7. előadás

Rendező algoritmusok

Gyorsrendezés, rendezés lineáris lépésszámmal

Adatszerkezetek és algoritmusok előadás

2018. március 6.

Kósa Márk, Pánovics János,
Szathmáry László és Halász Gábor

Debreceni Egyetem
Informatikai Kar

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Általános tudnivalók

Ajánlott irodalom:

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:
  Új algoritmusok, Scolar Informatika, 2003.
- Donald E. Knuth: A számítógépprogramozás művészete 1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- Donald E. Knuth: A számítógépprogramozás művészete 3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- Seymour Lipschutz: Adatszerkezetek, Panem-McGraw-Hill, Budapest, 1993.
- Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok, Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- Gyakorlati aláírás
  - 2 ZH
- Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

`http://hallg.inf.unideb.hu/~halasz`

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A gyorsrendezés leírása

A gyorsrendezés, az összefésülő rendezéshez hasonlóan az oszd- meg-és-uralkodj elven alapszik.

Felosztás: Az A[p . . r] tömböt két (esetleg üres) A[p . . q – 1] és A[q + 1 . . r] résztömbre osztjuk úgy, hogy az A[p . . q–1] minden eleme kisebb vagy egyenlő A[q]-nál, ez utóbbi elem viszont kisebb vagy egyenlő A[q + 1 . . r] minden eleménél. A q index kiszámítása része ennek a felosztó eljárásnak.

Uralkodás: Az A[p . . q–1] és A[q+1 . . r] résztömböket a gyorsrendezés rekurzív hívásával rendezzük.

Összevonás: Mivel a két résztömböt helyben rendeztük, nincs szükség egyesítésre: az egész A[p . . r] tömb rendezett.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A gyorsrendezés egy algoritmusa

```
procedure GYORS_RENDEZ(A, p, r)
 1: if  p < r  then
 2:       q ← FELOSZT(A, p, r)
 3:       GYORS_RENDEZ(A, p, q − 1)
 4:       GYORS_RENDEZ(A, q + 1, r)
 5: end if
end procedure
```

```
function FELOSZT(A, p, r)
 1: x ← A[r]
 2: i ← p − 1
 3: for  j ← p  to  r − 1  do
 4:       if  A[j] ≤ x  then
 5:             i ← i + 1
 6:             A[i]  és  A[j] felcserélése
 7:       end if
 8: end for
 9: A[i + 1]  és  A[r] felcserélése
10: return  i + 1
end function
```

A 3–6. sorokban levő ciklus minden iterációjának kezdetén a k tömbindexre fennáll:

1. Ha $p \le k \le i$, akkor $A[k] \le x$.
2. Ha $i < k < j$, akkor $A[k] > x$.
3. Ha $k = r$, akkor $A[k] = x$.
4. Ha $j \le k < r$, akkor $A[k]$???

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A gyorsrendezés egy algoritmusa

```
procedure GYORS_RENDEZ(A, p, r)
 1: if p < r then
 2:     q ← FELOSZT(A, p, r)
 3:     GYORS_RENDEZ(A, p, q – 1)
 4:     GYORS_RENDEZ(A, q + 1, r)
 5: end if
end procedure
function FELOSZT(A, p, r)
 1: x ← A[r]
 2: i ← p – 1
 3: for j ← p to r – 1 do
 4:     if A[j] ≤ x then
 5:         i ← i + 1
 6:         A[i] és A[j] felcserélése
 7:     end if
 8: end for
 9: A[i + 1] és A[r] felcserélése
10: return i + 1
end function
```

A 3–6. sorokban levő ciklus minden iterációjának kezdetén a k tömbindexre fennáll:

1. Ha $p \leq k \leq i$, akkor $A[k] \leq x$.
2. Ha $i < k < j$, akkor $A[k] > x$.
3. Ha $k = r$, akkor $A[k] = x$.
4. Ha $j \leq k < r$, akkor $A[k]$???

A gyorsrendezés egy algoritmusa

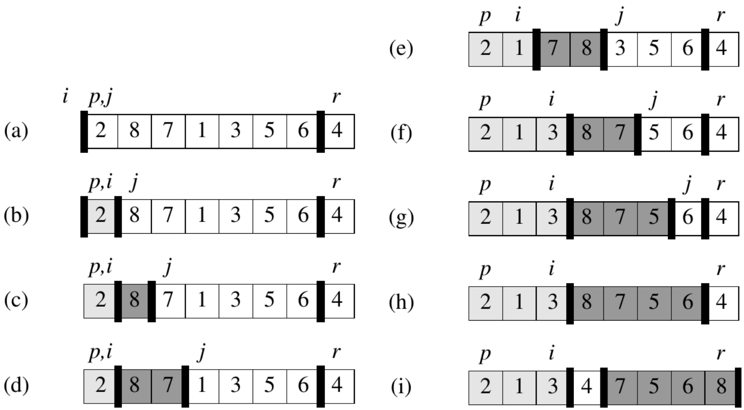
```
procedure GYORS_RENDEZ(A, p, r)
   if  p < r  then
        q ← FELOSZT(A, p, r)
        GYORS_RENDEZ(A, p, q − 1)
        GYORS_RENDEZ(A, q + 1, r)
   end if
end procedure
```

