Rendezés

Definíció: A reláció

Valamely A halmaz esetén a $\rho \subset A \times A$ részhalmazt az A halmazon értelmezett *relációnak* nevezzük. Azt mondjuk, hogy az A halmaz a és b eleme a ρ relációban van, ha $(a,b) \in \rho$. Röviden ezt így írjuk: $a \rho b$.

1. Példa: Legyen $A=R$, és a reláció a kisebb „<” jel. Az $a \rho b$ reláció azokat a számpárokat jelenti, amelyekre fennáll az $a<b$ összefüggés.

Definíció: A rendezési reláció

A ρ relációt *rendezési relációnak* nevezzük az A halmazon, ha

1. *reflexív*, azaz $a \rho a$ $\forall a \in A$ esetén;
2. $a \rho b$ és $b \rho a$ akkor és csak akkor áll fenn, ha $a=b$.
3. *tranzitív*, azaz $a \rho b$ és $b \rho c$ maga után vonja az $a \rho c$ teljesülését;
4. *antiszimmetrikus*, azaz vagy az $a \rho b$, vagy a $b \rho a$ fennáll $\forall a,b \in$ A esetén.

2. Példa: A valós számok közötti „≤” reláció rendezési reláció.

Rendezésről olyan adattípus esetén beszélhetünk, amelyre értelmezve van egy rendezési reláció. Tekintsük a sorozatot és annak is vegyük a tömbös realizációját. A rendezés a sorozat elemeinek olyan felsorolását jelenti, amelyben az egymást követő elemek a megadott relációban vannak. A tárgyalást valós számokon (leginkább egészek) visszük végig és relációnak a kisebb, egyenlő relációt (≤) tekintjük. ami nem csökkenti az általánosságot. A rendezés mind a mai időkig fontos informatikai probléma. Gyakran jelenik meg mint egy nagyobb probléma része. A rendezési algoritmusokkal kapcsolatban több szempont szerinti igény léphet föl. Ilyenek például az alábbiak:

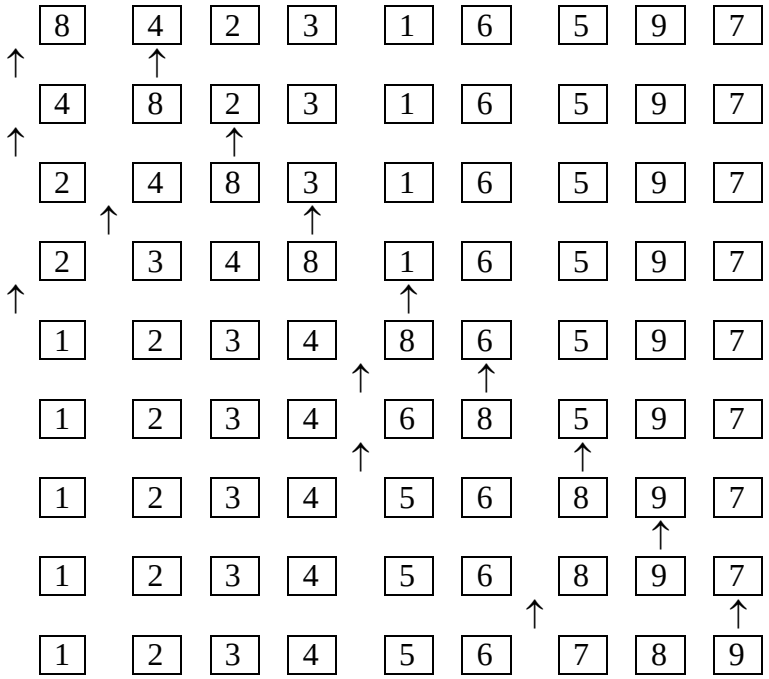
a. *Helyben rendezés*, azaz a rendezés eredménye az eredeti helyén jelenjen meg, legfeljebb konstans méretű többletmemória felhasználása révén.
b. *Gyorsaság*. A rendezési idő legyen minél rövidebb.
c. *Adaptivitás*. Az algoritmus használja ki a kulcsok között már meglévő rendezettséget.
d. *Stabilitás*. A rendezés őrizze meg az azonos kulcsú rekordok esetén a rekordok egymáshoz képesti eredeti sorrendjét. (Például telefonszámlák készítésekor az azonos kulcsú előfizetői hívások időrendi sorrendje maradjon meg.)
e. Az algoritmus csak a kulcsokat rendezze a rekordokra mutató pointerekkel, vagy az összes rekordot mozgassa.
f. *Belső rendezés* legyen (csak a belső memóriát vegye igénybe a rendezéshez), vagy *külső rendezés* legyen (háttértárakat is igénybe vehet).
g. *Összehasonlítás*on alapuljon a rendezés, vagy azt ne vegye igénybe az algoritmus. (Ez utóbbi esetben a kulcsokra további megszorításokat kell tenni.)
h. Optimális legyen a rendezési algoritmus, vagy sem. (Nem biztos, hogy az adatok az optimális algoritmus által megkívánt módon vannak megadva.)
i. Az összes rendezendő adatnak rendelkezésre kell-e állnia a rendezés teljes folyamata alatt, vagy sem.
j. A rendezésnek csak a befejeztével van eredmény, vagy menet közben is a már rendezett rész tovább nem változik.

