

# ALGORITMUSOK ÉS ADATSZERKEZETEK VIZSGA

## Tartalom

1. Függvények aszimptotikus viselkedése .....	2
2. Összehasonlító rendezések .....	3
2.1. Beszúró rendezés .....	5
2.2. Összefeszülő rendezés .....	8
2.3. Kupacrendezés .....	11
2.4. Gyorsrendezés .....	15
3. Absztrakt adattípusok .....	17
3.1. Verem .....	17
3.2. Sor .....	20
3.3. Prioritásos sor .....	22
4. Láncolt listák .....	25
4.1. Egyszerű listák (S1L) .....	25
4.2. Fejelemes listák (H1L) .....	26
4.3. Egyszerű kétirányú listák (S2L) .....	27
4.4. Fejelemes, kétirányú, ciklikus listák (C2L) .....	28
5. Bináris fák .....	29
5.1. Bináris keresőfák (és rendezőfák) .....	32
6. Rendezés lineáris időben .....	36
6.1. Edényrendezés a $[0;1]$ intervallumon (bucket sort) .....	36
6.2. Leszámláló rendezés (counting sort) .....	39
6.3. Radix rendezés leszámláló rendezéssel .....	40
6.4. Radix rendezés szétválogatással .....	41
7. Hasító táblák .....	43
7.1. Ütközés feloldása láncolással .....	43
7.2. Nyílt címzés .....	45

## 1. Függvények aszimptotikus viselkedése

1) Tegyük fel, hogy  $g : N \rightarrow R$ , aszimptotikusan nemnegatív függvény!

a. Adjuk meg az  $O(g)$  és az  $\Omega(g)$  függvényhalmazok definícióját!

8.4. Definíció. Az  $O(g)$  függvényhalmaz olyan  $f$  függvényekből áll, amiket elég nagy  $n$  helyettesítési értékekre, megfelelő pozitív konstans szorzóval fejlőről becsül a  $g$  függvény:

$$O(g) = \{f : \exists d \in \mathbb{P}, \text{ hogy elég nagy } n\text{-ekre } d * g(n) \geq f(n).\}$$

$f \in O(g)$  esetén azt mondjuk, hogy  $g$  aszimptotikus felső korlátja  $f$ -nek.

8.5. Definíció. Az  $\Omega(g)$  függvényhalmaz olyan  $f$  függvényekből áll, amiket elég nagy  $n$  helyettesítési értékekre, megfelelő pozitív konstans szorzóval alulról becsül a  $g$  függvény:

$$\Omega(g) = \{f : \exists c \in \mathbb{P}, \text{ hogy elég nagy } n\text{-ekre } c * g(n) \leq f(n).\}$$

$f \in \Omega(g)$  esetén azt mondjuk, hogy  $g$  aszimptotikus alsó korlátja  $f$ -nek.

b. Milyen alapvető összefüggést ismer az  $O(g)$ , az  $\Omega(g)$  és a  $\Theta(g)$  függvényhalmazok között?

8.6. Definíció.  $\Theta(g) = O(g) \cap \Omega(g)$

8.7. Következmény. A  $\Theta(g)$  függvényhalmaz olyan  $f$  függvényekből áll, amiket elég nagy  $n$  helyettesítési értékekre, megfelelő pozitív konstans szorzókkal alulról és felülről is becsül a  $g$  függvény:

$$\Theta(g) = \{f : \exists c, d \in \mathbb{P}, \text{ hogy elég nagy } n\text{-ekre} \\ c * g(n) \leq f(n) \leq d * g(n).\}$$

$f \in \Theta(g)$  esetén tehát azt mondhatjuk, hogy  $g$  aszimptotikus alsó és felső korlátja  $f$ -nek. (Ezt a 8.10 tulajdonságánál is láthatjuk.)

Arra, hogy egy függvény egy másikhoz képest nagy  $n$  értékekre elhanyagolható, bevezetjük az aszimptotikusan kisebb fogalmát.

c. Igaz-e, hogy  $(3n + 4)^2 \in \Theta(n^2)$ ? Miért?

d. Igaz-e, hogy  $n^n \in \Omega(2^n)$ ? Miért?

e. Igaz-e, hogy  $1000n^2(\lg n) \in O(n^3)$ ? Miért?

2) Tegyük fel, hogy  $g : N \rightarrow R$ , aszimptotikusan nemnegatív függvény!

a. Adjuk meg az  $O(g)$  és az  $\Omega(g)$  függvényhalmazok definícióját!

-> lásd 1) a. megoldás

b. Milyen alapvető összefüggést ismer az  $O(g)$ , az  $\Omega(g)$  és a  $\Theta(g)$  függvényhalmazok között?

-> lásd 1) b. megoldás

c. Igaz-e, hogy  $(2n + 1)(3n - 4) \in \Theta(n^2)$ ? Miért?

d. Igaz-e, hogy  $2^n \in O(n!)$ ? Miért?

e. Igaz-e, hogy  $(n^* \lg n) \in \Theta(n^2)$ ? Miért?

## 2. Összehasonlító rendezések

1)

a. Osztályozza az előadásról ismert négy összehasonlító rendezést műveletigény ( $MT(n)$ ,  $AT(n)$ ,  $mT(n)$ ) szempontjából!

	Kupac rendezés (heap sort)	Beszúró rendezés (insertion sort)	Összefésülő rendezés (merge sort)	Gyors rendezés (quick sort)
$mT(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n * \log n)$	$\Theta(n * \log n)$
$MT(n)$	$\Theta(n * \log n)$	$\Theta(n^2)$	$\Theta(n * \log n)$	$\Theta(n^2)$
$AT(n)$	$\Theta(n * \log n)$	$\Theta(n^2)$	$\Theta(n * \log n)$	$\Theta(n * \log n)$

b. Mondja ki az összehasonlító rendezések maximális műveletigényének alsó korlátjára vonatkozó alaptételt!

10.4. Tétel. Tetszőleges összehasonlító rendezésre  $MT(n) \in \Omega(n \log n)$ .

**Bizonyítás.** Mindegyik alprogram hívás és ciklusiteráció is csak korlátos számú kulcs összehasonlítást végez el (a belé ágyazott alprogram hívások és ciklusiterációk nélkül). Legyen ezeknek a korlátoknak a maximuma  $k$ .

Akkor  $MT(n) * k \geq MC(n) \Rightarrow MT(n) \geq \frac{1}{k}MC(n) \Rightarrow MT(n) \in \Omega(MC(n))$ . A 10.3. tétellel ( $MC(n) \in \Omega(n \log n)$ ) és az „ $\cdot \in \Omega(\cdot)$ ” reláció tranzitivitásával azonnal adódik ez a tételet ( $MT(n) \in \Omega(n \log n)$ ).  $\square$

c. Bizonyítsa be a kulcsösszehasonlítások számának maximumára vonatkozó lemmát!

10.3. Tétel. Bármiely összehasonlító rendezés végrehajtásához a legrosszabb esetben  $MC(n) \in \Omega(n \log n)$  kulcsösszehasonlítás szükséges.

**Bizonyítás.** A fenti eszmeffuttatás alapján elegendő alsó becslést adni a döntési fa  $h = MC(n)$  magasságára. Jelölje a levelei számát  $l$ , akkor  $n! \leq l$ , továbbá  $l \leq 2^h$ , ugyanis egy  $h$  magasságú bináris fának legfeljebb  $2^h$  levele van. Innét  $n! \leq l \leq 2^h$ , amiből  $n! \leq 2^h$  adódik.

Következésképpen

$$\begin{aligned} MC(n) &= h \geq \log n! = \sum_{i=1}^n \log i \geq \sum_{i=\lceil \frac{n}{2} \rceil}^n \log i \geq \sum_{i=\lceil \frac{n}{2} \rceil}^n \log \left\lceil \frac{n}{2} \right\rceil \geq \left\lceil \frac{n}{2} \right\rceil * \log \left\lceil \frac{n}{2} \right\rceil \geq \\ &\geq \frac{n}{2} * \log \frac{n}{2} = \frac{n}{2} * (\log n - \log 2) = \frac{n}{2} * (\log n - 1) = \frac{n}{2} \log n - \frac{n}{2} \in \Omega(n \log n) \end{aligned}$$

A  $\sum_{i=\lceil \frac{n}{2} \rceil}^n \log \left\lceil \frac{n}{2} \right\rceil \geq \left\lceil \frac{n}{2} \right\rceil * \log \left\lceil \frac{n}{2} \right\rceil$  egyenlőtlenséghez elegendő belátni, hogy az összegnek legalább  $\left\lceil \frac{n}{2} \right\rceil$  tagja van. A tagok száma nyilván  $n - (\lceil \frac{n}{2} \rceil - 1) = (n - \lceil \frac{n}{2} \rceil) + 1 = \lfloor \frac{n}{2} \rfloor + 1$ , ami, ha  $n$  páratlan szám, éppen  $\lceil \frac{n}{2} \rceil$ , ha pedig  $n$  páros, akkor  $\lceil \frac{n}{2} \rceil + 1$ .  $\square$

## d. Mi a jelentősége ennek a téTELNEk?

2)

- a. Osztályozza az előadásról ismert négy összehasonlító rendezést műveletigény ( $MT(n)$ ,  $AT(n)$ ,  $mT(n)$ ) szempontjából!

-> lásd 1) a. megoldás

- b. Mit tud mondani a rendezések minimális műveletigényének alsó korlátjáról? Miért?

**10.1. Tétel.** Tetszőleges rendező algoritmusra  $mT(n) \in \Omega(n)$ .

**Bizonyítás.** Világos, hogy tetszőleges helyes rendező algoritmusnak minden  $n$  rendezendő elem értékét ellenőriznie kell, és minden egyik alprogram hívás és ciklusiteráció is csak korlátos számú elemet ellenőriz (a belé ágyazott alprogram hívások és ciklusiterációk nélkül). Legyen ezeknek a korlátoknak a maximuma  $k$ . Akkor  $mT(n) * k \geq n \Rightarrow mT(n) \geq \frac{1}{k}n \Rightarrow mT(n) \in \Omega(n)$ .  $\square$

Legoptimálisabb esetben a sor rendezett, éppen ezért elég minden elemet egyszer megvizsgálni.

- c. Osztályozza az előadásról ismert négy összehasonlító tömbrendezést (maximális és minimális) tárigény szempontjából! (A tárigény = az algoritmus végrehajtásához használt temporális adatok számára + az alprogram hívások számára szükséges tárterület.)

Name	Best Case	Average Case	Worst Case	Memory	Stable	Method Used
Quick Sort	$n \log n$	$n \log n$	$n^2$	$\log n$	No	Partitioning
Merge Sort	$n \log n$	$n \log n$	$n \log n$	$n$	Yes	Merging
Heap Sort	$n \log n$	$n \log n$	$n \log n$	1	No	Selection
Insertion Sort	n	$n^2$	$n^2$	1	Yes	Insertion

Merge sort (összefésülő rendezés):

Tárigénye:  $O(n)$

Insertion sort (beszúró rendezés):

Tárigénye:  $O(1)$

Quick sort (gyorsrendezés):

Tárigénye:  $O(n * \log n)$

Heap sort (kupacrendezés):

Tárigénye:  $O(1)$

- d. A fenti négy rendezés közül melyeket javasolná egyirányú láncolt listák rendezésére is? Mit tud mondani ezen listarendezések tárigényéről?

Beszúró rendezést (insertion sort), valamint az összefésülő rendezés (merge sort). A merge sort tárigénye  $O(n)$ , míg az insertion sort tárigénye  $O(1)$ .

- e. A fenti négy rendezés közül melyiket javasolná előrendezett, kétirányú láncolt listák rendezésére? Miért?

A beszúró rendezést, mivel időkomplexitása  $O(n)$  előrendezett bemenet esetén, továbbá alapötlete az, hogy végigmegyünk a listán, összehasonlíthatunk minden elemet az előtte lévő elemekkel, majd beillesztjük a megfelelő helyre. Mivel a lista már előrendezett, kevesebb összehasonlításra és cserére van szükség, ami hatékony rendezést eredményez.

## 2.1. Beszúró rendezés

1)

a. Szemléltesse a beszúró rendezést az előadásról ismert módon az  $\langle 5; 7; 6; 4; 8; 4 \rangle$  tömbre!

Az utolsó beszúrást részletezzé!

Példa a beszúró rendezés lejátszására:

Gyakorlás:  
Insertion Sort.

24 9 2 10 19 28 24 12

beszűr: lépés: a kezdeti tömbökben elemet felosztjuk két részre; minden részben összesen 4 elem van.  
rendezett rész és egy rendezetlen rész.  
A rendezett rész balról fog növekedni és mindenben egy elemű lesz,  
amely az eredeti tömbökben a legelső elemű lesz.  
Etelenszerűen a rendezetlen részben a második elemtől kezdődően az összes többi elem a törlő utánig.

Lépések: minden egyes lépés igy fog kezdeni, hogy a rendezetlen részben utolsó elemet összehasonlítsa a rendezetlen részben elso elemmel.

1. lépés:  $24 | 9 2 10 19 28 24 12$  összehasonlítás  $\Rightarrow$  minél 24 > 9 =>  
1. mossa  $x=9$   $\rightarrow$  hozzáadja az x + vállalkozára  
 $\boxed{24} - 2 10 19 28 24 12$   
1. mossa  $\boxed{24} 2 10 19 28 24 12$   
1. mossa  $\boxed{9} \boxed{24} 2 10 19 28 24 12$

2. lépés:  $\boxed{9} \boxed{24} | 2 10 19 28 24 12$  összehasonlítás  $\Rightarrow$  minél 2 < 24 =>  
 $x=2$   
 $\boxed{9} \boxed{24} - 10 19 28 24 12$   
 $\boxed{9} \boxed{24} 10 19 28 24 12$   
 $- \boxed{9} \boxed{24} 10 19 28 24 12$   
 $\boxed{2} \boxed{9} \boxed{24} 10 19 28 24 12$

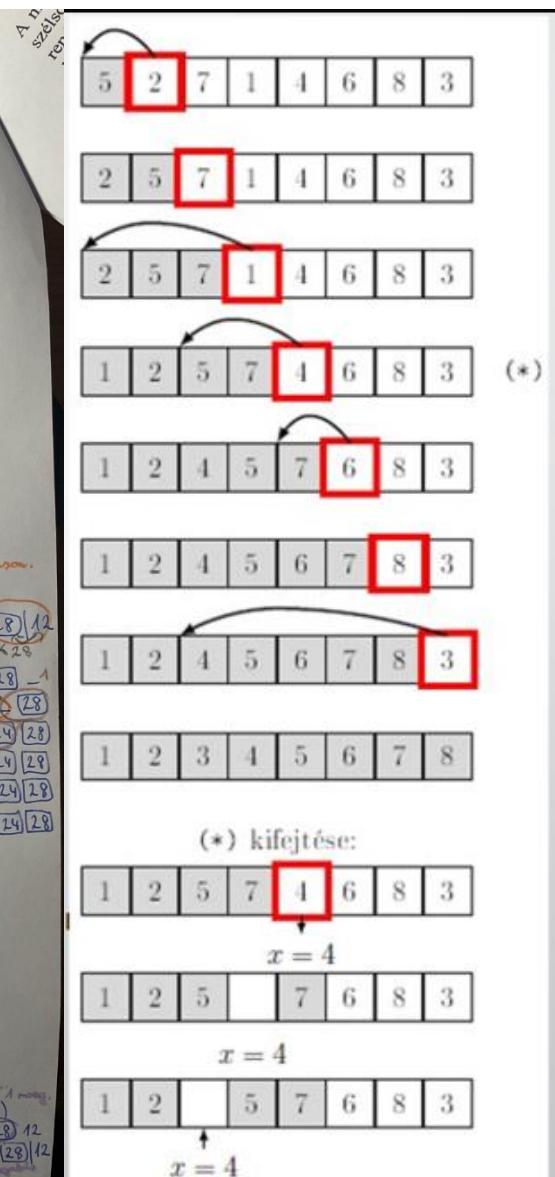
3. lépés:  $\boxed{2} \boxed{9} \boxed{24} | 10 19 28 24 12$  összehasonlítás  $\Rightarrow$  minél 10 > 24 =>  
 $x=10$   
 $\boxed{2} \boxed{9} \boxed{24} - 19 28 24 12$   
 $\boxed{2} \boxed{9} - \boxed{24} 19 28 24 12$   
 $\boxed{2} \boxed{9} \boxed{10} \boxed{24} | 19 28 24 12$

4. lépés:  $\boxed{2} \boxed{9} \boxed{10} \boxed{24} | 19 28 24 12$  összehasonlítás  $\Rightarrow$  minél 19 < 24 =>  
 $x=19$   
 $\boxed{2} \boxed{9} \boxed{10} \boxed{24} - 28 24 12$   
 $\boxed{2} \boxed{9} \boxed{10} - \boxed{24} 28 24 12$   
 $\boxed{2} \boxed{9} \boxed{10} \boxed{19} \boxed{24} | 28 24 12$

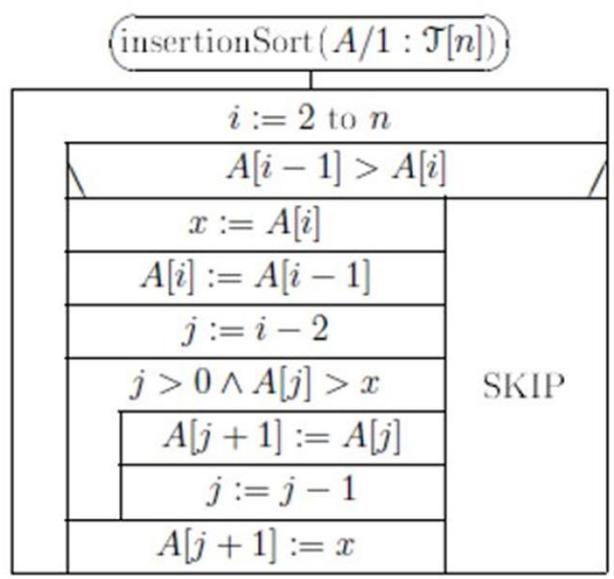
5. lépés:  $\boxed{2} \boxed{9} \boxed{10} \boxed{19} \boxed{24} | 28 12$  összehasonlítás  $\Rightarrow$  minél 28 > 24 =>  
a rendezetlen részben minden elemet követhetjük 1-vel.  
rendezetlen részben minden elemet követhetjük 1-vel (mossatás nem kötött)  
 $\boxed{2} \boxed{9} \boxed{10} \boxed{19} \boxed{24} - \boxed{28} 12$   
 $\boxed{2} \boxed{9} \boxed{10} \boxed{19} \boxed{24} - \boxed{28} - 12$

6. lépés:  $\boxed{2} \boxed{9} \boxed{10} \boxed{19} \boxed{24} \boxed{28} | 12$  összehasonlítás  $\Rightarrow$  minden elemet követhetjük 1-vel (mossatás nem kötött)  
 $\boxed{2} \boxed{9} \boxed{10} \boxed{19} \boxed{24} \boxed{28} - 12$

Címzetősítés: 5  
Mossatás: 6



b. Adja meg a megfelelő struktogramot!



c. Számolja ki a struktogramjának megfelelő pontos  $MT(n)$  és  $mT(n)$  értékeit!

$$mT(n) = n, MT(n) = (\frac{1}{2} * n^2) - (\frac{1}{2} * n) + 1$$

d. Mit jelentenek ezek az eredmények aszimptotikusan?

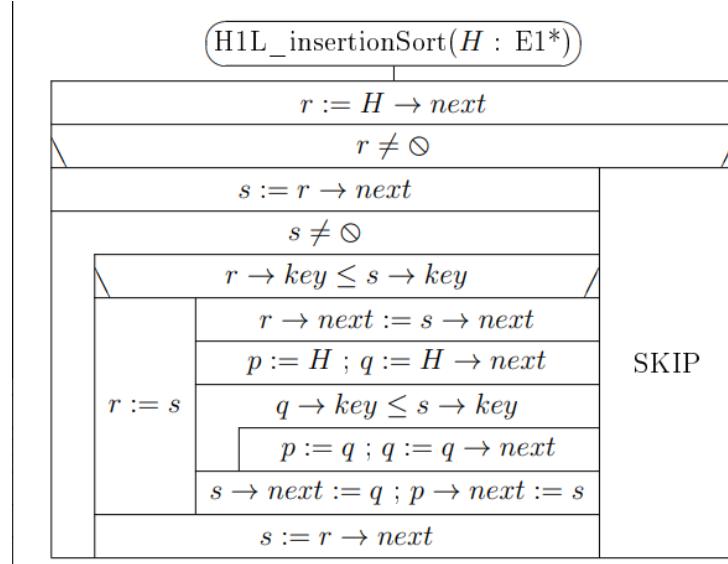
A legjobb eset:  $mT(n) \in \Theta(n)$ ; a legrosszabb eset:  $MT(n) \in \Theta(n^2)$

2)

a. Mikor nevezünk egy rendezést stabilnak?

Egy rendezés akkor stabil, ha megtartja az ekvivalens kulcsú elemek eredeti sorrendjét.

b. Adja meg fejelemes, egyirányú, nemciklikus listára (H1L) a beszúró rendezés struktogramját! A listát a key mezők szerint monoton növekvően kell rendezni. (A listaelémeknek a szokásos key és next mezőkön kívül más részei is lehetnek, de ezeket nem ismerjük.)



c. Hogyan biztosítottuk a fenti rendezés stabilitását?

A láncoolt lista rendezett szakaszába az újabb elemeket jobbról balra szúrjuk be, és a beszúrandóval egyenlő kulcsú elemeket már nem lépjük át (az algoritmusból kiolvasható); egyenlőség esetén tovább lépünk, és nem változtatunk az elemek sorrendjén.

d. Mekkora lesz a rendezés minimális és maximális műveletigénye? Miért?

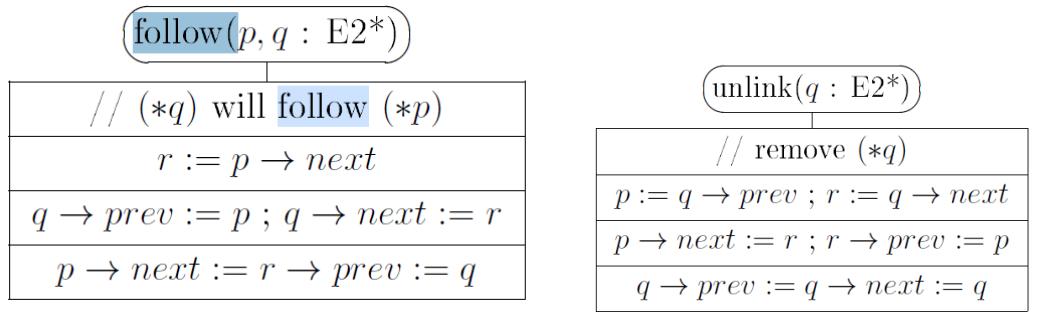
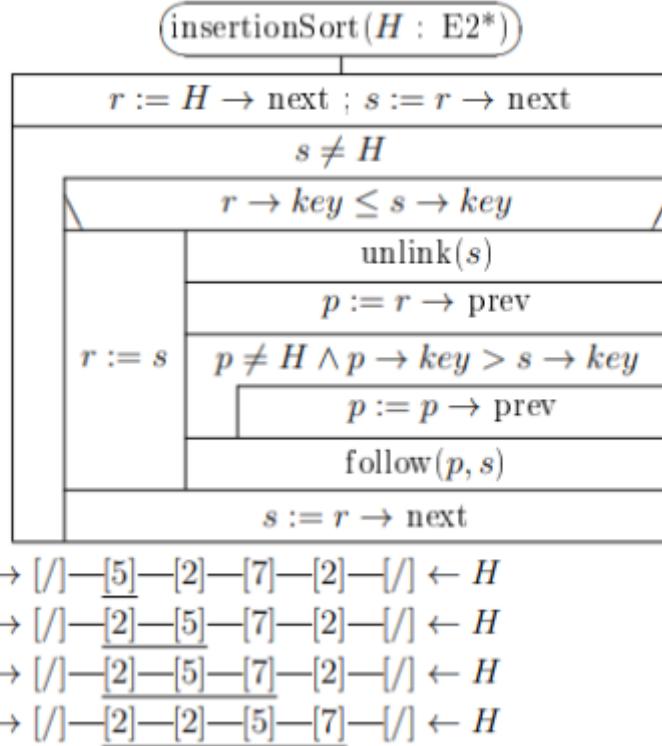
$$mT(n) \in \Theta(n); MT(n) \in \Theta(n^2)$$

3)

a. Mikor nevezünk egy rendezést stabilnak?