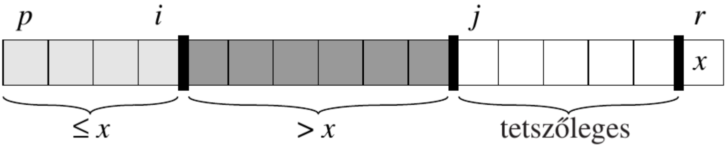
```
function FELOSZT(A, p, r)
 1: x ← A[r]
 2: i ← p − 1
 3: for  j ← p  to  r − 1  do
 4:      if  A[j] ≤ x  then
 5:           i ← i + 1
 6:           A[i]  és  A[j] felcserélése
 7:      end if
 8: end for
 9: A[i + 1]  és  A[r] felcserélése
10: return  i + 1
end function
```

A 3–6. sorokban levő ciklus minden iterációjának kezdetén a k tömbindexre fennáll:

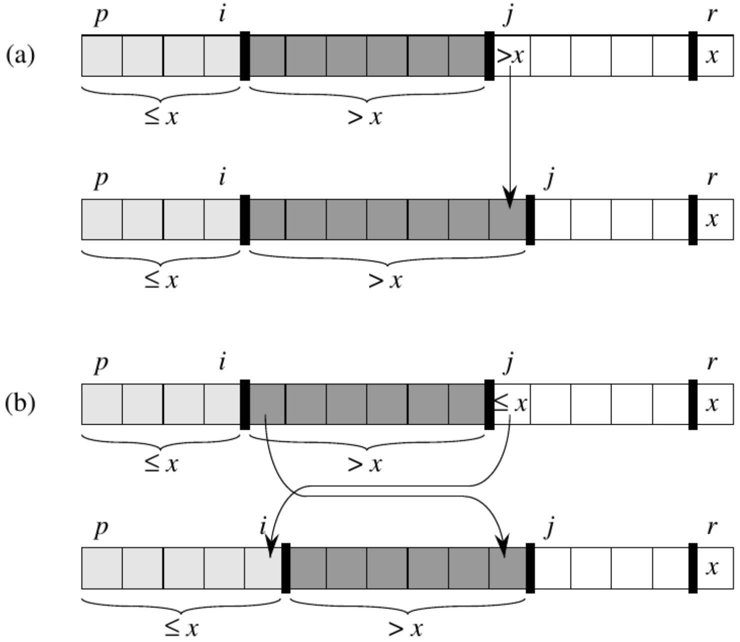
1. Ha $p \leq k \leq i$,
   akkor $A[k] \leq x$.
2. Ha $i < k < j$,
   akkor $A[k] > x$.
3. Ha $k = r$,
   akkor $A[k] = x$.
4. Ha $j \leq k < r$,
   akkor $A[k]$???

A FELOSZT algoritmus működése





A FELOSZT algoritmus működése



Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Elemzés

Teljesül: A ciklus első iterációja előtt $i = p - 1$ és $j = p$. Mivel egyetlen elem sincs p és i között, sem pedig $i + 1$ és $j - 1$ között, a ciklusinvariáns első két feltétele triviálisan igaz. Az 1. sor értékadása miatt a harmadik feltétel is igaz.

Megmarad: Amikor $A[j] > x$, akkor csupán a j értékét kell növelni. Amikor j értéke megnő, a második feltétel $A[j-1]$-re igaz, míg a többi elem változatlan marad.
Amikor $A[j] \leq x$, akkor i megnő, $A[i]$ és $A[j]$ felcserélődik, majd j megnő. A csere miatt most $A[i] \leq x$, és az 1. feltétel teljesül. Mivel a ciklusinvariáns miatt az az elem, amely az $A[j-1]$-be került nagyobb, mint x: $A[j-1] > x$.

Befejeződik: Befejezéskor $j = r$, ezért a tömb minden eleme az invariáns által leírt valamelyik halmazban van. A tömb elemeit három halmazba tettük: az elsőben x-nél kisebbek vagy vele egyenlők vannak, a másodikban x-nél nagyobbak, és a harmadikban csak az x.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A gyorsrendezés hatékonysága
Legrosszabb felosztás

A gyorsrendezés legrosszabb esete az, amikor a felosztó eljárás az eredeti tömböt egy $n - 1$ és egy 0 elemű tömbre osztja.

A felosztási idő $\Theta(n)$. A rekurzív hívás egy 0 nagyságú tömbre nem csinál egyebet, éppen csak visszatér, ezért $T(0) = \Theta(1)$, és a gyorsrendezés futási idejének rekurzív képlete:

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n). \end{aligned}$$

Intuitív módon, egy számtani sorhoz jutunk, aminek az összege alapján azt várhatjuk: $T(n) = \Theta(n^2)$.

