# SzA III. gyakorlat

## Rendezettek vagyunk és jól keresünk

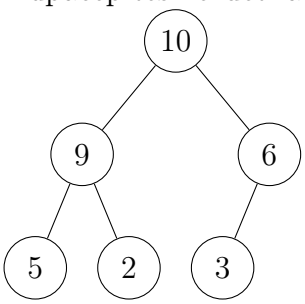
## 2012. szeptember 20.

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

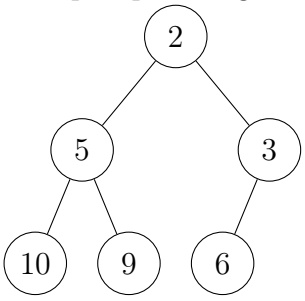
2. **Rendezzük a következő listát ládarendezéssel, ha tudjuk, hogy csak 0 és 10 közötti egész számok szerepelhetnek benne:** $[6, 4, 3, 8, 6, 3, 3, 5, 2]$
   Ez sem okoz senkinek nehézséget.

3. **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 $n + 2cn \log n \approx c'n \log n$.

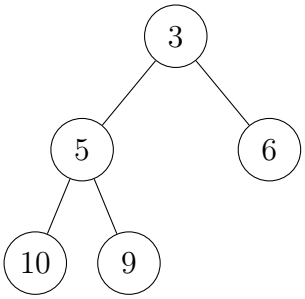
4. **Rendezzük a következő számokat kupacos rendezéssel: 10, 9, 6, 5, 2, 3**
   Kupacépítés kezdeti állapota a tömbből:




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

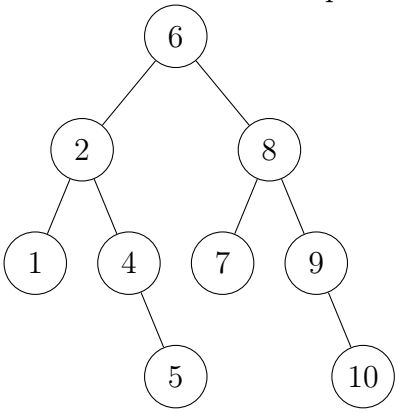



   Állapot az első MINTÖR után:

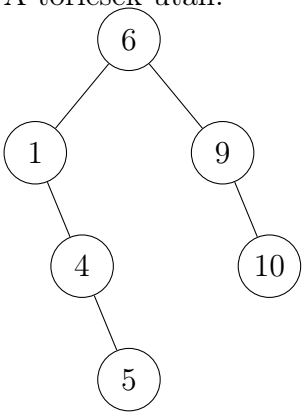



Után értelemszerűen meg kell csinálni egy csomó MINTÖR-t, amit most nem rajzolok le.

5. **Szúrjuk be egy kezdetben üres bináris keresőfába a 6, 2, 1, 4, 5, 8, 9, 7, 10 elemeket, majd töröljük a 7, 8, 2 elemeket!**
   A beszúrások utáni állapot:




   A törlések után:




---

6. **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.

7. **Szeretnénk** $n$ **db SZA-hallgató ZH-eredményeit (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.

8. **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 $4n$).

9. **[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.)

10. **Az $A[1 \ldots n]$ 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 $c_1 n \log n + c_2 n \approx cn \log n$.

11. **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 $A[1] = A[2] = \cdots = A[n]$, 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.

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

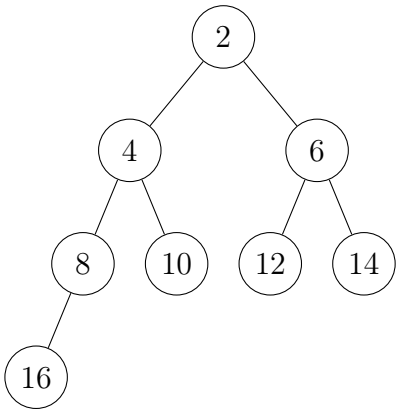
$$\sum_{i=1}^{n} (|a_i - x| + |b_i - y|)$$

