Adatszerkezetek

13. hét – Rendezések II.: összefésülő rendezés és gyorsrendezés

Dr. Koczka Ferenc · Eszterházy Károly Egyetem · 2026

Ismétlés

- **Egyszerű rendezések** (5. hét): buborék, kiválasztásos, beszúrásos – **$O(n^2)$**
- **Összefésülés** (3., 6. hét): két rendezett sorozatból egy rendezett – **$O(n + m)$**
- **Rekurzió** (12. hét): a feladat visszavezetése kisebb példányokra

A mai óra kérdései

- Hogyan lehet O(n log n) idő alatt rendezni?
- Mi az „oszd meg és uralkodj” elv?
- Miért a gyorsrendezés a gyakorlat kedvence – és mikor lassú?

Oszd meg és uralkodj

Divide and conquer – három lépés:

1. **Felosztás:** a feladatot kisebb, azonos típusú részfeladatokra bontjuk
2. **Uralkodás:** a részfeladatokat **rekurzívan** megoldjuk (a nagyon kicsiket közvetlenül)
3. **Összevonás:** a részmegoldásokból előállítjuk a teljes megoldást

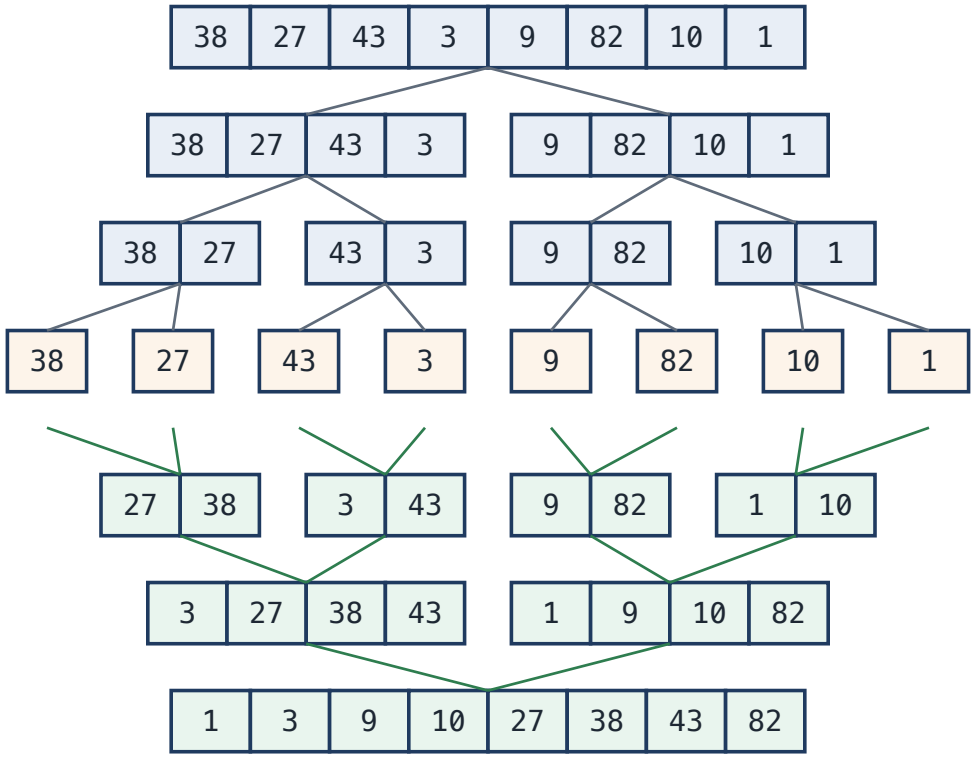
	Felosztás	Összevonás
Összefésülő rendezés	triviális (középen kettévágjuk)	**munka:** összefésülés
Gyorsrendezés	**munka:** particionálás	triviális (semmi)
Bináris keresés (2. hét)	a középső alapján	– (csak az egyik felét nézzük)

