4 | Rendezés

Korábban már láttunk egy rendező algoritmust, a kiválasztásos rendezést. Ebben a fejezetben további rendező algoritmusokkal fogunk megismerkedni. Ezek többsége úgynevezett összehasonlítás alapú rendezés lesz, azaz a rendezést csak az alapján fogjuk tudni végezni, hogy a tömb két elemét egymással összehasonlítjuk.

Buborékrendezés

A gyakorlaton már találkoztunk azzal a feladattal, amikor egy tömb legkisebb elemét úgy kerestük meg, hogy végigmentünk a tömbön, sorra összehasonlítottuk minden cella tartalmát az utána következővel, és ha az elöl álló elem kisebb volt, mint a hátul álló, akkor megcseréltük őket és végül visszatértünk az utolsó cella tartalmával. Nézzük most meg ugyanezt a feladatot, csak legkisebb helyett legnagyobb elemmel.

4.1. feladat.

Rendezzük át a T tömb elemeit úgy, hogy a legnagyobb elem az utolsó cellába kerüljön.

1. algoritmus: Legnagyobb elem tömb végére mozgatása

Input: egy $n \geq 1$ darab különböző számot tartalmazó T tömb
Output: T elemeinek olyan átrendezése, amiben a legnagyobb elem az utolsó cellában van

ciklus: i **fut** 0**-tól** $(n-2)$**-ig**
  ha $T[i] > T[i+1]$**:**
    csere$\big(T[i], T[i+1]\big)$
ciklus vége
return T

Helyesség. Az algoritmus jó kimenetet ad, mert minden cellát megnézünk, és ha elérünk a legnagyobb elemig, akkor azt addig mozgatjuk jobbra, amíg a tömb utolsó cellájába nem kerül. Az algoritmus véget ér, mert a tömb véges hosszúságú.

Lépésszám. A ciklusmag $O(n)$-szer fut le és minden lefutáskor $O(1)$ lépést teszünk, majd $O(1)$ lépésben visszatérünk az eredménnyel, vagyis az algoritmus teljes lépésszáma $O(n) \cdot O(1) + O(1) = O(n)$.

Ezt az eljárást használva rendezhetjük az egész tömböt: először a legnagyobb elemet mozgatjuk a

helyére, azaz az utolsó cellába, majd a második legnagyobb elemet mozgatjuk ugyanígy a helyére, azaz az utolsó előtti cellába, stb. Ehhez az 1. algoritmust először a $T[0:n-1]$ tömbön futtatjuk le, majd az így kapott tömb $T[0:n-2]$ résztömbjén, stb. Az utolsó lépésben a $T[0:1]$ résztömbben elég lesz a nagyobbik elemet a helyére mozgatni, hiszen ezzel már az egész tömb rendezetté válik. Azt, hogy éppen melyik résztömbben dolgozunk, a kiválasztásos rendezéshez hasonlóan egy külső ciklussal fogjuk szabályozni.

2. algoritmus: Buborékrendezés

Input: egy $n \geq 1$ darab különböző számot tartalmazó T tömb
Output: növekvően rendezett T tömb

ciklus: j **fut** $(n-1)$**-től** 1**-ig**
  ciklus: i **fut** 0**-tól** $(j-1)$**-ig**
    ha $T[i] > T[i+1]$**:**
      csere$\big(T[i], T[i+1]\big)$
  ciklus vége
ciklus vége
return T

Helyesség. Az algoritmus véget ér, hiszen az 1. algoritmust futtatjuk le véges sokszor. Az algoritmus jó kimenetet ad: az első fázisban a legnagyobb elem bekerül az utolsó cellába, és utána az utolsó cellával már nem csinálunk semmit; a második fázisban a második legnagyobb elem kerül be az utolsó előtti cellába, és utána ezzel a cellával már nem csinálunk semmit, stb.