Ezt pl. a helyettesítő módszerrel könnyű ellenőrizni.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A gyorsrendezés hatékonysága
Legjobb felosztás

A legjobb felosztásnál a FELOSZT eljárás két, *n/2* eleműnél nem nagyobb tömböt hoz létre, mivel az egyik $\lfloor n/2 \rfloor$, a másik pedig $\lceil n/2 \rceil - 1$ elemű.

A futási idő rekurzív képlete

$$T(n) \leq 2T\left(\frac{n}{2}\right) + \Theta(n),$$

amelynek megoldása a mester tétel második esete alapján

$$T(n) = \Theta(n lg n),$$

így ez a legjobb felosztás aszimptotikusan gyorsabb algoritmust eredményez.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Mester módszer

Theorem (Mester tétel)

Legyenek $a \geq 1$, $b > 1$ *állandók*, $f(n)$ *egy függvény*, $T(n)$ *pedig a nemnegatív egészeken a*

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

rekurzív egyenlettel definiált függvény, ahol n/b *jelentheti akár az* $\lfloor n/b \rfloor$ *egészrészt, akár az* $\lceil n/b \rceil$ *egészrészt. Ekkor:*

1. *Ha* $f(n) = O\left(n^{\log_b a - \epsilon}\right)$*, egy* $\epsilon > 0$ *állandóval, akkor*
   $\longrightarrow T(n) = \Theta\left(n^{\log_b a}\right)$.
2. *Ha* $f(n) = \Theta\left(n^{\log_b a}\right)$*, akkor*
   $\longrightarrow T(n) = \Theta\left(n^{\log_b a} \lg n\right) (= \Theta\left(f(n) \lg n\right))$.
3. *Ha* $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ *és* $a \cdot f(n/b) \leq c \cdot f(n)$ *valamely* $\epsilon > 0$ *és* $c < 1$ *állandóra és elég nagy* n*-re, akkor*
   $\longrightarrow T(n) = \Theta\left(f(n)\right)$.

A gyorsrendezés hatékonysága

Kiegyensúlyozott felosztás

Tételezzük fel, hogy a felosztó eljárás mindig 9 az 1-hez felosztást ad, amely kiegyensúlyozatlan felosztásnak tűnik.

Ekkor a gyorsrendezés futási idejének rekurzív képlete

$$T(n) \leq T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + cn,$$

ahol expliciten kiírtuk a $\Theta(n)$-ben rejlő c állandót. A rekurziós fa pedig:

A gyorsrendezés hatékonysága

Kiegyensúlyozott felosztás

Tételezzük fel, hogy a felosztó eljárás mindig 9 az 1-hez felosztást ad, amely kiegyensúlyozatlan felosztásnak tűnik.

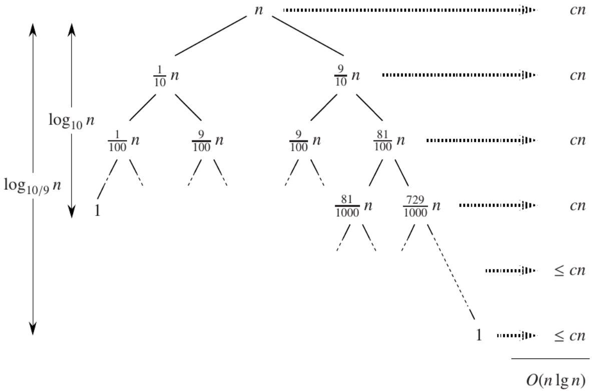
Ekkor a gyorsrendezés futási idejének rekurzív képlete

$$T(n) \leq T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + cn,$$

ahol expliciten kiírtuk a $\Theta(n)$-ben rejlő c állandót. A rekurziós fa pedig:

Minden állandó arányú felosztáskor a rekurziós fa mélysége $\Theta(\lg n)$, és minden szinten a költség $O(n)$.

A futási idő tehát $O(n \lg n)$.



A gyorsrendezés hatékonysága

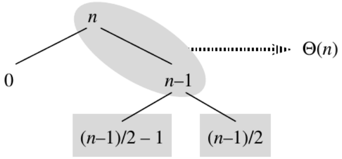
Az átlagos viselkedés megsejtése

- Ahhoz, hogy a gyorsrendezés átlagos viselkedését pontosan meghatározhassuk, egy feltevést kell megfogalmaznunk a bemenő tömbök gyakoriságáról.
- A gyorsrendezés viselkedése nem a bemenő elemektől, hanem azoknak az egymáshoz viszonyított helyétől függ.
- A legnyilvánvalóbb feltételezés az, hogy a bemeneti elemek minden permutációja ugyanolyan eséllyel fordulhat elő.
- Amikor a gyorsrendezés véletlen bemenetekre fut, nem valószínű, hogy a felosztás minden szinten egyformán történik.
  - Várható, hogy bizonyos felosztások kiegyensúlyozottak lesznek, mások pedig nem.
  - Például, bizonyítható, hogy a FELOSZT eljárás
    - 80% körül 9:1 aránynál kiegyensúlyozottabb, míg
    - 20% körül legfeljebb ennyire kiegyensúlyozott

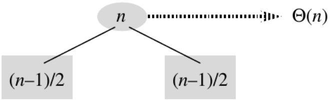
    felosztást ad.
  - Általában a FELOSZT eljárás a „jó” és „rossz” felosztások keverékét adja.
  - Egy rossz és utána egy jó felosztás három résztömböt hoz létre $\Rightarrow T(n) \leq T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \mathbf{2cn} = \Theta(n \lg n)$.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A gyorsrendezés hatékonysága



(a)



(b)

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A gyorsrendezés véletlenített változata

A „véletlenítés” egyik módszere lehetne, ha permutálnánk a bemenetet.

