

# Rendező algoritmusok

Ionescu Klára

[clara@cs.ubbcluj.ro](mailto:clara@cs.ubbcluj.ro)

# Rendezések

## 1. Összehasonlításon alapuló rendezések $O(n^2)$ :

- 1.a. Egyszerű felcseréléses rendezés
- 1.b. Minimumkiválasztásos rendezés
- 1.c. Buborékos rendezés (*bubblesort*)
- 1.d. Beszűrő rendezés (*insertsort*)
- 1.e. Számlálva szétoztó rendezés (válogatás)

## 2. Lineáris rendezések $O(n)$ :

- 2.a. Láda rendezés (*binsort*)
- 2.b. Számjegyes rendezés (*radixsort*)

## 3. Oszd meg és uralkodj módszert alkalmazó rendezések $O(n \log n)$ :

- 3.a. Gyorsrendezés (*quicksort*)
- 3.b. Összefésülő rendezés (*mergesort*)

# Bevezetés

Legyen az  $(a_1, a_2, \dots, a_n)$  sorozat.

**Rendezett sorozat:** a bemeneti sorozat olyan  $(a_1', a_2', \dots, a_n')$  permutációja, amelyben  
 $a_1' \leq a_2' \leq \dots \leq a_n'$ .

**D. Knuth** „A számítógép programozásának művészete” című könyvében, (III. kötet:  
Keresések és rendezések)

**33 rendező algoritmust ír le...**

# Stabil és helyben rendezés

## *Definíció*

Egy rendezésről azt mondjuk, hogy **helyben** (*in place*) rendez, ha konstans méretű pluszmemóriát használ, vagyis memóriabonyolultsága  $\Theta(1)$ .

## *Definíció*

Egy rendezésről azt mondjuk, hogy **stabil**, ha az egyenlő kulcsú elemek ugyanolyan sorrendben szerepelnek a rendezett sorozatban, mint az eredetiben. Ennek akkor van jelentősége, ha nem csak a kulcsokat kell rendezni, hanem más a kulcsokhoz „kötött” adatokat is.

# Buborékrendezés

- A rendezés során ellenőriznünk kell az elemek sorrendjét.
- A sorozat akkor rendezett, ha  $a_1 \leq a_2 \leq \dots \leq a_{n-1} \leq a_n$ .
- **Páronként** hasonlítjuk össze az elemeket, és ha a sorrend nem megfelelő, akkor az illető két elemet **felcseréljük**.
- **Ha volt csere**, a vizsgálatot újrakezdjük.
- Az algoritmus véget ér, ha az elemek páronként a megfelelő sorrendben találhatók, vagyis a sorozat rendezett.
- A buborékrendezés *helyben* dolgozik (a bemeneti sorozaton kívül csak néhány segédváltozóra van szükség) és stabil (ha megjelölnénk például, az első 9-est \*-gal, a másodikat \*\*-gal, a rendezett sorozatban egymás után következnének)

# Példák

a) Legyen a következő sorozat:

**2, 3, 5, 9, 10**

$\leq \leq \leq \leq$

Az ellenőrzés végeredményeként kiderül, hogy a sorozat rendezett.

b) Ha a következő sorozatot kell rendezni:

**2, 3, 5, 10, 9**

$\leq \leq \leq \geq$

Csak az utolsó két szám nincs megfelelő helyen.

Ha felcseréljük ezt a két számot:

**2, 3, 5, 9, 10**

$\leq \leq \leq \leq$

Egyetlen csere után rendezett sorozathoz jutottunk.

# Példa

c)

**5, 2, 3, 10, 9**

$\geq$

Miután felcseréljük az 5-öst a 2-sel, a sorozat a következőképpen alakul:

**2, 5, 3, 10, 9**

$\leq \geq$

Felcseréljük az 5-öst a 3-sal:

**2, 3, 5, 10, 9**

$\leq \leq \leq \geq$

Felcseréljük a 9-et a 10-sel:

**2, 3, 5, 9, 10**

$\leq \leq \leq \leq$

Ezen példa alapján úgy tűnik, hogy egyszeri bejárás és a megfelelő cserék után a sorozat rendezetté vált.

# Példa

**10, 9, 2, 3, 5**

$\geq$

**9, 10, 2, 3, 5**

$\geq$

**9, 2, 10, 3, 5**

$\geq$

**9, 2, 3, 10, 5**

$\geq$

**9, 2, 3, 5, 10**

$\geq$

Egyszeri bejárás után a sorozat  
nem rendezett.

**2, 9, 3, 5, 10**

$\geq$

**2, 3, 9, 5, 10**

$\geq$

**2, 3, 5, 9, 10**

Vége a második bejárásnak is.

„Nem biztos”, hogy a sorozat rendezett...

**2, 3, 5, 9, 10**

A harmadik ellenőrzés után eldönthetjük,  
hogy a sorozat rendezett.



**Algoritmus** buborékRendezés\_1( $n$ ,  $a$ ):

*// Bemeneti adatok: az  $n$  elemű  $a$  sorozat*

*// Kimeneti adatok: az  $n$  elemű rendezett  $a$  sorozat*

**Ismételd**

rendben  $\leftarrow igaz$

**Minden**  $i = 1, n - 1$  végezd el:

Ha  $a_i > a_{i+1}$  akkor

id  $\leftarrow a_i$

$a_i \leftarrow a_{i+1}$

$a_{i+1} \leftarrow id$

rendben  $\leftarrow hamis$

**vége**(ha)

**vége**(minden)

**ameddig** rendben

*// kilépünk, ha nem volt csere*

**Vége**(algoritmus)

# Optimalizálás

## Észrevételek

- A sorozat első bejárása után **legalább az utolsó elem a helyére kerül!**
  - A ciklusmag minden újabb végrehajtása után, jobbról balra haladva **újabb elemek kerülnek a megfelelő helyre!**
- ⇒ Ha az első lépésnél az ***i*** ciklusváltozó ***n – 1***-ig nő, akkor a következő lépésekben ez a felső határ legalább **1-gyel csökkenthető.**

**Algoritmus** buborékRendezés\_2( $n$ ,  $a$ ):

*// Bemeneti adatok: az  $n$  elemű  $a$  sorozat*

*// Kimeneti adatok: az  $n$  elemű rendezett  $a$  sorozat*

$nn \leftarrow n - 1$

**Ismételd**

$rendben \leftarrow igaz$

**Minden**  $i = 1, nn$  **végezd el:**

**Ha**  $a_i > a_{i+1}$  **akkor**

$rendben \leftarrow hamis$

$felcserél(a_i, a_{i+1})$

**vége(ha)**

**vége(minden)**

$nn \leftarrow nn - 1$

**ameddig**  $rendben$

**Vége(algoritmus)**

# Újabb észrevétel

- Az is előfordulhat, hogy a sorozat végén levő elemek már a megfelelő sorrendben vannak, és így azokat már nem kell rendeznünk.
- Észrevesszük, hogy ***elegendő a sorozatot csak az utolsó csere helyéig vizsgálni.***

**Algoritmus** buborékRendezés\_3( $n$ ,  $a$ ):

*// Bemeneti adatok: az  $n$  elemű  $a$  sorozat*

*// Kimeneti adatok: az  $n$  elemű rendezett  $a$  sorozat*

$k \leftarrow n$

**Ismételd**

$nn \leftarrow k - 1$

$rendben \leftarrow igaz$

**Minden**  $i = 1, nn$  **végezd el:**

**Ha**  $a_i > a_{i+1}$  **akkor**

$rendben \leftarrow hamis$

        felcserél( $a_i, a_{i+1}$ )

