Rendező algoritmusok

Ionescu Klára

clara@cs.ubbcluj.ro

Rendezések

Adott az ***N*** elemű ***X*** sorozat, amelynek keressük azt a permutációját, amelyben $X_1 \leq X_2 \leq \ldots \leq X_n$ (vagy $\leq$ helyett $\geq$).

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

1.a. Egyszerű felcseréléses rendezés

1.b. Minimumkiválasztásos rendezés

1.c. Buborékos rendezés (*bubblesort*)

1.d. Beszúró rendezés (*insertsort*)

1.e. Számlálva szétosztó rendezés (válogatás)

2. Lineáris rendezések $O(n)$:

2.a. Ládarendezés (*binsort*)

2.b. Számjegyes rendezés (*radixsort*)

Bevezetés

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

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

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

33 rendező algoritmust ír le...

Buborékrendezés

- A rendezés során ellenőriznünk kell az elemek sorrendjét.
- A sorozat akkor rendezett, ha $a_1 \leq a_2 \leq \ldots \leq a_{n-1} \leq a_n$.
- ***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.

Példák

a) Legyen a következő sorozat:

2, 3, 5, 9, 10 Az ellenőrzés végeredményeként kiderül, hogy

≤ ≤ ≤ ≤ a sorozat rendezett.

b) Ha a következő sorozatot kell rendezni,

2, 3, 5, 10, 9 Csak az utolsó két szám nincs megfelelő

≤ ≤ ≤ ≥ helyen.

Ha felcseréljük ezt a két számot:

2, 3, 5, 9, 10 Egyetlen csere után rendezett sorozathoz

≤ ≤ ≤ ≤ jutottunk.

Példa

c)

5, 2, 3, 10, 9 Miután felcseréljük az 5-öst a 2-sel, a sorozat a

≥ következőképpen alakul:

2, 5, 3, 10, 9 Felcseréljük az 5-öst a 3-sal:

≤ ≥

2, 3, 5, 10, 9 Felcseréljük a 9-est a 10-sel:

≤ ≤ ≤ ≥

2, 3, 5, 9, 10

≤ ≤ ≤ ≤

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

Példa

10, 9, 2, 3, 5

≥

9, 10, 2, 3, 5

≥

9, 2, 10, 3, 5

≥

9, 2, 3, 10, 5

≥

9, 2, 3, 5, 10

≥

Egyszeri bejárás után a sorozat nem rendezett.

2, 9, 3, 5, 10

≥

2, 3, 9, 5, 10

≥

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 Növekvő_sorrendbe_rendezés(n, a):

{ *Bemeneti adatok: az* n *elemű* a *sorozat* }

{ *Kimeneti adatok: az* n *elemű rendezett* a *sorozat* }

Ismételd

rendben ← *igaz*

Minden i=1,n-1 **végezd el:**

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

id ← a_i

a_i ← a_{i+1}

a_{i+1} ← id

rendben ← *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***!

$\Rightarrow$ Ha az első lépésnél az ***i*** ciklusváltozó ***n*-1**-ig nő, akkor a következő lépésekben ez a felső határ legalább ***1-gyel csökkenthető***.

Algoritmus Javított_Buborékrendezés_1(n, a):

{ *Bemeneti adatok: az* n *elemű* a *sorozat* }

{ *Kimeneti adatok: az* n *elemű rendezett* a *sorozat* }

```
    nn ← n - 1
    Ismételd
       rendben ← igaz
       Minden i = 1, nn végezd el:
           Ha a_i > a_{i+1} akkor
              rendben ← hamis
              id ← a_i; a_i ← a_{i+1}; a_{i+1} ← id
       vége(ha)
    vége(minden)
    nn ← 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 Javított_Buborékrendezés_2(n, a):

{ *Bemeneti adatok: az* n *elemű* a *sorozat* }

{ *Kimeneti adatok: az* n *elemű rendezett* a *sorozat* }

k ← n

Ismételd

nn ← k – 1; rendben ← *igaz*

Minden i=1,nn **végezd el:**

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

rendben ← *hamis*

id ← a_i; a_i ← a_{i+1}; a_{i+1} ← id

k ← 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. A 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: $\Omega(n)$
- ***Ha a sorozatot bejárjuk fordított irányban, és – az előbbi algoritmust a csökkenő rendezés érdekében alkalmazzuk, tovább javítható a futási idő!***

```
Algoritmus Javított_Buborékrendezés_3(n, a):
    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
                id ← a_i : a_i ← a_i+1 : a_i+1 ← id
                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* jobb *értékét* }

rendben ← igaz { *a bal szél* }

Minden i = régijobb, régibal **végezd el**:

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

rendben ← hamis

id ← a_i ; a_i ← a_{i+1} ; a_{i+1} ← id

Ha i < bal **akkor** bal ← i

vége(ha)

vége(ha)

vége(minden)

régibal ← bal { *aktualizáljuk a* bal *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):

Minden i = 1, n-1 **végezd el**:

Minden j = i+1,n **végezd el**:

Ha $a_i > a_j$ **akkor**

segéd ← a_i

a_i ← a_j

a_j ← segéd

vége(ha)

vége(minden)

vége(minden)

Vége(algoritmus)

Számlálva szétosztó rendezés

- Minden elemet ***összehasonlítunk*** az összes többi elemmel
- ***Megszámoljuk*** (***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 Válogatásos_Rendezés(n, a):

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 <* *elemeket* }

vége(ha)

vége(minden)

$rendezett\_sor_{k+1} \leftarrow a_i$

{ a_i *a rendezett sorozat megfelelő helyére kerül* }

vége(minden

$a \leftarrow rendezett\_sor$

{ *a rendezett sorozatot rámásoljuk az eredetire* }

Vége(algoritmus)

Megjegyzés

- Ez az algoritmus csak akkor alkalmazható, ha a sorozat csak ***különböző*** elemeket tartalmaz!
- Írjuk 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 Minimumkiválasztás(n, a):

Minden i = 1, n-1 **végezd el:**

ind_min ← i

Minden j = i+1, n **végezd el:**

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

ind_min ← j

vége(ha)

vége(minden)

segéd ← a_i

$a_i \leftarrow a_{ind\_min}$

a_{ind_min} ← segéd

vége(minden)

Vége(algoritmus)

Beszúró rendezés

- A beszúró rendezés hatékony algoritmus kis számú elem rendezésére.
- A beszúró 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 Beszúró_Rendezés(n, a):

Minden j = 2, n **végezd el**:

segéd ← a_j

i ← j - 1

Amíg (i > 0) **és** (a_i > segéd) **végezd el**:

a_{i+1} ← a_i

i ← i – 1

vége(amíg)

a_{i+1} ← segéd{ *beszúrjuk* 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.

