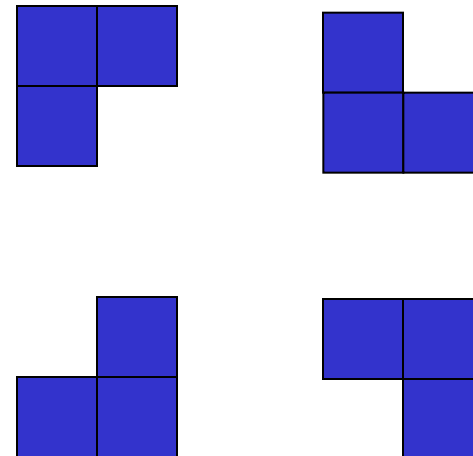
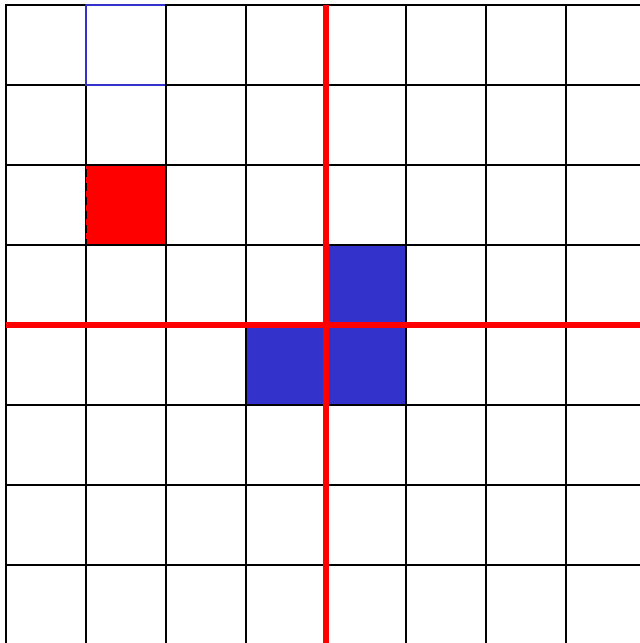
Alte aplicații ale divizării: Triomino

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)



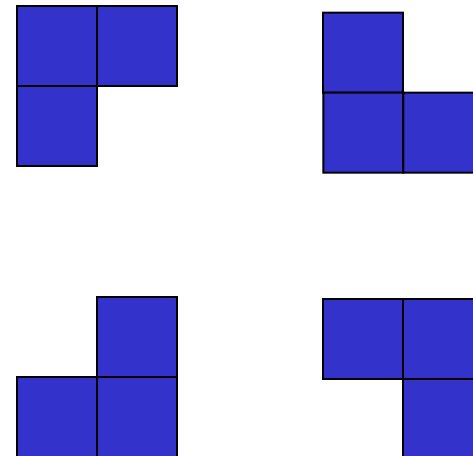
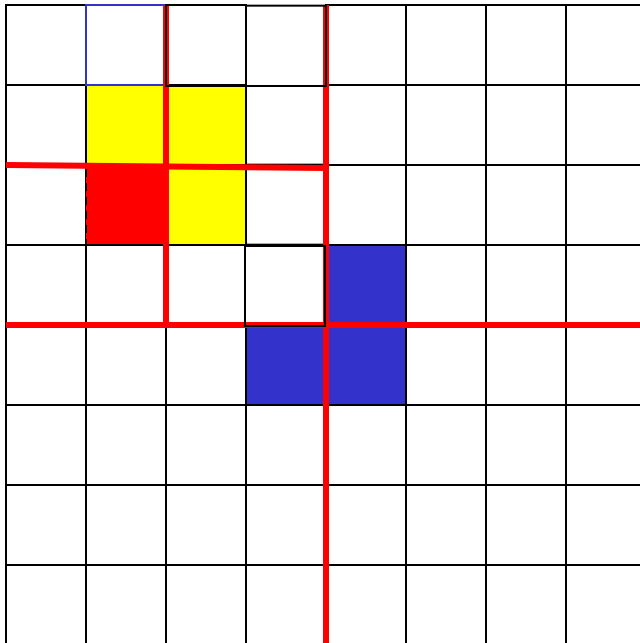
Alte aplicații ale divizării: Triomino