$k \leftarrow i$

*// az utolsó csere helye*

**vége**(ha)

**vége**(minden)

**ameddig**  $rendben$

**Vége**(algoritmus)

# Következtetések

***Elegendő a sorozatot csak az első és az utolsó csere helye között vizsgálni.***

## ***Megjegyzések***

- Ha a sorozat sok elemet tartalmaz, a buborékrendezés, még a fenti javításokkal sem tartozik a hatékony rendezési módszerek közé. Átlagos bonyolultsága:  **$\Theta(n^2)$**
- Ha a sorozat már eleve rendezett, akkor ***egyetlen bejárás után az algoritmus véget ér***, de gyors akkor is, ha a sorozat csak „majdnem” rendezett; a legjobb eset időbonyolultsága:  **$\Omega(n)$**
- ***Ha a sorozatot bejárjuk fordított irányban, és – az előbbi algoritmust a csökkenő rendezés érdekében alkalmazzuk, tovább javítható a futási idő!***

```
Algoritmus buborékRendezés_4(n, a):                                // koktélrendezés
    régiBal ← 1                                                    // a vizsgálandó sorozat bal széle
    régiJobb ← n - 1                                              // a jobb szél indexe
    Ismételd
        rendben ← igaz
        jobb ← 0                                                  // az utolsó csere helye
        Minden i = régiBal, régiJobb végezd el:
            Ha  $a_i > a_{i+1}$  akkor
                rendben ← hamis
                felcserél( $a_i$ ,  $a_{i+1}$ )
                Ha  $i > jobb$  akkor
                    jobb ← i                                        // megjegyeztük az utolsó csere helyét
                vége(ha)
        vége(ha)
    vége(minden)
```

```
Ha nem rendben akkor                                // volt csere
    régiJobb ← jobb                                // aktualizáljuk a régiJobb értékét
    bal ← n // a bal szél
    rendben ← igaz
    Minden i = régiJobb, régiBal, -1 végezd el:
        Ha  $a_i > a_{i+1}$  akkor
            rendben ← hamis
            felcserél( $a_i, a_{i+1}$ )
            Ha  $i < bal$  akkor
                bal ← i
            vége(ha)
        vége(ha)
    vége(minden)
    régiBal ← bal                                // aktualizáljuk a régiBal értékét
    vége(ha)
ameddig rendben
Vége(algoritmus)
```



# Egyszerű felcseréléses rendezés

- Hasonlít a buborékrendezéshez
- ***De kötelezően elvégez minden páronkénti összehasonlítást.***
- Ha egy elempár sorrendje nem megfelelő, felcseréli őket.
- A külső ciklus következő lépését mindig az aktuális elemtől folytatja, mivel az első lépésben az első helyre a sorozat legkisebb eleme kerül, a második lépésben második helyre kerül az aktuális részsorozat legkisebb eleme és így tovább.

**Algoritmus felcserélésesRendezés(n, a):**

*// Bemeneti adatok: az n elemű a sorozat*

*// Kimeneti adatok: az n elemű rendezett a sorozat*

**Minden  $i = 1, n - 1$  végezd el:**

**Minden  $j = i + 1, n$  végezd el:**

**Ha  $a_i > a_j$  akkor**

**felcserél( $a_i, a_j$ )**

**vége(ha)**

**vége(minden)**

**vége(minden)**

**Vége(algoritmus)**

# Számlálva szétosztó rendezés

- Minden elemet **összehasonlítunk** az összes többi elemmel
- **Megszámloljuk** ( $k$ -ban) azokat az elemeket, amelyek kisebbek mint a vizsgált elem.
- Így **meghatározzuk az illető elem sorszámát egy új - rendezett sorozatban**.
- Ha a feladat követelményei szerint a rendezett sorozat az eredeti tömbben szükséges, akkor az algoritmus végén a régi tömbváltozót **felülírjuk** a rendezett sorozatot tároló változóval.

**Algoritmus válogatásosRendezés(n, a):**

*// Bemeneti adatok: az n elemű a sorozat*

*// Kimeneti adatok: az n elemű rendezett a sorozat*

**Minden  $i = 1, n$  végezd el:**

$k \leftarrow 0$

**Minden  $j = 1, n$  végezd el:**

**Ha  $a_i > a_j$  akkor**

$k \leftarrow k + 1$  *// megszámoljuk az  $a_i$ -nél kisebb elemeket*

**vége(ha)**

**vége(minden)**

$\text{rendezettSor}_{k+1} \leftarrow a_i$

*//  $a_i$  a rendezett sorozat megfelelő helyére kerül*

**vége(minden)**

**Másol(a, rendezettSor)**

*// a rendezett sorozatot rámásoljuk az eredetire*

**Vége(algoritmus)**

# Megjegyzés

- Ez az algoritmus csak akkor alkalmazható, ha a sorozat csak *különböző* elemeket tartalmaz!
- Írjátok le az előbbi algoritmus azon változatát, amely akkor is alkalmazható, ha léteznek *azonos* értékű elemek is!

# Megjegyzés

A fenti módszer hátrányai:

- csak különböző elemek esetén alkalmazható és
- memóriapazarló.

***Kiküszöbölhetjük: a kiválasztott elemet az adott sorozaton belül a végleges helyére tesszük.***

# Minimum kiválasztásra épülő rendezés

- Növekvő sorrendbe rendezés esetén **kiválasztjuk a sorozat legkisebb elemét.**
- Ezt **az első helyre tesszük** úgy, hogy felcseréljük az első helyen található elemmel.
- A következő lépésben hasonlóan járunk el, de a minimumot a második helytől kezdődően keressük.
- A továbbiakban ugyanezt tesszük, míg a sorozat végére nem érünk.

**Algoritmus minimumKiválasztásraÉpülőRendezés(n, a):**

*// Bemeneti adatok: az n elemű a sorozat*

*// Kimeneti adatok: az n elemű rendezett a sorozat*

**Minden  $i = 1, n - 1$  végezd el:**

**ind\_min  $\leftarrow i$**

**Minden  $j = i + 1, n$  végezd el:**

**Ha  $a_{\text{ind\_min}} > a_j$  akkor**

**ind\_min  $\leftarrow j$**

**vége(ha)**

**vége(minden)**

**felcserél( $a_i, a_{\text{ind\_min}}$ )**

**vége(minden)**

**Vége(algoritmus)**



# Beszűrő rendezés

- A beszűrő rendezés hatékony algoritmus kis számú elem rendezésére.
- A beszűrő rendezés úgy dolgozik, ahogyan *bridzsezés* közben a kezünkben lévő lapokat rendezzük...
- A bemenő elemek *helyben* rendeződnek: a számokat az algoritmus az adott tömbön belül rakja a helyes sorrendbe, belőlük bármikor legfeljebb csak állandónyi tárolódik a tömbön kívül.

**Algoritmus beszűrőRendezés( $n$ ,  $a$ ):**

*// Bemeneti adatok: az  $n$  elemű  $a$  sorozat*

*// Kimeneti adatok: az  $n$  elemű rendezett  $a$  sorozat*

**Minden  $j = 2$ ,  $n$  végezd el:**

$\text{segéd} \leftarrow a_j$

$i \leftarrow j - 1$

**Amíg  $i > 0$  és  $a_i > \text{segéd}$  végezd el:**

$a_{i+1} \leftarrow a_i$

$i \leftarrow i - 1$

**vége(amíg)**

$a_{i+1} \leftarrow \text{segéd}$

*// beszűrjük  $a_j$ -t az  $a_{1..j-1}$  rendezett sorozatba*

**vége(minden)**

**Vége(algoritmus)**

# Rendezések lineáris időben

A bemutatott algoritmusok  $O(n^2)$  időben rendeznek  $n$  elemet.

⇒ időigényesek: ahhoz, hogy egy **1000** elemű sorozatot rendezzünk, körülbelül **1 millió összehasonlítást** végeznek.

- Közös tulajdonságuk: ***a rendezéshez csak a bemeneti tömb elemein történő összehasonlításokat használják.***
- A **lineáris** bonyolultságú algoritmusok nem az összehasonlítást használják a rendezéshez, hanem ***kihasználják a rendezendő sorozat bizonyos tulajdonságait.***
- ***Csak adott tulajdonságú sorozatokkal végezhetjük.***
  - Egy lineáris algoritmus az elemek számosságával arányos számú műveletet végez a bemenő adatokon: **1000** elem rendezéséhez  $c \cdot 1000$  műveletet hajt végre, ahol  $c$  egy konstans.

# Leszámláló rendezés (ládarendezés = binsort)

- Feltételezzük, hogy az  ***$n$  bemeneti elem mindegyike 1 és  $k$  közötti egész szám, ahol  $k$  egy egész szám.***
- Szükségünk lesz egy segéd tömbre:
- Az  ***$i$*** -edik elem azt tartja nyilván, hogy  ***$hány darab  $i$ -vel egyenlő elemet találtunk az eredeti tömbben.$***
- A lineáris feldolgozás után felülírjuk az eredeti tömb elemeit a segéd tömb elemeinek értékei alapján.

# Megjegyzések

- $k$  nem lehet túlságosan nagy, mivel *a segéd tömb mérete pontosan  $k$  lesz.*
- A rendezendő elemek értékeinek feltétlenül *sorszámozható típusú*aknak kell lenniük.
- Elégséges, ha az értékek egy bizonyos  $[x, y]$  intervallumhoz tartoznak, azzal a feltétellel, hogy lehető legyen a segéd tömb indexelése  $x$ -től  $y$ -ig.