-> lásd 2) a. megoldás

- b. Adja meg fejelemes, kétirányú, ciklikus listára (C2L) a beszúró rendezés struktogramját! A listát a key mezők szerint monoton növekvően kell rendezni. (A listaelemeknek a szokásos key és a két mutató mezőn kívül járulékos mezői is lehetnek, de ezeket nem ismerjük. A listát kizárólag az `unlink(q)`, `precede(q, r)` és `follow(p, q)` eljárásokkal szabad módosítani, ahogy azt az előadáson tanultuk.)



- c. Hogyan biztosítottuk a fenti rendezés stabilitását?

Egyenlőség esetén tovább lépünk, és nem változtatunk a sorrenden.

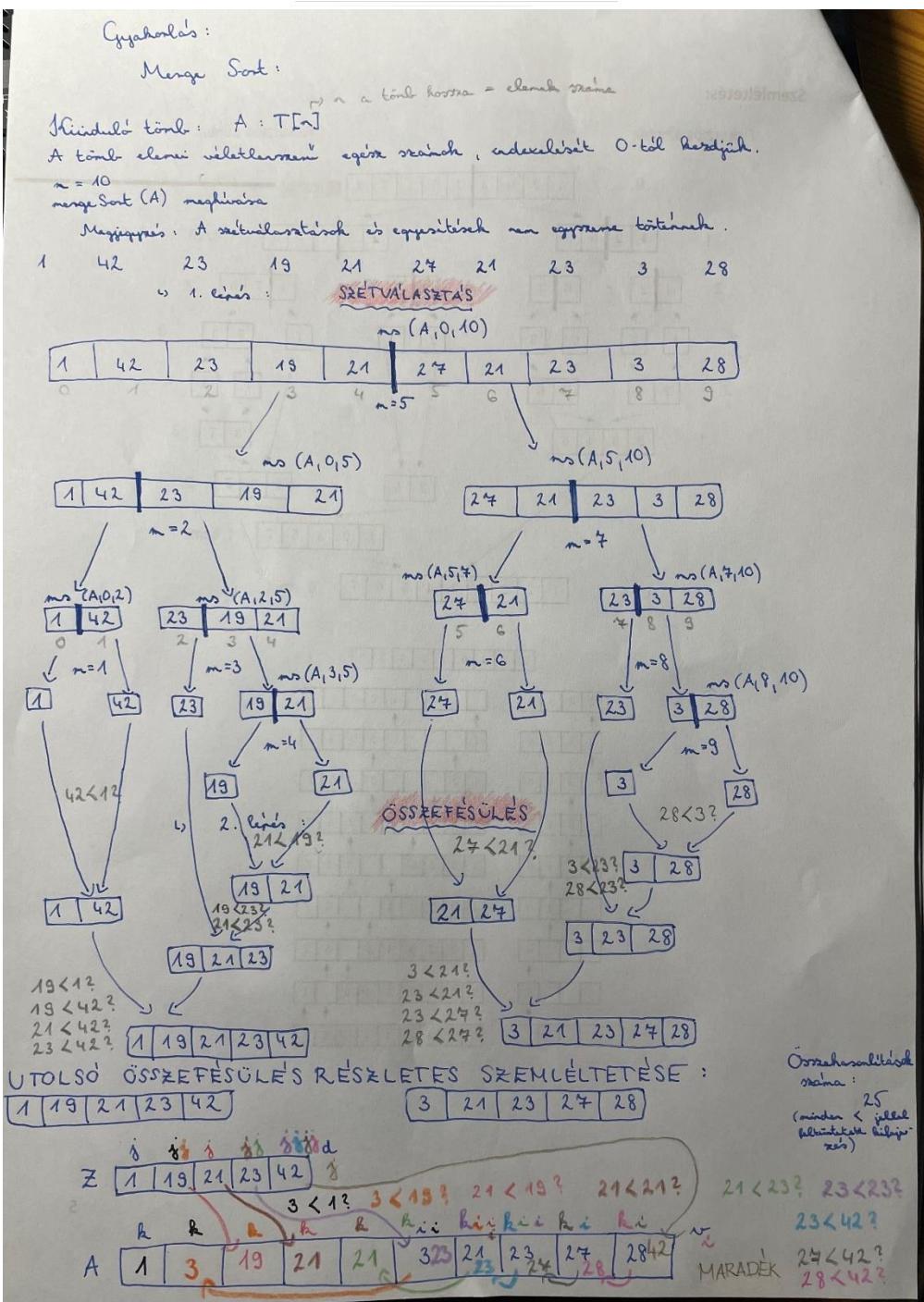
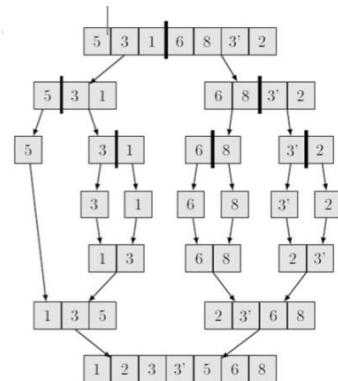
- d. Mekkora lesz a rendezés minimális és maximális műveletigénye? Miért?  
 $mT(n) \in \Theta(n)$ ;  $MT(n) \in \Theta(n^2)$

## 2.2. Összefésülő rendezés

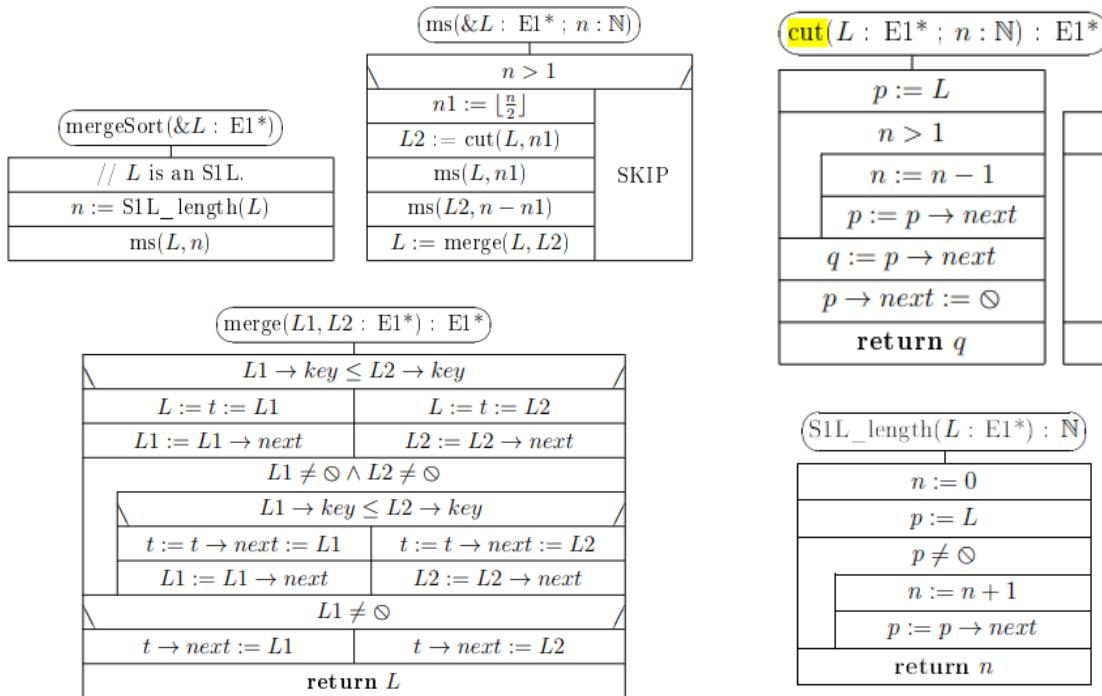
1)

- a. Szemléltesse az összefésülő rendezés (merge sort) működését az előadásról ismert módon az  $\langle 4; 3; 5; 2; 1; 8; 3 \rangle$  sorozatra! (Sem a szétyágásokat, sem az összefuttatásokat nem kell részletezni.)

Példa az összefésülő rendezés lejátszására:



b. Adjuk meg egyszerű láncolt listákra a rekurzív eljárás és a „cut” függvény struktogramját!



c. Mekkora a rendezés műveletigénye? Röviden indokoljuk állításunkat! (Csak a bizonyítás yálatát kell leírni.)

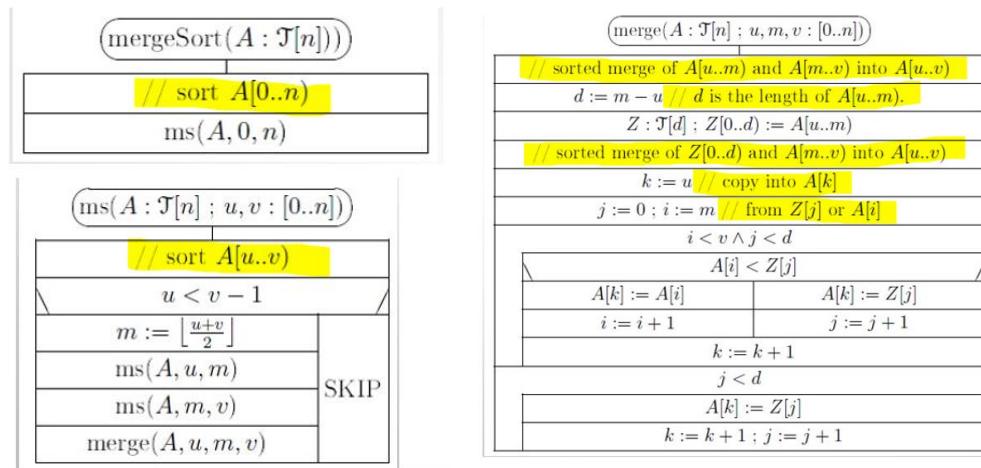
$$mT(n) \in \Theta(n * \log n); MT(n) \in \Theta(n * \log n)$$

## A merge sort műveletigénye: szemléletes megközelítés

- $MT_{MS}(n), mT_{MS}(n) \in \Theta(n \log n)$
- Bizonyítása szemléletesen: (matematikai biz. később)
  - A műveletek túlnyomó részét a merge eljárás végzi el
    - $mT_{merge}(l), MT_{merge}(l) \in \Theta(l)$  (ahol l az aktuális résztömb hossza ( $l = v - u$ ))
    - A rekurzív  $ms(A, u, v)$  eljárás minden hívásban felezi a rendezendő résztömb hosszát
  - A rekúzióknak kb.  $\log n + 1$  szintje van
  - minden rekúziós szinten: a merge hívások résztömbjei együtt lefedik az egész A tömböt
    - Kivétel az alsó egy vagy két szint: kevesebb
  - Egy tetszőleges szint összes merge hívásának műveletigényét összeadva:  $\Theta(n)$
  - A szintenkénti műveletigényt a szintek számával szorozva nagyságrendben  $\Theta(n \log n)$

2)

- a. Adja meg vektorokra a mergeSort(A) és segédeljárásai struktogramjait!



- b. Mekkora lesz a műveletigénye és a tárigénye? Miért? (Csak vázlatos indoklást kérünk.)

-> lásd 1) c. megoldás

3)

- a. Szemléltessük az összefésülő rendezés (merge sort) működését az előadásról ismert módon az  $<5; 3; 2; 1; 4; 9; 2>$  sorozatra!

-> lásd 1) a. megoldás

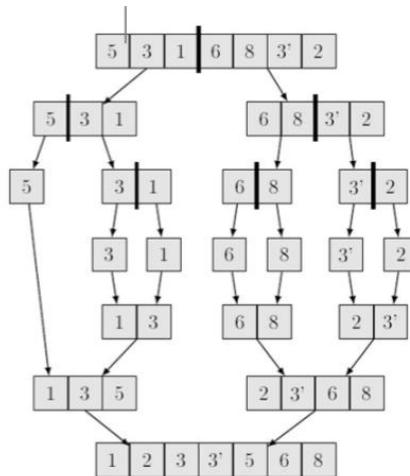
- b. Adjuk meg egyszerű láncolt listákra az összefésülő rendezés (merge sort) merge(L; L2) eljárásának struktogramját!

-> lásd 1) b. megoldás

- c. Mekkora a merge(L; L2) eljárás műveletigénye? Miért?  
 $mT(n) \in \Theta(n * \log n)$ ;  $MT(n) \in \Theta(n * \log n)$

4)

- a. Szemléltesse az összefésülő rendezés (merge sort) működését az előadásról ismert módon az  $<1; 9; 2; 5; 3; 2; 4>$  sorozatra! (Sem a szétyágásokat, sem az összefuttatásokat nem kell részletezni.)



- b. Adja meg egyszerű láncolt listákra az összefésülő rendezés (merge sort) főeljárásának és rekurzív eljárásának struktogramját!

-> lásd 1) b. megoldás

c. Mekkora a rendezés műveletigénye? Tárigénye?

$mT(n) \in \Theta(n * \log n)$ ;  $MT(n) \in \Theta(n * \log n)$

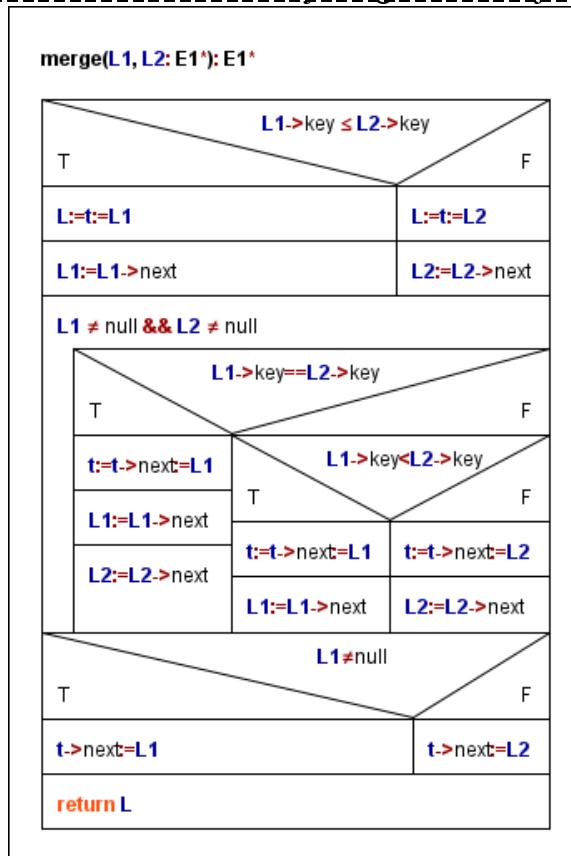
Tárigénye:  $O(n)$

5)

a. Szemléltesse az összefésülő rendezés (mergesort) működését az előadásról ismert módon a  $\{4; 5; 2; 3; 5; 7; 3; 9; 4\}$  sorozatra! (Az utolsó összefésülésnél azt is jelezze, hogy az input elemei milyen sorrendben kerülnek az outputra!)

-> lásd 1) a. megoldás

b. Egyszerű láncolt listákra a fenti rendezést úgy szeretnénk módosítani, hogy az azonos kulcsú elemekből csak egy-egy legyen az eredmény listán, így az szigorúan monoton növekvő legyen. A mergesort melyik eljárását kell módosítanunk? Adja meg ennek az új struktogramját!



c. Mit tud mondani az újfajta összefésülő rendezés műveletigényéről és tárigényéről?

$mT(n) \in \Theta(n * \log n)$ ;  $MT(n) \in \Theta(n * \log n)$ . Tárigénye kisebb, mint az eredeti  $O(n)$ .

## 2.3. Kupacrendezés

1)

a. Egy bináris fa mikor szigorúan bináris? Mikor teljes? Mikor majdnem teljes? Ez utóbbi mikor balra tömörített, és mikor kupac?

**Szigorúan bináris fa:** Azok a bináris fák, amelyekben minden belső (azaz nem-levél) csúcsnak két gyereke van.

**Teljes bináris fa:** Olyan szigorúan bináris fa, amelynek minden levele azonos szinten helyezkedik el.

- $h$  mélységű teljes bináris fa csúcsainak száma:  $1+2+4+\dots+2^h = 2^{h+1}-1$

**Majdnem teljes bináris fa:** Olyan teljes bináris fa, amelynek legalsó levélszintjéről elhagytunk néhány levelet (de nem az összeset).

- $h$  mélységű majdnem teljes bináris fa csúcsainak száma:  $n \in 2^h..2^{h+1}-1$ , és így  $h = \lfloor \log n \rfloor$

- Az alsó szinten levő leveleket elvéve egy  $h - 1$  mélységű teljes bináris fát kapunk

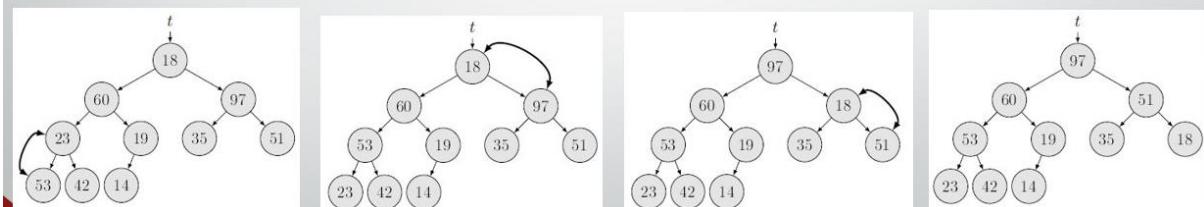
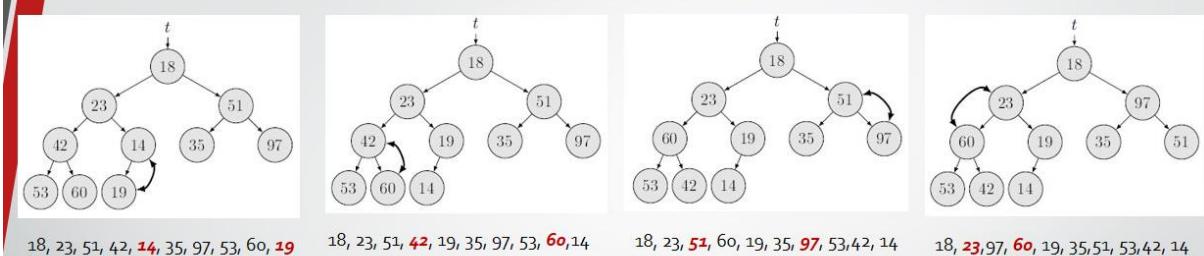
Egy **majdnem teljes balra tömörített bináris fa**: majdnem teljes bináris fa, de levelek csak a legalsó szintről, a jobb oldalról hiányozhatnak. Ezeket szintfolytonos fáknak is nevezzük.

**Kupac:** Maximum kupac: egy majdnem teljes, balra tömörített bináris fa, amelynek minden belső pontjára teljesül, hogy a belső pont kulcsa nagyobb vagy egyenlő a gyerekei kulcsánál. Így kupac tetején (a fa gyökerében) minden az egyik legnagyobb elem található.

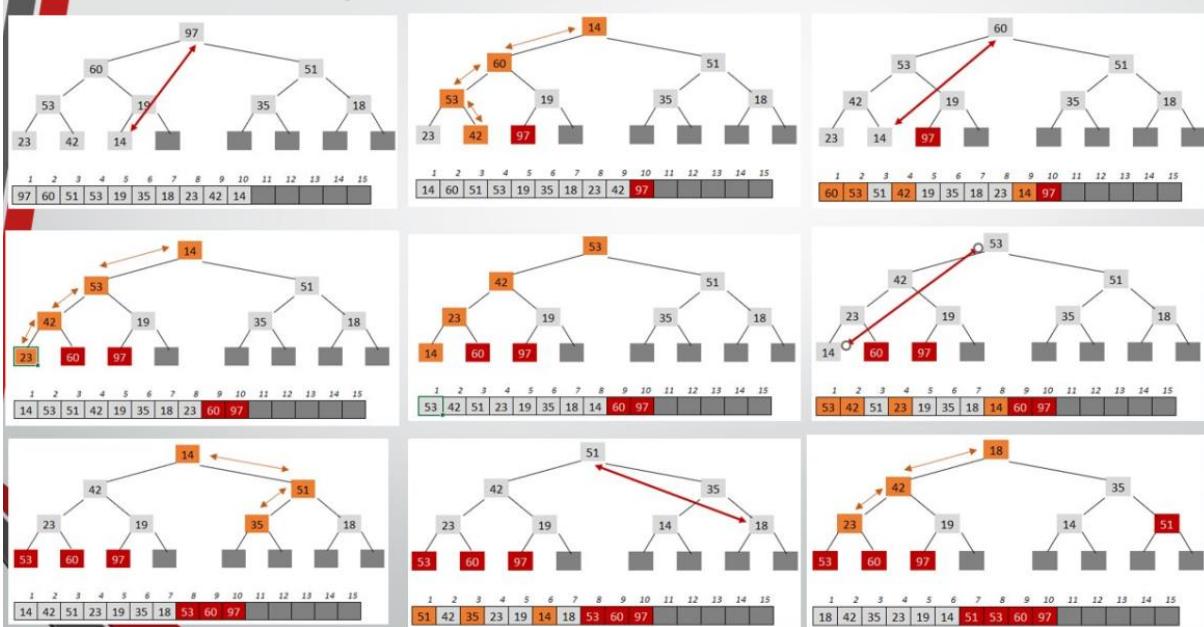
• **Minimum kupac hasonlóan:** a szülő kulcs kisebb vagy egyenlő a gyerekei kulcsánál.

b. Szemléltesse a kupacrendezést a következő tömbre! -  $<3; 9; 8; 2; 4; 6; 7; 5>$  - minden lesüllyesztés előtt jelölje a csúcs mellett egy kis körbe tett sorszámmal, hogy ez a rendezés során a hányadik lesüllyesztés; akkor is, ha az aktuális lesüllyesztés nem mozdítja el a csúcsban lévő kulcsot! minden valódi lesüllyesztés előtt jelölje a lesüllyesztés irányát és útvonalát! minden olyan lesüllyesztés előtt rajzolja újra a fát, ami az aktuális ábrán már módosított csúcsokat érinti! Újrarajzoláskor adja meg a fát tartalmazó tömb pillanatnyi állapotát is! Elég a kupaccá alakítást és még utána a fő ciklus első 3 iterációját (a 3. lesüllyesztés végéig) szemléltetni.