$\Rightarrow$ 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 segédtömbre:
- 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 segédtömb elemeinek értékei alapján.

Megjegyzések

- ***k*** nem lehet túlságosan nagy, mivel ***a segédtömb 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 segédtömb indexelése ***x***-től ***y***-ig.

```
Algoritmus Leszámlálás(n, a):
    Minden i = 1, k végezd el:
        segéd_i ← 0
    vége(minden)
    Minden j = 1, n végezd el:
        segéd_aj ← segéd_aj + 1
    vége(minden)
    q ← 0
    Minden i = 1, k végezd el:
        Minden j = 1, segéd_i végezd el:
            q ← q + 1                    { a segéd tömbnek k eleme van, }
            a_q ← i                      { segéd_i elemek összege n }
        vége(minden)                     { tehát a feldolgozások száma n }
    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 szétosztani 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.

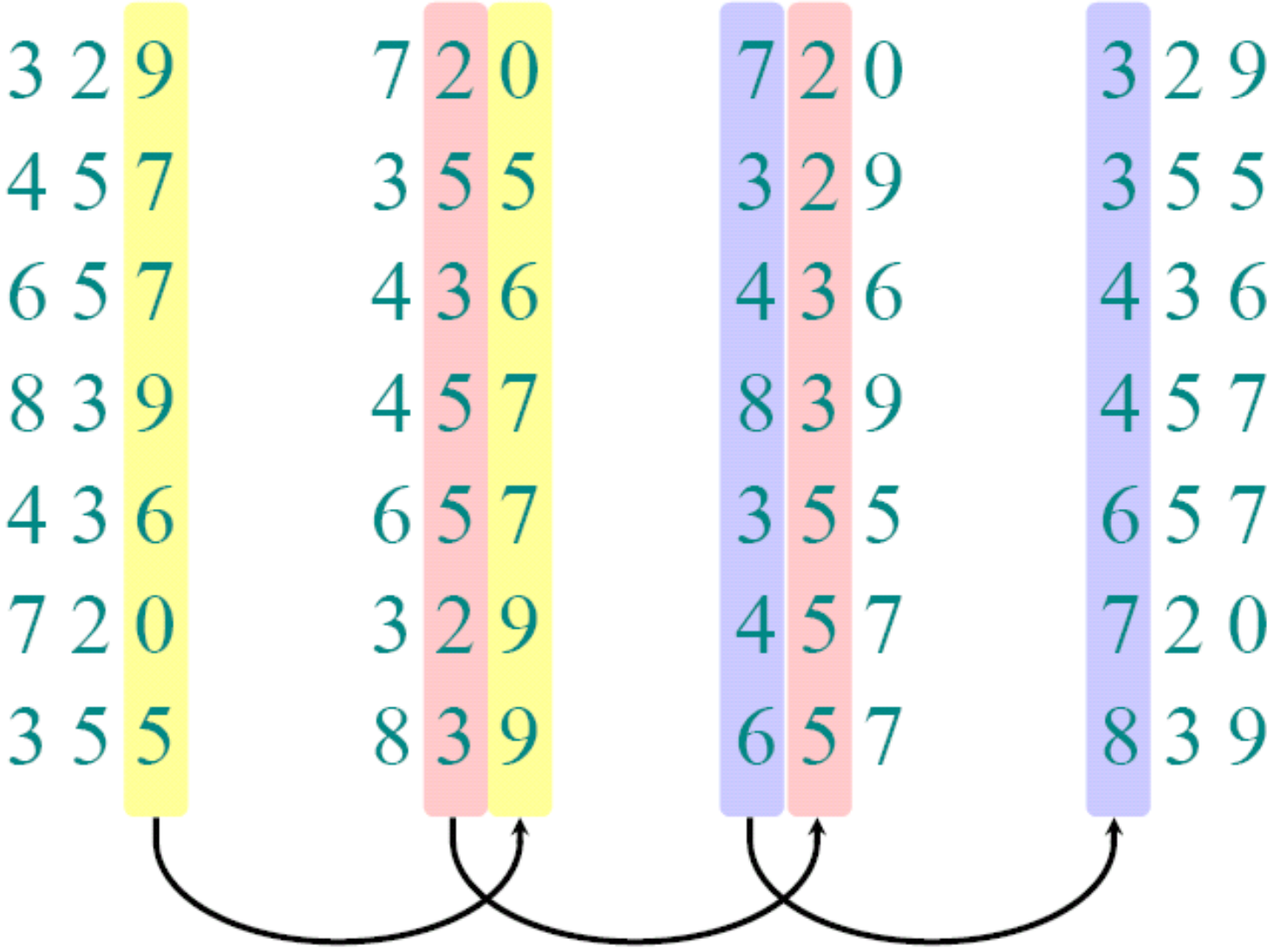
Számjegyes rendezés

- Decimális számjegyek esetén minden oszlopban 10 pozíció szükséges.
- Egy ***d*** számjegyű szám esetén ***d*** oszlopra van szükség.
- A rendező csak egy oszlopot vizsgál egyszerre: ***n*** darab ***d*** számjegyű szám rendezéséhez rendezőalgoritmusra van szükség.
- A számjegyes rendezés először a ***legkevésbé fontos számjegy alapján rendez***.

Számjegyes rendezés

- A számokat az utolsó számjegyük alapján rendezzük oly módon, hogy, ha csak ezt a számjegyet tekintjük, növekvő sorrendet lássunk.
- Ezután a számokat ***újra rendezzük a második legkevésbé értékes számjegyük alapján***.
- Ezt mindaddig végezzük, ameddig a számokat mind a ***d*** számjegy szerint nem rendeztük.
- Ezen a ponton a ***d*** számjegyű számok teljesen rendezettek.
