5 | Rendezés

Korábban már láttunk egy rendező algoritmust, a kiválasztásos rendezést. Ebben a fejezetben további rendező algoritmusokkal fogunk megismerkedni. Főleg úgynevezett összehasonlítás-alapú rendezésekkel foglalkozunk: ezek az elemek egymással való összehasonlítása alapján rendeznek.

Beszúrásos rendezés

Tekintsük először az alábbi feladatot.

5.1. 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.

16. algoritmus: Az utolsó elem kivételével rendezett tömb rendezése

Input: egy $n \geq 1$ darab (különböző) számot tartalmazó T tömb, aminek a $T[0 : n-2]$ résztömbje már növekvően rendezett
Output: növekvően rendezett T tömb

$i \coloneqq n-1$
ciklus: amíg $i > 0$ **és** $T[i-1] > T[i]$
  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 elértük vele a tömb elejét, akkor (a $T[0 : n-2]$ tömb rendezettsége miatt) a kapott tömb már teljesen rendezett lesz. Az algoritmus biztosan véget ér, amit rögtön látni fogunk a lépésszám-elemzésből.

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 16. algoritmussal a helyére mozgatja. Nyilván az egyelemű $T[0:0]$ résztömbre tekinthetünk rendezett tömbként. Ezután meghívjuk a 16. 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.

17. 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**
  $i \coloneqq j$
  ciklus: amíg $i > 0$ **és** $T[i-1] > T[i]$
    csere$\big(T[i-1], T[i]\big)$
    $i \coloneqq i - 1$
  ciklus vége
ciklus vége
return T

Helyesség. A 16. 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 16. algoritmust, ezért az eljárás végén az egész tömb rendezett lesz. Az algoritmus biztosan véget ér, amit rögtön látni fogunk a lépésszám-elemzésből.

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 16. 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.

Vagyis ezzel beláttuk a következő állítást.

5.2. tétel.
A beszúrásos rendezés (azaz a 17. algoritmus) lépésszáma $O(n^2)$.

Nézzünk most egy példát a beszúrásos rendezésre.

5.3. példa.

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, 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 pedig az algoritmus véget ér.

Összefésüléses rendezés

Tekintsük először az alábbi feladatot.

5.4. 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.

18. algoritmus: Összefésülés eljárás – 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 – az A és a B tömbök rendezettsége miatt – az A és a B 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 algoritmus biztosan véget ér, amit rögtön látni fogunk a lépésszám-elemzésből.

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.

Nem nehéz látni, hogy az összefésül eljárás (azaz a 18. algoritmus) abban az esetben is helyesen működik, ha a két rendezett tömbben tárolt elemek nem mind különbözőek.

Az összefésül eljárást (azaz a 18. 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.

19. 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 itt nem elemezzük részletesen.

5.5. 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.

5.6. 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.