Egy másik változat, a véletlen mintavétel nevezetű, egyszerűbb elemzést biztosít. (És gyorsabban kivitelezhető.)

Őrszemnek nem mindig az *A[r]* elemet tekintjük, hanem helyette az *A[p . . r]* résztömb egy véletlenszerűen kiválasztott elemét. Ezt úgy oldjuk meg egyszerűen, hogy az *A[r]* elemet felcseréljük az *A[p . . r]* résztömb egy véletlenszerűen választott elemével. Ez a módosítás biztosítja, hogy az $x = A[r]$ őrszem ugyanolyan valószínűséggel lehet az *A[p . . r]* résztömb bármelyik eleme. Mivel az őrszemet véletlenszerűen választjuk ki, az átlagos viselkedésben a felosztás várhatóan jól kiegyensúlyozott lesz.

```
function VÉLETLEN_FELOSZT(A, p, r)
    i ← VÉLETLEN(p, r)
    A[r] és A[i] cseréje
    return FELOSZT(A,p,r)
end function
```


Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A gyorsrendezés véletlenített változata

A „véletlenítés” egyik módszere lehetne, ha permutálnánk a bemenetet.

Egy másik változat, a véletlen mintavétel nevezetű, egyszerűbb elemzést biztosít. (És gyorsabban kivitelezhető.)

Őrszemnek nem mindig az *A[r]* elemet tekintjük, hanem helyette az *A[p . . r]* résztömb egy véletlenszerűen kiválasztott elemét. Ezt úgy oldjuk meg egyszerűen, hogy az *A[r]* elemet felcseréljük az *A[p . . r]* résztömb egy véletlenszerűen választott elemével. Ez a módosítás biztosítja, hogy az $x = A[r]$ őrszem ugyanolyan valószínűséggel lehet az *A[p . . r]* résztömb bármelyik eleme. Mivel az őrszemet véletlenszerűen választjuk ki, az átlagos viselkedésben a felosztás várhatóan jól kiegyensúlyozott lesz.

```
function VÉLETLEN_FELOSZT(A, p, r)
    i ← VÉLETLEN(p, r)
    A[r] és A[i] cseréje
    return FELOSZT(A,p,r)
end function
```


Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A „várható” futási idő

Lemma (Tankönyvben: 7.1. lemma)

Legyen X a GYORSRENDEZÉS *egész futási ideje alatt a* FELOSZT *4. sorában végrehajtott összehasonlítások száma, ha **n** elemű tömböt rendezünk.*
Ekkor a GYORSRENDEZÉS *futási ideje* $O(n + X)$.

Bizonyítás.

A FELOSZT eljárást legfeljebb ***n***-szer hívjuk meg, ennek futási ideje egy állandó, amelyhez hozzáadódik a **for** ciklus végrehajtásából adódó idő. A **for** minden iterációjakor végrehajtjuk a 4. sort is. □

Célunk, hogy kiszámítsuk X-et, az összehasonlítások számát a FELOSZT összes hívásában. Nem fogjuk megszámolni, hogy a FELOSZT egy-egy híváskor hány összehasonlítást végez, hanem inkább egy felső korlátot adunk az összehasonlítások összértékére.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A „várható” futási idő

Lemma (Tankönyvben: 7.1. lemma)

Legyen X a GYORSRENDEZÉS *egész futási ideje alatt a* FELOSZT *4. sorában végrehajtott összehasonlítások száma, ha **n** elemű tömböt rendezünk.*
Ekkor a GYORSRENDEZÉS *futási ideje O(n + X).*

Bizonyítás.

A FELOSZT eljárást legfeljebb ***n***-szer hívjuk meg, ennek futási ideje egy állandó, amelyhez hozzáadódik a **for** ciklus végrehajtásából adódó idő. A **for** minden iterációjakor végrehajtjuk a 4. sort is. □

Célunk, hogy kiszámítsuk X-et, az összehasonlítások számát a FELOSZT összes hívásában. Nem fogjuk megszámolni, hogy a FELOSZT egy-egy híváskor hány összehasonlítást végez, hanem inkább egy felső korlátot adunk az összehasonlítások összértékére.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A „várható” futási idő

Az elemzés megkönnyítéséért nevezzük át az A tömb elemeit, legyenek ezek z_1 , z_2 , . . . , z_n, ahol z_i az i-edik legkisebb elem. Ugyancsak értelmezzük a $Z_{ij} = \{z_i, z_{i+1}, ..., z_j\}$ halmazt, amely a z_i és z_j közötti elemeket tartalmazza, ezekkel bezárólag. (És tegyük fel, hogy minden elem eltérő értékű.)