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



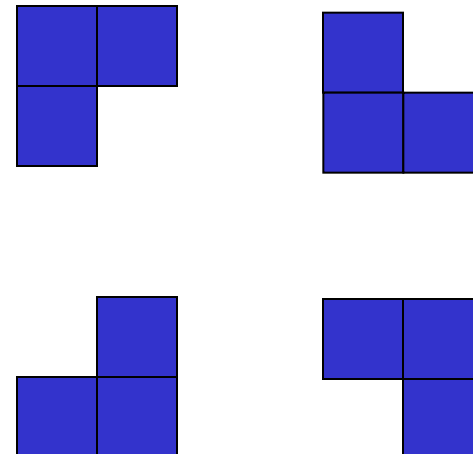
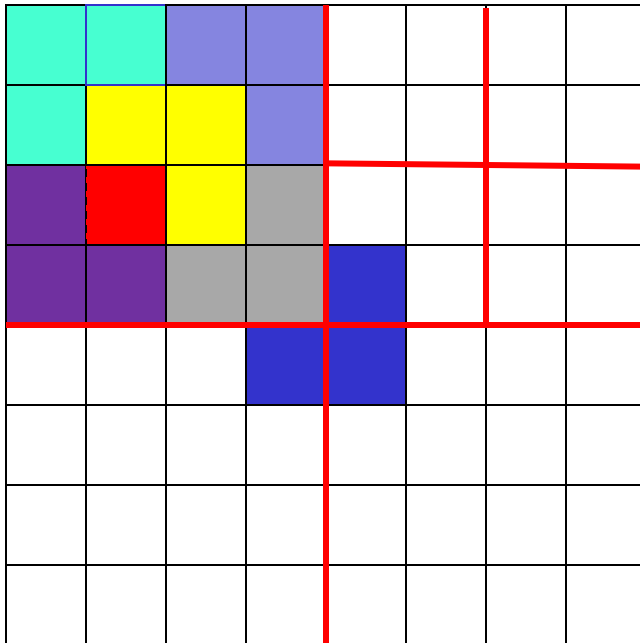
Alte aplicații ale divizării: Triomino

Idee: se aplică aceeași strategie pentru zona din stânga sus



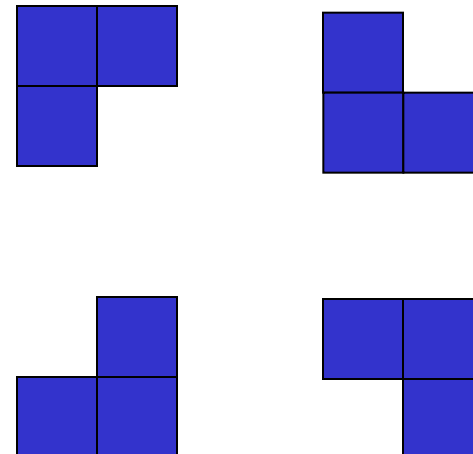
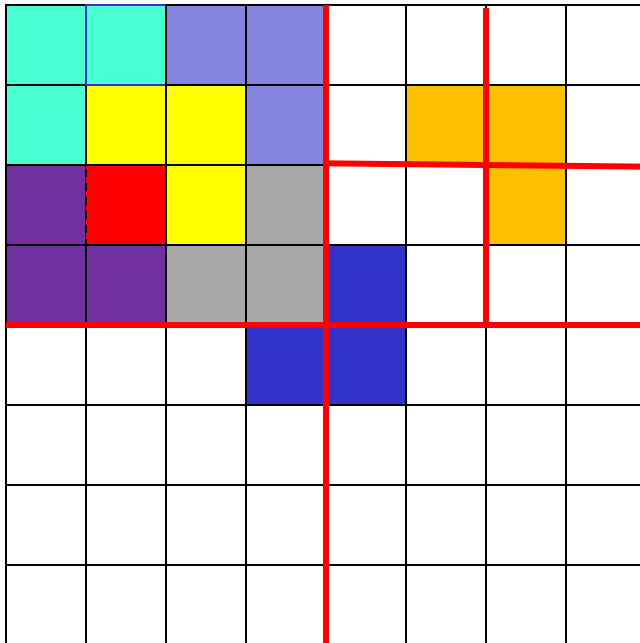
Alte aplicații ale divizării: Triomino

