

Rendező algoritmusok

Ionescu Klára

clara@cs.ubbcluj.ro

Rendezések

Adott az N elemű X sorozat, amelynek keressük azt a permutációját, amelyben $X_1 \leq X_2 \leq \dots \leq X_n$ (vagy \geq 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. Beszűrő 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, \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...

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.

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
≥
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 \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)

Programok rendezés\Cpp\01_BubbleSort.cpp

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 Javított_Buborékrende­zés_1(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**

$rendben \leftarrow hamis$

$id \leftarrow a_i; a_i \leftarrow a_{i+1}; a_{i+1} \leftarrow id$

vége(ha)

vége(minden)

$nn \leftarrow nn - 1$

ameddig $rendben$

Vége(algoritmus)

Programok rendezés\Cpp\02 Javitott 1 BubbleSort.cpp

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 Javitott_Buborékrendezés_2(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**

$rendben \leftarrow hamis$

$id \leftarrow a_i$; $a_i \leftarrow a_{i+1}$; $a_{i+1} \leftarrow id$

$k \leftarrow i$ *{ az utolsó csere helye }*

vége(ha)

vége(minden)

ameddig $rendben$

Vége(algoritmus)

Programok rendezés\Cpp\03_Javitott_2_BubbleSort.cpp

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 $\leftarrow 1$ *{ a vizsgálándó sorozat bal széle }*

régijobb $\leftarrow n - 1$ *{ a jobb szél indexe }*

Ismételd

rendben $\leftarrow igaz$

jobb $\leftarrow 0$ *{ az utolsó csere helye }*

Minden $i = \text{régibal}, \text{régijobb}$ **végezd el:**

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

rendben $\leftarrow hamis$

$id \leftarrow a_i : a_i \leftarrow a_{i+1} : a_{i+1} \leftarrow id$

Ha $i > jobb$ **akkor** jobb $\leftarrow i$

{ megjegyeztük az utolsó csere helyét }

vége(ha)

vége(ha)

vége(minden)

Ha nem rendben akkor { *volt csere* }
 régijobb \leftarrow jobb { *aktualizáljuk a jobb értékét* }
rendben \leftarrow igaz { *a bal szél* }

Minden $i = \text{régijobb}$, régibal **végezd el:**

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

 rendben \leftarrow hamis

 id $\leftarrow a_i$; $a_i \leftarrow a_{i+1}$; $a_{i+1} \leftarrow \text{id}$

Ha $i < \text{bal}$ **akkor** bal $\leftarrow i$

vége(ha)

vége(ha)

vége(minden)

 régibal \leftarrow bal { *aktualizáljuk a bal értékét* }

vége(ha)

ameddig rendben

Vége(algoritmus)

Programok rendezés\Cpp\04 Javitott 3 BubbleSort.cpp

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

$\text{segéd} \leftarrow a_i$

$a_i \leftarrow a_j$

$a_j \leftarrow \text{segéd}$

vége(ha)

vége(minden)

vége(minden)

Vége(algoritmus)

[Programok rendezés\Cpp\05_Felcsereleses.cpp](#)

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**

$k \leftarrow k + 1$ { megszámloljuk az a_i -nél $<$ elemeket }

vége(ha)

vége(minden)

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

{ a_i a rendezett sorozat megfelelő helyére kerül }

vége(minden)

$a \leftarrow \text{rendezett_sor}$

{ a rendezett sorozatot rámásoljuk az eredetire }

Vége(algoritmus)

[Programok rendezés\Cpp\06_SzámlálósRend_1.cpp](#)

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:**

$\text{ind_min} \leftarrow i$

Minden $j = i+1, n$ **végezd el:**

Ha $a_{\text{ind_min}} > a_j$ **akkor**

$\text{ind_min} \leftarrow j$

vége(ha)

vége(minden)

$\text{segéd} \leftarrow a_i$

$a_i \leftarrow a_{\text{ind_min}}$

$a_{\text{ind_min}} \leftarrow \text{segéd}$

vége(minden)

Vége(algoritmus)

[Programok rendezés\Cpp\08_MinimumRend.cpp](#)

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 Beszűrő_Rendezés(n, a):

Minden $j = 2, n$ **végezd el:**

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)

[Programok rendezés\Cpp\10_BeszuroRendezes.cpp](#)

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):

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:**

Minden $j = 1, \text{segéd}_i$ **végezd el:**

$q \leftarrow q + 1$

{ a segéd tömbnek k eleme van, }

$a_q \leftarrow i$

{ segéd_i elemek összege n }

vége(minden)

{ tehát a feldolgozások száma n }

vége(minden)

vége(algoritmus)

[Programok rendezés\Cpp\11_LadaRendezes.cpp](#)

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.