Összefésülő rendezés

Összefésülő rendezés – az ötlet

felosztás

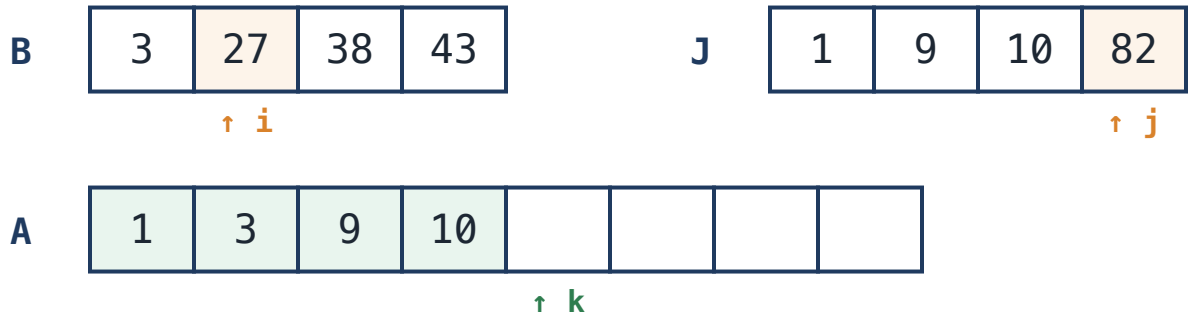
összefésülés



$\log_2 n$ szint

szintenként O(n) munka

Az összefésülés lépése



`B[i] = 27 < J[j] = 82`

`→ A[k] ← 27, i ← i + 1`

mindig a két rész elején álló
kisebbik elem kerül az eredménybe

- Két **rendezett** részből (B, J) egy rendezett tömb (A)
- Mindig a két rész **elején** álló elemek közül a kisebbiket tesszük át
- Ha az egyik rész elfogyott, a másik maradékát átmásoljuk
- Lépésszám: **O(n)**, segédtár: **O(n)**

Összefésülő rendezés – algoritmus

Pszeudokód

```
Eljárás ÖsszefésülőRendezés(A, bal, jobb)
  Ha bal < jobb akkor
    k ← (bal + jobb) div 2
    ÖsszefésülőRendezés(A, bal, k)
    ÖsszefésülőRendezés(A, k + 1, jobb)
    Összefésül(A, bal, k, jobb)
  Elágazás vége
Eljárás vége

Eljárás Összefésül(A, bal, k, jobb)
  B ← A[bal..k] ; J ← A[k+1..jobb]   // másolatok
  i ← 1 ; j ← 1 ; m ← bal
  Ciklus amíg i ≤ hossz(B) és j ≤ hossz(J)
    Ha B[i] ≤ J[j] akkor
      A[m] ← B[i] ; i ← i + 1
    különben
      A[m] ← J[j] ; j ← j + 1
    Elágazás vége
    m ← m + 1
  Ciklus vége
  B és J maradékának átmásolása A-ba
Eljárás vége
```

Java

```java
static void mergeSort(int[] a, int lo, int hi) {
    if (lo >= hi) return;
    int mid = lo + (hi - lo) / 2;
    mergeSort(a, lo, mid);
    mergeSort(a, mid + 1, hi);
    merge(a, lo, mid, hi);
}

static void merge(int[] a, int lo, int mid, int hi) {
    int[] l = Arrays.copyOfRange(a, lo, mid + 1);
    int[] r = Arrays.copyOfRange(a, mid + 1, hi + 1);
    int i = 0, j = 0, k = lo;
    while (i < l.length && j < r.length)
        a[k++] = (l[i] <= r[j]) ? l[i++] : r[j++];
    while (i < l.length) a[k++] = l[i++];
    while (j < r.length) a[k++] = r[j++];
}
```

Összefésülő rendezés – elemzés

- A felezések száma: **$\log_2 n$** szint
- Minden szinten összesen **n** elemet fésülünk össze → szintenként **O(n)**
- Összesen: **O(n log n) – minden** esetben (legjobb, átlagos, legrosszabb)

Tulajdonság	Összefésülő rendezés
Lépésszám	**O(n log n)** mindig
Segédtár	**O(n)** – nem helyben rendez
Stabil	**igen** (`≤` az összefésülésben)
Láncolt listán	jól működik (nem kell közvetlen elérés)

> $n = 10^6$ esetén: $n^2 = 10^{12}$ lépés, $n \log_2 n \approx 2 \cdot 10^7$ lépés – **50 000-szer** kevesebb!