**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.

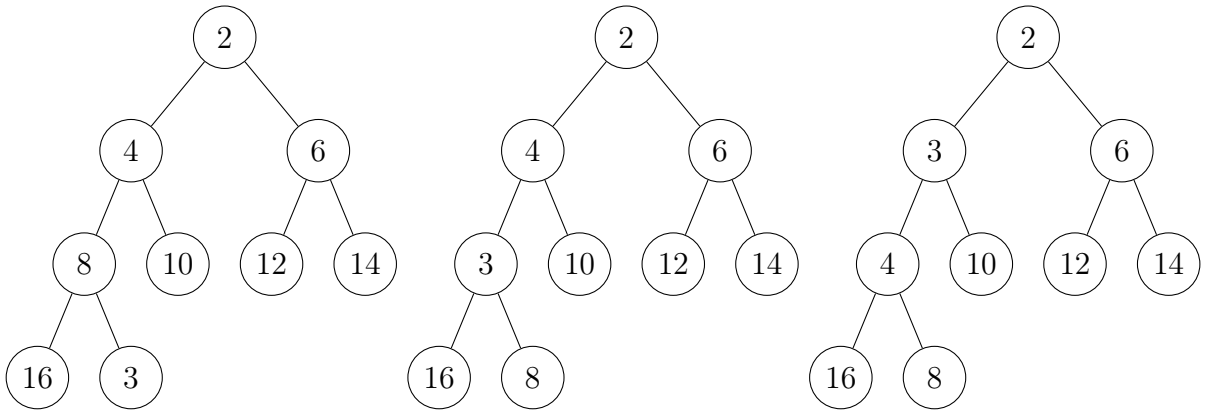
13. **Adottak a sík egész koordinátájú $P_1 = (x_1, y_1), \ldots, P_n = (x_n, y_n)$ koordinátájú pontjai. Javasoljunk egy legfeljebb $c \cdot n$ lépésszámú módszert olyan $P_i \neq P_j$ pontok kiválasztására, amelyeken átmenő egyenes által meghatározott félsíkok közül az egyik tartalmazza az összes pontot!**
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.)

---

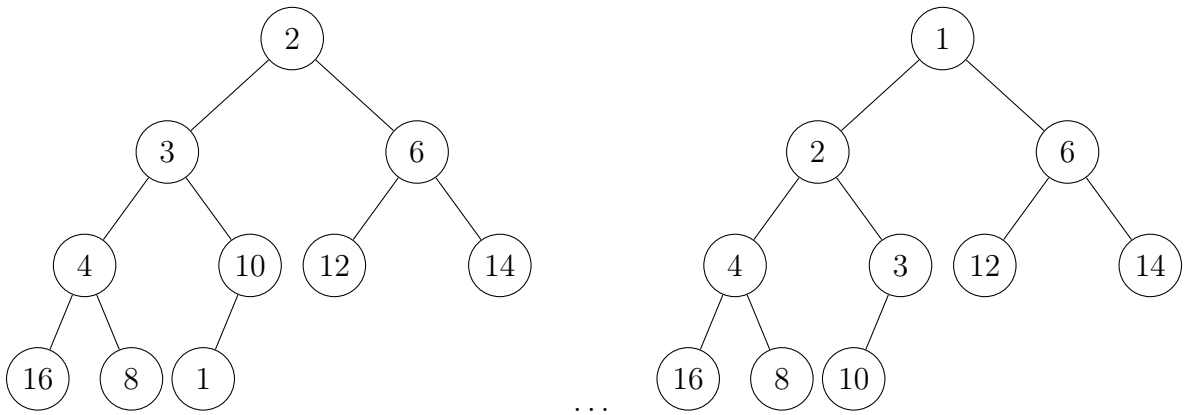
14. **A 10 elemű $A$ tömb első 8 elemére legyen $A[i] = 2i (1 \leq i \leq 8)$, és tekintsük ezt, mint egy 8 elemű kupacot. Rajzoljuk le az ehhez tartozó fát! Hajtsuk végre rajta a BESZÚR(3), BESZÚR(1), MINTÖR műveletsort!**