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



Alte aplicații ale divizării: Triomino

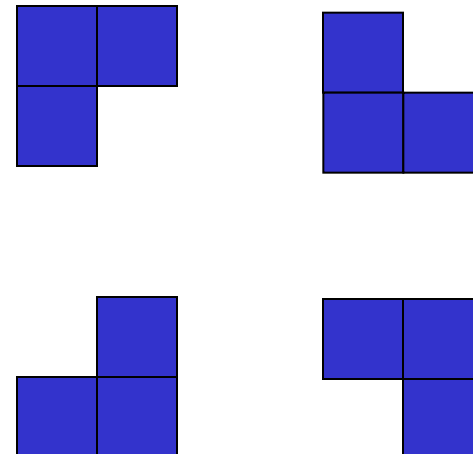
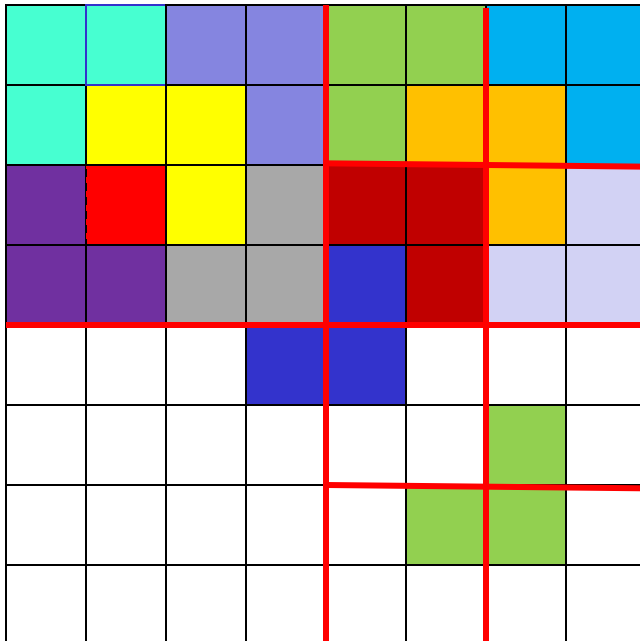
Idee: ... se aplică aceeași strategie pentru zona din dreapta sus



