Tehnici de dezvoltare a algoritmilor (II)

Divide and conquer

Divide and Conquer

- **Impart** (*divide)* problema in una sau mai multe ***sub-probleme similare*** cu problema initiala
- Se **rezolva** fiecare sub-problema ***independent***
- Solutiile sub-problemelor se **combina** pentru a obtine solutia la problema initiala
  - se implementeaza de regula folosind *recursivitatea*
  - In general, sub-problemele generate la fiecare impartire sunt *independente (en. non-overlapping)*

Divide and Conquer: pasi

- **Divide:** daca dimensiunea problemei de intrare este prea mare pentru a rezolva problema direct, o impartim in 2 sau mai multe sub-probleme disjuncte
- **Conquer:** se folosesc apeluri recursive pentru a rezolva sub-problemele, sau se rez. direct
- **Combine:** se iau solutiile sub-problemelor si se combina pentru a obtine solutia la problema originala

Supliment: Teorema lui Master

- Pt. estimarea complexitatii in timp a alg. recursivi
- Recurente de forma:
  - $\boldsymbol{T(n) = aT(n/b) + n^c}$, $a>=1$, $b>1$
    - daca $a< b^c$: $O(n^c)$
    - daca $a=b^c$: $O(n^c \log_b n)$
    - daca $a>b^c$: $O(n^{\log_b a})$
- Alte posibilitati: metoda substitutiei (inductie), metoda arborelui de recursivitate (pt a “ghici” complexitatea)
  - vezi Cormen, cap. 4

Inmultirea intregilor

- **Problema:** Se cere inmultirea a doi intregi x si y, fiecare avand n biti
  - **Divide**: se impart x si y in biti de ordin *inalt* si *jos*

$$m = \lfloor n/2 \rfloor$$

x			y	
a	b		c	d

  - **Inmultim** partile (recursiv), si **combinam:**

$$(10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd$$

  - $T(n) = 4T(n/2) + n \rightarrow T(n) = O(n^2)$
  - nu e mai bun decat algoritmul invatat in scoala

Inmultirea intregilor

$\text{MULTIPLY}(x, y, n)$

if $n = 1$
  then return $x \cdot y$
  else $m \leftarrow \lceil n/2 \rceil$
    $a \leftarrow \lfloor x/10^m \rfloor$
    $b \leftarrow x \mod 10^m$
    $d \leftarrow \lfloor y/10^m \rfloor$
    $c \leftarrow y \mod 10^m$
    $e \leftarrow \text{MULTIPLY}(a, c, m)$
    $f \leftarrow \text{MULTIPLY}(b, d, m)$
    $g \leftarrow \text{MULTIPLY}(b, c, m)$
    $h \leftarrow \text{MULTIPLY}(a, d, m)$
return $10^{2m}e + 10^m(g + h) + f$

Inmultirea intregilor (algoritm imbunatatit)

- **Problema:** Se cere inmultirea a doi intregi x si y, fiecare avand n biti
  - **Divide**: se impart x si y in biti de ordin *inalt* si *jos*

$m = \lfloor n/2 \rfloor$

x		y	
a	b	c	d

  - Observam ca exista o posibilitate mai buna de a inmulti partile:

$$bc + ad = \underline{ac} + \underline{bd} - (a-b)(c-d)$$

  - ***$T(n) = 3T(n/2) + n \rightarrow T(n) = O(n^{\log_2 3}) \sim O(n^{1.585})$***
  - mai bun decat precedentul pt. *$n > 500$*

Inmultirea intregilor (algoritm imbunatatit)

$\textsc{FastMultiply}(x, y, n)$

if $n = 1$
  then return $x \cdot y$
  else $m \leftarrow \lceil n/2 \rceil$
    $a \leftarrow \lfloor x/10^m \rfloor$
    $b \leftarrow x \mod 10^m$
    $d \leftarrow \lfloor y/10^m \rfloor$
    $c \leftarrow y \mod 10^m$
    $e \leftarrow \textsc{Multiply}(a, c, m)$
    $f \leftarrow \textsc{Multiply}(b, d, m)$
    $g \leftarrow \textsc{Multiply}(a - b, c - d, m)$
return $10^{2m}e + 10^m(e + f - g) + f$