- Így csak ***d***-szer kell megvizsgálni a sorozatot.

Példa


3 2 9
4 5 7
6 5 7
8 3 9
4 3 6
7 2 0
3 5 5
7 2 0
3 5 5
4 3 6
4 5 7
6 5 7
3 2 9
8 3 9
7 2 0
3 2 9
4 3 6
8 3 9
3 5 5
4 5 7
6 5 7
3 2 9
3 5 5
4 3 6
4 5 7
6 5 7
7 2 0
8 3 9

Algoritmus Számjegyes_Rendezés(a, d):

Minden i = 1, d **végezd el:**

stabil leszámlálással 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 tartalmaznak más adatokat is:*

- ***Bemenet***: $A[1..n]$, ahol $A_j \in \{1, \ldots, k\}$.
- ***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:**

$C_i \leftarrow 0$ { *még nem számoltunk egyetlen* i-*t sem* }

vége(minden)

Minden j = 1, n **végezd el:**

$C_{Aj} \leftarrow C_{Aj} + 1$ { C_i = 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 **végezd el:**

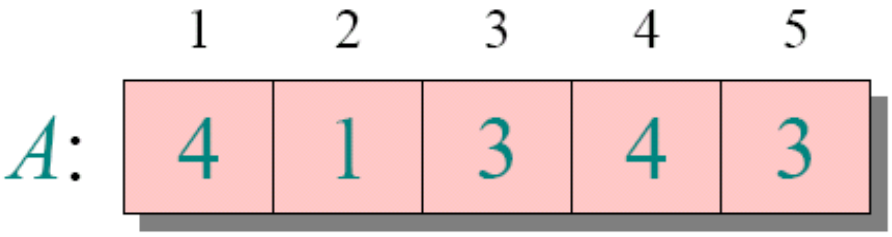
$B_{CAj} \leftarrow A_j$ { B = A *rendezve* }