## Kupaccá alakítás szemléltetése



## Kupacrendezés szemléltetése



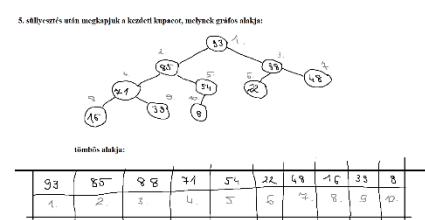
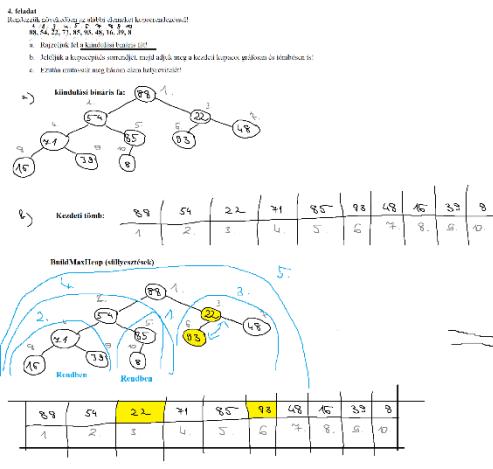
2)

a. Egy bináris fa mikor szigorúan bináris? Mikor teljes? Mikor majdnem teljes? Ez utóbbi mikor balra tömörített, és mikor kupac?

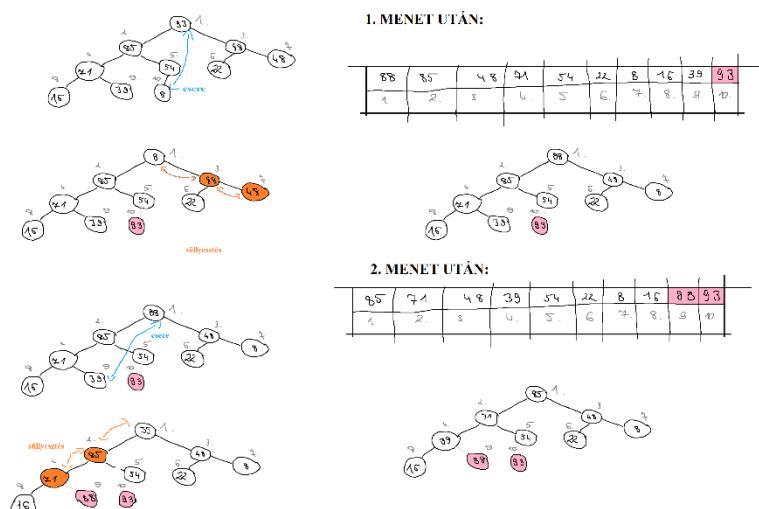
-> lásd 1) a. megoldás

b. Szemléltesse a kupacrendezést a következő tömbre! <5; 7; 6; 4; 2; 8; 9; 4; 3> minden lesüllyesztés előtt jelölje a csúcs mellett egy kis körbe tett sorszámmal, hogy ez a rendezés során a hányadik lesüllyesztés; akkor is, ha az aktuális lesüllyesztés nem mozdítja el a csúcsban lévő kulcsot! minden valódi lesüllyesztés előtt jelölje a lesüllyesztés irányát és útvonalát! minden olyan lesüllyesztés előtt rajzolja újra a fát, ami az aktuális ábrán már módosított csúcsokat érinti! Újrarajzoláskor adja meg a fát tartalmazó tömb pillanatnyi állapotát is! Elég a kupaccá alakítást és még utána a fő ciklus első 3 iterációját (a 3. lesüllyesztés végéig) szemléltetni!

Példa kupacrendezés szemléltetésére:

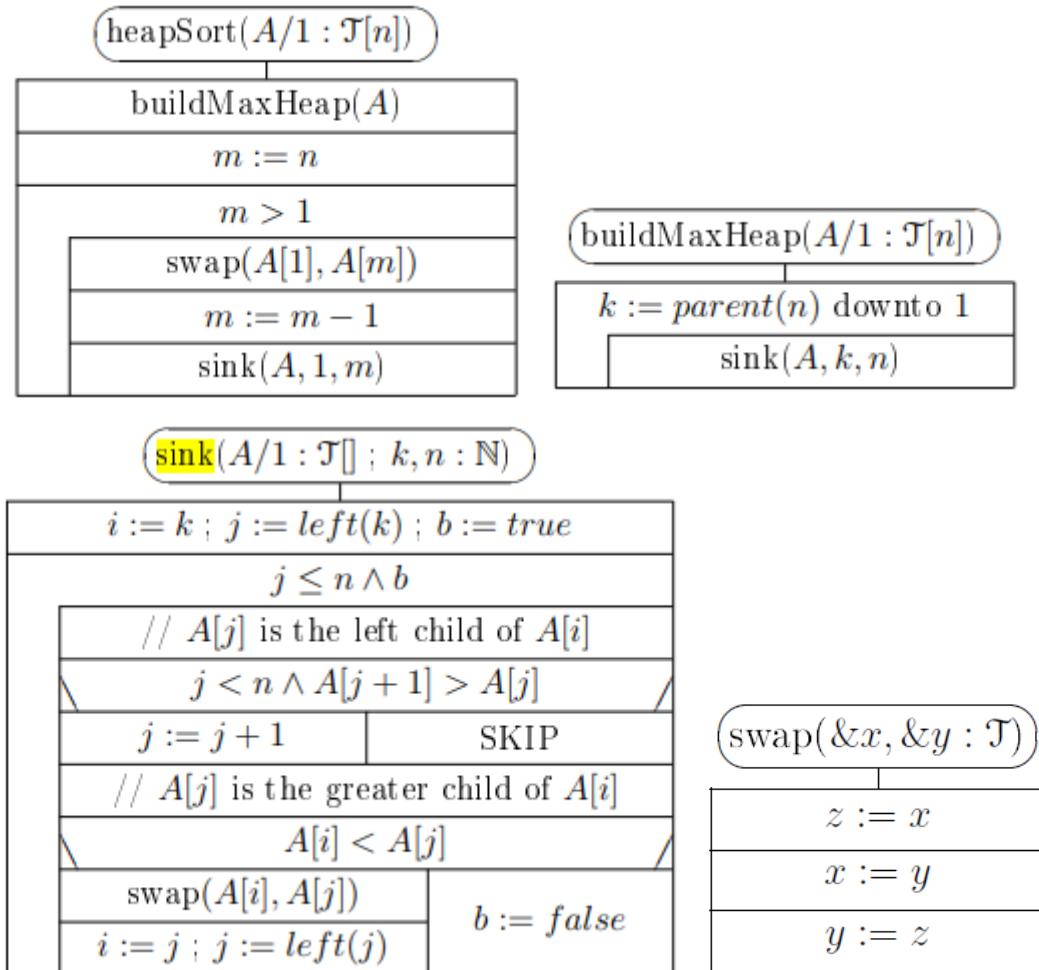


c) } elém helyeztelen



3)

- a. Adja meg a `heapSort(A: T[])` és segédeljárásai struktogramjait!



- b. Igaz-e, hogy  $MT(n) \in \Theta(n * \log n)$ ? Miért? [Vegyük észre, hogy az indokláshoz elegendő, ha a kupaccá alakítás és az utána következő rész műveletigényére is durva felső becslést adunk, továbbá használjuk az összehasonlító rendezések (alsókorlát-elemzésre vonatkozó) alaptételét!]

Igaz, mivel:

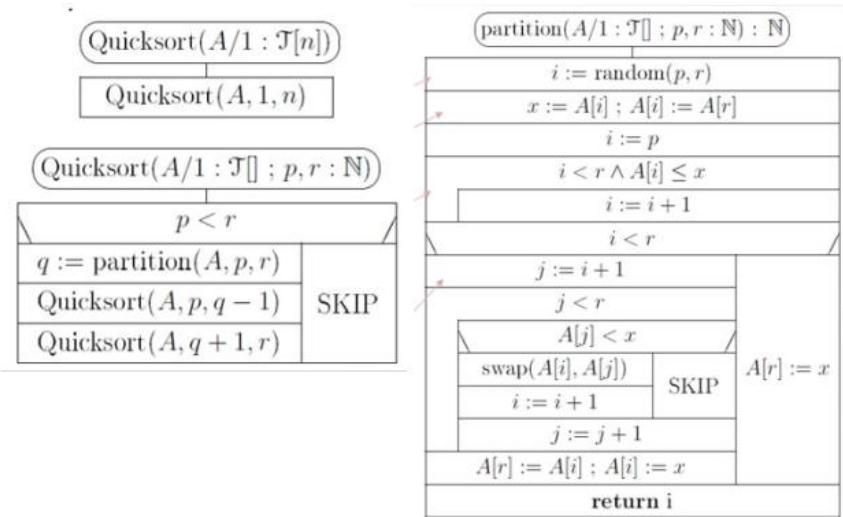
#### Műveletigény:

- A `buildMaxHeap()` műveletigénye:  $\Theta(n)$
- A rendezés többi részének műveletigénye:  $O(\log n)$
- Így a teljes rendezés műveletigénye:  $MT(n) \in \Theta(n * \log n)$ , mivel bármely összehasonlító algoritmusnak a legrosszabb esetben  $\Omega(n * \log n)$  összehasonlításra van szüksége  $n$  elem rendezéséhez.

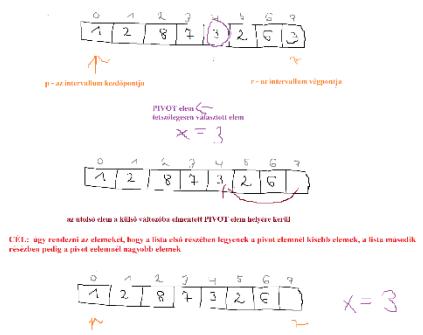
## 2.4. Gyorsrendezés

1)

a. Írja le az előadásról ismert formában a gyorsrendezés (quicksort) struktogramjait!



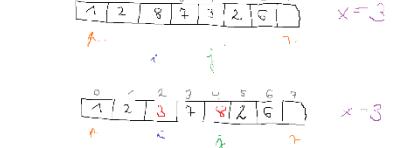
b. Szemléltesse a program „partition” függvényének működését a következő vektorra!  $\leq 3; 4; 8; 7; 1; 2; 6; 4;$



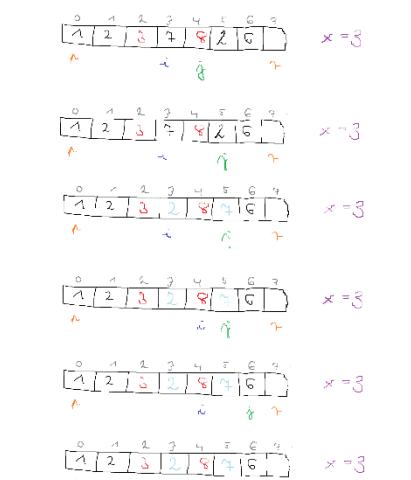
CÉL: meg rendezni az elemeket, hogy a lista első részén legyenek a pivot elemmel közeli elemek, a lista második részében pedig a pivot elemmel távoli elemek.



MEGOLDÁS: Egy i változóval indulunk, a lista elejéről, és keressük az olyan elemet, ami magasabb, mint a pivot elem. Ha azt megtalálunk, megcímlik, és az i-telleg egy j változóba kereshük a pivot elemmel egy kisebb elemet. Ha azt megtalálunk, ezt a pivot elemmel cseréljük le helyükön a j helyen elhelyezett elemmel.



Ezután az i-t követőjük legyél, és a j-val pedig közevünk továbbra is a pivot elemmel távoli elemeket.



Amint a j eléri az i-t, több vegyér a rendezni kívánt intervallumra, az így olyan helyre mutat, amely nem csatlakozik a többi elemhez.

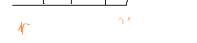
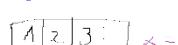
Amit az i mutat, hivatalosan nyugodtan az intervallum vége (a jobb), ami az n-nel van).



Ezután látunk, hogy az i egy olyan helyre mutat, ahova a PIVOT elem hivatalosan (hivatalosan elhelyezett helyen) már kihelyezett, minden másik kíváncsúgyan elhelyezett elemenél nagyobb.



Pihenés a rendezést hivatalos vége a fixált elemmel halálos leírás.



A 2-től is fizikaijuk.

Egy elemet listák rendszerezzük számlában, így ezekkel nem kell foglalkozunk.

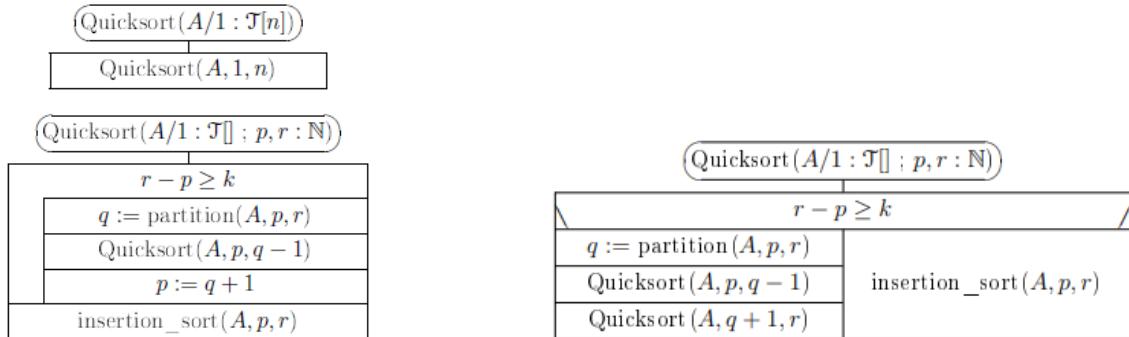
c. Mekkora a gyorsrendezés műveletigénye?

$$AT(n) \in \Theta(n * \log n)$$

d. Érdemes-e a gyorsrendezést és a beszúró rendezést egyetlen rendezésben egyesíteni? Hogyan? Miért?

Ismert, hogy legfeljebb néhányszor tíz rendezendő elem esetén a beszúró rendezés hatékonyabb, mint a gyors rendezések (merge sort, heap sort, quicksort). Ezért pl. a Quicksort(A, p, r) eljárás jelentősen gyorsítható, ha a kis méretű résztömbökönél áttérünk beszúró rendezésre. Mivel a szétvágások (partitions) során sok kicsi résztömb áll elő, így ezzel a program futása sok ponton gyorsítható:

A gyorsrendezés végerekurzió-optimalizált változata\*



Itt  $k \in \mathbb{N}$  konstans. Optimális értéke sok tényezőtől függ, de általában 20 és 40 között mozog.

2)

a. Írjuk le az előadásról ismert formában a gyorsrendezés (quicksort) struktogramjait!

-> lásd 1) a. megoldás

b. Szemléltessük a program „partition” függvényének működését a következő vektorra!  
≤1; 2; 8; 7; 3; 2; 6; 3≥.

lásd 1) b. megoldás

c. Mekkora a gyorsrendezés műveletigénye?

$$AT(n) \in \Theta(n * \log n)$$

d. Mekkora munkatárat igényel a gyorsrendezés (quicksort) alapváltozata a legjobb és a legrosszabb esetben? Miért?

A legjobb esetben:  $mT(n) \in \Theta(n * \log n)$ , a legrosszabb esetben:  $MT(n) \in \Theta(n^2)$ . A szétvágás (partition) műveletigénye lineáris, hiszen a két ciklus együttes  $r - p - 1$  vagy  $r - p$  iterációt végez. A quicksort műveletigényére ebből a következő adódik. A várható vagy átlagos műveletigény aszimptotikusan a legjobb esethez esik közel, és a legrosszabb eset valószínűsége nagyon kicsi.

3)

Tervezze újra a quicksort eljárást egyszerű láncolt listákra! Adja meg a szükséges struktogramokat, ha a szétvágásnál a rendezendő lista első eleme a tengely (pivot)! Vegye figyelembe, hogy a listaelemekben – a key mezőn és a következő listaelemre mutató mezőn kívül – lehetnek számunkra ismeretlen, járulékos mezők is!

### 3. Absztrakt adattípusok

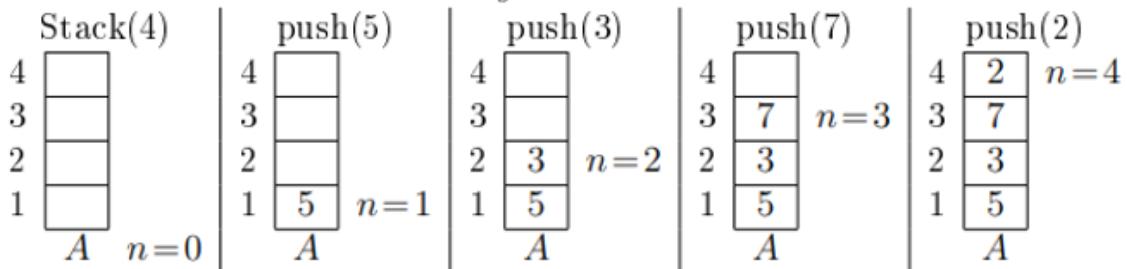
#### 3.1. Verem

1)

Adja meg az előadásról ismert módon a **Stack** osztály – tömbös reprezentációra alapozott – leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje?

Stack	
- $A/1 : \mathcal{T}[]$ // $\mathcal{T}$ is some known type ; $A.length$ is the physical	
- constant $m0 : \mathbb{N}_+ := 16$ // size of the stack, its default is $m0$ .	
- $n : \mathbb{N}$ // $n \in 0..A.length$ is the actual size of the stack	
+ Stack( $m : \mathbb{N}_+ := m0$ ) { $A := \text{new } \mathcal{T}[m]$ ; $n := 0$ } // create empty stack	
+ ~ Stack() { delete $A$ }	
+ push( $x : \mathcal{T}$ ) // push $x$ onto the top of the stack	
+ pop() : $\mathcal{T}$ // remove and return the top element of the stack	
+ top() : $\mathcal{T}$ // return the top element of the stack	
+ isEmpty() : $\mathbb{B}$ { return $n = 0$ }	
+ setEmpty() { $n := 0$ } // reinitialize the stack	

Néhány veremművelet



Stack::push( $x : \mathcal{T}$ )

doubleFullArray(& $A/1 : \mathcal{T}[]$ )

$n = A.length$	
doubleFullArray( $A$ )	SKIP
$n ++$	
$A[n] := x$	

$B/1 : \mathcal{T}[] := \text{new } \mathcal{T}[2 * A.length]$
$i := 1$ to $A.length$
$B[i] := A[i]$
delete $A$ ; $A := B$

Stack::pop(): $\mathcal{T}$

Stack::top(): $\mathcal{T}$

$n > 0$	
$n --$	
return $A[n + 1]$	emptyStackError

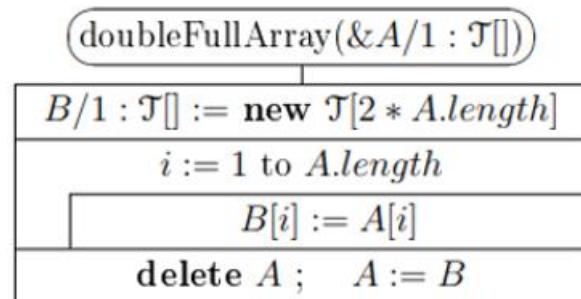
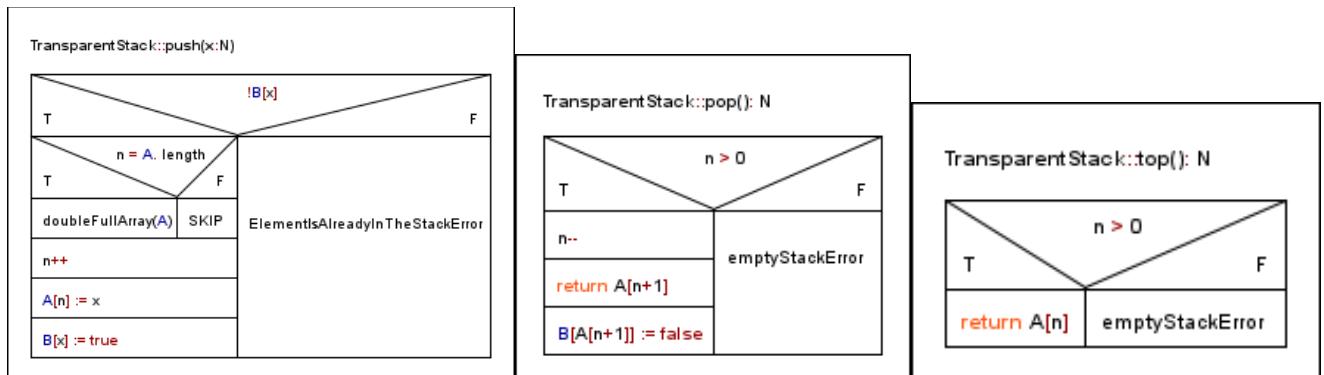
$n > 0$	
return $A[n]$	emptyStackError

A vermek műveleteit – a push művelet kivételével – egyszerű, rekurziót és ciklust nem tartalmazó metódusokkal írtuk le. Ezért mindegyik műveletigénye  $\Theta(1)$ , ami – legalábbis együtt az összes elvégzett különféle művelet átlagos műveletigényét tekintve – alapkötetelmény minden verem megvalósítással kapcsolatban. A push műveletre nyilván  $mT(n) \in \Theta(1)$  és  $MT(n) \in \Theta(n)$ .