Ridicarea la putere

- **Problema:** Calculati a^n, unde $n \in \mathbf{N}$

```
SlowPower(a, n)

x ← a
for i ← 2 to n
      do x ← x · a
return x
```

```
FastPower(a, n)

if n = 1
   then return a
   else  x ← FastPower(a, ⌊n/2⌋)
         if n is even
            then return x · x
            else  return x · x · a
```

- Comparati cei doi algoritmi:
  - cati pasi ia fiecare?
  - cata memorie?

Ridicarea la putere

- *SlowPower* (alg. naiv):

  $\Theta(n)$

- *FastPower* (divide and conquer):

$$a^n = \begin{cases} a^{n/2} \times a^{n/2} & \text{if } n \text{ is even} \\ a^{(n-1)/2} \times a^{(n-1)/2} \times a & \text{if } n \text{ is odd} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n)$$

Cautarea binara

- **Problema:** se da vectorul $A[1...n]$] ordonat crescator; se cere gasirea elementului q in A. Daca q nu este in A, sa se returneze pozitia unde q ar putea fi inserat
- **Solutie:** cautare secventiala

```
for i = 1 to n do
    if A [i] ≥ q then
        return index i
    return n + 1
```

- Timp: $\Theta(r)$, unde r este indexul returnat. In cazul defavorabil: $\Omega(n)$, respectiv $O(1)$ in cazul favorabil

Cautarea binara

- **Solutie:** cautare ***binara*** *(recursiva)*

$BinarySearch(A, i, j, q)$
    if $i = j$ then
        return i (index)
    $k = (i + j)/2$
    if $q \leq A\,[k]$
        then return $BinarySearch\,(A, i, k, q)$
        else return $BinarySearch\,(A, k+1, j, q)$

- $T(n) = \Theta(\log n)$
- care aparitie e gasita? (in cazul in care elementul apare de mai multe ori)

Cautarea binara

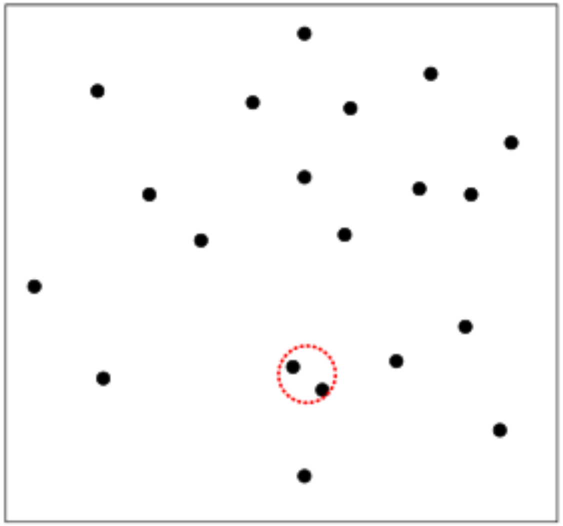
- **Solutie:** cautare ***binara*** *(iterativa)*

```
BinarySearch(A, q)
    if q > A [n]
        then return n + 1
    i = 1;
    j = n;
    while i < j do
        k = (i + j)/2
        if q ≤ A [k]
            then j = k
            else i = k + 1
    return i (the index)
```