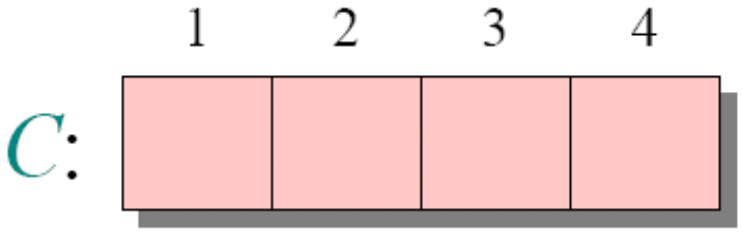
$C_{Aj} \leftarrow C_{Aj} - 1$

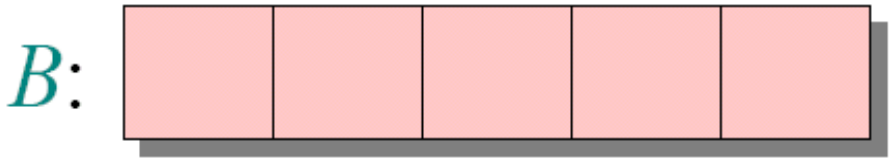
vége(minden)

Vége(algoritmus)

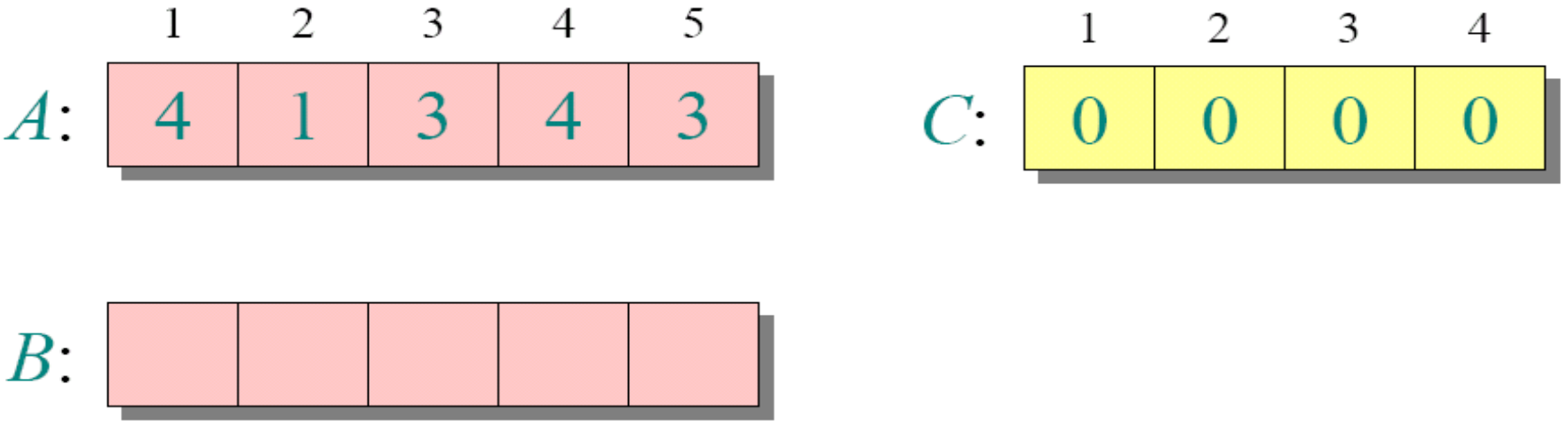
Leszámláló rendezés (példa)







Leszámláló rendezés (példa)



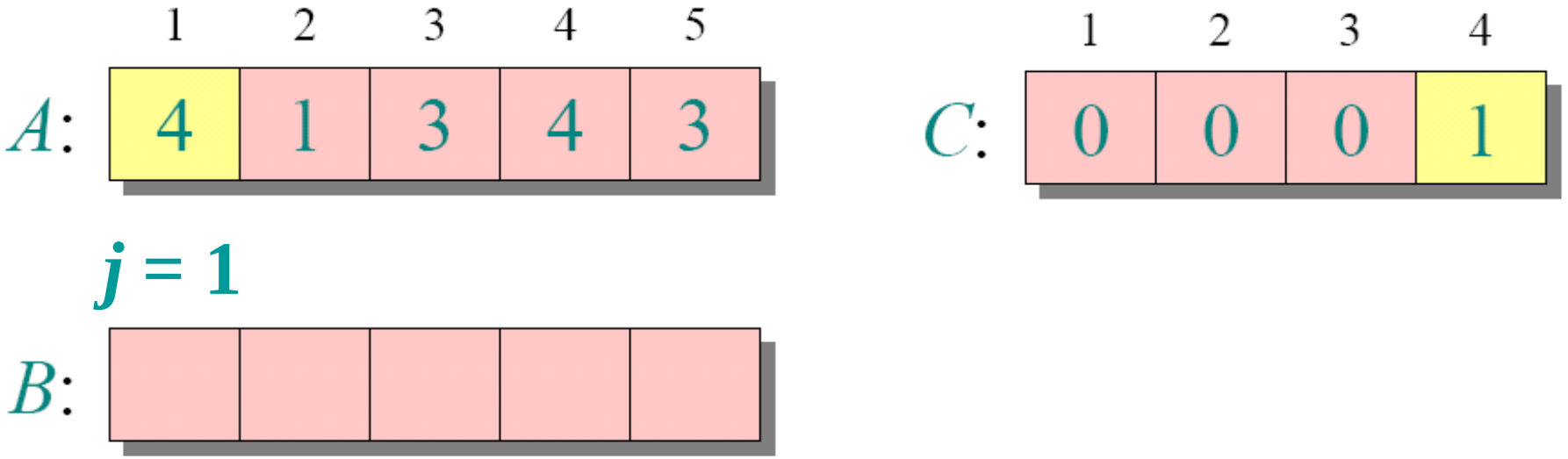
Minden i=1,k **végezd el:**

$C_i \leftarrow 0$ **{ még nem számoltunk egyetlen }**

{ i-vel egyenlő elemet sem }

vége(minden)

Leszámláló rendezés (példa)



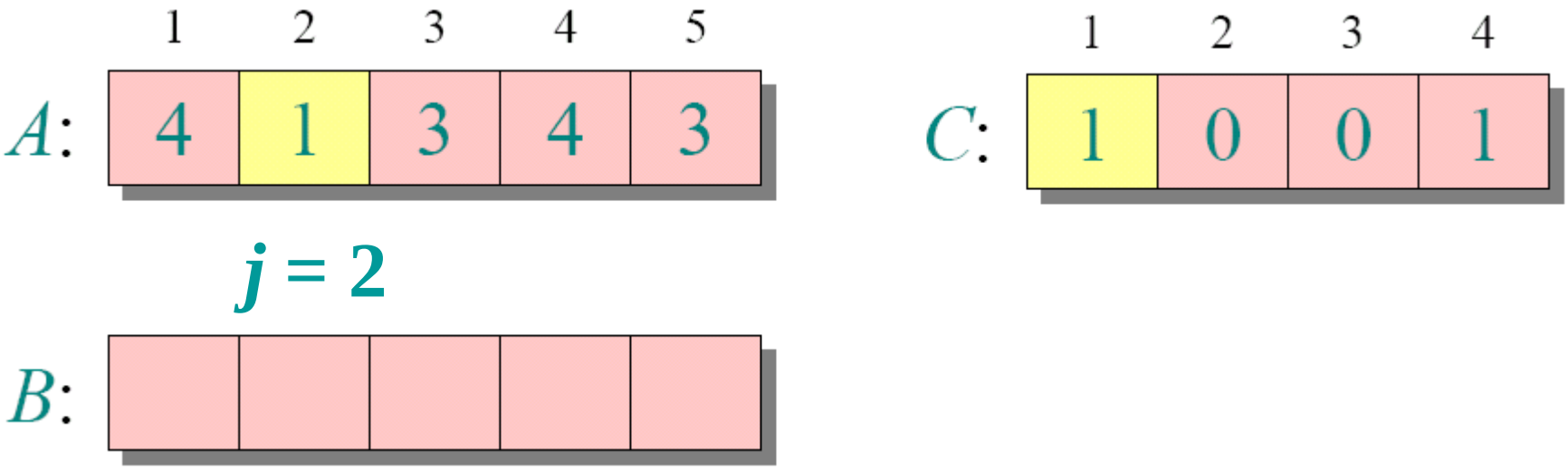
Minden j=1,n **végezd el:**

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

{ C_i = az i értékű elemek száma }

vége(minden)

Leszámláló rendezés (példa)



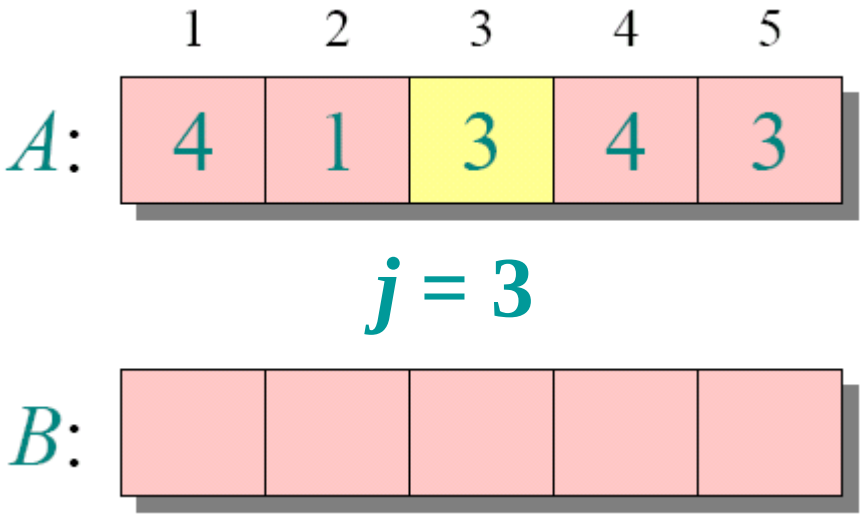
Minden j=1,n **végezd el:**

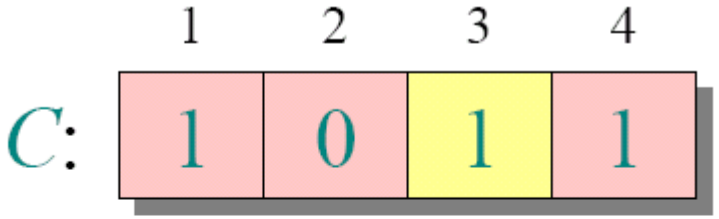
$C_{A_j} \leftarrow C_{A_j} + 1$

{ C_i = az i értékű elemek száma }

vége(minden)