2)

A **TransparentStack** adattípus műveletei a szokásos verem műveletek, de a `v.push(x)` csak akkor teszi `x`-et `v`-be, ha éppen nincs benne (ha benne van, nem csinál semmit). A **TransparentStack** elemtípusa az `1..k` egész intervallum típus, és valószínű **TransparentStack**néretű vermekre van szükségünk, ahol `n` és `k` pozitív egész számok. Adjuk meg az előadásról ismerthez hasonló módon a fenti típust megvalósító osztályt, a metódusok struktogram szintű leírásával együtt! A konstruktur műveletigénye  $O(k)$ , a többi metódus műveletigénye  $\Theta(1)$  legyen! {Használhatunk egy `b[1..k]` logikai tömböt „`b[x] = x` a veremben van” jelentéssel, ahol `x ∈ 1..k`.}

<ul style="list-style-type: none"> <li>- <code>A/1 : T[]</code> // <math>T</math> is some known type ; <code>A.length</code> is the physical</li> <li>- <code>B /1 : B[]</code> / <math>B</math> is Boolean type (true or false)      <code>B.length</code> is the physical</li> <li>- constant <code>m0 : N<sub>+</sub> := 16</code> // size of the stack, its default is <code>m0</code>.</li> <li>- <code>n : N</code> // <math>n \in 0..A.length</math> is the actual size of the stack      <math>\hookrightarrow : \mathbb{N}</math></li> </ul>
<ul style="list-style-type: none"> <li>+ <code>Stack(m : N<sub>+</sub> := m0) {A := new T[m] ; n := 0}</code> // create empty stack</li> <li>+ <code>~ Stack() { delete A } B := new B[n] ; s := 0</code></li> <li>+ <code>push(x : N) // push <code>x</code> onto the top of the stack</code></li> <li>+ <code>pop() : N // remove and return the top element of the stack</code></li> <li>+ <code>top() : N // return the top element of the stack</code></li> <li>+ <code>isEmpty() : B {return n = 0}</code></li> <li>+ <code>setEmpty() {n := 0} // reinitialize the stack</code></li> </ul>



3)

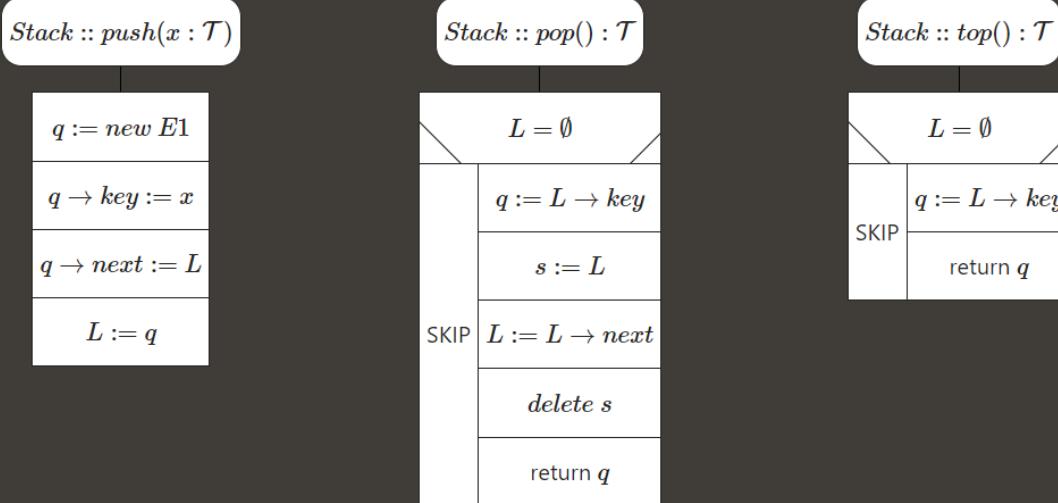
Adja meg az előadásról ismert **Stack** osztály – egyszerű láncolt listás reprezentációra alapozott – leírását, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! Mekkora az egyes műveletek aszimptotikus futási ideje?

Láncolt ábrázolás:

A verem láncolt reprezentációjában a  $L$  pointer nem csak az adattípushoz biztosít hozzáférést, hanem egyben a verem felső elemére mutat, ugyanis minden a láncolt lista elejére szúrja be az új elemet. Ezért nem kell külön bevezetni egy felső elemre mutató pointert és ezért elég lesz az  $S1L$ .

Stack
- $L : E1^*$
+ $Stack() \{ L := \emptyset \}$
+ $push(x : T)$
+ $pop() : T$
+ $top() : T$
+ $isEmpty() : B \{ return L = \emptyset \}$
+ $setEmpty() \{ L := \emptyset \}$

A láncolt ábrázolású verem műveletei között nem szerepel az  $IsFull()$  művelet, mivel ebben a reprezentációban nem számunk a tárolókapacitás gyakorlati felső korlátjával. A destruktur futási ideje, így  $\Theta(n)$ .



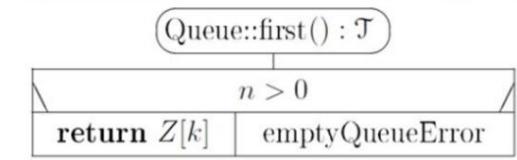
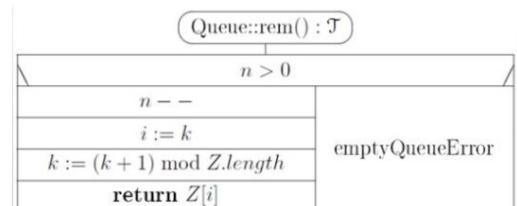
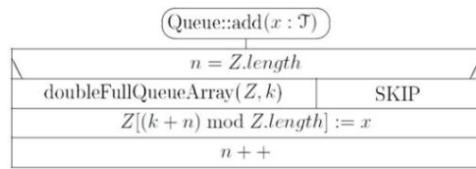
A  $push()$  a lista elejére szúrja be a paraméterként kapott elemet. A  $pop()$  visszaadja a lista első elemét és ki is törli a lista elejéről. A  $top()$  visszaadja a lista elején lévő elemet.

### 3.2. Sor

1)

Adja meg az előadásról ismert módon a **Queue** osztály – tömbös reprezentációra alapozott – leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje?

Queue
<code>- Z : T[] // T is some known type ; Z.length is the physical length of the queue</code>
<code>- constant m0 : N<sub>+</sub> := 16 // length of the queue, its default is m0.</code>
<code>- n : N // n ∈ 0..Z.length is the actual length of the queue</code>
<code>- k : N // k ∈ 0..(Z.length - 1) : the starting position of the queue in Z</code>
<code>+ Queue(m : N<sub>+</sub> := m0) { Z := new T[m] ; n := 0 ; k := 0 }</code>
<code>// create an empty queue</code>
<code>+ add(x : T) // join x to the end of the queue</code>
<code>+ rem() : T // remove and return the first element of the queue</code>
<code>+ first() : T // return the first element of the queue</code>
<code>+ length() : N {return n}</code>
<code>+ isEmpty() : B {return n = 0}</code>
<code>+ ~ Queue() { delete Z }</code>
<code>+ setEmpty() {n := 0} // reinitialize the queue</code>



<code>double Full QueueArray(&amp;A[1 : T], &amp;k : N)</code>
<code>B[0 : T] := new T[2 * A.length];</code>
<code>int count := 0;</code>
<code>i := k to A.length</code>
<code>  B[count] := A[i]</code>
<code>  count++</code>
<code>k := 0</code>
<code>A := B; delete A</code>

A sorok műveleteit – az add művelet kivételével – egyszerű, rekúziót és ciklust nem tartalmazó metódusokkal írtuk le. Ezért mindegyik műveletigénye  $\Theta(1)$ , ami – legalábbis együtt az összes elvégzett különféle művelet átlagos műveletigényét tekintve – alapkötetelmény minden sor megvalósítással kapcsolatban. Az add műveletre nyilván  $mT(n) \in \Theta(1)$  és  $MT(n) \in \Theta(n)$ .

2)

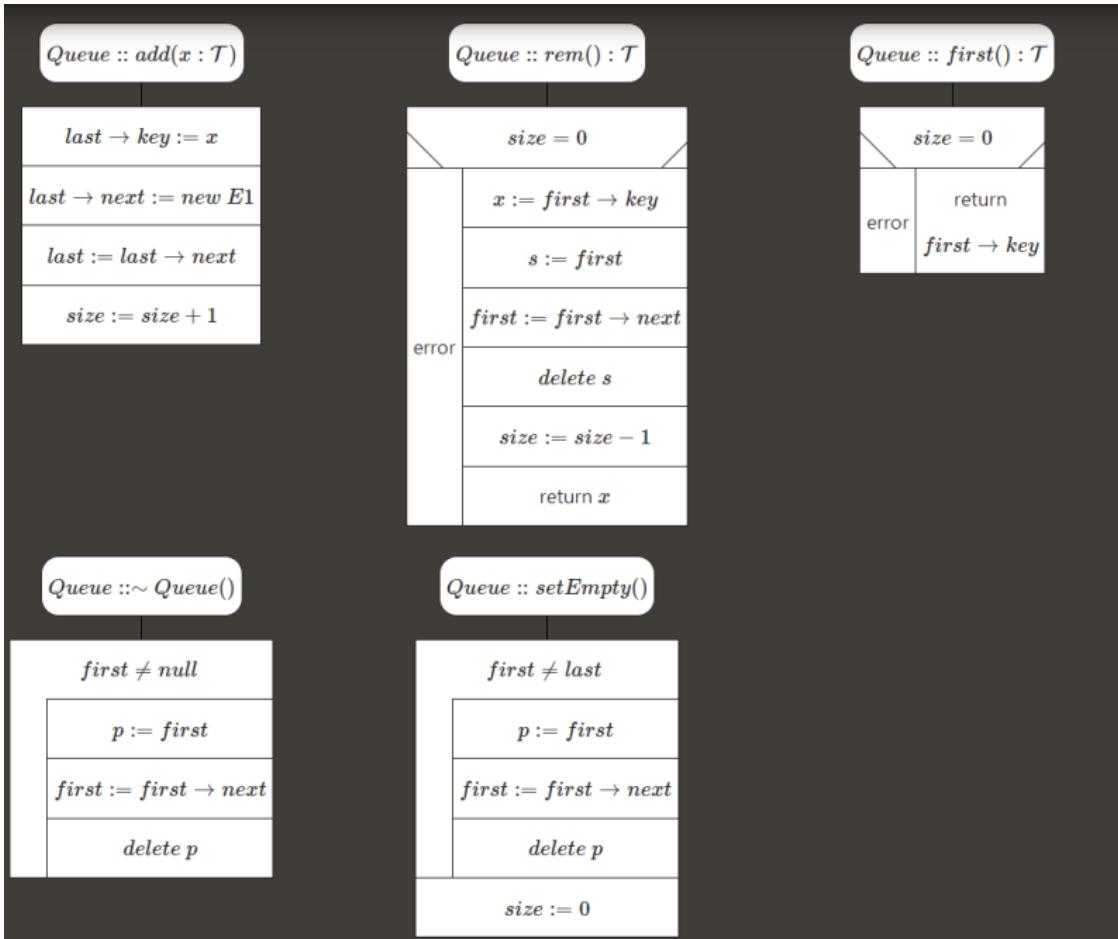
Adja meg az előadásról ismert Queue osztály – egyirányú, nem ciklikus, láncolt listás reprezentációra (S1L) alapozott – leírását, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! (Szüksége lesz arra, hogy a lista elejét és végét is közvetlenül elérje.) Mekkora az egyes műveletek aszimptotikus futási ideje?

#### Láncolt ábrázolás:

A sor láncolt reprezentációjában a *first* pointer nem csak az adattípushoz biztosít hozzáférést, hanem egyben a sor első elemére is mutat. Bevezetjük a *last* pointert is, amely egy „joker” (végelem vagy trailer) elemre mutat, amiben nincs adat. Új elem beszúrásakor a beszúrandó kulcsot a végelemenben tároljuk el és készítünk egy új végelement. Üres sor esetén *first = last*, mivel a konstruktur létrehozza a „joker” elemet és a *first* erre az elemre mutat kezdetben.

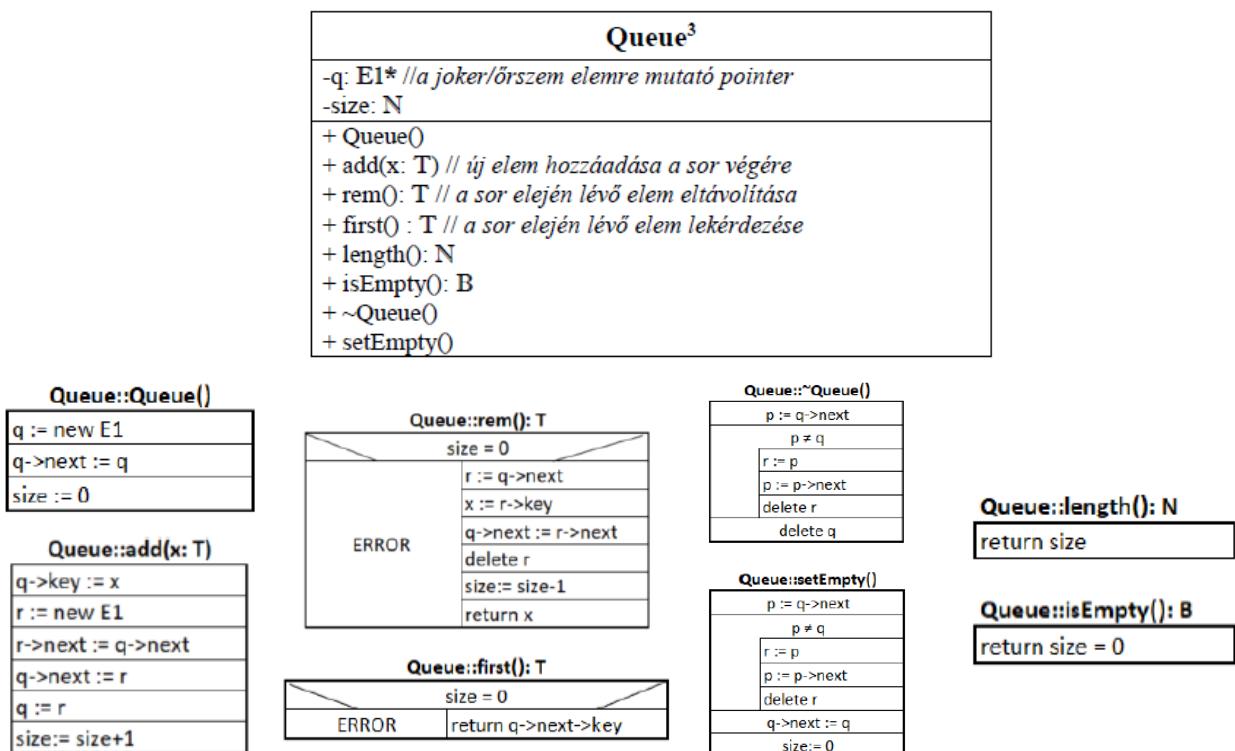
Queue
<code>- first, last : E1*</code>
<code>- size : N</code>
<code>+ Queue() {first := last := new E1; size := 0}</code>
<code>+ ~ Queue()</code>
<code>+ add(x : T)</code>
<code>+ rem() : T</code>
<code>+ first() : T</code>
<code>+ length() : N {return size}</code>
<code>+ isEmpty() : B {return size = 0}</code>
<code>+ setEmpty()</code>

A láncolt ábrázolású sor műveletei között nem szerepel az *IsFull()* művelet, mivel ebben a reprezentációban nem számolunk a tárolókapacitás gyakorlati felső korlátjával. A destruktur és a *setEmpty()* műveletek futási ideje, így  $\Theta(n)$ .



A sorok műveleteit – az add művelet kivételével – egyszerű, rekurziót és ciklust nem tartalmazó metódusokkal írtuk le. Ezért mindegyik műveletigénye  $\Theta(1)$ , ami – legalábbis együtt az összes elvégzett különféle művelet átlagos műveletigényét tekintve – alapkötetelmény minden sor megvalósítással kapcsolatban. Az add műveletre nyilván  $mT(n) \in \Theta(1)$  és  $MT(n) \in \Theta(n)$ .

- 3) Adja meg az előadásról ismert Queue osztály – egyirányú, ciklikus, fejelemes láncolt listás reprezentációra (H1L) alapozott – leírását, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! A listát csak a fejelemre mutató pointeren keresztül szabad elérni. Mekkora az egyes műveletek aszimptotikus futási ideje?

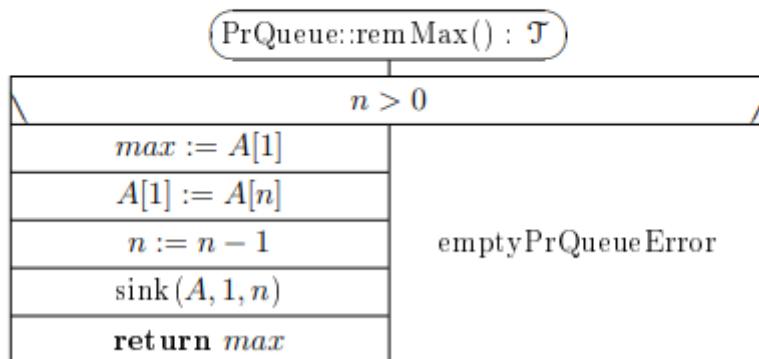
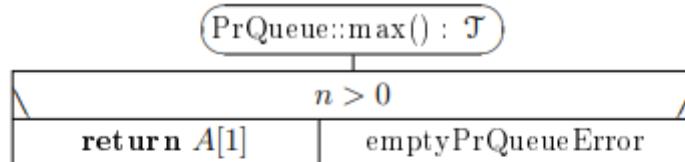
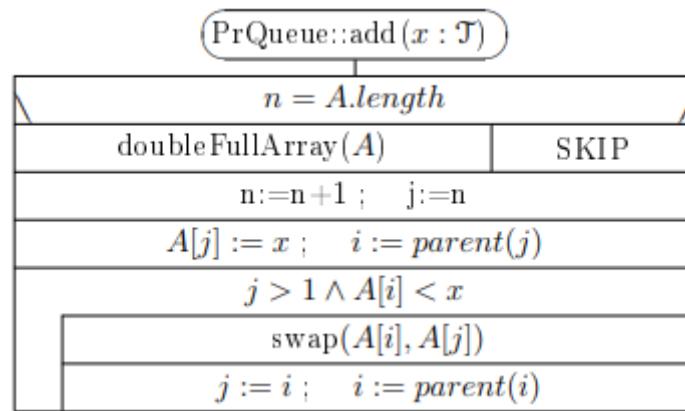


### 3.3. Prioritásos sor

1)

Tegyük fel, hogy a prioritásos sort tömbben, maximum kupaccal reprezentáljuk! Adja meg az előadásról ismert módon a PrQueue osztály leírását, a metódusok struktogramjaival együtt! (A lesüllyesztést nem kell leírni.) Mekkora az egyes műveletek aszimptotikus futási ideje?

PrQueue	
- $A/1 : \mathcal{T}[]$ // $\mathcal{T}$ is some known type ; $A.length$ is the physical	
- constant $m0 : \mathbb{N}_+ := 16$ // size of the PrQ, its default is $m0$ .	
- $n : \mathbb{N}$ // $n \in [0..A.length]$ is the actual length of the PrQ	
+ PrQueue( $m : \mathbb{N}_+ := m0$ ) { $A := \text{new } \mathcal{T}[m]$ ; $n := 0$ } // create an empty PrQ	
+ add( $x : \mathcal{T}$ ) // insert $x$ into the priority queue	
+ remMax() : $\mathcal{T}$ // remove and return the maximal element of the priority queue	
+ max() : $\mathcal{T}$ // return the maximal element of the priority queue	
+ isEmpty() : $\mathbb{B}$ { return $n = 0$ }	
+ ~ PrQueue() { delete $A$ }	
+ setEmpty() { $n := 0$ } // reinitialize the priority queue	



$$MT_{\text{add : } n < A.length}(n) \in \Theta(\log n)$$

- a ciklus legfeljebb annyiszor iterál, amennyi a fa magassága:  $\lfloor \log(n + 1) \rfloor$
- $\log n \leq MT_{\text{add : } n < A.length}(n) = \lfloor \log(n + 1) \rfloor + 1 \leq \log n + 2$

$$MT_{\text{add : } n = A.length}(n) \in \Theta(n)$$

- a doubleFullArray(A) ciklusa:  $n$  iterációt hajt végre
- ez dominál a többi tevékenységek műveletigénye fölött

$$mT_{\text{add}}(n) \in \Theta(1)$$

- $X$  elég kicsi -> ciklus egyet sem iterál

$$AT_{\text{add}}(n) \in O(\log n)$$
 (amortizált műveletigény számítás)

- egyrészt  $MT_{\text{add : } n < A.length}(n) \in \Theta(\log n)$
- másrészt  $n = A.length \rightarrow \text{doubleFullArray}(A)$   $n$  iteráció
- ez kettesével képezetben szütsztható az előző  $n/2$  add() hívás között:
  - biztosan nem hívtuk meg a doubleFullArray(A) eljárást
  - így az az  $n/2$  add() hívás  $O(\log n)$  műveletigénye kettővel nő meg
  - azaz  $O(\log n)$  marad

$$T_{\max}(n) \in \Theta(1)$$

$$MT_{\text{sink}}(h) = h + 1$$
 ( $h$ :  $k$  gyökerű lyukas kupac magassága)

- a ciklus legfeljebb  $h$  iterációt végez
- $1 \leq mT_{\text{sink}}(h) \leq 2$
- Ha  $k = 1 \rightarrow \log n \leq MT_{\text{sink}}(n) = h + 1 = \lfloor \log n \rfloor + 1 \leq \log n + 2$

$$MT_{\text{remMax}}(n) \in \Theta(\log n)$$

$$\log n \leq \log(n - 1) + 1 \leq MT_{\text{remMax}}(n) \leq \log(n - 1) + 2 \leq \log n + 2$$

$$mT_{\text{remMax}}(n) \in \Theta(1)$$

$$2 = 1 + 1 \leq mT_{\text{remMax}}(n) = 1 + mT_{\text{sink}}(n-1) \leq 1 + 2 = 3.$$

2)

- a. Egy bináris fa mikor szigorúan bináris? Mikor teljes? Mikor majdnem teljes? Ez utóbbi mikor balra tömörített, és mikor kupac?

**Szigorúan bináris fa:** Azok a bináris fák, amelyekben minden belső (azaz nem-levél) csúcsnak két gyereke van.

**Teljes bináris fa:** Olyan szigorúan bináris fa, amelynek minden levele azonos szinten helyezkedik el.

- $h$  mélységű teljes bináris fa csúcsainak száma:  $1+2+4+\dots+2^h = 2^{h+1} - 1$

**Majdnem teljes bináris fa:** Olyan teljes bináris fa, amelynek legalsó levélszintjéről elhagytunk néhány levelet (de nem az összeset).

- $h$  mélységű majdnem teljes bináris fa csúcsainak száma:  $n \in 2^h \dots 2^{h+1} - 1$ , és így  $h = \lfloor \log n \rfloor$
- Az alsó szinten levő leveleket elvéve egy  $h - 1$  mélységű teljes bináris fát kapunk

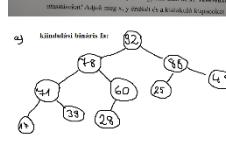
Egy majdnem teljes balra tömörített bináris fa: majdnem teljes bináris fa, de levelek csak a legalsó szintről, a jobb oldalról hiányozhatnak. Ezeket szintfolytonos fáknak is nevezzük.

**Kupac:** Maximum kupac: egy majdnem teljes, balra tömörített bináris fa, amelynek minden belső pontjára teljesül, hogy a belső pont kulcsa nagyobb vagy egyenlő a gyerekei kulcsánál. Így kupac tetején (a fa gyökerében) minden elem legnagyobb elem található.

- Minimum kupac hasonlóan: a szülő kulcs kisebb vagy egyenlő a gyerekei kulcsánál.