Gyorsrendezés

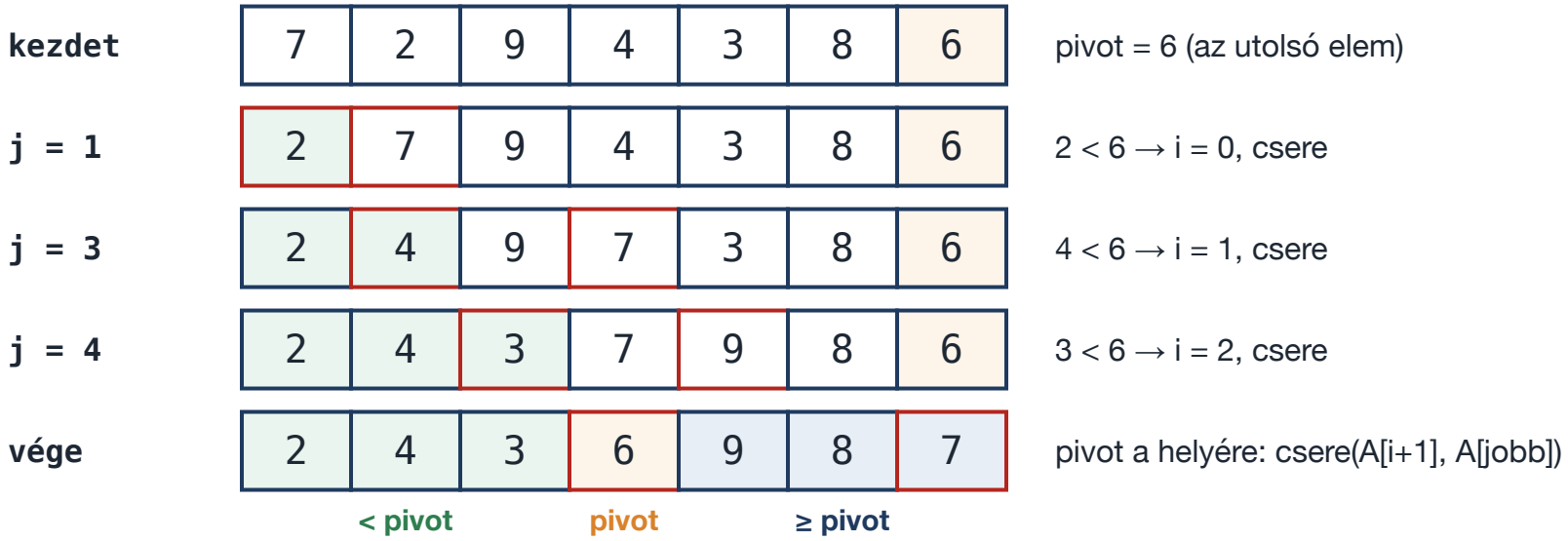
Gyorsrendezés – az ötlet

C. A. R. Hoare (1959)

1. Válasszunk egy elemet: ez a **pivot** (vezérelem)
2. **Particionálás:** rendezzük át a tömböt úgy, hogy a pivotnál **kisebbek balra**, a **nagyobbak jobbra** kerüljenek – a pivot így a **végleges helyére** kerül
3. Rendezzük **rekurzívan** a bal és a jobb részt

> Az összefésülő rendezéssel ellentétben itt a **felosztás** a munka, az összevonás „ingyen” van: ha a két rész rendezett, az egész is az.

Particionálás (Lomuto-módszer)



Gyorsrendezés – algoritmus

Pszeudokód

```
Eljárás Gyorsrendezés(A, bal, jobb)
  Ha bal < jobb akkor
    p ← Particionál(A, bal, jobb)
    Gyorsrendezés(A, bal, p - 1)
    Gyorsrendezés(A, p + 1, jobb)
  Elágazás vége
Eljárás vége

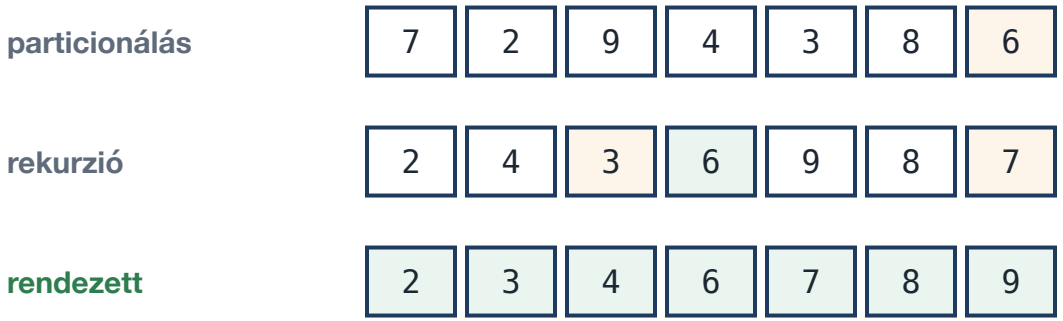
Függvény Particionál(A, bal, jobb)
  pivot ← A[jobb]
  i ← bal - 1
  Ciklus j ← bal-tól jobb - 1-ig
    Ha A[j] < pivot akkor
      i ← i + 1
      csere(A[i], A[j])
    Elágazás vége
  Ciklus vége
  csere(A[i + 1], A[jobb])
  Vissza: i + 1
Függvény vége
```