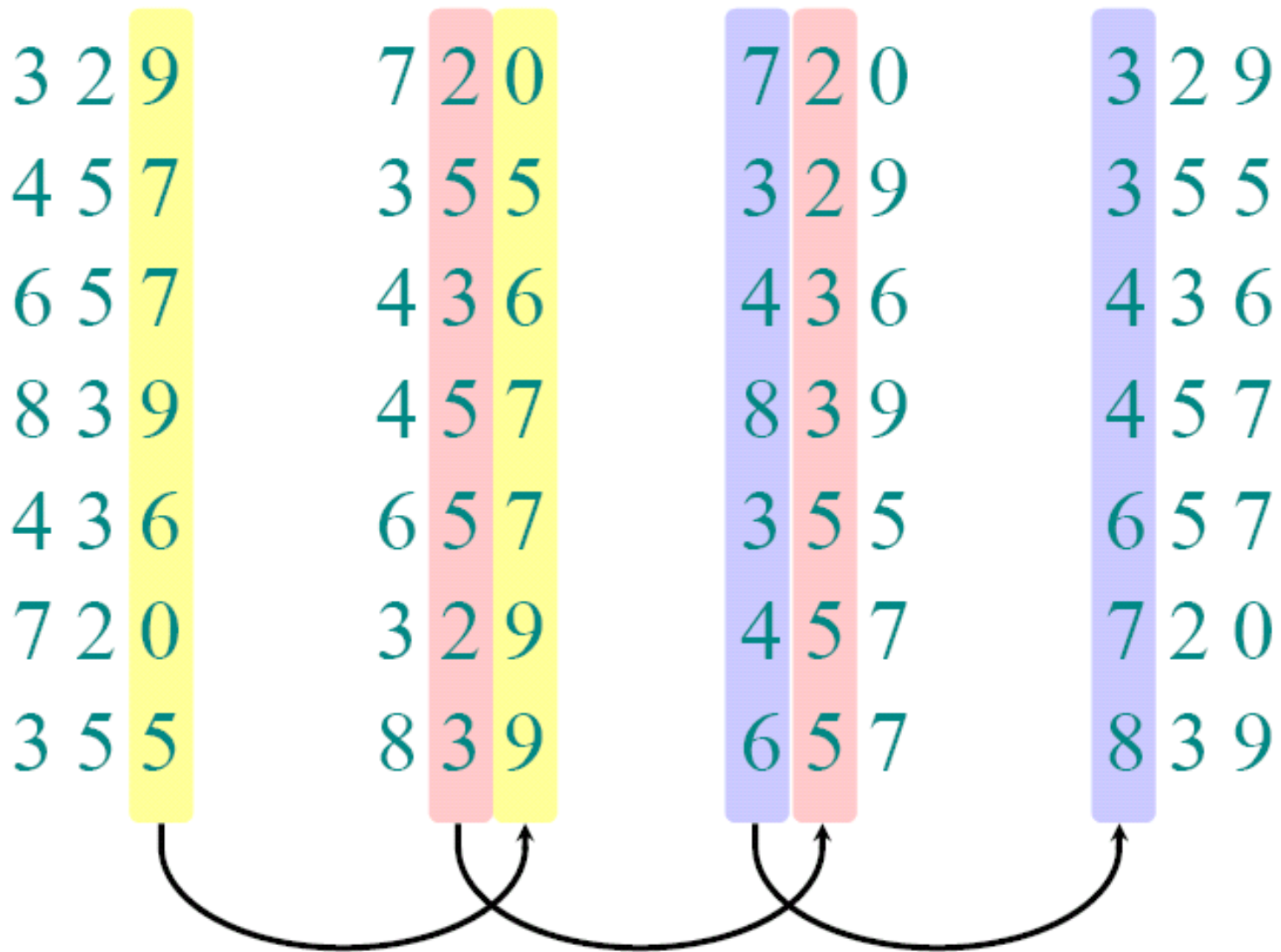
Számjegyes rendezés

- 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 legkevesbé 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