Nem lehet kizárni a nem optimális algoritmusokat sem az alkalmazásokból, mert egy probléma megoldásában nem csak a rendezési algoritmus optimalitása az egyetlen szempont a problémamegoldás hatékonyságára. (Hiába gyors az algoritmus, ha az adatok nem a kívánt formában állnak rendelkezésre, és a konverzió lerontja a hatékonyságot.)

1. A beszúró rendezés

A beszúró rendezés alapelve nagyon egyszerű. A sorozat második elemétől kezdve (az első önmagában már rendezett) egyenként a kulcsokat a sorozat eleje felé haladva a megfelelő helyre mozgatjuk összehasonlítások révén. A sorozatnak a vizsgált kulcsot megelőző elemei mindig rendezettek az algoritmus során.

1. Példa: Az alábbi kulcsok esetén nyíl mutatja a mozgatandó kulcsot és a beszúrás helyét.



	1. algoritmus **Beszúró rendezés**
	// $T(n) = \Theta(n^2)$
1	BESZÚRÓ_RENDEZÉS (*A*)
2	// Input paraméter: *A* - a rendezendő tömb
3	// Output paraméter: *A* - a rendezett tömb
4	//
5	**FOR** $j \leftarrow 2$ **TO** ***hossz*** *[A]* **DO**
6	$kulcs \leftarrow A_j$
7	Beszúrás az $A_{1 \ldots j-1}$ rendezett sorozatba
8	$i \leftarrow j - 1$
9	**WHILE** $i > 0$ és $A_i > kulcs$ **DO**
10	$A_{i+1} \leftarrow A_i$
11	DEC(*i*)
12	$A_{i+1} \leftarrow kulcs$
13	**RETURN** (*A*)

2. A minimum kiválasztásos rendezés

***Hossz**[A]*-1 -szer végigmegyünk a tömbön. Minden alkalommal eggyel magasabb indexű elemtől indulunk. Megkeressük a minimális elemet, és azt az aktuális menet kezdő elemével felcseréljük.

	2. algoritmus **Minimum kiválasztásos rendezés**
	// $T(n) = \Theta(n^2)$
1	MINIMUM_KIVÁLASZTÁSOS_RENDEZÉS(*A*)
2	// Input paraméter: *A* - a rendezendő tömb
3	// Output paraméter: *A* - a rendezett tömb
4	//
5	**FOR** $i \leftarrow 1$ **TO** ***hossz[A]***-1 **DO**
6	// minimumkeresés
7	$k \leftarrow i$
8	$x \leftarrow A_i$
9	**FOR** $j \leftarrow i + 1$ **TO** ***hossz[A]*** **DO**
10	**IF** $A_j < x$
11	**THEN** $k \leftarrow j$
12	$x \leftarrow A_j$
13	// az *i*. elem és a minimum felcserélése
14	$A_k \leftarrow A_i$
15	$A_i \leftarrow x$
16	**RETURN** (*A*)

3. A buborékrendezés

A buborékrendezésnél az egymás mellett álló elemeket hasonlítjuk össze, és szükség esetén sorrendjüket felcseréljük. Ezt mindaddig folytatjuk, míg szükség van cserére.

	3. algoritmus **Buborékrendezés (Bubble Sort)**
	// $T(n)=\Theta(n^2)$
1	BUBORÉKRENDEZÉS(*A*)
2	// Input paraméter: *A* - a rendezendő tömb
3	// Output paraméter: *A* - a rendezett tömb
4	//
5	$j \leftarrow 2$
6	**REPEAT** *nemvoltcsere* $\leftarrow$ IGAZ
7	**FOR** $i \leftarrow$ ***hossz[A]*** **DOWNTO** *j* **DO**
8	**IF** $A_i < A_{i-1}$
9	**THEN** csere $A_i \leftrightarrow A_{i-1}$
10	*nemvoltcsere* $\leftarrow$ HAMIS
11	INC**(** *j* **)**
12	**UNTIL** *Nemvoltcsere*
13	**RETURN** (*A*)

Időigény a legrosszabb esetben: $T(n)=\frac{n\cdot(n-1)}{2}$

Az algoritmusra jellemző a sok csere, az elem lassan kerül a helyére.