- $T(n) = \Theta(\log n)$

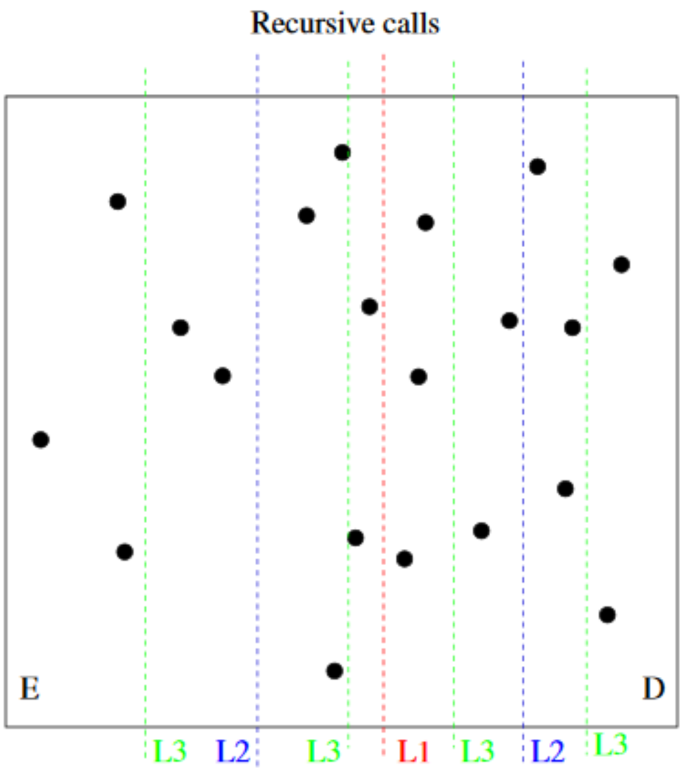
Cea mai apropiata pereche de puncte in 2D

- **Problema**: se da o multime de puncte in plan; gasiti perechea de puncte cu cea mai mica distanta Euclideana
- Presupunere: Nu exista 2 puncte cu aceeasi valoare pt. coordonata x
- Algoritmul de forta bruta: Calculeaza distanta intre oricare pereche (i, j) de puncte si o compara cu celelalte: $O(n^2)$
- 1D: sortare dupa coordinate $O(n \lg n)$
- Metoda sortarii nu generalizeaza in spatiu multi-dimensional (e.g. 2D). De ce?



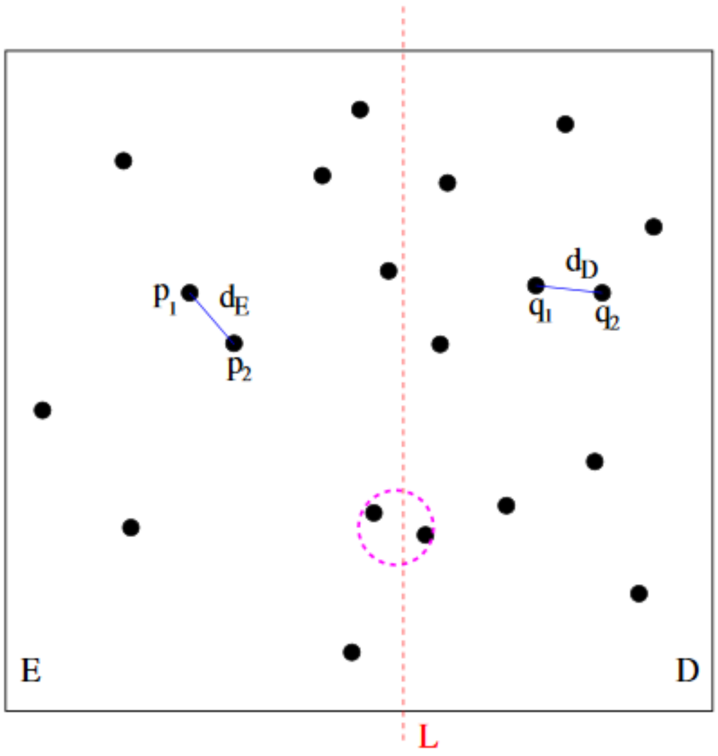
Cea mai apropiata pereche de puncte in 2D

- **Divide**: se imparte planul printr-o dreapta *L*, in 2 jumatati - *E* si *D* - avand un numar aproximativ egal de pcte (+/- 1)
- **Conquer**: recursiv se gaseste distanta minima dintre puncte aflate in fiecare partitie
- **Combine**: se iau in considerare perechi de puncte (p, q) cu $p \in E$, si $q \in D$