Java

```java
static void quickSort(int[] a, int lo, int hi) {
    if (lo >= hi) return;
    int p = partition(a, lo, hi);
    quickSort(a, lo, p - 1);
    quickSort(a, p + 1, hi);
}

static int partition(int[] a, int lo, int hi) {
    int pivot = a[hi];
    int i = lo - 1;
    for (int j = lo; j < hi; j++)
        if (a[j] < pivot) {
            i++;
            swap(a, i, j);
        }
    swap(a, i + 1, hi);
    return i + 1;
}
```

A pivot választásának jelentősége



Jó pivot (a középső körüli érték): a részek kb. feleződnek → log n szint → O(n log n)

Rossz pivot (mindig a min. vagy max.): n szint → $O(n^2)$

Gyorsrendezés – elemzés

Eset	Mikor?	Lépésszám
Legjobb	a pivot mindig a medián	O(n log n)
Átlagos	véletlenszerű adat	**O(n log n)** (~1,39 · n $\log_2$ n összehasonlítás)
Legrosszabb	a pivot mindig a min. vagy a max. (pl. **rendezett** tömb, utolsó elem pivot)	**O(n^2)**

Mérés (n = 5000, rendezett bemenet): fix pivot **12 497 500**, véletlen pivot **68 648** összehasonlítás

Védekezés a legrosszabb eset ellen:

- **Véletlen pivot** – a rossz eset valószínűsége elhanyagolható
- **Háromból a középső** (az első, a középső és az utolsó elem mediánja)
- A kisebbik részre rekurzió, a nagyobbikra ciklus → a veremmélység O(log n)

Gyorsrendezés – tulajdonságok

Tulajdonság	Gyorsrendezés
Lépésszám	O(n log n) átlagosan, O(n^2) legrosszabb esetben
Segédtár	**O(log n)** (a rekurzió verme) – **helyben** rendez
Stabil	**nem** (a távoli cserék miatt)
A gyakorlatban	általában a **leggyorsabb** – kevés adatmozgatás, gyorsítótár-barát

A Java `Arrays.sort` primitív tömbökre **Dual-Pivot Quicksort**-ot, a Go `slices.Sort` **pdqsort**-ot (a gyorsrendezés továbbfejlesztett változatát) használ.

Összehasonlítás

A rendezések összehasonlítása

	Legrosszabb	Átlagos	Legjobb	Segédtár	Stabil
Buborék	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	igen
Kiválasztásos	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	nem
Beszúrásos	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	igen
Összefésülő	**$O(n \log n)$**	$O(n \log n)$	$O(n \log n)$	$O(n)$	**igen**
Gyors	$O(n^2)$	**$O(n \log n)$**	$O(n \log n)$	$O(\log n)$	nem
Kupacrendezés (15. hét)	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	nem
Számláló (5. hét)	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	igen

Alsó korlát: csak összehasonlításokra épülő rendezés legrosszabb esetben **nem lehet gyorsabb $\Omega(n \log n)$-nél**. A számláló rendezés azért gyorsabb, mert nem hasonlít, hanem számol (csak kis értéktartományra).

Mérési eredmények (véletlen adat)

n	beszúrásos	összefésülő	gyors	beépített
1 000	0,8 ms	0,2 ms	0,2 ms	0,9 ms
10 000	13,3 ms	0,7 ms	0,6 ms	1,5 ms
100 000	**1 159 ms**	16,5 ms	4,4 ms	5,7 ms
1 000 000	(~2 perc)	65,6 ms	49,8 ms	43,6 ms

Java, `SortBenchmark` program. A beépített rendezés kis **n**-nél a JVM bemelegedése miatt tűnik lassúnak.