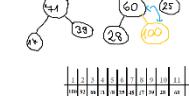
- b. Szemléltesse az alábbi kupacra a 9, majd az eredmény kupacra a 8 beszúrásának műveletét!  
 $\leq 8; 8; 6; 6; 5; 2; 3; 1; 5; 4 \geq$

Zer rendszert használunk az eljárásban, amely minden lehetséges műveletet leírja. Ez a részleges művelet a 9-es beszúráshoz.



a) hibásan hiba! 1

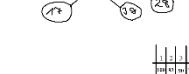
b) add(100)



c) add(90)



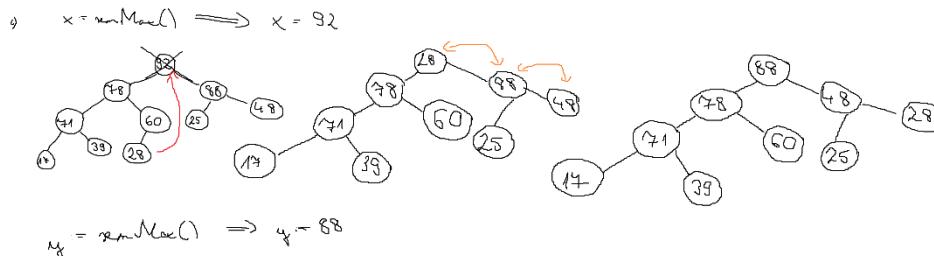
d) add(80)



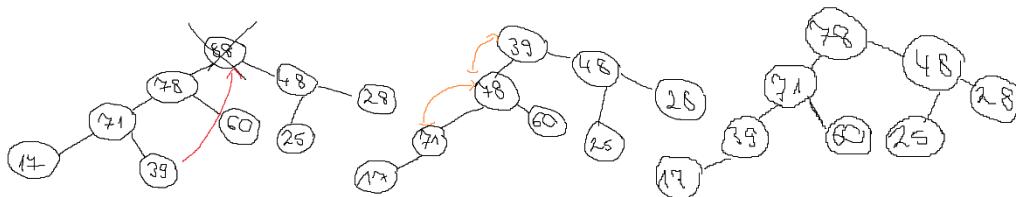
e) add(70)



c. Szemléltesse az **eredeti kupacra** a remMax() eljárás kétszeri végrehajtását! minden művelet után rajzolja újra a fát!



$$x = \text{remMax}() \Rightarrow x = 88$$



3)

Tegyük fel, hogy a prioritásos sort tömbben, rendezetlen módon reprezentáljuk! Adja meg az előadásról ismert módon a **PrQueue** osztály leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje?

4)

Tegyük fel, hogy a prioritásos sort tömbben monoton növekvően rendezett sorozatként tároljuk! Adja meg az előadásról ismert módon a **PrQueue** osztály leírását, a metódusok struktogramjaival együtt! Mekkora az egyes műveletek aszimptotikus futási ideje?

5)

Tegyük fel, hogy a prioritásos sort fejelemes láncolt listával, rendezetlen módon reprezentáljuk! Adja meg az előadásról ismert módon a **PrQueue** osztály leírását erre az esetre, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! Mekkora az egyes műveletek aszimptotikus futási ideje?

6)

Tegyük fel, hogy a prioritásos sort fejelemes láncolt listával, monoton csökkenően rendezett sorozatként tároljuk! Adja meg az előadásról ismert módon a **PrQueue** osztály leírását erre az esetre, a metódusok struktogramjaival együtt! Törekedjen hatékony megvalósításra! Mekkora az egyes műveletek aszimptotikus futási ideje?

#### Prioritásos sor ábrázolásainak összehasonlítása:

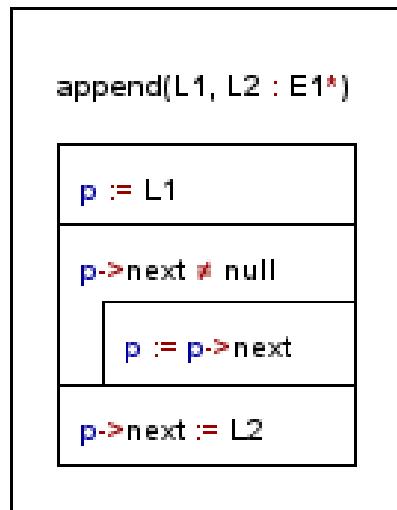
	$\text{add}(x : \mathcal{T})$	$\text{remMax}()$	$\text{max}()$
rendezetlen tömbbel (ha a maximális elem indexét nyilvántartjuk)	$\Theta(1)$	$\Theta(n)$	$\Theta(1)$
növekvően rendezett tömbbel	$O(n)$	$\Theta(1)$	$\Theta(1)$
maximum kupaccal	$O(\log n)$	$O(\lg n)$	$\Theta(1)$

## 4. Láncolt listák

### 4.1. Egyszerű listák (S1L)

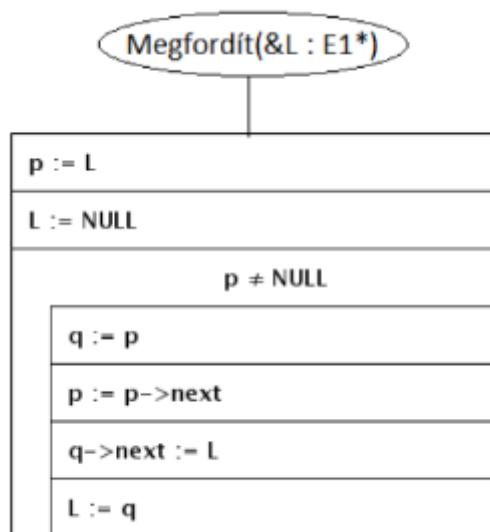
1)

Az  $L_1$  és  $L_2$  pointerek két egyszerű láncolt listát azonosítanak. Írja meg az  $\text{append}(L_1; L_2)$  eljárást, ami  $MT(n) \in \Theta(n)$  és  $mT(n) \in \Theta(1)$  ( $n = |L_1|$ ) műveletigénnel az  $L_1$  lista után fűzi az  $L_2$  listát!



2)

Írja meg a  $\text{reverse}(L)$  eljárást, ami megfordítja az  $L$  egyszerű láncolt lista elemeinek sorrendjét!  $T(n) \in \Theta(n)$ , ahol  $n$  a lista hossza.



## 4.2. Fejelemes listák (H1L)

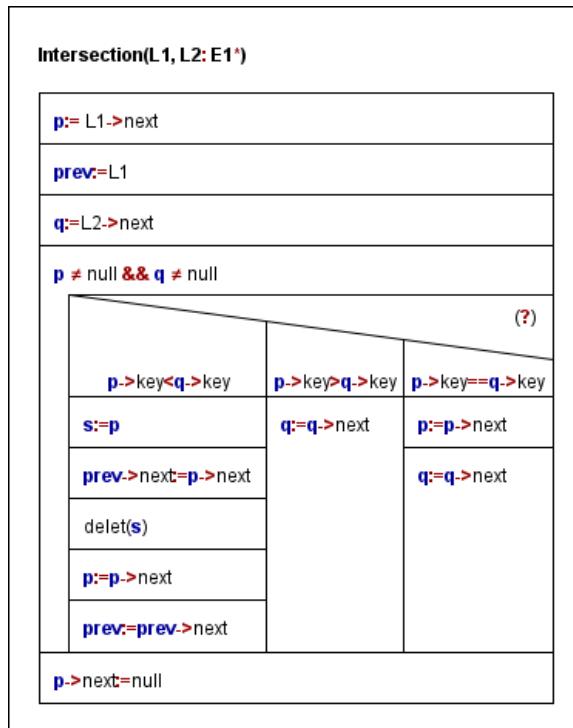
1)

Az  $L_1, L_2$  pointerek egy-egy szigorúan monoton növekvő H1L (fejelemes, egyirányú, nem ciklikus, láncolt lista) fejelemére mutatnak.

Írjuk meg az intersection( $L_1, L_2$ ) eljárást, ami az  $L_1$  lista elemei közül törli azokat az elemeket, amelyek kulcsa nem szerepel az  $L_2$  listán! Így az  $L_1$  listán a két input lista metszete jön létre, míg az  $L_2$  lista változatlan marad.

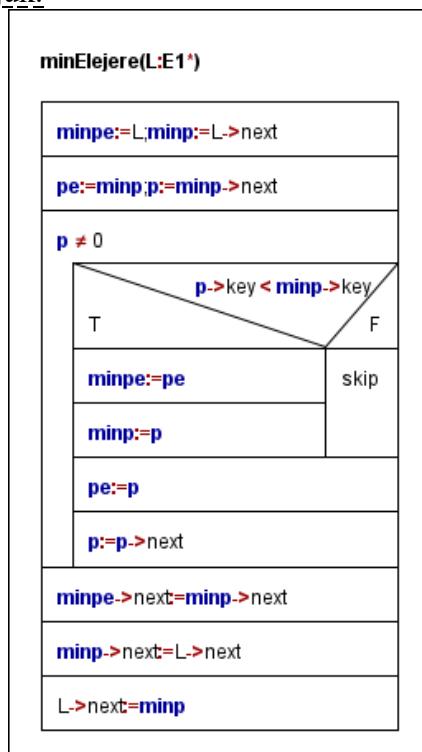
Mindkét listán legfeljebb egyszer menjünk végig! Listaelemeket ne hozzunk létre!

$MT(n_1, n_2) \in O(n_1 + n_2)$  és  $mT(n_1, n_2) \in O(n_1)$  legyen, ahol  $n_1$  az  $L_1$ ,  $n_2$  az  $L_2$  lista hossza.



2)

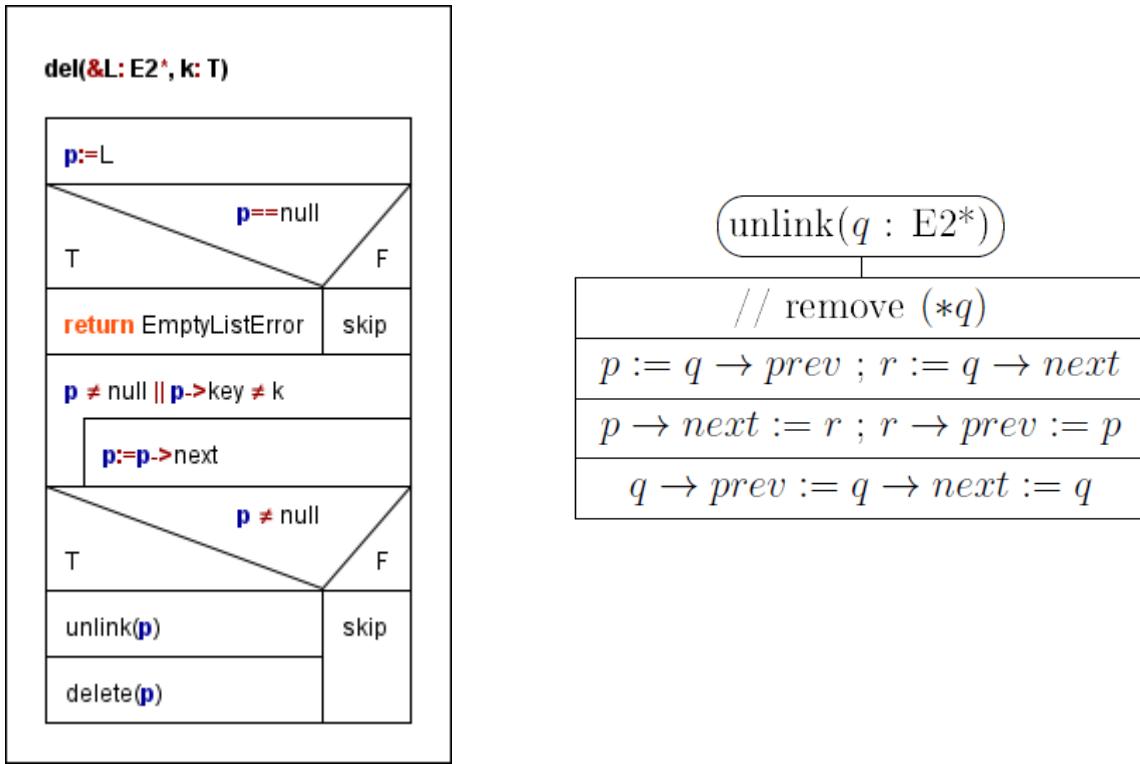
Az  $L$  pointer egy nemüres, fejelemes láncolt lista fejelemére mutat. Írjuk meg a minElejére( $L$ ) eljárást, ami az  $L$  lista legkisebb kulcsú elemét átfűzi az  $L$  lista elejére! A program a listán csak egyszer menj végig!  $T(n) \in \Theta(n)$ , ahol  $n$  az  $L$  lista hossza. A listaelemeknek a key és a next mezőkön kívül más részei is lehetnek, de ezeket nem ismerjük.



### 4.3. Egyeszerű kétirányú listák (S2L)

1)

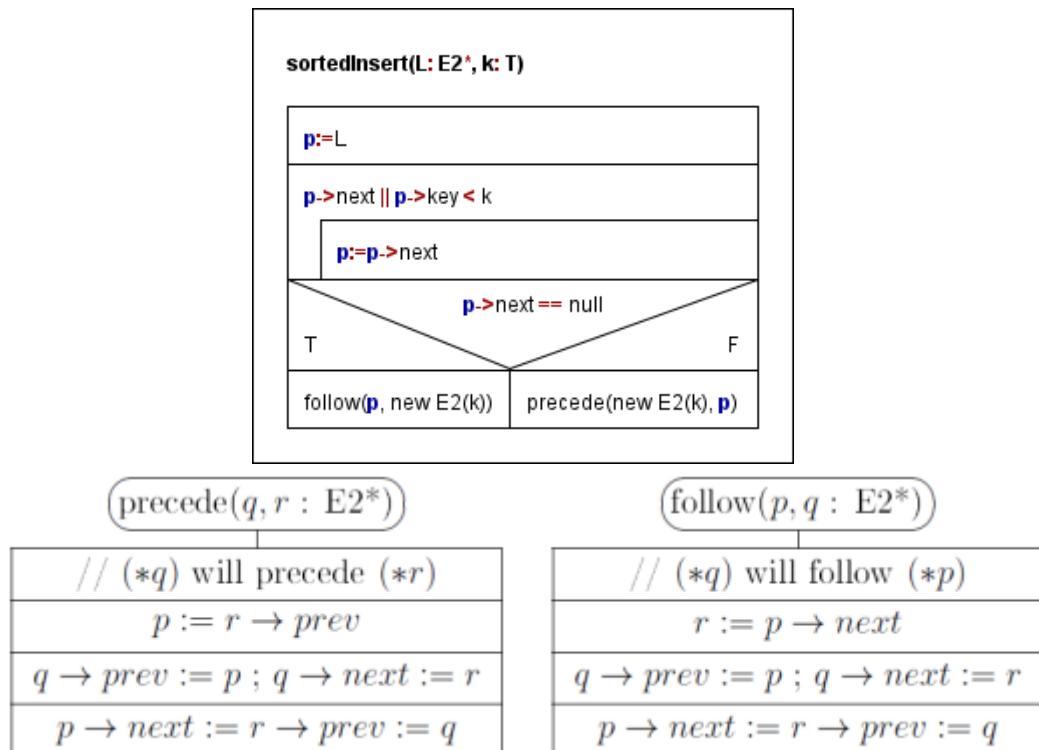
Az  $L$  pointer egy rendezetlen, kétirányú, fejelem nélküli, nem ciklikus, láncolt listát azonosít. Írjuk meg a  $\text{del}(L, k)$  eljárást, ami az  $L$  listából törli a  $k$  kulcsú listaelemeket! A listán legfeljebb egyszer menjünk végig! Listaelement ne hozzunk létre, a feleslegessé váltó listaelemeket viszont deallokáljuk! Mekkora a fenti eljárás műveletigénye? Miért?



2)

Az  $L$  pointer egy monoton növekvő kétirányú, fejelem nélküli, nem ciklikus, láncolt listát azonosít. Írjuk meg a  $\text{sortedInsert}(L, k)$  eljárást, ami az  $L$  listába rendezetten beszűr egy  $k$  kulcsú listaelemet! A listán legfeljebb egyszer menjünk végig! Pontosan egy listaelement hozzunk létre!

$MT(n) \in \Theta(n)$  és  $mT(n) \in O(1)$  legyen, ahol  $n$  az  $L$  lista hossza.



#### 4.4. Fejelemes, kétirányú, ciklikus listák (C2L)

1)

Az  $L_1, L_2$  pointerek egy-egy szigorúan monoton növekvő C2L (fejelemes, kétirányú, ciklikus, láncolt lista) fejelemére mutatnak. A listákat kizárolag az  $\text{unlink}(q)$ ,  $\text{precede}(q, r)$  és  $\text{follow}(p, q)$  eljárásokkal szabad módosítani, ahogy azt az előadáson tanultuk.

Írjuk meg a  $\text{difference}(L_1, L_2)$  eljárást, ami az  $L_1$  lista elemei közül törli az  $L_2$  listán is szereplő elemeket!  
Az  $L_2$  lista változatlan, de az  $L_1$  is szigorúan monoton növekvő marad. Mindkét listán legfeljebb egyszer menjünk végig!

A felszabaduló listaelemeket adjuk vissza a szabad területnek!

$MT(n_1, n_2) \in O(n_1 + n_2)$  és  $mT(n_1, n_2) \in O(\min(n_1, n_2))$  legyen, ahol  $n_1$  az  $L_1$ ,  $n_2$  az  $L_2$  lista hossza.

difference(&L1,L2 : E2*)		
$p := L1 \rightarrow \text{next}$		
$q := L2 \rightarrow \text{next}$		
$p \neq L1 \&& q \neq L2$		???
$p \rightarrow \text{key} > q \rightarrow \text{key}$	$p \rightarrow \text{key} < q \rightarrow \text{key}$	$p \rightarrow \text{key} = q \rightarrow \text{key}$
$q := q \rightarrow \text{next}$	$p := p \rightarrow \text{next}$	$r := p$
		$p := p \rightarrow \text{next}$
		$q := q \rightarrow \text{next}$
		$\text{unlink}(r)$
		$\text{delete } r$

2)

Az  $L_1, L_2$  pointerek egy-egy szigorúan monoton növekvő C2L (fejelemes, kétirányú, ciklikus, láncolt lista) fejelemére mutatnak. A listákat kizárolag az  $\text{unlink}(q)$ ,  $\text{precede}(q, r)$  és  $\text{follow}(p, q)$  eljárásokkal szabad módosítani, ahogy azt az előadáson tanultuk.

Írjuk meg a  $\text{unionIntersection}(L_1, L_2)$  eljárást, ami az  $L_1$  lista elemei közé átfűzi az  $L_2$  listáról az  $L_1$  listán eredetileg nem szereplő elemeket! Így az  $L_1$  listán a két input lista uniója, míg az  $L_2$  listán a metszetük jön létre, és minden lista szigorúan monoton növekvő marad. Mindkét listán legfeljebb egyszer menjünk végig! Listaelemeket ne hozunk létre és ne is töröljünk!

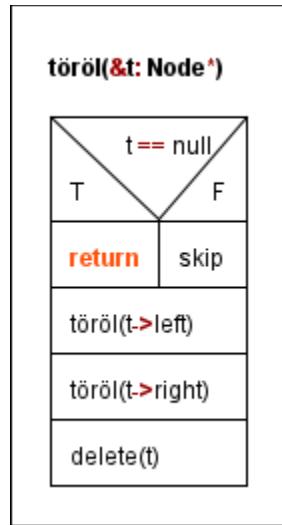
$MT(n_1, n_2) \in O(n_1 + n_2)$  és  $mT(n_1, n_2) \in O(n_2)$  legyen, ahol  $n_1$  az  $L_1$ ,  $n_2$  az  $L_2$  lista hossza.

(unionIntersection( $H_u, H_i : E2*$ ))		
$q := H_u \rightarrow \text{next} ; r := H_i \rightarrow \text{next}$		
$q \neq H_u \wedge r \neq H_i$		
$q \rightarrow \text{key} < r \rightarrow \text{key}$	$q \rightarrow \text{key} > r \rightarrow \text{key}$	$q \rightarrow \text{key} = r \rightarrow \text{key}$
$q := q \rightarrow \text{next}$	$p := r$	
	$r := r \rightarrow \text{next}$	$q := q \rightarrow \text{next}$
	$\text{unlink}(p)$	$r := r \rightarrow \text{next}$
	$\text{precede}(p, q)$	
$r \neq H_i$		
$p := r ; r := r \rightarrow \text{next} ; \text{unlink}(p)$		
$\text{precede}(p, H_u)$		

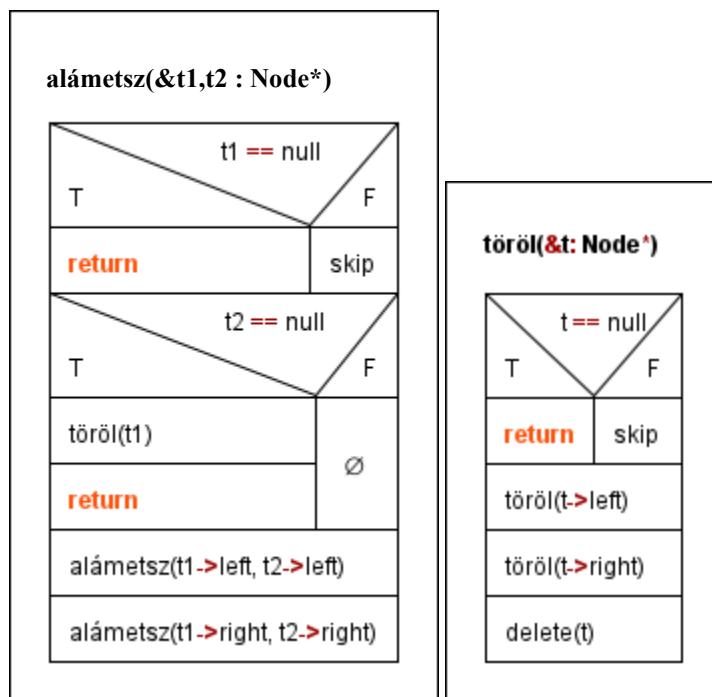
## 5. Bináris fák

1)

- a. A `t : Node*` típusú pointer egy láncoltan ábrázolt bináris fát azonosít. A fa csúcsaiban nincsenek „parent” pointerek. Írjuk meg a `töröl(t)` rekurzív eljárást, ami törli a t fa csúcsait,  $\Theta(|t|)$  műveletigénnel, a posztorder bejárás szerint! Rendelkezésre áll ehhez a `delete p` utasítás, ami a p pointer által mutatott csúcsot törli. A t fa végül legyen üres!

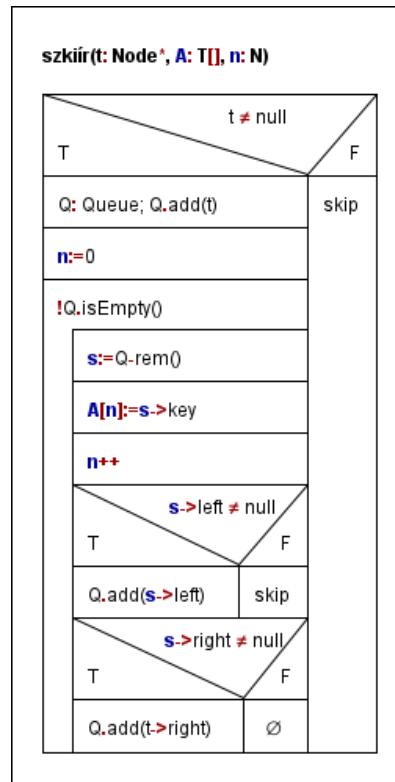


- b. A `t1, t2 : Node*` típusú pointerek egy-egy láncoltan ábrázolt bináris fát azonosítanak. A fák csúcsaiban nincsenek „parent” pointerek. A `t2` fa akkor fedи le a `t1` fát, ha `t1` üres, vagy ha egyikük sem üres, és a `t1` bal és jobb részfáját is lefedik a `t2` megfelelő oldali részfái. Egy fa megmetszése alatt bizonyos részfái törlését értjük. Írjuk meg az alámetsz(`t1, t2`) rekurzív eljárást, ami úgy metszi meg a `t1` fát, hogy a `t2` fa lefedje, de a `t1` fa a lehető legnagyobb maradjon! A `t1` feleslegessé váló részfait törljük a `töröl(t)` eljárás segítségével!  $T(n) \in \Theta(n)$  legyen, ahol  $n$  a `t1` fa mérete.



