SzA III. gyakorlat, 2013. szeptember 24/26.

Rendezettek vagyunk és jól keresünk

Drótos Márton
drotos@cs.bme.hu

1. **Rendezzük a következő listát buborék, kiválasztásos, beszúrásos, összefésüléses és gyorsrendezés segítségével:** $[4, 11, 9, 10, 5, 6, 8, 1, 2, 16]$.
   De tényleg!

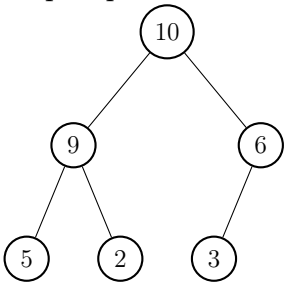
2. **Rendezzük a következő listát ládarendezéssel, ha tudjuk, hogy csak 0 és 10 közötti egész számok szerepelhetnek benne:** $[6, 4, 3, 8, 6, 3, 3, 5, 2]$
   Ez sem okoz senkinek nehézséget.

3. **Legalább hány összehasonlítás kell ahhoz, hogy egy n elemű tömbből egy olyan tagot találjunk meg, ami a tömb 10 legkisebb eleme közé tartozik?**
   $n - 10$ elég, hiszen ha kidobunk 9 random számot, a maradékból $(n - 9) - 1$ lépésben egy minimálisat keresve biztos, hogy a 10 legkisebb közül egyet találunk. $n-10$ kell is, mert tfh ennél kevesebbet használva az összehasonlítottsági gráfnak legalább 11 komponense lesz, így (a minimukeresés alsó korlátjának bizonyításával analóg módon) egy ellenség meg tudja akadályozni, hogy jó megoldást találjunk.

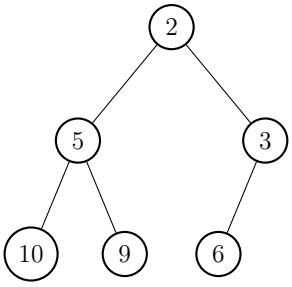
4. **Az A tömbben n különböző számot tárolunk. Tudjuk, hogy $A[1] > A[2]$ és $A[n-1] < A[n]$. Adjunk algoritmust, mely $c \cdot \log n$ összehasonlítással megtalál a tömbben egy lokális minimumot (ha van), azaz egy olyan $1 \leq i \leq n$ indexet, hogy $A[i]$ tömbbeli szomszédai nagyobbak, mint $A[i]$.**
   A feltételekből következik, hogy biztos van lokális minimum (tfh nincs, ekkor a számok rendezettek, ami ellentmond a feltételnek). Bináris keresést használunk, ha az adott körben $A[i]$ egy lokális minimum, akkor kész vagyunk, ha $A[i-1] < A[i]$, akkor a jobb oldalt dobjuk el, egyébként a balt. Tehát egy lépésben vagy megtaláltuk a kérdéses elemet, vagy kaptunk egy feleakkora tömböt, ami ugyanolyan tulajdonságú, mint az eredeti. Ez így $c \cdot \log n$ a bináris keresés elve miatt.

5. **Adottak a $p_0 = (0, 0), p_1 = (x_1, y_1), \ldots p_n = (x_n, y_n), p_{n+1} = (100, 0)$ pontok a síkban ($n \geq 1$) úgy, hogy $1 \leq i \leq n$ esetén x_i és y_i racionális számok, $0 < x_i < 100$, és semelyik három pont nem esik egy egyenesbe. Egyenes szakaszokkal akarjuk ezeket a pontokat valamilyen sorrendben összekötni úgy, hogy egy $n + 2$ csúcsú zárt töröttvonalat kapjunk, amiben a behúzott szakaszok nem metszik egymást. Adjunk egy legfeljebb $c \cdot n \log n$ lépést használó algoritmust annak meghatározására, melyik pontot melyikkel kössük össze!**
   Észrevehetjük, hogy amennyiben a p_0-ból indulunk, és p_{n+1}-ig nem érintünk negatív y koordinátájú pontot, visszafele pedig pozitívat, akkor amennyiben odafele x szerint növekvően, visszafele pedig csökkenően haladunk, akkor nem hozunk létre metszéspontot (a feltétel felhasználásával belátandó). Vagyis két részre osztjuk a pontjainkat y koordináta szerint, és ezt a két részt x szerint rendezzük, pl. öf rendezéssel. Így a lépésszám $n + 2cn \log n \approx c' n \log n$.