Lépésszám. A külső ciklus $O(n)$-szer fut le, minden lefutáskor $O(n)$-szer futtatjuk le a belső ciklust, aminek minden lefutásakor $O(1)$ lépést teszünk. A végén $O(1)$ lépésben visszatérünk az eredménnyel. Ez összesen $O(n) \cdot O(n) \cdot O(1) + O(1) = O(n^2)$ lépés.

4.2. példa.

Nézzünk egy példát a buborékrendezésre.

Bemenet. A bemenet az alábbi $T[0:3]$ tömb.

T:

7	1	4	2

1. fázis.

Először összehasonlítjuk a $T[0] = 7$ és a $T[1] = 1$ elemeket, és mivel az előrébb álló a nagyobb, ezért megcseréljük őket.

T:

1	7	4	2

Ezután összehasonlítjuk az így kapott T tömbben a $T[1] = 7$ és a $T[2] = 4$ elemeket, és mivel az előrébb álló a nagyobb, ezért megcseréljük őket.

T:

1	4	7	2

Végül összehasonlítjuk az így kapott T tömbben a $T[2] = 7$ és a $T[3] = 2$ elemeket, és mivel az előrébb álló a nagyobb, ezért megcseréljük őket.

T:

1	4	2	7

2. fázis.

Innentől már csak a $T[0:2]$ résztömbbel dolgozunk tovább.
Először összehasonlítjuk a $T[0] = 1$ és a $T[1] = 4$ elemeket, és mivel az előrébb álló kisebb, ezért nem cseréljük meg őket. Ezután összehasonlítjuk a $T[1] = 4$ és a $T[2] = 2$ elemeket, és mivel az előrébb álló a nagyobb, ezért megcseréljük őket.

T:

1	2	4	7

3. fázis.

Már csak a $T[0:1]$ résztömbbel dolgozunk tovább. Összehasonlítjuk a $T[0] = 1$ és a $T[1] = 2$ elemeket, és mivel az előrébb álló kisebb, ezért nem cseréljük meg őket.

Ezzel az algoritmus véget ér.

Beszúrásos rendezés

Tekintsük az alábbi feladatot.

4.3. feladat.
Egy tömbben növekvően rendezetten tároltunk n darab számot, de a tömb utolsó elemét valaki megváltoztatta. Rendezzük $O(n)$ lépésben az új tömböt.
Megoldás. Mozgassuk a tömb utolsó elemét balra addig, amíg a helyére nem kerül.

3. algoritmus: Az utolsó elem kivételével rendezett tömb rendezése

Input: egy $n \geq 1$ darab különböző számot tartalmazó T tömb, aminek a $T[0 : n-1]$ résztömbje már növekvően rendezett
Output: növekvően rendezett T tömb

$i \coloneqq n - 1$
ciklus: amíg $T[i-1] > T[i]$ **és** $i > 0$
  csere$\big(T[i-1], T[i]\big)$
  $i \coloneqq i - 1$
ciklus vége
return T

Helyesség. Az algoritmus összehasonlítja az utolsó cella tartalmát a tömb többi elemével hátulról visszafelé haladva, és ha kell, akkor megcseréli őket. Ha az eredetileg utolsó szám már nem kisebb az éppen előtte állónál vagy az utolsó számmal elértük a tömb elejét, akkor (a $T[0 : n-1]$ tömb rendezettsége miatt) a kapott tömb már teljesen rendezett lesz. Az algoritmus biztosan véget ér, legkésőbb a tömb elején.

Lépésszám. A ciklusmag legfeljebb $(n-1)$-szer, azaz $O(n)$-szer fut le, hiszen i értéke kezdetben $n-1$, amit egyesével csökkentünk legfeljebb addig, amíg 0 nem lesz (az utolsó lefutáskor az értéke 1). Minden lefutáskor $O(1)$ lépést végzünk (két feltétel ellenőrzését, egy cserét, és egy értékadást). A végén még $O(1)$ lépésben ellenőrizzük a feltételek nem-teljesülését, majd visszaadjuk az eredményt. Ez összesen $O(n) \cdot O(1) + O(1) = O(n)$ lépés.