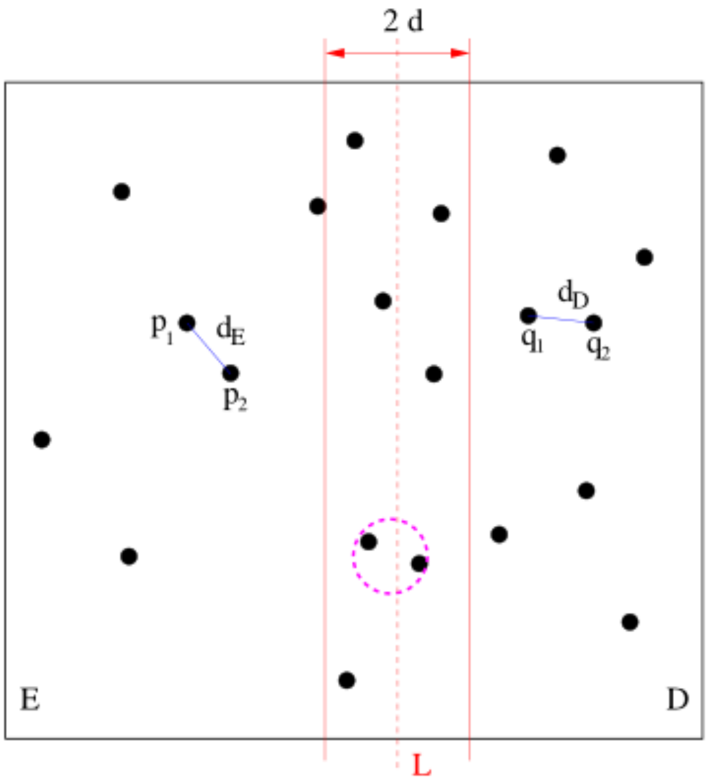
Cea mai apropiata pereche de puncte in 2D

- **Divide**:
  - se sorteaza cele n puncte dupa coord x
  - se iau $\lceil n/2 \rceil$ in stanga $\boldsymbol{L(E)}$ si $\lceil n/2 \rceil$ in dreapta $\boldsymbol{L(D)}$ lu L: $(\boldsymbol{O}(\boldsymbol{n} \lg \boldsymbol{n}))$
- **Conquer**: returneaza ***d*** = *min{**d$_E$** , **d$_D$** }* (2***T***(***n***/2))
- **Combine**: s-ar putea sa existe 2 pcte, unul in E, celalalt in D, care sunt mai apropiate decat distanta d



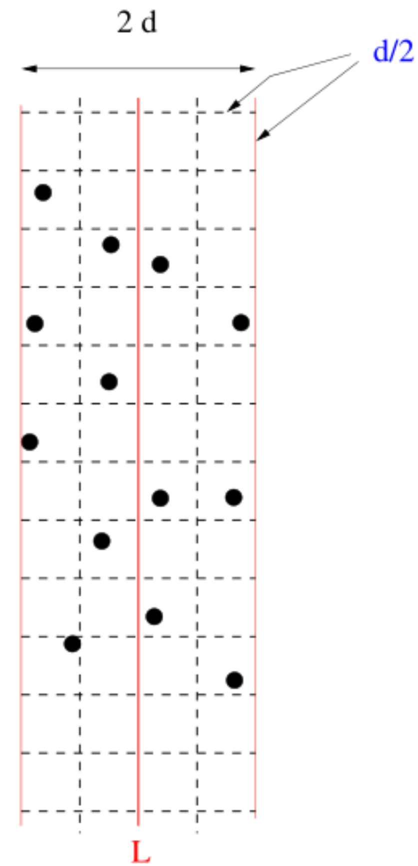
Cea mai apropiata pereche de puncte in 2D