Legyen

$$X_{ij} = I\{z_i \text{ összehasonlítása } z_j\text{-vel}\},$$
$$\left(= \begin{cases} 1 & \text{ha } z_i \text{ és } z_j \text{ össze lesznek hasonlítva valamikor,} \\ 0 & \text{egyébként (I=indikátor)} \end{cases} \right)$$

Mikor hasonlítja össze az algoritmus a z_i és z_j elemeket? Hogy megválaszolhassuk a kérdést, vegyük észre, hogy bármely két számpárt legfeljebb egyszer hasonlítjuk össze. Miért?

Az elemeket csak az őrszemmel hasonlítjuk össze, és a FELOSZT egy hívása után az abban használt őrszemet többé már nem használjuk összehasonlításra.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

A „várható” futási idő

Az elemzés megkönnyítéséért nevezzük át az A tömb elemeit, legyenek ezek z_1 , z_2 , . . . , z_n, ahol z_i az i-edik legkisebb elem. Ugyancsak értelmezzük a $Z_{ij} = \{z_i, z_{i+1}, ..., z_j\}$ halmazt, amely a z_i és z_j közötti elemeket tartalmazza, ezekkel bezárólag. (És tegyük fel, hogy minden elem eltérő értékű.)

Legyen

$$X_{ij} = I\{z_i \text{ összehasonlítása } z_j\text{-vel}\},$$
$$\left(= \begin{cases} 1 & \text{ha } z_i \text{ és } z_j \text{ össze lesznek hasonlítva valamikor,} \\ 0 & \text{egyébként (I=indikátor)} \end{cases} \right)$$

Mikor hasonlítja össze az algoritmus a z_i és z_j elemeket? Hogy megválaszolhassuk a kérdést, vegyük észre, hogy bármely két számpárt legfeljebb egyszer hasonlítjuk össze. Miért?

Az elemeket csak az őrszemmel hasonlítjuk össze, és a FELOSZT egy hívása után az abban használt őrszemet többé már nem használjuk összehasonlításra.

A „várható” futási idő

Innen az összehasonlítások száma:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}.$$

Ennek várható értéke:

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr\{z_i \text{ összehasonlítása } z_j\text{-vel}\}$$

Ha az x őrszemet úgy választjuk meg, hogy $z_i < x < z_j$, akkor z_i és z_j sem most, sem később nem lesznek összehasonlítva.

Ellenben, ha $x = z_i$ vagy $x = z_j$, akkor most lesznek összehasonlítva.

A „várható” futási idő

Ennek alapján:

$$\begin{aligned}
Pr\{z_i \text{ összehasonlítása } z_j\text{-vel}\} &= Pr\{z_i \text{ vagy } z_j \text{ az első őrszem } Z_{ij}\text{-ből}\} \\
&= Pr\{z_i \text{ az első őrszem } Z_{ij}\text{-ből}\} \\
&+ Pr\{z_j \text{ az első őrszem } Z_{ij}\text{-ből}\} \\
&= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}.
\end{aligned}$$

Innen:

$$\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) = O(n \lg n)
\end{aligned}$$

$$\left(\sum_{k=1}^{n} \frac{1}{k} \leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i+j} \leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i} = \sum_{i=0}^{\lfloor \lg n \rfloor} 1 \leq \lg n + 1 \right)$$

Összehasonlító rendezések

Az eddig tárgyalt rendező algoritmusoknak van egy érdekes, közös tulajdonságuk:

- a rendezéshez csak a bemeneti elemek összehasonlítását használják.
  (Az a_i és a_j elemek esetén az $a_i < a_j$, $a_i \leq a_j$, $a_i = a_j$, $a_i \geq a_j$ vagy $a_i > a_j$ egyikét végezzük el, hogy megtudjuk a két elem egymáshoz viszonyított sorrendjét.)

Éppen ezért ezeket az algoritmusokat összehasonlító rendezéseknek nevezzük.

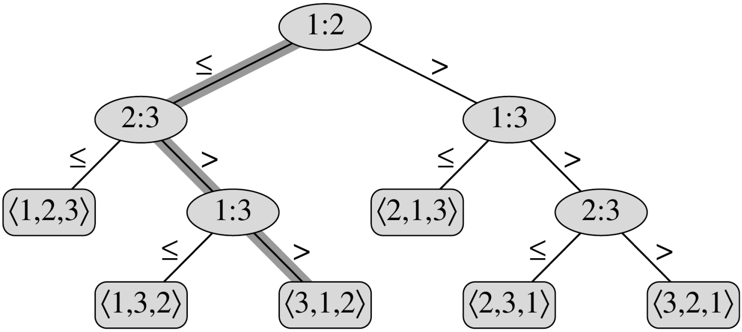
A következőkben megmutatjuk, hogy bármely összehasonlító algoritmusnak a legrosszabb esetben $\Omega(n \lg n)$ összehasonlításra van szüksége n elem rendezéséhez.