4. A Shell rendezés
(rendezés fogyó növekménnyel)

A Shell rendezés a buborékrendezésnél tapasztalt lassú helyrekerülést igyekszik felgyorsítani azáltal, hogy egymástól távol álló elemeket hasonlít és cserél fel. A távolságot (itt *növekménynek* nevezik) fokozatosan csökkenti, míg az 1 nem lesz. Minden növekmény esetén beszúrásos rendezést végez az adott növekménynek megfelelő távolságra álló elemekre. Mire a növekmény 1 lesz, sok elem már majdnem a helyére kerül.

A növekmények felépítése. Használjunk t számú növekményt. Legyenek ezek $h_{1\ldots t}$.
A követelmény: $h_t = 1$, és $h_{i+1} < h_i$, $i = 1,\ldots t-1$.

A szakirodalomban javasolt növekményadatok: $t = \lceil \log n \rceil - 1$

$h_{i-1} = 2h_i$	…, 32, 16, 8, 4, 2, 1
$h_{i-1} = 3h_i + 1$	…121, 40, 13, 4, 1
$h_{i-1} = 2h_i - 1$	…31, 15, 7, 3, 1

	4. algoritmus **Shell rendezés**
	// $T(n) = \Theta(n^{1,5})$
1	SHELL_RENDEZÉS(*A*)
2	// Input paraméter: *A* - a rendezendő tömb
3	// Output paraméter: *A* - a rendezett tömb
4	//
5	**FOR** $s \leftarrow 1$ **TO** *t* **DO**
6	$m \leftarrow h_s$
8	**FOR** $j \leftarrow m + 1$ **TO** ***hossz[****A****]*** **DO**
9	$i \leftarrow j - m$
10	$k \leftarrow$ ***kulcs***$[A_j]$
11	$r \leftarrow A_j$
12	**WHILE** $i>0$ és $k < A_j$ **DO**
13	$A_{i+m} \leftarrow A_i$
14	$i \leftarrow i - m$
15	$A_{i+m} \leftarrow$ r
16	**RETURN** (*A*)

Megjegyzés: A hossz[A] a rendezendő elemek számát jelöli.

Példa: Shell rendezésre

Időigény: alkalmas növekmény választással leszorítható $T(n) = \Theta(n^{1,5})$-re.

5. Az összefésülő rendezés

Az összefésülő rendezés alapelve az összefésülés műveletén alapszik, amely két rendezett tömbből egy új rendezett tömböt állít elő. Az összefésülés folyamata: Mindkét tömbnek megvizsgáljuk az első elemét. A két elem közül a kisebbiket beírjuk az eredménytömb első szabad eleme helyére. A felszabaduló helyre újabb elemet veszünk abból a tömbből, ahonnan előzőleg a kisebbik elem jött. Ezt a tevékenységet folytatjuk mindaddig, míg valamelyik kiinduló tömbünk ki nem ürül. Ezután a még vizsgálat alatt lévő elemet, valamint a megmaradt másik tömb további elemeit sorba az eredménytömbhöz hozzáírjuk a végén. Az eredménytömb nem lehet azonos egyik bemeneti tömbbel sem, vagyis az eljárás nem helyben végzi az összefésülést.

Legyen ÖSSZEFÉSÜL (*A, p, q, r*) az az eljárás, amely összefésüli az $A_{p\,..\,q}$ és az $A_{q+1\,..\,r}$ résztömböket, majd az eredményt az eredeti $A_{p\,..\,r}$ helyre másolja vissza. Az eljárás lineáris méretű további segédmemóriát igényel. Az összefésülés időigénye $\Theta(n)$, ha összesen *n* elemünk van. (Egy menetben elvégezhető és az kell is hozzá.)

Definíció: Az oszd meg és uralkodj elv
Az oszd meg és uralkodj elv egy algoritmus tervezési stratégia A problémát olyan kisebb méretű, azonos részproblémákra osztjuk föl, amelyek rekurzívan megoldhatók. Ezután egyesítjük a megoldásokat.

Az összefésülő rendezés oszd meg és uralkodj típusú algoritmus, melynek az egyes fázisai:

Felosztás: A tömböt két $\frac{n}{2}$ elemű részre osztjuk

Uralkodás: Rekurzív összefésüléses módon mindkettőt rendezzük .(Az 1 elemű már rendezett)

Egyesítés: A két részsorozatot összefésüljük.