Algoritmus Számjegyes_Rendezés(a, d):

Minden $i = 1, d$ **végezd el:**

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

vége(minden)

vége(algoritmus)

Programok rendezés\Cpp\13_RadixSort.cpp

STABIL Leszámláló (láda)rendezés

1. Ha a rendezendő adatok a kulcson kívül tartalmazznak 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:**

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

vége(minden)

Minden $j = 1, n$ **végezd el:**

$C_{A_j} \leftarrow C_{A_j} + 1$ { $C_i = i$ 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:**

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

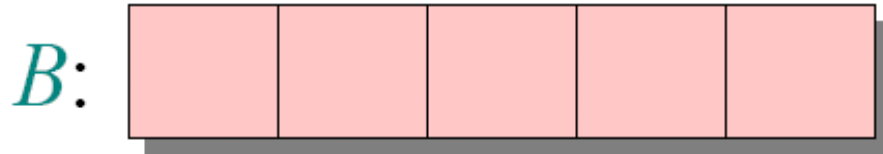
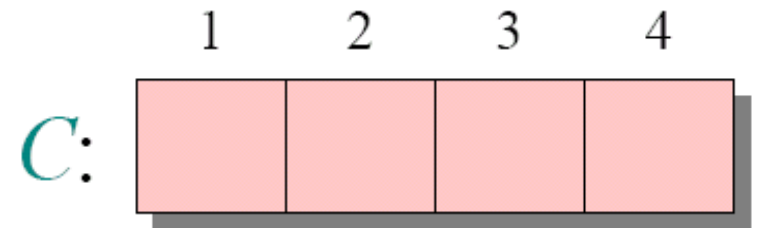
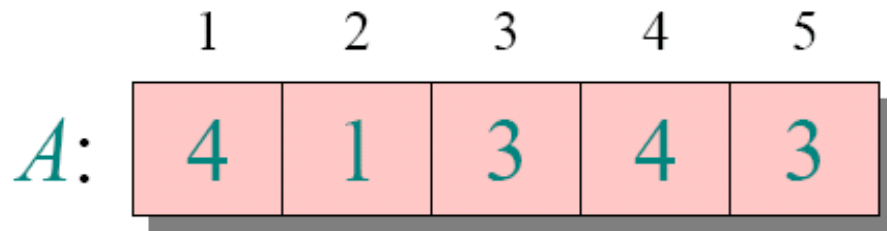
$C_{A_j} \leftarrow C_{A_j} - 1$

vége(minden)

Vége(algoritmus)

[Programok rendezés\Cpp\12 StabilladaRendezes.cpp](#)

Leszámláló rendezés (példa)



Leszámláló rendezés (példa)

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

	1	2	3	4
C :	0	0	0	0

B :					
-------	--	--	--	--	--

Minden $i=1, k$ végezd el:

$C_i \leftarrow 0$ { még nem számoltunk egyetlen }
 { i -vel egyenlő elemet sem }

vége(minden)

Leszámláló rendezés (példa)

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

	1	2	3	4
C :	0	0	0	1

$j = 1$

B :					
-------	--	--	--	--	--

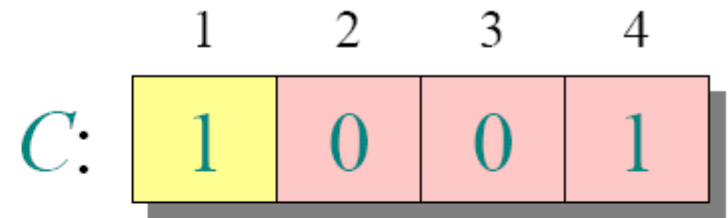
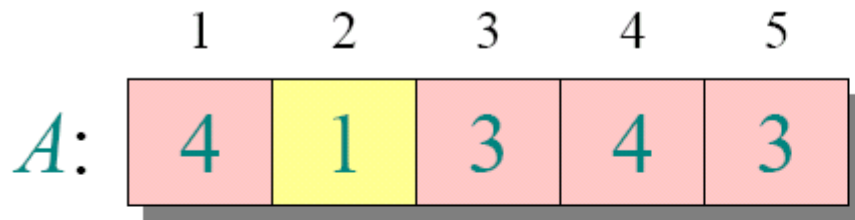
Minden $j=1,n$ végezd el:

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

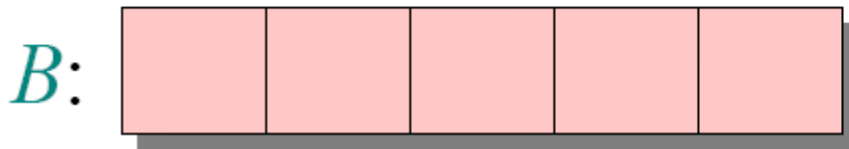
{ C_i = az i értékű elemek száma }

vége(minden)

Leszámláló rendezés (példa)



$j = 2$



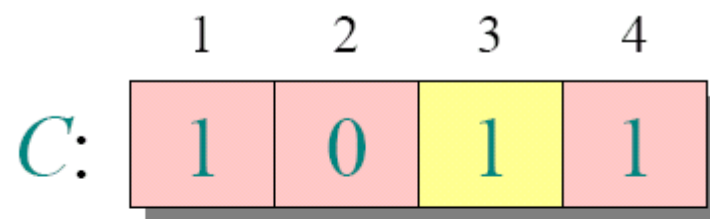
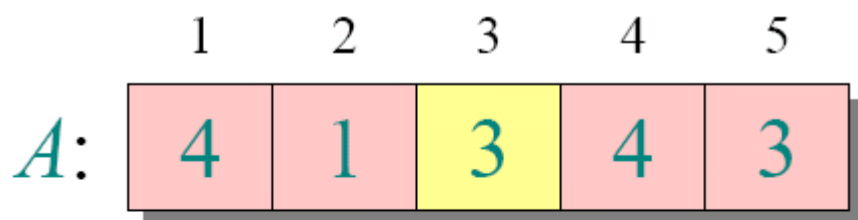
Minden $j=1,n$ végezd el:

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

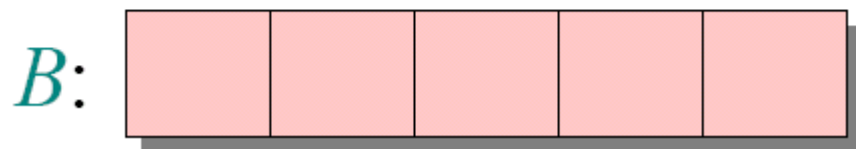
{ C_i = az i értékű elemek száma }

vége(minden)

Leszámláló rendezés (példa)



$j = 3$



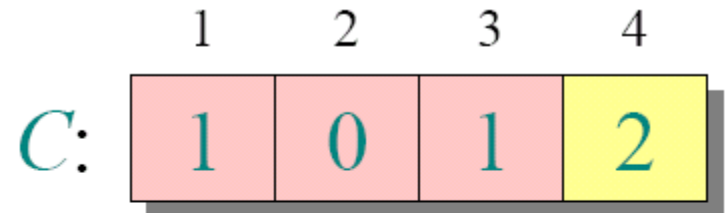
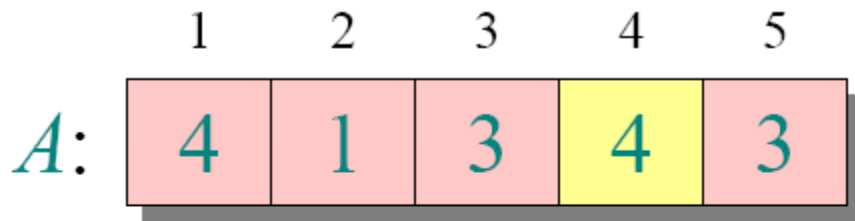
Minden $j=1,n$ végezd el:

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

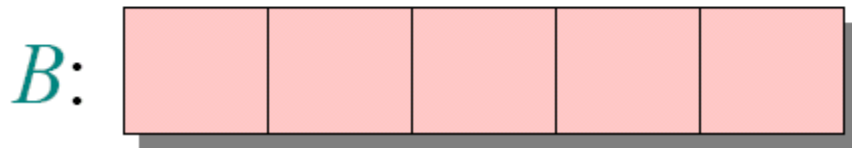
{ C_i = az i értékű elemek száma }

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)

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

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

$j = 5$

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

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
$A:$	4	1	3	4	3

B :

--	--	--	--	--

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

$$i = 2$$

C' :

1	1	2	2
---	---	---	---

Minden $i=2, k$ végezd el:

$$C_i \leftarrow C_i + C_{i-1} \quad \{ C_i = i\text{-nél nagyobb vagy} \}$$

{ i-vel egyenlő 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>B</i> :					
------------	--	--	--	--	--

i = 3

<i>C'</i> :	1	1	3	2
-------------	---	---	---	---

Minden $i=2,k$ végezd el:

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

$\{ i\text{-vel egyenlő elemek száma} \}$

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

 $i = 4$

B :

--	--	--	--	--

C' :

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

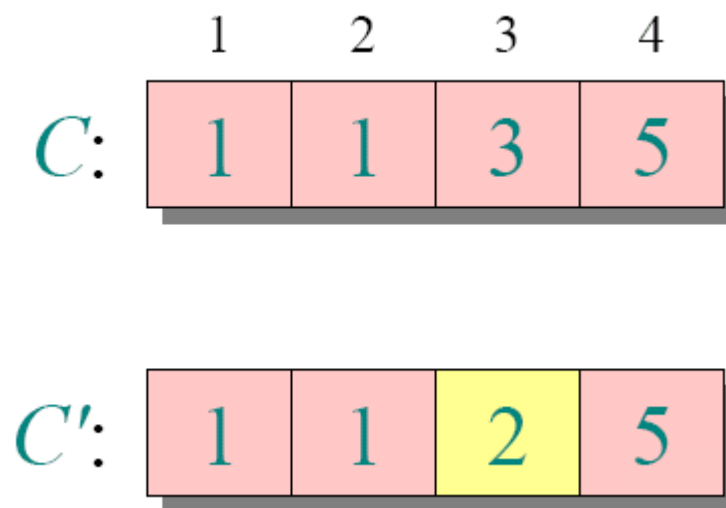
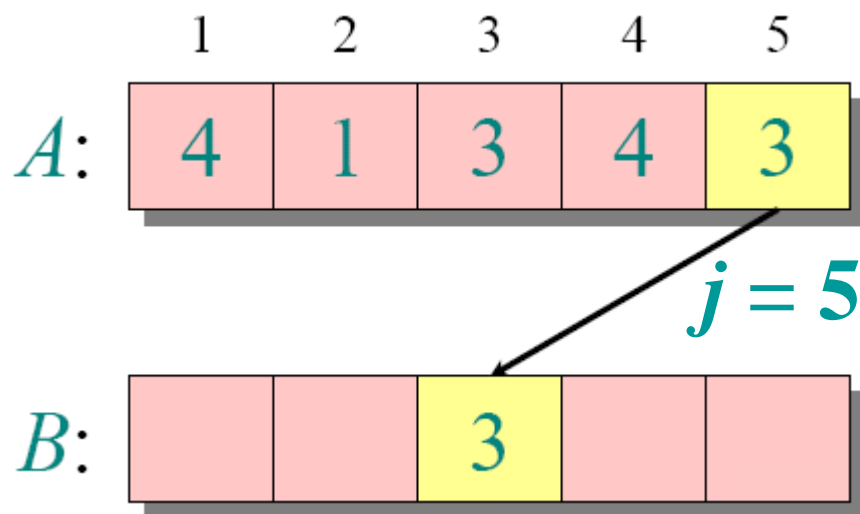
Minden $i=2, k$ végezd el:

$$C_i \leftarrow C_i + C_{i-1} \quad \{ C_i = i\text{-nél nagyobb vagy} \}$$

{ i-vel egyenlő elemek száma }

vége(minden)