A beszúrásos rendezés úgy működik, hogy ha a tömb elején lévő néhány elemet már rendezte, akkor a tömb következő elemét a 3. algoritmussal a helyére mozgatja. Nyilván az egyelemű $T[0 : 0]$ résztömbre tekinthetünk rendezett tömbként. Ezután meghívjuk a 3. algoritmust a $T[0 : 1]$ résztömbre, majd az átrendezés után kapott $T[0 : 2]$ résztömbre, és így tovább, egészen addig, míg az egész $T[0 : n-1]$ tömbre meg tudjuk hívni. Az eljárás végére így rendezni fogjuk az egész T tömböt.

4. algoritmus: Beszúrásos rendezés

Input: egy $n \geq 1$ darab különböző számot tartalmazó T tömb
Output: növekvően rendezett T tömb

ciklus: j **fut** 1**-től** $(n-1)$**-ig**
  $i := j$
  ciklus: amíg $T[i-1] > T[i]$ **és** $i > 0$
    csere$\big(T[i-1], T[i]\big)$
    $i := i - 1$
  ciklus vége
ciklus vége
return T

Helyesség. A 3. algoritmust mindig egy olyan résztömbre hívjuk meg, ami az utolsó elem kivételével már rendezett, amit aztán ez az algoritmus teljesen rendez. Az utolsó fázisban a teljes T tömbre hívjuk meg a 3. algoritmust, ezért az eljárás végén az egész tömb rendezett lesz.

Mivel $(n-1)$-szer hívjuk meg a 3. algoritmust, amiről már láttuk, hogy véget ér, ezért ez az eljárás is biztosan véget fog érni.

Lépésszám. A külső ciklusmag $(n-1)$-szer, azaz $O(n)$-szer fut le. Minden lefutáskor végzünk $O(1)$ lépést (értékadás) és lefuttatjuk legfeljebb $(j-1)$-szer, azaz $j \leq n-1$ miatt $O(n)$-szer a belső ciklust. A belső ciklusmag minden lefutásakor $O(1)$ lépést végzünk (ahogy ezt már a 3. algoritmus elemzésénél láttuk), és még $O(1)$ lépésben ellenőrizzük a feltételek nem-teljesülését. Legvégül $O(1)$ lépésben visszaadjuk az eredményt. Ez összesen $O(n) \cdot \big(O(1) + O(n) \cdot O(1) + O(1)\big) + O(1) = O(n^2)$ lépés.

4.4. példa.
Nézzünk egy példát a beszúrásos rendezésre.
Bemenet. A bemenet az alábbi $T[0:3]$ tömb.

T:

7	1	4	2

1. fázis.

Először úgy tekintünk a tömbre, aminek a $T[0:0]$ része már rendezett, a vége (azaz a $T[1:3]$ résztömb) pedig még rendezetlenül tartalmazza az elemeket. Az első lépésben a helyére mozgatjuk a $T[1]$ elemet. Ehhez összehasonlítjuk a $T[1] = 1$ elemet az előtte álló $T[0] = 7$ elemmel, és mivel kisebb lesz nála, ezért ezt a két elemet megcseréljük.

T: | 1 | 7 | 4 | 2 |

Ezzel az 1-es a tömb elejére kerül, így már biztosan nem áll előtte olyan elem, aminél kisebb lenne (hiszen semmilyen elem nem áll előtte). Vagyis most már a kapott $T[0:1]$ résztömb rendezett.

2. fázis.

Most a $T[0:1]$ rendezett résztömbbe szúrjuk be a $T[2]$ elemet. Ehhez először összehasonlítjuk a $T[2] = 4$ és a $T[1] = 7$ elemeket, és mivel az előrébb álló a nagyobb, ezért megcseréljük őket.