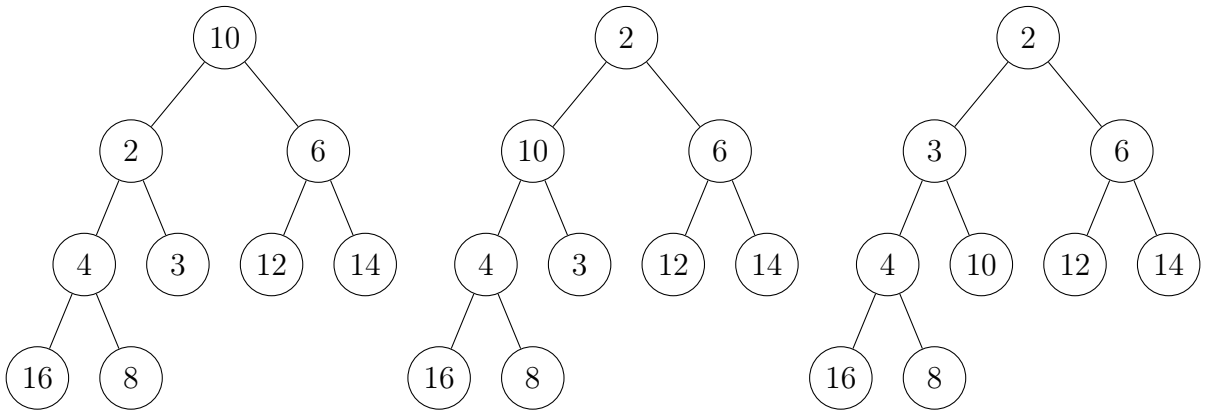

BESZÚR(3):




BESZÚR(1):



MINTÖR:




15. **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.

16. **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.

17. **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.

18. **Adott egy $n$ csúcsú és egy $k$ csúcsú keresőfa. A két fában tárolt összes elemből $c(n+k)$ lépésben készítsünk rendezett tömböt!**
Kiolvassuk inorder bejárással rendezetten a két fában tárolt elemet egy-egy tömbbe/listába ($n$ és $k$ lépés), majd ezeket összefésüljük ($n+k$ lépés). Összegezve $c(n+k)$ lépés. (Persze az inorder bejárás közben azonnal is csinálhatjuk az összefésülést.)

19. **Egy bináris keresőfában csupa különböző egész számot tárolunk. Lehetséges-e, hogy egy $KERES(x)$ hívás során a keresési út mentén a $20, 18, 3, 15, 5, 8, 9$ kulcsokat látjuk ebben a sorrendben? Ha lehetséges, határozzuk meg az összes olyan $x$ egész számot, amire ez megtörténhet! Ha nem lehetséges, miért nem?**
Lehet, hiszen az $x < 20$, $x < 18$, $x > 3$, $x < 15$, $x > 8$, $x \neq 9$-et kielégítő szám lehet, vagyis $9 < x < 15$ közül bármi jó, valamint a keresőfa tulajdonság sem sérül sehol.

20. **Egy bináris fa csúcsai 0 és 9 közötti egész számokkal vannak megcímkézve. Az inorder bejárás során a címkék sorrendje: 9, 3, 1, 0, 4, 2, 7, 6, 8, 5, a postorder bejárásnál pedig 9, 1, 4, 0, 3, $x$, 7, 5, $y$, 2. Mi lehet az $x$ és mi az $y$?**
2=gyökér (postorder miatt), 7,6,8,5 a jobb részfában van, többi a balban (inorder miatt) (3 pont). Két lehetőségünk van, $x = 6, y = 8$ és $x = 8, y = 6$. Ezeket kipróbálva (az akutális részfa gyökerét meghatározva, mint először) kijön, hogy előbbi lehet (3 pont), utóbbi nem (4 pont) (ezeket végig kell számolni!).