Külső rendezés

Mi van, ha az adat **nem fér el a memóriában** (pl. 100 GB-os fájl, 8 GB RAM)?

Külső összefésülő rendezés:

1. Olvassunk be annyit, amennyi befér (pl. 1 GB), rendezzük a memóriában, és írjuk ki egy ideiglenes fájlba → 100 rendezett „futam”
2. **Fésüljük össze** a futamokat: mindegyikből csak az elejét tartjuk a memóriában, és mindig a legkisebbet írjuk ki (**k**-utas összefésülés – prioritási sorral)

- A lassú lemezműveletek száma minimális: minden adatot **kétszer** olvasunk és írunk
- Adatbázis-kezelők, nagy adatfeldolgozó rendszerek így rendeznek

Összefoglalás

- **Oszd meg és uralkodj:** felosztás → rekurzív megoldás → összevonás
- **Összefésülő rendezés:** felezés + összefésülés · mindig **O(n log n)** · stabil · O(n) segédtár
- **Gyorsrendezés:** pivot + particionálás · átlagosan **O(n log n)**, legrosszabb $O(n^2)$ · helyben · nem stabil
- A pivot választása kritikus – **véletlen pivot** vagy háromból a középső
- Összehasonlító rendezés nem lehet gyorsabb **O(n log n)**-nél
- **Külső rendezés:** futamok + **k**-utas összefésülés

Következő hét: fák – fogalmak, bináris fák, ábrázolás, bejárások.

Ellenőrző kérdések

1. Rajzold le a `[5, 1, 4, 2, 8, 3]` tömb összefésülő rendezésének felosztási és összefésülési fáját!
2. Particionáld a `[4, 8, 1, 6, 3, 7, 5]` tömböt a Lomuto-módszerrel (pivot = 5)! Hová kerül a pivot?
3. Igaz vagy hamis: a gyorsrendezés legrosszabb esetben is O(n log n) lépésben rendez.
4. Melyik rendezést választanád, ha a stabilitás fontos, és garantált O(n log n) kell?
5. Miért lassú a gyorsrendezés rendezett tömbön, ha mindig az utolsó elem a pivot?
6. Hány szintje van az összefésülő rendezés rekurziós fájának n = 1024 esetén?

Gyakorlat

✎ Bemutató programok

Program	Java	Go	Mit mutat?
Összefésülő	`MergeSort.java`	`h13/mergesort`	az összefésülések sorrendje, rekurziós mélység
Gyorsrendezés	`QuickSort.java`	`h13/quicksort`	particionálás, rossz és véletlen pivot
Mérés	`SortBenchmark.java`	`h13/benchmark`	$O(n^2)$ vs $O(n \log n)$ vs beépített

Futtasd a `SortBenchmark` programot! Hányszorosára nő az idő, ha **n** tízszeresére nő – a beszúrásos és az összefésülő rendezésnél?

✎ Feladatok

1. **Csökkenő sorrend:** módosítsd az összefésülő rendezést csökkenő sorrendre!
2. **Stabil rendezés:** rendezz szavakat a hosszuk szerint úgy, hogy az azonos hosszúak sorrendje megmaradjon!
3. **Inverziók O(n log n)-ben:** számold meg az inverziókat összefésülés közben! (vesd össze az 5. heti $O(n^2)$-es megoldással)
4. **Quickselect:** keresd meg a **k**-adik legkisebb elemet rendezés nélkül, a particionálással – átlagosan O(n)!
5. **Holland zászló:** rendezz egy csak 0, 1 és 2 értékeket tartalmazó tömböt egyetlen bejárással!
6. **Medián:** két rendezett tömb közös mediánja összefésüléssel.

Megoldások: `Exercises13.java`, `h13/exercises`

✎ Házi feladat

1. Valósítsd meg a gyorsrendezést **háromból a középső** pivotválasztással! Mérd meg rendezett és fordított bemeneten!
2. **Hibrid rendezés:** ha a rész 16 elemnél kisebb, használj beszúrásos rendezést! Mennyit gyorsít ez a mérések szerint?
3. Írd meg az összefésülő rendezést **láncolt listára** (segédtömb nélkül)! (Tipp: a lista kettévágása a 9. heti nyúl–teknős módszerrel.)
4. Valósítsd meg a **Hoare-féle** particionálást (két index halad egymással szemben)! Hasonlítsd össze a cserék számát a Lomuto-módszerrel!