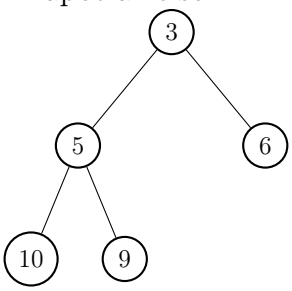
6. **Rendezzük a következő számokat kupacos rendezéssel: 10, 9, 6, 5, 2, 3**
   Kupacépítés kezdeti állapota a tömbből:



A kupacépítés végeredménye (ZH-n a lépések lerajzolandók!):



Állapot az első MINTÖR után:



Után értelemszerűen meg kell csinálni egy csomó MINTÖR-t, amit most nem rajzolok le.

---

7. **A $[6, 4, 8, 3, 7, 2, 5, 1]$ tömb rendezése során (a rendező algoritmus néhány lépése után) a következő közbülső állapot jött létre: $[4, 6, 3, 8, 7, 2, 5, 1]$. Az alább felsorolt módszerek közül mely(ek) alkalmazásakor fordulhatott elő?**

   (a) **beszúrásos rendezés** nem, mert teljes méretű listánk csak a rendezés legvégén van, viszont akkor már rendezett.

   (b) **buborékrendezés** igen, először 4-6 csere, aztán 6-8 nem csere, majd 8-3 csere, és pont ez jön ki.

   (c) **összefésüléses rendezés** A $[6, 4]$, $[8, 3]$, $[7, 2]$, $[5, 1]$ kis listákból az első kettőt már rendezte, a többit még nem, tehát ez egy lehetséges közbülső állapot.

   (d) **gyorsrendezés** Nem, hiszen be lehet látni, hogy egyik elem kiválasztása esetén se juthattunk erre az állapotra.

8. **Szeretnénk n db SZA-hallgató ZH-ereményeit (csak az összpontszámot) növekvő sorrendben felsorolni. Adjunk erre $c \cdot n$ lépést felhasználó algoritmust!**
   Mivel a ZH-eredmény egy $[0, 60]$ intervallumba eső egész szám, ezért tudunk ládarendezést alkalmazni 61 láda felhasználásával, vagyis a lépésszám $n + 61 \approx c \cdot n$.

9. **Adjunk $c \cdot n$ lépésszámú algoritmust n olyan egész számból álló sorozat rendezésére, melynek elemei az $\{1, \ldots, 3n\}$ tartományba esnek!**
   Ládarendezés $3n$ ládával (a lépésszám ekkor beírás (n) + minden láda kiolvasása $(3n)$, azaz összesen $4n$).

10. **[pótpótZH, 2010. ősz] A valós számokból álló $a_1, \ldots, a_n$ sorozat olyan, hogy az $a_1^3, a_2^3, \ldots, a_n^3$ sorozat egy darabig nő, utána csökken. Adjunk konstansszor n összehasonlítást használó algoritmust, ami rendezi az $a_1, \ldots, a_n$ sorozatot.**
    Az a^3 függvény szigorúan monoton nő, így a sorozat az a_i értékek szerint is úgy néz ki, hogy egy darabig nő, utána csökken. Kezdjük el olvasni a sorozatot elölről és hátulról (mintha két különböző sorozat lenne), és az értékeket fésüljük össze. Ha összeértünk a két olvasással, akkor készen vagyunk. n elem összefésülése $c \cdot n$ lépés, így ez megfelelő. (Természetesen azt is megcsinálhatjuk, hogy megkeressük a töréspontot, szétszedjük a sorozatot két külön önmagában rendezett listába, aztán azokat fésüljük össze.)

11. **Az $A[1 \dots n]$ tömbben egész számokat tárolunk, ugyanaz a szám többször is szerepelhet. Határozzuk meg $c \cdot n \log n$ lépésben az összes olyan számot, amelyik egynél többször fordul elő a tömbben!**
A tömböt tudjuk rendezni $c_1 n \log n$ lépésben pl összefésüléses rendezéssel. Így biztos, hogy a többször előforduló elemek példányai szomszédosak lesznek. Már csak annyi a dolgunk, hogy végigolvassuk a tömböt, és minden lépésben a következő dolgokat tartjuk nyilván: az előző elem, a legutolsó többszörösként kírt elem, és az aktuális elem. Ha az aktuális nem egyezik az utolsóval, akkor továbblépünk és az utolsót aktualizáljuk, ha egyezik, és az utolsó kiírt ugyanez volt, akkor továbblépünk, ha nem ugyanez volt, akkor kiírjuk és az utolsó kiírtat aktualizáljuk. A végigolvasás során minden cellában konstans sok műveletet végzünk, így $c_2 n$ lépésben kész vagyunk. A teljes lépésszám $c_1 n \log n + c_2 n \approx cn \log n$.