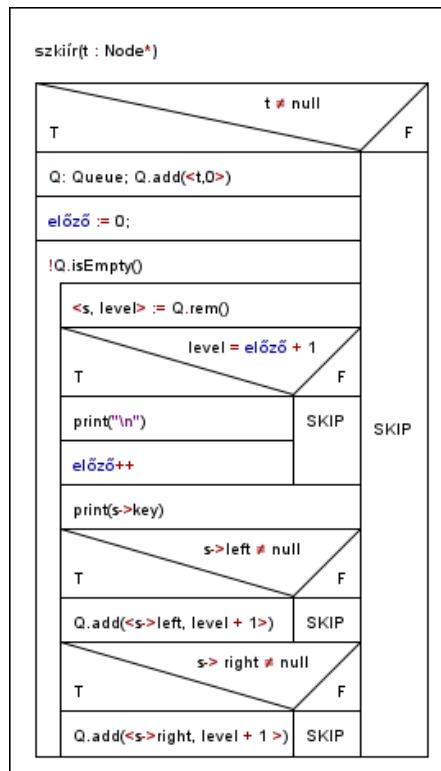
2)

A  $t : \text{Node}^*$  típusú pointer egy láncoltan ábrázolt bináris fát azonosít, ami minden teljes és balra tömörített. A fa csúcsaiban nincsenek „parent” pointerek. Írja meg az szkiír( $t, A, n$ ) eljárást, ami  $\Theta(|t|)$  műveletigénnel a fa kulcsait szintfolytonosan az  $A$  tömbbe írja, és  $n$ -ben visszaadja a fa méretét! Feltehető, hogy a tömb elég nagy. Az eljárás a fát ne változtassa meg!



3)

A  $t : \text{Node}^*$  típusú pointer egy láncoltan ábrázolt bináris fát azonosít. A fa csúcsaiban nincsenek „parent” pointerek. Írja meg az szkiír( $t$ ) eljárást, ami  $\Theta(|t|)$  műveletigénnel szintenként írja ki a fa kulcsait úgy, hogy minden szint külön sorba kerül, azaz az első sorban csak a fa gyökér csúcsának kulcsa jelenik meg, a második sorban a gyerekei kulcsai, a harmadikban az unokáié stb. Mindegyik szintet balról jobbra írjuk ki! Az eljárás a fát ne változtassa meg!

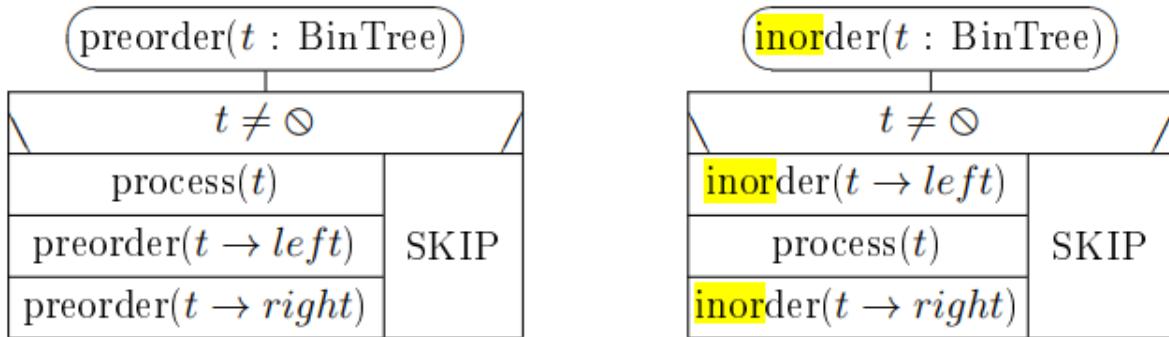


4)

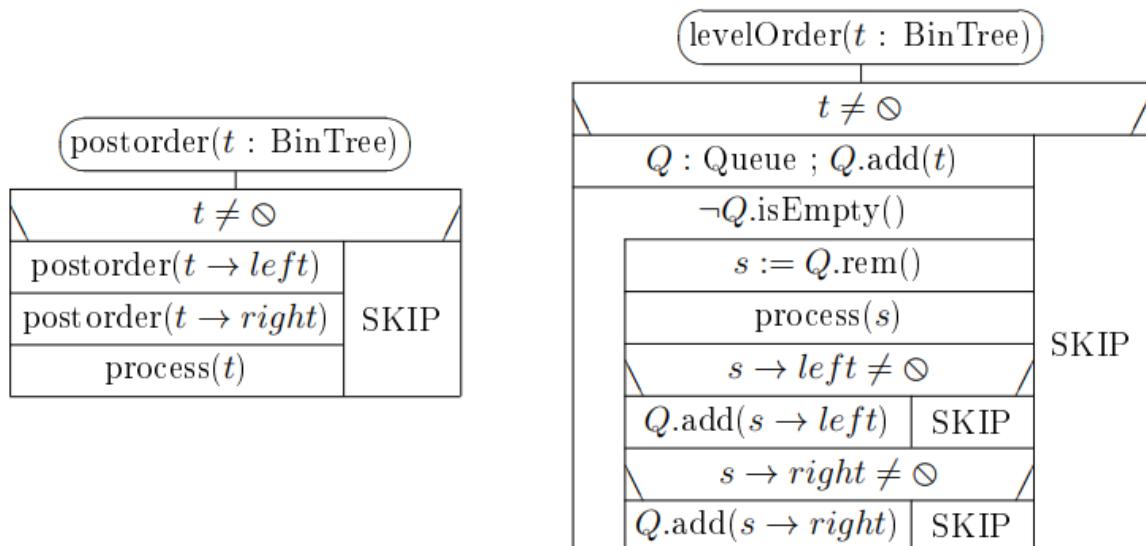
Milyen bináris fa bejárásokat ismer?

Inorder, preorder, postorder, valamint levelorder bejárásokat.

a. Adja meg a struktogramjaikat!



<sup>18</sup>A struktogramokban a „\*t” csúcs feldolgozását „process(t)” jelöli.



b. Számolja ki a struktogramokhoz tartozó műveletigényeket!

## Bináris fák bejárásai

- Üres fára mindegyik bejárás az üres program
- BinTree: a bináris fák absztrakt típusát jelöli
- A struktogramokban a \*t csúcs feldolgozását process(t) jelöli
- $T(\text{process}(t)) \in \Theta(1)$
- **Állítás:**  $T_{\text{preorder}}(n), T_{\text{inorder}}(n), T_{\text{postorder}}(n), T_{\text{levelOrder}}(n) \in \Theta(n)$ , ahol n a fa mérete, azaz csúcsainak száma.
- **Igazolás:**
- Rekurzív bejárások
  - Hívások száma= Binfa részfáinak száma (az üres részfák is)
    - $n$  szerinti teljes indukcióval belátható, hogy tetszőleges  $n$  csúcsú bináris fának  $2n + 1$  részfája van.
    - egy-egy hívás végrehajtása:  $\Theta(1)$
- A szintenkénti bejárás:
  - minden csúcsnál: a ciklus egy-egy végrehajtása
  - egy fa csúcsainak szintenkénti felsorolásában a korábbi csúcs gyerekei megelőzik a későbbi csúcs gyerekeit.
  - Ez az állítás nyilvánvaló, akár egy szinten, akár különböző szinten van a két csúcs a fában.

## 5.1. Bináris keresőfák (és rendezőfák)

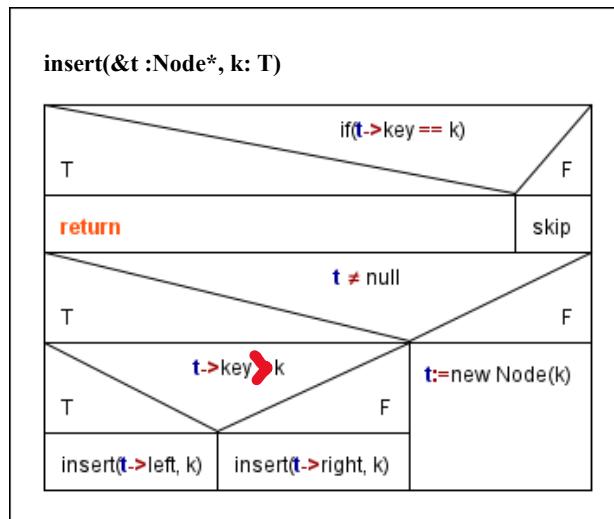
1)

- a. A bináris fa fogalmát ismertnek feltételezve, mondjuk ki a bináris keresőfa definícióját!

Egy bináris fát keresőfának nevezünk, ha minden belső csúcsára és annak  $y$  kulcsára igaz:

- A csúcs bal részfájában minden csúcs  $x$  kulcsára  $x \leq y$ ,
- A csúcs jobb részfájában minden csúcs  $z$  kulcsára  $z \geq y$

- b. A  $t$  egy láncoltan ábrázolt bináris keresőfa gyökérpointere. A csúcsokban nincsenek „parent” pointerek. (A fa üres is lehet.) Írja meg az előadásról ismert módon az  $\text{insert}(t, k)$  eljárás rekurzív struktogramját, ami megkeresi a  $t$  fában a  $k$  kulcs helyét, és ha ott egy üres részfát talál, akkor az üres részfa helyére tesz egy új levélcsecsöt,  $k$  kulccsal.



- c. Mekkora az  $\text{insert}(t, k)$  eljárás maximális, illetve minimális műveletigénye? Miért?

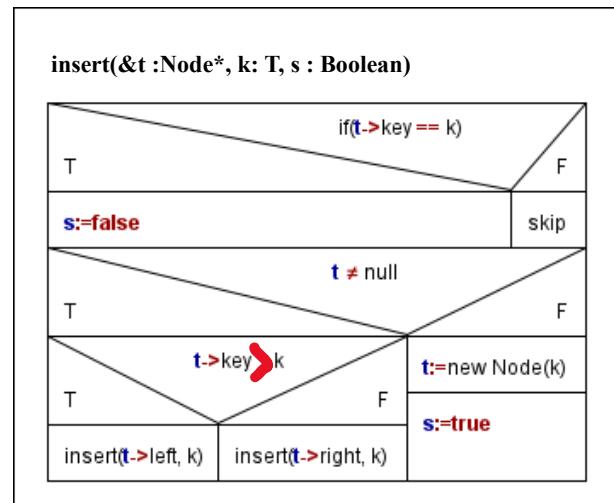
Maximális műveletigénye:  $MT(n) \in \Theta(n)$ , minimális műveletigénye:  $mT(n) \in \Theta(1)$ .

2)

- a. A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!

-> lásd 1) a. megoldás

- b. Írja meg az  $\text{insert}(t, k, s)$  – ciklust nem tartalmazó –  $MT(h) \in \Theta(h)$  hatékonyágú rekurzív eljárást ( $h = h(t)$ ), ami megróbál beszűrni a  $t$  bináris keresőfába egy  $k$  kulcsú csúcsot (akkor tudja beszűrni, ha a fában nem talál ilyet), és az  $s$ , logikai típusú paraméterben visszaadja, hogy sikeres volt-e a beszűrés! A fa csúcsai **Node** típusúak; szülő pointert nem tartalmaznak.



c. Igaz-e, hogy a fent insert( $t, k, s$ ) eljárásra  $mT(h) \in \Theta(1)$ ? Miért?

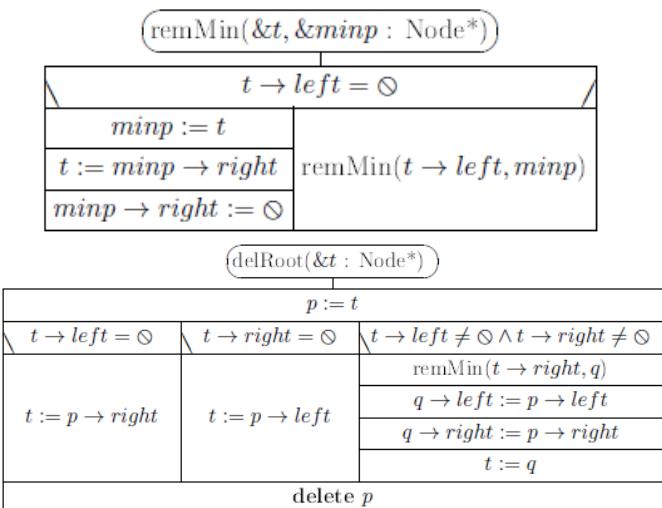
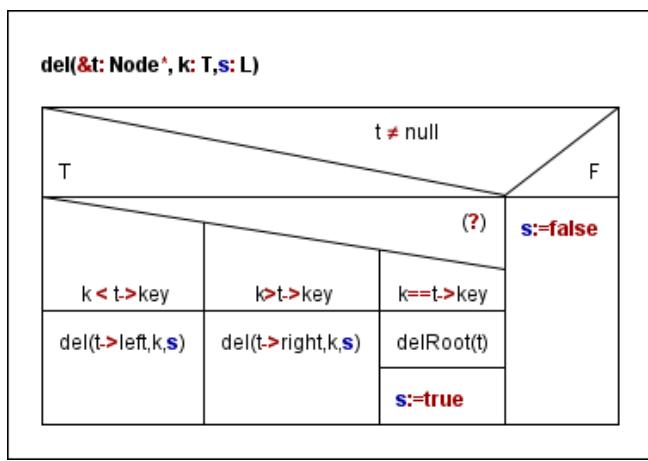
Igaz, mivel legjobb esetben üres a fa, vagy csak egy eleme van és így konstans időben be tudjuk szúrni a  $k$  kulcsot.

3)

a. A bináris fával kapcsolatos egyéb fogalmakat ismertnek feltételezve, definiálja a bináris keresőfa fogalmát! Adott a remMin( $t, minp$ ) eljárás az előadásról ismert jelentéssel. (Ezt nem kell megírni.)

-> lásd 1) a. megoldás

b. Írja meg a del( $t, k, s$ ) – ciklust nem tartalmazó –  $MT(h(t)) \in \Theta(h(t))$  hatékonyságú rekurzív eljárást, ami megpróbálja törölni a láncoltan ábrázolt  $t$  bináris keresőfából a  $k$  kulcsú csúcsot! (Akkor tudja törölni, ha talál ilyet a fában.) Az  $s$ , logikai típusú paraméterben visszaadjuk, hogy sikeres volt-e a törlés! A fa csúcsai „parent” pointert nem tartalmaznak számunkra ismeretlen, járulékos adatmezőket viszont tartalmazhatnak.



c. Igaz-e, hogy a fent del( $t, k, s$ ) eljárásra  $mT(h(t)) \in \Theta(1)$ ? Miért?

Igaz, abban az esetben, ha a fa gyökerét szeretnénk kitörölni.

4)

a. A bináris fa fogalmát ismertnek feltételezve, mondjuk ki a bináris keresőfa definícióját!

-> lásd 1) a. megoldás

b. Adott a  $t$  bináris fa. A csúcsok kulcsai pozitív egész számok. Írja meg a bst( $t$ ) logikai függvényt, ami a  $t$  egyszeri (Inorder) bejárásával előönti, hogy keresőfa-e!  $MT(n) \in O(n)$ , ahol  $n = |t|$ .  $MS(h) \in O(h)$ , ahol  $h = h(t)$ ;  $MS$  pedig az algoritmus maximális tárigényét jelöli. (A bejárást és előöntést a megfelelően inicializált, rekurzív, bst( $t, k$ ) logikai segédfüggvény végezze, ami híváskor  $k$ -ban a  $t$  kulcsainál kisebb értéket vár, visszatéréskor pedig, amennyiben  $t$  nemüres keresőfa, a  $t$ -beli legnagyobb kulcsot tartalmazza! Ha  $t$  üres, akkor  $k$ -ban maradjon a függvényhívásnál kapott érték!)

isSearchTree(t: Node\*) : B

Node prev:=∅
return isSearchTreeH(t, &prev)

isSearchTreeH( t: Node\*, &prev: Node ) : B

t≠∅	
isSearchTreeH(t->left, prev)	return false
prev≠∅ ∧ prev->key ≥ t->key	return true
prev=t	
return isSearchTreeH(t->right, prev)	

c. Igaz-e az Ön által megfogalmazott bst( $t$ ) logikai függvényre, hogy  $mT(h) \in O(h)$ ? Miért?

Igaz, mivel legjobb esetben 1 összehasonlítással (gyökér, illetve balgyereke nagyobb-e) kiderülhet, hogy nem keresőfa.

5)

A fa magassága (height) egyenlő a legmélyebben fekvő levelei szintszámával. Az üres fa magassága  $h(\emptyset) = -1$ .

Emlékeztetünk, hogy tetszőleges bináris fában a gyökér van a *nulladik* szinten. Az  $i$ -edik szintű csúcsok gyerekeit az  $(i + 1)$ -edik szinten találjuk. A fa *magassága* (*height*) egyenlő a legmélyebben fekvő levelei szintszámával.

A  $t$  bináris fa magasságát  $h(t)$ , vagy ha a szövegösszefüggésből egyértelmű, melyik fáról van szó, egyszerűen csak  $h$  jelöli.

Az üres fa magassága  $h(\emptyset) = -1$ . Így tetszőleges nemüres  $t$  bináris fára:

$$h(t) = 1 + \max(h(t \rightarrow \text{left}), h(t \rightarrow \text{right}))$$

**9.1. Tétel.** *Tetszőleges  $n > 0$  méretű és  $h \geq 0$  magasságú (azaz nemüres) bináris fára*

$$\lfloor \log n \rfloor \leq h \leq n - 1$$

**Bizonyítás.** Először a  $\lfloor \log n \rfloor \leq h$  egyenlőtlenséget bizonyítjuk be. A  $h$  mélységű bináris fák között nyilván a teljes bináris fának van a legtöbb csúcsa. Mivel erre  $n = 2^{h+1} - 1$  (ld. 9.1.), tetszőleges bináris fára  $n < 2^{h+1}$ . Innét  $n > 0$  miatt  $\log n < \log 2^{h+1} = h + 1$ , amiből  $\lfloor \log n \rfloor \leq h$  közvetlenül adódik. Mint 9.1-ben láttuk, tetszőleges majdnem teljes bináris fára teljesül az egyenlőség.

A  $h \leq n - 1$  egyenlőtlenséghez gondoljuk meg, hogy tetszőleges  $h$  magasságú fa szintjeit 0-tól  $h$ -ig sorszámoztuk, ami összesen  $h + 1$  szintet jelent. Mivel a fa minden szintjén van legalább egy csúcs, ezért tetszőleges fára  $n \geq h + 1$ , azaz  $h \leq n - 1$ , ahol  $h = n - 1$  pontosan akkor teljesül, ha a fa listává torzult.  $\square$

- a. Bizonyítsa be, hogy tetszőleges,  $n$  csúcsú és  $h$  magasságú bináris fára az  $n - 1 \geq h$  egyenlőtlenség teljesül!

## Bináris fa magasságáról szóló téTEL

- $h \leq n - 1$ 
  - tetszőleges  $h$  magasságú fa szintjeit 0-tól  $h$ -ig sorszámoztuk:  $h + 1$  szint
  - Mivel a fa minden szintjén van legalább egy csúcs
  - tetszőleges fára  $n \geq h + 1$
  - $h \leq n - 1$
  - $h = n - 1$  pontosan akkor teljesül, ha a fa listává torzult.

- b. Mikor lesz  $h = n - 1$ , és miért?

$h = n - 1$  pontosan akkor teljesül, ha a fa listává torzult.

- c. Bizonyítsa be, hogy  $h \geq \lfloor \log n \rfloor$ , ha a bináris fa nemüres!

## Bináris fa magasságáról szóló téTEL

- **1. Tétel.** Tetszőleges  $n > 0$  méretű és  $h \geq 0$  magasságú (azaz nemüres) bináris fára:

$$\lfloor \log n \rfloor \leq h \leq n - 1$$

- **Bizonyítás.**

- $\lfloor \log n \rfloor \leq h$

- A  $h$  mélységű bináris fák között a legtöbb csúcs: teljes bináris fának van:  $n = 2^{h+1} - 1$ 
  - tetszőleges bináris fára:  $n < 2^{h+1}$
  - $n > 0 \Rightarrow \log n < \log 2^{h+1} = h + 1$
  - $\lfloor \log n \rfloor \leq h$

- d. Bizonyítsa be, hogy  $h = \lceil \lg n \rceil$ , ha az előbbi fa majdnem teljes!
- e. Lehetséges-e, hogy  $h = \lceil \lg n \rceil$ , holott a nemüres bináris fára a majdnem teljesség kritériuma nem teljesül? Miért?
- 6)

a. A bináris fa fogalmát ismertnek feltételezve, definiálja a bináris keresőfa fogalmát!

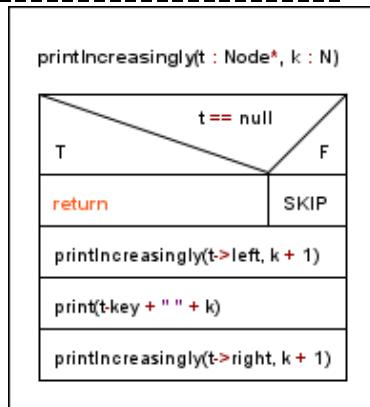
-> lásd 1) a. megoldás

b. Milyen kapcsolat van a bináris keresőfák és az inorder bejárás között? (Indokolja az állítást!)

Az inorder bejárás először bejárja a nulladik részfát, ezután a fa gyökerét dolgozza fel, majd sorban bejárja az 1..r – 1-edik részfákat.

Amennyiben egy bináris fa transzformáció a fa inorder bejárását nem változtatja meg, és adott egy bináris keresőfa, akkor a fa a transzformáció végrehajtása után is bináris keresőfa marad (mivel a kiindulási fa inorder bejárása szigorúan monoton növekvő kulcssorozatot ad, és ez a transzformáció után is így marad).

c. A  $t : Node^*$  típusú pointer egy láncoltan ábrázolt bináris keresőfát azonosít. A fa csúcsaiban nincsenek „parent” pointerek. Írja meg a  $printIncreasingly(t)$  eljárást, ami  $\Theta(|t|)$  műveletigénnel kiírja a fa kulcsainak szigorúan monoton növekvő sorozatát, és minden kulcs mellett megjeleníti azt is, hogy az a fa hányadik szintjén található, mégpedig „(kulcs,szint)” alakban! Az eljárás a fát ne változtassa meg! A struktogramok ne tartalmazzanak ciklust!