T: | 1 | 4 | 7 | 2 |

Ezután összehasonlítjuk a $T[1] = 4$ és a $T[0] = 1$ elemeket, és mivel az előrébb álló a kisebb, ezért nem cseréljük meg őket.

3. fázis.

Végül a $T[0:2]$ résztömbbe szúrjuk be a $T[3]$ elemet. Ehhez először összehasonlítjuk a $T[3] = 2$ és a $T[2] = 7$ elemeket, és mivel az előrébb álló a nagyobb, ezért megcseréljük őket.

T: | 1 | 4 | 2 | 7 |

Ezután összehasonlítjuk a $T[2] = 2$ és a $T[1] = 4$ elemeket, és mivel az előrébb álló a nagyobb, ezért megcseréljük őket.

T: | 1 | 2 | 4 | 7 |

Ezután összehasonlítjuk a $T[1] = 2$ és a $T[0] = 1$ elemeket, és mivel az előrébb álló a kisebb, ezért nem cseréljük meg őket.

Ezzel az algoritmus véget ér.

Összefésüléses rendezés

Tekintsük az alábbi feladatot.

4.5. feladat.

Adott két rendezett tömb $A[0 : k-1]$ és $B[0 : \ell-1]$, melyek együtt $k+\ell$ darab különböző számot tartalmaznak. Készítsünk egy olyan $C[0 : k+\ell-1]$ tömböt, ami az A-beli és B-beli tömbök elemeit növekvő sorrendben tartalmazza.

Megoldás.

5. algoritmus: Két rendezett tömb összefésülése

Input: egy növekvően rendezett $A[0 : k-1]$ tömb és egy növekvően rendezett $B[0 : \ell-1]$ tömb, melyek együtt $k+\ell$ darab különböző számot tartalmaznak
Output: egy növekvően rendezett $C[0 : k+\ell-1]$ tömb, ami az A-beli és B-beli tömbök elemeit tartalmazza

```
i := 0
j := 0
h := 0
ciklus: amíg i < k és j < ℓ
    ha A[i] < B[j]:
        C[h] := A[i]
        i := i + 1
        h := h + 1
    különben:
        C[h] := B[j]
        j := j + 1
        h := h + 1
ciklus vége
ha i == k:
    ciklus: amíg j < ℓ
        C[h] := B[j]
        j := j + 1
        h := h + 1
    ciklus vége
különben ha j == ℓ:
    ciklus: amíg i < k
        C[h] := A[i]
        i := i + 1
        h := h + 1
    ciklus vége
return C
```

Helyesség. Elindulunk az A és a B tömbök elejéről, és összehasonlítjuk az első elemeiket. Mivel külön-külön A és B is rendezett, ezért az első elemeik közül a kisebbik lesz az összes tárolt elem minimuma, amit tehát helyesen írunk be a C tömb elejére. Amelyik tömben volt ez

a legkisebb elem, abban eggyel tovább lépünk, és most azt az elemet hasonlítjuk össze a másik tömb aktuális (jelen esetben első) elemével. Ezt az eljárást folytatjuk, amíg az egyik tömb el nem fogy. Ekkor a másik tömbben maradt elemeket bemásoljuk a C tömb végére. Nyilván az eljárás minden fázisában

- a C tömb rendezetten tartalmazza az addig feldolgozott elemeket,
- a C-be addig beírt elemek mindegyike kisebb a még fel nem dolgozott A- és B-beli elemeknél,
- a még fel nem dolgozott elemek közül a legkisebb – a B és a C tömbök rendezettsége miatt – a B és a C tömb első fel nem dolgozott eleme közül a kisebbik.

Ezekből következik, hogy az eljárás végén a C tömb rendezetten fogja tartalmazni az összes A-beli és B-beli elemet.

Az eljárás véget ér, hiszen minden fázisban egy újabb elemet írunk be a C tömbbe, és mivel véges sok elem van, ezért a C tömb egyszer csak megtelik.