**Algoritmus** Leszámlálás( $n, a$ ):

*// Bemeneti adatok: az  $n$  elemű  $a$  sorozat*

*// Kimeneti adatok: az  $n$  elemű rendezett  $a$  sorozat*

**Minden**  $i = 1, k$  **végezd el:**

$\text{segéd}_i \leftarrow 0$

**vége(minden)**

**Minden**  $j = 1, n$  **végezd el:**

$\text{segéd}_{aj} \leftarrow \text{segéd}_{aj} + 1$

**vége(minden)**

$q \leftarrow 0$

**Minden**  $i = 1, k$  **végezd el:** *// két Minden egymásba ágyazva, de a*

**Minden**  $j = 1, \text{segéd}_i$  **végezd el:** *// lépések száma  $n$ , mivel*

$q \leftarrow q + 1$  *// a segéd tömbnek  $k$  eleme van*

$a_q \leftarrow i$  *// ezeknek a  $\text{segéd}_i$  értékeknek összege  $n$*

**vége(minden)**

**vége(minden)**

**Vége(algoritmus)**

# Számjegyes rendezés (radixsort)

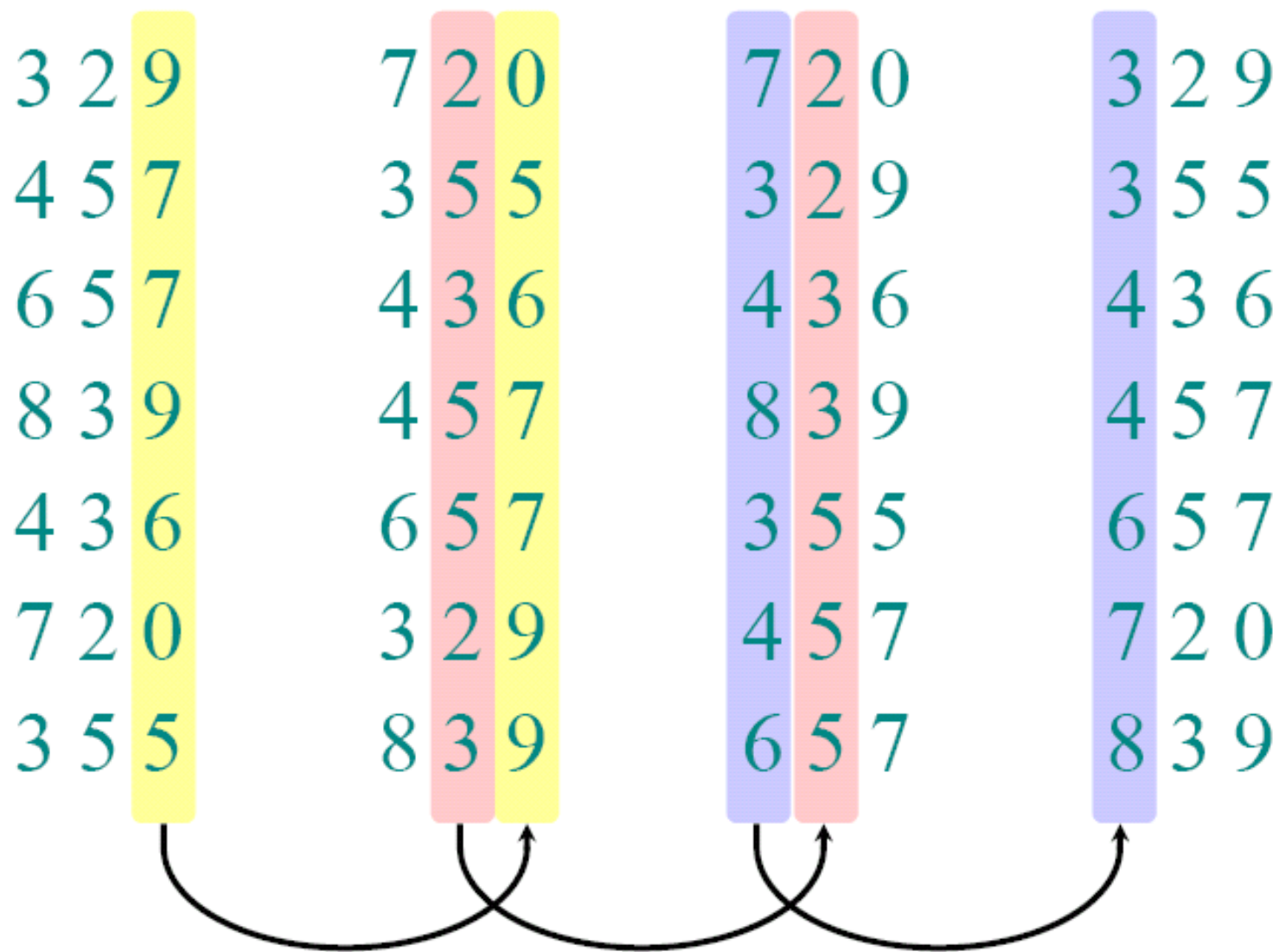
- Lyukkártya-rendező berendezéseknél használták.
- Lyukkártya: 80 oszlopa volt, minden oszlopban a 12 pozíció közül egy ki volt lyukasztva.
- A rendező-berendezés képes volt szétszítani a lapokat 12 dobozba, aszerint, hogy melyik pozíció volt kilyukasztva.
- Ezután a kártyákat összegyűjthették úgy, hogy az első pozícióban kilyukasztott lapok a második pozícióban kilyukasztott lapok felett legyenek, és így tovább.

# Számjegyes rendezés

- Feltételezzük, hogy mind az  $n$  szám legtöbb  $d$  számjegyű lehet.
- Sorban dolgozzuk fel a számjegyeket jobbról balra, a legkevésbé fontos számjeggyel kezdve.
- Először rendezzük a számokat az **utolsó számjegyük alapján** úgy, hogy ha csak ezt a számjegyet tekintjük, növekvő sorrendet lássunk.
- Ezután a számokat **újra rendezzük a második legkevésbé értékes számjegyük alapján**.
- Ezt addig folytatjuk, ameddig a számokat mind a  $d$  számjegy szerint nem rendeztük.
- Fontos, hogy egy adott számjegy szerinti rendezés stabil legyen, hogy ne rontsuk el az addigi munkánkat.
- Decimális számjegyek esetén minden számnak 10 lehetséges számjegye van.



## Példa



**Algoritmus Számjegyes\_Rendezés(a, n, d):**

*// Bemeneti adatok: az n elemű a sorozat, d a Legtöbb számjegy*

*// Kimeneti adatok: az n elemű rendezett a sorozat*

**Minden  $i = 1, d$  végezd el:**

*stabil leszámplálással rendezzük az a tömböt az i-edik*

*számjegy szerint*

**vége(minden)**

**Vége(algoritmus)**

# STABIL Leszámláló (láda)rendezés

**1. Ha a rendezendő adatok a kulcson kívül tartalmazzanak más adatokat is:**

- **Bemenet:**  $A[1..n]$ , ahol  $A_j \in \{1, \dots, k\}$ .
- **Kimenet:**  $B[1..n]$ ,  $A$  elemei rendezve.
- Átmeneti munkaterület:  $C[1..k]$ .