12. **[ZH, 2012. október 11.] Adott $n + 2$ rendezett tömb, méreteik rendre $1, 1, 2, 4, 8, \dots, 2^n$. Adjunk olyan eljárást, ami legfeljebb 2^{n+2} összehasonlítással rendezi a tömbökben tárolt rekordokat.**
Tanultuk, hogy egy n és egy k méretű rendezett tömb összefésüléséhez legfeljebb $n + k$ összehasonlítás szükséges. (Valójában elegendő $n + k - 1$ is, de a számolás egyszerűbb a fenti, rosszabb becslésse.) (3 pont)
Fésüljük össze a két 1 méretű tömböt, majd az így kapott 2 méretűt az eredetileg kapott 2 méretűvel, az így kapott 4 méretű tömböt az inputban szereplő 4 méretűvel, sít. (4 pont)
Az egyes összefésülésekhez legfeljebb $2, 4, 8, \dots, 2^{n+1}$ összehasonlítás szükséges, így az összehasonlítások száma a teljes eljárásban legfeljebb $2 + 4 + 8 + \dots + 2^{n+1} = 2^{n+2} - 2 < 2^{n+2}$. (3 pont)

13. **Az $A[1 : n]$ tömbben levő elemekről tudjuk, hogy $A[1] \neq A[n]$. Adjunk $c \log n$ összehasonlítást használó algoritmust, amely talál egy olyan i indexet, hogy $A[i] \neq A[i + 1]$!**
Egy ilyen tömbben biztosan létezik megfelelő elempár, hiszen a nem létezése azt jelentené, hogy bármely két szomszédos elem megegyezik, így $A[1] = A[2] = \dots = A[n]$, ami ellentmond a feltételnek. Így viszont a bináris keresés használható, ugyanis ha megfelezzük a tömböt, akkor a felezésnél lévő elem az első és utolsó közül legalább az egyikkel nem egyenlő, így a megfelelő irányban folytatva az eljárást az eredeti tulajdonsággal rendelkező, de feleakkora tömböt kapunk. Vagyis az algoritmus $c \log n$ lépés alatt végez.

14. **A $2^k - 1$ elemű A tömb elemei mind különbözőek és növekvő sorrendben vannak. Minden elemet egy k hosszú bitsorozat ír le, tehát tekinthetjük úgy, hogy a $0, 1, 2, \dots, 2^k - 1$ számokat tároljuk egy kivételével. A feladat ennek a hiányzó számnak a megkeresése. Ehhez egy lépésben valamelyik elem egy bitjére kérdezhetünk rá: a $BIT(i, j)$ eljárás az $A[i]$ elem j-edik bitjét mondja meg. Adjunk olyan algoritmust, amely a BIT eljárás $c \cdot k$-szori hívásával megtalálja a hiányzó számot (bitsorozatot)!**
Mivel egy hiányzik, egy helyen elromlik a paritás (aminek ellenőrzéséhez elég az utolsó számjegyre rákérdezni). Ezt a helyet keressük meg bináris kereséssel, ami $c \cdot \log(2^k - 1)) \approx c \cdot k$.

15. **Adott a síkon n pont, melyek koordinátái $(a_1, b_1) \dots (a_n, b_n)$. Olyan $P = (x, y)$ pontot keresünk a síkon, amire az alábbi összeg minimális.**

$$\sum_{i=1}^{n} (|a_i - x| + |b_i - y|)$$

Adjunk algoritmust, ami $c \cdot n \log n$ lépésben meghatároz egy ilyen P pontot!
Észrevesszük, hogy x és y koordináta független. Elég tehát egy dimenzióban gondolkodni:

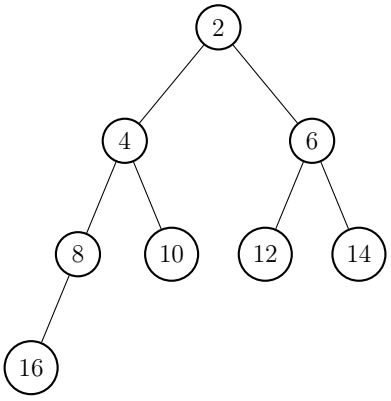
ha csak két pont van, akkor két pont között bárhol lehet, rajtuk kívül viszont rosszabb. Tehát a két szélső pont között kell lennie, de rajtuk kívül a két szélső között is stb, tehát páratlan pont esetén a középsőn, páros esetén a középső kettő közül valamelyiken vagy közöttük (indoklás: ha nem így lenne, akkor biztos, hogy tudnánk jobbat csinálni). Tehát koordinátánként egy rendezés, és középső választása, vagyis $2cn \log n$ lépés.

16. **Adottak a sík egész koordinátájú $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$ koordinátájú pontjai. Javasoljunk egy legfeljebb $c \cdot n$ lépésszámú módszert olyan $P_i \neq P_j$ pontok kiválasztására, amelyeken átmenő egyenes által meghatározott félsíkok közül az egyik tartalmazza az összes pontot!**
Minimális y kooridántájú pont lesz az egyik, ami meghatározza, a másik pedig az y_{min}-es pontból abszolútértékben legnagyobb meredekségű. Ha ezeken kívül esne egy pont, akkor annak a meredeksége abszolútértékben nagyobb lenne. Meredekséget számolni két pont ismeretében konstans, így a lépésszám két minimumkeresés, azaz $c_1 \cdot n + c_2 \cdot n = c \cdot n$. (Persze ugyanezt lehet x koordináta szerint is, valamint minimum helyett maximummal is.)

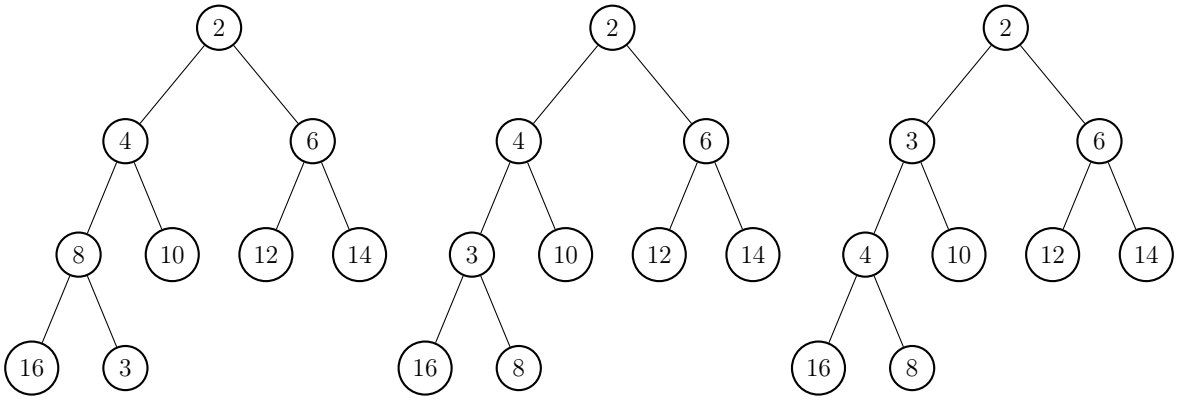
---

17. **[ZH, 2012. október 11.] Kupac-e az** | **1** | **4** | **2** | **8** | **3** | **7** | **5** | **tömb?**
A kupactulajdonság akkor teljesül egy tömbre, ha minden i-re igaz, hogy a tömbben i-ediknek tárolt rekord nem nagyobb sem a tömbben $2i$-ediknek, sem pedig a $(2i+1)$-ediknek tárolt rekordnál. (Természetesen akkor, ha léteznek a tömbben a kérdéses elemek.) (5 pont)
Azonban az 5-ödiknek tárolt rekord kisebb a második rekordnál, (3 pont)
a kupactulajdonság tehát nem teljesül, (1 pont)
a kérdezett tömb nem kupac. (1 pont)