- **Combine**:
  - se ia o banda verticala de latime 2*d* in jurul lui L
  - Orice $p \in E$ si $q \in D$ a.i. $d(p, q) \leq d$ trebuie sa se gaseasca in aceasta banda (de ce?)
  - S-ar putea sa gasim mai multe pcte in interiorul benzii
  - Pt a gasi perechea p,q cu cele mai apropiate puncte: sortam crescator punctele dupa coordonata y, $Y = \{y_1, y_2, \ldots y_m\}$
  - Cost: $O(n \lg n)$



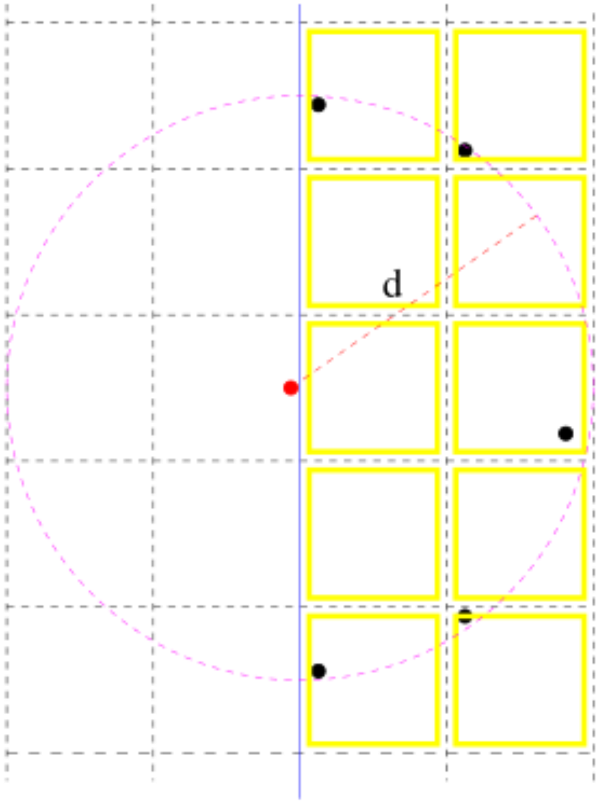
Cea mai apropiata pereche de puncte in 2D

- **Combine**:
  - Se considera un grid avand dimensiunea celulei $d/2$ in interiorul benzii
    - Exista cel mult 1 punct in interiorul fiecarei celule $d/2 \times d/2$ (diagonala celulei = $d/\sqrt{2} < d$)
    - 2 pcte aflate la o distanta mai mare de > 2 randuri au sigur o distanta $> d$ (distanta intre 2 pcte in celule consecutive este $d\sqrt{(5/4)} = 1.118d$.)
    - 2 pcte aflate la o distanta mai mare de > 2 coloane au sigur o distanta $> d$ (acelasi argument ca inainte)



Cea mai apropiata pereche de puncte in 2D

- Cate celule poate sa influenteze 1 punct?
- Pentru fiecare punct in secventa sortata $Y = \{y_1, y_2, \ldots y_m\}$, pornind de la y_1 trebuie sa exploram doar distanta intre y_i si cele mai apropiate 10 puncte din Y.
  - $\boldsymbol{d}(\boldsymbol{y_i}, \boldsymbol{y_j}) \leq \boldsymbol{d}$ if $|\boldsymbol{i} - \boldsymbol{j}| \leq 10$
- Deci, fiecare punct din banda trebuie comparat cu cele mai apropiate 10 din Y, generand un cost total de $10n$.