**Algoritmus** Leszámláló\_Rendezés( $n$ ,  $A$ ,  $B$ ,  $k$ ):

Minden  $i = 1, k$  végezd el: *//  $k$  az  $A$  tömb legnagyobb eleme*

$C_i \leftarrow 0$  *// még nem találtunk egyetlen  $i$  értékű elemet sem*

**vége(minden)**

Minden  $j = 1, n$  végezd el:

$C_{A_j} \leftarrow C_{A_j} + 1$  *//  $C_i$  az  $i$  értékű elemek száma*

**vége(minden)**

Minden  $i = 2, k$  végezd el:

$C_i \leftarrow C_i + C_{i-1}$  *//  $i$ -nél kisebb és vele egyenlő elemek száma*

**vége(minden)**

Minden  $j = n, 1, -1$  végezd el:

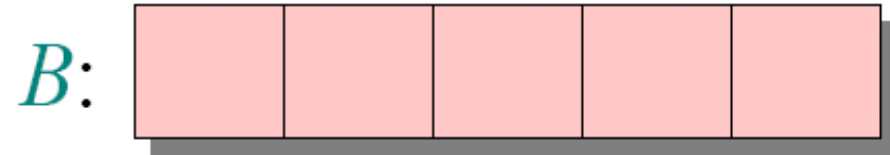
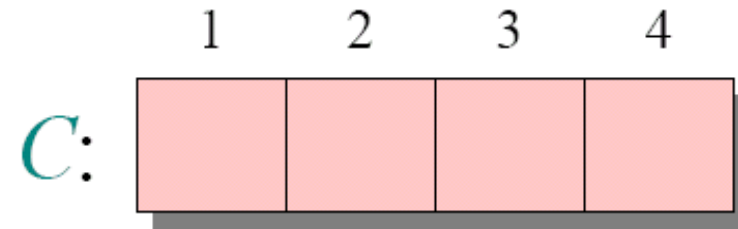
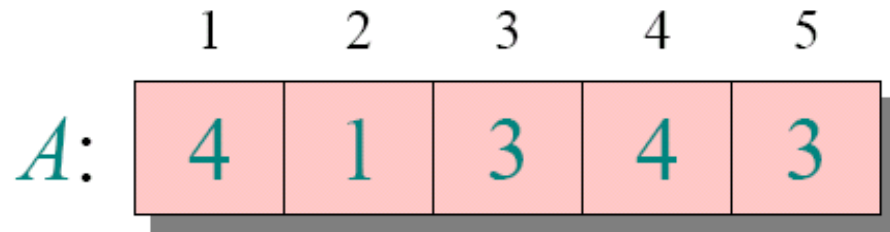
$B_{C_{A_j}} \leftarrow A_j$  *//  $B = A$  rendezve*

$C_{A_j} \leftarrow C_{A_j} - 1$

**vége(minden)**

**Vége(algoritmus)**

# Leszámláló rendezés (példa)



# Leszámláló rendezés (példa)

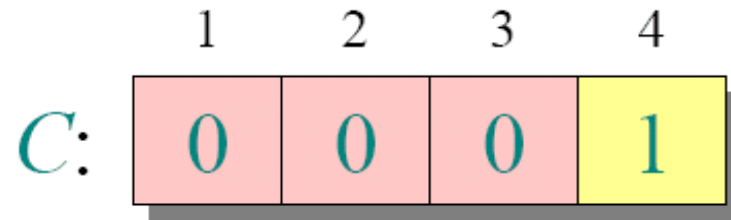
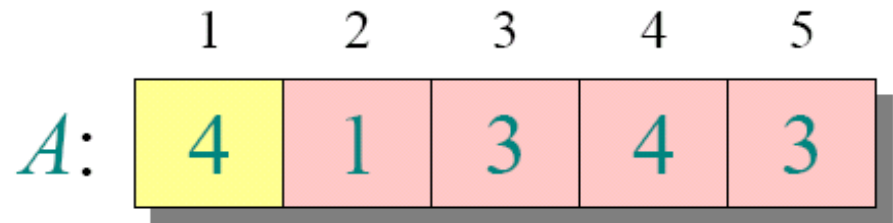
	1	2	3	4	5
<i>A</i> :	4	1	3	4	3

	1	2	3	4
<i>C</i> :	0	0	0	0

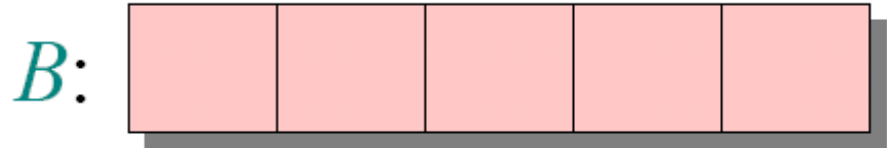
<i>B</i> :					
------------	--	--	--	--	--

**Minden  $i = 1$ ,  $k$  végezd el:**                    *// k az A tömb legnagyobb eleme*  
     $C_i \leftarrow 0$                     *// még nem találtunk egyetlen  $i$  értékű elemet sem*  
**vége(minden)**

# Leszámláló rendezés (példa)



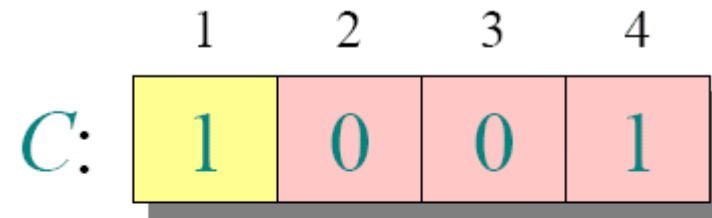
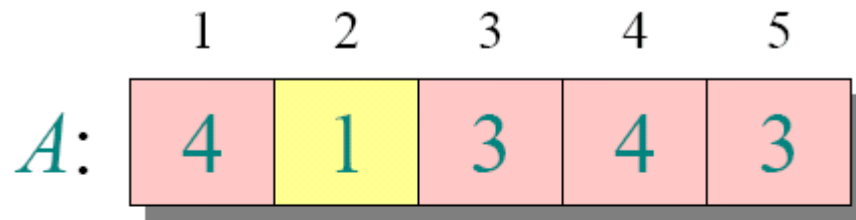
$j = 1$



Minden  $j = 1, n$  végezd el:

$C_{A_j} \leftarrow C_{A_j} + 1$  //  $C_i$  az  $i$  értékű elemek száma  
vége(minden)

# Leszámláló rendezés (példa)



$j = 2$

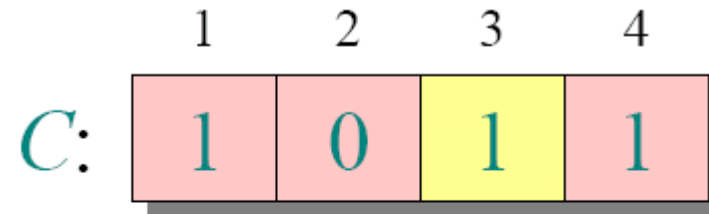
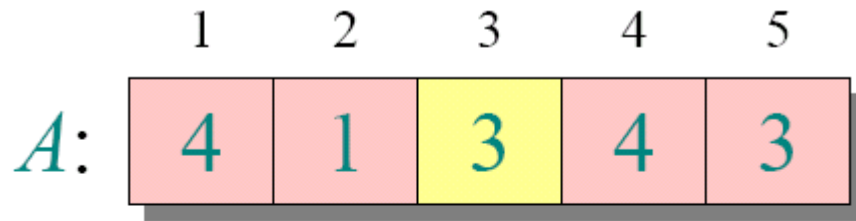


Minden  $j = 1, n$  végezd el:

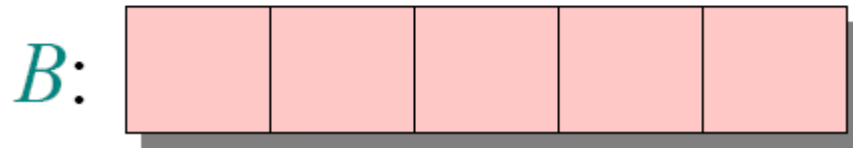
$C_{A_j} \leftarrow C_{A_j} + 1$  //  $C_i$  az  $i$  értékű elemek száma  
vége(minden)



# Leszámláló rendezés (példa)



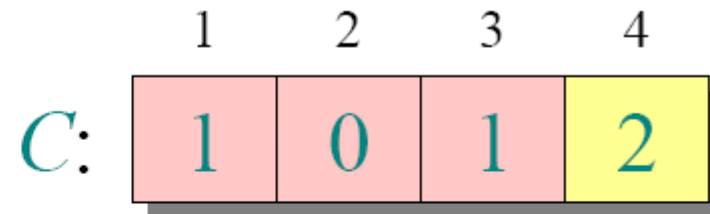
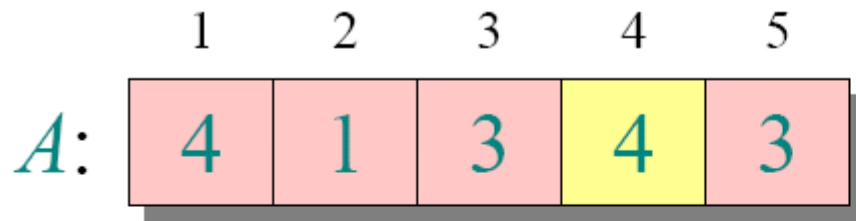
$j = 3$



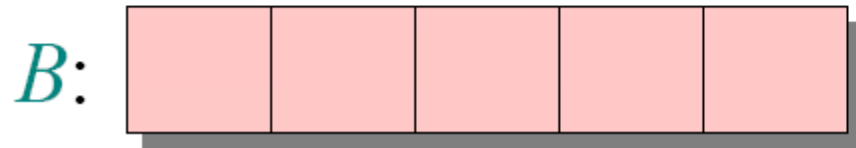
Minden  $j = 1, n$  végezd el:

$C_{A_j} \leftarrow C_{A_j} + 1$  //  $C_i$  az  $i$  értékű elemek száma  
vége(minden)

# Leszámláló rendezés (példa)



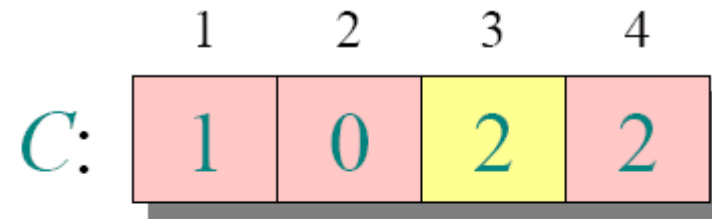
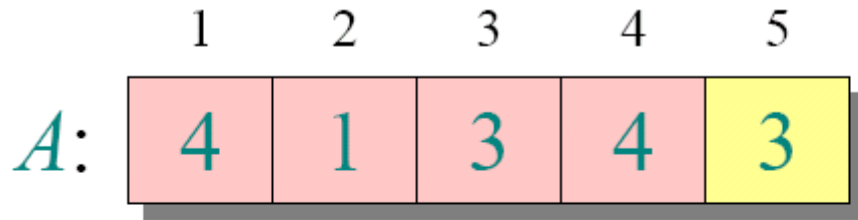
*j* = 4



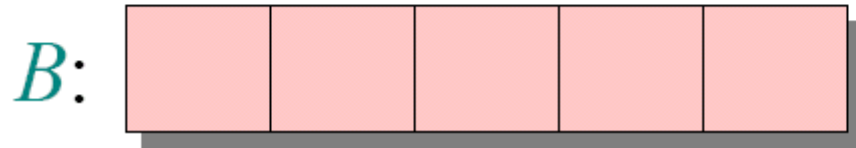
Minden  $j = 1, n$  végezd el:

$C_{A_j} \leftarrow C_{A_j} + 1$       //  $C_i$  az  $i$  értékű elemek száma  
vége(minden)

# Leszámláló rendezés (példa)



$j = 5$



Minden  $j = 1, n$  végezd el:

$C_{A_j} \leftarrow C_{A_j} + 1$  //  $C_i$  az  $i$  értékű elemek száma  
vége(minden)

# Leszámláló rendezés (példa)

	1	2	3	4	5
<i>A</i> :	4	1	3	4	3

	1	2	3	4
<i>C</i> :	1	0	2	2

*i* = 2

<i>B</i> :					
------------	--	--	--	--	--

<i>C'</i> :	1	1	2	2
-------------	---	---	---	---

Minden *i* = 2, *k* végezd el:

$C_i \leftarrow C_i + C_{i-1}$  // *i*-nél kisebb és vele egyenlő elemek száma  
vége(minden)

# Leszámláló rendezés (példa)

	1	2	3	4	5
$A$ :	4	1	3	4	3

	1	2	3	4
$C$ :	1	0	2	2

$B$ :					
-------	--	--	--	--	--

$i = 3$

$C'$ : 

1	1	3	2
---	---	---	---

Minden  $i = 2$ ,  $k$  végezd el:

$C_i \leftarrow C_i + C_{i-1}$  //  $i$ -nél kisebb és vele egyenlő elemek száma  
vége(minden)

# Leszámláló rendezés (példa)

	1	2	3	4	5
$A$ :	4	1	3	4	3

	1	2	3	4
$C$ :	1	0	2	2

$B$ :					
-------	--	--	--	--	--

$C'$ :

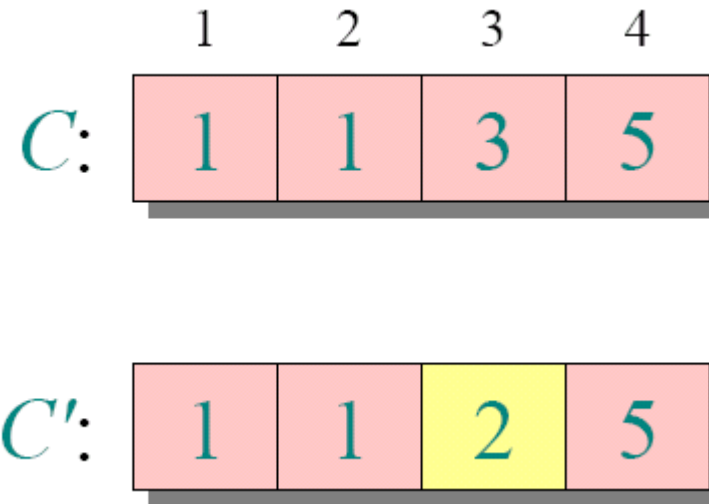
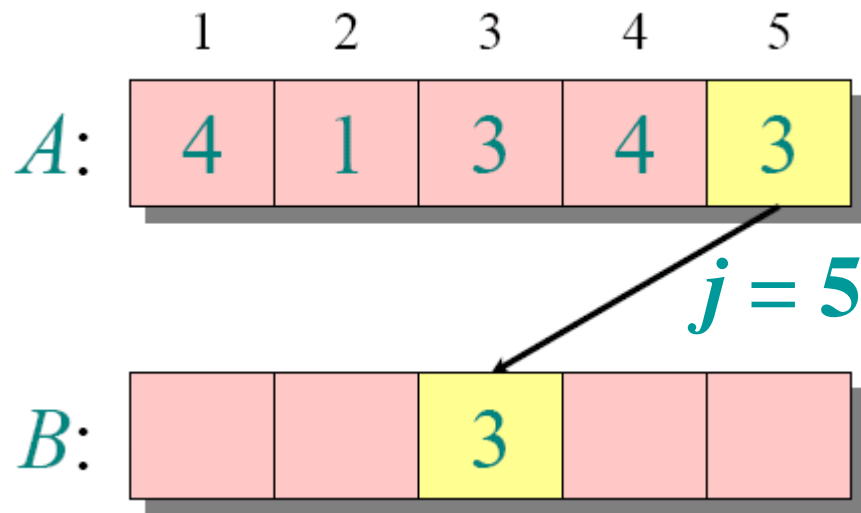
1	1	3	5
---	---	---	---

$i = 4$

Minden  $i = 2$ ,  $k$  végezd el:

$C_i \leftarrow C_i + C_{i-1}$  //  $i$ -nél kisebb és vele egyenlő elemek száma  
vége(minden)

# Leszámláló rendezés (példa)



Minden  $j = n, 1, -1$  végezd el:

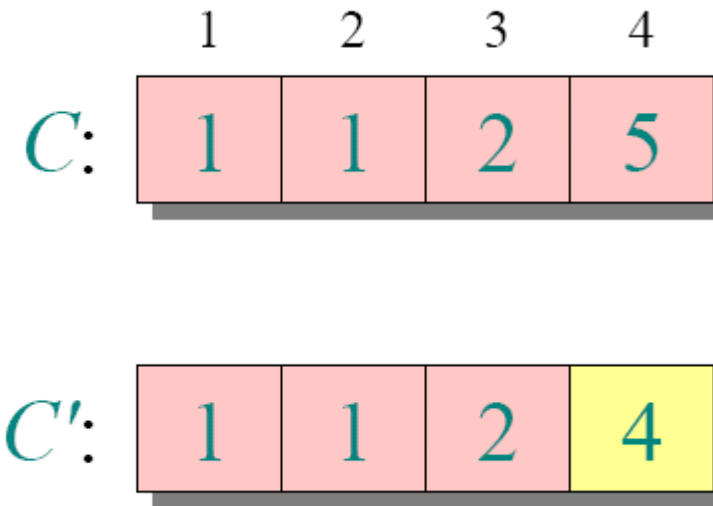
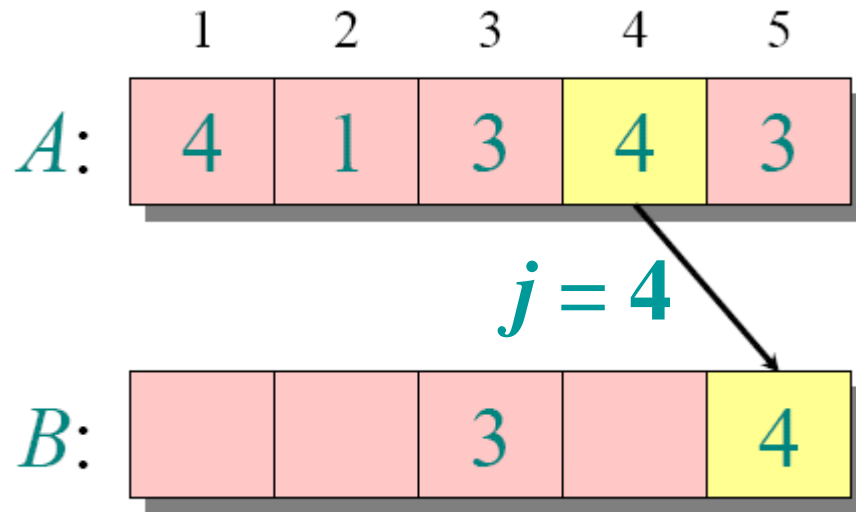
$B_{CAj} \leftarrow A_j$  //  $B = A$  rendezve