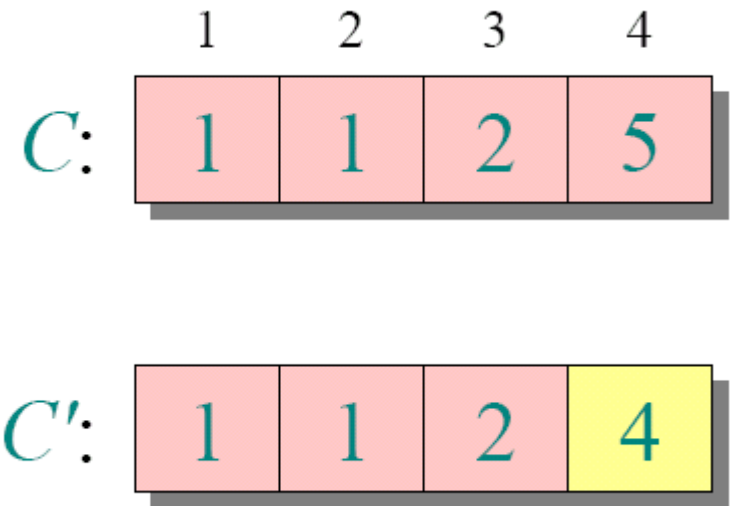
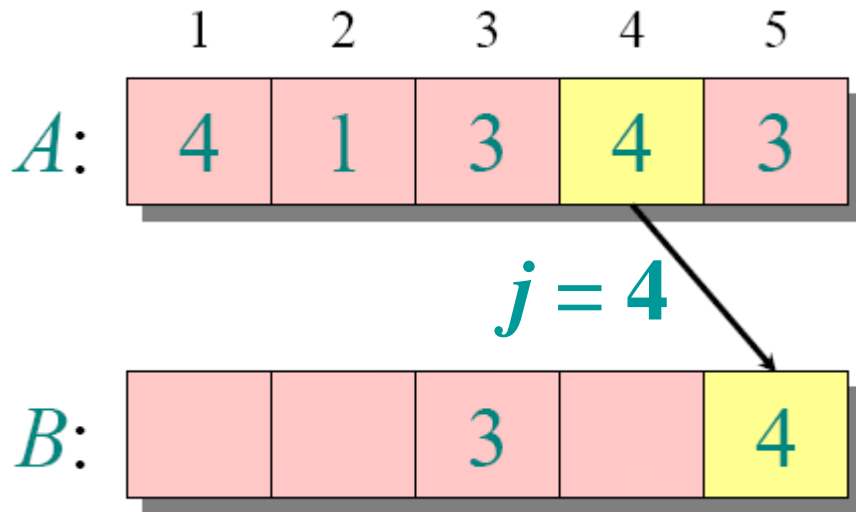
Leszámláló rendezés (példa)



Minden $j=n,1$ végezd el: $A_5 = 3 \Rightarrow C_3 = 3 \Rightarrow B_3 = A_5$
és $C'_3 = 3 - 1 = 2$

$B_{CA_j} \leftarrow A_j$
 $C_{A_j} \leftarrow C_{A_j} - 1$
vége(minden)

Leszámláló rendezés (példa)



Minden $j=n,1$ végezd el:

$$B_{CA_j} \leftarrow A_j$$

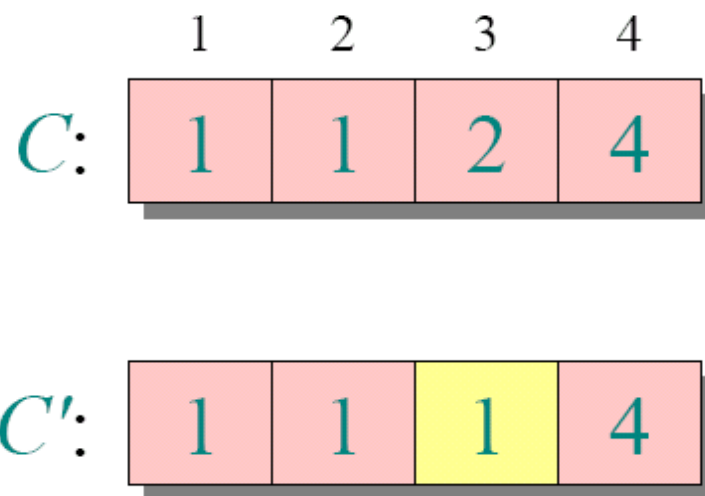
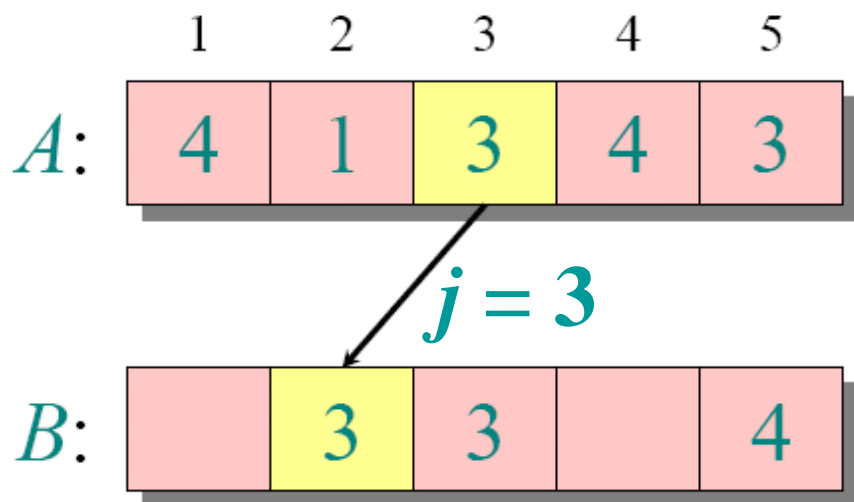
$$C_{A_j} \leftarrow C_{A_j} - 1$$

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:

$$B_{CA_j} \leftarrow A_j$$

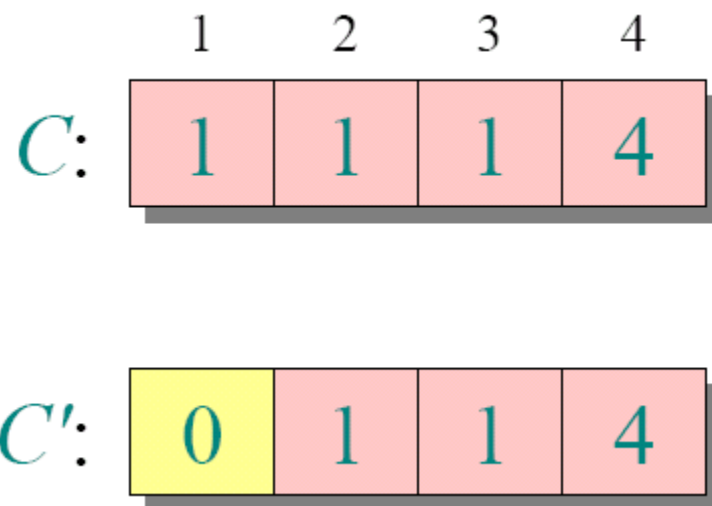
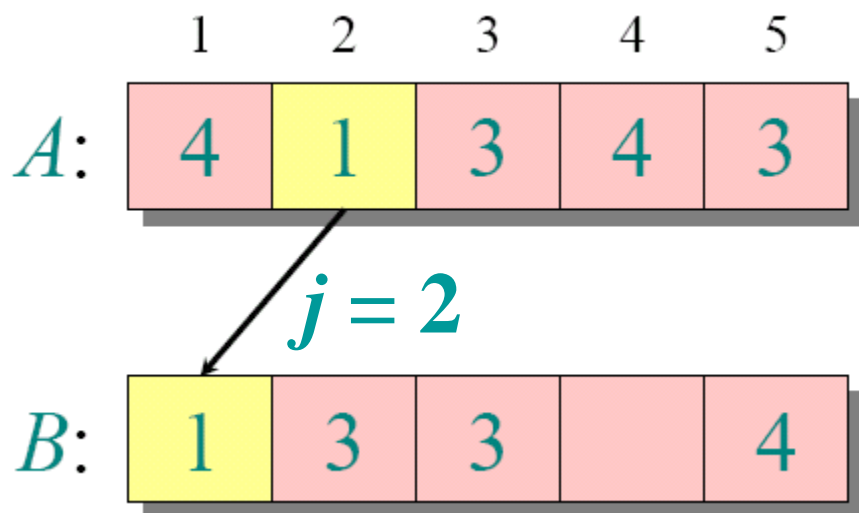
$$C_{A_j} \leftarrow C_{A_j} - 1$$