- Következésképpen az összefésüléses rendezés és a kupacrendezés aszimptotikusan optimális, és
- minden összehasonlító rendezés legfeljebb egy állandó szorzóval lehet gyorsabb.

(Továbbiakban feltételezzük, hogy minden elem eltérő értékű: $a_i \neq a_j$, így csak $a_i \leq a_j$-t fogunk használunk.)

A döntésifa-modell

Az összehasonlító rendezéseket tekinthetjük döntési fáknak:



A döntési fában:

- Minden belső csúcsot egy $i : j$ számpárral jelölünk, ahol $1 \leq i, j \leq n$, n pedig a bemenet elemeinek a száma.
- Minden levél egy $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ permutációval jelölhető.

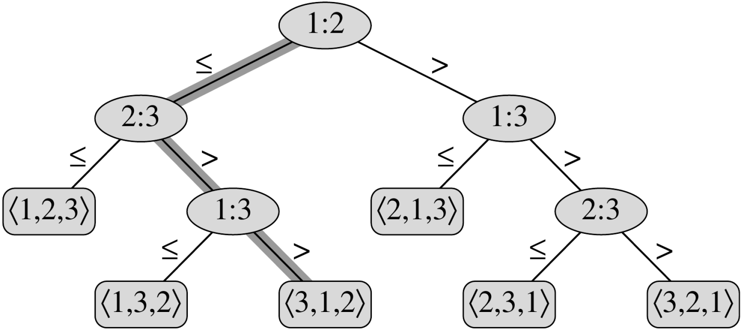
A rendezési algoritmus végrehajtása megfelel a döntési fában a gyökértől valamely levélig vezető út bejárásának.

A döntésifa-modell

Mivel minden helyes rendezési algoritmusnak elő kell tudni állítania a bemeneti elemek összes permutációját, ezért a helyes rendezés szükséges feltétele, hogy

- az *n* elem összes *n!* permutációjának meg kell jelennie a döntési fa levelei között,
- és ezen levelek mindegyikének elérhetőnek kell lennie a gyökérből egy összehasonlító rendezéshez tartozó úttal.

Vagyis csak olyan döntési fákat tekintünk, ahol minden permutáció megjelenik egy elérhető levélként.



Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Összehasonlító rendezések

Theorem (Tankönyv: 8.1. tétel.)

Bármely összehasonlító rendezőalgoritmus a legrosszabb esetben $\Omega(n \lg n)$ összehasonlítást végez.

Bizonyítás.

Vegyünk egy n elemet rendező döntési fát, amelynek magassága h, elérhető leveleinek száma pedig l. Mivel a bemenet összes $n!$ permutációja meg kell jelenjen levélként, ezért $n! \leq l$. Mivel egy h mélységű bináris fa leveleinek a száma nem lehet nagyobb, mint 2^h, ezért

$$n! \leq l \leq 2^h,$$

ahonnan mindkét oldal logaritmusát véve

$$\begin{aligned} h &\geq \lg(n!) && \text{(a logaritmusfüggvény monoton növekvő)} \\ &= \Omega(n \lg n) && \text{(Stirling-formula alapján).} \end{aligned}$$

$\square$

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Rendezés lineáris időben

A következőkben tárgyalandó algoritmusok nem az összehasonlítást használják a rendezéshez,

- így az $\Omega(n \lg n)$ alsó korlát rájuk nem vonatkozik.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Leszámláló rendezés

A leszámláló rendezésnél **feltételezzük**, hogy az n bemeneti elem mindegyike *0* és k közötti egész szám, ahol k egy ismert egész.

- Ha $k = O(n)$, a rendezés futási ideje $\Theta(n)$.

A leszámláló rendezés alapötlete az, hogy minden egyes x bemeneti elemre meghatározza azoknak az elemeknek a számát, amelyek kisebbek, mint az x.

Ezzel az információval az x elemet közvetlenül a saját pozíciójába tudjuk helyezni a kimeneti tömbben. Ha például *17* olyan elem van, amelyik kisebb, mint az x, akkor az x elem a *18*. helyen lesz a kimeneti tömbben.

Ez a séma valamelyest megváltozik abban az esetben, amikor néhány elem egyenlő, hiszen nem akarjuk mindet ugyanabba a pozícióba helyezni.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Leszámláló rendezés

```
– – A[1..n], B[1..n], C[0..k]
procedure LESZÁMLÁLÓ_RENDEZÉS(A, B, k)
  for i ← 0 to k do
      C[i] ← 0
  end for
  for j ← 1 to méret(A) do
      C[A[j]] ← C[A[j]]+1
  end for
  for i ← 1 to k do
      C[i] ← C[i]+C[i-1]
  end for
  for j ← méret(A) downto 1 do
      B[C[A[j]]] ← A[j]
      C[A[j]] ← C[A[j]]-1
  end for
end procedure
```

Könnyű belátni, hogy:

$T(n) = \Theta(n+k)$

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Leszámláló rendezés

```
– – A[1..n], B[1..n], C[0..k]
procedure LESZÁMLÁLÓ_RENDEZÉS(A, B, k)
  for i ← 0 to k do
      C[i] ← 0
  end for
  for j ← 1 to méret(A) do
      C[A[j]] ← C[A[j]]+1
  end for
  for i ← 1 to k do
      C[i] ← C[i]+C[i-1]
  end for
  for j ← méret(A) downto 1 do
      B[C[A[j]]] ← A[j]
      C[A[j]] ← C[A[j]]-1
  end for
end procedure
```

Könnyű belátni, hogy:

$$T(n) = \Theta(n+k)$$

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Leszámláló rendezés

A leszámláló rendezés egy fontos tulajdonsága az, hogy stabil: az azonos értékű elemek ugyanabban a sorrendben jelennek meg a kimeneti tömbben, mint ahogyan a bemeneti tömbben szerepeltek.

Azaz két azonos értékű szám között a kapcsolat megmarad azon szabály szerint, hogy amelyikük először jelenik meg a bemeneti tömbben, az jelenik meg először a kimeneti tömbben is.

- Általában a stabilitás csak akkor fontos tulajdonság, amikor a rendezendő elemek mellett kísérő adatok is vannak.
- A leszámláló rendezés stabilitása egy másik szempontból is fontos:
  - a leszámláló rendezést gyakran felhasználjuk a számjegyes rendezés eljárásaként.
  - Ahogyan ez a következő alfejezetben kiderül, a leszámláló rendezés stabilitása életbevágó a számjegyes rendezés helyes működéséhez.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Számjegyes rendezés

A számjegyes rendezés – másképpen: radix rendezés – kódja értelemszerű.

A következő eljárás feltételezi, hogy az n elemű A tömb minden egyes eleme d jegyű, ahol az első számjegy a legalacsonyabb helyértékű számjegy és a d-edik számjegy a legmagasabb helyértékű számjegy.

procedure SZÁMJEGYES_RENDEZÉS(A, d)

1: **for** i ← 1 **to** d **do**

2: stabil algoritmussal rendezzük az A tömböt az i-edik számjegy szerint.

3: **end for**

end procedure

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Számjegyes rendezés

Lemma (Tankönyv: 8.3. lemma.)

Legyen adott n darab d jegyből álló szám, ahol a számjegyek legfeljebb k értéket vehetnek fel. Ekkor a SZÁMJEGYES-RENDEZÉS $\Theta(d \cdot (n + k))$ *időben rendezi helyesen ezeket a számokat.*

Bizonyítás.

A számjegyes rendezés helyessége a rendezendő oszlopon végzett indukcióból következik. Ha minden számjegy 0 és $k - 1$ között van (azaz k lehetséges értéket vehet fel), és k nem túl nagy, a leszámláló rendezés a kézenfekvő választás a használandó stabil algoritmusra. Így n darab d számjegyű szám esetén egy lépés $\Theta(n + k)$ időt igényel. Mivel d lépésünk van, a számjegyes rendezés teljes időigénye $\Theta(d \cdot (n + k))$. □

Ha d állandó és $k = O(n)$, a számjegyes rendezés lineáris idejű.

Általánosabb esetben, a kulcsokat bizonyos rugalmassággal bonthatjuk számjegyekre.

Számjegyes rendezés

Lemma (Tankönyvben: 8.4. lemma.)

Legyen adott n darab b bites szám és egy tetszőleges $1 \leq r \leq b$ pozitív egész. Ekkor a SZÁMJEGYES-RENDEZÉS *$\Theta((b/r) \cdot (n + 2^r))$ időben rendezi helyesen ezeket a számokat.*

Bizonyítás.

Egy $r \leq b$ értékre minden kulcs $d = \lceil b/r \rceil$ darab egyenként r bites számjegyként tekinthető. Minden számjegy egy 0 és $2^r - 1$ közötti egész, ezért alkalmazhatjuk a leszámláló rendezést a $k = 2^r - 1$ paraméterrel.

A leszámláló rendezés minden lépése $\Theta(n + k) = \Theta(n + 2^r)$ ideig tart, így a d lépés megtételéhez szükséges teljes idő $\Theta(d \cdot (n + 2^r)) = \Theta((b/r) \cdot (n + 2^r))$. □

(Egy 32 bites szót tekinthetünk például 4 darab 8 bites számjegynek, azaz $b = 32$, $r = 8$, $k = 2^r - 1 = 255$ és $d = b/r = 4$ és így $(b/r) \cdot (n + 2^r) = 4 \cdot (n + 256) = 4n + 1024$.)

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Edényrendezés

Az edényrendezés – szokták vödör rendezésnek is hívni – várható futási ideje lineáris, amennyiben a bemenet egyenletes eloszlásból származik.

Éppúgy, mint a leszámláló rendezés, az edényrendezés is azért lehet gyors, mert feltesz valamit a bemenetről.