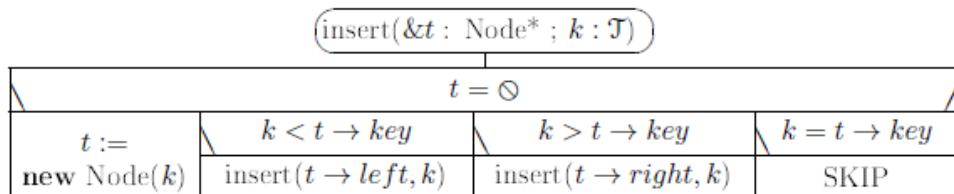


7)

a. Mondja ki a bináris keresőfa definícióját!

-> lásd 1) a. megoldás

b. A  $t : Node^*$  egy láncoltan ábrázolt bináris keresőfát azonosít. Írja meg az előadásról ismert módon az  $insert(t, k)$  eljárást rekurzív struktogramját!



c. Mekkora aszimptotikusan az  $mT(n)$  és az  $MT(n)$ ? Miért?

A legjobb eset:  $mT(n) \in \Theta(1)$ , ekkor üres fába szűrunk be elemet; legrosszabb eset:  $MT(n) \in \Theta(n)$ , ebben az esetben listává torzult a keresőfa és a legvégre kell beszúrnunk.

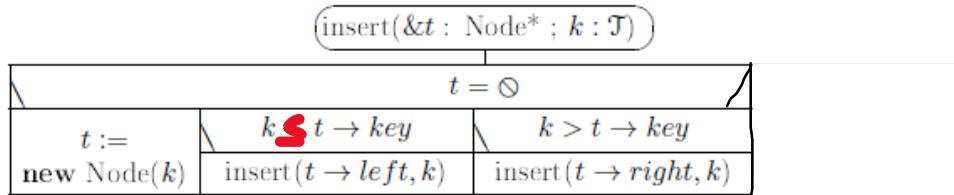
8)

- a. Mondja ki a bináris rendezőfa definícióját!

Egy bináris fát rendezőfának nevezünk, ha minden belső csúcsára és annak  $y$  kulcsára igaz:

- A csúcs bal részfájában minden csúcs  $x$  kulcsára  $x \leq y$ ,
- A csúcs jobb részfájában minden csúcs  $z$  kulcsára  $z \geq y$

- b. A  $t:\text{Node}^*$  egy láncoltan ábrázolt bináris rendezőfát azonosít. Írja meg az  $\text{insert}(t, k)$  eljárás rekurzív, ciklust nem tartalmazó struktogramját,  $MT(h) \in \Theta(h)$  műveletigénnel!



- c. Hogyan tudna a fenti eljárásra stabil rendezést építeni?

Amennyiben az if ciklusban felcseréljük az egyenlőség megengedését, vagyis a ' $k \leq t \rightarrow \text{key}$ ' helyett ' $k < t \rightarrow \text{key}$ ' és a ' $k > t \rightarrow \text{key}$ ' helyett ' $k \geq t \rightarrow \text{key}$ ' lenne, akkor biztosítanánk, hogy egyenlőség esetén az új elem jobboldalra legyen beszúrva, vagyis az elemek sorrendje az eredeti állapotában maradna.

## 6. Rendezés lineáris időben

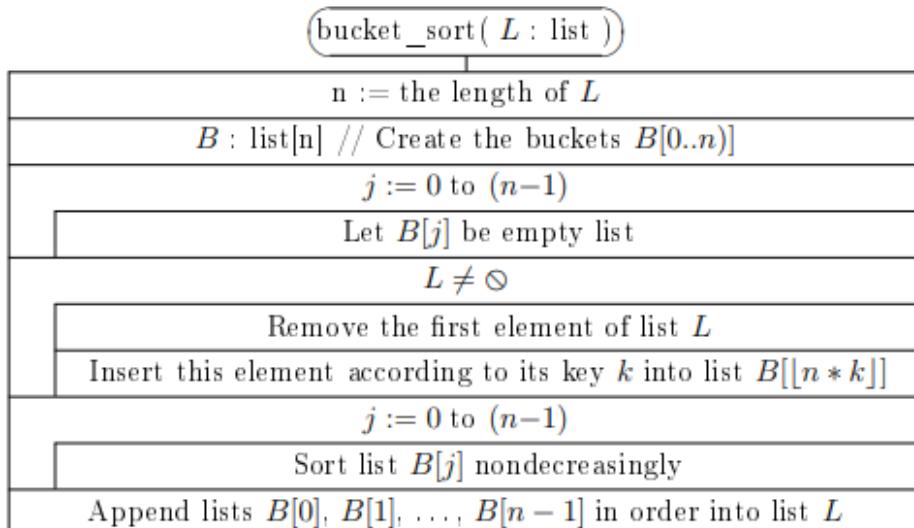
### 6.1. Edényrendezés a $[0;1]$ intervallumon (bucket sort)

1)

- a. Adja meg a bucket sort algoritmusát  $[0; 1]$ -beli kulcsokra!

-> lásd 2) a. megoldás

- b. Adja meg a bucket sort struktogramját!



- c. Mekkora a minimális műveletigénye? Mekkora az átlagos műveletigénye, és milyen feltételel? Hogyan tudnánk biztosítani, hogy a maximális műveletigénye  $\Theta(n * \lg n)$  legyen?

**Minimális:  $\Theta(n)$**

**Maximális:  $\Theta(n^2)$**

**Átlagos:  $\Theta(n)$**

Nyilván  $mT(n) \in \Theta(n)$ , és a fenti egyenletes eloszlást feltételezve  $AT(n) \in \Theta(n)$  is teljesül.  $MT(n)$  pedig attól függ, hogy a  $B[j]$  listákat milyen módszerrel rendezzük. Pl. egyszerű beszűró rendezést használva  $|MT(n)| \in \Theta(n^2)$ , összefésülő rendezéssel viszont  $MT(n) \in \Theta(n \log n)$ .

2)

- a. A  $\langle 0.42; 0.16; 0.64; 0.39; 0.20; 0.89; 0.13; 0.79; 0.53; 0.71 \rangle$ , tíz elemű listán mutassa be a bucket sort algoritmusát  $[0; 1)$ -beli kulcsokra!

## Edényrendezés lejátszása

Kezdeti lista:  $\langle 0.79, 0.13, 0.16, 0.64, 0.39, 0.20, 0.89, 0.53, 0.71, 0.42 \rangle$

	<b>Beszűrás után</b>	<b>Rendezés után</b>	<b>Amelyik listába több kulcs is kerül, ott mindenkor a lista elejére szúrjuk be az új elemet, majd lerendezzük a listákat: Mivel L S1L, az elejére a legegyszerűbb beszűrni.</b>
0			
1	$\langle 0.16, 0.13 \rangle$	$\langle 0.13, 0.16 \rangle$	
2	$\langle 0.20 \rangle$		
3	$\langle 0.39 \rangle$		
4	$\langle 0.42 \rangle$		
5	$\langle 0.53 \rangle$		
6	$\langle 0.64 \rangle$		
7	$\langle 0.71, 0.79 \rangle$	$\langle 0.71, 0.79 \rangle$	
8	$\langle 0.89 \rangle$		
9			

Végeredmény:  $\langle 0.13, 0.16, 0.20, 0.39, 0.42, 0.53, 0.64, 0.71, 0.79, 0.89 \rangle$

- b. Mekkora a (szokásos, beszűró rendezéses változatának) minimális és maximális műveletigénye? Miért? Mekkora az átlagos műveletigénye?

-> lásd 1) c. megoldás

- c. Hogyan tudná a maximális műveletigényt aszimptotikusan csökkenteni?

A bucket sort összefésüléses rendezéses változatával aszimptotikusan csökkenteni tudjuk a maximális műveletigényt.

- d. Mit értünk stabil rendezés alatt? Hogyan tudná a bucket sort-ot stabil rendezéssé alakítani?

Egy rendezés akkor stabil, ha megtartja az ekvivalens kulcsú elemek eredeti sorrendjét. A bucket sort alapvetően nem stabil rendezési algoritmus, mivel a vödrökbe történő besoroláskor elveszíthetjük az eredeti sorrendet az azonos kulcsértékű elemek között. Azonban, ha a vödrökbe történő besorolás során alkalmazunk egy stabil rendezési algoritmust – például az összefésülő rendezési algoritmust –, akkor a bucket sort-ot stabil rendezéssé lehet alakítani.

3)

Adott az  $L$  egyszerű láncolt lista, aminek  $n \geq 0$  eleme van. minden elemének kulcsa a  $[0; 1]$  intervalumon egyenletes eloszlás szerint választott érték. Írja meg a bucketSort( $L, n$ ) utasítással meghívható egyszerű edényrendezés struktogramját.  $AT(n) \in \Theta(n)$ ,  $MT(n) \in \Theta(n * \log n)$  műveletigénnel és  $M(n) \in O(n)$  tárigénnel! Segédrendezésként felhasználható a megfelelő, ebben a félévben tanult, egyszerű láncolt listákat kulcsösszehasonlításokkal rendező eljárás. Ezt nem kell megírni, a kód többi részét viszont teljes részletességgel kérjük.

## 6.2. Leszámláló rendezés (counting sort)

1)

- a. Adja meg a leszámláló rendezés előfeltételeit, struktogramját és aszimptotikus műveletigényét!

A leszámláló rendezés a radix sort ideális segédrendezése, ha a rendezendő adatokat egy tömb tartalmazza.

<code>counting_sort(<math>A, B : \mathbb{T}[n] ; r : \mathbb{N} ; \varphi : \mathbb{T} \rightarrow [0..r]\rangle)</math></code>	
$C : \mathbb{N}[r]$ // counter array	
$k := 0$ to $r-1$	
$C[k] := 0$ // init the counter array	
$i := 0$ to $n - 1$	
$C[\varphi(A[i])]++$ // count the items with the given key	
$k := 1$ to $r-1$	
$C[k] += C[k - 1]$ // $C[k] :=$ the number of items with key $\leq k$	
$i := n - 1$ downto 0	
$k := \varphi(A[i])$ // $k :=$ the key of $A[i]$	
$C[k] --$ // The next one with key $k$ must be put before $A[i]$ where	
$B[C[k]] := A[i]$ // Let $A[i]$ be the last of the {unprocessed items with key $k$ }	

A műveletigény nyilván  $\Theta(n + r)$ . Feltételezve, hogy  $r \in O(n)$ ,  $\Theta(n + r) = \Theta(n)$ , azaz  $T(n) \in \Theta(n)$ .

- b. Szemléltesse a  $<30; 20; 11; 22; 23; 13>$  négyes számrendszerbeli számok tömbjén, ha a kulcsfüggvény a **baloldali számjegyet** választja ki!

Példa a leszámláló rendezés szemléltetésére:

**Leszámláló rendezés:**  
 **$A = <11, 20, 10, 23, 21, 30>$**

- Leszámlálás a második számjegy szerint:

C	11	20	10	23	21	30	GY	KGY				
0	0		1	2			3	3	3	2		1 0
1	0	1			2		2		5		4	3
2	0					1		0	5			
3	0						1	6			5	

Helyre rakáskor a Z tömbből olvassuk ki, hova kell tenni az elemet majd csökkentjük eggyel Z megfelelő értékét.

B=	1	2	3	4	5	6
	20	10	30	11	21	23

- $B = <20, 10, 30, 11, 21, 23>$

- c. Minek kellett teljesülnie a bemenetre, és minek a rendezésre, hogy a fenti példában a végeredmény, mint számsor is rendezett lett? Hogyan biztosítottuk a rendezés e tulajdonságát?

A bemenetre teljesülnie kellett, hogy az utolsó számjegy szerint már rendezett legyen (hiszen a feladat a baloldali számjegyet szerinti rendezést kért), míg a rendezésre???

### 6.3. Radix rendezés leszámláló rendezéssel

1)

- a. Mutassa be a számjegypozíciós (Radix) rendezés működését a következő, négyes számrendszerbeli számok tömbjén: <20; 02; 21; 01; 31; 20>! Az egyes menetekben leszámláló rendezést alkalmazzon!

#### Számjegypozíciós ("Radix") rendezés-leszámláló rendezéssel

A: <20; 02; 21; 01; 31; 20>

2. szj. szerint:

	20	02	21	01	31	20	C	C	20	31	01	21	02	20
0	1					2	2	2	1					0
1			1	2	3	3	5		4	3	2			
2		1				1	6						5	
3						0	6							

1 szj. szerint:

	1	2	3	4	5	6	02	GY	KGY	02	31	01	21	20	20
B:	20	20	21	01	31		2	2	2	1		0			
0				1			2	0	2						
1								3	5						
2	1	2	3		1			1	6		5		4	3	2
3															

- b. Mekkora a fenti rendezés aszimptotikus műveletigénye, és miért?

Esetünkben a leszámláló rendezés műveletigénye  $\Theta(n + r)$ , így az egész rendezésé (a radix rendezéssel együtt),  $\Theta(d(n + r))$ . Mivel azonban az  $r$  és a  $d$  is konstans, a költség átirható  $\Theta(n)$ -re.

#### Műveletigény:

Ez  $\Theta(d * \text{rendezés}(n))$  költségű, ahol a  $\text{rendezés}(n)$  a belső rendezésünk költsége.

- c. A leszámláló rendezés – mint segédprogram – mely tulajdonságára épül a Radix rendezés? Mit jelent ez a tulajdonság az adott esetben?

A leszámláló rendezés stabil rendezés, azaz az azonos kulcsú elemek egymáshoz viszonyított sorrendjét megőrzi. Továbbá tömbökre hatékony, lineáris műveletigényű, nem helyben rendező és nem összehasonlítható rendezés.

A leszámláló rendezés egy számjegy szerint rendezi a tömb elemeit. A Radix rendezés fogja biztosítani azt, hogy a legkevésbé fontos számjegytől a legfontosabb számjegyig mindenki számjegyre lefuttassa a leszámláló rendezést.

2)

- a. Mutassa be a számjegypozíciós („Radix”) rendezés működését a <11; 20; 10; 23; 21; 30> négyes számrendszerbeli számok tömbjén! Az egyes menetekben leszámláló rendezést alkalmazzon!

-> lásd 1) a. megoldás

- b. Mekkora a Radix rendezés műveletigénye, és miért?

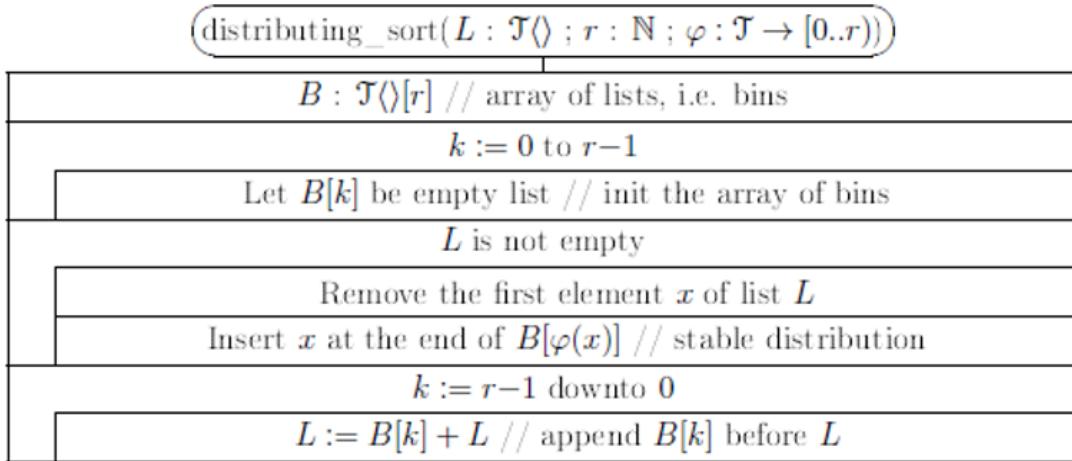
-> lásd 1) b. megoldás

- c. A leszámláló rendezés – mint segédprogram – mely tulajdonságára épül a Radix rendezés? Mit jelent ez a tulajdonság az adott esetben?

-> lásd 1) c. megoldás

## 6.4. Radix rendezés szétválogatással

<https://people.inf.elte.hu/kinga/algoritmusok1/Radix.mp4>



1)

- a. Mutassa be a számjegypozíciós („Radix”) rendezés működését a  $<31; 20; 11; 23; 21; 10>$  négyes számrendszerbeli számok listáján! Az egyes menetekben a megfelelő számjegy szerinti szétválogatást alkalmazzon!

**Radix rendezés listákra - lejátszás**

$L = <103, 232, 111, 013, 211, 002, 012> (r = 4; d = 3)$

- 1. mentet (utolsó szj. szerint)
  - $B_0 = \langle \rangle$
  - $B_1 = \langle 111, 211 \rangle$
  - $B_2 = \langle 232, 002, 012 \rangle$
  - $B_3 = \langle 103, 013 \rangle$
- Összefűzve:  $L = (111, 211, 232, 002, 012, 103, 013)$
- 2. Menet: (utolsó előtti szj. szerint)
  - $B_0 = \langle 002, 103 \rangle$
  - $B_1 = \langle 111, 211, 012, 013 \rangle$
  - $B_2 = \langle \rangle$
  - $B_3 = \langle 232 \rangle$
- Összefűzve:  $L = (002, 103, 111, 211, 012, 013, 232)$
- 3. Menet: (első szj. szerint)
  - $B_0 = \langle 002, 012, 013 \rangle$
  - $B_1 = \langle 103, 111 \rangle$
  - $B_2 = \langle 211, 232 \rangle$
  - $B_3 = \langle \rangle$
- Összefűzve:  $L = (002, 012, 013, 103, 111, 211, 232)$

- b. Mekkora a Radix rendezés műveletigénye, és miért?

**Radix rendezés listákra**

- A radix rendezés által felhasznált szétválogató rendezés
  - $r$  alapú számrendszer
  - 1. számok szétválogatása
    - $i$ -edik számjegyük értéke szerint
    - $B_0, B_1, B_2, \dots, B_{r-1}$  -kezdetben üres- polcokra (bins)
    - fontos: az egyes polcokon **megmaradjon** a számoknak a szétválogatás előtti sorrendje
  - 2. a polcok tartalmát összefűzi L-be.
- Műveletigény:
  - Belső (szétválogató rendezés):  $\Theta(n + r)$
  - $r$  darab polc üresre inicializálása
  - $n$  elemet szétválogatása
  - $r$  db polc összefűzése
  - A radix rendezés:  $\Theta(d * (n + r))$
  - a belső (stabil) rendezést  $d$ -szer hívja meg
  - $d$  konstans és  $r \in O(n)$

➤  **$T_{\text{radix\_sort}}(n) \in \Theta(n)$**

c. A felhasznált rendezés – mint segédprogram – mely tulajdonságára épül a Radix rendezés? Mit jelent ez a tulajdonság az adott esetben?

A szétválogató rendezés stabil rendezés, azaz az azonos kulcsú elemek egymáshoz viszonyított sorrendjét megőrzi. Továbbá láncolt listákra optimális, lineáris műveletigényű, nem helyben rendező és nem összehasonlító rendezés.

A szétválogató rendezés egy számjegy szerint rendezi a lista elemeit. A Radix rendezés fogja biztosítani azt, hogy a legkevésbé fontos számjegytől a legfontosabb számjegyig minden egyik számjegyre lefuttassa a szétválogató rendezést.

2)

Oldja meg az előző feladatot a  $\{11; 20; 10; 23; 21; 30\}$  input listával!

-> lásd 1) a. megoldás

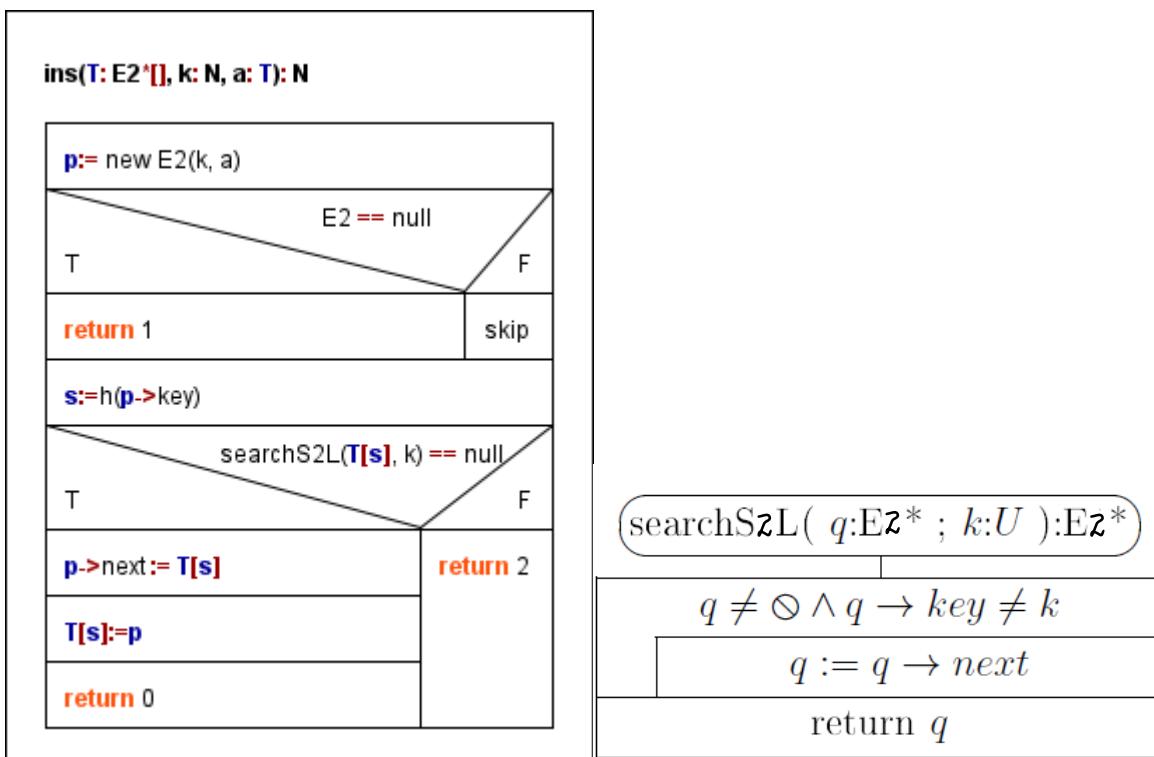
## 7. Hasító táblák

### 7.1. Ütközés feloldása láncolással

1)

A  $Z[0..(m-1)]$  hasító tábla rései kétirányú, nemciklikus, fejelem nélküli, rendezetlen láncolt listák pointerei. Adott a  $k \bmod m$  hasító függvény. A kulcsütközéseket láncolással oldjuk fel. Mindegyik kulcs csak egyszer szerepelhet  $Z$ -ben.

- a. Írja meg az  $\text{ins}(Z, k, a)$ : 0..2 értékű függvényt, ami beszúrja a hasító táblába a  $(k, a)$  kulcs-adat párt! Ha a táblában már volt  $k$  kulcsú elem, a beszúrás meghiúsul, és a 2 hibakódot adja vissza. Különben, ha nem tudja már a szükséges listaelemet allokálni, az 1 hibakódot adja vissza. (Feltessük, hogy a new művelet, ha sikertelen, akkor  $\emptyset$  pointert ad vissza.) Az  $\text{ins}()$  művelet akkor ad vissza 0 kódot, ha sikeres volt a beszúrás. Ilyenkor az új listaelemet a megfelelő lista elejére szúrja be.