	5. algoritmus **Összefésülő rendezés (Merge Sort)**
	// $T(n)=\Theta(n\cdot\log n)$
1	ÖSSZEFÉSÜLŐ_RENDEZÉS (*A, p, r*)
2	// Input paraméter: *A* - a tömb, melynek egy részét rendezeni kell
3	// *p* - a rendezendő rész kezdőindexe
4	// *r* - a rendezendő rész végindexe
5	// Output paraméter: *A* - a rendezett résszel rendelkező tömb
6	//
7	**IF** $p < r$
8	**THEN** $q \leftarrow \left\lfloor \frac{p+r}{2} \right\rfloor$
9	ÖSSZEFÉSÜLŐ_RENDEZÉS (*A, p, q*)
10	ÖSSZEFÉSÜLŐ_RENDEZÉS (*A, q* + 1, *r*)
11	ÖSSZEFÉSÜL (*A, p, q, r*)
12	**RETURN** (A)

A teljes tömb rendezését megoldó utasítás: ÖSSZEFÉSÜLŐ_RENDEZÉS(*A,1,hossz[A]*).

Az összefésülő rendezés időigénye

Felosztás: $\Theta(1)$

Uralkodás: $2 \cdot T\left(\frac{n}{2}\right) \Rightarrow T(n) = \begin{cases} \Theta(1), & ha \quad n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n), & ha \quad n > 1 \end{cases}$

Egyesítés: $\Theta(n)$

Az algoritmus időigénye megkapható a mester tétel 2. pontja alapján: $T(n) = \Theta(n \cdot \log n)$.

5.1. A Batcher-féle páros-páratlan összefésülés

Az eljárás csak az összefésülést teszi hatékonyabbá. Nem önálló rendező módszer. Nagy előnye, hogy párhuzamosíthatók a lépései. Legyen két rendezett sorozatunk, az *n* elemű *A* sorozat és az *m* elemű *B* sorozat.

$A = \{a_1, \ldots, a_n\}$
$B = \{b_1, \ldots, b_m\}$

A két sorozat összefésülése adja a $C = \{c_1, \ldots, c_{n+m}\}$ sorozatot. Az összefésülés módja a következő: Mindkét kiinduló sorozatból kettőt képezünk, a páratlan indexű és a páros indexű elemek sorozatait:

$A_1 = \{a_1, a_3, a_5, \ldots\}$ $\quad$ $B_1 = \{b_1, b_3, b_5, \ldots\}$
$A_2 = \{a_2, a_4, a_6, \ldots\}$ $\quad$ $B_2 = \{b_2, b_4, b_6, \ldots\}$

Összefésüljük az A_1, B_2 sorozatokat, eredménye az *U* sorozat.
Összefésüljük az A_2, B_1 sorozatokat, eredménye a *V* sorozat.
Összefésüljük az *U* és *V* sorozatokat, eredmény a *C* sorozat.

Példa: Batcher összefésülésre

Tétel: A Batcher-féle összefésülés tétele
A Batcher összefésülés során

$$c_{2i-1} = \min\{u_i, v_i\} \text{ és } c_{2i} = \max\{u_i, v_i\}, \quad 1 \le i \le \frac{n+m}{2}$$

Bizonyítás

Fogadjuk el kiindulásként igaznak azt a feltevést, hogy *C* elejéből páros számú elemet véve azok között azonos számú *U* és *V* elem van. Ekkor
$\{c_1, \ldots, c_{2(i-1)}\} = \{u_1, \ldots, u_{i-1}\} \cup \{v_1, \ldots, v_{i-1}\}$ és $\{c_1, \ldots, c_{2i}\} = \{u_1, \ldots, u_i\} \cup \{v_1, \ldots, v_i\}$
Ebből viszont $\{c_{2i-1}, c_{2i}\} = \{u_1, v_i\}$, ahonnan $c_{2i-1} < c_{2i}$ miatt adódik a tétel állítása.
A feltételezésünk bizonyítása:
Legyen $\{c_1, \ldots, c_{2k}\} = \{a_1, \ldots, a_k\} \cup \{b_1, \ldots, b_{2k-s}\}$.
Ezek közül *U*-ba kerül $\left\lceil \frac{s}{2} \right\rceil$ elem az *A*-ból (az *A* páratlan indexű elemei) és

$\left\lfloor \frac{2k-s}{2} \right\rfloor$ elem a *B*-ből (a *B* páros indexű elemei),