Cea mai apropiata pereche de puncte in 2D

Closest-Pair $(p_1, \ldots, p_n)$

Sort by the x–coordinate to compute L

d_1 =***Closet-Pair***(E)

d_2 =***Closet-Pair***(D)

$d = \min\{d_1, d_2\}$

Delete points > d from L

Sort the remaining points by y–coordinate to form listed Y

Scan in order Y list computing the distance with next 11 elements

If any of those distances is < d update d

- $T(n) = 2T(n/2) + O(n \lg n) = O(n \lg^2 n)$

Inmultirea matricilor

- **Problema**: Inmultiti doua matrici $\boldsymbol{n} \times \boldsymbol{n}$:
- Metoda standard: $$C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$$
- Asta ia $\boldsymbol{O}(\boldsymbol{n})$ pentru fiecare element al lui $\boldsymbol{C}$, rezultand un efort total de $\boldsymbol{O}(\boldsymbol{n}^3)$ pt. calculul lui $\boldsymbol{C}$.
- Algoritm cunoscut depe vremea lui Gauss's
- Strassen (1969) a propus un algoritm mai eficient decat limita de $\boldsymbol{O}(\boldsymbol{n}^3)$.
- Metoda bazata pe *divide and concquer*, bazat pe alegerea potrivita a sub-matricilor de inmultit

Inmultirea matricilor

- Fie A, B doua matrici $n \times n$. Se doreste calcularea matricei $C = AB$

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

Intrarile a_{ij} sunt matrici de dimensiune $n/2 \times n/2$

Inmultirea matricilor

- Matricea produs poate fi scrisa ca:

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{21}b_{12} + a_{22}b_{22}$$

- Recurenta pt acest algoritm divide and conquer:

$$\boldsymbol{T}(\boldsymbol{n}) = 8\boldsymbol{T}(\boldsymbol{n}/2) + \boldsymbol{O}(\boldsymbol{n}^2)$$

- Dar asta duce tot la:

$$\boldsymbol{T}(\boldsymbol{n}) = \boldsymbol{O}(\boldsymbol{n}^3)!$$

Algoritmul lui Strassen

- Submatricile de inmultit alese de Strassen

$$
\begin{aligned}
P_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\
P_2 &= (a_{21} + a_{22})b_{11} \\
P_3 &= a_{11}(b_{12} - b_{22}) \\
P_4 &= a_{22}(b_{21} - b_{11}) \\
P_5 &= (a_{11} + a_{12})b_{22} \\
P_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\
P_7 &= (a_{12} - a_{22})(b_{21} + b_{22})
\end{aligned}
$$

Algoritmul lui Strassen

$$\begin{aligned} c_{11} &= P_1 + P_4 - P_5 + P_7 \\ c_{12} &= P_3 + P_5 \\ c_{21} &= P_2 + P_4 \\ c_{22} &= P_1 + P_3 - P_2 + P_6 \end{aligned}$$

- Recurenta pentru acest algoritm:

$$T(n) = 7T(n/2) + O(n^2).$$

- …care se rezolva la:

$$T(n) = O(n^{\log_2 7}) \; = \; O(n^{2.81}).$$

Sortarea prin interclasare (Mergesort)

- **Problema**: sortati o secventa de numere
- Algoritm *Divide-and-conquer*: se imparte secventa in 2 jumatati, se apeleaza recursiv procedura pe fiecare jumatate, se interclaseaza cele doua sub-secvente sortate

mergesort(***A***, ***left, right***)
    if $\boldsymbol{n} > 1$:
        return ***merge***(***mergesort***(***A***, 1, $\lfloor n/2 \rfloor$)), ***mergesort***(***A***, $\lfloor n/2 \rfloor + 1$, ***n***))
    else:
        return ***A***

Mergesort

merge(x[1…k], y[1…l])

  if $\boldsymbol{k} = 0$: return $\boldsymbol{y}[1\ldots \boldsymbol{l}]$

  if $\boldsymbol{l} = 0$: return $\boldsymbol{x}[1\ldots \boldsymbol{k}]$

  if $\boldsymbol{x}[1] \leq \boldsymbol{y}[1]$:

    return $\boldsymbol{x}[1] \circ$ ***merge***$(\boldsymbol{x}[2\ldots \boldsymbol{k}], \boldsymbol{y}[1\ldots \boldsymbol{l}])$

  else:

    return $\boldsymbol{y}[1] \circ$ ***merge***$(\boldsymbol{x}[1\ldots \boldsymbol{k}], \boldsymbol{y}[2\ldots \boldsymbol{l}])$

$\circ$ = concatenare

merge: cantitate constanta de efort per apel recursiv (daca spatiul pentru vector este alocat in avans), pentru un timp de rulare total de $\boldsymbol{O}(\boldsymbol{k} + \boldsymbol{l})$.