vége(minden)

$$\mathbf{A_3 = 3 \Rightarrow C_3 = 2 \Rightarrow B_2 = A_3}$$

és $\mathbf{C'_3 = 2 - 1 = 1}$

Leszámláló rendezés (példa)



Minden $j=n,1$ végezd el:

$$B_{CA_j} \leftarrow A_j$$

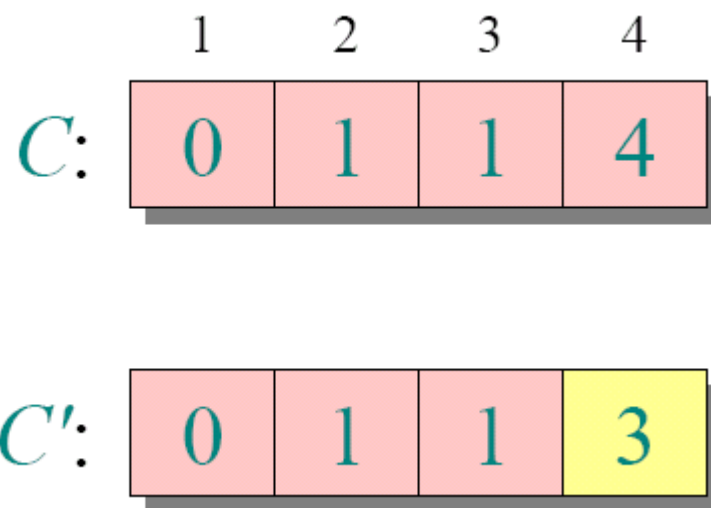
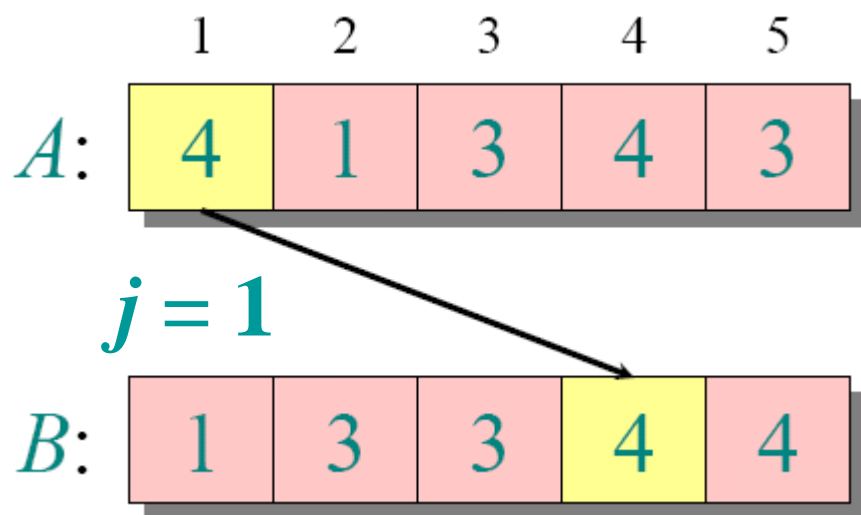
$$C_{A_j} \leftarrow C_{A_j} - 1$$

vége(minden)

$$\mathbf{A_2 = 1 \Rightarrow C_1 = 1 \Rightarrow B_1 = A_2}$$

és $\mathbf{C'_1 = 1 - 1 = 0}$

Leszámláló rendezés (példa)



Minden $j=n,1$ végezd el:

$$B_{CA_j} \leftarrow A_j$$

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

vége(minden)

$$\mathbf{A_1 = 4 \Rightarrow C_4 = 4 \Rightarrow B_4 = A_1}$$

és $\mathbf{C'_4 = 4 - 1 = 3}$