Valamint *V*-be kerül $\left\lfloor \frac{s}{2} \right\rfloor$ elem az *A*-ból (az *A* páros indexű elemei) és $\left\lceil \frac{2k-s}{2} \right\rceil$ elem a *B*-ből (a *B* páratlan indexű elemei). Innen az *U*-beliek száma $\left\lceil \frac{s}{2} \right\rceil + \left\lfloor \frac{2k-s}{2} \right\rfloor = k$ és a V-beliek száma $\left\lfloor \frac{s}{2} \right\rfloor + \left\lceil \frac{2k-s}{2} \right\rceil = k$

■

6. Gyorsrendezés

Felosztás: Az $A_{p\,..\,r}$ tömböt két nemüres $A_{p\,...\,q}$ és $A_{q+1\,...\,r}$ részre osztjuk úgy, hogy $A_{p\,...\,q}$ minden eleme kisebb egyenlő legyen, mint $A_{q+1\,...\,r}$ bármely eleme. (A megfelelő q meghatározandó.)
Uralkodás: Az $A_{p\,...\,q}$ és $A_{q+1\,...\,r}$ résztömböket rekurzív gyorsrendezéssel rendezzük.
Egyesítés: Nincs rá szükség, mivel a tömb már rendezett. (A saját helyén rendeztük.)

	6. algoritmus **Gyorsrendezés (Quick Sort)**
	// $T(n)=\Theta(n^2)$, átlagos: $T(n)=\Theta(n\cdot\log n)$
1	GYORSRENDEZÉS(*A,p,r*)
2	// Input paraméter: *A* – a tömb, melynek egy részét rendezeni kell
3	// *p* - a rendezendő rész kezdőindexe
4	// *r* - a rendezendő rész végindexe
5	// Output paraméter: *A* - a rendezett résszel rendelkező tömb
6	//
7	**IF** $p < r$
8	**THEN** FELOSZT (*A, p, r,* A_p*, q*)
9	GYORSRENDEZÉS (*A, p, q*)
10	GYORSRENDEZÉS (*A, q*+1, *r*)
11	**RETURN** (A)

A gyorsrendezés időigénye:

A legrosszabb eset: a felosztás minden lépésben $n-1$, 1 elemű
$T(1)=\Theta(1)$,
$T(n)=T(n-1)+\Theta(n)$

$$T(n) = T(n-1)+\Theta(n)=T(n-2)+\Theta(n-1)+\Theta(n)=\ldots=$$
$$=T(1)+\Theta(1)+\Theta(2)+\ldots+\Theta(n)=\Theta\left(\sum_{k=1}^{n}k\right)=\Theta(n^2)$$

A legjobb eset: $\frac{n}{2},\frac{n}{2}$ a felosztás, ekkor
$T(1)=\Theta(1)$,
$T(n)=2\cdot T\left(\frac{n}{2}\right)+\Theta(n)$, ha $n > 1$
Ez megegyezik az összefésülő módszer formulájával, tehát $T(n)=\Theta(n\cdot\log n)$

Megjegyzés: ha a felosztás aránya állandó pl. $\frac{9}{10}$, $\frac{1}{10}$, akkor a rekurziós formula:
$T(n)=T\left(\frac{9}{10}\right)+T\left(\frac{1}{10}\right)+\Theta(n)$.
Bizonyítható, hogy ekkor is $T(n)=\Theta(n\cdot\log n)$. Ezen túlmenően az átlagos értékre is $T(n)=\Theta(n\cdot\log n)$ adódik.

7. Négyzetes rendezés

Felosztjuk az n elemű A tömböt (közel) $\sqrt{n}$ számú részre (alcsoportra). Mindegyikben (közel) $\sqrt{n}$ elemet helyezünk el, majd mindegyikből kiemeljük (eltávolítjuk) a legkisebbet. A kiemeltekből egy főcsoportot képzünk. Kiválasztjuk a főcsoport legkisebb elemét és azt az eredménytömbbe írjuk, a főcsoportból pedig eltávolítjuk (töröljük). Helyére abból az alcsoportból ahonnan ő származott újabb legkisebbiket emelünk be a főcsoportba. Az eljárást folytatjuk, míg az elemek el nem fogynak a főcsoportból.

Időigény: összehasonlításszám $T(n)=\Theta\left(n\cdot\sqrt{n}\right)=\Theta\left(n^{1,5}\right)$.

Továbbfejlesztett változat, amikor $\sqrt[3]{n}$ számú elem van egy fő-főcsoportban és $\sqrt[3]{n}$ számú főcsoport van, mindegyikben $\sqrt[3]{n}$ számú elemmel, melyek mindegyikéhez egy $\sqrt[3]{n}$ elemszámú alcsoport tartozik. A rendezés elve az előző algoritmuséhoz hasonló.