Leszámláló rendezés (példa)





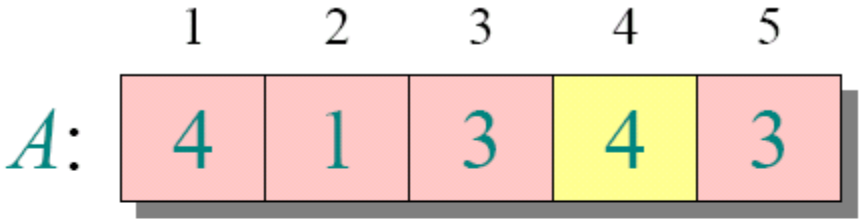
Minden j=1,n **végezd el:**

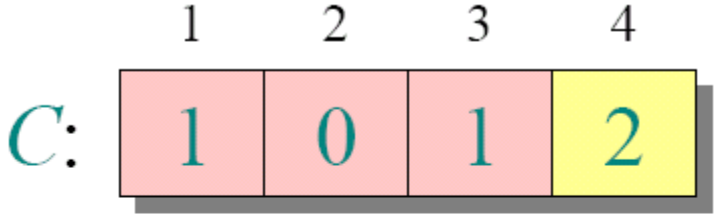
$C_{Aj} \leftarrow C_{Aj} + 1$

{ C_i = az i értékű elemek száma }

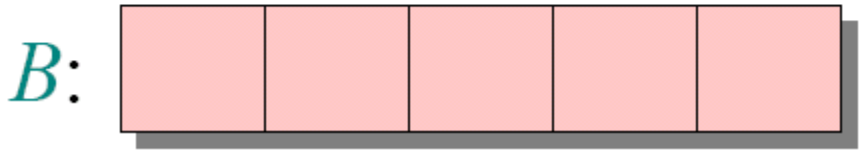
vége(minden)

Leszámláló rendezés (példa)





$j = 4$



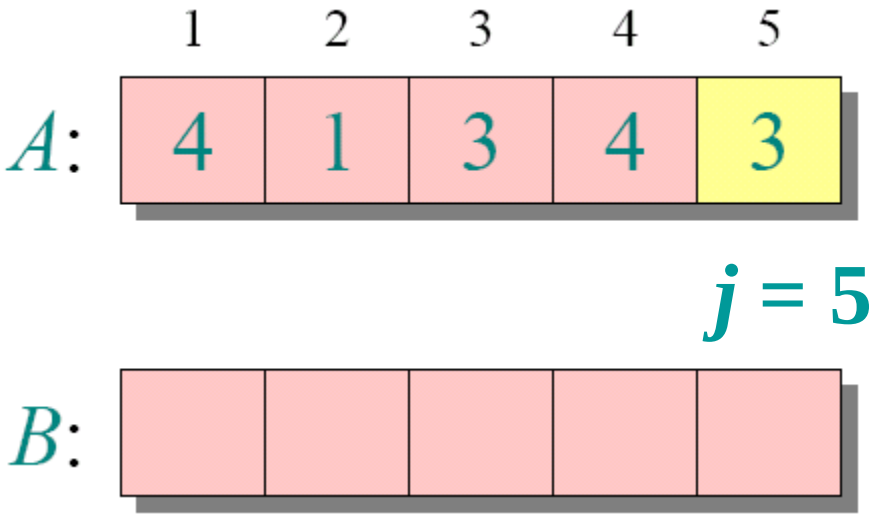
Minden j=1,n **végezd el:**

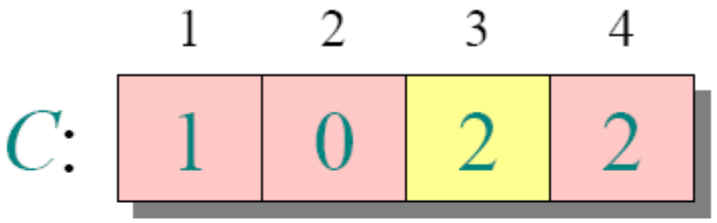
$C_{Aj} \leftarrow C_{Aj} + 1$

{ C_i = az i értékű elemek száma }

vége(minden)

Leszámláló rendezés (példa)





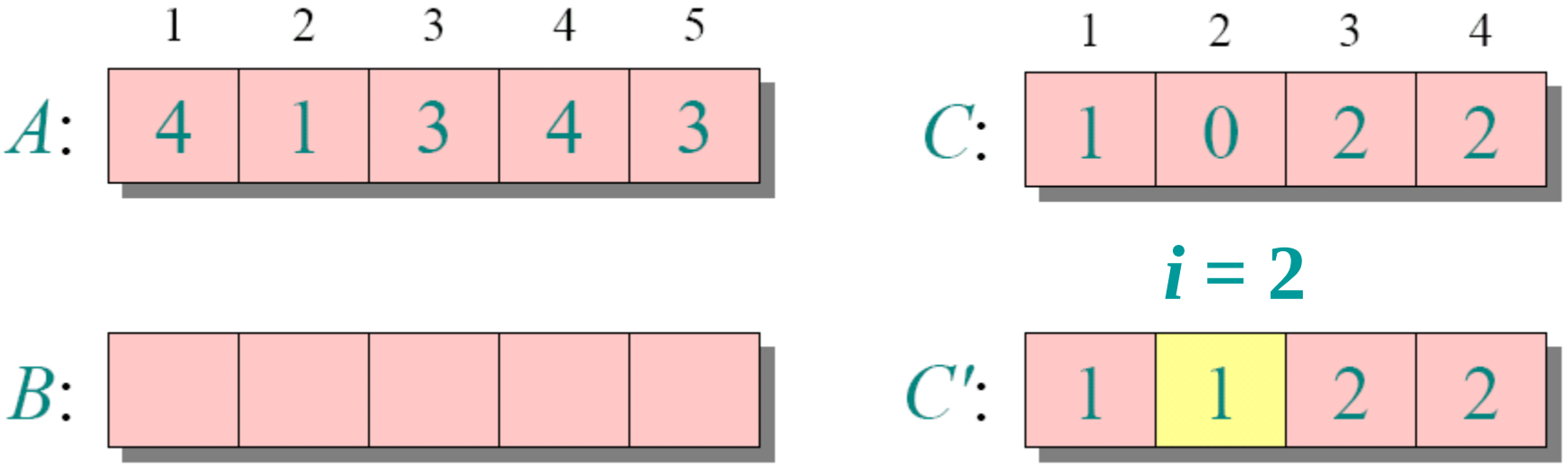
Minden j=1,n **végezd el:**

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

{ C_i = az i értékű elemek száma }

vége(minden)

Leszámláló rendezés (példa)



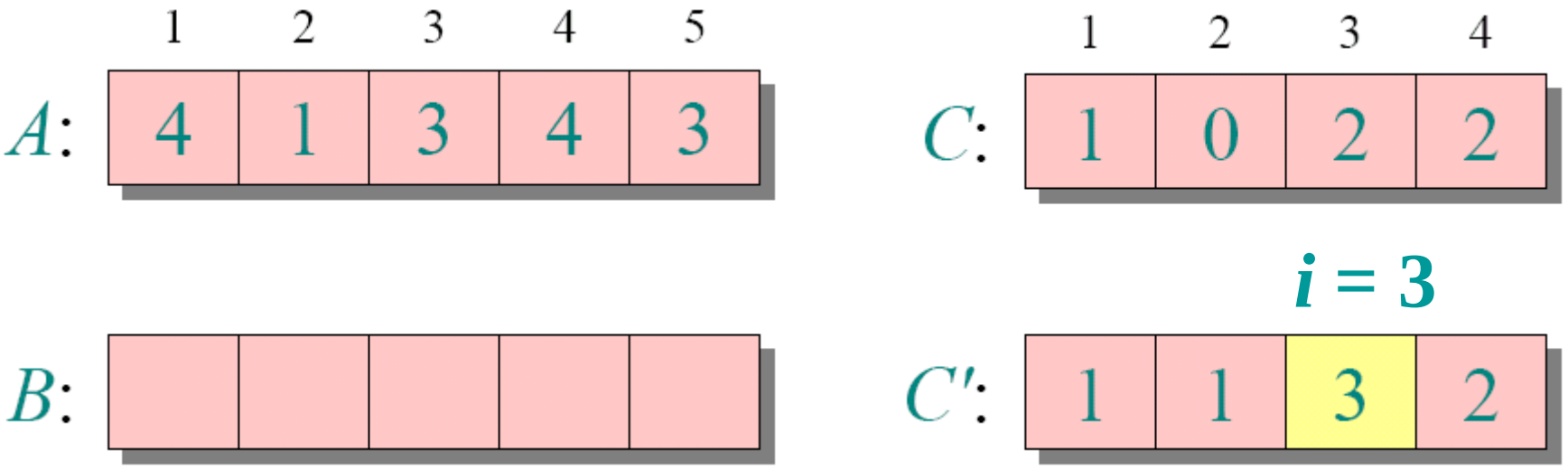
Minden i=2,k **végezd el:**

$C_i \leftarrow C_i + C_{i-1}$ **{ C_i = i-nél kisebb vagy }**