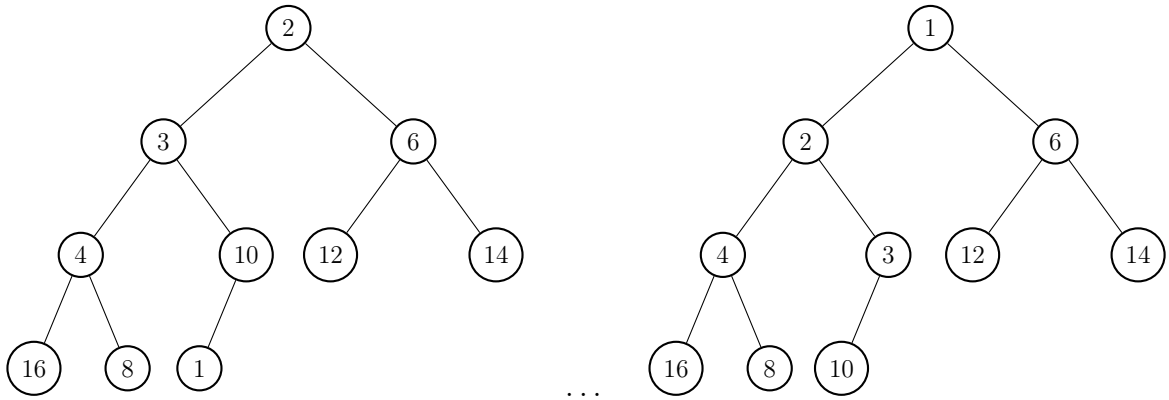
18. **A 10 elemű A tömb első 8 elemére legyen $A[i] = 2i (1 \leq i \leq 8)$, és tekintsük ezt, mint egy 8 elemű kupacot. Rajzoljuk le az ehhez tartozó fát! Hajtsuk végre rajta a BESZÚR(3), BESZÚR(1), MINTÖR műveletsort!**



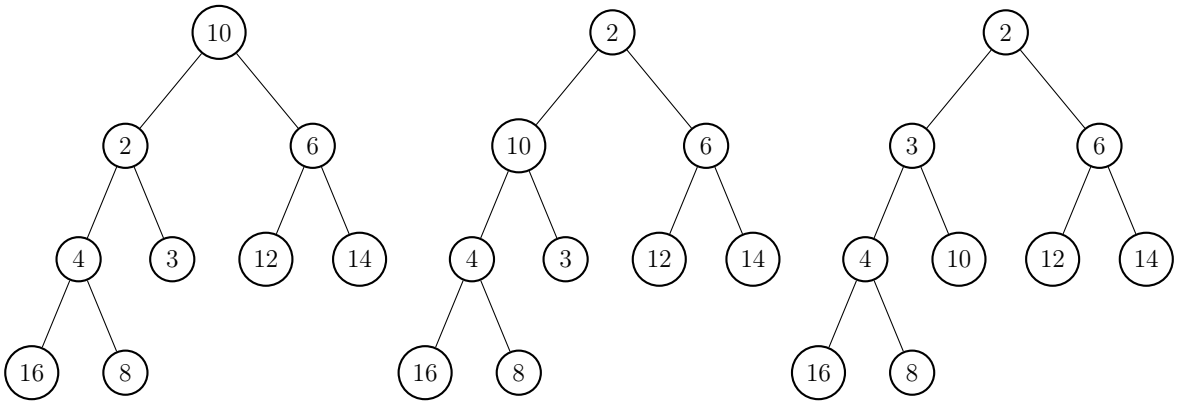
BESZÚR(3):



BESZÚR(1):



MINTÖR:



19. **[pótpótZH, 2012. december 12.] Hányféleképpen lehet egy 5 méretű tömbbe úgy beleírni az** $1, 2, 3, 4, 5$ **rekordokat, hogy kupacot kapjunk?**
Legyen $A[1 \dots 5]$ a tömb. Világos, hogy $A[1] = 1$, hiszen ez a kupacban tárolt legkisebb rekord. (1 pont)
A kupactulajdonságból még annyi következik, hogy az $A[2]$ rekord kisebb az $A[4]$ és $A[5]$-ben tároltnál, egyéb feltétele nincs annak, hogy kupac legyen a tömb. (2 pont)
Ezért $A[2]$ csak 2 vagy 3 lehet. (1 pont)
Ha $A[2] = 3$, akkor $A[4]$ és $A[5]$ a 4 és 5 értéket kapja meg, míg $A[3] = 2$ lesz, kizárásos alapon. (2 pont)
Ez tehát 2 lehetőség. (1 pont)
Ha pedig $A[2] = 2$, akkor az $A[3], A[4], A[5]$ helyre a $3, 4, 5$ rekordokat tetszőleges permutációban beírhatjuk, a kupactulajdonság teljesülni fog. (2 pont)
Ekkor tehát $3! = 6$ lehetőségünk van a permutáció választására, (1 pont)
a feladatbeli kérdésre a válasz tehát $2 + 6 = 8$. (1 pont)
(Ez így 11 pont, de ez hiba a javítókulcsban, ugyanúgy 10 volt a max.)