21. **Egy bináris keresőfa csúcsait egy, a gyökértől egy levélig menő út szerint három osztályba soroljuk: $B$ az úttól balra levő, $U$ az útra eső, $J$ pedig az úttól jobbra levő csúcsok halmazát jelöli. Igaz-e mindig, hogy minden $B$-beli csúcs kulcsa kisebb tetszőleges $U$-beli csúcs kulcsánál, és minden $U$-belis csúcs kulcsa kisebb tetszőleges $J$-beli csúcs kulcsánál?**
Nem, ellenpéldát lehet rá adni egyszerűen: a gyökérben mondjuk legyen 1, ennek (egyetlen) fia 3, ennek pedig két fia, 2 és 4. Az út: $\{1, 3, 4\}$, ettől balra van 2, ami nem kisebb, mint 1.

22. **Az MSc-re jelentkezőknek a felvételit alkotó 3 témakör mindegyikéből lesz egy írásbeli pontszámuk ($P_1, P_2, P_3$), és keletkezik egy felvélteli pontszámuk is ($FP$). Tegyük fel, hogy a $P_i$-k 1 és 30 közötti egészek, míg az $FP$ tetszőleges pozitív egész szám lehet. Adjunk meg egy olyan adatszerkezetet, amivel a következő műveletek az adott időben végrehajthatóak ($n$ a jelentkezők számát jelöli)!**
**BESZÚR($P_1, P_2, P_3, FP$): az adott pontszámok beillesztése – átlagosan $c \log n$**
**KERES($p$): a pontosan $p$ felvételi ponttal ($FP = p$) rendelkező jelentkezők számát határozza meg – átlagosan $c \log n$**
**KORLÁT($i, q$): az írásbelin az $i$-edik témakörből legalább $q$ pontot elért jelentkezők számát határozza meg – konstans**
Többféle adatszerkezetet is felhasználunk. $FP$-t tároljuk egy keresőfában annyi módosítással, hogy az egyes értékekhez egy számlálót is rendelünk, amit pontazonos beszúrás esetén használunk értelemszerűen. Innen az $FP$-re vonatkozó műveletek egyértelműek, a lépésszámok is jók.

$P_i$ nyilvántartására a ládarendezéshez hasonlóan tartsunk nyilván egy $P_i[1, \ldots, 30]$ tömböt kezdetben 0-ra inicializálva. Ekkor egy $P_i$ pontos dolgozat beszúrásakor a megfelelő számlálót megnöveljük eggyel (konstans lépés). KORLÁT$(i, q)$ ekkor így számolható: $\sum_{j=1}^{q} P_i[j]$, és mivel $q \leq 30$, ez tényleg konstans lépés.

A kétféle adatszerkezt lépésszámait összevetve láthatjuk, hogy az előírt műveletek az előírt lépésszámokba belefférnek.

23. **Adott $2^k - 1$ különböző szám, mindegyik az $\{1, 2, \ldots, n\}$ halmazból, ezekből kell egy $k$ mélységű bináris keresőfát készíteni. Adjunk olyan algoritmust, amely ezt**

$c \cdot n$ **lépésben megcsinálja!**
$2^k - 1$ számot tudunk egy pontosan $k$ szintű teljes bináris keresőfában tárolni, így ilyet fogunk csinálni. Mivel a fa alakja adott, csak azt kell kitalálni, hogy melyik szám melyik csúcsba kerüljön. Viszont azt is tudjuk, hogy egy bináris keresőfát inorder bejárva a benne tárolt elemeket növekvő sorrendben kapjuk meg. Így meg tudjuk tenni, hogy felépítünk egy „biankó" $k$ szintű teljes bináris fát, futtatunk rá egy inorder bejárást, és a növekvően rendezett elemeinkből egy csúcs meglátogatásakor mindig odarakjuk a következőt. A számok különbözősége miatt $2^k - 1 \leq n$, így a fa felépítése $O(2^k - 1) = O(n)$, a számok ládarendezése (minden feltétel adott az alkalmazhatósághoz) $O(2^k - 1 + n) = O(n)$, majd a bejárás $O(2^k - 1) = O(n)$. Ezeket összegezve a lépésszám $O(n)$.