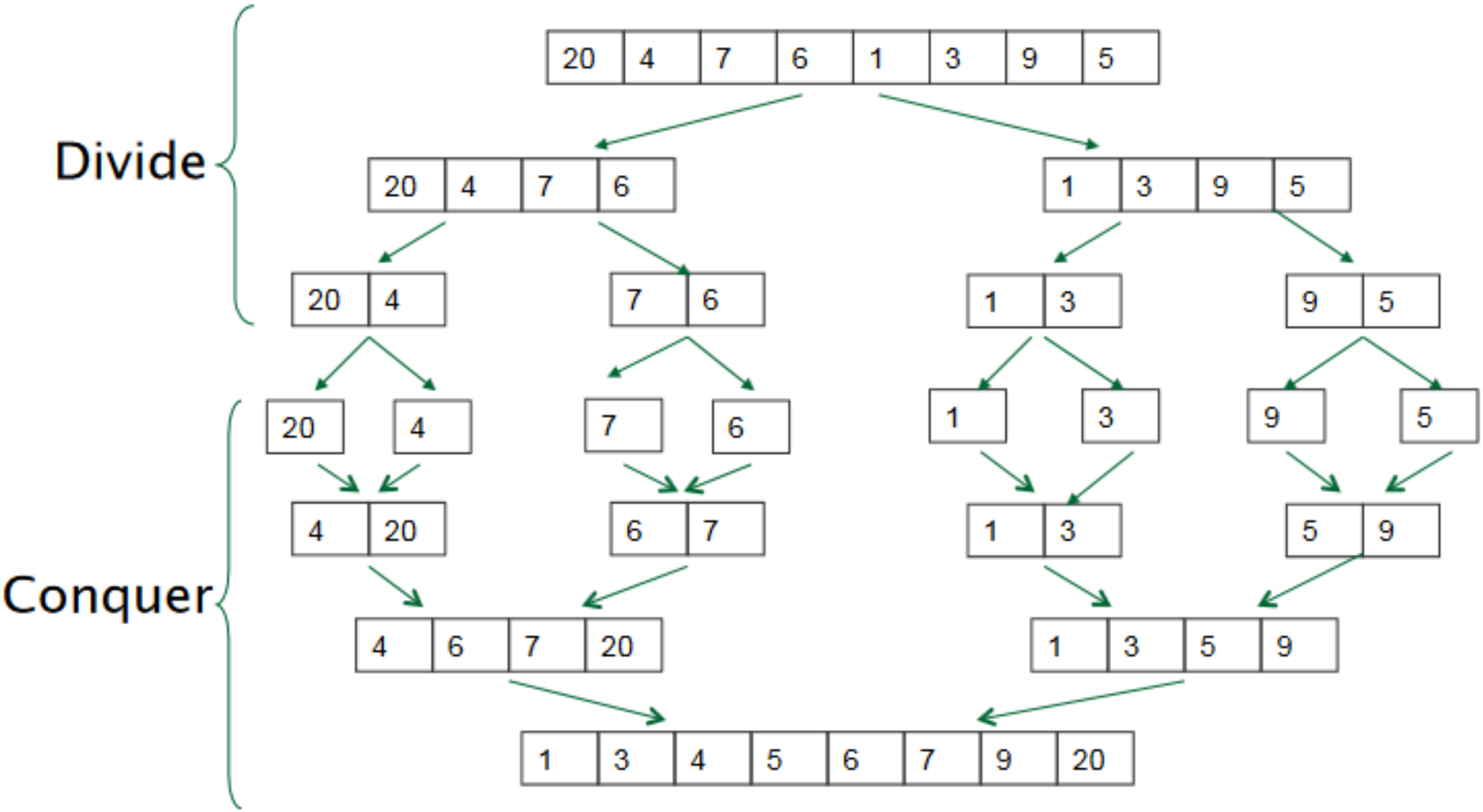
mergesort: $\boldsymbol{T}(\boldsymbol{n}) = 2\boldsymbol{T}(\boldsymbol{n}/2) + \boldsymbol{O}(\boldsymbol{n})$, or $\boldsymbol{O}(\boldsymbol{n} \log \boldsymbol{n})$