- b. Mi a kitöltöttségi hányados? Milyen aszimptotikus becslést tud adni a fenti művelet minimális, átlagos és maximális futási idejére? Miért?

A kitöltöttségi hányados a hasító táblában eltárolt elemek számának és a hasítótáblában rendelkezésre álló helyek összességének hányadosa. A fenti metódus műveletigénye a következőképpen alakul:

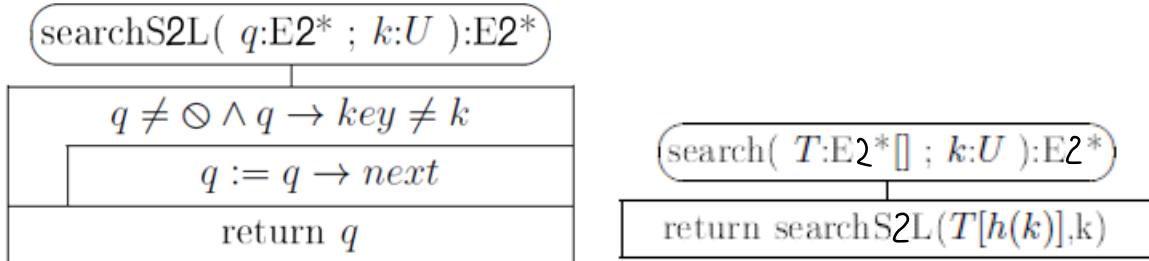
$$mT \in \Theta(1), MT(n) \in \Theta(n), AT(n,m) \in \Theta(1 + n/m)$$

Megjegyzés:  $AT(n,m) \in \Theta(1 + n/m)$  feltétele, hogy a  $h : U \rightarrow 0..(m-1)$  függvény egyszerű egyenletes hasítás legyen, és még az indokolja, hogy a résekhez tartozó listák átlagos hossza  $= n/m = \alpha$ .

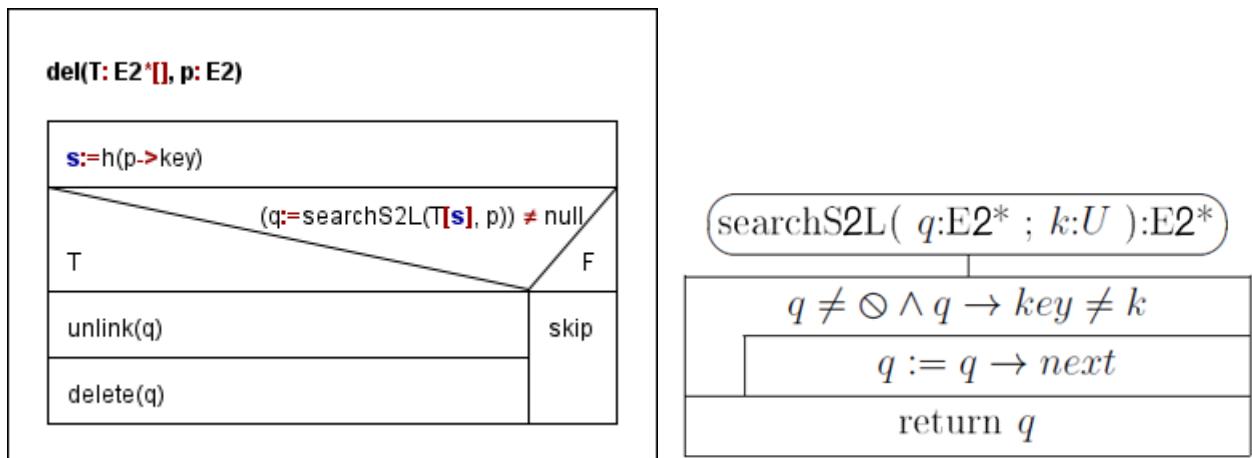
2)

A  $Z[0..(m-1)]$  hasító tábla rései kétirányú, nemciklikus, fejelem nélküli, rendezetlen láncolt listák pointerei. Adott a  $k \bmod m$  hasító függvény. A kulcsütöközéseket láncolással oldjuk fel. Mindegyik kulcs csak egyszer szerepelhet  $Z$ -ben.

a. Írja meg a  $\text{search}(Z, k)$  függvényt, ami visszaadja a  $Z$ -beli,  $k$  kulcsú listaelem címét, vagy a  $\emptyset$  pointert, ha ilyen nincs!



b. Írja meg a  $\text{del}(Z, p)$  eljárást, ami törli a  $Z$  hasító táblából (és deallokálja is) a  $p$  pointer által mutatott listaelemet!



c. Mi a kitöltöttségi hányados? Milyen aszimptotikus becslést tudunk adni a fenti műveletek minimális, átlagos és maximális futási idejére? Miért?

-> lásd 1) b. megoldás

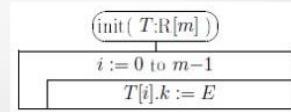
## 7.2. Nyílt címzés

# Nyílt címzéses hashelés

- Az adatrekordok közvetlenül a résekben vannak

- hasító tábla:  $T : R[m]$
- rekordok: R típus
- Egyedi kulcsok
- Rések
  - Szabad (üres / törlt)
  - Foglalt

R
+ $k : U \cup \{E, D\}$ // $k$ is a key or it is Empty or Deleted
+ ... // satellite data



- Hashfüggvény

- $h : U \times 0..(m-1) \rightarrow 0..m-1$  - próbafüggvény
- Tetszőleges  $k$  kulcsra a  $h(k, 0), h(k, 1), \dots, h(k, m-1)$  -potenciális próbasorozat
- Első elem:  $h_1(k)$
- $m$  darab hasító függvény:  $h(\cdot, i) : U \rightarrow 0..(m-1)$  ( $i \in 0..(m-1)$ )

1)

A  $Z[0..(m-1)]$  hasító táblában a kulcsütközést nyílt címzéssel oldjuk fel.

- a. Mit értünk kitöltöttségi hánnyados, próbasorozat és egyenletes hasítás alatt? Mit tudunk a keresés és a beszúrás során a próbák várható számáról?

A kitöltöttségi hánnyados a hasító táblában eltárolt elemek számának és a hasítótáblában rendelkezésre álló helyek összességének hánnyadosa.

A  $\langle h(k, 0), h(k, 1), \dots, h(k, m-1) \rangle$  sorozatot a azért nevezük potenciális próbasorozatnak, mert a beszúrás, keresés vagy törlés során ennek ténylegesen csak egy prefixét állítjuk elő.

Ideális esetben egy tetszőleges potenciális próbasorozat a  $\langle 0, 1, \dots, (m-1) \rangle$  sorozatnak minden az  $m!$  permutációját azonos valószínűsséggel állítja elő. Ilyenkor egyenletes hasításról beszélünk.

(A  $h : U \rightarrow [0..m]$  függvény egyszerű egyenletes hasítás, ha a kulcsokat a rések között egyenletesen szórja szét, azaz hozzávetőleg ugyanannyi kulcsot képez le az  $m$  rés mindegyikére. Tetszőleges hasító függvények kapcsolatos elvárás, hogy egyszerű egyenletes hasítás legyen.)

Amennyiben a táblában nincsenek törlt rések – egyenletes hasítást és a hasító tábla  $0 < \alpha < 1$  kitöltöttségét feltételezve –, egy sikertelen keresés illetve egy sikeres beszúrás várható hossza legfeljebb

$$\frac{1}{1-\alpha}$$

míg egy sikeres keresés illetve sikertelen beszúrás várható hossza legfeljebb

$$\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

Ez azt jelenti, hogy egyenletes hasítást feltételezve, pl. 50%-os kitöltöttség mellett egy sikertelen keresés (illetve egy sikeres beszúrás) várható hossza legfeljebb 2, míg egy sikeres keresésé (illetve egy sikertelen beszúrásé) kisebb, mint 1,387; 90%-os kitöltöttség mellett pedig egy sikertelen keresés (illetve egy sikeres beszúrás) várható hossza legfeljebb 10, míg egy sikeres keresésé (illetve egy sikertelen beszúrásé) kisebb, mint 2,559 [4].

b. Mekkora egy sikertelen keresés várható hossza 80%-os kitöltöttség esetén, ha nincs törölt rés? Egy sikeres keresésé ennél több vagy kevesebb? Miért?

Sikertelen keresés várható hossza legfeljebb:  $1 / (1 - \alpha)$

Sikeres keresés várható hossza legfeljebb:  $(1 / \alpha) * \ln(1 / (1 - \alpha))$

80%-os kitöltöttség esetén ezek értékei: 5 (sikertelen keresés várható hossza),  $1,25 * \ln(5)$  (sikeres keresés várható hossza). Ezek alapján a sikeres keresés várható hossza kisebb-egyenlő, mint a sikertelen keresésé, hiszen a sikertelen keresésnél végig kell járnunk az összes elemet, míg sikeres keresésnél nem feltétlenül szükséges az összes elem bejárása.

c. Legyen most  $m = 11$ ,  $h_1(k) = k \bmod m$ , és alkalmazzon lineáris próbát! Az alábbi műveletek mindegyikére adja meg a próbasorozatot  $\langle \dots \rangle$  alakban! Szúrja be a táblába sorban a következő kulcsokat: 10; 22; 31; 4; 15; 28; 16; 26; 62; ezután törölje a 16-ot, majd próbálja megkeresni a 27-et és a 62-t, végül pedig szúrja be a 27-et! Szemléltesse a hasító tábla változásait! Rajzolja újra, valahányszor egy művelet egy nemüres rés állapotát változtatja meg!

Példa a lineáris próba szemléltetésre:

### Lineáris próba

$m=11$

$h_1(k) := k \bmod m$

	$h_1(k)$	Próbásorozat
I(10)	10	$\langle 10 \rangle$
I(22)	0	$\langle 0 \rangle$
I(31)	9	$\langle 9 \rangle$
I(4)	4	$\langle 4 \rangle$
I(15)	4	$\langle 4, 5 \rangle$
I(28)	6	$\langle 6 \rangle$
I(16)	5	$\langle 5, 6, 7 \rangle$

	0	1	2	3	4	5	6	7	8	9	10
22	22	62			4	15	28	16	26	31	10
											X

	$h_1(k)$	Próbásorozat
I(26)	4	$\langle 4, 5, 6, 7, 8 \rangle$
I(62)	7	$\langle 7, 8, 9, 10, 0, 1 \rangle$
D(16)	5	$\langle 5, 6, 7 \rangle$
S(27)	X	$\langle 5, 6, 7, 8, 9, 10, 0, 1, 2 \rangle$
S(62)	7	$\langle 7, 8, 9, 10, 0, 1 \rangle$ ✓
I(27)	5	$\langle 5, 6, 7, 8, 9, 10, 0, 1, 2 \rangle$

$m = 11$ ,  $h_1(k) = k \bmod m$

Művelet	kulcs	$h_1(k)$	próbásorozat	siker?	0	1	2	3	4	5	6	7	8	9	10
i	24	2	2	✓			24								
i	16	5	5	✓			24				16				
i	57	2	2, 3	✓			24	57			16				
i	32	10	10	✓			24	57		16					32
i	15	4	4	✓			24	57	15	16					32
d	57	2	2, 3	✓			24	D	15	16					32
i	21	10	10, 0	✓	21		24	D	15	16					32
s	2	2	2, 3, 4, 5, 6	✗	21		24	D	15	16					32
i	2	2	2, 3, 4, 5, 6	✓	21		24	2	15	16					32
i	2	2	2, 3	✗	21		24	2	15	16					32
s	21	10	10, 0	✓	21		24	2	15	16					32
i	35	2	2, 3, 4, 5, 6	✓	21		24	2	15	16	35				32
d	15	4	4	✓	21		24	2	D	16	35				32
i	35	2	2, 3, 4, 5, 6	✗	21		24	2	D	16	35				32

2)

A  $Z[0..(m-1)]$  hasító táblában a kulcsütközést nyílt címzéssel oldjuk fel.

a. Mit értünk kitöltöttségi hánnyados és egyenletes hasítás alatt? Mit tudunk a keresés és a beszúrás során a próbák várható számáról?

-> lásd 1) a. megoldás

b. Mekkora egy sikertelen keresés várható hossza 90%-os kitöltöttség esetén, ha nincs töröl rés? Egy sikeres keresésé ennél több vagy kevesebb? Miért?

Sikertelen keresés várható hossza legfeljebb:  $1 / (1 - \alpha)$

Sikeres keresés várható hossza legfeljebb:  $(1 / \alpha)^* \ln(1 / (1 - \alpha))$

90%-os kitöltöttség esetén ezek értékei: 10 (sikertelen keresés várható hossza), 10/9 (sikeres keresés várható hossza). Ezek alapján a sikeres keresés várható hossza kisebb-egyenlő, mint a sikertelen keresésé, hiszen a sikertelen keresésnél végig kell járnunk az összes elemet, míg sikeres keresésnél nem feltétlenül szükséges az összes elem bejárása.

c. Legyen most  $m = 11$ , (az egyszerűség kedvéért)  $h_1(k) = k \bmod m$ , és alkalmazzunk négyzetes próbát a szokásos  $c_1 = c_2 = \frac{1}{2}$  konstansokkal! Az alábbi műveletek mindegyikére adja meg a próbásorozatát  $\leq \dots \geq$  alakban! Szemléltesse a hasító tábla változásait! Rajzolja újra, valahányszor egy művelet egy nemüres rés állapotát változtatja meg! Szúrja be a táblába sorban a következő kulcsokat: 22; 31; 4; 28; 15; 14; 30; ezután törölje a 14-et, majd próbálja megkeresni a 38-at és a 30-at, végül pedig szúrja be a 27-et!

Példa a négyzetes próba vérehajtására:

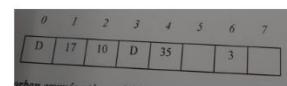
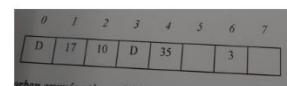
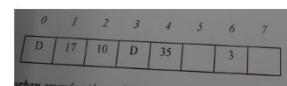
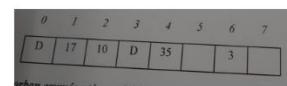
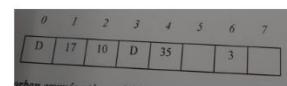
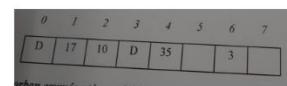
5. feladat							
Adott egy 8 méretű nyílt címzésű, négyzetes próbásorozatot használó hasító tábla.							
Előszörösen hasító flagerecny: $h_1(k) = k \bmod 8$							
$c_1 = c_2 = 0,5$							
Mivel a tábla tartalma:							
0	1	2	3	4	5	6	7
D	17	10	D	35		3	

az el sorban egymás után az alább megadott műveleteket a táblán Adjja meg a próbásorozatot!

Beszür: 3 Keres: 26 Töröl: 3 Beszür: 2 Beszür: 72 Töröl: 81

$$h_n(k) = k \bmod 8$$

0	1	2	3	4	5	6	7
D	17	10	D	35		3	

	$h_n(k)$	Próbásorozat	
$i(3)$	3	$\langle 3_{01}, 4_{16} \rangle \times$	
$i(26)$	2	$\langle 2_{1}, 3_{01}, 5_{\epsilon} \rangle \times$	
$d(3)$	3	$\langle 3_{01}, 4_{16} \rangle$	
$i(2)$	2	$\langle 2_{1}, 3_{01}, 5_{\epsilon} \rangle \checkmark$	
$i(72)$	0	$\langle 0_{1}, 1_{2}, 3_{16}, 2_{4}, 7_{\epsilon} \rangle \checkmark$	
$d(81)$	1	$\langle 1_{2}, 2_{1}, 4_{16} \rangle \times$	

3)

a. Adott egy  $m = 7$  méretű, üres hasító tábla. Nyílt címzést és kettős hasítást alkalmazunk a  $h_1(k) = k \bmod m$ ,  $h_2(k) = 1 + (k \bmod (m - 2))$  tördelő függvények segítségével. Az alábbi műveletek mindegyikére adja meg a próbasorozatot  $\langle \dots \rangle$  alakban, és a hasító táblát is rajzolja újra, valahányszor egy művelet egy nemüres rés státuszát változtatja meg (i-beszúrás, s-keresés, d-törlés): i37, i45, i19, i72, i33, d19, s12, i33, d33, i33.

Példa a kettős hasítás elvégzésére:

5. feladat

Adott egy 11 méretű nyílt címzésű, kettős hasítás próbasorozatot használó hasító tábla.

Elsődleges hasító függvény:  $h_1(k) = k \bmod 11$ Másodlagos hasító függvény:  $h_2(k) = (k \bmod 10) + 1$ 

Induláskor a tábla tartama:

0	1	2	3	4	5	6	7	8	9	10
34	84			37	18	D	7			44

a) Hány eleme volt biztosan a táblának a 44 beszúrásakor?

b) Végezz el sorban egymás után az alább megadott műveleteket a táblán! Adja meg a próbasorozatokat!

Beszür: 23 Keres: 33 Töröl: 34 Beszür: 20 Töröl: 39

4)

	$h_1(k)$	$h_2(k)$	Próbásorozat
i(23)	1	4	$\langle 1, 5, 8_E \rangle$
s(33)	0	4	$\langle 0, 4, 8_E \rangle \times$
d(34)	1	5	$\langle 1, 6_E, 0 \rangle \sim$
i(20)	5	1	$\langle 9, 10, 0_D, 1, 2_E \rangle$
d(39)	6	10	$\langle 6_D, 5, 4, 3_E \rangle \times$

	$h_1(k)$	$h_2(k)$	Próbásorozat
a) i(44)		0	5 $\langle 0, 5, 10 \rangle$

2 eleme biztosan volt a táblának a 44 beszúrásakor.

0	1	2	3	4	5	6	7	8	9	10
34	84			37	18	D	7		23	44
0	1	2	3	4	5	6	7	8	9	10
34	84			37	18	D	7		23	44

0	1	2	3	4	5	6	7	8	9	10
34	84			37	18	D	7		23	44
0	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10
34	84			37	18	D	7		23	44
0	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10
34	84			37	18	D	7		23	44
0	1	2	3	4	5	6	7	8	9	10

b. Magyarázza meg, mi a szerepe nyílt címzés esetén a foglalt, az üres és a törlött réseknek, a beszúrás, a keresés és a törlés során!

A  $k$  kulcsú  $r$  adat beszúrásánál például először a  $h(k, 0)$  réssel próbálkozunk. Ha ez foglalt és a kulcsa nem  $k$ , folytatjuk a  $h(k, 1)$ -gyel stb., míg nem üres rést találunk, vagy  $k$  kulcsú foglalt rést találunk, vagy kimerítjük az összes lehetséges próbát, azaz a  $\langle h(k, 0), h(k, 1), \dots, h(k, m - 1) \rangle$

A keresésnél tehát átlépjük a törlött réseket is, és csak akkor állunk meg,

- ha megtaláltuk a keresett kulcsú elemet (sikeres keresés),
- ha üres rést találunk vagy kimerítjük a potenciális próbasorozatot (sikertelen keresés).

A beszúrásnál is egy teljes keresést végezünk el, most a beszúrandó adat kulcsára, de ha közben találunk törlött rést, az első ilyent megjegyezzük.

- Ha a keresés sikeres, akkor a beszúrás sikertelen (hiszen duplikált kulcsot nem engedünk meg).
- Ha a keresés sikertelen, és találtunk közben törlött rést, akkor a beszúrandó adatot az elsőként talált törlött réssé tesszük (hogy a jövőbeli keresések hossza a lehető legkisebb legyen).
- Ha a keresés sikertelen, de nem talál törlött rést, viszont üres résen áll meg, akkor a beszúrandó adatot ebbe az üres résbe tesszük.
- Ha a keresés sikertelen, de nem talál sem törlött, sem üres rést, és így azért áll meg, mert a potenciális próbasorozatot kimerít, akkor a beszúrás sikertelen (mert a hasítótábla tele van).

Ha elég sokáig használunk egy nyílt címzésű hasító táblát, elszaporodhatnak a törlött rések, és elfogyhatnak az üres rések, holott a tábla esetleg közel sincs tele. Ez azt jelenti, hogy pl. a sikertelen keresések az egész táblát végig fogják nézni. Ez ellen a tábla időnként frissítésével védekezhetünk, amikor megszüntetjük a törlött réseket. (A legegyszerűbb megoldás kimásolja az adatokat egy temporális területre, üresre inicializálja a táblát, majd a kimentett adatokat egyesével újra beszűrja.)

A törlés egy sikeres keresést követően a megtalált  $i$  rés kulcsának törlött-re ( $D$ ) állításából áll. Itt az üres-re ( $E$ ) állítás helytelen lenne, mert ha például feltesszük, hogy a  $k$  kulcsú adatot kulcsütözés miatt a  $T[h(k, 1)]$  helyre tettük, majd törlöttük a  $h(k, 0)$  helyen levő adatot (azaz a  $T[h(k, 0)]$  rés üresre állítottuk), akkor egy ezt követő keresés nem találná meg a  $k$  kulcsú adatot.

A  $k$  kulcsú adat keresésénél is a fenti potenciális próbasorozatot követjük, és akkor állunk meg, ha megtaláltuk a keresett kulcsú foglalt rést (sikeres keresés), illetve ha üres rést találunk vagy kimerítjük a potenciális próbasorozatot (sikertelen keresés). Ha a keresés a  $h(k, i-1)$  próbánál áll meg, akkor (és csak akkor) a keresés (azaz az aktuális próbasorozat) hossza  $i$ .

c. Mi a kitöltöttségi hányados? Milyen becslést tudunk adni az egyes műveletigényekre, és milyen feltétellel?

A kitöltöttségi hányados a hasító táblában eltárolt elemek számának és a hasítótáblában rendelkezésre álló helyek összességének hányadosa. A beszúrás, keresés és törlés metódusok műveletigénye a következőképpen alakul:

$$mT \in \Theta(1), MT(n) \in \Theta(n), AT(n,m) \in \Theta(1 + n/m)$$

4)

Adott egy  $m = 11$  méretű üres hasító tábla. Nyílt címzést és kettős hasítást alkalmazunk a „ $h_1(k) = k \bmod m$ ” és a „ $h_2(k) = 1 + (k \bmod (m - 1))$ ” tördelő függvények segítségével.

a. Az alábbi műveletek mindegyikére adja meg a próbasorozatát  $<\dots>$  alakban! Szemléthesse a hasító tábla változásait! Rajzolja újra, valahányszor egy művelet egy nemüres rés állapotát változtatja meg! Szűrja be a táblába sorban a következő kulcsokat: 12; 17; 23; 105. Ezután törölje a 23-at, majd próbálja megkeresni a 133-at, végül pedig szűrja be a 133-at!

-> lásd 3) a. megoldás

b. Magyarázza meg, mi a különbség nyílt címzés esetén a foglalt, az üres és a törölt rések között, az alkalmazható műveletek szempontjából!

-> lásd 3) b. megoldás