$CAj \leftarrow CAj - 1$

vége(minden)

$A_5 = 3 \Rightarrow C_3 = 3 \Rightarrow B_3 = A_5$   
és  $C'_3 = 3 - 1 = 2$

# Leszámláló rendezés (példa)



Minden  $j = n, 1, -1$  végezd el:

$B_{CAj} \leftarrow A_j$  //  $B = A$  rendezve

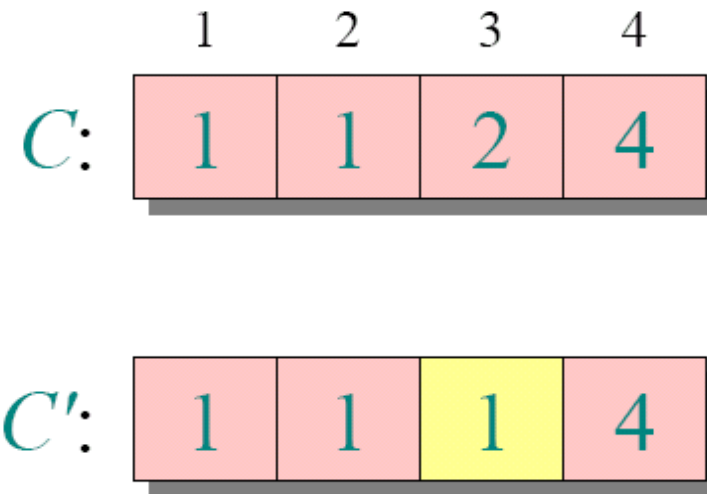
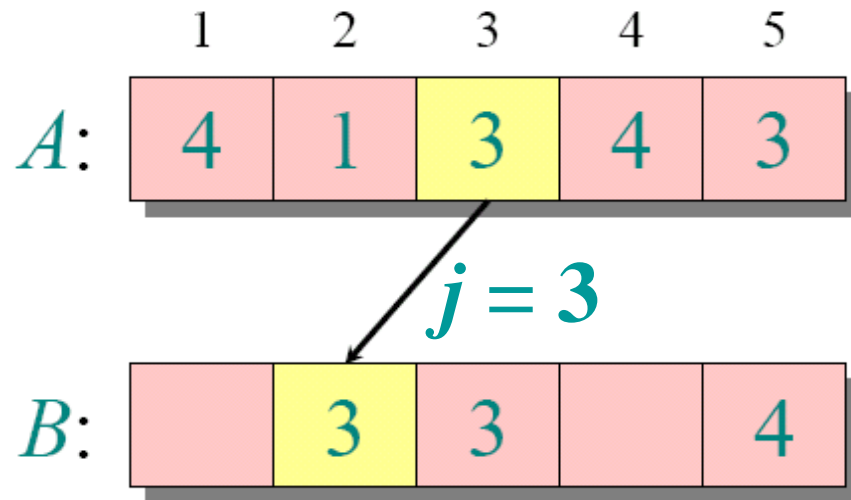
$C_{Aj} \leftarrow C_{Aj} - 1$

vége(minden)

$A_4 = 4 \Rightarrow C_4 = 5 \Rightarrow B_5 = A_4$   
és  $C'_4 = 5 - 1 = 4$



# Leszámláló rendezés (példa)



Minden  $j = n, 1, -1$  végezd el:

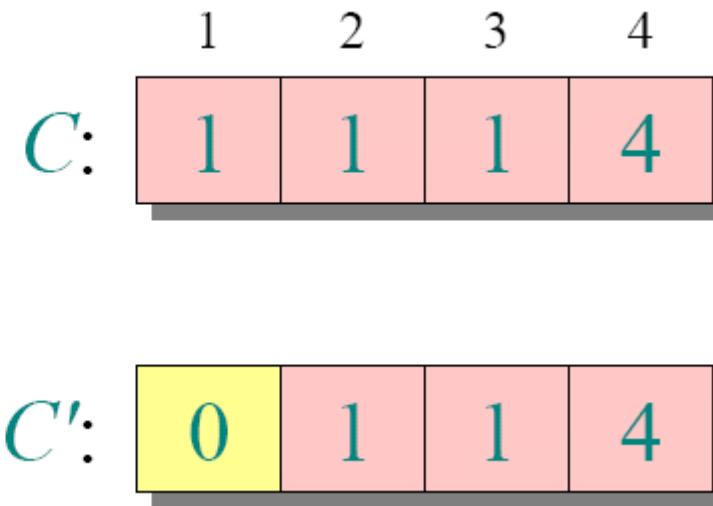
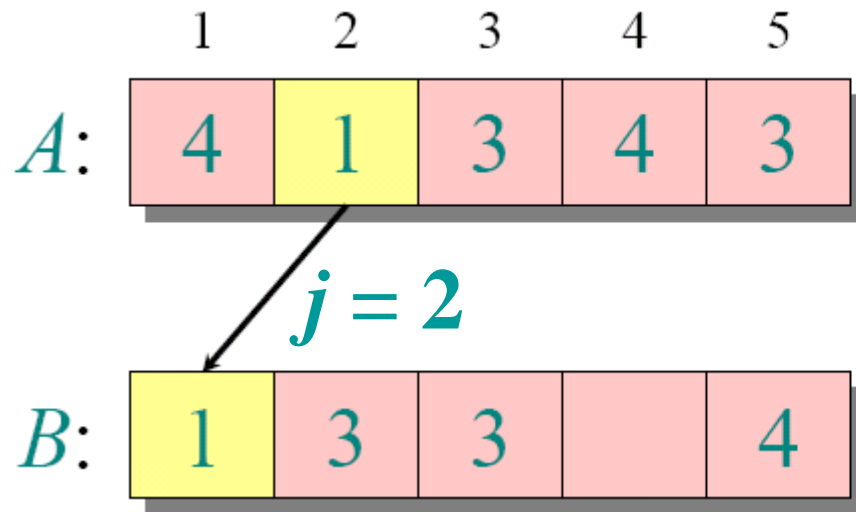
$B_{CAj} \leftarrow A_j$  //  $B = A$  rendezve

$C_{Aj} \leftarrow C_{Aj} - 1$

vége(minden)

$A_3 = 3 \Rightarrow C_3 = 2 \Rightarrow B_2 = A_3$   
és  $C'_3 = 2 - 1 = 1$

# Leszámláló rendezés (példa)



Minden  $j = n, 1, -1$  végezd el:

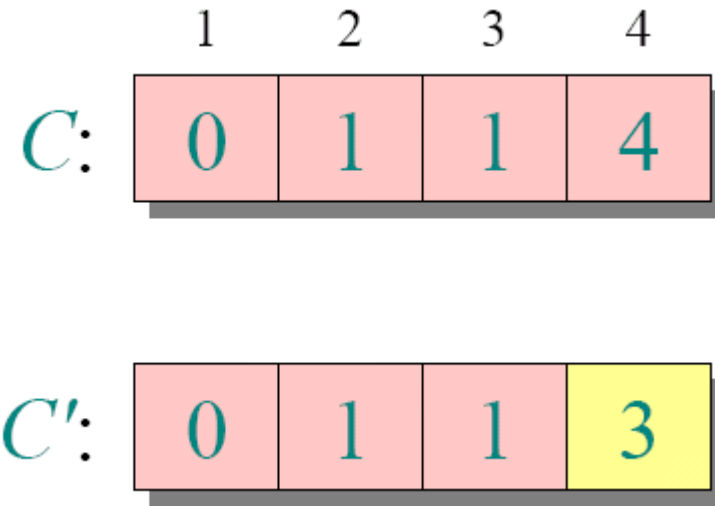
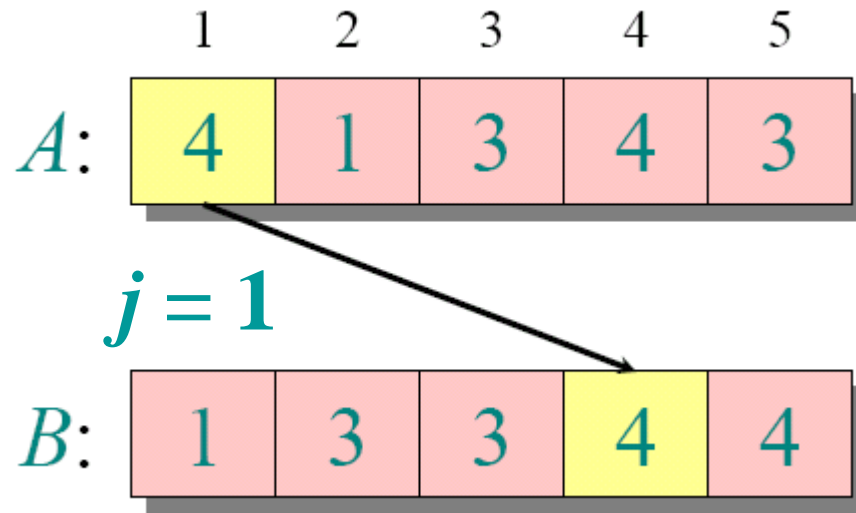
$B_{CAj} \leftarrow A_j$  //  $B = A$  rendezve

$C_{Aj} \leftarrow C_{Aj} - 1$

vége(minden)

$A_2 = 1 \Rightarrow C_1 = 1 \Rightarrow B_1 = A_2$   
és  $C'_1 = 1 - 1 = 0$

# Leszámláló rendezés (példa)



Minden  $j = n, 1, -1$  végezd el:

$B_{CAj} \leftarrow A_j$  //  $B = A$  rendezve

$C_{Aj} \leftarrow C_{Aj} - 1$

vége(minden)

$A_1 = 4 \Rightarrow C_4 = 4 \Rightarrow B_4 = A_1$   
és  $C'_4 = 4 - 1 = 3$

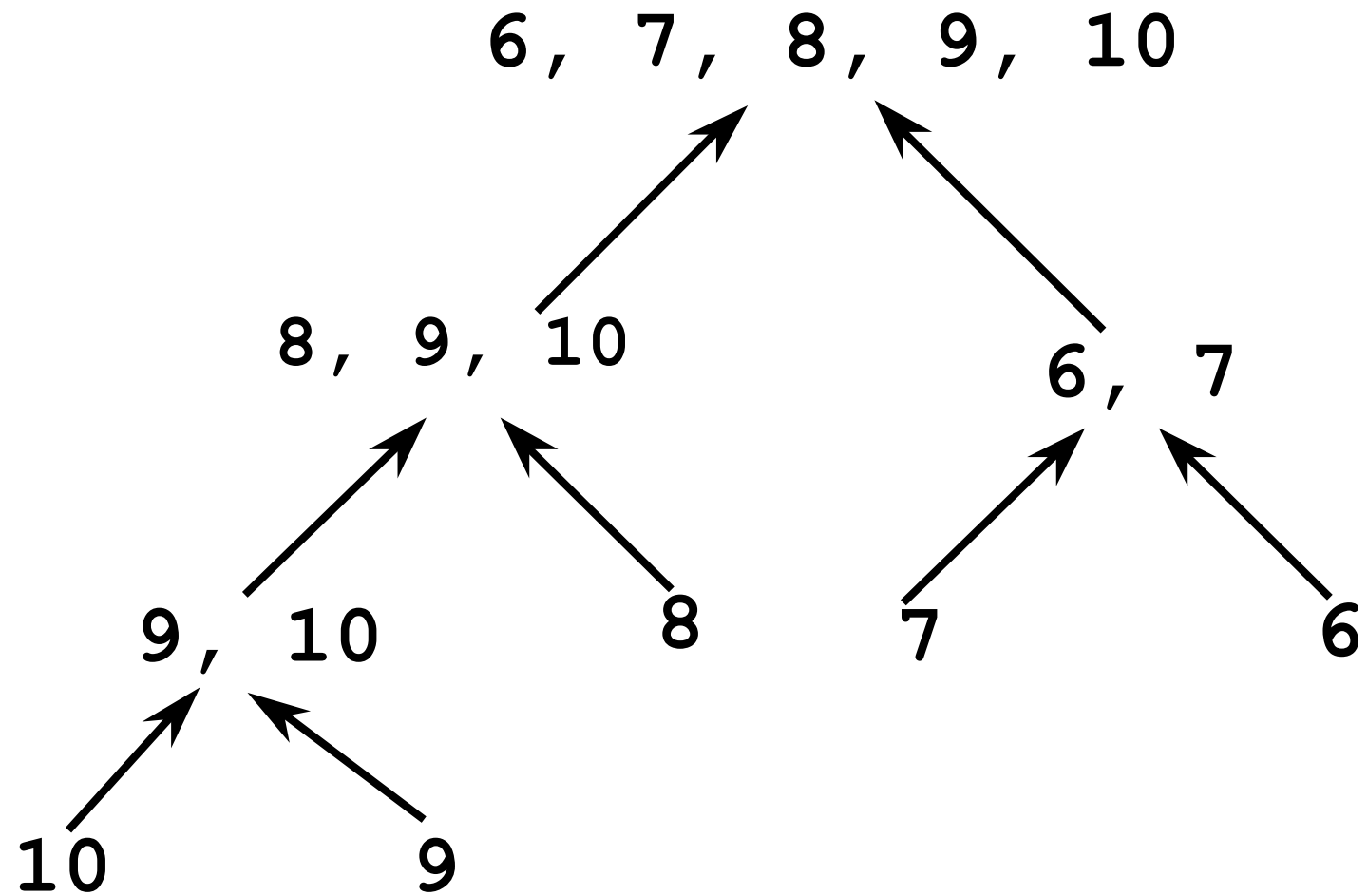
# Összefésülésen alapuló rendezés (merge-sort)

*Rendezzünk növekvő sorrendbe egy egész számokból álló sorozatot összefésüléssel!*

## **Megoldás**

**Összefésülés:** két rendezett sorozatból előállítunk egy harmadik növekvő sorrendű sorozatot.

**Rendezés céljából:** *az adott sorozatot két részre osztjuk*. Ezeket újból felosztjuk addig amíg a kapott rendezendő sorozat egyelemű; az egyelemű sorozatok rendezettek és megkezdődhet a tulajdonképpeni összefésülés.



**Algoritmus Összefésül(bal, közép, jobb):**

**Minden  $i = \text{bal}$ , közép végezd el:**

$a_i \leftarrow x_i$

**vége(minden)**

**Minden  $i = \text{közép}+1$ , jobb végezd el:**

$b_i \leftarrow x_i$

**vége(minden)**

$a_{\text{közép}+1} \leftarrow \text{végtelen}$

$b_{\text{jobb}+1} \leftarrow \text{végtelen}$

$i \leftarrow \text{bal}$

$j \leftarrow \text{közép}+1$

**Minden  $k = \text{bal}$ , jobb végezd el:**

**Ha  $a_i < b_j$  akkor**

$x_k \leftarrow a_i: i \leftarrow i + 1$

**különben**

$x_k \leftarrow b_j: j \leftarrow j + 1$

**vége(ha)**

**vége(minden)**

**Vége(algoritmus)**

**Algoritmus** Rendez(bal, jobb):

Ha  $bal < jobb$  akkor

közép  $\leftarrow [(bal+jobb)/2]$

Rendez(bal, közép)

Rendez(közép+1, jobb)

Összefésül(bal, közép, jobb)

vége(ha)

**Vége(algoritmus)**

A hívó programegységben a **Rendez(1, n)** algoritmust hívjuk.

# Gyorsrendezés (Quicksort)

*Fölhasználva a QuickSort algoritmust, rendezzünk növekvő sorrendbe **n** egész számot!*

## **Megoldás**

A gyorsrendezés az **oszd meg és uralkodj** módszeren alapszik, mivel az eredeti sorozatot úgy rendezi, hogy **két rendezendő részsorozatra meg a köztük levő elemre** bontja.

A részsorozatok rendezése egymástól függetlenül történik.

A részeredmények összerakása hiányzik.