Alte aplicații ale divizării: Triomino

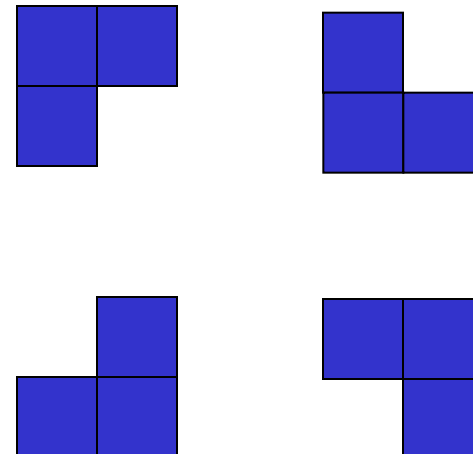
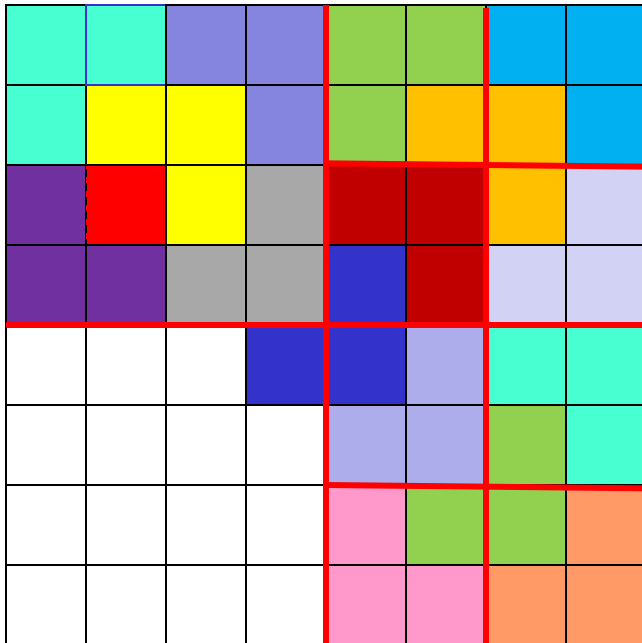
Idee: apoi pentru zona din dreapta jos și în final pentru zona din stânga jos

Obs: nu contează ordinea în care sunt rezolvate subproblemele



Alte aplicații ale divizării: Triomino

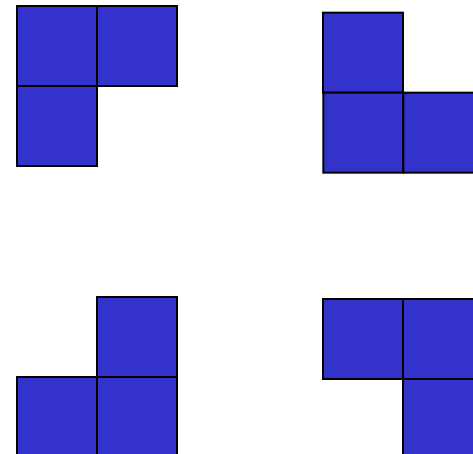
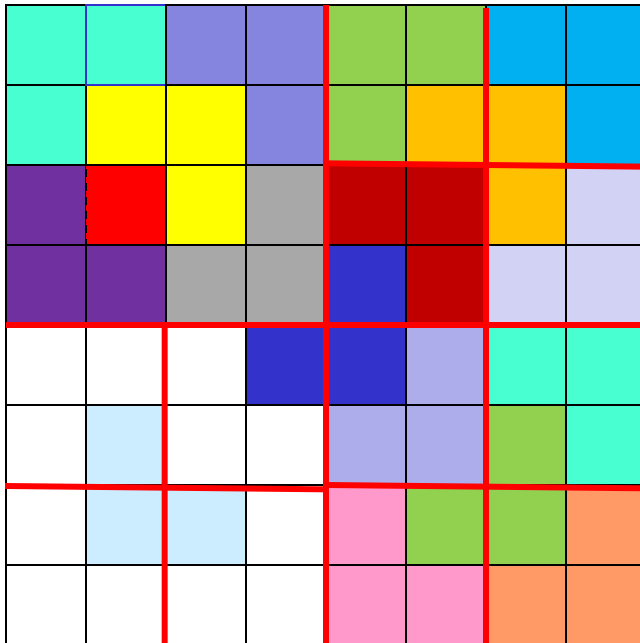
Idee: apoi pentru zona din dreapta jos



Alte aplicații ale divizării: Triomino

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

Obs: nu contează ordinea în care sunt rezolvate subproblemele



Alte aplicații ale divizării: Triomino

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 căutării locale optime

... și aplicații

Intrebare de final

Se consideră tabloul:

[6,9,8,5,3,2,4]

Care dintre valorile următoare poate fi considerată pivot în cazul în care se dorește ordonare descrescătoare folosind quicksort?

- a) 9
- b) 5
- c) 4
- d) 6