20. **Egy orvosi rendelőben a regisztrációnál kell bejelentkezni, ahol az ott dolgozók eldöntik, hogy a beteg az épp rendelő két orvos közül** A**-hoz vagy** B**-hez kell kerüljön, vagy bármelyikükhöz kerülhet. Ezen kívül, a beutaló ismeretében, a beteghez egy, a sürgősséget kifejező, számot is rendelnek. Amikor valamelyik orvos végzett egy beteggel, akkor azon betegek közül, akiket nem csak a másik orvos láthat el behívja a legnagyobb sürgősségi számút. Tegyük fel, hogy a kiosztott sürgősségi számok egymástól különbözőek. Írjunk le egy olyan adatszerkezetet, ami abban az esetben ha** n **beteg várakozik, akkor a regisztráción**

az új beteg beillesztését, illetve az orvosoknak a következő beteg kiválasztását $c \log n$ lépésben lehetővé teszi!
3 max-kupacot fogunk nyilvántartani, az egyikbe mennek A betegei, a másikba B betegei, a harmadikba $A \vee B$ betegei. Beszúrás: a beteg típusától függő kupacba (ahol jelenleg $m \leq n$ beteg van), $c \log m \leq c \log n$. Ha A orvos végez, akkor a $\max(\max_A, \max_{A \vee B})$ sorszámú beteget hívja be (mivel a keresett elemek a kupacok gyökerében vannak, ezért ez konstans lépés), majd a kiválasztott beteg kupacában csinálni kell egy MAXTÖR-t, ami, ha $m < n$ eleme van, akkor $c \log m \leq c \log n$ lépés. A helyesség triviálisan látható.

21. **Adott egy n elemet tartalmazó kupac és egy k kulcs. Keressük meg a kupac k-nál kisebb elemeit! Ha m ilyen elem van, akkor az algoritmus $c \cdot m$ elemi lépést használhat.**
Lényegében egy bejárás, elindulunk a gyökérből, és amíg k-nál kisebb elemet találunk, addig kiírjuk, ha $\geq k$-t, akkor az adott ágat hanyagoljuk, hiszen a kupac-tulajdonság miatt arrafele csak még nagyobbak lehetnek. Legfeljebb a nekünk jó elemek mindkét gyerekét vizsgáljuk meg feleslegesen, vagyis legfeljebb $2m = cm$ vizsgálatot végzünk.

22. 🍃 **Egy kupacba beraktunk egy új x elemet, majd végrehajtottunk egy MINTÖR műveletet. Mikor fordul elő, hogy végül az eredeti kupacot kapjuk vissza?**
Pontosan akkor, ha az eddig tárolt elemeknél kisebb x. Ez a feltétel szükséges, hiszen a MINTÖR a legkisebbet fogja visszaadni, és ha az új nem az lenne, akkor mindenképpen bennemaradna. Az elégségesség kicsit macerább. Amit tudunk: x a gyökérig felment, és legalulról indult. MINTÖRNÉL a legalsó kerül x helyébe (nevezzük y-nak), az fog lefele menni. Azt akarjuk bizonyítani, hogy pont az eredeti helyére kerül vissza (és mindenki más is, aki az úton volt). Indukcióval az aktuális szintre: a gyökérben most y van, egyik gyereke az eddigi legkisebb, és pont azon az oldalon van, ahonnan x eredetileg „feljött". Mindenképpen ő fog a gyökérbe kerülni, ezért y vele fog helyet cserélni, azaz ugyanazon az úton indul lefele, ahonnan x jött. Ez az érvelés pontosan ugyanígy megy tetszőleges közbenső szinten is, ha y aktuális pozícióját vesszük a gyökérnek.

23. 🍃 **Igazoljuk, hogy egy n elemből álló kupac felépítése legalább $c \cdot n$ összehasonlítást igényel!**
Fejet asztalbaverően egyszerű bizonyítás: mivel a gyökérben a legkisebb elem van, ezért nem lehet hamarabb felépíteni, mint a legkisebb elemet megtalálni, ami a tanultak alapján pedig önmagában is legalább $n - 1$ összehasonlítás.