Lépésszám. Az algoritmus minden fázisában az i és j értéke közül pontosan az egyiket fogjuk eggyel növelni. Az eljárás akkor ér véget, amikor $i = k$ és $j = \ell$ teljesül. Mivel kezdetben $i = j = 0$, ezért az eljárásnak összesen $k + \ell$ fázisa van.

Az algoritmus elején van $O(1)$ értékadás (egészen pontosan 3 darab), majd minden fázis elején $O(1)$ összehasonlítással eldöntjük, hogy elértük-e már az A vagy a B tömb végét. Ha még nem értük el egyik tömb végét sem, akkor $O(1)$ összehasonlítással meghatározzuk, hogy melyik elem lesz a C tömb következő eleme; ha már elértük az egyik tömb végét, akkor erre nincsen szükség – de ebben az esetben is becsülhetjük a nulla darab lépést $O(1)$-gyel. Ezután $O(1)$ lépésben beírjuk a C tömbbe a következő elemet, majd $O(1)$ lépésben növeljük a megfelelő indexek értékét. Amikor már mindkét tömb végét elértük – ennek a megállapításához még szükség van $O(1)$ összehasonlításra –, akkor $O(1)$ lépésben visszaadjuk az eredményt. Mivel $k + \ell$ fázis van, ezért ez összesen

$$O(1) + O(k + \ell) \cdot \big(O(1) + O(1) + O(1) + O(1)\big) + O(1) + O(1) = O(k + \ell)$$

lépés.

Az 5. algoritmust felhasználhatjuk egy tömb rendezésére is: félbevágjuk a tömböt, rendezzük külön az első felét, külön a második felét, és az így kapott részeket összefésüljük. Természetesen a tömb első, illetve második felét is úgy rendezzük, hogy félbevágjuk őket, külön-külön rendezzük ezeket a részeket, majd összefésüljük őket. Addig vagdossuk ismételten félbe a tömbünket, amíg a kapott

részek egyeleműek nem lesznek, hiszen azokat önmagukban már tekinthetjük rendezett részeknek. Vagyis az összefésüléses eljárás rekurzív: minden fázisban saját magát hívja meg (kétszer is) egyre kisebb bemeneteken.

6. algoritmus: Összefésüléses rendezés

Input: egy $n \geq 1$ darab különböző számot tartalmazó T tömb
Output: növekvően rendezett T tömb

ha $n > 1$**:**
  $A \coloneqq$ **osszefesulesesrendezes**$\big(T[0 : \lfloor (n-1)/2 \rfloor]\big)$
  $B \coloneqq$ **osszefesulesesrendezes**$\big(T[\lfloor (n-1)/2 \rfloor + 1 : n-1]\big)$
  $T \coloneqq$ **osszefesul**(A, B)
return T

Az összefésüléses rendezés lépésszámát most nem elemezzük részletesen.

4.6. tétel. Az összefésüléses rendezés lépésszáma $O(n \log_2 n)$.

Az alábbi tételt, miszerint nagyságrendileg ennél nincsen gyorsabb összehasonlítás-alapú rendező algoritmus, szintén nem bizonyítjuk.

4.7. tétel. Minden összehasonlítás-alapú rendező algoritmushoz tetszőleges n esetén van olyan n-méretű tömb, amin az algoritmus legalább $\frac{1}{4} n \log_2 n$ lépést tesz.

Kulcsmanipulációs rendezések.

Vannak nem összehasonlítás-alapú rendező algoritmusok is, melyek az összehasonlításokon kívül más módon is nyernek információt a rendezendő tömb elemeiről, például előre tudják, hogy milyen elemek fordulhatnak elő a rendezendő tömb celláiban. Ezek bizonyos speciális esetekben akár gyorsabban is tudnak rendezni, mint $O(n \log_2 n)$.

Egy ilyen rendezési algoritmus például a ládarendezés.

Ládarendezés. (Hamarosan...)