{ i-vel egyenlő elemek száma }

vége(minden)

Leszámláló rendezés (példa)



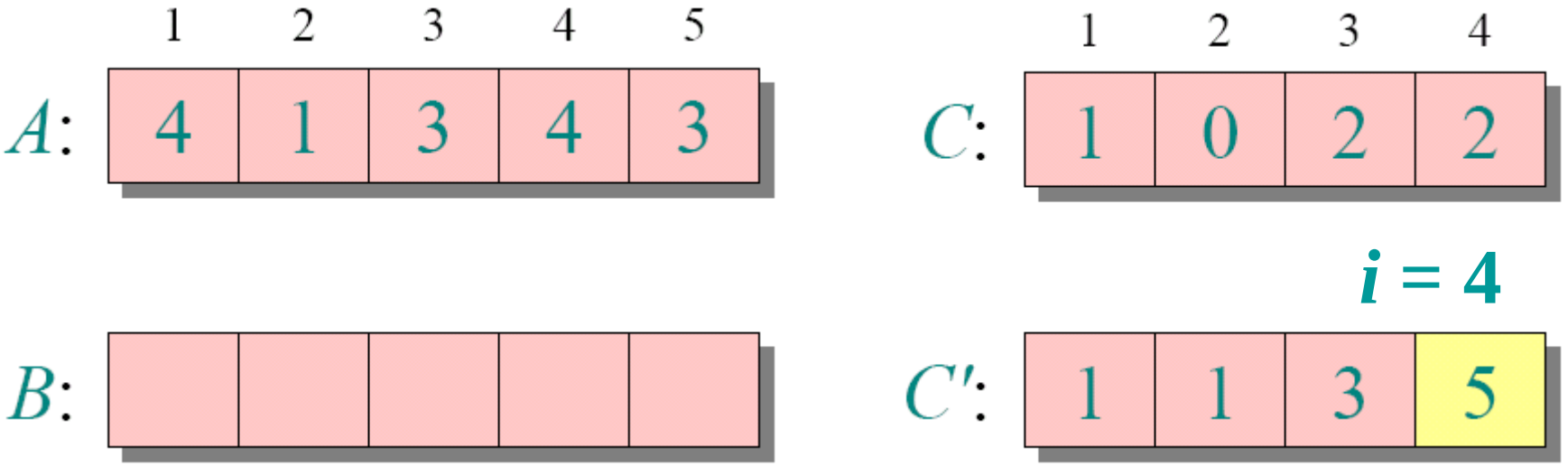
Minden i=2,k **végezd el:**

$C_i \leftarrow C_i + C_{i-1}$ **{ C_i = i-nél kisebb vagy }**

{ i-vel egyenlő elemek száma }

vége(minden)

Leszámláló rendezés (példa)



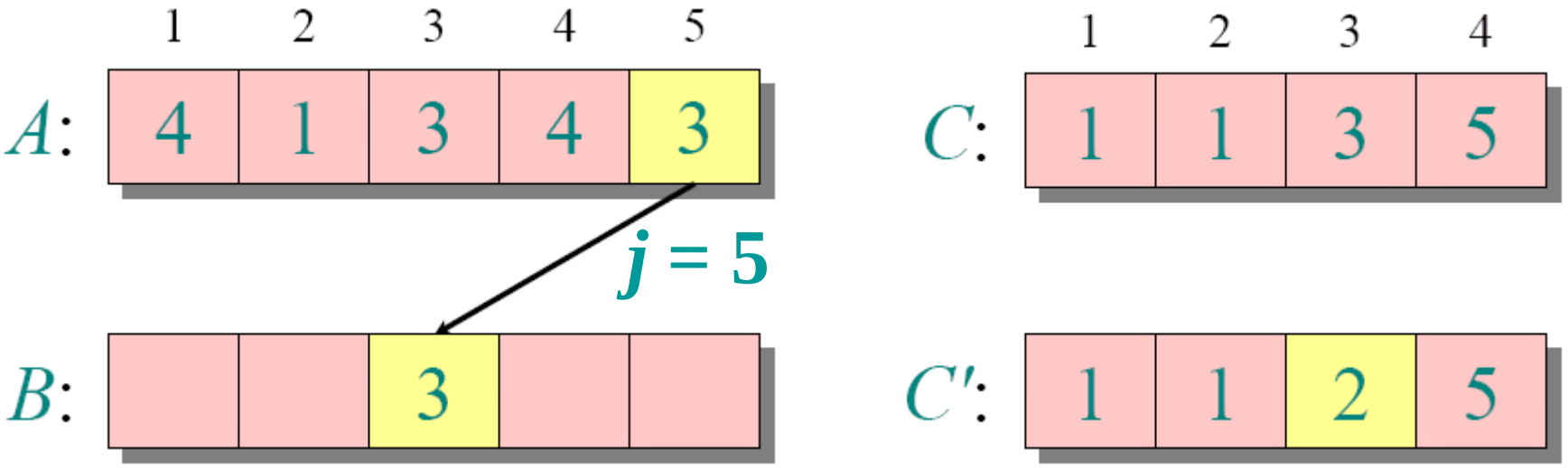
Minden i=2,k **végezd el:**

$C_i \leftarrow C_i + C_{i-1}$ **{ C_i = i-nél kisebb vagy }**

{ i-vel egyenlő elemek száma }

vége(minden)

Leszámláló rendezés (példa)



Minden j=n,1 **végezd el:**

$BC_{A_j} \leftarrow A_j$

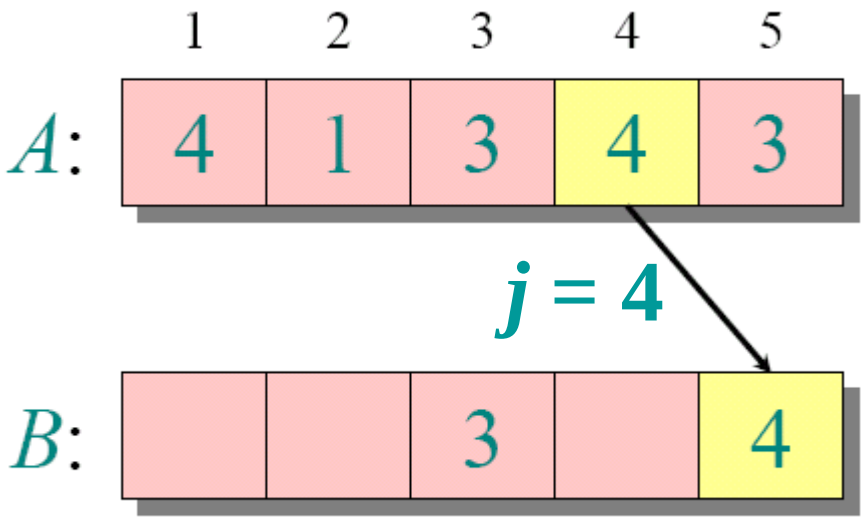
$C_{A_j} \leftarrow C_{A_j} - 1$

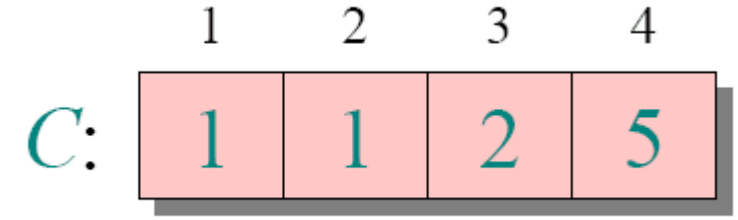
vége(minden)

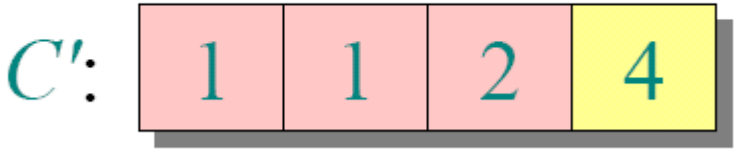
$A_5 = 3 \Rightarrow C_3 = 3 \Rightarrow B_3 = A_5$

és $C'_3 = 3 - 1 = 2$

Leszámláló rendezés (példa)







Minden j=n,1 **végezd el:**