Mergesort

- La fiecare moment, avem o multime de vectori “activi” - initial - vectori compusi dintr-un singur element - care sunt interclasati pentru a da urmatoarea colectie de vectori activi
- Acesti vectori se pot stoca intr-o coada, si procesati prim eliminarea repetata a 2 vectori de la inceputul cozii (dequeue), interclasarea loc, si inserarea rezultatului la finalul cozii (enqueue)

***iterative-mergesort*(*A*, *n*)**
    Q = [ ] (empty queue)
    for ***i*** = 1 to ***n***:
        ***enqueue*(*Q*,** ***A*[*i*])**
    while |***Q***| > 1:
        ***enqueue*(*Q*,*merge*(*dequeue*(*Q*),** ***dequeue*(*Q*)))**
    return ***dequeue*(*Q*)**

Mergesort


Divide
Conquer
20 4 7 6 1 3 9 5
20 4 7 6
1 3 9 5
20 4
7 6
1 3
9 5
20
4
7
6
1
3
9
5
4 20
6 7
1 3
5 9
4 6 7 20
1 3 5 9
1 3 4 5 6 7 9 20

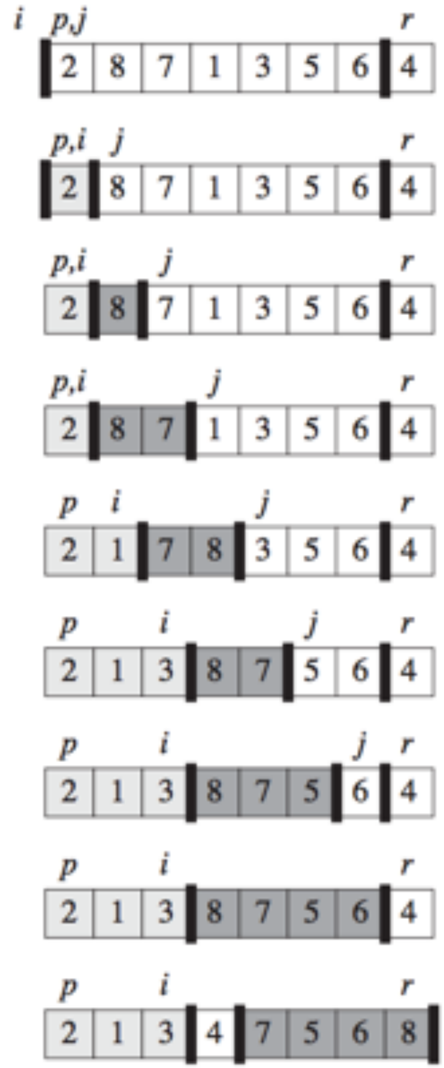
Quicksort - Sortarea rapida

- ***Divide***: se impart elementele vectorului $A[p \ldots r]$ in $A[p \ldots q]$ si $A[q+1 \ldots r]$ astfel incat toate elementele din $A[p \ldots q]$ sunt mai mici sau egale decat toate elementele din $A[q+1 \ldots r]$, pentru un q.
- ***Conquer***: se sorteaza $A[p \ldots q]$ si A$[q+1 \ldots r]$ independent (*recursiv*)
- ***Combine***: sortarea se realizeaza *in-place*, la combinare nu este nevoie de nimic

Quicksort

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```



Bibliografie

- Th. Cormen et al.: Introduction to Algorithms, cap. 4, cap. 2.3