Időigény: $T(n)=\Theta\left(n\cdot\sqrt[3]{n}\right)=\Theta\left(n^{\frac{4}{3}}\right)$

A Stirling formula és az Alsó korlát összehasonlító rendezésre tétel

Tétel: A Stirling formula

Igaz az alábbi összefüggés az n!-ra:

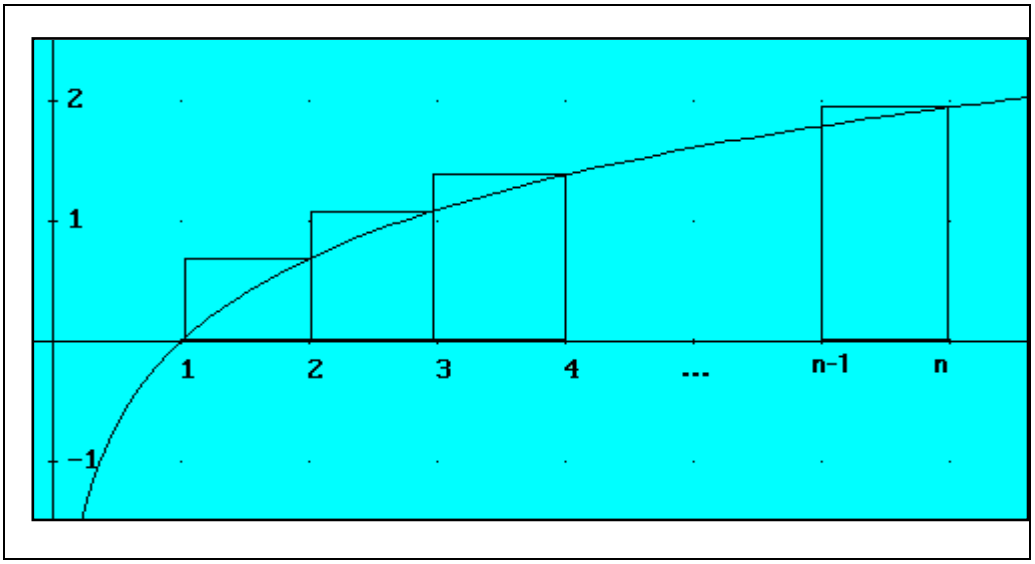
$$\frac{n^n}{e^n}<n!<\frac{(n+1)^{n+1}}{e^n},\ n=3,4,5,\ldots$$

Bizonyítás

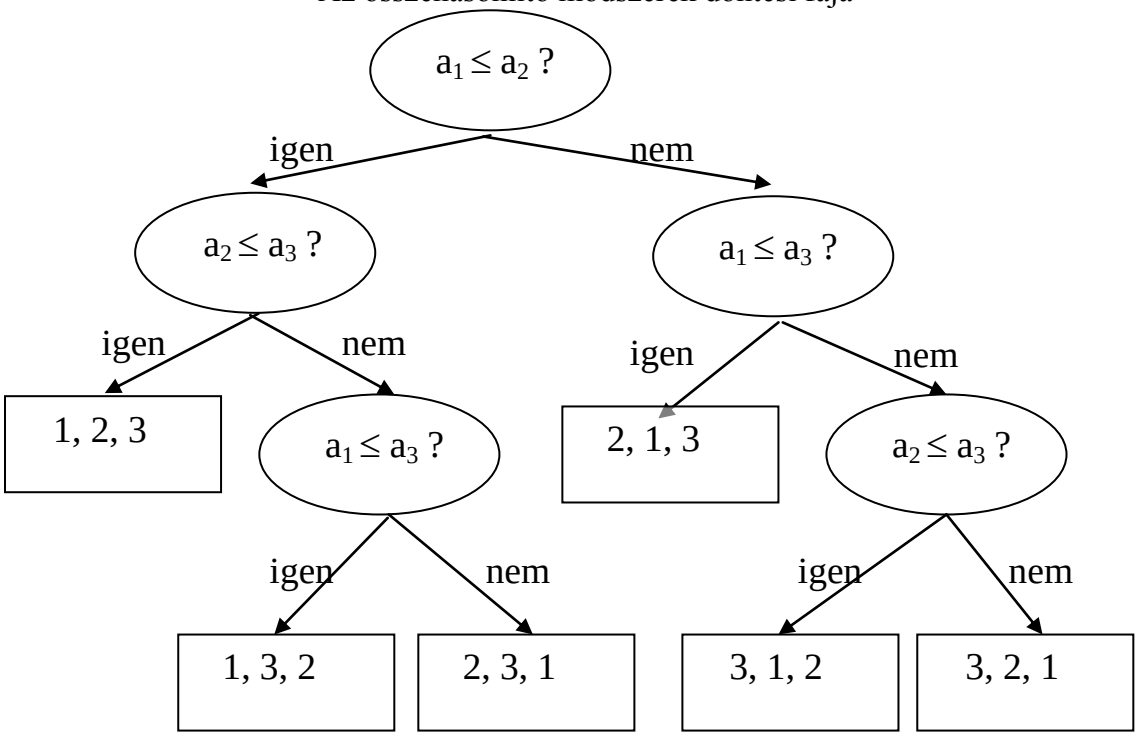
Az egyenlőtlenséget a logaritmusra látjuk be. A logaritmus függvény konkáv és emiatt írható:

$$\begin{aligned}
&\ln(n!)=1\cdot\ln 2+1\cdot\ln 3+1\cdot\ln 4+\ldots+1\cdot\ln n\\
&\ln(n!)>\int_1^n\ln x\,dx=\int_1^n 1\cdot\ln x\,dx=\left[x\cdot\ln x\right]_1^n-\int_1^n x\cdot\frac{1}{x}\,dx=\\
&\quad=n\ln n-\left[x\right]_1^n=n\ln n-(n-1)=n\ln n-n+1>n\ln n-n\\
&\ln(n!)=1\cdot\ln 1+1\cdot\ln 2+1\cdot\ln 3+1\cdot\ln 4+\ldots+1\cdot\ln n\\
&\ln(n!)<\int_1^{n+1}\ln x\,dx=\int_1^n 1\cdot\ln x\,dx=\left[x\cdot\ln x\right]_1^{n+1}-\left[x\right]_1^{n+1}=\\
&\quad=(n+1)\ln(n+1)-(n+1-1)=(n+1)\ln(n+1)-n
\end{aligned}\qquad(1)$$

■



Az összehasonlító módszerek döntési fája



Tétel: <u>Alsó korlát összehasonlító rendezésre</u>
Bármely *n* elemet rendező döntési fa magassága $T(n)=\Omega(n\cdot\log n)$

Bizonyítás

Egy h magasságú döntési fa leveleinek száma legfeljebb 2^h · Mivel minden permutációt rendezni kell tudnia az algoritmusnak, és összesen n! permutáció lehetséges, ezért a döntései fának legalább n! levele kell legyen. Tehát $n!\le 2^h$ fennáll. Logaritmálva: $h\ge\log(n!)$. A Stirling formula szerint $n!>\left(\frac{n}{e}\right)^n$. Behelyettesítve:

$$h\ge\log\left(\left(\frac{n}{e}\right)^n\right)=n\log n-n\log e.$$

Tehát: $h=\Omega(n\cdot\log n)$ ■

Lineáris idejű rendezők
8. A leszámláló rendezés

A lineáris idejű rendezők nem használják az összehasonlítást.

A leszámláló rendezés (= binsort, ládarendezés) **bemenete 1 és k közötti egész szám**.
Időigény: $T(n)=\Theta(n+k)$. Ha $k=\Theta(n)$, akkor a rendezési idő is $T(n)=\Theta(n)$, ahol n=hossz[A].

Az elemeket az $A_{1..n}$ tömbben helyezzük el. Szükség van további két tömbre: $B_{1..n}$ az eredményt tárolja majd, $C_{1..k}$ segédtömb.

A rendezés lényege, hogy A minden elemére meghatározza a nála kisebb elemek számát. Ez alapján tudja az elemet a kimeneti tömb megfelelő helyére tenni. Stabil eljárás: az azonos értékűek sorrendje megegyezik az eredetivel

	8. algoritmus **Leszámláló rendezés**
	// $T(n)=\Theta(n)$
1	LESZÁMLÁLÓ_RENDEZÉS (A, k, B)
2	// Input paraméter: A - a rendezendő tömb
3	// k – kulcs felső korlát, pozitív egész
4	// Output paraméter: B - a rendezett tömb
5	//
6	**FOR** $i \leftarrow 1$ **TO** k **DO**
7	$C_i \leftarrow 0$
8	**FOR** $j \leftarrow 1$ **TO hossz[**A**] DO**
9	**INC** (C_{A_j})
10	// C_i azt mutatja, hogy hány i értékű számunk van
11	**FOR** $i \leftarrow 2$ **TO** k **DO**
12	$C_i \leftarrow C_i + C_{i-1}$
13	// C_i most azt mutatja, hogy hány i-től nem nagyobb számunk van
14	**FOR** $j \leftarrow$ **hossz[**A**] DOWNTO** 1 **DO**
15	$B_{C_{A_j}} \leftarrow A_j$
16	**DEC (** C_{A_j} **)**
17	**RETURN** (B)

9. A számjegyes rendezés (radix rendezés)

Azonos hosszúságú szavak, stringek rendezésére használhatjuk. (Dátumok, számjegyekből álló számok, kártyák, stb.) Legyen d a szó hossza, k pedig az egy karakteren, mezőben előforduló lehetséges jegyek, jelek száma, n pedig az adatok száma.