- Míg a leszámláló rendezés azt feltételezi, hogy a bemenet olyan egészekből áll, amelyek egy kis intervallumba tartoznak,
- addig az edényrendezés azt, hogy a bemenetet egy olyan véletlen folyamat generálja, amelyik egyenletesen osztja el az elemeket a [0, 1) intervallumon.

Edényrendezés

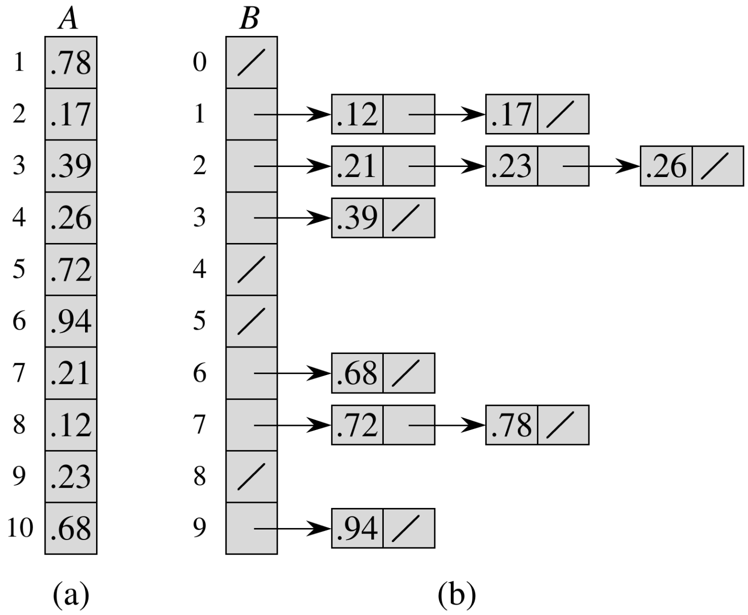
Az edényrendezés alábbi kódja feltételezi, hogy a bemenet egy *n* elemű *A* tömb, és a tömb minden egyes *A[i]* elemére teljesül, hogy $0 \le A[i] < 1$. A kódban szükség van továbbá egy, a láncolt listák (edények – vödrök) *B[0 . . n–1]* segédtömbjére, és feltesszük, hogy az ilyen listák kezeléséhez szükséges eljárások is rendelkezésünkre állnak.

```
procedure EDÉNY_RENDEZÉS(A)
    n ← méret(A)
    for i ← 1 to n do
        szúrjuk be az A[i] elemet a B[⌊nA[i]⌋] listába.
    end for
    for i ← 0 to n-1 do
        rendezzük a B[i] listát beszúrásos rendezéssel.
    end for
    sorban összefűzzük a B[0], B[1], . . ., B[n – 1] listákat.
end procedure
```


Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Edén446ényrendezés

Egy példa



Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Edényrendezés
Helyesség

Ahhoz, hogy lássuk az algoritmus helyes működését, tekintsünk két elemet, $A[i]$-t és $A[j]$-t.

- Tegyük fel az általánosság megszorítása nélkül, hogy $A[i] \leq A[j]$.
- Mivel ilyenkor $\lfloor nA[i] \rfloor \leq \lfloor nA[j] \rfloor$ is teljesül, az $A[i]$ elem
  - vagy ugyanabba az edénybe kerül, mint az $A[j]$,
  - vagy egy kisebb indexű edénybe.
- Ha $A[i]$ és $A[j]$ ugyanabba az edénybe került, akkor a 4–5. sorban található **for** ciklus a helyes sorrendbe állítja őket.
- Ha $A[i]$ és $A[j]$ különböző edényekbe kerültek, akkor a 6. sor állítja a helyes sorrendbe őket.

Tehát az edényrendezés helyesen működik.

Edényrendezés

Hatékonyság

A futási időt vizsgálva észrevehetjük, hogy az 5. sort kivéve az összes sor időigénye legrosszabb esetben $O(n)$. Marad még az 5. sorban levő n_i elem beszúrásos rendezése időigényének az elemzése

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

Ennek várható értéke:

$$\begin{aligned}
E\left[T(n)\right] &= E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\
&= \Theta(n) + \sum_{i=0}^{n-1} E\left[O(n_i^2)\right] && \text{(a várható érték linearitása miatt)} \\
&= \Theta(n) + \sum_{i=0}^{n-1} O(E\left[n_i^2\right]) && \text{(nem bizonyított tétel alapján)} \\
&= \Theta(n) + \sum_{i=0}^{n-1} O(2 - \frac{1}{n}) && \text{(lásd a tankönyv 8.4 fejezetben)}
\end{aligned}$$

Edényrendezés
Hatékonyság

Még ha a bemenet nem is egyenletes eloszlásból származik, az edényrendezés futhat lineáris időben.

Amíg a bemenet rendelkezik azzal a tulajdonsággal, hogy

- az edényméretek négyzeteinek összege lineáris a teljes elemszámban,

addig az edényrendezés futási ideje lineáris lesz.

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor

Most akkor melyik rendezést használjuk?

Kósa Márk
Pánovics János
Szathmáry László
Halász Gábor