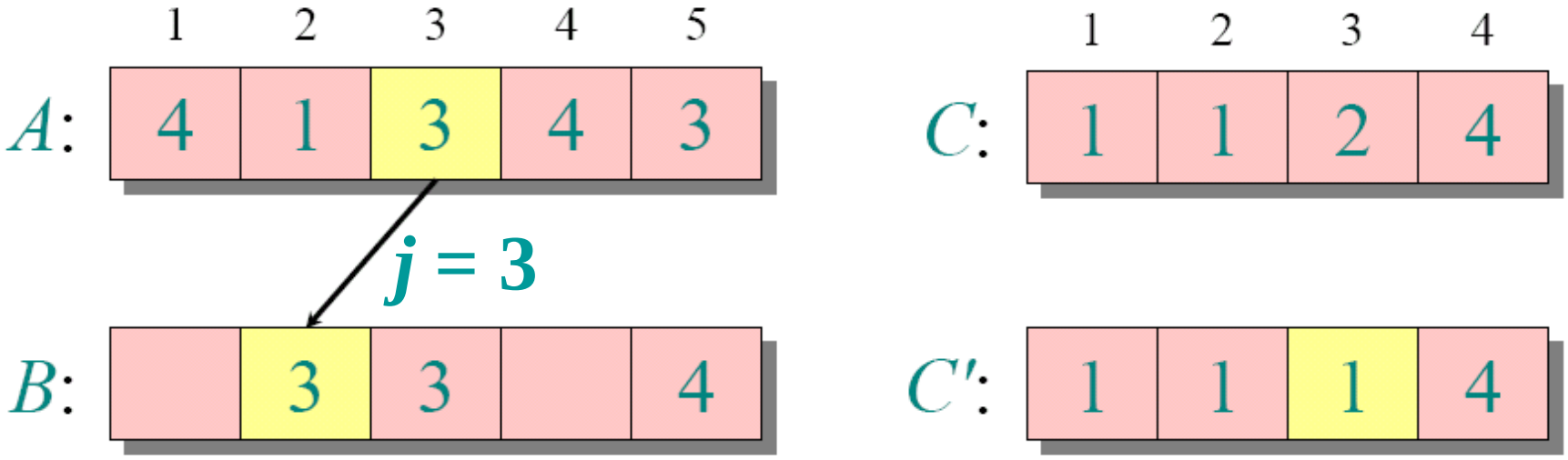
$B_{C_{A_j}} \leftarrow A_j$

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

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 **végezd el:**

$B_{C_{A_j}} \leftarrow A_j$

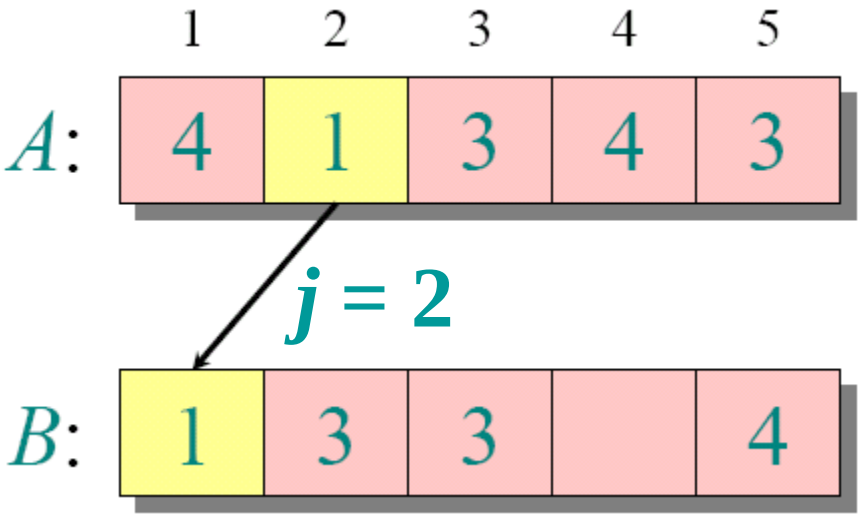
$C_{A_j} \leftarrow C_{A_j} - 1$

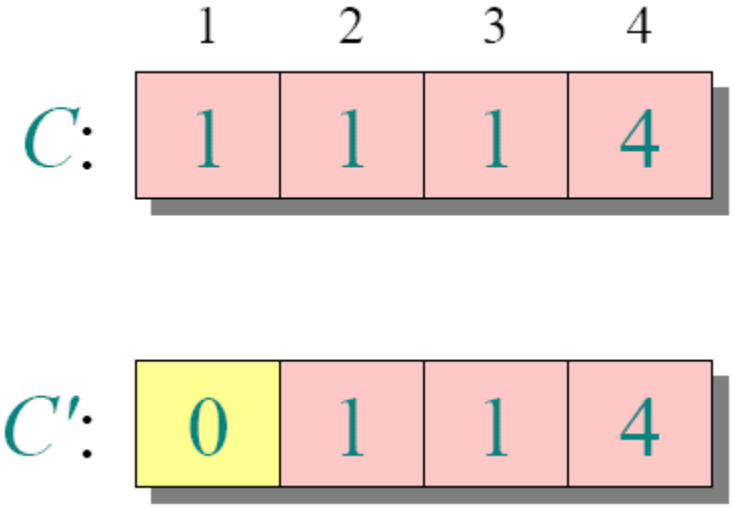
vége(minden)

$A_3 = 3 \Rightarrow C_3 = 2 \Rightarrow B_2 = A_3$

és $C'_3 = 2 - 1 = 1$

Leszámláló rendezés (példa)





Minden j=n,1 **végezd el:**

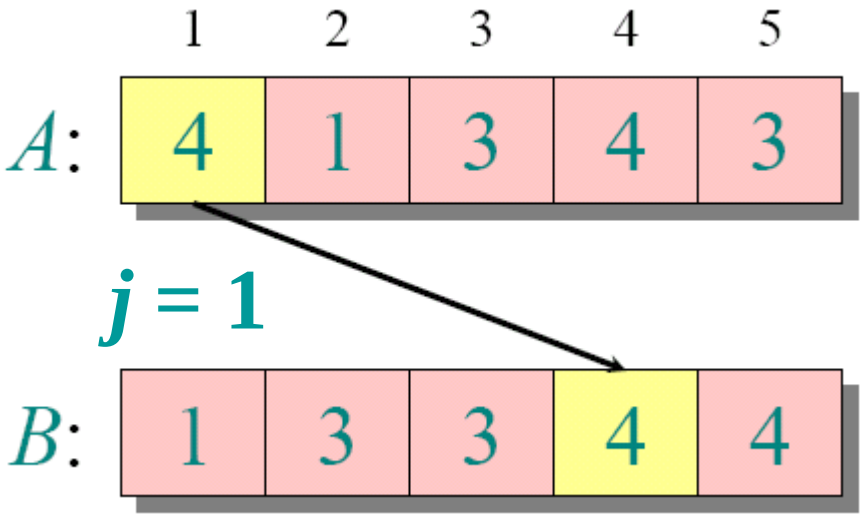
$B_{C_{A_j}} \leftarrow A_j$

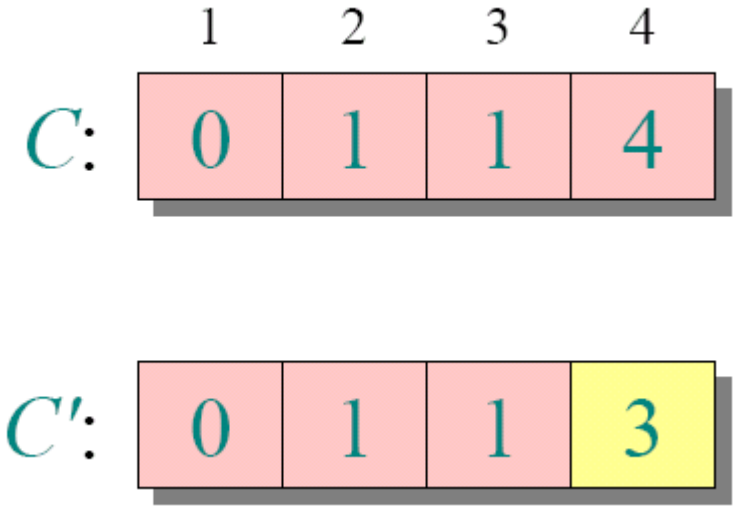
$C_{A_j} \leftarrow C_{A_j} - 1$

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 **végezd el:**

$B_{C_{A_j}} \leftarrow A_j$

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

vége(minden)

$A_1 = 4 \Rightarrow C_4 = 4 \Rightarrow B_4 = A_1$

és $C'_4 = 4 - 1 = 3$