Időigény: $T(n) = \Theta(d \cdot (n+k))$

	9. algoritmus **Számjegyes rendezés**
	// $T(n) = \Theta(d \cdot (n+k))$
1	SZÁMJEGYES_RENDEZÉS (A)
2	// Input paraméter: A - a rendezendő tömb
4	// Output paraméter: A - a rendezett tömb
4	//
5	**FOR** i ← d **DOWNTO** 1 **DO**
6	Stabil módszerrel rendezzük az A tömböt az i. számjegyre
7	**RETURN** (A)

10. Edényrendezés

Feltételezzük, hogy a bemenet a [0, 1) intervallumon egyenletes eloszlású számok sorozata. Felosztjuk a [0, 1) intervallumot n egyenlő részre (edények). A bemenetet szétosztjuk az edények között, minden edényben egy listát kezelve. Az azonos edénybe esőket beszúrásos módon rendezzük. A végén a listákat egybefűzzük az elsővel kezdve.

Várható időigény: $T(n) = \Theta(n)$

	10. algoritmus **Edényrendezés**
	// $T(n) = \Theta(n)$
1	EDÉNYRENDEZÉS (A, L)
2	// Input paraméter: A - a rendezendő tömb, n elemű
3	// Output paraméter: L - a rendezett elemek listája
4	// Menet közben szükség van egy n elemű B tömbre, mely listafejeket tárol. Indexelése 0-val indul.
5	**n ←hossz[A]**
6	**FOR** i ←1 **TO** n **DO**
7	Beszúrjuk az A_i elemet a $B_{\lfloor n \cdot A_i \rfloor}$ listába
8	**FOR** i ←0 **TO** n - 1 **DO**
9	Rendezzük a B_i listát beszúrásos rendezéssel
10	Sorban összefűzzük a $B_0, B_1, \ldots, B_{n-1}$ listákat. képezve az L listát
11	**RETURN** (L)

11. Külső tárak rendezése

Külső tárak rendezésénél az elérési és a mozgatási idő szerepe drasztikusan megnő. Az összefésüléses módszerek jönnek elsősorban számításba.

Definíció: A k hosszúságú futam file-ban
Egy file k szomszédos rekordjából álló részét **k hosszúságú futam**nak nevezzük, ha benne a rekordkulcsok rendezettek (pl.: növekedő sorrendűek).

Először alkalmas k-val (k=1 mindig megfelel) a rendezendő file-t két másik file-ba átmásoljuk úgy, hogy ott k hosszúságú futamok jöjjenek létre. Ezután a két file-t összefésüljük egy-egy elemet véve mindkét file-ból. Az eredményfile-ban már $2k$ lesz a futamhossz.(esetleg a legutolsó rövidebb lehet). Ezt ismételgetjük a teljes rendezettségig mindig duplázva k értékét. Legyen a rekordok száma n. Egy menetben n rekordmozgás van a szétdobásnál és n az összefésülésnél. A menetek száma legfeljebb $\lceil \log n \rceil$. **Az időigény:** $T(n) = \Theta(n \cdot \log n)$.

Külső tárak rendezésének gyorsítása

Az n log n nem javítható az összehasonlítások miatt. A szorzó konstansokat lehet csökkenteni.

Változó futamhosszak: a file-ban meglévő természetes módon kialakult futamhosszakat vesszük figyelembe. A futamhatárok figyelésének adminisztrálása bejön, mint további költség.
Több részfile használata esetén a szétdobás nem két, hanem több részre történik. Külön adminisztráció összefésülésnél a kiürült file-ok figyelése.
Polifázisú összefésülés alkalmazásakor nem folytatjuk végig minden menetben az összefésüléseket, hanem a célfile szerepét mindig a kiürült file veszi át és ide kezdjük összefésülni a többit

Egy eset polifázisú összefésülésre, amikor kínosan lassú a módszer. A tábla belsejében a file-ok futamszáma szerepel, zárójelben a futamok mérete. Kezdetben az első file 1 rekordból áll és egy a futamhossz, a második file 5 rekordból áll és szintén egy a futamhossz.

	menet és futamszám					
File	**1**	**2**	**3**	**4**	**5**	**6**
1	1(1)	0	1(3)	0	1(5)	0
2	5(1)	4(1)	3(1)	2(1)	1(1)	0
3	0	1(2)	0	1(4)	0	1(6)