# Megoldás

Előkészítünk két részsorozatot:

- az  $x_1, \dots, x_{m-1}$  részsorozat elemei  $<$  mint az  $x_{m+1}, \dots, x_n$  tömb elemei
- Köztük található az  $x_m >$  mint az  $x_1, \dots, x_{m-1}$  részsorozat bármely eleme és  $<$  mint az  $x_{m+1}, \dots, x_n$  részsorozat összes eleme.

**Strázsa (őrszem):** az az elem, amely meghatározza a helyet ahol az adott tömb két részre oszlik.

Ennek a helynek a meghatározása kulcskérdés az algoritmus végrehajtása során.

# A strázsa helye

Gyakran: az  $x_1$ -et választjuk strázsának.

Elindulunk a tömb két szélső elemétől és *felcseréljük egymás közt azokat az elemeket,*

- amelyek *nagyobbak*, vagy egyenlők a strázsával (és a tömb első részében található) azokkal,
- amelyek *kisebbek* mint a strázsa (és a tömb második részében található).

Ahol ez a bejárás véget ér, ott fogjuk két részre osztani a tömböt.

## Példa

8, 7, 6, 10, 4, 11, 2, 5, 9 a rendezendő sorozat

**8**, 7, 6, 10, 4, 11, 2, 5, **9**      (**8 < 9**)

**8**, 7, 6, 10, 4, 11, 2, **5**, 9      (**8 > 5**)  $\Rightarrow$  **csere**

5, **7**, 6, 10, 4, 11, 2, **8**, 9      (**7 < 8**)

5, 7, **6**, 10, 4, 11, 2, **8**, 9      (**6 < 8**)

5, 7, 6, **10**, 4, 11, 2, **8**, 9      (**10 > 8**)  $\Rightarrow$  **csere**

5, 7, 6, **8**, 4, 11, **2**, 10, 9      (**8 > 2**)  $\Rightarrow$  **csere**

5, 7, 6, 2, **4**, 11, **8**, 10, 9      (**4 < 8**)

5, 7, 6, 2, 4, **11**, **8**, 10, 9      (**11 > 8**)  $\Rightarrow$  **csere**

5, 7, 6, 2, 4, **8**, 11, 10, 9      **8** a *végleges* helyére került

# Példa

a **8**-tól *balra* eső részsorozatban csak **8-nál kisebb** számok találhatók,  
a **8**-tól *jobbra* eső részsorozatban csak **8-nál nagyobb** számok vannak.

(5, 7, 6, 2, 4), **8**, (11, 10, 9)

Ezt a két részsorozatot feldolgozzuk az előbbi módon. A bal részsorozat a következő átalakításokon megy át:

(**5**, 7, 6, 2, **4**) → (4, **7**, 6, 2, **5**) → (4, **5**, 6, **2**, 7) → (4, 2, **6**, **5**, 7) → (4, 2), **5**, (6, 7)  
(**4**, **2**) → (2, 4),

(6, 7) marad változatlanul.

Az eredeti sorozat jobb részsorozata a következő lépések során rendeződik:

(**11**, 10, **9**) → (9, **10**, **11**) → (9, 10), **11**

Így az eredeti sorozat most:

2, 4, 5, 6, 7, 8, 9, 10, 11.

**Algoritmus QuickSort(bal, jobb):**

**Ha** bal < jobb **akkor**

Feloszt(bal, jobb, m) *// meghatározzuk azt az m*

*// helyet, ahol a sorozatot két részsorozatra bontjuk,*

*// miközben egy elem a végleges m helyére kerül*

QuickSort(bal, m - 1)

QuickSort(m + 1, jobb)

**vége(ha)**

**Vége(algoritmus)**

**Algoritmus Feloszt(bal, jobb, m):**

strázsa  $\leftarrow x_{\text{bal}}$

$i \leftarrow \text{bal} - 1$

$j \leftarrow \text{jobb} + 1$

**Ismételd**

**Ismételd**

$j \leftarrow j - 1$

*// megkeressük azt a  $j$ -t amelyre  $x_j > \text{strázsa}$*

**amедdig**  $x_j \leq \text{strázsa}$

**Ismételd**

$i \leftarrow i + 1$

*// megkeressük azt az  $i$ -t amelyre  $x_i < \text{strázsa}$*

**amедdig**  $x_i \geq \text{strázsa}$

**Ha**  $i < j$  **akkor**

*// felcseréljük ezt a két elemet*

    felcserél( $x_i$ ,  $x_j$ )

**vége(ha)**

**amедdig**  $i \geq j$

*// ezeket addig végezzük, amíg  $i$  kisebb mint  $j$*

$m \leftarrow j$

**Vége(algoritmus)**

# Feloszt másképp

- Minden lépésben *nyilvántartjuk annak a két elemnek az indexét, amelyeket össze kell hasonlítanunk.*
- Minden lépésben megváltozik a két index közül valamelyik, aszerint, hogy *jobbról haladunk balra*, vagy *balról jobbra*.
- *ii* és *jj* segítségével minden lépésben nyilvántartjuk, hogy melyik index fog nőni és melyik fog csökkenni.
- Kezdetben az első elemet hasonlítjuk azokkal, amelyek a sorozat végén találhatók jobbról balra haladva.
- A bal index (*i*) nem változik, és a jobb index (*j*) csökken  $\Rightarrow ii \leftarrow 0$ , és  $jj \leftarrow -1$ .
- Az első felcserélés után *i* nő és *j* rögzített  $\Rightarrow ii \leftarrow 1$  és  $jj \leftarrow 0$ .
- Az összehasonlításokat addig folytatjuk míg *i* egyenlő lesz *j*-vel.
- Amikor  $i = j$ , az első elem az *i*-edik helyre került, amely egyben a végleges hely.

**Algoritmus Feloszt\_2(bal,jobb,i):**

$i \leftarrow \text{bal}; j \leftarrow \text{jobb}$

$ii \leftarrow 0; jj \leftarrow -1$

**Amíg  $i < j$  végezd el:**

**Ha  $x_i > x_j$  akkor**

*// ha az elemek nincsenek a megfelelő sorrendben*

**felcserél( $x_i, x_j$ )**

$id \leftarrow ii$

$ii \leftarrow -jj$

$jj \leftarrow -id$

**vége(ha)**

$i \leftarrow i + ii$

$j \leftarrow j + jj$

**vége(amíg)**

**Vége(algoritmus)**



# Feloszt másképp (3. módszer)

Algoritmus Feloszt\_3(bal, jobb, q):

*// jobbról indulva a felosztással*

$x \leftarrow A_{\text{bal}}$

*// őrszem =  $A_{\text{bal}}$*

$i \leftarrow \text{bal}$

Minden  $j = \text{bal} + 1$ , jobb végezd el:

Ha  $A_j \leq x$  akkor

$i \leftarrow i + 1$

felcserél( $A_i$ ,  $A_j$ )

vége(ha)

vége(minden)

felcserél( $A_{\text{bal}}$ ,  $A_i$ )

$q \leftarrow i$

Vége(algoritmus)

# A gyorsrendezés egy véletlenített változata

- Egy másik ötlet: őrszemnek az  $A[\text{bal..jobb}]$  résztömb egy ***véletlenszerűen választott*** elemét vesszük.
- Ez a módosítás biztosítja, hogy az  $x = A_{\text{jobb}}$  őrszem ugyanolyan valószínűséggel lehet az  $A[\text{bal..jobb}]$  résztömb bármelyik eleme.
- Mivel az őrszemet véletlenszerűen választjuk ki, a felosztás várhatóan jól kiegyensúlyozott lesz.
- A **Feloszt** és **Gyorsrendezés** eljárások kevésbé módosulnak: ***beépítjük két elem cseréjét az új felosztási eljárásba a felosztás előtt.***

**Algoritmus Véletlen\_Feloszt(bal, jobb, q):**

$i \leftarrow \text{Véletlen}(\text{bal}, \text{jobb})$                       // véletlenszám generálás

    felcserél( $A_{\text{jobb}}$ ,  $A_i$ )

    Véletlen\_Feloszt(bal, jobb, q)

**Vége(algoritmus)**

**Algoritmus Véletlen\_Gyorsrendezés(bal, jobb, q):**

    Ha  $\text{bal} < \text{jobb}$  akkor

        Véletlen\_Feloszt(bal, jobb, q)

        Véletlen\_Gyorsrendezés(bal, q-1, q)

        Véletlen\_Gyorsrendezés(q +1, jobb, q)

    vége(ha)

**Vége(algoritmus)**